2.1 Direct Solvers

Linear systems of equations $Ax = b$

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- known $A \in \mathbb{R}^{n,n}$ and $b \in \mathbb{R}^n$
- A *invertible*, i.e., inverse $A^{-1} \in \mathbb{R}^{n,n}$ exists
- unknown $x \in \mathbb{R}^n$

$$
\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\
a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\
&\vdots \\
a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n
\end{aligned}
$$

Questions

- Find examples of invertible matrices
- Find examples of not-invertible (singular) matrices
- Discuss invertibility and solution of $Ax = 0$
- What is geometric interpretation of determinant $\det(A)$ and application to systems of equations
- Discuss systems $Ax = b$ with non-square matrix A

Think about the questions. Write down answers. Discuss with colleagues. Assess your answers. Find your own questions.

Applications

- Linear systems appear in many applications:
  - Regression by least squares method in statistics
  - Linear programming in optimization
  - Numerical solutions of ordinary differential equations
  - Numerical solutions of partial differential equations
  - Solving nonlinear equations by linearization
  - ...
- The size of $Ax = b$ is usually huge in applications such that it is not possible to solve "by hand"
- Need to develop algorithms to let computer do the job.

Algorithms

- The algorithms for solving linear systems fall into two categories:
  - Direct methods
  - Iterative methods
- Direct methods produce (exact) solution using a finite number of arithmetic operations
- Most common method: Gaussian elimination
- Basic idea: Reduce system $Ax = b$ to equivalent $Ux = y$ where U is upper triangular
- We will see: Gaussian elimination leads to matrix factorisation $A = LU$ where L is lower triangular

Question: Write down examples of upper and lower triangular matrices.

- If we know $A = LU$, we can solve two systems to get solution of $Ax = b$

$$Ly = b, \quad \text{and} \quad Ux = y$$

- Gaussian elimination can break down and LU factorisation may not exist, use factorisation

$$A = PLU$$

  with permutation matrix P
- Alternative: QR factorisation $A = QR$, with upper triangular R and orthogonal Q

Questions:

- Show that $Ax = b$ if $Ly = b$ and $Ux = y$ and $A = LU$
- How would you solve $Qy = b$?

```
# Solving linear systems in Python

A = np.array([[2,3],[5,9]])
print("A = ", A, "\n")

b = np.array([12,33])
print("b = ", b, "\n")

x = la.solve(A,b)
print("x = ", x, "\n")

np.allclose(np.dot(A,x), b)

A =  [[2 3]
 [5 9]]

b =  [12 33]

x =  [ 3.  2.]
```

Question:

Check the documentation of numpy.linalg.solve. Can you find out what method is used? The following code determines the LU factorisation with partial pivoting. Try this out. Can you find examples where no LU factorisation exists but where the code below till gives a solution?

```
# Solving linear systems of equations in Python
# good for solving system with same A and different b

import numpy as np
import scipy.linalg as la

A = np.array([[2,3],[5,9]])
b = np.array([12,33])

lu, p = la.lu_factor(A)
x = la.lu_solve((lu,p), b)

np.allclose(np.dot(A,x), b)


True
```

Elimination by Elementary Operations

- We will use combinations of three types of elementary row operations:
  - adding a multiple of one row to another row
  - swapping two rows
  - multiplying a row by a non-zero number

Convert $a = [a_1, a_2, a_3, a_4]^T$ with $a_2 \neq 0$ into $[a_1, a_2, 0, 0]^T$ by

- multiplying the second row by $-\frac{a_3}{a_2}$ and adding to the third row;
- multiplying the second row by $-\frac{a_4}{a_2}$ and adding to the fourth row.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -\frac{a_3}{a_2} & 1 & 0 \\ 0 & -\frac{a_4}{a_2} & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ 0 \\ 0 \end{bmatrix}$$

Question: What are the matrices of the other two elementary operations (row swapping and scaling)?

Elementary Matrices (Multipliers)

- vector $a = [a_1, a_2, \cdots, a_n]^T$ with $a_k \neq 0$
- action of elementary matrix

$$E_k a = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -m_{k+1} & 1 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -m_n & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- E_k is designed to nullify all of the elements below a_k in a, and $m_j = \frac{a_j}{a_k}$

Properties of Elementary Matrices

Let e_k denote the column vector with 1 on spot k and 0 elsewhere.

1. E_k is lower triangular with unit main diagonal.
2. $E_k = I - m_k e_k^T$, where $m_k = [0 \ \cdots \ 0 \ m_{k+1} \ \cdots \ m_n]^T$.
3. $E_k^{-1} = I + m_k e_k^T$ (E_k^{-1} will be denoted by L_k).
4. If $k < j$ then $E_k E_j = I - m_k e_k^T - m_j e_j^T$.
5. $E_1 E_2 \cdots E_{n-1} = I - \sum_{k=1}^{n-1} m_k e_k^T$ – lower triangular matrix.

The first two items are obvious. For the third item, use $e_k^T m_k = 0$ and

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

The last two items can be checked by using $e_k^T m_j = 0$ for $k < j$.

Elementary Matrices – examples

Given $a = [2 \quad 4 \quad -2]^T$, we have

$$E_1 a = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix},$$

$$E_2 a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix},$$

$$L_1 = E_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix},$$

$$L_2 = E_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{bmatrix}.$$

LU Factorisation using Elementary Matrices

Algorithm for matrix $A = (a_{i,j})$.

1. Gaussian elimination first column of A to make elements below $a_{1,1}$ zero
2. Gaussian elimination on second column to make elements below $a_{2,2}$ zero
3. continue this procedure until last column

This leads to $E_{n-1} \cdots E_2 E_1 A = U$ and

$$\begin{aligned} A = & \quad E_1^{-1} E_2^{-1} \cdots E_{n-1}^{-1} U \\ = & \quad\quad L_1 L_2 \cdots L_{n-1} U \\ = & \quad\quad\quad\quad LU. \end{aligned}$$

L can be obtained easily once $E_1, \cdots, E_{n-1}$ are available. Indeed

$$E_k = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -l_{k+1,k} & 1 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -l_{n,k} & 0 & \cdots & 1 \end{bmatrix}$$

$$\Longrightarrow L_k = E_k^{-1} = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & l_{k+1,k} & 1 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & l_{n,k} & 0 & \cdots & 1 \end{bmatrix}$$

$$\Longrightarrow L = L_1 L_2 \cdots L_{n-1} = \begin{bmatrix} 1 & & & & \\ l_{2,1} & 1 & & & \\ \vdots & l_{3,2} & \ddots & & \\ \vdots & \vdots & \ddots & 1 & \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n-1} & 1 \end{bmatrix}$$

Hence L is a lower triangular matrix with unit main diagonal.

LU factorisation – example

Consider the matrix

$$A = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix}.$$

We have

$$E_1 A = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix}.$$

$$E_2 E_1 A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} = U.$$

Therefore, we obtain the LU Factorisation $A = LU$, where

$$L = L_1 L_2 = E_1^{-1} E_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix}.$$

Algorithm (LU factorisation – pseudo code version)

```
L = I
for k=1:n-1
  for i=k+1:n
    L(i,k) = A(i,k)/A(k,k)
    A(i,k) = 0.0
    for j=k+1:n
       A(i,j)=A(i,j)-L(i,k)*A(k,j)

U = A
```

```
## Algorithm (LU factorisation -- Python version)
## deals with exact breakdown and non-square A

def LU(A):
    (n,m) = A.shape
    s = min(n,m)
    L = np.eye(n)
    U = A.copy()

    for k in range(s-1):
        if (U[k,k]!=0):
            L[k+1:,k] = U[k+1:,k]/U[k,k]   # multipliers
        elif (np.sum(abs(U[k+1:,k])) != 0): # zero pivot
            raise RuntimeError('LU breakdown')
        U[k+1:,k+1:] -= np.outer(L[k+1:,k],U[k,k+1:])
        U[k+1:,k] = 0

    return L, U
```

```
A = np.array(((3.0, 4.0, 6), (3.0, 5.0, 2), (3,1,4)))
#A = np.array(((0.0, 1.0), (1.0, 0.0)))  # implement partia
#A = np.array(((0.0, 0.0), (0.0, 0.0)))  # no elimination t
#A = np.ones((3,2))

print(A)
L, U = LU(A.copy())
print(L)
print(U)
```

```
[[ 3.  4.  6.]
 [ 3.  5.  2.]
 [ 3.  1.  4.]]
[[ 1.  0.  0.]
 [ 1.  1.  0.]
 [ 1. -3.  1.]]
[[  3.   4.   6.]
 [  0.   1.  -4.]
 [  0.   0. -14.]]
```

Existence and Uniqueness

- $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ does not have an LU factorisation
- Let A^r be $r \times r$ submatrix with first r rows and columns of A.
- The r-th **principle minor** of A is the determinant $\det(A^r)$.
- **existence of LU factorisation**: if first $n-1$ principle subminors of A do not vanish then the LU factorisation $A = LU$ exists and is unique
- Every symmetric positive definite matrix has LU factorisation.
- Proof: By induction or by constructing the elimination process and observing that the process may be continued as long as the pivots are non-zero. The pivots will be non-zero since the product of the first j pivots is equal to $\det(A^j) \neq 0$.

QR decomposition

- The QR decomposition also uses an elimination process and (in this case orthogonal) elmentary matrices. The most commonly used matrices are reflections or Householder matrices of the form $H = I - 2uu^T$ where u have length one.
- Rotation (Givens or Jacobi) matrices are also used and even the Gram-Schmidt process can be implemented numerically.

Solution using the LU factorisation

- Let $A = LU$ with

$$L = \begin{bmatrix} l_{1,1} & & & \\ l_{2,1} & l_{2,2} & & \\ \vdots & \vdots & \ddots & \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{bmatrix}, \quad U = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ & u_{2,2} & \cdots & u_{2,n} \\ & & \ddots & \vdots \\ & & & u_{n,n} \end{bmatrix}.$$

- Solving $Ax = b$ in 2 steps:
  1. Solve the lower triangular system $Ly = b$ for y by forward substitution.
  2. Solve the upper triangular system $Ux = y$ for x by back substitution.

Forward Substitution

The lower triangular system $Ly = b$ takes the form

$$\begin{array}{ccccccc} l_{1,1}y_1 & & & & & = & b_1 \\ l_{2,1}y_1 & + & l_{2,2}y_2 & & & = & b_2 \\ \vdots & & \vdots & \ddots & & = & \vdots \\ l_{n,1}y_1 & + & l_{n,2}y_2 & + \ldots + & l_{n,n}y_n & = & b_n \end{array}$$

By the first equation we can obtain

$$y_1 = \frac{b_1}{l_{1,1}}.$$

Inductively, the solution is given by

$$y_i = \frac{b_i - \sum_{k=1}^{i-1} l_{i,k}y_k}{l_{i,i}}, \quad i = 1, 2, \ldots, n.$$

Backward Substitution

The upper triangular system $Ux = y$ takes the form

$$\begin{array}{ccccccc} u_{1,1}x_1 & + & u_{1,2}x_2 & + \ldots + & u_{1,n}x_n & = & y_1 \\ & & u_{2,2}x_2 & + \ldots + & u_{2,n}x_n & = & y_2 \\ & & & \ddots & \vdots & \vdots & \vdots \\ & & & & u_{n,n}x_n & = & y_n \end{array}$$

By the last equation we obtain

$$x_n = \frac{y_n}{u_{n,n}}.$$

Inductively, the solution x is given by

$$x_i = \frac{y_i - \sum_{k=i+1}^{n} u_{i,k}x_k}{u_{i,i}}, \quad i = n, n-1, ..., 1.$$

Solving $Ly = b$ while computing $LU = A$

- forward substitution can be done simultaneously as LU factorisation:
  - factorise the augmented matrix $[A \;\; b]$

$$\begin{aligned} E_{n-1} \dots E_1 [A \;\; b] &= [E_{n-1} \dots E_1 A \;\; E_{n-1} \dots E_1 b] \\ &= [U \;\; L^{-1} b] \\ &= [U \;\; y]. \end{aligned}$$

Example

Consider the linear system $Ax = b$ where

$$A = \begin{bmatrix} 1 & 3 & 1 \\ 1 & -2 & -1 \\ 2 & 1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 10 \\ -6 \\ 10 \end{bmatrix}.$$

We work on the augmented matrix $[A \;\; b]$. Then

$$E_1[A \;\; b] = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 & 1 & 10 \\ 1 & -2 & -1 & -6 \\ 2 & 1 & 2 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 1 & 10 \\ 0 & -5 & -2 & -16 \\ 0 & -5 & 0 & -10 \end{bmatrix},$$

$$E_2E_1[A \quad b] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 & 1 & 10 \\ 0 & -5 & -2 & -16 \\ 0 & -5 & 0 & -10 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 3 & 1 & 10 \\ 0 & -5 & -2 & -16 \\ 0 & 0 & 2 & 6 \end{bmatrix}.$$

By back substitution we obtain the solution

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

Algorithm (Solve linear system by LU factorisation)

```
M = [A  b]
for k=1:n-1
  for i=k+1:n
     q = M(i,k)/M(k,k)
     for j = k:n+1
        M(i,j) = M(i, j) - q*M(k,j)
     end
  end
end
x(n) = M(n,n+1)/M(n,n)
for i = n-1:-1:1
   z = 0
   for j = i+1:n
      z = z + M(i,j)*x(j)
   end
   x(i) = (M(i,n+1)-z)/M(i,i)
end
```

Flops - Gaussian Elimination

The most expensive part of the algorithm on LU factorisation involves the row operations which can be written as three nested for loops

```
L = I
for k=1:n-1
  for i=k+1:n
    L(i,k) = A(i,k)/A(k,k)
    A(i,k) = 0.0
    for j=k+1:n
       A(i,j)=A(i,j)-L(i,k)*A(k,j)

U = A
```