

Numerical Analysis

SMIE SYSU

Chang-Dong Wang

Homepage: www.scholat.com/~ChangDongWang

Course website: www.scholat.com/course/SMIENA

Email: wangchd3@mail.sysu.edu.cn

QQ Group: 342983926

Outline

- [March 11](#)
- [March 18](#)
- [March 25](#)
- [April 1](#)

Systems of Equations

- Gaussian Elimination
- The LU Factorization
- Sources of Error
- The $PA = LU$ Factorization
- Iterative Methods
- Methods for symmetric positive-definite matrices
- Systems of Nonlinear Equations

Gaussian Elimination

Consider the system

$$\begin{aligned}x + y &= 3 \\ 3x - 4y &= 2.\end{aligned}$$

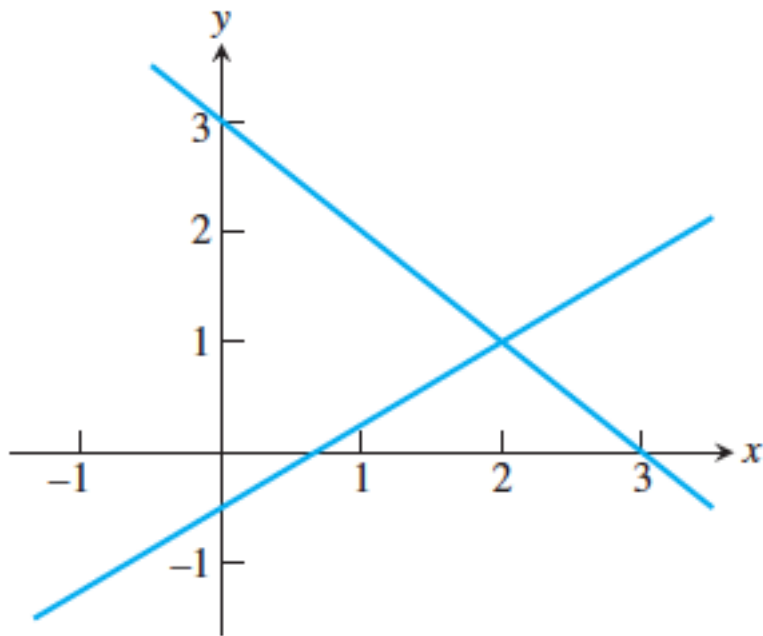


Figure 2.1 Geometric solution of a system of equations. Each equation corresponds to a line in the plane. The intersection point is the solution.

Naive Gaussian elimination S1



- Exchange: Swap one equation for another.
- Replacement: Add or subtract a multiple of one equation from another.
- Scaling: Multiply an equation by a nonzero constant.

$$\begin{array}{l} x + y = 3 \\ 3x - 4y = 2 \end{array} \xrightarrow{\text{Elimination}} \begin{array}{l} x + y = 3 \\ -7y = -7 \end{array} \xrightarrow{\text{Back-substitution}} \begin{array}{l} -7y = -7 \rightarrow y = 1 \\ x + y = 3 \rightarrow x + (1) = 3 \rightarrow x = 2 \end{array}$$

Naive Gaussian elimination S2

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & | & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & | & b_2 \\ \vdots & \vdots & \dots & \vdots & | & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & | & b_n \end{bmatrix} \xrightarrow{\text{Elimination step}} \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & | & b_1 \\ 0 & a_{22} & \dots & a_{2n} & | & b_2 \\ \vdots & \vdots & \ddots & \vdots & | & \vdots \\ 0 & 0 & \dots & a_{nn} & | & b_n \end{bmatrix}$$

Back-substitution step

```

for j = 1 : n-1
    if abs(a(j,j)) < eps;
        error('zero pivot encountered'); end
    for i = j+1 : n
        mult = a(i,j)/a(j,j);
        for k = j+1:n
            a(i,k) = a(i,k) - mult*a(j,k);
        end
        b(i) = b(i) - mult*b(j);
    end
end

```

```

for i = n : -1 : 1
    for j = i+1 : n
        b(i) = b(i) - a(i,j)*x(j);
    end
    x(i) = b(i)/a(i,i);
end

```

$$\begin{aligned}
 x_1 &= \frac{b_1 - a_{12}x_2 - \dots - a_{1n}x_n}{a_{11}} \\
 x_2 &= \frac{b_2 - a_{23}x_3 - \dots - a_{2n}x_n}{a_{22}} \\
 &\vdots \\
 x_n &= \frac{b_n}{a_{nn}}.
 \end{aligned}$$

Naive Gaussian elimination S3

- Example:

Apply Gaussian elimination in tableau form for the system of three equations in three unknowns:

$$x + 2y - z = 3$$

$$2x + y - 2z = 3$$

$$-3x + y + z = -6.$$

$$\left[\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 2 & 1 & -2 & 3 \\ -3 & 1 & 1 & -6 \end{array} \right] \xrightarrow{\quad} \left[\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 0 & -3 & 0 & -3 \\ -3 & 1 & 1 & -6 \end{array} \right] \xrightarrow{\quad} \left[\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 0 & -3 & 0 & -3 \\ 0 & 7 & -2 & 3 \end{array} \right]$$

$$x = 3 - 2y + z$$

$$-3y = -3$$

$$-2z = -4$$

$$x + 2y - z = 3$$

$$-3y = -3$$

$$-2z = -4,$$

$$\left[\begin{array}{ccc|c} 1 & 2 & -1 & 3 \\ 0 & -3 & 0 & -3 \\ 0 & 0 & -2 & -4 \end{array} \right]$$

Operation counts

Operation count for the elimination step of Gaussian elimination

The elimination step for a system of n equations in n variables can be completed in $\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n$ operations.

Operation count for the back-substitution step of Gaussian elimination

The back-substitution step for a triangular system of n equations in n variables can be completed in n^2 operations.

- The time complexity of Naïve Gaussian elimination is $O(n^3)$.

Coding assignment (Computer Problems 1 at page 79)

Put together the code fragments in this section to create a MATLAB program for “naive” Gaussian elimination (meaning no row exchanges allowed). Use it to solve the systems of Exercise 2.

The LU Factorization

- Consider the problem of solving a series of linear systems

$$Ax = b_1$$

$$Ax = b_2$$

$$\vdots$$

$$Ax = b_k$$

- How to avoid the redundant elimination steps used in Naïve Gaussian elimination?
- LU factorization.

Matrix form of Gaussian elimination S1

- The linear system can be written as a matrix form $Ax = b$, e.g., the previous linear system equals

$$\begin{array}{ccc} \begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} & \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} & = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \\ \text{coefficient matrix} & & \text{right-hand-side vector} \end{array}$$

- Definition

An $m \times n$ matrix L is **lower triangular** if its entries satisfy $l_{ij} = 0$ for $i < j$. An $m \times n$ matrix U is **upper triangular** if its entries satisfy $u_{ij} = 0$ for $i > j$. \square

- LU factorization: $A=LU$

Matrix form of Gaussian elimination S2

- Example: Find the LU factorization for the matrix $A = \begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix}$

$$\begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} \xrightarrow[\text{from row 2}]{\text{subtract } 3 \times \text{row 1}} \begin{bmatrix} 1 & 1 \\ 0 & -7 \end{bmatrix} = U$$

$$L = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}$$

$$LU = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & -7 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} = A$$

Matrix form of Gaussian elimination S3

- Example: Find the LU factorization of $A = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix}$

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix} \xrightarrow[\text{from row 2}]{\text{subtract } 2 \times \text{row 1}} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ -3 & 1 & 1 \end{bmatrix} \xrightarrow[\text{from row 3}]{\text{subtract } -3 \times \text{row 1}} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 7 & -2 \end{bmatrix}$$

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

Diagram illustrating the steps of Gaussian elimination and the construction of L and U matrices. Red dashed arrows show the elimination steps: from row 1 to row 2 (multiplier 2), from row 1 to row 3 (multiplier -3), and from row 2 to row 3 (multiplier -7/3). Blue solid arrows show the row operations: subtract 2 × row 1 from row 2, and subtract -3 × row 1 from row 3.

$$LU = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -\frac{7}{3} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{bmatrix} = A$$

Matrix form of Gaussian elimination S4

- Fact 1:

Let $L_{ij}(-c)$ denote the lower triangular matrix whose only nonzero entries are 1's on the main diagonal and $-c$ in the (i, j) position. Then $A \rightarrow L_{ij}(-c)A$ represents the row operation "subtracting c times row j from row i ."

$$\begin{bmatrix} 1 & 0 & 0 \\ -c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} - ca_{11} & a_{22} - ca_{12} & a_{23} - ca_{13} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

- Fact 2: $L_{ij}(-c)^{-1} = L_{ij}(c)$

$$\begin{bmatrix} 1 & 0 & 0 \\ -c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Fact 3: The following matrix product equation holds.

$$\begin{bmatrix} 1 & & \\ c_1 & 1 & \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ c_2 & & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & c_3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ c_1 & 1 & \\ c_2 & c_3 & 1 \end{bmatrix}$$

Back substitution with the LU factorization

Once L and U are known, the problem $Ax = b$ can be written as $LUx = b$. Define a new “auxiliary” vector $c = Ux$. Then back substitution is a two-step procedure:

- (a) Solve $Lc = b$ for c .
- (b) Solve $Ux = c$ for x .

Both steps are straightforward since L and U are triangular matrices.

$$\text{LU} \quad \begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} = LU = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & -7 \end{bmatrix}$$

$$\text{Step (a)} \quad \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \longrightarrow c_1 = 3, c_2 = -7$$

$$\text{Step (b)} \quad \begin{bmatrix} 1 & 1 \\ 0 & -7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ -7 \end{bmatrix} \longrightarrow \begin{array}{l} x_1 + x_2 = 3 \\ -7x_2 = -7 \end{array} \longrightarrow x_2 = 1, x_1 = 2.$$

Complexity of the LU factorization

That is, we are presented with the set of problems

$$Ax = b_1$$

$$Ax = b_2$$

$$\vdots$$

$$Ax = b_k$$

with various right-hand side vectors b_i

Methods	# Elimination operations	# Back-substitution operations
Naïve Gaussian	$2kn^3/3$	kn^2
LU factorization	$2n^3/3$	$2kn^2$

Coding assignment (Computer Problems 1 at page 85)

Use the code fragments for Gaussian elimination in the previous section to write a MATLAB script to take a matrix A as input and output L and U . No row exchanges are allowed—the program should be designed to shut down if it encounters a zero pivot. Check your program by factoring the matrices in Exercise 2.

Sources of Error

- Two major potential sources of error in Gaussian elimination:
 - ✧ Error caused by ill-conditioning
 - ✧ Error caused by swamping

Error magnification and condition number S1

- Definition

The **infinity norm**, or **maximum norm**, of the vector $x = (x_1, \dots, x_n)$ is $\|x\|_\infty = \max |x_i|, i = 1, \dots, n$, that is, the maximum of the absolute values of the components of x . \square

- Definition

Let x_a be an approximate solution of the linear system $Ax = b$. The **residual** is the vector $r = b - Ax_a$. The **backward error** is the norm of the residual $\|b - Ax_a\|_\infty$, and the **forward error** is $\|x - x_a\|_\infty$. \square

Error magnification and condition number S2

- Example

Find the backward and forward errors for the approximate solution $x_a = [1, 1]$ of the system

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

The correct solution is $x = [2, 1]$. In the infinity norm, the backward error is

$$\begin{aligned} \|b - Ax_a\|_\infty &= \left\| \begin{bmatrix} 3 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\|_\infty \\ &= \left\| \begin{bmatrix} 1 \\ 3 \end{bmatrix} \right\|_\infty = 3, \end{aligned}$$

and the forward error is

$$\|x - x_a\|_\infty = \left\| \begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\|_\infty = \left\| \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\|_\infty = 1.$$

Error magnification and condition number S3

- Example

Find the forward and backward errors for the approximate solution $[-1, 3.0001]$ of the system

$$x_1 + x_2 = 2$$

$$1.0001x_1 + x_2 = 2.0001.$$

the solution $[x_1, x_2] = [1, 1]$.

The backward error is the infinity norm of the vector

$$\begin{aligned} b - Ax_a &= \begin{bmatrix} 2 \\ 2.0001 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 1.0001 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 3.0001 \end{bmatrix} \\ &= \begin{bmatrix} 2 \\ 2.0001 \end{bmatrix} - \begin{bmatrix} 2.0001 \\ 2 \end{bmatrix} = \begin{bmatrix} -0.0001 \\ 0.0001 \end{bmatrix}, \end{aligned}$$

which is 0.0001. The forward error is the infinity norm of the difference

$$x - x_a = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} -1 \\ 3.0001 \end{bmatrix} = \begin{bmatrix} 2 \\ -2.0001 \end{bmatrix},$$

which is 2.0001.

Error magnification and condition number S4

- Example (Cont...)

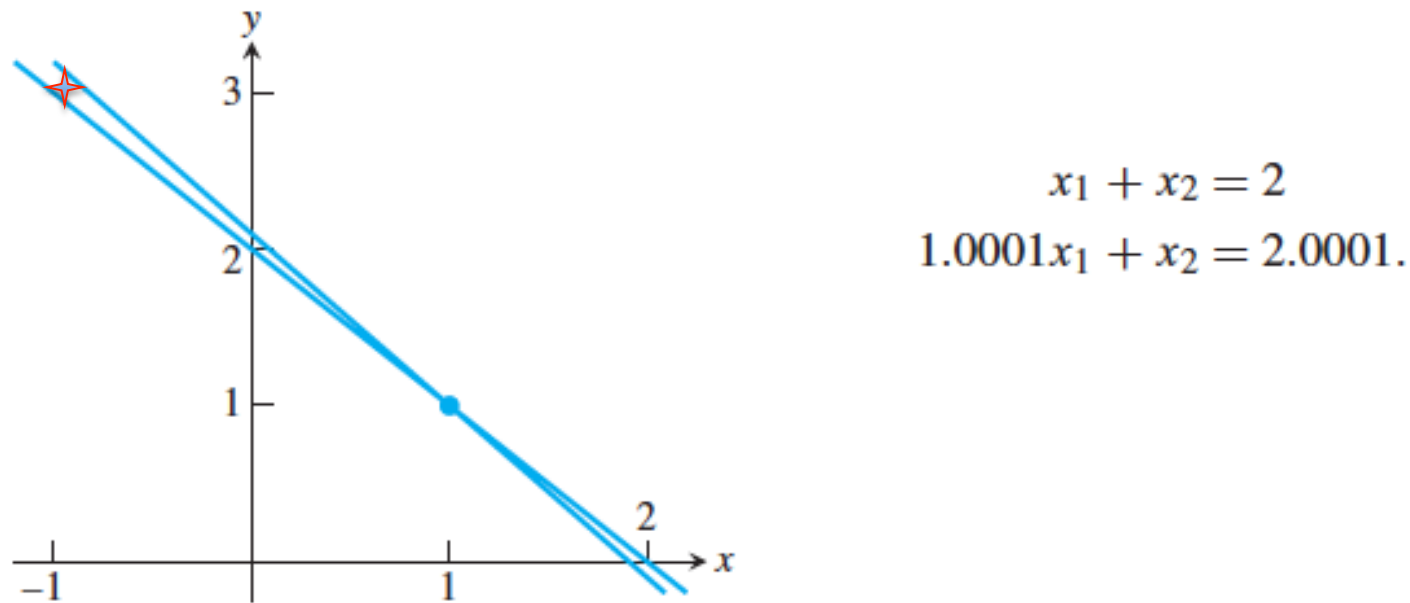


Figure 2.2 The geometry behind Example 2.11. System (2.17) is represented by the lines $x_2 = 2 - x_1$ and $x_2 = 2.0001 - 1.0001x_1$, which intersect at (1,1). The point (-1, 3.0001) nearly misses lying on both lines and being a solution. The differences between the lines is exaggerated in the figure—they are actually much closer.

Error magnification and condition number S5

Denote the residual by $r = b - Ax_a$. The relative backward error of system $Ax = b$ is defined to be $\frac{\|r\|_\infty}{\|b\|_\infty}$,

and the relative forward error is $\frac{\|x - x_a\|_\infty}{\|x\|_\infty}$.

The error magnification factor for $Ax = b$ is the ratio of the two, or

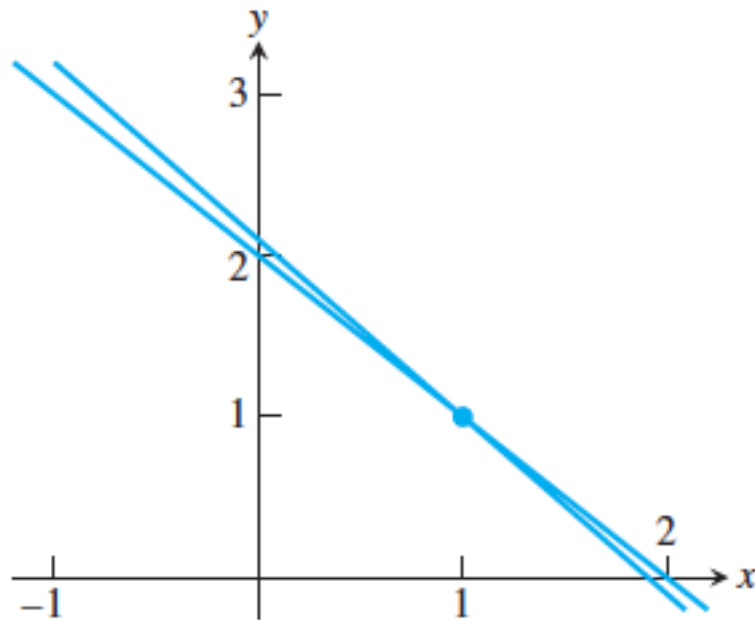
$$\text{error magnification factor} = \frac{\text{relative forward error}}{\text{relative backward error}} = \frac{\frac{\|x - x_a\|_\infty}{\|x\|_\infty}}{\frac{\|r\|_\infty}{\|b\|_\infty}}.$$

the relative backward error is $\frac{0.0001}{2.0001} \approx 0.00005 = 0.005\%$,

the relative forward error is $\frac{2.0001}{1} = 2.0001 \approx 200\%$.

The error magnification factor is $2.0001/(0.0001/2.0001) = 40004.0001$.

Error magnification and condition number S6



$$\begin{aligned}x_1 + x_2 &= 2 \\ 1.0001x_1 + x_2 &= 2.0001.\end{aligned}$$

In this case of two equations with two variables, the similar slopes of the two lines causes so large error magnification factor.

However, in the case of system consisting of more than two equations with more variables, how to predict such large error magnification factor?

Error magnification and condition number S7

- Definition

The **condition number** of a square matrix A , $\text{cond}(A)$, is the maximum possible error magnification factor for solving $Ax = b$, over all right-hand sides b . \square

Surprisingly, there is a compact formula for the condition number of a square matrix. Analogous to the norm of a vector, define the **matrix norm** of an $n \times n$ matrix A as

$$\|A\|_{\infty} = \text{maximum absolute row sum}, \quad (2.19)$$

that is, total the absolute values of each row, and assign the maximum of these n numbers to be the norm of A .

- Theorem

The condition number of the $n \times n$ matrix A is

$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\|.$$

Error magnification and condition number S8

The norm of $A = \begin{bmatrix} 1 & 1 \\ 1.0001 & 1 \end{bmatrix}$ is $\|A\| = 2.0001$, according to (2.19).

The inverse of A is $A^{-1} = \begin{bmatrix} -10000 & 10000 \\ 10001 & -10000 \end{bmatrix}$, which has norm $\|A^{-1}\| = 20001$.

The condition number of A is $\text{cond}(A) = (2.0001)(20001) = 40004.0001$.

- Applications of condition number

if $\text{cond}(A) \approx 10^k$, we should prepare to lose k digits of accuracy in computing x .

In Example 2.11, $\text{cond}(A) \approx 4 \times 10^4$, so in double precision we should expect about $16 - 4 = 12$ correct digits in the solution x .

[TestCond.m](#)

Error magnification and condition number S9

It is an example of a **vector norm** $\|x\|$, which satisfies three properties:

- (i) $\|x\| \geq 0$ with equality if and only if $x = [0, \dots, 0]$
- (ii) for each scalar α and vector x , $\|\alpha x\| = |\alpha| \cdot \|x\|$
- (iii) for vectors x, y , $\|x + y\| \leq \|x\| + \|y\|$.

In addition, $\|A\|_\infty$ is an example of a **matrix norm**, which satisfies three similar properties:

- (i) $\|A\| \geq 0$ with equality if and only if $A = 0$
- (ii) for each scalar α and matrix A , $\|\alpha A\| = |\alpha| \cdot \|A\|$
- (iii) for matrices A, B , $\|A + B\| \leq \|A\| + \|B\|$.

As a different example, the vector **1-norm** of the vector $x = [x_1, \dots, x_n]$ is $\|x\|_1 = |x_1| + \dots + |x_n|$. The matrix 1-norm of the $n \times n$ matrix A is $\|A\|_1 = \text{maximum absolute column sum}$ —that is, the maximum of the 1-norms of the column vectors.

Swamping S1

- Example

Consider the system of equations

$$\begin{aligned}10^{-20}x_1 + x_2 &= 1 \\ x_1 + 2x_2 &= 4.\end{aligned}$$

- Three methods:

1. With complete accuracy,
2. Mimicking a computer following IEEE double precision arithmetic,
3. Exchanging the order of the equations first.

Swamping S2

- Example: Exact solution

$$\left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 1 & 2 & 4 \end{array} \right] \xrightarrow{\text{subtract } 10^{20} \times \text{row 1} \text{ from row 2}} \left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 0 & 2 - 10^{20} & 4 - 10^{20} \end{array} \right].$$

$$(2 - 10^{20})x_2 = 4 - 10^{20} \rightarrow x_2 = \frac{4 - 10^{20}}{2 - 10^{20}},$$

$$10^{-20}x_1 + \frac{4 - 10^{20}}{2 - 10^{20}} = 1$$

$$x_1 = 10^{20} \left(1 - \frac{4 - 10^{20}}{2 - 10^{20}} \right)$$

$$x_1 = \frac{-2 \times 10^{20}}{2 - 10^{20}}.$$

The exact solution is

$$[x_1, x_2] = \left[\frac{2 \times 10^{20}}{10^{20} - 2}, \frac{4 - 10^{20}}{2 - 10^{20}} \right] \approx [2, 1].$$

Swamping S3

- Example: IEEE double precision

$$\left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 1 & 2 & 4 \end{array} \right] \xrightarrow{\text{subtract } 10^{20} \times \text{row 1} \\ \text{from row 2}} \left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 0 & 2 & -10^{20} \end{array} \right].$$

In IEEE double precision, $2 - 10^{20}$ is the same as -10^{20} , due to rounding. Similarly, $4 - 10^{20}$ is stored as -10^{20} . Now the bottom equation is $-10^{20}x_2 = -10^{20} \rightarrow x_2 = 1$.

The machine arithmetic version of the top equation becomes

$$10^{-20}x_1 + 1 = 1,$$

so $x_1 = 0$. The computed solution is exactly

$$[x_1, x_2] = [0, 1].$$

This solution has large relative error compared with the exact solution!

Swamping S4

- Example: IEEE double precision with row exchange

$$\left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 1 & 2 & 4 \end{array} \right] \xrightarrow{\text{row exchange}} \left[\begin{array}{cc|c} 1 & 2 & 4 \\ 10^{-20} & 1 & 1 \end{array} \right] \rightarrow \text{subtract } 10^{-20} \times \text{row 1} \text{ from row 2}$$

$$\rightarrow \left[\begin{array}{cc|c} 1 & 2 & 4 \\ 0 & \boxed{1 - 2 \times 10^{-20}} & \boxed{1 - 4 \times 10^{-20}} \end{array} \right].$$

In IEEE double precision, $1 - 2 \times 10^{-20}$ is stored as 1 and $1 - 4 \times 10^{-20}$ is stored as 1. The equations are now

$$x_1 + 2x_2 = 4$$

$$x_2 = 1, \quad \text{which yield the computed solution } x_1 = 2 \text{ and } x_2 = 1.$$

This is not the exact answer, but it is correct up to approximately 16 digits!

Swamping S5

- Example: Analysis

Overpower or Swamp

$$\left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 1 & 2 & 4 \end{array} \right] \xrightarrow[\text{from row 2}]{\text{subtract } 10^{20} \times \text{row 1}} \left[\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 0 & 2 - 10^{20} & 4 - 10^{20} \end{array} \right].$$

$$\left[\begin{array}{cc|c} 1 & 2 & 4 \\ 10^{-20} & 1 & 1 \end{array} \right] \xrightarrow[\text{from row 2}]{\text{subtract } 10^{-20} \times \text{row 1}} \left[\begin{array}{cc|c} 1 & 2 & 4 \\ 0 & 1 - 2 \times 10^{-20} & 1 - 4 \times 10^{-20} \end{array} \right].$$

The $PA = LU$ Factorization

- Partial pivoting
- Permutation matrices
- $PA=LU$ factorization

Partial pivoting S1

- **Recall:** In Gaussian elimination, the first step is to use the diagonal element a_{11} as a pivot to eliminate the first column.
- **Partial pivoting:**

Before eliminating column k , the p with $k \leq p \leq n$ and largest $|a_{pk}|$ is located, and rows p and k are exchanged if necessary before continuing with the elimination.

All multipliers will be no greater than 1 in absolute value! No more swamping problem!


Partial pivoting S2

- Example

Apply Gaussian elimination with partial pivoting to solve the system $\left[\begin{array}{cc|c} 1 & 1 & 3 \\ 3 & -4 & 2 \end{array} \right]$

Since $|a_{21}| > |a_{11}|$, we must exchange rows 1 and 2.

$$\left[\begin{array}{cc|c} 3 & -4 & 2 \\ 1 & 1 & 3 \end{array} \right] \xrightarrow[\text{from row 2}]{\text{subtract } \frac{1}{3} \times \text{row 1}} \left[\begin{array}{cc|c} 3 & -4 & 2 \\ 0 & \frac{7}{3} & \frac{7}{3} \end{array} \right].$$

After back substitution, the solution is $x_2 = 1$ and then $x_1 = 2$, as we found earlier. When we solved this system the first time, the multiplier was 3, but under partial pivoting this would never occur. 

Partial pivoting S3

- Example

Apply Gaussian elimination with partial pivoting to solve the system

$$x_1 - x_2 + 3x_3 = -3$$

$$-x_1 - 2x_3 = 1$$

$$2x_1 + 2x_2 + 4x_3 = 0.$$

$$\begin{aligned}
 & \left[\begin{array}{ccc|c} 1 & -1 & 3 & -3 \\ -1 & 0 & -2 & 1 \\ 2 & 2 & 4 & 0 \end{array} \right] \xrightarrow{\text{exchange row 1 and row 3}} \left[\begin{array}{ccc|c} 2 & 2 & 4 & 0 \\ -1 & 0 & -2 & 1 \\ 1 & -1 & 3 & -3 \end{array} \right] \\
 & \xrightarrow{\text{subtract } -\frac{1}{2} \times \text{row 1 from row 2}} \xrightarrow{\text{subtract } \frac{1}{2} \times \text{row 1 from row 3}} \left[\begin{array}{ccc|c} 2 & 2 & 4 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & -2 & 1 & -3 \end{array} \right] \\
 & \xrightarrow{\text{exchange row 2 and row 3}} \left[\begin{array}{ccc|c} 2 & 2 & 4 & 0 \\ 0 & -2 & 1 & -3 \\ 0 & 1 & 0 & 1 \end{array} \right] \xrightarrow{\text{subtract } -\frac{1}{2} \times \text{row 2 from row 3}} \left[\begin{array}{ccc|c} 2 & 2 & 4 & 0 \\ 0 & -2 & 1 & -3 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} \end{array} \right]
 \end{aligned}$$

By back-substituting, we find that $x = [1, 1, -1]$.

Permutation matrices S1

- Definition


A **permutation matrix** is an $n \times n$ matrix consisting of all zeros, except for a single 1 in every row and column. \square

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Permutation matrices S2

- Theorem

Fundamental Theorem of Permutation Matrices. Let P be the $n \times n$ permutation matrix formed by a particular set of row exchanges applied to the identity matrix. Then, for any $n \times n$ matrix A , PA is the matrix obtained by applying exactly the same set of row exchanges to A . 

For example, the permutation matrix $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

is formed by exchanging rows 2 and 3 of the identity matrix. Multiplying an arbitrary matrix on the left with P has the effect of exchanging rows 2 and 3:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} a & b & c \\ g & h & i \\ d & e & f \end{bmatrix}.$$

PA = LU factorization S1

- What is PA=LU factorization?
- It is the matrix formulation of Gaussian elimination with partial pivoting.
- The PA= LU factorization is the established workhorse for solving systems of linear equations.
- Simply the LU factorization of a row-exchanged version of A.

PA = LU factorization S2

- Example

Find the PA=LU factorization of the matrix $A = \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix}$.

$$\begin{aligned}
 & \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix} \xrightarrow{\text{exchange rows 1 and 2}} \begin{bmatrix} 4 & 4 & -4 \\ 2 & 1 & 5 \\ 1 & 3 & 1 \end{bmatrix} \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
 & \xrightarrow{\substack{\text{subtract } \frac{1}{2} \times \text{row 1} \\ \text{from row 2}}} \begin{bmatrix} 4 & 4 & -4 \\ \frac{1}{2} & -1 & 7 \\ 1 & 3 & 1 \end{bmatrix} \xrightarrow{\substack{\text{subtract } \frac{1}{4} \times \text{row 1} \\ \text{from row 3}}} \begin{bmatrix} 4 & 4 & -4 \\ \frac{1}{2} & -1 & 7 \\ \frac{1}{4} & 2 & 2 \end{bmatrix}
 \end{aligned}$$

PA = LU factorization S3

- Example (cont...)

$$\begin{aligned}
 P &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \\
 \rightarrow \text{exchange rows 2 and 3} \rightarrow & \begin{bmatrix} 4 & 4 & -4 \\ \left(\frac{1}{4}\right) & 2 & 2 \\ \left(\frac{1}{2}\right) & -1 & 7 \end{bmatrix} \\
 \rightarrow \text{subtract } -\frac{1}{2} \times \text{row 2} & \\
 \text{from row 3} \rightarrow & \begin{bmatrix} 4 & 4 & -4 \\ \left(\frac{1}{4}\right) & 2 & 2 \\ \left(\frac{1}{2}\right) & \left(-\frac{1}{2}\right) & 8 \end{bmatrix}.
 \end{aligned}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 4 & 4 & -4 \\ 0 & 2 & 2 \\ 0 & 0 & 8 \end{bmatrix}$$

$P \qquad A \qquad L \qquad U$

PA = LU factorization S4

Multiply through the equation $Ax = b$ by P on the left, and then proceed as before:

$$PAx = Pb$$

$$LUx = Pb.$$

Solve

1. $Lc = Pb$ for c .
2. $Ux = c$ for x .

PA = LU factorization S5

- Example

Use the PA=LU factorization to solve the system $Ax = b$, where

$$A = \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 0 \\ 6 \end{bmatrix}.$$

The PA=LU factorization is known from (2.22). It remains to complete the two back substitutions. 1. $Lc = Pb$:

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 6 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 5 \end{bmatrix}.$$

$c_1 = 0$

$$\frac{1}{4}(0) + c_2 = 6 \Rightarrow c_2 = 6$$

$$\frac{1}{2}(0) - \frac{1}{2}(6) + c_3 = 5 \Rightarrow c_3 = 8.$$

PA = LU factorization S6

- Example (cont...)

2. $Ux = c$:

$$\begin{bmatrix} 4 & 4 & -4 \\ 0 & 2 & 2 \\ 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 8 \end{bmatrix}$$

Starting at the bottom,

$$\begin{aligned} 8x_3 &= 8 \Rightarrow x_3 = 1 \\ 2x_2 + 2(1) &= 6 \Rightarrow x_2 = 2 \\ 4x_1 + 4(2) - 4(1) &= 0 \Rightarrow x_1 = -1. \end{aligned} \tag{2.25}$$

Therefore, the solution is $x = [-1, 2, 1]$.

[TestLu.m](#)

Writing assignment: Page 101, Exercise 1, 3

Brief Summary

- In the case of single equation, we have learned the Bisection method, the Fixed-Point-Iteration (FPI) method, the Newton's method and the Secant method.
- **Q&A:** What's major difference between **the methods of Bisection, FPI, Newton, Secant**, and **the Gaussian elimination a.w.a. its extension?** (refer to the textbook, three minutes)
- **ONLY single equation v.s. system of equations?**

Iterative Methods S1

- Two types of methods for solving systems:
 1. **Direct method** gives the exact solution within a finite number of steps (e.g. Gaussian elimination complexity $O(n^3)$), no convergence issue.
 2. **Iterative method** begins with an initial guess and refines the guess at each step, converging to the solution vector (like Fixed-Point Iteration in single equation).

Iterative Methods S2

- Jacobi Method
- Gauss–Seidel Method
- Comparing Iterative methods and Direct methods from the perspective of **Sparse matrix computations**

Jacobi Method S1

- A form of fixed-point iteration for a system of equations with the following steps:
 1. Initial step: rewrite the equations to solve for the unknown, i.e., solving the i th equation for the i th unknown.
 2. Iterative step: iterate as in Fixed-Point Iteration, starting with an initial guess.

Jacobi Method S2

- Example:

Apply the Jacobi Method to the system $3u + v = 5, u + 2v = 5$.

Begin by solving the first equation for u and the second equation for v .

We will use the initial guess $(u_0, v_0) = (0, 0)$. We have

$$u = \frac{5 - v}{3}$$
$$v = \frac{5 - u}{2}.$$

The two equations are iterated:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{2} \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-5/2}{3} \\ \frac{5-5/3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{6} \\ \frac{5}{3} \end{bmatrix} \quad \begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} \frac{5-5/3}{3} \\ \frac{5-5/6}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{25}{12} \end{bmatrix}$$

Further steps of Jacobi show convergence toward the solution, which is $[1, 2]$.

Jacobi Method S3

- Example:

Apply the Jacobi Method to the system $u + 2v = 5, 3u + v = 5$.

Solve the first equation for the first variable u and the second equation for v .

We begin with $u = 5 - 2v$

$$v = 5 - 3u.$$

The two equations are iterated as before, but the results are quite different:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 5 - 2v_0 \\ 5 - 3u_0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} 5 - 2v_1 \\ 5 - 3u_1 \end{bmatrix} = \begin{bmatrix} -5 \\ -10 \end{bmatrix} \quad \begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} 5 - 2(-10) \\ 5 - 3(-5) \end{bmatrix} = \begin{bmatrix} 25 \\ 20 \end{bmatrix}$$

In this case the Jacobi Method fails, as the iteration diverges.

Jacobi Method S4

Project one: design/implement a method to transform a matrix into strict diagonal dominance, if possible. Submit your project before May 1st.

- Definition:

The $n \times n$ matrix $A = (a_{ij})$ is strictly diagonally dominant if, for each $1 \leq i \leq n$, $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$. In other words, each main diagonal entry dominates its row in the sense that it is greater in magnitude than the sum of magnitudes of the remainder of the entries in its row. \square

- Theorem:

If the $n \times n$ matrix A is strictly diagonally dominant, then (1) A is a nonsingular matrix, and (2) for every vector b and every starting guess, the Jacobi Method applied to $Ax = b$ converges to the (unique) solution. \blacksquare

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

strictly diagonally dominant because $3 > 1$ and $2 > 1$

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

not diagonally dominant

Jacobi Method S5

- Jacobi v.s. Fixed-Point-Iteration (FPI)
 1. Both of them are in some way finding fixed-points.
 2. Different rewrite results in different convergence property (Recall the three FPI formulas and the two matrix equations)
 3. In FPI, there is only one equation, we can easily rewrite the formula as input to FPI. **However in Jacobi, is it still very convenient to rewrite the formula as input to FPI??? No!**

Jacobi Method S6

- $A = L + D + U$
 - L: the lower triangle of A (entries below the main diagonal)
 - D: the main diagonal of A
 - U: the upper triangle (entries above the main diagonal)

$$Ax = b$$

$$(D + L + U)x = b$$

$$Dx = b - (L + U)x$$

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

Jacobi Method S7

- Jacobi iteration:

$$x_0 = \text{initial vector}$$
$$x_{k+1} = D^{-1}(b - (L + U)x_k) \quad \text{for } k = 0, 1, 2, \dots$$

- For example: $\begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$

$$\begin{aligned} \begin{bmatrix} u_{k+1} \\ v_{k+1} \end{bmatrix} &= D^{-1}(b - (L + U)x_k) \\ &= \begin{bmatrix} 1/3 & 0 \\ 0 & 1/2 \end{bmatrix} \left(\begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_k \\ v_k \end{bmatrix} \right) \\ &= \begin{bmatrix} (5 - v_k)/3 \\ (5 - u_k)/2 \end{bmatrix}, \end{aligned}$$

Gauss–Seidel Method S1

$$\begin{bmatrix} u_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} (5 - v_k)/3 \\ (5 - u_k)/2 \end{bmatrix} \quad \text{vs} \quad \begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_2}{2} \end{bmatrix}$$

Jacobi:

Only uses the estimate from the preceding step

Gauss-Seidel:

The most recently updated values of the unknowns are used at each step

- Theorem

If the $n \times n$ matrix A is strictly diagonally dominant, then (1) A is a nonsingular matrix, and (2) for every vector b and every starting guess, the Gauss–Seidel Method applied to $Ax = b$ converges to a solution. ■

Gauss–Seidel Method S2

Gauss–Seidel can be written in matrix form and identified as a fixed-point iteration where we isolate the equation $(L + D + U)x = b$ as

$$(L + D)x_{k+1} = -Ux_k + b.$$

Note that the usage of newly determined entries of x_{k+1} is accommodated by including the lower triangle of A into the left-hand side. Rearranging the equation gives the Gauss–Seidel Method.

Gauss–Seidel Method

$$x_0 = \text{initial vector}$$

$$x_{k+1} = D^{-1}(b - Ux_k - Lx_{k+1}) \quad \text{for } k = 0, 1, 2, \dots$$

How to implement Gauss-Seidel?

Coding assignment (Computer Problems 3 at page 116)

Coding assignment (Computer Problems 1 at page 116)

Gauss–Seidel Method S3

- Example

Apply the Gauss–Seidel Method to the system

$$\begin{bmatrix} 3 & 1 & -1 \\ 2 & 4 & 1 \\ -1 & 2 & 5 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}$$

The Gauss–Seidel iteration is

$$u_{k+1} = \frac{4 - v_k + w_k}{3}$$

$$v_{k+1} = \frac{1 - 2u_{k+1} - w_k}{4}$$

$$w_{k+1} = \frac{1 + u_{k+1} - 2v_{k+1}}{5}$$

Starting with some initial guess, we can find the solution within some iterations.

Iterative methods vs Direct methods S1

- Two advantages of Iterative methods over Direct methods:
 1. When a good approximation to the solution is already known;
 2. Solve sparse systems of equations

In iterative methods, the number of operations in each iteration is only $O(n^2)$;

In direct methods, the Gaussian-elimination takes $O(n^3)$ operations.
If the number of iterations is far less than n , then iterative methods WINS!

Iterative methods vs Direct methods S2

- Scenario one: When a good approximation to the solution is already known
 1. Suppose that a solution to $Ax = b$ is known.
 2. A and/or b change by a small amount, e.g. in dynamic system.
 3. Using the solution to the previous problem as initial guess for Jacobi or Gauss-Seidel will converge very fast!

This technique is called **polishing**!

Notice that direct methods cannot reuse the solution to the previous problem but have to restart from scratch!

Iterative methods vs Direct methods S3

- Scenario one: to solve sparse systems of equations.
- Definition: A coefficient matrix is called **sparse** if many of the matrix entries are known to be zero, e.g., only $O(n)$ non zeros in a $n \times n$ matrix.
- Gaussian elimination applied to a sparse matrix usually causes **fill-in**, where the coefficient matrix changes from sparse to full due to the necessary row operations.

Iterative methods vs Direct methods S4

- Example: Compare Jacobi with Gaussian-elimination to solve the system with the following coefficient matrix

$$A = \begin{bmatrix} 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 \end{bmatrix}$$

$$U = \begin{bmatrix} 3 & -1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.500 \\ 0 & 2.7 & -1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.500 & 0.165 \\ 0 & 0 & 2.6 & -1.0 & 0 & 0 & 0 & 0 & 0 & 0.500 & 0.187 & 0.062 \\ 0 & 0 & 0 & 2.6 & -1.000 & 0 & 0 & 0 & 0.500 & 0.191 & 0.071 & 0.024 \\ 0 & 0 & 0 & 0 & 2.618 & -1.000 & 0 & 0.500 & 0.191 & 0.073 & 0.027 & 0.009 \\ 0 & 0 & 0 & 0 & 0 & 2.618 & -1.000 & 0.191 & 0.073 & 0.028 & 0.010 & 0.004 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2.618 & -0.927 & 0.028 & 0.011 & 0.004 & 0.001 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.562 & -1.032 & -0.012 & -0.005 & -0.001 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.473 & -1.047 & -0.018 & -0.006 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.445 & -1.049 & -0.016 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.440 & -1.044 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.458 \end{bmatrix}$$

What about in the case when $n=10^5$?

The Gaussian-elimination methods completely fail due to: 1) huge memory consumption (80 GB); 2) long computational time (over one year);

The Jacobi method only about one second within 8 MB memory consumption.

Methods for symmetric positive-definite matrices

- Symmetric positive-definite matrices
- Cholesky factorization
- Conjugate Gradient Method

Symmetric positive-definite matrices

S1

- Definition


The $n \times n$ matrix A is **symmetric** if $A^T = A$. The matrix A is **positive-definite** if $x^T A x > 0$ for all vectors $x \neq 0$. \square

- Example

Show that the matrix $A = \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix}$ is symmetric positive-definite.

Clearly A is symmetric. To show it is positive-definite, one applies the definition:

$$\begin{aligned} x^T A x &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= 2x_1^2 + 4x_1x_2 + 5x_2^2 \\ &= 2(x_1 + x_2)^2 + 3x_2^2 \end{aligned}$$

This expression is always non-negative, and cannot be zero unless both $x_2 = 0$ and $x_1 + x_2 = 0$, which together imply $x = 0$. 

Symmetric positive-definite matrices

S2

- Property 1

If the $n \times n$ matrix A is symmetric, then A is positive-definite if and only if all of its eigenvalues are positive.

- Property 2

If A is $n \times n$ symmetric positive-definite and X is an $n \times m$ matrix of full rank with $n \geq m$, then $X^T A X$ is $m \times m$ symmetric positive-definite.

- Definition

A principal submatrix of a square matrix A is a square submatrix whose diagonal entries are diagonal entries of A . □

- Property 3

Any principal submatrix of a symmetric positive-definite matrix is symmetric positive-definite.

Symmetric positive-definite matrices

S3

- Example

For example, if

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

is symmetric positive-definite, then so is

$$\begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix}.$$

Cholesky factorization S1

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

$$a > 0$$

the determinant $ac - b^2$ of A is positive

Writing $A = R^T R$ with an upper triangular R implies the form

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \sqrt{a} & 0 \\ u & v \end{bmatrix} \begin{bmatrix} \sqrt{a} & u \\ 0 & v \end{bmatrix} = \begin{bmatrix} a & u\sqrt{a} \\ u\sqrt{a} & u^2 + v^2 \end{bmatrix}$$

$$u = b/\sqrt{a} \text{ and } v^2 = c - u^2 \quad \longrightarrow \quad v^2 = c - (b/\sqrt{a})^2 = c - b^2/a > 0$$

$$\longrightarrow A = \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \sqrt{a} & 0 \\ \frac{b}{\sqrt{a}} & \sqrt{c - b^2/a} \end{bmatrix} \begin{bmatrix} \sqrt{a} & \frac{b}{\sqrt{a}} \\ 0 & \sqrt{c - b^2/a} \end{bmatrix} = R^T R$$

Cholesky factorization S2

- Theorem

(Cholesky Factorization Theorem) If A is a symmetric positive-definite $n \times n$ matrix, then there exists an upper triangular $n \times n$ matrix R such that $A = R^T R$. ■

Cholesky factorization

```
for  $k = 1, 2, \dots, n$ 
  if  $A_{kk} < 0$  stop, end
   $R_{kk} = \sqrt{A_{kk}}$ 
   $u^T = \frac{1}{R_{kk}} A_{k,k+1:n}$ 
   $R_{k,k+1:n} = u^T$ 
   $A_{k+1:n,k+1:n} = A_{k+1:n,k+1:n} - uu^T$ 
end
```

Conjugate Gradient Method S1

The ideas behind conjugate gradients rely on the generalization of the usual idea of inner product. The Euclidean inner product $(v, w) = v^T w$ is symmetric and linear in the inputs v and w , since $(v, w) = (w, v)$ and $(\alpha v + \beta w, u) = \alpha(v, u) + \beta(w, u)$ for scalars α and β . The Euclidean inner product is also positive-definite, in that $(v, v) > 0$ if $v \neq 0$.

- Definition

Let A be a symmetric positive-definite $n \times n$ matrix. For two n -vectors v and w , define the **A -inner product**

$$(v, w)_A = v^T A w.$$

The vectors v and w are **A -conjugate** if $(v, w)_A = 0$. □

Note that the new inner product inherits the properties of symmetry, linearity, and positive-definiteness from the matrix A .

Conjugate Gradient Method S2

Conjugate Gradient Method

$x_0 =$ initial guess
 $d_0 = r_0 = b - Ax_0$
for $k = 0, 1, 2, \dots, n - 1$
 if $r_k = 0$, **stop**, **end**
 $\alpha_k = \frac{r_k^T r_k}{d_k^T A d_k}$
 $x_{k+1} = x_k + \alpha_k d_k$
 $r_{k+1} = r_k - \alpha_k A d_k$
 $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
 $d_{k+1} = r_{k+1} + \beta_k d_k$
end

α_k is chosen precisely so that the new residual r_{k+1} is orthogonal to the direction d_k

β_k is chosen to ensure the pairwise A -conjugacy of the d_k

The conjugate gradient iteration updates three different vectors on each step.

The vector x_k is the approximate solution at step k .

The vector r_k represents the residual of the approximate solution x_k .

Finally, the vector d_k represents the new search direction used to update the approximation x_k to the improved version x_{k+1} .

Conjugate Gradient Method S3

- Example Solve $\begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \end{bmatrix}$ using the Conjugate Gradient Method.

Following the above algorithm we have $x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, r_0 = d_0 = \begin{bmatrix} 6 \\ 3 \end{bmatrix}$

$$\alpha_0 = \frac{\begin{bmatrix} 6 \\ 3 \end{bmatrix}^T \begin{bmatrix} 6 \\ 3 \end{bmatrix}}{\begin{bmatrix} 6 \\ 3 \end{bmatrix}^T \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 6 \\ 3 \end{bmatrix}} = \frac{45}{6 \cdot 18 + 3 \cdot 27} = \frac{5}{21}$$

$$x_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \frac{5}{21} \begin{bmatrix} 6 \\ 3 \end{bmatrix} = \begin{bmatrix} 10/7 \\ 5/7 \end{bmatrix}$$

$$r_1 = \begin{bmatrix} 6 \\ 3 \end{bmatrix} - \frac{5}{21} \begin{bmatrix} 18 \\ 27 \end{bmatrix} = 12 \begin{bmatrix} 1/7 \\ -2/7 \end{bmatrix}$$

$$\beta_0 = \frac{r_1^T r_1}{r_0^T r_0} = \frac{144 \cdot 5/49}{36 + 9} = \frac{16}{49}$$

$$d_1 = 12 \begin{bmatrix} 1/7 \\ -2/7 \end{bmatrix} + \frac{16}{49} \begin{bmatrix} 6 \\ 3 \end{bmatrix} = \begin{bmatrix} 180/49 \\ -120/49 \end{bmatrix}$$

$$\alpha_1 = \frac{\begin{bmatrix} 12/7 \\ -24/7 \end{bmatrix}^T \begin{bmatrix} 12/7 \\ -24/7 \end{bmatrix}}{\begin{bmatrix} 180/49 \\ -120/49 \end{bmatrix}^T \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 180/49 \\ -120/49 \end{bmatrix}} = \frac{7}{10}$$

$$x_2 = \begin{bmatrix} 10/7 \\ 5/7 \end{bmatrix} + \frac{7}{10} \begin{bmatrix} 180/49 \\ -120/49 \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \end{bmatrix}$$

$$r_2 = 12 \begin{bmatrix} 1/7 \\ -2/7 \end{bmatrix} - \frac{7}{10} \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 180/49 \\ -120/49 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Since $r_2 = b - Ax_2 = 0$, the solution is $x_2 = [4, -1]$.

Conjugate Gradient Method S4


- Theorem

Let A be a symmetric positive-definite $n \times n$ matrix and let $b \neq 0$ be a vector. In the Conjugate Gradient Method, assume that $r_k \neq 0$ for $k < n$ (if $r_k = 0$ the equation is solved). Then for each $1 \leq k \leq n$,

(a) The following three subspaces of R^n are equal:


$$\langle x_1, \dots, x_k \rangle = \langle r_0, \dots, r_{k-1} \rangle = \langle d_0, \dots, d_{k-1} \rangle,$$

(b) the residuals r_k are pairwise orthogonal: $r_k^T r_j = 0$ for $j < k$,

(c) the directions d_k are pairwise A -conjugate: $d_k^T A d_j = 0$ for $j < k$. 

- Example

Apply the Conjugate Gradient Method to system (2.45) with $n = 100,000$.

After 20 steps of the Conjugate Gradient Method, the difference between the computed solution x and the true solution $(1, \dots, 1)$ is less than 10^{-9} in the vector infinity norm. The total time of execution was less than one second on a PC. 

Coding assignment (Computer Problems 1, 5 at page 130)

Nonlinear Systems of Equations

- Multivariate Newton's Method
- Broyden's Method

Multivariate Newton's Method S1

- Recall:

The one-variable Newton's Method

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

- Multivariate case:

For example, let

$$\begin{aligned} f_1(u, v, w) &= 0 \\ f_2(u, v, w) &= 0 \\ f_3(u, v, w) &= 0 \end{aligned} \tag{2.49}$$

be three nonlinear equations in three unknowns u, v, w . Define the vector-valued function $F(u, v, w) = (f_1, f_2, f_3)$, and denote the problem (2.49) by $F(x) = 0$, where $x = (u, v, w)$.

Multivariate Newton's Method S2

The analogue of the derivative f' in the one-variable case is the **Jacobian matrix** defined by

$$DF(x) = \begin{bmatrix} \frac{\partial f_1}{\partial u} & \frac{\partial f_1}{\partial v} & \frac{\partial f_1}{\partial w} \\ \frac{\partial f_2}{\partial u} & \frac{\partial f_2}{\partial v} & \frac{\partial f_2}{\partial w} \\ \frac{\partial f_3}{\partial u} & \frac{\partial f_3}{\partial v} & \frac{\partial f_3}{\partial w} \end{bmatrix}.$$

The Taylor expansion for vector-valued functions around x_0 is

$$F(x) = F(x_0) + DF(x_0) \cdot (x - x_0) + O(x - x_0)^2.$$

Multivariate Newton's Method S3

For example, the linear expansion of $F(u, v) = (e^{u+v}, \sin u)$ around $x_0 = (0, 0)$ is

$$\begin{aligned} F(x) &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} e^0 & e^0 \\ \cos 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + O(x^2) \\ &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} u+v \\ u \end{bmatrix} + O(x^2). \end{aligned}$$

Newton's Method is based on a linear approximation, ignoring the $O(x^2)$ terms. As in the one-dimensional case, let $x = r$ be the root, and let x_0 be the current guess. Then

$$0 = F(r) \approx F(x_0) + DF(x_0) \cdot (r - x_0),$$

or

$$-DF(x_0)^{-1} F(x_0) \approx r - x_0. \quad (2.50)$$

Therefore, a better approximation for the root is derived by solving (2.50) for r .

Multivariate Newton's Method S4

Multivariate Newton's Method

$x_0 = \text{initial vector}$

$$x_{k+1} = x_k - (DF(x_k))^{-1} F(x_k) \quad \text{for } k = 0, 1, 2, \dots$$

- Need to find the inverse of Jacobian matrix.

$$s = (DF(x_k))^{-1} F(x_k)$$

$$DF(x_k)s = F(x_k)$$

- After getting s , we have the following iterative

$x_0 = \text{initial vector}$

$$\begin{cases} DF(x_k)s = -F(x_k) \\ x_{k+1} = x_k + s. \end{cases} \quad (2.51)$$

Multivariate Newton's Method S5

- Example

Use Newton's Method with starting guess $(1, 2)$ to find a solution of the system

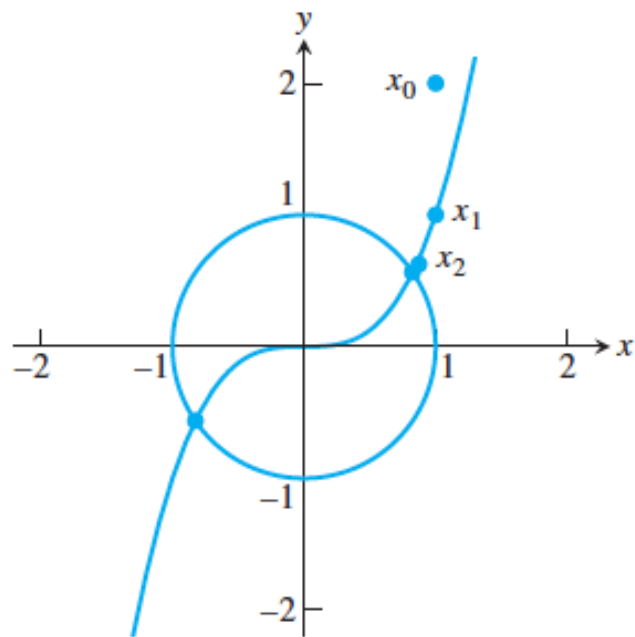
$$\begin{aligned} v - u^3 &= 0 \\ u^2 + v^2 - 1 &= 0. \end{aligned} \qquad DF(u, v) = \begin{bmatrix} -3u^2 & 1 \\ 2u & 2v \end{bmatrix}$$

$$\begin{aligned} x_0 &= (1, 2) \\ DF(x_k)s &= -F(x_k) \end{aligned} \Rightarrow \begin{bmatrix} -3 & 1 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = -\begin{bmatrix} 1 \\ 4 \end{bmatrix} \Rightarrow s = (0, -1)$$

$$\begin{aligned} x_1 &= x_0 + s = (1, 1) \\ DF(x_k)s &= -F(x_k) \end{aligned} \Rightarrow \begin{bmatrix} -3 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = -\begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow s = (-1/8, -3/8)$$

$$x_2 = x_1 + s = (7/8, 5/8) \quad \dots\dots$$

Multivariate Newton's Method S6



step	u	v
0	1.0000000000000000	2.0000000000000000
1	1.0000000000000000	1.0000000000000000
2	0.8750000000000000	0.6250000000000000
3	0.82903634826712	0.56434911242604
4	0.82604010817065	0.56361977350284
5	0.82603135773241	0.56362416213163
6	0.82603135765419	0.56362416216126
7	0.82603135765419	0.56362416216126

Figure 2.5 Newton's Method for Example 2.32. The two roots are the dots on the circle. Newton's Method produces the dots that are converging to the solution at approximately $(0.8260, 0.5636)$.

Multivariate Newton's Method S7

- Example

Use Newton's Method to find the solutions of the system

$$\begin{aligned}f_1(u, v) &= 6u^3 + uv - 3v^3 - 4 = 0 \\f_2(u, v) &= u^2 - 18uv^2 + 16v^3 + 1 = 0.\end{aligned}$$


Notice that $(u, v) = (1, 1)$ is one solution. It turns out that there are two others. The Jacobian matrix is

$$DF(u, v) = \begin{bmatrix} 18u^2 + v & u - 9v^2 \\ 2u - 18v^2 & -36uv + 48v^2 \end{bmatrix}.$$

Which solution is found by Newton's Method depends on the starting guess, just as in the one-dimensional case. Using starting point $(u_0, v_0) = (2, 2)$, iterating the preceding formula yields the following table:

Multivariate Newton's Method S8

step	u	v
0	2.000000000000000	2.000000000000000
1	1.37258064516129	1.34032258064516
2	1.07838681200443	1.05380123264984
3	1.00534968896520	1.00269261871539
4	1.00003367866506	1.00002243772010
5	1.00000000111957	1.00000000057894
6	1.000000000000000	1.000000000000000
7	1.000000000000000	1.000000000000000

Other initial vectors lead to the other two roots, which are approximately $(0.865939, 0.462168)$ and $(0.886809, -0.294007)$. See Computer Problem 2. 

Coding assignment: Page 136, Computer Problem 2.

Broyden's Method I S1

- Single-variable equation case:

Newton's method -> Secant method

$$f'(r) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

- Multivariate system case:

Newton's method -> Broyden's method

Replacing Jacobian matrix with an approximation matrix A that can be easily updated.

Broyden's Method I S2

Suppose A_i is the best approximation available at step i to the Jacobian matrix, and that it has been used to create

$$x_{i+1} = x_i - A_i^{-1} F(x_i). \quad (2.52)$$

To update A_i to A_{i+1} for the next step, we would like to respect the derivative aspect of the Jacobian DF , and satisfy

$$A_{i+1} \delta_{i+1} = \Delta_{i+1}, \quad (2.53)$$

where $\delta_{i+1} = x_{i+1} - x_i$ and $\Delta_{i+1} = F(x_{i+1}) - F(x_i)$. On the other hand, for the orthogonal complement of δ_{i+1} , we have no new information. Therefore, we ask that

$$A_{i+1} w = A_i w \quad (2.54)$$

for every w satisfying $\delta_{i+1}^T w = 0$. One checks that a matrix that satisfies both (2.53) and (2.54) is

$$A_{i+1} = A_i + \frac{(\Delta_{i+1} - A_i \delta_{i+1}) \delta_{i+1}^T}{\delta_{i+1}^T \delta_{i+1}}$$

Attention: There is one typo in eq. (2.55) of the text book.

Two questions (10 mins): What does the (2.54) mean? [Broyden's paper](#)

How to check the matrix A_{i+1} satisfying both (2.53) and (2.54)?

Broyden's Method I S3

Broyden's Method I

x_0 = initial vector

A_0 = initial matrix

for $i = 0, 1, 2, \dots$

$$x_{i+1} = x_i - A_i^{-1} F(x_i)$$

$$A_{i+1} = A_i + \frac{(\Delta_{i+1} - A_i \delta_{i+1}) \delta_{i+1}^T}{\delta_{i+1}^T \delta_{i+1}}$$

end

where $\delta_{i+1} = x_{i+1} - x_i$ and $\Delta_{i+1} = F(x_{i+1}) - F(x_i)$.

Note that the Newton-type step is carried out by solving $A_i \delta_{i+1} = F(x_i)$, just as for Newton's Method. Also like Newton's Method, Broyden's Method is not guaranteed to converge to a solution.

Can we avoid the computation of the inverse of the matrix A_i ?

Broyden's Method II S1

We redo the derivation of Broyden from the point of view of $B_i = A_i^{-1}$. We would like to have

$$\delta_{i+1} = B_{i+1} \Delta_{i+1}, \quad (2.56)$$

where $\delta_{i+1} = x_{i+1} - x_i$ and $\Delta_{i+1} = F(x_{i+1}) - F(x_i)$, and for every w satisfying $\delta_{i+1}^T w = 0$, still satisfy $A_{i+1} w = A_i w$, or

$$B_{i+1} A_i w = w. \quad (2.57)$$

A matrix that satisfies both (2.56) and (2.57) is How to check that? 10 mins

$$B_{i+1} = B_i + \frac{(\delta_{i+1} - B_i \Delta_{i+1}) \delta_{i+1}^T B_i}{\delta_{i+1}^T B_i \Delta_{i+1}}. \quad (2.58)$$

The new version of the iteration, which needs no matrix solve, is

$$x_{i+1} = x_i - B_i F(x_i). \quad (2.59)$$

Broyden's Method II S2

The resulting algorithm is called Broyden's Method II.

Broyden's Method II

x_0 = initial vector

B_0 = initial matrix

for $i = 0, 1, 2, \dots$

$$x_{i+1} = x_i - B_i F(x_i)$$

$$B_{i+1} = B_i + \frac{(\delta_{i+1} - B_i \Delta_{i+1}) \delta_{i+1}^T B_i}{\delta_{i+1}^T B_i \Delta_{i+1}}$$

end

where $\delta_i = x_i - x_{i-1}$ and $\Delta_i = F(x_i) - F(x_{i-1})$.

Both versions of Broyden's Method converge superlinearly (to simple roots), slightly slower than the quadratic convergence of Newton's Method.

Writing assignment: Page 1, Exercise 1.

Coding assignment: Page 137, Computer Problem 7.