Python Basics, Programming Fundamentals, and Introduction to Numerical Linear Algebra

Rhudaina Mohammad

Institute of Mathematics, UP Diliman

rzmohammad@up.edu.ph

DMCS Graduate Bootcamp 2024
5 - 8 August 2024 · UP Baguio

Linear Systems

Solve a system of n linear equations with m unknowns:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_1 &= y_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_2 &= y_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_n &= y_n \end{aligned}$$

In matrix notation, we have

$$Ax = y$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Gaussian Elimination ($m = n$)

- **Forward elimination.** Reduce $[A \mid y]$ to an upper triangular system

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & y_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & y_n \end{array}\right] \longrightarrow \left[\begin{array}{cccc|c} * & * & \cdots & * & * \\ & * & \cdots & * & * \\ & & \ddots & \vdots & \vdots \\ & & & * & * \end{array}\right]$$

using the row following operation

$$-\frac{a_{ji}}{a_{ii}} R_i + R_j \to R_j, \qquad a_{ii} \neq 0$$

for $j = i+1, i+2, \ldots, n$

Gaussian Elimination

- Backward substitution

$$\begin{aligned} x_n &= \frac{y_n}{a_{nn}} \\ x_i &= \frac{y_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}}, \qquad i = n-1, n-2, \ldots, 2, 1 \end{aligned}$$

- If A is singular (non-invertible) and has rank r, the elimination process will terminate after r steps. In this case, the linear system is solvable if and only if

$$y_{r+1} = \cdots = y_n = 0$$

The solution can be found by arbitrarily choosing $x_{r+1}, \ldots, x_n$.

LU Factorization

- If Gaussian elimination can be performed on the linear system $Ax = b$ without row interchanges, then matrix A can be factored into a product of lower tringular matrix L and upper triangular matrix U, that is,

$$A = LU.$$

Here,

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ m_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ m_{n1} & \cdots & m_{n,n-1} & 1 \end{bmatrix} \qquad U = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1,n}^{(n-1)} \\ 0 & \cdots & 0 & a_{nn}^{(n)} \end{bmatrix}$$

LU Factorization

- However, not every nonsingular (invertible) matrix allows an LU factorization.
  - *Example*. $A = \begin{bmatrix} 0 & 2 \\ 3 & 0 \end{bmatrix}$ has no LU factorization.
- For each nonsingular (invertible) $n \times n$ matrix A, there exists a permutation matrix P such that PA has an LU factorization.
  - Note that $P = (e_{p(1)}, \ldots, e_{p(n)})$ where

    $e_1, \ldots, e_n$ are the columns of the identity matrix I_n

    $p(1), \ldots, p(n)$ is a permutation of $1, \ldots, n$

  Example. Left multiplying A by permutation matrix $P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ interchanges its rows:

$$PA = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 \\ 3 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix},$$

  which can now be factored into LU.

LU Factorization

To solve linear system $Ax = y$:

1. In Python, decompose $A = PLU$. This means that

$$P^T A = LU \quad \Rightarrow \quad P^T \underbrace{Ax}_{y} = L \underbrace{Ux}_{z}$$

```
from scipy.linalg import lu

P, L, U = lu(A)
print(P);    # prints permutation matrix P
print(L);    # prints lower triangular matrix L
print(U);    # prints upper triangular matrix U
```

2. solve $Lz = P^T y$ via forward substitution
3. solve $Ux = z$ via backward substitution

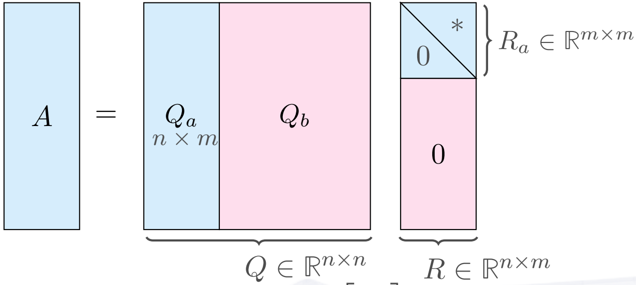
QR Factorization

- Gram-Schmidt produces orthonormal vectors from linearly independent vectors of A.

$$
\begin{aligned}
A &= \begin{bmatrix} \vdots & \vdots & & \vdots \\ a_1 & a_2 & \cdots & a_n \\ \vdots & \vdots & & \vdots \end{bmatrix} \\
&= \underbrace{\begin{bmatrix} \vdots & \vdots & & \vdots \\ q_1 & q_2 & \cdots & q_n \\ \vdots & \vdots & & \vdots \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{bmatrix}}_{R}, \qquad r_{ij} = q_i^T a_j
\end{aligned}
$$

QR Factorization

Consider a matrix $A \in \mathbb{R}^{n\times m}$ with rank $m < n$.



$$
\begin{aligned}
&= QR = \begin{bmatrix} Q_a & Q_b \end{bmatrix} \begin{bmatrix} R_a \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \vdots & & \vdots & \vdots & & \vdots \\ q_1 & \cdots & q_m & q_{m+1} & \cdots & q_n \\ \vdots & & \vdots & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} R_a \\ 0 \end{bmatrix} \\
&= Q_a R_a
\end{aligned}
$$

QR Factorization

- Using `numpy`:

```
from numpy.linalg import qr

Q, R = qr(A);                          # default: reduced factorization
Q, R = qr(A, mode = "complete");       # complete factorization
```

- Using `scipy`:

```
from scipy.linalg import qr

Q, R = qr(A);                          # default: complete decomposition
Q, R = qr(A, mode = "economic");       # reduced decomposition
```

Linear Least Squares Problem

- *Algorithm.* Linear Least Squares via QR Factorization

1. Factor
$$A = QR$$
where $Q = [Q_a, Q_b] \in \mathbb{R}^{n \times n}$ with $Q_a \in \mathbb{R}^{n \times m}$
2. Compute $z_a = Q_a^T y$.
3. Solve $R_a x = z_a$ by back substitution.

Eigendecomposition

- Consider a data matrix A with a full set of n independent eigenvectors $x_1, x_2, \ldots, x_n$ with corresponding eigenvalues $\lambda_1, \ldots, \lambda_n$. Denote eigenvector matrix

$$X = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix},$$

which is invertible. Hence,

$$AX = \begin{bmatrix} Ax_1 & \cdots & Ax_n \end{bmatrix} = \begin{bmatrix} \lambda_1 x_1 & \cdots & \lambda_n x_n \end{bmatrix} = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} \underbrace{\begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}}_{\Lambda}$$

- This gives the factorization: $A = X\Lambda X^{-1}$.

Singular Value Decomposition

- Denote orthogonal matrices

$$U = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix} \qquad V = \begin{bmatrix} v_1 & \cdots & v_m \end{bmatrix}$$

Then, we can write

$$AV = A\begin{bmatrix} v_1 & \cdots & v_m \end{bmatrix} = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix} \underbrace{\left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & 0 \\ & & \sigma_r & \\ \hline & 0 & & 0 \end{array}\right]}_{\Sigma} = U\Sigma$$

- This gives the factorization: $A = U\Sigma V^T$, called Singular Value Decomposition

Singular Value Decomposition

$$\underset{n\times m}{A} = \underbrace{\begin{bmatrix} \underset{n\times r}{U_r} & * & * \end{bmatrix}}_{U \in \mathbb{R}^{n\times n}} \underbrace{\begin{bmatrix} \underset{r\times r}{\Sigma_r} & 0 \\ 0 & 0 \\ \multicolumn{2}{c}{0} \end{bmatrix}}_{\Sigma \in \mathbb{R}^{n\times m}} \underbrace{\begin{bmatrix} \underset{r\times m}{V_r^T} \\ * \end{bmatrix}}_{V \in \mathbb{R}^{m\times m}}$$

$$= \underset{n\times r}{U_r} \; \underset{r\times r}{\Sigma_r} \; \underset{r\times m}{V_r^T}$$

Singular Value Decomposition

- Full SVD:

```python
from numpy.linalg import svd
U, S, Vh = svd(A);
```

- Reduced SVD:

```python
from numpy.linalg import svd
U, S, Vh = svd(A, full_matrices=False);  # reduced SVD
```

- If $A = U\Sigma V^T = U_r \Sigma_r V_r^T$, then a solution to linear system $Ax = y$ is given by

$$x_* = \underbrace{V_r \Sigma_r^{-1} U_r^T}_{A^\dagger} y, \qquad \text{where } \Sigma_r^{1} = \begin{bmatrix} \lambda_1^{-1} & & \\ & \ddots & \\ & & \lambda_r^{-1} \end{bmatrix},$$

$A^\dagger$ is called pseudo-inverse or Moore-Penrose inverse of A.

Thank you for your attention!