

# MATH5312 - Advanced Numerical Methods II

Taught by Jiafeng Cai

Notes by Aaron Wang

## Contents

<b>1</b>	<b>February 2nd, 2021</b>	<b>3</b>
1.1	Class Logistics . . . . .	3
1.1.1	Assessment Scheme . . . . .	3
1.1.2	Reference Books . . . . .	3
1.2	Direct Methods for Systems of Linear Equations . . . . .	3
<b>2</b>	<b>February 4th, 2021</b>	<b>6</b>
2.1	Efficient Gaussian Elimination . . . . .	6
2.2	Complexity of Gaussian Elimination . . . . .	7
2.3	LU Decomposition view of Gaussian Elimination . . . . .	8
<b>3</b>	<b>February 9th, 2021</b>	<b>10</b>
3.1	Pivoting LU Decomposition . . . . .	10
3.2	LU Decomposition on SPD Matrices . . . . .	13
3.3	Cholesky Decomposition . . . . .	13
<b>4</b>	<b>February 11th, 2021</b>	<b>14</b>
4.1	LU Decomposition Continued . . . . .	14
4.2	LU on Banded Matrices . . . . .	16
4.3	LU on Sparse Matrix . . . . .	18
<b>5</b>	<b>February 16th, 2021</b>	<b>18</b>
5.1	Basic Iterative Method . . . . .	18
5.2	Jacobi Iteration . . . . .	19
5.3	Review on Norms . . . . .	20
5.3.1	Vector Norms . . . . .	20
5.3.2	Matrix Norm . . . . .	21
<b>6</b>	<b>February 18th, 2021</b>	<b>25</b>
6.1	Spectral Radius Cont. . . . .	25
6.2	Convergence of Jacobi Iteration . . . . .	26
6.3	Computation Cost of Jacobi Iteration . . . . .	27
6.4	Jacobi Iteration for 1D Discrete Laplacian . . . . .	27

<b>7</b>	<b>February 23rd, 2021</b>	<b>29</b>
7.1	Jacobi for 2D Discrete Laplacian . . . . .	29
7.2	Jacobi for SPD Matrices . . . . .	30
7.3	Lower Bound of Jacobi Convergence Rate . . . . .	32
<b>8</b>	<b>February 25th, 2021</b>	<b>34</b>
8.1	Gauss-Seidel Iteration . . . . .	34
8.1.1	Convergence of Gauss-Seidel . . . . .	35
8.2	Acceleration of G-S / and Jacobi (SOR) . . . . .	37
8.2.1	Convergence of SOR . . . . .	38
<b>9</b>	<b>March 2nd, 2021</b>	<b>39</b>
9.1	General Framework of Stationary Iterations . . . . .	39
9.1.1	Matrix Splitting . . . . .	39
9.1.2	Preconditioned Richardson Iteration . . . . .	39
9.1.3	Projeciton Methods . . . . .	42
	<b>Index</b>	<b>44</b>

# 1 February 2nd, 2021

## 1.1 Class Logistics

This course is a continuation of MATH5311. In MATH5311, we introduced how to discretize differential equations. In the course, we focus on the algebraic side of numerical analysis, introducing some more advanced solvers for these numerical methods.

### 1.1.1 Assessment Scheme

**Biweekly Homework** - 60%

**Take-home Final Exam** - 40%

### 1.1.2 Reference Books

**Lecture Notes** - Main content that will be covered

**Yousef Saad: Iterative Methods for Sparse Linear Systems** - Free ebook can be found on Canvas

**Golub, Van Lean: Matrix Computations**

**Trefethen, Bau: Numerical Linear Algebra**

## 1.2 Direct Methods for Systems of Linear Equations

The goal of this chapter is very simple, solve:

$$Ax = b$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \in \mathbb{R}^{n \times n}$$

and:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$$

which is the unknown vector, and:

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \in \mathbb{R}^n$$

is the right hand side (RHS).

The solution of system of linear equations is a fundamental task in numerical analysis. If we want to solve a non-linear equation, we can do it iteratively, where each step is solving a linear system. In this chapter, we will introduce **Direct Methods**.

**Definition 1.1.** A **Direct Method** is a method in which, if there is no truncation error in the computer, can solve the system exactly in finite time.

One example of a direct method is **Gaussian Elimination**.

### Example 1.2

Consider:

$$\begin{cases} 2x_1 + x_2 - x_3 = 1 \\ 4x_1 + 5x_2 - 3x_3 = -3 \\ -2x_1 + 5x_2 - 2x_3 = -8 \end{cases}$$

To perform Gaussian Elimination, we eliminate  $x_1$  from Equations 2 and 3 by subtracting Equation 1, and then  $x_2$  from Equation 3 by subtracting Equation 2. Doing so, we get:

$$\begin{cases} 2x_1 + x_2 - x_3 = 1 \\ 3x_2 - x_3 = -5 \\ -x_3 = 3 \end{cases}$$

After this, we can solve  $x_3$  and then using its value solve  $x_2$  and  $x_1$ .

Typically we use matrix terminology to reformulate the methods. We do this by writing the **augmented matrix** of the above linear system:

$$\left[ \begin{array}{ccc|c} 2 & 1 & -1 & -1 \\ 4 & 5 & -2 & -3 \\ -2 & 5 & -3 & -8 \end{array} \right]$$

### Lemma 1.3

Row Operations are equivalent to left matrix multiplications.

### Example 1.4

In Example 1.2, the row operation to eliminate  $x_1$  for Equations 2 and 3 can be represented as:

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \left[ \begin{array}{ccc|c} 2 & 1 & -1 & -1 \\ 4 & 5 & -2 & -3 \\ -2 & 5 & -3 & -8 \end{array} \right] = \left[ \begin{array}{ccc|c} 2 & 1 & -1 & -1 \\ 0 & 3 & -1 & -5 \\ 0 & 6 & -3 & -7 \end{array} \right]$$

As such, if  $A$  is the augmented matrix and we want to solve the system  $Ax = b$ , we have:

$$L_{n-1} \dots L_1 Ax = L_{n-1} \dots L_1 b$$

and until the LHS is an upper triangular matrix.

**Remark 1.5** — Note that the row operations  $L_i$  are lower triangular matrix. In particular, it is a rank 1 modification of the identity matrix:

$$L_i = I - v e_i^T$$

where  $v \in \mathbb{R}^n$  and  $v_1 = v_2 = \dots = v_i = 0$  and  $e_i$  is the  $i$ -th unit vector.

**Definition 1.6.**  $L_i$  are called **Gauss Transformations**.

In summary, Gaussian Elimination consists of the following steps:

1. Left multiply Gaussian Transformations to reduce  $A$  to an upper triangular matrix:

$$A^{(1)} = A = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{bmatrix} \text{ and } b^{(1)} = b = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix}$$

In step 1, we have:

$$L_1 = \begin{bmatrix} 1 & & & \\ -\frac{a_{21}^{(1)}}{a_{11}^{(1)}} & 1 & & \\ \vdots & & \ddots & \\ -\frac{a_{n1}^{(1)}}{a_{11}^{(1)}} & 0 & \dots & 1 \end{bmatrix} = I - \ell_1 e_1^T, \quad \ell_1 = \begin{bmatrix} 0 \\ \frac{a_{21}^{(1)}}{a_{11}^{(1)}} \\ \vdots \\ \frac{a_{n1}^{(1)}}{a_{11}^{(1)}} \end{bmatrix}$$

After step 1, we have:

$$L_1 A^{(1)} = A^{(2)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & \dots & a_{nn}^{(2)} \end{bmatrix} \text{ and } L_1 b^{(1)} = b^{(2)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix}$$

We can continue this process until the LHS is upper left triangular. The pseudocode of this first part is given in 1.

2. For part 2, we solve the resulting upper triangular linear system:

$$A^{(n)} x = b^{(n)}$$

This is quite straightforward, as we just start from the last row and solve each equation, since we are only introducing one unknown with each equation. The pseudocode of this second part is given in 2.

**Remark 1.7** — Note that a naive implementation would take  $O(n^3)$  memory, as three indexes are used for  $a_{ij}^{(k)}$ .

We can implement this more efficiently by dropping the superscript, since we do not change it. In other words, we use  $a_{ij}^{(k)}$  to overwrite  $a_{ij}$ . This will take up the upper triangular part of the matrix. Since the lower triangular part of the matrix is 0, we can use write  $\ell_{ik}$  to this memory. If we do not need  $b$  anymore, we can overwrite it as well, meaning that we can do Gaussian Elimination without requiring additional memory.

---

**Algorithm 1** Gaussian Elimination Part 1

---

```

1: for  $k = 1 : n - 1$  do
2:   for  $i = k + 1 : n$  do
3:      $\ell_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)}$ 
4:   end for
5:   for  $i = k + 1 : n$  do
6:     for  $j = k + 1 : n$  do
7:        $a_{ij}^{(k+1)} = a_{ij}^{(k+1)} - \ell_{ik} a_{kj}^{(k)}$  ▷ Compute  $A^{(k+1)} = L_k A^{(k)}$ 
8:     end for
9:   end for
10:  for  $i = k + 1 : n$  do
11:     $b_i^{(k+1)} = b_i^{(k+1)} - \ell_{ik} b_i^{(k)}$  ▷ Compute  $b^{(k+1)} = L_k b^{(k)}$ 
12:  end for
13: end for

```

---



---

**Algorithm 2** Gaussian Elimination Part 2

---

```

1: for  $k = n : -1 : 1$  do
2:   for  $j = n : -1 : k + 1$  do
3:      $y_k = y_k - u_{kj} x_j$ 
4:   end for
5:    $x_k = y_k / a_{kk}$ 
6: end for

```

---

## 2 February 4th, 2021

### 2.1 Efficient Gaussian Elimination

Last time, we introduced a memory efficient way to perform Gaussian Elimination by reusing the inputs. However, this implementation is **not** the most time efficient. To do this, we need to consider the computer architecture. Recall that computers have a few different layers of memory:

- CPU register
- Cache
- RAM
- Disk

The lower the level, the larger the memory but communicating is slower. As such, to improve the efficiency, we should minimize the communication between the different levels of memory.

**Remark 2.1** — This principle of memory management should always be considered when performing matrix multiplication.

Note that we can reformulate the previous algorithms into matrix operations. First we perform a vector scaling, then we perform a rank 1 update, and then a matrix vector

product. Since these are very basic operations, there are many very efficient implementations of these **Basic Linear Algebra Subroutines (BLAS)**. There are a few different levels of these BLAS.

**BLAS1** : Vector operations, e.g.

$$\beta = x^T y, y = \beta x + y, \quad \text{where } x, y \in \mathbb{R}^n, \beta \in \mathbb{R}$$

**BLAS2** : Matrix-Vector operations

$$y = Ax + y, \quad \text{where } A \in \mathbb{R}^{m \times n}, x, y \text{ are vectors}$$

**BLAS3** : Matrix-Matrix operations

$$C = AB + C, \quad \text{where } A, B, C \text{ are matrices}$$

These BLAS are implemented to be optimized on the given computer architecture.

**Remark 2.2** — Since BLAS3 can be constructed using BLAS2, etc. in terms of the efficiency, we have:

$$\text{BLAS3} > \text{BLAS2} > \text{BLAS1}$$

where efficiency is the time needed

**Remark 2.3** — BLAS will come with the CPU.

**AMD** Core Math Library (ACML)

**Intel** Math Kernel Lib (MKL)

Replacing the algorithm with BLAS, we have:

---

**Algorithm 3** Gaussian Elimination with BLAS

---

```

1: for  $k = 1 : n - 1$  do
2:    $A(k + 1 : n, k) = A(k + 1 : n, k) / A(k, k)$ 
3:    $A(k + 1 : n, k + 1 : n) = A(k + 1 : n, k + 1 : n) - A(k + 1 : n, k)A(k, k + 1 : n)$ 
4:    $b(k + 1 : n, 1) = b(k + 1 : n, 1) - b(k)A(k + 1 : n, k)$ 
5: end for
6: for  $k = n : -1 : 1$  do
7:    $b(k) = b(k) - A(k, k + 1 : n)b(k + 1 : n)$ 
8:    $b(k) = b(k) / A(k, k)$ 
9: end for

```

---

## 2.2 Complexity of Gaussian Elimination

For this, we want to see how many scalar operations are needed. Counting the cost, we have:

$$\sum_{k=1}^{n-1} [(n-k) + 2(n-k)^2 + 2(n-k)] + \sum_{k=1}^n [2(n-k) + 1]$$

$$\begin{aligned}
& \sum_{k=1}^{n-1} [(n-k) + 2(n-k)^2 + 2(n-k)] + \sum_{k=1}^n [2(n-k) + 1] \\
&= 3 \sum_{k=1}^{n-1} (n-k) + 2 \sum_{k=1}^{n-1} (n-k)^2 + 2 \sum_{k=1}^{n-1} k + \sum_{k=1}^n 1 \\
&= \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{1}{3}n \\
&= \frac{2}{3}n^3 + O(n^2) \\
&= O(n^3).
\end{aligned}$$

**Theorem 2.4**

The computational complexity of Gaussian Elimination is  $O(n^3)$ .

**2.3 LU Decomposition view of Gaussian Elimination**

Remember we can express Gaussian Elimination as:

$$L_{n-1} \dots L_2 L_1 A x = L_{n-1} \dots L_1 b$$

Note that  $L_{n-1} \dots L_2 L_1 A = A^{(n)}$  which is upper triangular. Let us denote this by:

$$U = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{bmatrix} \in \mathbb{R}^{n \times n}$$

Note that since each  $L_i$  is a Gaussian transformation which is invertable, we have:

$$A = L_1^{-1} L_2^{-1} \dots L_{n-1}^{-1} U$$

**Theorem 2.5**

$$L_k^{-1} = I + \ell_k e_k^T$$

*Proof.*

$$\begin{aligned}
& (I - \ell_k e_k^T)(I + \ell_k e_k^T) \\
&= I - \ell_k e_k^T + \ell_k e_k^T - \ell_k e_k^T \ell_k e_k^T \\
&= I.
\end{aligned}$$

□



**Theorem 2.6**

$$L_1^{-1}L_2^{-1}\dots L_{n-1}^{-1} = I + \sum_{i=1}^{n-1} \ell_i e_i^T$$

which is lower triangular.

*Proof.* Direct computation. □

**Theorem 2.7**

Gaussian Elimination gives:

$$A = LU$$

where  $L$  is unit lower triangular, and  $U$  is upper triangular. This is called the **LU Decomposition**.

**Remark 2.8** — Note that in the memory efficient implementation of GE, we are computing  $L$  and  $U$  and it is stored in the original matrix.

With this, we have:

$$Ax = b \iff LUx = b \iff \begin{cases} Ly = b \\ Ux = y \end{cases}$$

**Theorem 2.9**

The computation cost is:  $\begin{cases} \text{LU decomposition} & \frac{2}{3}n^3 + O(n^2) \\ \text{Forward Substitution} & O(n^2) \\ \text{Backward Substitution} & O(n^2) \end{cases}$

**Remark 2.10** — If we use the same coefficient matrix, we only need to compute the decomposition once.

This LU decomposition view of Gaussian Elimination has a number of advantages:

1. If we want to solve  $Ax_i = b_i$  for  $i = 1, \dots, m$ , then we only need to do LU decomposition once
2. LU decomposition can be used for other matrix computation tasks. e.g.
  - (a)  $\det(A) = \det(LU) = \det(L)\det(U) = u_{11}u_{22}\dots u_{nn}$
  - (b)  $A^{-1} = U^{-1}L^{-1}$

### 3 February 9th, 2021

#### 3.1 Pivoting LU Decomposition

##### Example 3.1

Consider  $Ax = b$ :

$$\begin{bmatrix} 2 & 1 & -1 \\ 4 & 5 & -3 \\ -2 & 5 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \\ -8 \end{bmatrix}$$

At each step, we have:

1.

$$A = \begin{bmatrix} 2 & 1 & -1 \\ 2 & 3 & -1 \\ -1 & 6 & -3 \end{bmatrix}$$

2.

$$A = \begin{bmatrix} 2 & 1 & -1 \\ 2 & 3 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

Thus, we have:

$$L = \begin{bmatrix} 1 & & \\ 2 & 1 & \\ -1 & 2 & 1 \end{bmatrix} \quad U = \begin{bmatrix} 2 & 1 & -1 \\ 3 & -1 & \\ & & -1 \end{bmatrix}$$

Now we solve  $Ly = b$  by forward substitution:

$$\begin{bmatrix} 1 & & \\ 2 & 1 & \\ -1 & 2 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \\ -8 \end{bmatrix} \implies \begin{cases} y_1 = 1 \\ y_2 = -5 \\ y_3 = 3 \end{cases}$$

Then we solve  $Ux = y$  by back substitution:

$$\begin{bmatrix} 2 & 1 & -1 \\ 3 & -1 & \\ & & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -5 \\ 3 \end{bmatrix} \implies \begin{cases} y_1 = 1/3 \\ y_2 = -8/3 \\ y_3 = -3 \end{cases}$$

In LU Decomposition,  $A(k, k)$  at step  $k$  is in the denominator, which means:

1. The procedure cannot continue if  $A(k, k) = 0$

##### Example 3.2

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

2. If  $A(k, k)$  is small, then the computation becomes inaccurate.

As such, LU Decomposition is unstable unless  $A(k, k)$  is large.

**Definition 3.3.** We call  $A(k, k)$  at step  $k$  a **pivot entry**.

In order to avoid small pivots, we use “pivoting”, where we interchange the row or columns of the matrix. There are many pivoting schemes, for example Row Pivoting, where at each step  $k$ , we choose the largest entry in the  $k$  column under  $A(k, k)$ .

---

**Algorithm 4** Row Pivoting

---

```

1: for  $k = 1 : n - 1$  do
2:   Find the max entry in abs. value in  $A(k : n, k)$  denoted by  $i_k$ 
3:    $A(i_k, :) \iff A(k, :)$  ▷ Swap row  $i_k$  and  $k$ 
4:    $A(k + 1 : n, k) = A(k + 1 : n, k) / A(k, k)$ 
5:    $A(k + 1 : n, k + 1 : n) = A(k + 1 : n, k + 1 : n) - A(k + 1 : n, k)A(k, k + 1 : n)$ 
6: end for

```

---

Row interchanging (row  $i_k \iff$  row  $k$ ) is equivalent to Left multiplying by a permutation matrix:

$$P_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 0 & & 1 \\ & & & \ddots & \\ & & 1 & & 0 & \\ & & & & & \ddots \\ & & & & & & 1 \end{bmatrix}$$

Thus when we perform the LU decomposition, we have:

$$L_{n-1}P_{n-1} \dots L_2P_2L_1P_1A = U$$

However, this is too complicated, so we will do some simplification. It is easy to check that  $P_kP_k = I$ , since we are swapping the two rows again, and that  $P_k^T = P_k$ .

**Lemma 3.4**

$$P_{n-1}P_{n-2} \dots P_{k+1}L_kP_{k+1} \dots P_{n-2}P_{n-1} = I - (P_{n-1}P_{n-2} \dots P_{k+1}\ell_k)e_k^T$$

*Proof.* We have:

$$\begin{aligned}
L_k &= I - \ell e_k^T \\
P_{k+1}L_kP_{k+1} &= P_{k+1}(I - \ell e_k^T)P_{k+1} = I - P_{k+1}\ell e_k^T P_{k+1} \\
&= I - (P_{k+1}\ell_k)e_k^T \\
&\vdots
\end{aligned}$$

□

If we denote  $P_{n-1}P_{n-2} \dots P_{k+1}\ell_k = \tilde{\ell}_k$ , we have:

$$\begin{aligned}
&L_{n-1}P_{n-1} \dots L_2P_2L_1P_1A \\
&= L_{n-1}(P_{n-1}L_{n-2}P_{n-1})(P_{n-1}P_{n-2} \dots L_2P_2L_1P_1A \\
&= \tilde{L}_{n-1}\tilde{L}_{n-2} \dots \tilde{L}_1P_{n-1}P_{n-2} \dots P_1A.
\end{aligned}$$

Denoting  $P_{n-1}P_{n-2}\dots P_1 = P$ , we have:

$$PA = L_1^{-1}L_2^{-1}\dots L_{n-1}^{-1}U$$

Giving us

$$PA = LU$$

### Example 3.5

Using the same example as before,  $A = \begin{bmatrix} 2 & 1 & -1 \\ 4 & 5 & -3 \\ -2 & 5 & -2 \end{bmatrix}$  we have:

$$1. \quad \begin{bmatrix} 2 & 1 & -1 \\ 4 & 5 & -3 \\ -2 & 5 & -2 \end{bmatrix} \xrightarrow{\text{row } 1 \leftrightarrow \text{row } 2} \begin{bmatrix} 4 & 5 & -3 \\ 2 & 1 & -1 \\ -2 & 5 & -2 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 5 & -3 \\ \frac{1}{2} & -\frac{3}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{15}{2} & -\frac{7}{2} \end{bmatrix}$$

$$2. \quad \begin{bmatrix} 4 & 5 & -3 \\ \frac{1}{2} & -\frac{3}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{15}{2} & -\frac{7}{2} \end{bmatrix} \xrightarrow{\text{row } 2 \leftrightarrow \text{row } 3} \begin{bmatrix} 4 & 5 & -3 \\ \frac{1}{2} & -\frac{3}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{15}{2} & -\frac{7}{2} \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 5 & -3 \\ -\frac{1}{2} & \frac{15}{2} & -\frac{7}{2} \\ \frac{1}{2} & -\frac{3}{2} & \frac{1}{2} \end{bmatrix}$$

Giving us an output:

$$PA = \begin{bmatrix} 4 & 5 & -3 \\ -\frac{1}{2} & \frac{15}{2} & -\frac{7}{2} \\ \frac{1}{2} & -\frac{3}{2} & \frac{1}{2} \end{bmatrix}$$

With this, we have:

$$L = \begin{bmatrix} 1 & & \\ -\frac{1}{2} & 1 & \\ \frac{1}{2} & -\frac{1}{5} & 1 \end{bmatrix} \quad U = \begin{bmatrix} 4 & 5 & -3 \\ \frac{15}{2} & -\frac{7}{2} & -\frac{1}{5} \end{bmatrix} \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

**Remark 3.6** — All entries in  $L$  have an abs. value smaller than 1.

There are also other pivoting strategy, such as **full pivoting** where you swap the columns as well to get the maximum pivot in the lower submatrix. This would give us:

$$PAQ = LU$$

where  $Q$  is also a permutation matrix.

**Remark 3.7** — Full pivoting not only rearranges the equation, but it also rearranges the unknowns.

**Remark 3.8** — Full pivoting is more stable than row pivoting but it is computationally more expensive.

For any non-singular matrix, we can solve it using pivoting LU decomposition. However, we can consider the structure of the matrix to improve the algorithm.

## 3.2 LU Decomposition on SPD Matrices

**Definition 3.9.**  $A \in \mathbb{R}^{n \times n}$  is **symmetric positive definite (SPD)** if it is:

- Symmetric:  $A = A^T$
- Positive Definite:  $x^T A x > 0$  for all  $x \in \mathbb{R}^n$  and  $x \neq 0$

### Example 3.10

Examples of SPD matrices include:

- Discrete Laplacian is SPD
- Normal equation of Least Squares
- Hessian of strictly convex functions

For general matrices, the LU decomposition is  $\frac{2}{3}n^3 + O(n^2)$ . For SPD matrices, we can reduce this to  $\frac{1}{3}n^3 + O(n^2)$ , meaning we can reduce the computation cost by half. In addition, no pivoting is necessary for SPD matrices.

There is a slight modification of LU for SPD matrices called the **Cholesky Decomposition**.

## 3.3 Cholesky Decomposition

Assume  $A \in \mathbb{R}^{n \times n}$  is SPD. Then there exists a decomposition such that:

$$A = LL^T$$

where  $L$  is a lower triangular matrix. With Cholesky decomposition, we can solve the linear system:

$$Ax = b \iff LL^T x = b \iff \begin{cases} Ly = b \\ L^T x = y \end{cases}$$

Since  $A = LL^T$  with:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & & a_{nn} \end{bmatrix} \quad L = \begin{bmatrix} \ell_{11} & & & \\ \ell_{21} & \ell_{22} & & \\ \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & & \ell_{nn} \end{bmatrix} \quad L^T = \begin{bmatrix} \ell_{11} & \ell_{21} & \cdots & \ell_{n1} \\ & \ell_{22} & \cdots & \ell_{n2} \\ & & \ddots & \vdots \\ & & & \ell_{nn} \end{bmatrix}$$

We have:

- $a_{11} = \ell_{11}^2 \implies \ell_{11} = \sqrt{a_{11}}$
- $a_{21} = \ell_{21}\ell_{11} \implies \ell_{21} = a_{21}/\ell_{11}$
- $a_{n1} = \ell_{n1}\ell_{11} \implies \ell_{n1} = a_{n1}/\ell_{11}$

To be continued in the next lecture.

## 4 February 11th, 2021

### 4.1 LU Decomposition Continued

Recall that:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad L = L = \begin{bmatrix} \ell_{11} & & & \\ \ell_{21} & \ell_{22} & & \\ \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & & \ell_{nn} \end{bmatrix} \quad L^T = \begin{bmatrix} \ell_{11} & \ell_{21} & \cdots & \ell_{n1} \\ & \ell_{22} & \cdots & \ell_{n2} \\ & & \ddots & \vdots \\ & & & \ell_{nn} \end{bmatrix}$$

Observing the first column of  $LL^T$ , we have:

- $a_{11} = \ell_{11}^2 \implies \ell_{11} = \sqrt{a_{11}}$
- $a_{21} = \ell_{21}\ell_{11} \implies \ell_{21} = a_{21}/\ell_{11}$
- $a_{n1} = \ell_{n1}\ell_{11} \implies \ell_{n1} = a_{n1}/\ell_{11}$

**Remark 4.1** — Note that we only need to compare the lower triangular part of  $LL^T$ , since it is symmetric.

For the second column, note that we have:

$$a_{k2} = \ell_{k1}\ell_{21} + \ell_{k2}\ell_{22}$$

As such, we have:

- $a_{22} = \ell_{21}^2 + \ell_{22}^2 \implies \ell_{22} = (a_{22} - \ell_{21}^2)^{\frac{1}{2}}$
- $\ell_{k2} = (a_{k2} - \ell_{k1}\ell_{21})/\ell_{22}$  for  $k = 3, \dots, n$ .

We can continue this process, and for the  $k$ -th column, we have:

$$a_{kk} = \ell_{k1}^2 + \ell_{k2}^2 + \dots + \ell_{kk}^2 \implies \ell_{kk} = (a_{kk} - \sum_{i=1}^{k-1} \ell_{ki}^2)^{\frac{1}{2}}$$

and:

$$a_{ik} = \sum_{j=1}^k \ell_{ij}\ell_{kj} \implies \ell_{kj} = (a_{ik} - \sum_{j=1}^{k-1} \ell_{ij}\ell_{kj})/\ell_{kk}$$

for  $i = k+1, \dots, n$ . Thus, we have Algorithm 5.

However, Algorithm 5 is not memory efficient. To create a memory efficient version, we should use the lower triangular part of  $L$  to overwrite that of  $A$ . This gives us Algorithm 6.

**Remark 4.2** — The only difference between Algorithm 5 and 6 is that we change  $\ell$  to  $a$ .

Using BLAS, we have Algorithm 7.

**Algorithm 5** Cholesky Decomposition Version 1

---

```

1: for  $k = 1 : n$  do
2:    $\ell_{kk} = (a_{kk} - \sum_{i=1}^{k-1} \ell_{ki}^2)^{1/2}$ 
3:   for  $i = k + 1 : n$  do
4:      $\ell_{ik} = (a_{ik} - \sum_{j=1}^{k-1} \ell_{ij} \ell_{kj}) / \ell_{kk}$ 
5:   end for
6: end for

```

---

**Algorithm 6** Cholesky Decomposition Version 2 - Memory Efficient

---

```

1: for  $k = 1 : n$  do
2:    $a_{kk} = (a_{kk} - \sum_{i=1}^{k-1} a_{ki}^2)^{1/2}$ 
3:   for  $i = k + 1 : n$  do
4:      $a_{ik} = (a_{ik} - \sum_{j=1}^{k-1} a_{ij} a_{kj}) / a_{kk}$ 
5:   end for
6: end for

```

---

**Algorithm 7** Cholesky Decomposition Version 3 - BLAS

---

```

1: for  $k = 1 : n$  do
2:    $A(k, k) = A(k, k) - \text{norm}(A(k, 1 : k - 1), 2)^2$ 
3:    $A(k, k) = (A(k, k))^{1/2}$ 
4:    $A(k + 1 : n, k) = A(k + 1 : n, k) - A(k + 1 : n, 1 : k - 1)(A(k, 1 : k - 1))^T$ 
5:    $A(k + 1 : n, k) = A(k + 1 : n, k) / A(k, k)$ 
6: end for

```

---

**Theorem 4.3**

Cholesky decomposition has a complexity of:

$$\frac{1}{3}n^3 + O(n^2)$$

which is only half of LU for general  $A$ . In addition, the memory required is also halved.

**Example 4.4**

Consider  $A = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$ . We have:

- Step 1:  $\begin{bmatrix} \sqrt{2} & & \\ \frac{1}{\sqrt{2}} & 2 & \\ 0 & 1 & 2 \end{bmatrix}$

- Step 2:  $\begin{bmatrix} \sqrt{2} & & \\ \frac{1}{\sqrt{2}} & \sqrt{\frac{3}{2}} & \\ 0 & \sqrt{\frac{2}{3}} & 2 \end{bmatrix}$

- Step 3:  $\begin{bmatrix} \sqrt{2} & & \\ \frac{1}{\sqrt{2}} & \sqrt{\frac{3}{2}} & \\ 0 & 1 & \sqrt{\frac{4}{3}} \end{bmatrix}$

This gives us:

$$L = \begin{bmatrix} \sqrt{2} & & \\ \frac{1}{\sqrt{2}} & \sqrt{\frac{3}{2}} & \\ 0 & 1 & \sqrt{\frac{4}{3}} \end{bmatrix}$$

**Remark 4.5** — The SPD-ness of  $A$  guarantees that  $a_{kk} - \sum_{i=1}^{k-1} \ell_{ki}^2$  is positive.

## 4.2 LU on Banded Matrices

**Definition 4.6.** A matrix is **banded** if there exists an integer  $p > 0$  such that:

$$a_{ij} \neq 0 \implies |i - j| \leq p$$

**Example 4.7**

. For  $p = 1$ , the matrix is called a **tri-diagonal matrix**:

$$A = \begin{bmatrix} \times & \times & & & \\ \times & \times & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & \ddots & \times \\ & & & & \times & \times \end{bmatrix} \in \mathbb{R}^{n \times n}$$

If we apply LU decomposition to a tri-diagonal matrix, we have:



- Step 1:

$$A(2, 1) = A(2, 1)/A(1, 1)$$

$$A(2, 2) = A(2, 2) - A(2, 1)A(1, 2)$$

- Step  $k$ :

$$A(k + 1, k) = A(k + 1, k)/A(k, k)$$

$$A(k + 1, k + 1) = A(k + 1, k + 1) - A(k + 1, k)A(k, k + 1)$$

This leads to Algorithm 8

**Remark 4.8** — Note that the resulting matrix is also tri-diagonal.

---

**Algorithm 8** LU Decomposition for Tri-Diagonal Matrix

---

```

1: for  $k = 1 : n - 1$  do
2:    $A(k + 1, k) = A(k + 1, k)/A(k, k)$ 
3:    $A(k + 1, k + 1) = A(k + 1, k + 1) - A(k + 1, k)A(k, k + 1)$ 
4: end for

```

---

**Theorem 4.9**

The computation complexity for Algorithm 8 is  $\sum_{k=1}^{n-1} 3 = 3(n - 1) \sim O(n)$ .

**Theorem 4.10**

In general, for a banded matrix with band width  $p$ , the computation cost is  $O(np^2)$ .

**Example 4.11**

The discrete Laplacian matrix in 1-D is a banded matrix with  $p = 1$  (tri-diagonal).

**Example 4.12**

The discrete Laplacian in 2-D on a grid is:

$$A_1 \otimes I + I \otimes A_2 \in \mathbb{R}^{N \times N},$$

where  $N = n^2$ ,  $A_1, A_2 \in \mathbb{R}^{n \times n}$  are the 1-D discrete Laplacian matrix for each dimension, it is a banded matrix with  $p = n$ .

**Remark 4.13** — If we applied GE to example 4.12, it would be  $O(Np^2) = O(N^2)$ , which is significantly lower than  $O(N^3)$ .

Our goal is to improve this cost to  $O(N)$ .

## 4.3 LU on Spare Matrix

**Definition 4.14.** A **sparse matrix** is a matrix with many zero entries.

**Remark 4.15** — Banded matrices with small bandwidth can be considered to be sparse.

Sparse matrices arise from many numerical solutions of PDE, as differential operators are local, making many of the entries zero. Sometimes, we can exploit the structure of sparse matrix to speed up LU decomposition. However, LU does not always work well.

### Example 4.16

Consider  $\begin{bmatrix} 5 & 4 & 3 & 2 \\ 4 & 4 & & \\ 3 & & 3 & \\ 2 & & & 2 \end{bmatrix}$ . Note that this is sparse. However, when we divide the first row, we will modify the lower sub matrix, making the final result a dense matrix. Thus it would be the same as applying LU to a general dense matrix, as the sparse structure is destroyed.

Now, we should ask, can we keep as many zeros as possible in LU? The answer is yes, by re-ordering both the equations and the unknowns, i.e. permute the matrix.

### Example 4.17

If we consider the matrix in 4.16, we could permute it so we get:

$$\begin{bmatrix} 2 & & & 2 \\ & 3 & & 3 \\ & & 4 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}$$

If we perform LU on this matrix, the sparse structure is preserved.

Now we must ask, how should we re-order to allow LU to produce the minimum number of **fill-in's**? This is a very difficult topic to answer, and was an active research topic. To answer this, we use graph theory. This is beyond the scope of this course.

## 5 February 16th, 2021

### 5.1 Basic Iterative Method

In this chapter, we will introduce iterative methods. There will be a lot of overlap with MATH5311. For iterative methods, we make use of the fact that matrix vector products are fast for sparse matrices.

**Remark 5.1** — If the matrix is sparse, then the matrix vector product is on the order of non-zero entries.

### Example 5.2

For the Discrete Laplacian in 2D, the matrix vector product is  $O(N)$ .

We will solve  $Ax = b$  by stationary iterative methods. Given  $x_k \in \mathbb{R}^n$ , we want to improve the quality of  $x_k$  using:

$$x_{k+1} = Gx_k + f, \quad k \in 0, 1, 2, \dots$$

where  $G \in \mathbb{R}^{n \times n}$  and  $f \in \mathbb{R}^n$  are stationary matrices and vectors.

**Definition 5.3.**  $G$  is a **stationary matrix**, as it does not depend on  $k$ .

## 5.2 Jacobi Iteration

- $(y)_i$  denotes the  $i$ -th component of a vector  $y$
- $\xi_i^{(k)}$  denotes the  $i$ -th component of  $x_k$
- $\xi_i$  denotes the  $i$ -th component of  $x$  (true solution)
- $\xi_i$  denotes the  $i$ -th component of  $b$

The idea of the Jacobi iteration is, given  $x_k$ , we obtain  $x_{k+1}$  by solving the  $i$ -th unknown from the  $i$ -th equation. More precisely, we are solving:

$$(Ax - b)_i = 0,$$

with  $\xi_j$ ,  $j \neq i$ , fixed to be  $\xi_j^{(k)}$ , for  $i = 1, \dots, n$ . As such, we have:

$$\begin{aligned} (Ax - b)_i &= 0 \\ \iff a_{ii}\xi_i^{(k+1)} + \sum_{j \neq i} a_{ij}\xi_j^{(k)} &= \beta_i \\ \iff \xi_i^{(k+1)} &= (\beta_i - \sum_{j \neq i} a_{ij}\xi_j^{(k)})/a_{ii}. \end{aligned}$$

This can be expressed as Algorithm 9. In order to perform this effeciently, we reformulate

---

### Algorithm 9 Element Wise Jacobi Iteration

---

```

1: for  $k = 0, 1, 2, \dots$  do
2:   for  $i = 1, \dots, n$  do
3:      $\xi_i^{(k+1)} = (\beta_i - \sum_{j \neq i} a_{ij}\xi_j^{(k)})/a_{ii}$ 
4:   end for
5: end for
```

---

this in matrix notation to make use of BLAS. Let  $A = D - E - F$ , where:

$$A = \begin{bmatrix} d_1 & & * \\ & \ddots & \\ * & & d_n \end{bmatrix} = \begin{bmatrix} d_1 & & 0 \\ & \ddots & \\ 0 & & d_n \end{bmatrix} - \begin{bmatrix} 0 & & 0 \\ & \ddots & \\ -* & & 0 \end{bmatrix} - \begin{bmatrix} 0 & & -* \\ & \ddots & \\ 0 & & 0 \end{bmatrix} = D - E - F$$

Thus, we have Algorithm 10, which is in stationary form.

**Algorithm 10** Jacobi Iteration in Matrix Form

---

```

1: for  $k = 0, 1, 2, \dots$  do
2:    $x_{k+1} = D^{-1}(b + (E + F)x_k)$ 
3: end for

```

---

**Remark 5.4** — Some other equivalent forms of the Jacobi Iteration are:

$$x_{k+1} = D^{-1}(E + F)x_k + D^{-1}b$$

$$x_{k+1} = D^{-1}(D - A)x_k + D^{-1}b$$

$$x_{k+1} = (I - D^{-1}A)x_k + D^{-1}b$$

## 5.3 Review on Norms

### 5.3.1 Vector Norms

**Definition 5.5.** A (vector) **norm** on  $\mathbb{R}^n$  is a function  $\|\cdot\| : \mathbb{R}^n \rightarrow \mathbb{R}$  that satisfies:

1.  $\|x\| \geq 0 \quad \forall x \in \mathbb{R}^n$  and  $\|x\| = 0 \iff x = 0$ .
2.  $\|\alpha x\| = |\alpha| \|x\| \quad \forall \alpha \in \mathbb{R}$  and  $x \in \mathbb{R}^n$ .
3.  $\|x + y\| \leq \|x\| + \|y\| \quad \forall x, y \in \mathbb{R}^n$  (**triangle inequality**).

This defines a **metric** on  $\mathbb{R}^n$ .

**Definition 5.6.** A  $p$ -norm on  $\mathbb{R}^n$  is defined as:

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

#### Example 5.7 (Special $p$ Norm)

Here are a few common norms on  $\mathbb{R}^n$ .

- **$p$ -norm** ( $p \geq 1$ ):
- **Euclidean norm** ( $p = 2$ )

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

- **1-norm** ( $p = 1$ )

$$\|x\|_1 = \sum_{i=1}^n |x_i|$$

- **$\infty$ -norm** ( $p = \infty$ )

$$\|x\|_\infty = \max_{i=1}^n |x_i|$$

**Theorem 5.8 (Holder's Inequality)**

$$|x^T y| \leq \|x\|_p \|y\|_q$$

if  $\frac{1}{p} + \frac{1}{q} = 1$ , with  $p, q \geq 1$ .

**Theorem 5.9 (Cauchy-Schwartz Inequality)**

$$|\langle u, v \rangle| \leq \|u\| \|v\|, \quad \forall u, v \in \mathbb{R}^n$$

**Example 5.10 (Weighted Norm)**

Let  $A \in \mathbb{R}^{n \times n}$  be a symmetric positive definite matrix. Then:

$$\|x\|_A = (x^T A x)^{1/2}$$

is a norm, called the **weighted norm**.

From functional analysis, because  $\mathbb{R}^n$  is finite dimensional, any two norms are equivalent. More formally.

**Theorem 5.11 (Norm equivalence of  $\mathbb{R}^n$ )**

Given  $\|\cdot\|_a$  and  $\|\cdot\|_b$ ,  $\exists C_1, C_2 > 0$  independent of  $x$ , s.t.

$$C_1 \|x\|_b \leq \|x\|_a \leq C_2 \|x\|_b \quad \forall x \in \mathbb{R}^n$$

Consequently, from Theorem 5.11, the convergence of vectors in  $\mathbb{R}^n$  under any norm is the same. Thus, we can analyze the convergence under any norm.

**Remark 5.12** — Theorem 5.11 does not hold for infinite dimensional space. However, for numerical analysis, we always work with finite dimensional space.

**Example 5.13 (Equivalence of 1-norm and other  $p$ -norms)**

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{n} \|x\|_2 \quad \forall x \in \mathbb{R}^n$$

$$\|x\|_\infty \leq \|x\|_1 \leq n \|x\|_\infty \quad \forall x \in \mathbb{R}^n$$

**5.3.2 Matrix Norm**

Let  $\|\cdot\|$  be a norm on  $\mathbb{R}^n$ . Let  $A \in \mathbb{R}^{n \times n}$  be a matrix.

**Definition 5.14.** The **norm of  $A$  induced by the vector norm  $\|\cdot\|$**  is:

$$\|A\| = \sup_{x \in \mathbb{R}^n, x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|$$

**Remark 5.15** — The second equality in 5.14 is due to the scaling property of  $A$  and because the norm is continuous. However, this might not be the case in infinite-dimensional spaces.

We can check that  $\|A\|$  is a matrix, i.e.:

- $\|A\| \geq 0 \quad \forall A \in \mathbb{R}^{n \times n}$  and  $\|A\| = 0 \iff A = 0$ .
- $\|\alpha A\| = |\alpha| \|A\| \quad \forall \alpha \in \mathbb{R}$  and  $A \in \mathbb{R}^{n \times n}$ .
- $\|A + B\| \leq \|A\| + \|B\| \quad \forall A, B \in \mathbb{R}^{n \times n}$  (**triangle inequality**).

In addition, since it is an **operator norm** that is induced, it has some consistency properties, namely

- $\|AB\| \leq \|A\| \|B\| \quad \forall A, B \in \mathbb{R}^{n \times n}$
- $\|Ax\| \leq \|A\| \|x\| \quad \forall A \in \mathbb{R}^{n \times n}, x \in \mathbb{R}^n$

**Example 5.16** (matrix 2-norm)

$$\begin{aligned} \|A\|_2 &= \max_{\|x\|_2=1} \|Ax\|_2 = \left( \max_{\|x\|_2=1} \|Ax\|_2^2 \right)^{\frac{1}{2}} = \left( \max_{x^T x=1} x^T A^T A x \right)^{\frac{1}{2}} \\ &= (\text{maximum eigenvalue of } A^T A)^{\frac{1}{2}} \end{aligned}$$

which is the maximum **singular value** of  $A$ .

**Remark 5.17** — The last equality in Example 5.16 can be shown by taking the eigenvalue decomposition of  $A$ .

**Theorem 5.18**

$$\|A\|_1 = \max_{1 \leq j \leq n} \|a_j\|_1, \quad \text{where } A = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix}, a_j \in \mathbb{R}^n,$$

i.e. the maximum column 1-norm (column sum).

*Proof.* •  $\forall x \in \mathbb{R}^n$  with  $\|x\|_1=1$ , we have:

$$\|Ax\|_1 = \left\| \sum_{j=1}^n x_j a_j \right\|_1 \leq \sum_{j=1}^n |x_j| \|a_j\|_1 \leq \max_{1 \leq j \leq n} \|a_j\|_1 \sum_{j=1}^n |x_j| = \max_{1 \leq j \leq n} \|a_j\|_1$$

Taking the max over all  $x : \|x\|_1 = 1$ , we obtain:

$$\|A\|_1 \leq \max_{1 \leq j \leq n} \|a_j\|_1$$

- Let  $j_0 = \arg \max_{1 \leq j \leq n} \|a_j\|_1$ . Consider  $x = e_{j_0}$ . Then  $\|x\|_1 = 1$  and  $Ax = Ae_{j_0} = a_{j_0}$ . Thus:

$$\|Ax\|_1 = \|a_{j_0}\|_1 = \max_{1 \leq j \leq n} \|a_j\|_1$$

Therefore:

$$\|A\|_1 \geq \|Ax\|_1 = \max_{1 \leq j \leq n} \|a_j\|_1$$

□

**Remark 5.19** — This means that for the matrix 1-norm, the maximum is attained at the image of one of the standard unit vector. This is true, since the 1-ball is a convex polytope.

### Theorem 5.20

$$\|A\|_\infty = \max_{1 \leq i \leq n} \|a^{(i)}\|_\infty, \quad \text{where } A = \begin{bmatrix} (a^{(1)})^T \\ \vdots \\ (a^{(n)})^T \end{bmatrix}, a^{(i)} \in \mathbb{R}^n,$$

i.e. the maximum row 1-norm (maximum row sum).

*Proof.* (omitted).

□

**Definition 5.21.** The **spectral radius** of a matrix  $A$  is defined as:

$$\rho(A) = \max\{|\lambda_i| : \lambda_i \text{ is an eigenvalue of } A\}$$

### Theorem 5.22

Let  $A \in \mathbb{C}^{n \times n}$ . Then:

1.  $\|A\| \geq \rho(A)$  for any matrix norm induced by  $\|\cdot\|$ .
2. For any  $\epsilon > 0$ , we can find a vector norm  $\|\cdot\|$ , s.t. the induced matrix norm satisfies:

$$\|A\| \leq \rho(A) + \epsilon$$

3. From (1) and (2), we have:

$$\rho(A) = \inf \|A\|$$

4. If  $A$  is diagonalizable, there exists a matrix operator norm s.t.

$$\rho(A) = \|A\|$$

5. In particular, when  $A$  is symmetric,  $\rho(A) = \|A\|_2$ .

*Proof.* 1. Let  $\lambda_0, x_0$  be an eigenpair of  $A$  satisfying  $|\lambda_0| = \rho(A)$ . Assume that  $\|x_0\| = 1$ . Then, for any vector norm  $\|\cdot\|$ , its induced operator norm satisfies:

$$\|A\| \geq \|Ax_0\| = \|\lambda_0 x_0\| = |\lambda_0| \|x_0\| = \rho(A)$$

2. We use a construction proof by finding such vector norm. Let

$$A = X \begin{bmatrix} \lambda_1 & \delta_1 & & & \\ & \lambda_2 & \delta_2 & & \\ & & \ddots & \ddots & \\ & & & \lambda_{n-1} & \delta_{n-1} \\ & & & & \lambda_n \end{bmatrix}$$

be the Jordan decomposition, where:  $\delta_i \in \{0, 1\}$ , and  $\lambda_i$  are eigenvalues of  $A$ .

Given  $\epsilon > 0$ , we define:

$$\|x\|_\epsilon = \|(XD_\epsilon)^{-1}x\|_\infty, \quad \text{with } D_\epsilon = \begin{bmatrix} 1 & & & & \\ & \epsilon & & & \\ & & \epsilon^2 & & \\ & & & \ddots & \\ & & & & \epsilon^{n-1} \end{bmatrix}.$$

We can check that  $\|\cdot\|_\epsilon$  is a norm on  $\mathbb{C}^n$ . So:

$$\|A\|_\epsilon = \max_{\|x\|_\epsilon=1} \|Ax\|_\rho = \max_{\|(XD_\epsilon)^{-1}x\|_\infty=1} \|(XD_\epsilon)^{-1}Ax\|_\infty$$

Let  $y = (XD_\epsilon)^{-1}x$ , we have:

$$\|A\|_\epsilon = \max_{\|y\|_\infty=1} \|(XD_\epsilon)^{-1}A(XD_\epsilon)y\|_\infty = \|(XD_\epsilon)^{-1}A(XD_\epsilon)\|_\infty$$

Note that we have:

$$\begin{aligned} (XD_\epsilon)^{-1}A(XD_\epsilon) &= D_\epsilon^{-1}X^{-1}AXD_\epsilon = D_\epsilon \begin{bmatrix} \lambda_1 & \delta_1 & & & \\ & \lambda_2 & \delta_2 & & \\ & & \ddots & \ddots & \\ & & & \lambda_{n-1} & \delta_{n-1} \\ & & & & \lambda_n \end{bmatrix} D_\epsilon \\ &= \begin{bmatrix} \lambda_1 & \epsilon\delta_1 & & & \\ & \lambda_2 & \epsilon\delta_2 & & \\ & & \ddots & \ddots & \\ & & & \lambda_{n-1} & \epsilon\delta_{n-1} \\ & & & & \lambda_n \end{bmatrix} \end{aligned}$$

Thus, since the infinity norm is the maximum row sum, we have:

$$\|A\|_\epsilon \leq \max_{1 \leq i \leq n} (|\lambda_i| + \epsilon) \leq \rho(A) + \epsilon$$

3. By part 2, if  $A$  is diagonalizable,  $\delta_i = 0$  for all  $i$ . Then  $\|A\|_\epsilon = \max_i |\lambda_i| = \rho(A)$ . If  $A = A^T$ , then  $\delta_i = 0$  for all  $i$ ,  $\lambda_i$  are real, and  $X$  is unitary. Thus:

$$\begin{aligned} A^T A &= X^{-1} \begin{bmatrix} \lambda_1^2 & & & \\ & \lambda_2^2 & & \\ & & \ddots & \\ & & & a\lambda_n^2 \end{bmatrix} \\ \rho(A) &= (\rho(A^T A))^{\frac{1}{2}} = \|A\|_2 \end{aligned}$$

□



**Remark 5.23** — 1. and 2. imply

$$\rho(A) = \inf\{\|A\| : \|\cdot\| \text{ is an operator norm}\}$$

In particular

- If  $A$  diagonalizable, then minimum is attainable, meaning:

$$\rho(A) = \min\{\|A\| : \|\cdot\| \text{ is an operator norm}\}$$

- If  $A$  is symmetric:

$$\rho(A) = \|A\|_2 = \min\{\|A\| : \|\cdot\| \text{ is an operator norm}\}$$

## 6 February 18th, 2021

### 6.1 Spectral Radius Cont.

#### Corollary 6.1

Let  $A \in \mathbb{R}^{n \times n}$ . Then:

$$\lim_{k \rightarrow \infty} A^k = 0 \iff \rho(A) < 1$$

*Proof.* • “ $\implies$ ” Assume  $\lim_{k \rightarrow \infty} A^k = 0$ . Let  $\lambda$  be the eigenvalue of  $A$  s.t.  $\rho(A) = |\lambda|$ . For any  $k$ , then  $\lambda^k$  is an eigenvalue of  $A^k$ . We have:

$$(\rho(A))^k = |\lambda|^k = |\lambda^k| \leq \rho(A^k) \leq \|A^k\|$$

for any operator norm. Thus:

$$\lim_{k \rightarrow \infty} (\rho(A))^k \leq \lim_{k \rightarrow \infty} \|A^k\| = 0 \implies \rho(A) < 1$$

- “ $\impliedby$ ” Assume  $\rho(A) < 1$ . Choose  $\epsilon = \frac{1}{2}(1 - \rho(A)) > 0$ . Thus there exists  $\|\cdot\|_\epsilon$  s.t.:

$$\|A\|_\epsilon \leq \rho(A) + \epsilon = \rho(A) + \frac{1}{2}(1 - \rho(A)) = \frac{1}{2} + \frac{1}{2}\rho(A) < 1$$

Then:

$$\|A^k\|_\epsilon \leq (\|A\|_\epsilon)^k \rightarrow 0 \text{ as } k \rightarrow \infty \implies \lim_{k \rightarrow \infty} \|A^k\|_\epsilon = 0$$

Since norms are continuous functions for finite dimension, we have:

$$\|\lim_{k \rightarrow \infty} A^k\| = 0 \rightarrow \lim_{k \rightarrow \infty} A^k = 0$$

□

## 6.2 Convergence of Jacobi Iteration

Recall that the Jacobi iteration can be written in the stationary iteration form:

$$x_{k+1} = Gx_k + f$$

where  $G = I - D^{-1}A$ ,  $f = D^{-1}b$ . Let  $x_*$  be the solution of  $Ax = b$ . i.e. ( $Ax_* = b$ ). Then:

$$\begin{aligned} Dx_* - b &= (D - A)x_* \\ Dx_* &= (D - A)x_* + b \\ x_* &= D^{-1}(D - A)x_* + D^{-1}b \\ x_* &= Gx_* + f. \end{aligned}$$

Taking the difference, we have:

$$(x_{k+1} - x_*) = G(x_k - x_*)$$

Now, taking the norms on both sides, we have:

$$\begin{aligned} \|x_{k+1} - x_*\| &= \|G(x_k - x_*)\| \\ &\leq \|G\| \|x_k - x_*\|. \end{aligned}$$

If  $\rho(G) < 1$ , then we can choose  $\epsilon = \frac{1}{2}(1 - \rho(G))$  and construct the norm  $\|\cdot\|$  (depending on  $G$ ) s.t.

$$\|G\|_\epsilon \leq \rho(G) + \epsilon = \frac{1}{2} + \frac{1}{2}\rho(G) < 1$$

Then:

$$\|x_{k+1} - x_*\|_\epsilon \leq \|G\|_\epsilon \|x_k - x_*\|_\epsilon = \rho \|x_k - x_*\|_\epsilon, \quad \forall k.$$

As a result, we have:

$$\|x_k - x_*\|_\epsilon \leq \rho^k \|x_0 - x_*\|_\epsilon \rightarrow 0 \text{ as } k \rightarrow +\infty,$$

i.e.  $x_k \rightarrow x_*$ .

In addition, the convergence rate is “linear”, because:

$$\frac{\|x_k - x_*\|}{\|x_{k-1} - x_*\|} \leq \rho < 1$$

In order to obtain a  $\tilde{\epsilon}$ -precision solution, i.e.:

$$\begin{aligned} \|x_k - x_*\|_\epsilon &\leq \rho^k \|x_0 - x_*\|_\epsilon < \tilde{\epsilon} \\ \iff \rho^k &\leq \frac{\tilde{\epsilon}}{\|x_0 - x_*\|_\epsilon} \\ \iff k &\geq \frac{\log\left(\frac{\|x_0 - x_*\|_\epsilon}{\tilde{\epsilon}}\right)}{\log(1/\rho)} \sim O(1/\log \rho^{-1}). \end{aligned}$$

**Remark 6.2** — Note that  $\rho$  can be arbitrarily close to  $\rho(G)$ . Thus,  $\rho$  is called the **convergence factor**.

**Remark 6.3** — If  $\rho \approx \rho(G) = 1 - O(1/n^\alpha)$ , where  $\alpha > 0$ , then:

$$\log \rho^{-1} \sim O(n^\alpha)$$

meaning we require  $k \sim O(n^\alpha \cdot \log \tilde{\epsilon}^{-1})$ . Usually  $\tilde{\epsilon}^{-1}$  can be treated as a constant.

#### Corollary 6.4

Jacobi converges to  $x_*$  for any  $x_0$  if and only if  $\rho(G) < 1$ .

### 6.3 Computation Cost of Jacobi Iteration

Note that the Jacobi iteration only uses matrix-vector product (and  $O(n)$  operations for calculating  $D^{-1}$ ). Thus:

$$\text{computational cost per step: } \begin{cases} O(n^2) & \text{for general } A \\ O(m+n) & \text{for sparse } A \text{ with } m \text{ non-zero entries} \end{cases}$$

Thus the total computational cost is:

$$O(m+n) \times O(n^\alpha \cdot \log \tilde{\epsilon}^{-1}) = O((m+n)n^\alpha \cdot \log \tilde{\epsilon}^{-1})$$

### 6.4 Jacobi Iteration for 1D Discrete Laplacian

Recall that the Laplacian equation in 1D is:

$$\begin{cases} -u_{xx} = f & x \in (0, 1) \\ u(0) = u(1) = 0 \end{cases}$$

Using central difference, we have  $Ax = b$ , where:

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix}$$

We have  $Au = \lambda u$ , i.e.

$$\begin{cases} -u_{j-1} + 2u_j - u_{j+1} = \lambda u_j & j = 1, \dots, n \\ u_0 = u_{n+1} = 0 \end{cases}$$

Recall that this is a discrete difference eq. whose solutions are given by:

$$u_j = c_1 \alpha_1^j + c_2 \alpha_2^j$$

where  $c_1, c_2$  are constants,  $\alpha_1, \alpha_2$  are roots of

$$-1 + 2\alpha - \alpha^2 = \lambda\alpha.$$

i.e.  $\alpha_1 + \alpha_2 = 2 - \lambda$  and  $\alpha_1\alpha_2 = 1$ . Because  $u_0 = u_{n+1} = 0$ , we have:

$$\begin{cases} c_1 + c_2 = u_0 = 0 \\ c_1\alpha_1^{n+1} + c_2\alpha_2^{n+1} = u_{n+1} = 0 \end{cases}.$$

$$\begin{aligned} \det \begin{pmatrix} 1 & 1 \\ \alpha_1^{n+1} & \alpha_2^{n+1} \end{pmatrix} = 0 &\iff \alpha_1^{n+1} = \alpha_2^{n+1} \\ &\iff \left( \frac{\alpha_1}{\alpha_2} \right)^{n+1} = 1 \\ &\iff \frac{\alpha_1}{\alpha_2} = e^{i \cdot \frac{2\pi}{n+1} \cdot k}, \quad k = 0, 1, \dots, n. \end{aligned}$$

Since,  $\alpha_1\alpha_2 = 1$ , we have:

$$\begin{aligned} \implies 1 &= \alpha_2^2 \frac{\alpha_1}{\alpha_2} = \alpha_2^2 e^{i \frac{2\pi}{n+1} k} \\ \implies \begin{cases} \alpha_2 &= e^{-i \frac{\pi}{n+1} k} \\ \alpha_1 &= e^{i \frac{\pi}{n+1} k} \end{cases}, \quad k = 0, 1, \dots, n. \end{aligned}$$

Thus:

$$\alpha_1 + \alpha_2 = 2 - \lambda \implies \lambda = 2 - (\alpha_1 + \alpha_2) = 2 - 2\operatorname{Re}(e^{i \frac{\pi}{n+1} k}) = 2(1 - \cos \left( \frac{\pi}{n+1} k \right))$$

However, there are  $n+1$  values of  $k$ , but there are only  $n$  eigenvalues of  $A$ . When  $k = 0$ ,  $\alpha_1 = \alpha_2 = 1$ . However, in this case, we have:

$$u_j = c_1\alpha_1^j + c_2\alpha_2^j \implies c_1 + c_2 = 0$$

which is a contradiction since  $c_1 + c_2 = 0 \implies u = 0$ . Thus when  $k = 1, 2, \dots, n$ , we can find the corresponding  $u$  (left as homework).

Thus, the eigenvalues of  $A$  are:

$$\lambda_k = 2(1 - \cos \left( \frac{\pi}{n+1} k \right)), \quad k = 1, 2, \dots, n$$

Consequently:

$$\rho(G) = \max_{k=1,2,\dots,n} \left| 1 - \frac{1}{2}\lambda_k \right| = \max_{k=1,2,\dots,n} \left| \cos \left( \frac{\pi}{n+1} k \right) \right| = \cos \left( \frac{\pi}{n+1} \right) < 1$$

Thus the Jacobi converges, and:

$$\|x_k - x_*\|_2 \leq \|G\|_g \|x_{k-1} - x_*\|_2 = \rho(G) \|x_{k-1} - x_*\|_2$$

Thus,

$$\rho = \rho(G) = \cos \frac{\pi}{n+1} = 1 - 2\sin^2 \frac{\pi}{2(n+1)} = 1 - O\left(\frac{1}{n^2}\right)$$

This gives us  $\alpha = 2$ .

As such, the number of iteration for  $\|x_k - x_*\|_2 \leq \tilde{\epsilon}$  is:

$$k \sim O(n^2 \cdot \log \tilde{\epsilon}^{-1})$$

Since we only need matrix product, we only need  $O(n)$  FLOPs per iteration, meaning that the total FLOPs needed is:

$$O(n^3 \cdot \log \tilde{\epsilon}^{-1})$$

As a comparison, Gaussian Elimination needs  $O(n^3)$ . Thus, the Jacobi iteration (in this version), is not efficient.

## 7 February 23rd, 2021

### 7.1 Jacobi for 2D Discrete Laplacian

For the 2D Laplacian, we have:

$$\begin{cases} -u_{xx} - u_{yy} = f & (x, y) \in \Omega = (0, 1)^2 \\ u(x, y) = 0 & \text{on } \partial\Omega \end{cases}$$

By central difference:

$$A_2 x = b, x \in \mathbb{R}^N \quad A_2 \in \mathbb{R}^{N \times N}$$

with  $N = n^2$  and:

$$A_2 = A \otimes I + I \otimes A$$

, where  $A$  is the 1D discrete Laplacian.

**Definition 7.1 (Kronecker Product (Tensor Product)).**

$$B \otimes C = \begin{bmatrix} b_{11}C & b_{12}C & \dots & b_{1q}C \\ \vdots & & & \vdots \\ b_{p1}C & b_{p2}C & \dots & b_{pq}C \end{bmatrix}$$

#### Theorem 7.2

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$$

#### Lemma 7.3

The eigenvalues of  $A_2$  are  $\lambda_i + \lambda_j$ , where  $\lambda_i, \lambda_j$  are eigenvalues of  $A$  and  $1 \leq i, j \leq n$ .

*Proof.* Let  $(\lambda_i, u_i)$  be eigenpairs of  $A$ . Then:

$$A_2(u_i \otimes u_j) = (A \otimes I)(u_i \otimes u_j) + (I \otimes A)(u_i \otimes u_j)$$

$$\begin{aligned}
A_2(u_i \otimes u_j) &= (A \otimes I)(u_i \otimes u_j) + (I \otimes A)(u_i \otimes u_j) \\
&= Au_i \otimes u_j + u_j \otimes Au_i \\
&= \lambda_i u_i \otimes u_j + u_i \otimes (\lambda_j u_j) \\
&= (\lambda_i + \lambda_j)(u_i \otimes u_j).
\end{aligned}$$

□

Thus:

$$\begin{aligned}
G_2 &= I - D_2^{-1} A_2 \\
&= I - \frac{1}{4} A_2.
\end{aligned}$$

Meaning that the eigenvalues of  $G_2$  are:

$$\begin{aligned}
1 - \frac{1}{2}(\lambda_i + \lambda_j) &= 1 - \frac{1}{4} \left( 4 - 2 \cos \frac{i\pi}{n+1} - 2 \cos \frac{j\pi}{n+1} \right) \\
&= \frac{1}{2} \left( \cos \frac{i\pi}{n+1} + \cos \frac{j\pi}{n+1} \right).
\end{aligned}$$

Thus:

$$\rho(G_2) = \max_{1 \leq i, j \leq n} \left| \frac{1}{2} \left( \cos \frac{i\pi}{n+1} + \cos \frac{j\pi}{n+1} \right) \right| = \cos \frac{\pi}{n+1} < 1$$

Because  $G_2$  is symmetric,  $\|G_2\|_2 = \rho(G_2)$ :

$$\|x_k - x_*\|_2 \leq \rho(G_2) \cdot \|x_{k-1} - x_*\|_2$$

Similar to before, we have:

$$1 - O\left(\frac{1}{n^2}\right) = 1 - O\left(\frac{1}{N}\right)$$

This gives us  $\alpha = 1$ , meaning that:

- number of iterations needed:  $O(N \log \tilde{\epsilon}^{-1})$
- number of FLOPs needed per iterations:  $O(N)$ , which is the number of non-zero entries

Thus the total computation cost is  $O(N^2 \cdot \log \tilde{\epsilon}^{-1})$ . This is the same order as Gaussian Elimination, since  $\tilde{\epsilon}$  is usually a constant.

**Remark 7.4** — More examples of Jacobi Iteration include the strictly/irreducibly diagonally dominant matrix, which have been covered in MAT5311.

## 7.2 Jacobi for SPD Matrices

**Theorem 7.5**

Let  $A \in \mathbb{R}^{n \times n}$  be SPD. Then Jacobi converges to  $x_*$  for any  $x_0$  if and only if  $2D - A$  is SPD too.

*Proof.* Recall that Jacobi converges to  $x_*$  for any  $x_0$  if and only if  $\rho(G) < 1$ .

- Assume Jacobi converges, then:

$$\rho(I - D^{-1}A) < 1 \iff \rho(I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}) < 1$$

because  $D^{\frac{1}{2}}(I - D^{-1}A)D^{-\frac{1}{2}}$  is similar, thus meaning they share the same eigenvalue and thus spectral radius. Let  $\lambda$  be an eigenvalue of  $I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ . Then  $|\lambda| < 1$  and  $1 + \lambda$  is an eigenvalue of  $2I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ .

Since  $A$  is symmetric,  $\lambda$  is real, meaning that  $1 + \lambda$  is positive, thus meaning  $2I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$  is SPD.

Consider:

$$D^{\frac{1}{2}}(2I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}})D^{\frac{1}{2}} = 2D - A$$

which is also SPD, since they are similar.

- The reverse is very similar, just in reverse. Assume  $2D - A$  is SPD. We have:

$$\begin{aligned} 2D - A \text{ is SPD} &\implies 2I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \text{ is SPD} \\ &\implies 1 + \lambda, \text{ where } \lambda \text{ is an eigenvalue of } I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}. \end{aligned}$$

We also have:

$$\begin{aligned} A \text{ is SPD} &\implies D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = I - (I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}) \text{ is SPD} \\ &\implies 1 - \lambda > 0 \\ &\implies \lambda > -1. \end{aligned}$$

where  $\lambda$  is an eigenvalue of  $I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ . As such, we have:

$$-1 < \lambda < 1 \implies |\lambda| < 1 \implies \rho(I - D^{-1}A) < 1$$

□

**Example 7.6**

Consider the 1D Laplacian:

$$\begin{cases} A \text{ is SPD} \\ 2D - A = 4I - A \text{ is SPD} \end{cases} \implies \text{Jacobi converges}$$

### 7.3 Lower Bound of Jacobi Convergence Rate

We have:

$$\frac{\|x_k - x_*\|}{\|x_{k-1} - x_*\|} \leq \rho(G) + \epsilon$$

for an arbitrarily small  $\epsilon$ , or  $\epsilon = 0$  if  $G$  is symmetric. This is a worse-case, i.e. an upper bound. However, this factor is asymptotically optimal, meaning:

$$\lim_{k \rightarrow \infty} \frac{\|x_k - x_*\|}{\|x_{k-1} - x_*\|} \geq \rho(G)$$

As such, convergence factor  $\rho$  is **tight**.

Let us demonstrate this when  $G$  is symmetric first.

**Remark 7.7** — For nonsymmetric matrices, this is also true, we will prove later.

$|\lambda_1| > |\lambda_2|$  where  $\lambda_1, \lambda_2$  are the largest and 2nd largest eigenvalue of  $G$  in absolute value.

Let  $G = U \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} U^T$  be the eigenvalue decomposition of  $G$ , where  $U = [u_1 \ u_2 \ \dots \ u_n]$  is unitary.

Let  $x_k - x_* = z_k$ . Then:

$$z_k = G^k z_0 = \left( U \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} U^T \right)^k z_0 = U \begin{bmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_n^k \end{bmatrix} U^T z_0.$$

Denote  $U^T z_0 = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}$ . Thus we have:

$$\begin{aligned} \|z_k\|_2 &= \left\| \begin{bmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_n^k \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} \right\| \\ &= \left( \sum_{i=1}^n \lambda_i^{2k} \alpha_i^2 \right)^{1/2} \\ &= |\lambda_1|^k \left( \sum_{i=1}^n \left( \frac{\lambda_i}{\lambda_1} \right)^{2k} \alpha_i^2 \right)^{1/2}. \end{aligned}$$

Thus, we have:

- $\lim_{k \rightarrow \infty} \frac{\|z_k\|_2}{\|z_{k-1}\|_2} = \rho(G)$



*Proof.*

$$\begin{aligned}
 \lim_{k \rightarrow \infty} \frac{\|z_k\|_2}{\|z_{k-1}\|_2} &= |\lambda_1| \lim_{k \rightarrow \infty} \frac{\left( \sum_{i=1}^n \left( \frac{\lambda_i}{\lambda_1} \right)^{2k} \alpha_i^2 \right)^{1/2}}{\left( \sum_{i=1}^n \left( \frac{\lambda_i}{\lambda_1} \right)^{2(k-1)} \alpha_i^2 \right)^{1/2}} \\
 &= |\lambda_1| \lim_{k \rightarrow \infty} \frac{\alpha_1^2 + \left( \frac{\lambda_2}{\lambda_1} \right)^{2k} \alpha_2^2 + \dots}{\alpha_1^2 + \left( \frac{\lambda_2}{\lambda_1} \right)^{2(k-1)} \alpha_2^2 + \dots} \\
 &= |\lambda_1| \\
 &= \rho(G).
 \end{aligned}$$

□

- $|\langle z_k, u_j \rangle| = |\lambda_j^k \alpha_j| = |\lambda_j|^k |\langle z_0, u_j \rangle|$ , meaning that the convergence speed of error projected onto  $u_j$  is  $|\lambda_j|$ .

**Remark 7.8** — This means that different error components have different convergence speed.

- The error converges to the same direction as  $u_1$ .

*Proof.*

$$\begin{aligned}
 \lim_{k \rightarrow \infty} \frac{|\langle z_k, u_j \rangle|}{\|z_k\|_2} &= \lim_{k \rightarrow \infty} \frac{|\lambda_j|^k |\langle z_0, u_j \rangle|}{|\lambda_1|^k \left( \sum_{i=1}^n \left( \frac{\lambda_i}{\lambda_1} \right)^{2k} \alpha_i^2 \right)^{1/2}} \\
 &= \lim_{k \rightarrow \infty} \frac{|\lambda_j|^k}{|\lambda_1|^k} \cdot \frac{|\alpha_1|}{|\alpha_1|} \\
 &= \lim_{k \rightarrow \infty} \left| \frac{\lambda_j}{\lambda_1} \right| \\
 &= \begin{cases} 1 & j = 1 \\ 0 & j \neq 1 \end{cases}.
 \end{aligned}$$

□

**Example 7.9**

Let's consider the 1D discrete Laplacian:

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix}, \quad G = I - \frac{1}{2}A$$

Recall that the eigenvalues of  $A$  are :

$$\lambda_j(A) = 2 \left( 1 - \cos \frac{j\pi}{n+1} \right), \quad j = 1, 2, \dots, n$$

meaning that the eigenvalues of  $G$  are:

$$\lambda_j(G) = 1 - \frac{1}{2}\lambda_j(A) = \cos \frac{j\pi}{n+1}, \quad j = 1, 2, \dots, n$$

Let  $u_j$  be the eigenvectors of  $A$  (and of  $G$ , since it is a shift of  $A$ ). Let the error be

$$z_k = x_k - x_* = \sum_{j=1}^n \alpha_j^{(k)} u_j$$

Then the convergence speed of  $\alpha_j^{(k)}$  is  $\lambda_j(G)$ . As such, we have:

- If  $j$  is close to 1 or  $n$ , then  $|\lambda_j(G)|$  is close to 1, meaning it has a slower convergence.
- If  $j$  is close  $\frac{n}{2}$ , then  $|\lambda_j(G)|$  is close to 0, meaning it has a faster convergence.

**Remark 7.10** — To improve Jacobi, in the Multigrid, we use different grids. The main motivation is that the convergence speed of different components depends on the eigenvalues. For components that are difficult to decrease, we project them onto a coarse grid to reduce the error.

## 8 February 25th, 2021

### 8.1 Gauss-Seidel Iteration

Recall that in Jacobi,  $\xi_i^{(k+1)}$  are updated parallelly in Algorithm 9.

**Gauss-Seidel** is just modifying Jacobi to be updated “successively”. When we modify  $\xi_i^{(k+1)}$  we can use  $\xi_1^{(k+1)}, \dots, \xi_{i-1}^{(k+1)}$ . This gives us Algorithm 11.

In the matrix reformulation, recalling that  $A = D - E - F$ , where  $E$  is lower triangular and  $F$  is upper triangular, we have:

**Algorithm 11** Gauss-Seidel Iteration

---

```

1: for  $k = 0, 1, 2, \dots$  do
2:   for  $i = 1, \dots, n$  do
3:      $\xi_i^{(k+1)} = (\beta_i - \sum_{j>i} a_{ij}\xi_j^{(k)} - \sum_{j<i} a_{ij}\xi_j^{(k+1)})/a_{ii}$ 
4:   end for
5: end for

```

---

$$\begin{aligned}
x_{k+1} &= D^{-1}(b + Fx_k + Ex_{k+1}) \\
&\iff (D - E)x_{k+1} = b + Fx_k \\
&\iff x_{k+1} = (D - E)^{-1}Fx_k + (D - E)^{-1}b \\
&\iff x_{k+1} = \underbrace{(I - (D - E)^{-1}A)}_G x_k + \underbrace{(D - E)^{-1}b}_f.
\end{aligned}$$

which is in the stationary iteration form.

**Remark 8.1** — In the algorithm above, we are updating  $\xi_i^{(k+1)}$  in regular numerical order (from 1 to  $n$ ). However, we can use other ordering, such as going from  $n$  to 1 in reverse order. Gauss-Seidel is sensitive to the ordering of updating of unknowns.

**8.1.1 Convergence of Gauss-Seidel**

Since Gauss-Seidel is a stationary iteration, we can use the same framework as the Jacobi iteration, with:

$$(x_k - x_*) = G^k(x_0 - x_*)$$

**Theorem 8.2**

Gauss-Seidel converges to  $x_*$  for any  $x_0$  if and only if  $\rho(G) < 1$ .

The convergence speed is  $\|G\|$ , as we have:

$$\|x_{k+1} - x_*\| \leq \|G\|\|x_k - x_*\|$$

In particular, we have:

$$\|G\| = \begin{cases} \rho(G) & \text{if } G \text{ is symmetric and } \|\cdot\| = \|\cdot\|_2 \\ \rho(G) + \epsilon & \text{otherwise} \end{cases}$$

with  $\epsilon$  arbitrarily small.

**Theorem 8.3**

Let  $A \in \mathbb{R}^{n \times n}$  be SPD. Then  $\rho(G) < 1$ .

*Proof.* Recall  $G = I - (D - E)^{-1}A$ . Let  $(\lambda, u)$  be an eigenpair of  $G$  (note that  $\lambda, u$  may be complex). We have:

$$\begin{aligned} Gu &= \lambda u \\ (I - (D - E)^{-1}Au) &= \lambda u \\ (D - E)^{-1}Fu &= \lambda u \\ Fu &= \lambda(D - E)^{-1}u. \end{aligned}$$

Since  $A$  is symmetric, we have:

$$A = A^T \implies F = E^T.$$

Thus, we have:

$$\begin{aligned} \lambda(D - E)u &= E^T u \\ \lambda u^*(D - E)u &= u^* E^T u. \end{aligned}$$

by left-multiplying both side by  $u^* = \bar{u}^T$

Let us set  $u^* E^T u = \alpha + i\beta \in \mathbb{C}$  and  $u^* Du = \delta \in \mathbb{R}$ , since  $D$  is real and symmetric. Then, we have:

$$u^* E^T u = (u^* E^T u)^T = u^T E(u^*)^T = \overline{u^* E u} = \overline{u^* E u} = \alpha - i\beta$$

Since  $A$  is SPD,  $D$  is also SPD, giving us  $\lambda = u^* Du > 0$ .

Thus

$$\lambda(\delta - (\alpha + i\beta)) = \alpha - i\beta \implies \lambda = \frac{\alpha - i\beta}{(\delta - \alpha) - i\beta}$$

Giving us:

$$|\lambda|^2 = \frac{(\alpha^2 + \beta^2)}{(\delta - \alpha)^2 + \beta^2}$$

Because:

$$(\delta - \alpha)^2 + \beta^2 = \delta^2 + \alpha^2 + \beta^2 - 2\alpha\delta = (\alpha^2 + \beta^2) + \underbrace{\delta}_{>0}(\delta - 2\alpha)$$

we need to check the sign of  $\delta - 2\alpha$ .

Since  $A$  is SPD,  $u^* Au > 0 \implies u^*(D - E - E^T)u = \delta - (\alpha + i\beta) - (\alpha - i\beta) = \delta - 2\alpha > 0$ . As such:

$$(\delta - \alpha)^2 + \beta^2 > \alpha^2 + \beta^2 \implies |\lambda|^2 < 1$$

□

**Remark 8.4** — This is more general than Jacobi, which is convergent if and only if  $2D - A$  is SPD.

**Example 8.5**

Recall that the 1D discrete Laplacian is SPD, meaning that Gauss-Seidel converges. Similarly for the 2D case.

**Example 8.6**

Gauss-Seidel converges for irreducible diagonally dominant matrices. This is used for laplacian on irregular domains.

**8.2 Acceleration of G-S / and Jacobi (SOR)**

Here we try to accelerate Gauss-Seidel and Jacobi by considering momentum acceleration. We can consider the iteration of  $x_k$  to  $x_{k+1}$  as a movement by the Gauss-Seidel iteration. The idea of SOR is to consider the momentum of the movement until some stage.

For Gauss-Seidel, the components are:

- $x_k : \xi_i^{(k)}$
- $x_{k+1} : \xi_i^{(k+1)} = (\beta_i - \sum_{j>i} a_{ij}\xi_j^{(k)} - \sum_{j<i} a_{ij}\xi_j^{(k+1)})/a_{ii}$

To consider the momentum, we would have:

$$\begin{aligned}\xi_i^{(k+1)} &= \xi_i^{(k)} + \omega(\xi_i^{(k+1)} - \xi_i^{(k)}) \\ &= \xi_i^{(k)} + \omega \left( (\beta_i - \sum_{j>i} a_{ij}\xi_j^{(k)} - \sum_{j<i} a_{ij}\xi_j^{(k+1)})/a_{ii} - \xi_i^{(k)} \right).\end{aligned}$$

for some  $\omega > 1$ , giving us the Algorithm 12.

**Algorithm 12** SOR Iteration

---

```

1: for  $k = 0, 1, 2, \dots$  do
2:   for  $i = 1, \dots, n$  do
3:      $\xi_i^{(k+1)} = \xi_i^{(k)} + \omega \left[ (\beta_i - \sum_{j>i} a_{ij}\xi_j^{(k)} - \sum_{j<i} a_{ij}\xi_j^{(k+1)})/a_{ii} - \xi_i^{(k)} \right]$ 
4:   end for
5: end for

```

---

In the matrix reformulation, we have:

$$x_{k+1} = x_k + \omega [D^{-1}(b + Fx_k + Ex_{k+1}) - x_k]$$

Expressing this in its stationary form, we have:

$$x_{k+1} = \underbrace{(D - \omega E)^{-1} ((1 - \omega)D + \omega F)}_{G_\omega} x_k + \underbrace{\omega(D - \omega E)^{-1} b}_{f_\omega}$$

Now the question is finding a suitable  $\omega$ .

## 8.2.1 Convergence of SOR

**Theorem 8.7**

If  $\omega \notin (0, 2)$ , then  $\rho(G_\omega) > 1$ .

*Proof.* We have:

$$\det(G_\omega) = \det((D - \omega E)^{-1}) \cdot \det((1 - \omega)D + \omega F)$$

- Since  $D - \omega E$  is lower triangular,  $(D - \omega E)^{-1}$  is also lower triangular. Thus,  $\det((D - \omega E)^{-1})$  is the product of its diagonals, which are the inverse of the diagonal of  $D - \omega E$  which are  $\frac{1}{a_{ii}}$ . As such, we have:

$$\det((D - \omega E)^{-1}) = \prod_{i=1}^n a_{ii}^{-1}$$

- Since  $(1 - \omega)D + \omega F$  is upper triangular, we have:

$$\det((1 - \omega)D + \omega F) = \prod_{i=1}^n (1 - \omega) a_{ii}$$

Thus, we have:

$$\det(G_\omega) = \prod_{i=1}^n (1 - \omega) a_{ii} \cdot a_{ii}^{-1} = (1 - \omega)^n = \prod_{i=1}^n \lambda_i$$

This gives us:

$$\begin{aligned} |1 - \omega|^n &= |\lambda_1| |\lambda_2| \dots |\lambda_n| \leq (\rho(G_\omega))^n \\ \implies |1 - \omega| &\leq \rho(G_\omega). \end{aligned}$$

□

**Corollary 8.8**

In order for SOR to converge, we must have  $\omega < 2$ .

- If  $\omega = 1$ , we have Gauss Seidel
- If  $\omega \in (0, 1)$ , we have successive under relaxation
- If  $\omega \in (1, 2)$  we have successive over relaxation (SOR).

**Theorem 8.9**

If  $A$  is SPD, then  $\rho(G_\omega) < 1$  for all  $\omega \in (0, 2)$ .

*Proof.* Similar to Gauss-Seidel.

□

How do we pick the optimal  $\omega$ ? Roughly speaking, with the optimal  $\omega$ :

- $\rho(G_{\omega_{\text{opt}}}) \sim \rho(G_1)^2$ . Where  $G_1$  is the iteration matrix in G-S.

**Remark 8.10** — This means that SOR is roughly two times faster than G-S, since it is the square.

- $\rho(G_1) \sim (\rho(G_{\text{Jacobi}}))^2$ .

## 9 March 2nd, 2021

### 9.1 General Framework of Stationary Iterations

#### 9.1.1 Matrix Splitting

Given non singular matrix  $A \in \mathbb{R}^{n \times n}$ , we split it as:

$$A = M - N,$$

where  $M, N \in \mathbb{R}^{n \times n}$ . Then:

$$Ax = b \iff (M - N)x = b \iff Mx = Nx + b$$

Now, if we assume that  $M$  is easy to invert, e.g. diagonal, we can obtain:

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

We can then construct an iteration:

$$x_{k+1} = (I - M^{-1}A)x_k + M^{-1}b$$

For different stationary iterations, we have:

**Jacobi:**  $M = A$

**Gauss-Seidel:**  $M = D - E$

**Backward Gauss-Seidel:**  $M = D - F$

**SOR:**  $M = \frac{1}{\omega}(D - \omega E)$

For the convergence, the algorithm converges to the solution of  $Ax = b$  with any  $x_0$  if and only if  $\rho(I - M^{-1}A) < 1$ .

#### 9.1.2 Preconditioned Richardson Iteration

Assume  $A$  is SPD. Solve  $Ax = b$  is the same as solving the optimizaiton problem:

$$\min_{x \in \mathbb{R}^n} f(x), \quad f(x) = \frac{1}{2}x^T Ax - x^T b$$

This is because:

$$\nabla f(x) = Ax - b \implies \nabla^2 f(x) = A \implies f \text{ is convex}$$

**Remark 9.1** — Since  $A$  is SPD,  $f(x)$  is strongly convex.

Since this is a convex optimization problem, we can apply gradient descent:

$$\begin{aligned} x_{k+1} &= x_k - \alpha \nabla f(x_k) \\ \implies x_{k+1} &= x_k - \alpha \nabla \alpha(Ax_k - b) \end{aligned}$$

i.e.

$$x_{k+1} = (I - \alpha A)x_k + \alpha b$$

where  $\alpha > 0$  is a constant. This is called the **Richardson Iteration**.

**Remark 9.2** — Richardson is a special case of matrix splitting where  $M = \frac{1}{\alpha}I$ .

For the convergence, we have:

$$G = I - \alpha A.$$

Let  $\Lambda(A) = \{\lambda : \lambda \text{ is an eigenvalue of } A\}$ . We have:

$$\rho(G) = \max_{\lambda \in \Lambda(A)} |1 - \alpha \lambda|$$

If we let  $\lambda_{\min}$  and  $\lambda_{\max}$  be the min and max eigenvalues of  $A$ , we have:

$$\rho(G) = \max\{|1 - \alpha \lambda_{\min}|, |1 - \alpha \lambda_{\max}|\}$$

Since  $|1 - \alpha \lambda|$  is a piecewise linear function. By direct calculation, we have:

$$\rho(G) < 1 \implies |1 - \alpha \lambda_{\max}| = \alpha \lambda_{\max} - 1 < 1 \implies \alpha < \frac{2}{\lambda_{\max}}$$

Thus, we have:

$$\alpha \in (0, \frac{2}{\lambda_{\max}})$$

for the iteration to converge.

In order to have optimal convergence speed, we consider:

$$\alpha_{\text{opt}} = \arg \min_{\alpha} \rho(G) \iff \min_{\alpha > 0} \max_{\lambda \in \Lambda(A)} |1 - \alpha \lambda|$$

Then it is easy to check that:

$$1 - \alpha_{\text{opt}} = \alpha_{\text{opt}} \lambda_{\max} - 1 \implies \alpha_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}}$$

and

$$\rho_{\text{opt}}(G) = 1 - \alpha_{\text{opt}} \lambda_{\min} = \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} = \frac{\gamma - 1}{\gamma + 1}$$

where  $\gamma = \frac{\lambda_{\max}}{\lambda_{\min}} = \frac{\|A\|_2}{1/\|A^{-1}\|_2} = \|A\|_2 \cdot \|A^{-1}\|_2$  which is the **condition number** of  $A$  as shown in Assignment 1.



**Remark 9.3** — The convergence is slow if  $\gamma$  is big, as such we want to see if we can improve it.

**Remark 9.4** — Intuitively, we would have a slow convergence if we have a flat level set. Meanwhile, if we have a round level set, the gradient descent would be fast. This is because of the ratio of  $\lambda_{\max}$  and  $\lambda_{\min}$ .

In addition, we should note that the gradient depends on the inner product in  $\mathbb{R}^n$ . As such, to improve the Richardson iteration, we change the inner product such that the level set of  $f(x)$  in the new inner product space is very round.

**Definition 9.5 (Weighted Inner Product).**

$$\langle x, y \rangle_P = x^T P y, \quad \text{where } P \text{ is SPD.}$$

Under the weighted inner product space, since:

$$f(y) = f(x) + \langle y - x, Ax - b \rangle + o(\|x - y\|_2)$$

we have:

$$f(y) = f(x) + \langle y - x, P^{-1}(Ax - b) \rangle_P + o(\|x - y\|_P)$$

Thus, we have:

$$\nabla_P f(x) = P^{-1}(Ax - b)$$

**Remark 9.6** — This is because the gradient ( $\langle y - x, Ax - b \rangle$ ) is a linear approximation at point  $x$ . This is the definition of the **Frechet derivative** in Hilbert spaces.

**Remark 9.7** —  $o(\|x - y\|_2) = o(\|x - y\|_P)$  since vector norms are equivalent in finite dimensional space.

As such, the gradient descent under weighted inner product is:

$$x_{k+1} = x_k - \alpha P^{-1}(Ax_k - b)$$

Note that  $\alpha$  can be absorbed into  $P^{-1}$  since  $P$  is SPD, thus giving us:

$$x_{k+1} = x_k - P^{-1}(Ax_k - b) \iff x_{k+1} = (I - P^{-1}A)x_k + P^{-1}b$$

This is called the **preconditioned gradient descent**. Similarly, for the convergence, we have:

$$\rho(I - P^{-1}A) < 1$$

and the optimal convergence rate is:

$$p_{\text{opt}} = \frac{\gamma(P^{-1}A) - 1}{\gamma(P^{-1}A) + 1}$$

where  $\gamma(P^{-1}A)$  is the condition number of  $P^{-1}A$ .

**Remark 9.8** — Note that the condition number of  $P$  before and after absorbing  $\alpha$  is the same, since we are just scaling it.

To be a good preconditioner,  $P$  has to satisfy the following:

1.  $P$  is SPD.
2.  $P$  is easy to invert so that  $P^{-1}$  is easy to compute
3.  $\gamma(P^{-1}A)$  has to be small (or equivalent  $P \approx A$ ).

There are a few special cases:

- $P = D$  (diagonal part of  $A$ ) - Jacobi iteration
- Symmetric G-S

### 9.1.3 Projection Methods

Let  $K$  and  $L$  be two  $m$ -dimensional subspaces in  $\mathbb{R}^n$ . Given  $x_0 \in \mathbb{R}^n$ , we obtain a better solution  $\tilde{x}$  of  $Ax = b$  by:

$$\begin{cases} \text{Find } \tilde{x} \in x_0 + K \\ \text{s.t. } b - A\tilde{x} \perp L \end{cases} \iff \begin{cases} \tilde{x} = x_0 + \delta, & \delta \in K \\ \langle b - A(x_0 + \delta), \omega \rangle = 0, & \forall \omega \in L \end{cases}$$

A pictorial illustration is shown in Figure 1.

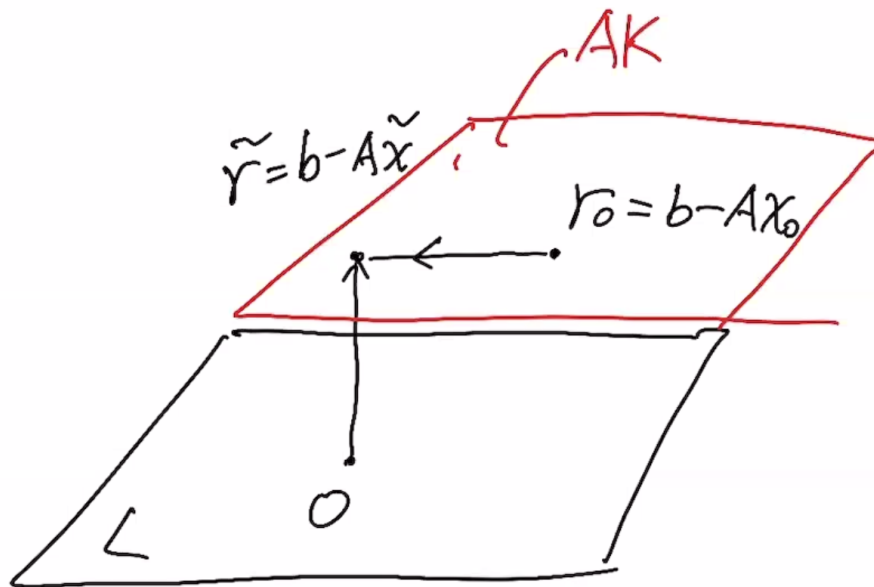


Figure 1: Pictorial Representation of the Projection Method

If we choose  $K = L = \text{span}\{e_i\}$

- $\tilde{x} = x_0 + \delta$ ,  $\delta \in \text{span}\{e_i\}$  (only the  $i$ -th component of  $x_0$  is changed)
- $b - A\tilde{x} \perp \text{span}\{e_i\}$  (the  $i$ -th equation has an error 0).

we obtain Gauss-Seidel.

There are a few other variants of the projection methods. For example, we can choose two families of subspaces:  $K_i, L_i, i = 1, \dots, \ell$ . Given  $x_0$ , we obtain  $\tilde{x}$  by:

$$\tilde{x} = x_0 + \delta_1 + \dots + \delta_\ell, \quad \text{where } \begin{cases} \delta_i \in K_i \\ b - A(x_0 + \delta_i) \perp L_i \end{cases}$$

If we have  $K_i = L_i = \text{span}\{e_i\}$  we have the Jacobi iteration.

We can have several other choices of  $K$  and  $L$ :

**Multigrid Method:**  $K = L = \text{span}\{e_1\} \dots \text{span}\{e_n\}$  on fine grid.

Then we do  $\text{span}\{e_1\} \dots \text{span}\{e_{n/2}\}$  on the coarse grid, etc.

**Domain Decomposition:** We first partition  $\Omega$  into overlapping spaces into  $\Omega_1, \Omega_2$ , and then we set  $K = L = \text{span}\{e_i, i \in \Omega_1\}$ , and then  $\text{span}\{e_i, i \in \Omega_2\}$ .

**Remark 9.9** — For both the methods mentioned above,  $K$  and  $L$  are fixed, making the iterative methods fixed.

# Index

$p$  norm, 20

banded matrix, 16

Cauchy-Schwartz inequality, 21

condition number, 40

convergence factor, 27

direct method, 4

Frechet derivative, 41

gauss transformation, 5

Gauss-Seidel Iteration, 34

gaussian elimination, 4

Holder's inequality, 21

matrix norm, 22

norm, 20

pivot entry, 11

Richardson iteration, 40

SOR iteration, 37

spare matrix, 18

spectral radius, 23

stationary matrix, 19

symmetric positive definite (SPD), 13

tri-diagnoal matrix, 16

weighted inner product, 41

weighted norm, 21