Mathematics for Robotics (ROB-GY 6013 Section A)

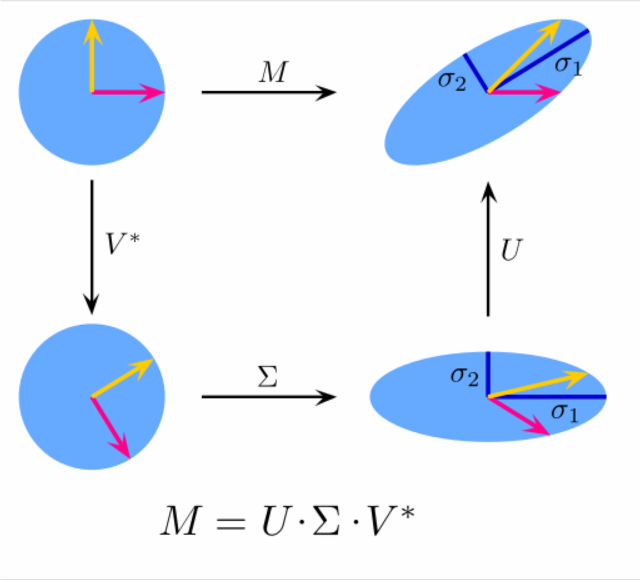
- **Week 14:**
  - Matrix factorizations
    - QR factorization
    - SVD
    - LU factorization
  - Euler’s Method
  - Newton-Raphson Method
- **Extra (not on final):** A taste of optimization

Mathematics for Robotics (ROB-GY 6013 Section A)

- Exam Review Tomorrow
- Expanded Office Hours this week
- Homework Extension

Thinking about Matrix Factorizations

- A matrix factorization can:
  - Reveal the structure present inside a matrix **(diagonalization)**
  - Decomposing a linear transformation into a sequence of simpler linear transformations **(singular value decomposition)**
  - Be useful in a numerical algorithm **(QR factorization with QR algorithm to find eigenvalues)**
  - So on and so forth…



QR Factorization

- **Definition:** An $n \times m$ matrix R is upper triangular if $R_{ij} = 0$ for all $i > j$.

QR Factorization

- **Definition:** An $n \times m$ matrix R is upper triangular if $R_{ij} = 0$ for all $i > j$.
- **Theorem: (QR Decomposition or Factorization)** Let A be a real $n \times m$ matrix with **linearly independent columns**. Then there exist an $n \times m$ matrix Q with orthonormal columns and an $m \times m$ upper triangular matrix R such that

$$A = QR$$

QR Factorization

- **Definition:** An $\underline{n \times m}$ matrix R is upper triangular if $R_{ij} = 0$ for all $i > j$.
- **Theorem: (QR Decomposition or Factorization)** Let A be a real $\underline{n \times m}$ matrix with **linearly independent columns**. Then there exist an $\underline{n \times m}$ matrix Q with orthonormal columns and an $\underline{m \times m}$ upper triangular matrix R such that

$$A = QR$$

QR Factorization

- **Definition:** An $\underline{n \times m}$ matrix R is upper triangular if $R_{ij} = 0$ for all $i > j$.
- **Theorem: (QR Decomposition or Factorization)** Let A be a real $\underline{n \times m}$ matrix with **linearly independent columns**. Then there exist an $\underline{n \times m}$ matrix Q with orthonormal columns and an $\underline{m \times m}$ upper triangular matrix R such that

$$A = QR$$

- **Remarks:**
  - $Q^TQ = I$
  - Columns of A linearly independent $\iff R$ is invertible
  - Proof/derivation by Gram-Schmidt Process

Proof: QR Factorization

Partition A into columns, $A = \begin{bmatrix} A_1 & A_2 & \cdots & A_m \end{bmatrix}$, $A_i \in \mathbb{R}^n$, and use the inner product $\langle x, y \rangle = x^\top y$. For $1 \le k \le n$, $\{A_1,\ A_2,\ \cdots,\ A_m\} \to \{v_1,\ v_2,\ \cdots,\ v_m\}$ by

for $k = 1 : m$
    $v^k = A_k$
    for $j = 1 : k-1$
        $v^k = v^k - \langle A_k, v^j \rangle v^j$
    end
    $v^k = \frac{v^k}{\|v^k\|}$
end

By construction, $Q := \begin{bmatrix} v^1 & v^2 & \cdots & v^m \end{bmatrix}$ has orthonormal columns, and hence $Q^\top Q = I_{m\times m}$ because $[Q^\top Q]_{ij} = \langle v^i, v^j \rangle = 1, i = j$ and zero otherwise.

What about R? By construction, $A_i \in \text{span}\{v^1, \cdots, v^i\}$, with $A_i = \langle A_1, v^1 \rangle v^1 + \langle A_2, v^2 \rangle v^2 + \cdots + \langle A_i, v^i \rangle v^i$. We define

$$R_i := \begin{bmatrix} \langle A_1, v^1 \rangle \\ \vdots \\ \langle A_i, v^i \rangle \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

where $R_{ij} = 0$ for $i < j \le n$. The coefficients in R can be extracted directly from the Gram-Schmdit Algorithm; no extra computations are required. By construction, $A_i = QR_i$ and thus we have $A = QR$.

QR Factorization: Over/under-determined equations

- If $Ax = b$ is ***over*determined** with columns of A linearly independent.

- If $Ax = b$ is ***under*determined** with columns of A linearly independent.

QR Factorization: Over/under-determined equations

- If $Ax = b$ is ***overdetermined*** with columns of A linearly independent.
    - Solving normal equations:

- If $Ax = b$ is ***underdetermined*** with columns of A linearly independent.
    - Solving normal equations:

QR Factorization: Over/under-determined equations

- If $Ax = b$ is ***over*determined** with columns of A linearly independent.
  - Solving normal equations: $R\hat{x} = Q^T b$

- If $Ax = b$ is ***under*determined** with columns of A linearly independent.
  - Solving normal equations: $\hat{x} = Q(R^T)^{-1} b$

QR Factorization: Over/under-determined equations

- If $Ax = b$ is ***over*determined** with columns of A linearly independent.
  - Solving normal equations: $R\hat{x} = Q^T b$ **No inverse!**

$$\begin{bmatrix} r_1 & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{bmatrix} \hat{x} = Q^T b$$

Back-substitution!

- If $Ax = b$ is ***under*determined** with columns of A linearly independent.
  - Solving normal equations: $\hat{x} = Q(R^T)^{-1} b$

QR Factorization: Overdetermined equations

- Begin from normal equations and substitute QR for A:

$$A^T A\hat{x} = A^T b$$

$$(QR)^T (QR)\hat{x} = (QR)^T b$$

$$R^T Q^T QR\hat{x} = R^T Q^T b$$

$$R^T R\hat{x} = R^T Q^T b$$

- Invertibility of R and thus, R^T: $R\hat{x} = Q^T b$

QR Factorization: Underdetermined equations

- Begin from normal equations and factorize A^T into QR: (A^T has linearly independent columns)

$$\hat{x} = A^T (AA^T)^{-1} b$$

$$\hat{x} = (QR)[(QR)^T (QR)]^{-1} b$$

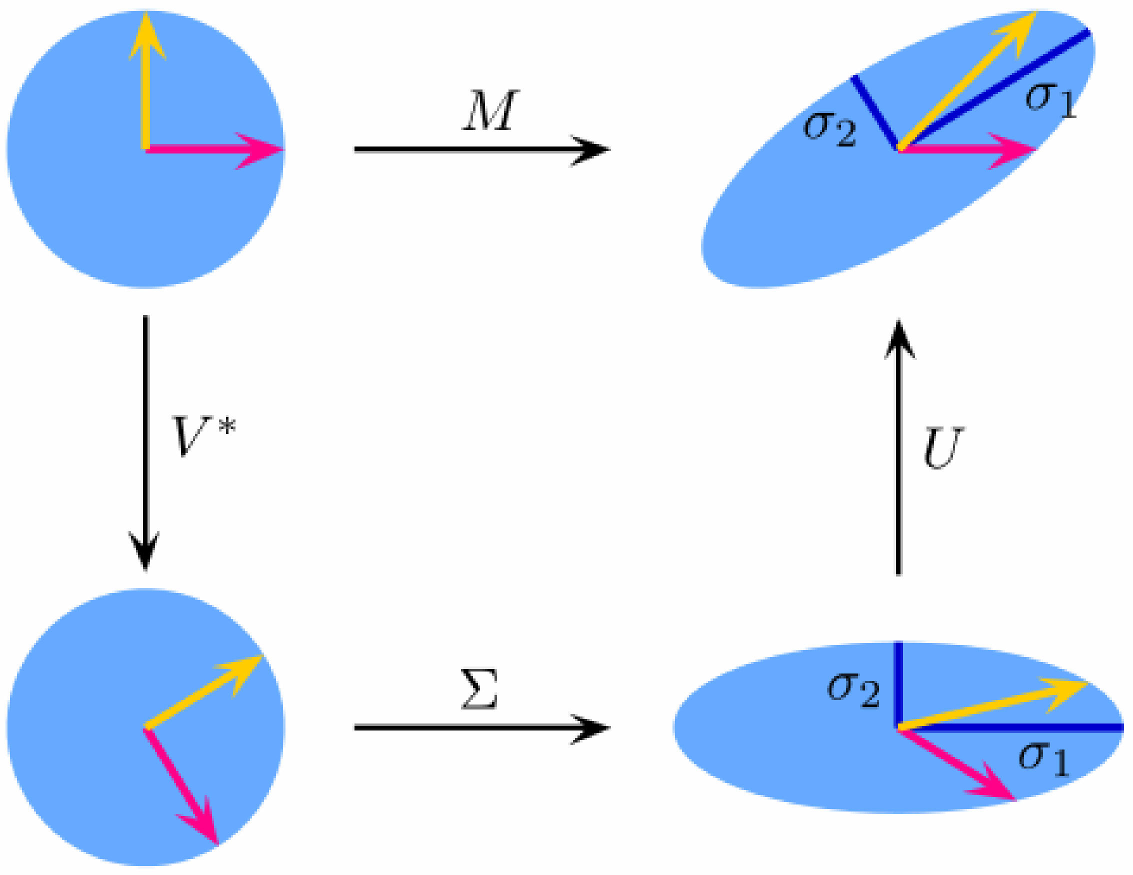
$$\hat{x} = QR(R^T Q^T QR)^{-1} b$$

$$\hat{x} = QR(R^T R)^{-1} b$$

$$\hat{x} = QRR^{-1} (R^T)^{-1} b$$

- Invertibility of R and R^T, thus: $\hat{x} = Q(R^T)^{-1} b$
- Note that inverting triangular matrices is much easier than inverting full matrices

Singular Value Decomposition (SVD)



$$M = U \cdot \Sigma \cdot V^*$$

Definition: Rectangular Diagonal Matrix

- An $n \times m$ matrix Σ is a **Rectangular Diagonal Matrix** if $\Sigma_{ij} = 0$ for $i \neq j$.

Definition: Rectangular Diagonal Matrix

- An $n \times m$ matrix Σ is a **Rectangular Diagonal Matrix** if $\Sigma_{ij} = 0$ for $i \neq j$.
- The **diagonal** of Σ is the set of all Σ_{ii}, $1 \leq i \leq \min(n, m)$.

Definition: Rectangular Diagonal Matrix

- An $n \times m$ matrix Σ is a **Rectangular Diagonal Matrix** if $\Sigma_{ij} = 0 \text{ for } i \neq j$.
- The **diagonal** of Σ is the set of all Σ_{ii}, $1 \leq i \leq \min(n, m)$.
- An alternative and equivalent way to define a **Rectangular Diagonal Matrix** is:
  a) (tall matrix) $n > m$

     Σ_d is an $m \times m$ diagonal matrix.
     $$\Sigma = \begin{bmatrix} \Sigma_d \\ 0 \end{bmatrix}$$

  b) (wide matrix) $n < m$

     Σ_d is an $n \times \mathrm{n}$ diagonal matrix.
     $$\Sigma = \begin{bmatrix} \Sigma_d & 0 \end{bmatrix}$$
- The diagonal of Σ is equal to the diagonal of Σ_d.

Theorem: Singular Value Decomposition

- Every $n \times m$ **real** matrix A can be factored as $A = U \cdot \Sigma \cdot V^T$

Theorem: Singular Value Decomposition

- Every $n \times m$ **real** matrix A can be factored as $A = U \cdot \Sigma \cdot V^T$

Note: If A were **complex**, we take the conjugate transpose of V, as indicated by *

$$A = U \cdot \Sigma \cdot V^*$$

Theorem: Singular Value Decomposition

- Every $n \times m$ **real** matrix A can be factored as $A = U \cdot \Sigma \cdot V^T$

  where U is an $n \times n$ orthogonal matrix, V is an $m \times m$ orthogonal matrix,
  Σ is an $n \times m$ rectangular diagonal matrix,

Theorem: Singular Value Decomposition

- Every $n \times m$ **real** matrix A can be factored as $A = U \cdot \Sigma \cdot V^T$

  where U is an $n \times n$ orthogonal matrix, V is an $m \times m$ orthogonal matrix,
  Σ is an $n \times m$ rectangular diagonal matrix,

  and the diagonal of Σ, $\mathrm{diag}(\Sigma) = \left[\sigma_1, \sigma_2, \cdots, \sigma_p\right]$

  satisfies $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$, for $p := \min(n, m)$.

Theorem: Singular Value Decomposition

- Every $n \times m$ **real** matrix A can be factored as $A = U \cdot \Sigma \cdot V^T$

  where U is an $n \times n$ orthogonal matrix, V is an $m \times m$ orthogonal matrix,
  Σ is an $n \times m$ rectangular diagonal matrix,

  and the diagonal of Σ, $\mathrm{diag}(\Sigma) = \left[\sigma_1, \sigma_2, \cdots, \sigma_p\right]$

  satisfies $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$, for $p := \min(n, m)$.

$\{\sigma_1, \sigma_2, \ldots, \sigma_p\}$ are called the singular values of A

Theorem: Singular Value Decomposition

- Every $n \times m$ **real** matrix A can be factored as $A = U \cdot \Sigma \cdot V^T$

  where U is an $n \times n$ orthogonal matrix, V is an $m \times m$ orthogonal matrix,
  Σ is an $n \times m$ rectangular diagonal matrix,

  and the diagonal of Σ, $\mathrm{diag}(\Sigma) = \left[\sigma_1, \sigma_2, \cdots, \sigma_p\right]$

  satisfies $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$, for $p := \min(n, m)$.

$\{\sigma_1, \sigma_2, \ldots, \sigma_p\}$ are called the singular values of A

Theorem: Singular Value Decomposition

- Every $n \times m$ **real** matrix A can be factored as $A = U \cdot \Sigma \cdot V^T$

  where U is an $n \times n$ orthogonal matrix, V is an $m \times m$ orthogonal matrix, Σ is an $n \times m$ rectangular diagonal matrix,

  and the diagonal of Σ, $\operatorname{diag}(\Sigma) = \left[\sigma_1, \sigma_2, \cdots, \sigma_p\right]$

  satisfies $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$, for $p := \min(n, m)$.

  - columns of U are eigenvectors of AA^T
  - columns of V are eigenvectors of A^TA
  - $\{\sigma_1{}^2, \sigma_2{}^2, \ldots, \sigma_p{}^2\}$ are eigenvalues of both AA^T and A^TA

$\{\sigma_1, \sigma_2, \ldots, \sigma_p\}$ are called the singular values of A

Proof: SVD (Textbook Section 4.2.2)

- We first get the eigenvectors of A^TA and play around with them

Proof: $A^\top A$ is $m \times m$, real, and symmetric. Hence, there exists a set of orthonormal eigenvectors $\{v^1, \ldots, v^m\}$ such that

$$A^\top A v^j = \lambda_j v^j.$$

Without loss of generality, we can assume that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m \geq 0$. If not, we simply re-order the v^i's to make it so. For $\lambda_j > 0$, say $1 \leq j \leq r$, we define

$$\sigma_j = \sqrt{\lambda_j}$$

and

$$q^j = \frac{1}{\sigma_j} A v^j \in \mathbb{R}^n$$

Proof: SVD (Textbook Section 4.2.2)

- More playing around

Claim 4.9 *For* $1 \leq i,\ j \leq r$, $\left(q^i\right)^\top q^j = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$. *That is, the vectors* $\{q^1, q^2, \ldots, q^r\}$ *are orthonormal.*

Proof of Claim:

$$
\begin{aligned}
\left(q^i\right)^\top q^j &= \frac{1}{\sigma_i}\frac{1}{\sigma_j}\left(v^i\right)^\top A^\top A v^j \\
&= \frac{\lambda_j}{\sigma_i \sigma_j}\left(v^i\right)^\top v^j \\
&= \begin{cases} \frac{\lambda_i}{(\sigma_i)^2} & i = j \\ 0 & i \neq j \end{cases} \\
&= \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}
\end{aligned}
$$

Proof: SVD (Textbook Section 4.2.2)

- Now we have the eigenvectors of AA^T! And they have the same eigenvalues!

Claim 4.10 *The vectors* $\{q^1, q^2, \ldots, q^r\}$ *are eigenvectors of* $AA^\top$ *and the corresponding e-values are* $\{\lambda_1, \lambda_2, \ldots, \lambda_r\}$.

Proof of Claim: For $1 \leq i \leq r$, $\lambda_i > 0$ and

$$\begin{aligned} AA^\top q^i :=& AA^\top \left(\frac{1}{\sigma_i} A v^i \right) \\ =& \frac{1}{\sigma_i} A \left(A^\top A \right) v^i \\ =& \frac{\lambda_i}{\sigma_i} A v^i \\ =& \lambda_i q^i, \end{aligned}$$

and thus q^i is an e-vector of $AA^\top$ with e-value λ_i. The claim is also an immediate consequence of Lemma 2.63.

Proof: SVD (Textbook Section 4.2.2)

- Let's create the 3 ingredients of SVD

From Fact 2.61, if $r < n$, then the remaining e-values of $AA^\top$ are all zero. Moreover, we can extend the q^i's to an orthonormal basis for $\mathbb{R}^n$ satisfying $AA^\top q^i = 0$, for $r+1 \leq i \leq n$. Define

$$U := \begin{bmatrix} q^1 & q^2 & \cdots & q^n \end{bmatrix} \text{ and } V := \begin{bmatrix} v^1 & v^2 & \cdots & v^m \end{bmatrix}.$$

Also, define $\Sigma = n \times m$ by

$$\Sigma_{ij} = \begin{cases} \sigma_i \delta_{ij} & 1 \leq i,\ j \leq r \\ 0 & \text{otherwise.} \end{cases}$$

Then, Σ is rectangular diagonal with

$$\text{diag}\,(\Sigma) = [\sigma_1,\ \sigma_2,\ \cdots,\ \sigma_r,\ 0,\ \cdots,\ 0]$$

Proof: SVD (Textbook Section 4.2.2)

- Now we must show when we put everything together, that $U\Sigma V^T$ is equal to A.
- Or equivalently, $U^TAV = \Sigma$, which is easier to show.
  - The entries of Σ are either 0 or a singular value.
  - Just compute each entry of U^TAV and check each entry is either 0 or the correct singular value

Proof: SVD (Textbook Section 4.2.2)

To complete the proof of the theorem, it is enough to show[1] that $U^\top AV = \Sigma$. We note that the ij element of this matrix is

$$(U^\top AV)_{ij} = q_i^\top Av^j$$

If $j > r$, then $A^\top Av^j = 0$, and thus $(q^i)^\top Av^j = 0$, as required. If $i > r$, then q^i was selected to be orthogonal to

$$\{q^1, \cdots, q^r\} = \{\frac{1}{\sigma_1}Av^1, \frac{1}{\sigma_2}Av^2, \cdots, \frac{1}{\sigma_r}Av^r\}$$

and thus $\left(q^i\right)^\top Av^j = 0$. Hence we now consider $1 \leq i, j \leq r$ and compute that

$$\begin{aligned}
\left(U^\top AV\right)_{ij} &= \frac{1}{\sigma_i}\left(v^i\right)^\top A^\top Av^j \\
&= \frac{\lambda_j}{\sigma_i}\left(v^i\right)^\top v^j \\
&= \sigma_i\delta_{ij}
\end{aligned}$$

[1] Because $U^\top U = I$ and $V^\top V = I$, it follows that $A = U\Sigma V^\top \iff U^\top AV = \Sigma$.

Computing SVD by hand

- When do you need to do this?

Computing SVD by hand

- When do you need to do this?
    - Yes, this will ***probably*** be on the final exam.

Computing SVD by hand

- When do you need to do this?
  - Yes, this will ***probably*** be on the final exam.
- How do you do it?
  - Theorem provides blueprint
    - Square root of eigenvalues → Singular Values
    - Eigenvectors → column vectors U and V

Computing SVD by hand

- When do you need to do this?
  - Yes, this will ***probably*** be on the final exam.
- How do you do it?
  - Theorem provides blueprint
    - Square root of eigenvalues → Singular Values
    - Eigenvectors → column vectors U and V

Computing SVD by hand

- Try this example at home as "homework" for studying for the exam

$$M = U.\Sigma.V^{\dagger}$$

where

$$M = \begin{pmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{pmatrix}$$

$$U = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix}$$

$$V = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{3\sqrt{2}} & -\frac{2}{3} \\ \frac{1}{\sqrt{2}} & \frac{1}{3\sqrt{2}} & \frac{2}{3} \\ 0 & -\frac{2\sqrt{2}}{3} & \frac{1}{3} \end{pmatrix}$$

Meaning of “Small” Singular Values

- Numerical rank
- Image compression

Numerical Rank

- Is this LI?

$$A = \begin{bmatrix} 1 & 10^4 \\ 0 & 1 \end{bmatrix}$$

Numerical Rank

- Is this LI?
    - Yes..?

$$A = \begin{bmatrix} 1 & 10^4 \\ 0 & 1 \end{bmatrix}$$

Numerical Rank

- Is this LI?
  - Yes..?
  - *Numerically*, no.

$$A = \begin{bmatrix} 1 & 10^4 \\ 0 & 1 \end{bmatrix}$$

Numerical Rank

- Is this LI?
  - Yes..?
  - *Numerically*, no.
  - Try svd() in MATLAB

$$A = \begin{bmatrix} 1 & 10^4 \\ 0 & 1 \end{bmatrix}$$

Numerical Rank

- Is this LI?
  - Yes..?
  - *Numerically*, no.
  - Try svd() in MATLAB

$$A = \begin{bmatrix} 1 & 10^4 \\ 0 & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & -0.0001 \\ 0.0001 & 1 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 10000 & 0 \\ 0 & 0.0001 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.0001 & -1 \\ 1 & 0.0001 \end{bmatrix}$$

Numerical Rank

- Is this LI?
  - Yes..?
  - *Numerically*, no.
  - Try svd() in MATLAB
- Numerical test for linear dependence:
  - Compare the value of the smallest to the largest singular value

$$A = \begin{bmatrix} 1 & 10^4 \\ 0 & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & -0.0001 \\ 0.0001 & 1 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 10000 & 0 \\ 0 & 0.0001 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.0001 & -1 \\ 1 & 0.0001 \end{bmatrix}$$

Smallest Non-Zero Singular Value

- What is the “length” of a matrix?

Smallest Non-Zero Singular Value

- **What is the "length" of a matrix?**
- **Definition:** Given an $n \times m$ real matrix A, the matrix norm induced by the Euclidean vector norm is given by:

Smallest Non-Zero Singular Value

- **What is the "length" of a matrix?**
- **Definition:** Given an $n \times m$ real matrix A, the matrix norm induced by the Euclidean vector norm is given by:

$$||A|| := \max_{x^\top x = 1} ||Ax||$$

Smallest Non-Zero Singular Value

- **What is the "length" of a matrix?**
- **Definition:** Given an $n \times m$ real matrix A, the matrix norm induced by the Euclidean vector norm is given by:

$$||A|| := \max_{x^\top x = 1} ||Ax|| = \sqrt{\lambda_{\max}(A^\top A)}$$

Smallest Non-Zero Singular Value

- **What is the "length" of a matrix?**
- **Definition:** Given an $n \times m$ real matrix A, the matrix norm induced by the Euclidean vector norm is given by:

$$||A|| := \max_{x^\top x = 1} ||Ax|| = \sqrt{\lambda_{\max}(A^\top A)}$$

- If A is square and invertible, the smallest non-zero singular value measures the distance from A to the nearest singular matrix

Image Compression with SVD

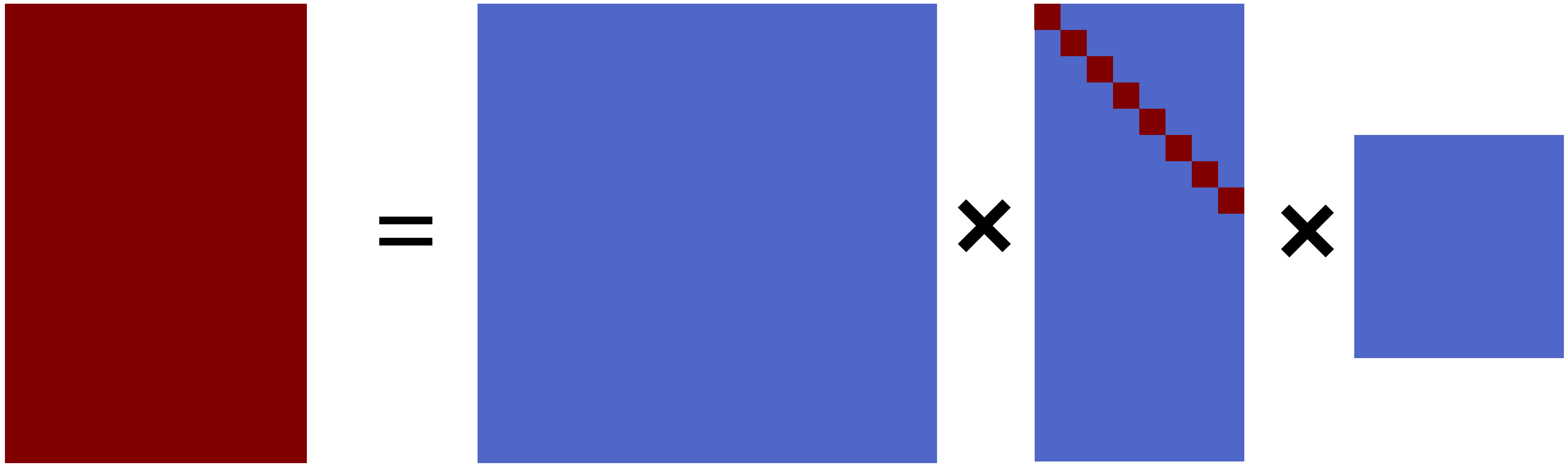


Image Compression with SVD

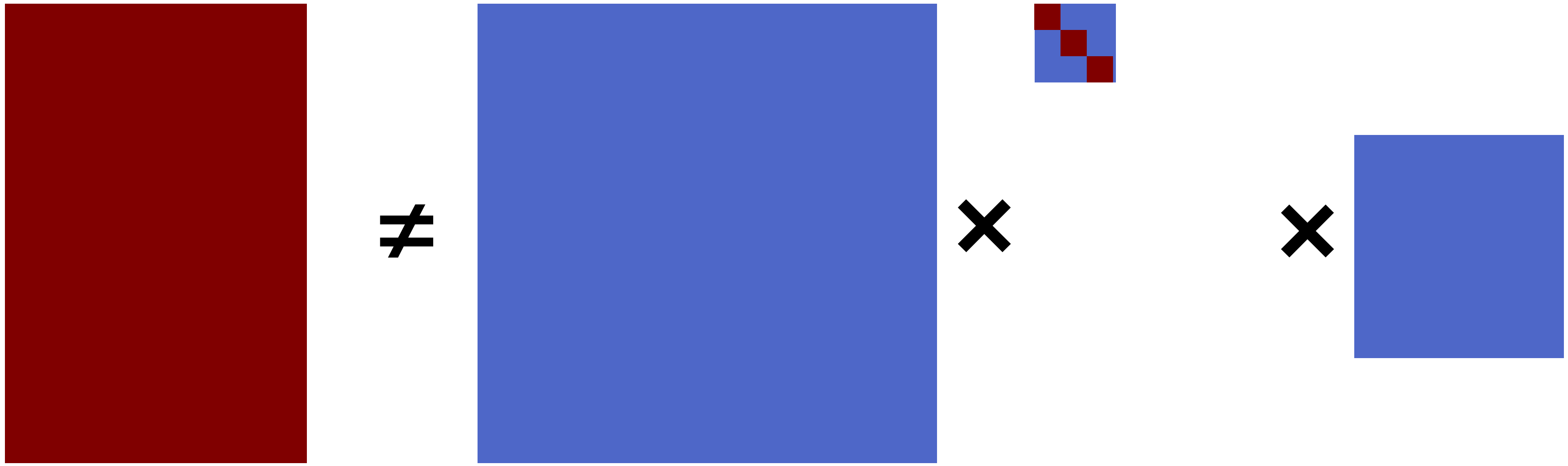


Image Compression with SVD

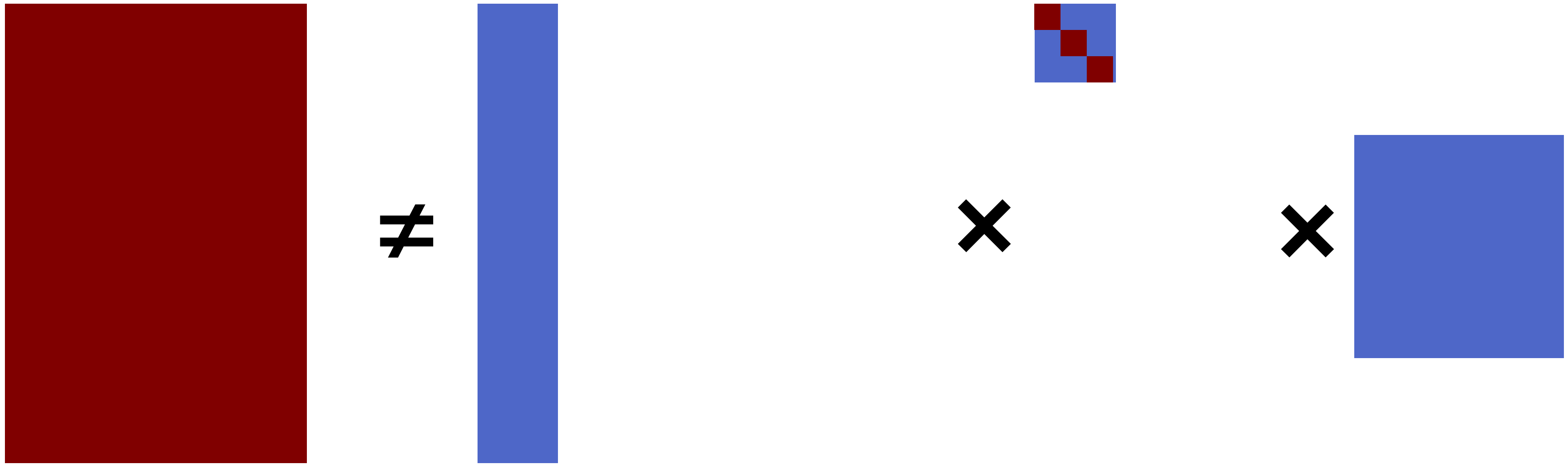


Image Compression with SVD

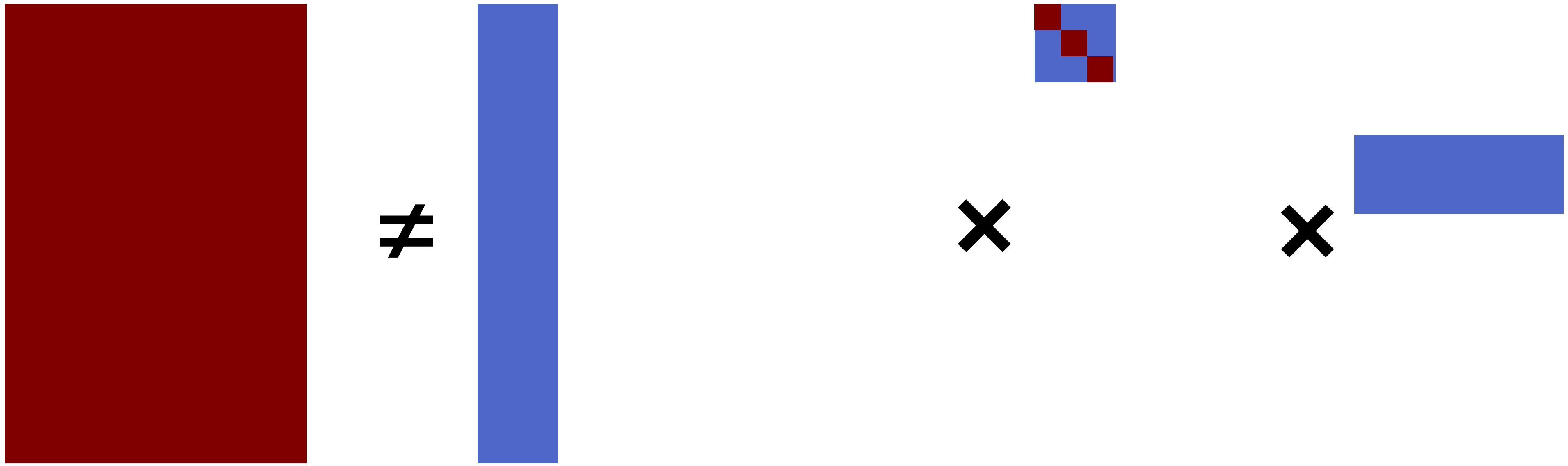


Image Compression with SVD

- Smaller matrices → much less data

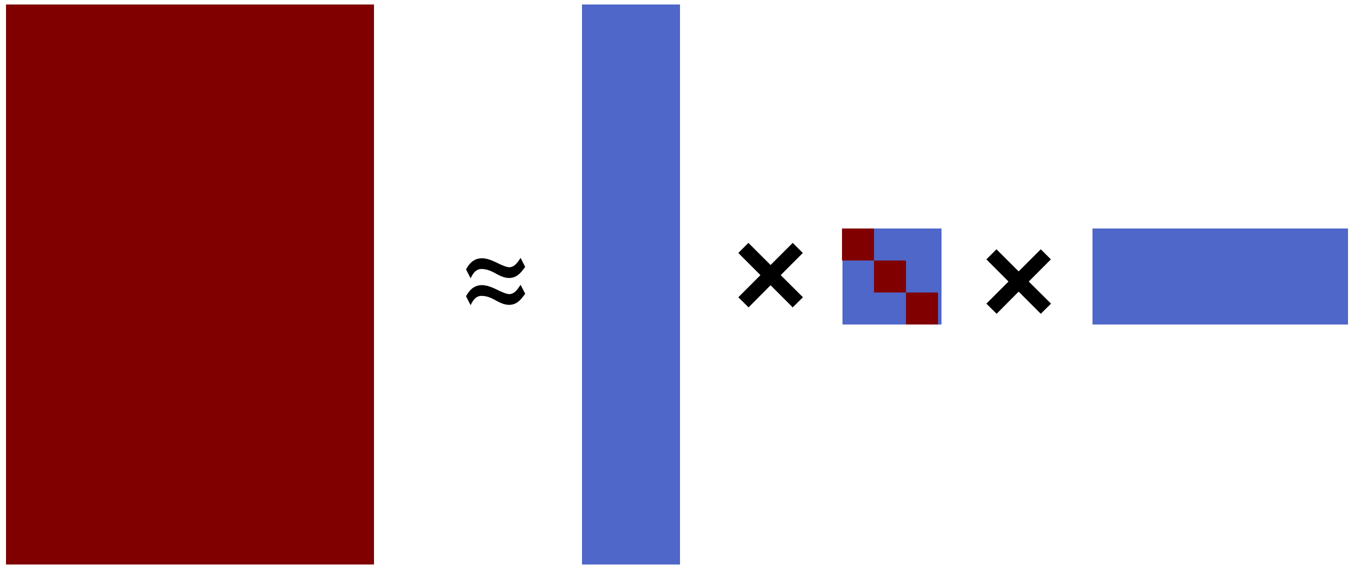


Image Compression with SVD

- https://timbaumann.info/svd-image-compression-demo/
- Lots of linear algebra in Photoshop and similar image processing methods
- Think back to principal component analysis:
    - “Rotating” data to get the principal directions

LU Factorization

- Lower upper factorization
  - Useful because systems of equations represented by triangular matrices are easy to solve
    - Lower triangular matrix: forward-substitution
    - Upper triangular matrix: back-substitution
  - **Uni-lower triangular matrix:** lower triangular matrix with ones along diagonal)

LU Factorization: Solve $Ax = b$

- Given a solvable system of equations $Ax = b$
  - A is square and invertible
- If I can factor A into LU

$$Ax = b$$

$$LUx = b$$

- Then: $Ux = y$ and $Ly = b$
- Solve $Ly = b$ with forward substitution and then $Ux = y$ with back substitution

LU Factorization: Solve $Ax = b$

- Given a solvable system of equations $Ax = b$
  - A is square and invertible
- If I can factor A into LU

$$Ax = b$$

$$LUx = b$$

- Then: $Ux = y$ and $Ly = b$
- Solve $Ly = b$ with forward substitution and then $Ux = y$ with back substitution
- Can we always do this?

LU Factorization: Solve $Ax = b$

- Given a solvable system of equations $Ax = b$
  - A is square and invertible
- If I can factor A into LU

$$Ax = b$$

$$LUx = b$$

- Then: $Ux = y$ and $Ly = b$
- Solve $Ly = b$ with forward substitution and then $Ux = y$ with back substitution
- Can we always do this?
  - Not quite. Sometime we will need to play around with A (swap its rows)

Permutation Matrix

- An $n \times n$ matrix P consisting of only zeros and ones and satisfying $P^T P = P P^T = I$ is called a permutation matrix.

Permutation Matrix

- An $n \times n$ matrix P consisting of only zeros and ones and satisfying $P^T P = P P^T = I$ is called a permutation matrix.

- A permutation matrix permutes.
- It is a jumbled up identity matrix.
- If you multiply A with P:
  - On the right (AP): permutes the order of the columns
  - On the left (PA): permutes the order of the rows

Permutation Matrix

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

$$PA = \begin{bmatrix} d & e & f \\ g & h & i \\ a & b & c \end{bmatrix} \qquad AP = \begin{bmatrix} c & a & b \\ f & d & e \\ i & g & h \end{bmatrix}$$

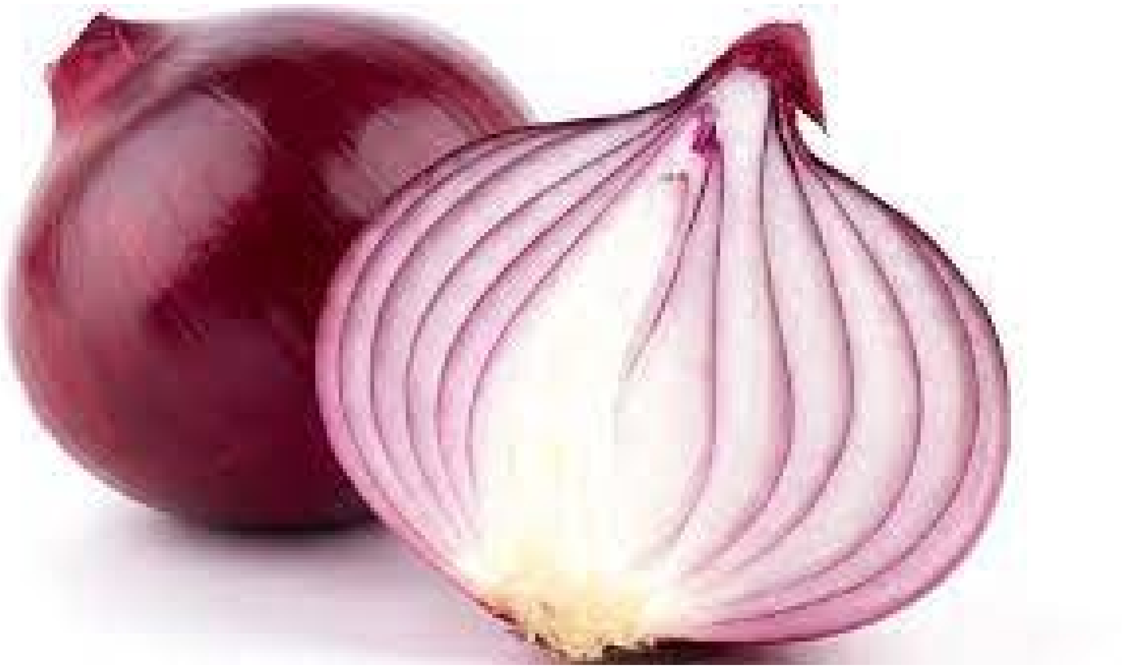
Permutation Matrix

- We will make use of this fact
  - On the left (PA): permutes the order of the rows
- Also note that the product of permutation matrices is another permutation matrix

LU Factorization Example

- There are different methods to do LU factorization, which will have different performance when implemented in code
- I will show you how the textbook does it by “peeling the onion.” You are free to do whatever you want on the exam as long as the answer is right

Peeling the Onion



Peeling the Onion

$$M = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix}$$

Peeling the Onion

$$\begin{array}{rl} M = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix} & \begin{array}{l} \textbf{row: } R_1 \\ \\ \\ \end{array} \\ \textbf{column: } C_1 & \end{array}$$

Peeling the Onion

$$\begin{array}{rl} & \begin{array}{l} \\ \end{array} \\ M = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix} & \begin{array}{l} \textbf{row: } R_1 \\ \\ \\ \end{array} \\ \textbf{column: } C_1 & \end{array}$$

- Note that C_1R_1 has the same numbers in the first row and column as M

$$C_1R_1 = \begin{bmatrix} 1 & 4 & 5 \\ 2 & \multicolumn{2}{c}{\boxed{\textbf{stuff}}} \\ 3 & & \end{bmatrix} \qquad M - C_1R_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 7 \\ 0 & 6 & 43 \end{bmatrix}$$

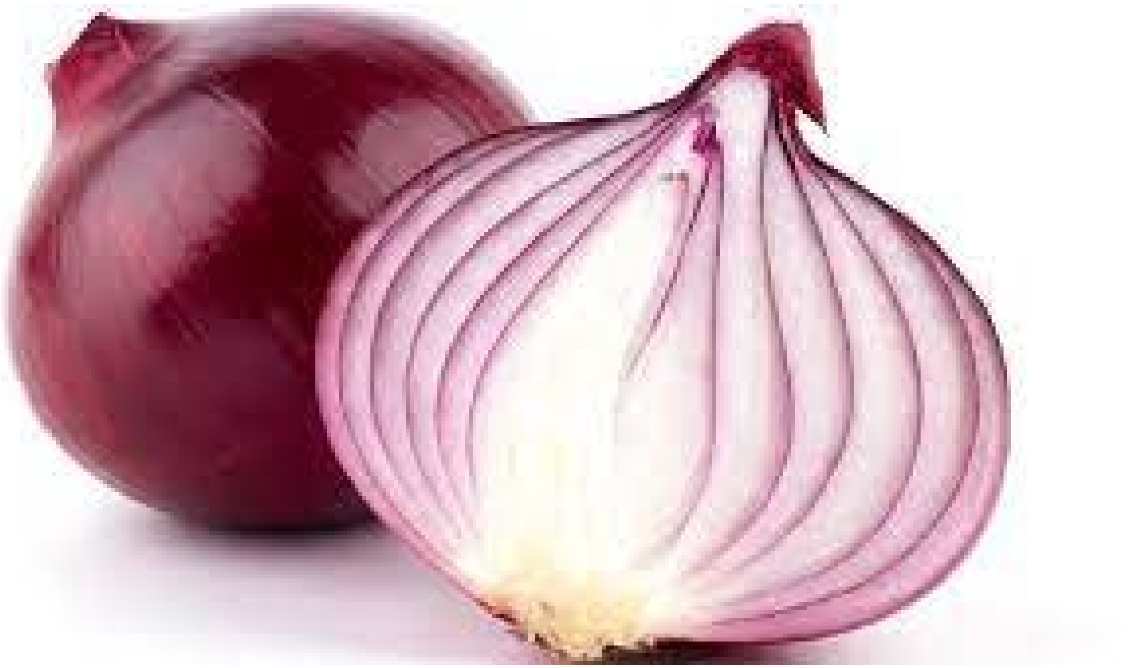
Peeling the Onion

$$M = \begin{array}{cl} \begin{bmatrix} 1 & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix} & \textbf{row: } R_1 \\ \textbf{column: } C_1 & \end{array}$$

- Note that C_1R_1 has the same numbers in the first row and column as M

$$C_1R_1 = \begin{bmatrix} 1 & 4 & 5 \\ 2 & \multicolumn{2}{c}{\text{stuff}} \\ 3 & & \end{bmatrix} \qquad M' = M - C_1R_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 7 \\ 0 & 6 & 43 \end{bmatrix}$$

Keep Peeling the Onion



Keep Peeling the Onion

$$M' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 7 \\ 0 & 6 & 43 \end{bmatrix}$$

$$C_2 R_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 7 \\ 0 & 6 & 42 \end{bmatrix}$$

$$M'' = M' - C_2 R_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Keep peeling until there is nothing left

$$M'' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C_3 R_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M''' = M'' - C_3 R_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Summary

$$M' = M - C_1 R_1$$

$$M'' = M' - C_2 R_2$$

$$M''' = M'' - C_3 R_3 = 0$$

Summary

$$M' = M - C_1 R_1$$

$$M'' = M' - C_2 R_2$$

$$M''' = M'' - C_3 R_3 = 0$$

$$M - C_1 R_1 - C_2 R_2 - C_3 R_3 = 0$$

$$M = C_1 R_1 + C_2 R_2 + C_3 R_3$$

$$M = \begin{bmatrix} C_1 & C_2 & C_3 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix}$$

Summary

$$M' = M - C_1 R_1$$

$$M'' = M' - C_2 R_2$$

$$M''' = M'' - C_3 R_3 = 0$$

$$M - C_1 R_1 - C_2 R_2 - C_3 R_3 = 0$$

$$M = C_1 R_1 + C_2 R_2 + C_3 R_3$$

$$M = \begin{bmatrix} C_1 & C_2 & C_3 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix}$$

$$M = LU$$

$$L := \begin{bmatrix} C_1 & C_2 & C_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 6 & 1 \end{bmatrix}$$

$$U := \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 5 \\ 0 & 1 & 7 \\ 0 & 0 & 1 \end{bmatrix}$$

Summary

$$M' = M - C_1 R_1$$
$$M'' = M' - C_2 R_2$$
$$M''' = M'' - C_3 R_3 = 0$$

$$M - C_1 R_1 - C_2 R_2 - C_3 R_3 = 0$$
$$M = C_1 R_1 + C_2 R_2 + C_3 R_3$$
$$M = \begin{bmatrix} C_1 & C_2 & C_3 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix}$$

$$M = LU$$
$$L := \begin{bmatrix} C_1 & C_2 & C_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 6 & 1 \end{bmatrix}$$
$$U := \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 5 \\ 0 & 1 & 7 \\ 0 & 0 & 1 \end{bmatrix}$$

- By recording our moves (when peeling the onion) we obtain the LU factorization

Case 1: Peeling the Onion (Slightly harder)

- We were lucky that the first entry of M was 1

$$M = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix} \quad \textbf{row: } R_1$$

$$\textbf{column: } C_1$$

$$C_1R_1 = \begin{bmatrix} 1 & 4 & 5 \\ 2 & \multicolumn{2}{c}{\text{stuff}} \\ 3 & & \end{bmatrix} \qquad M - C_1R_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 7 \\ 0 & 6 & 43 \end{bmatrix}$$

Case 1: Peeling the Onion (Slightly harder)

- If the first entry of M was some number k not equal to 1, this does not work.

$$M = \begin{bmatrix} 2 & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix}$$

row: R_1

column: C_1

Case 1: Peeling the Onion (Slightly harder)

- If the first entry of M was some number k not equal to 1, this does not work.

$$\begin{array}{cc} M = \begin{bmatrix} 2 & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix} & \textbf{row: } R_1 \\ \textbf{column: } C_1 & \end{array}$$

$$C_1 R_1 = \begin{bmatrix} 4 & 8 & 5 \\ 4 & \text{stuff} & \\ 6 & & \end{bmatrix}$$

Case 1: Peeling the Onion (Slightly harder)

- If the first entry of M was some number k not equal to 1, this does not work.

$$\begin{array}{cl} M = \begin{bmatrix} 2 & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix} & \begin{array}{l} \textbf{row: } R_1 \\ \\ \\ \end{array} \\ \textbf{column: } C_1 & \end{array}$$

$$C_1R_1 = \begin{bmatrix} 4 & 8 & 5 \\ 4 & \text{stuff} & \\ 6 & & \end{bmatrix}$$

$$M - C_1R_1 = \begin{bmatrix} -2 & -4 & -5 \\ -2 & \text{stuff} & \\ -3 & & \end{bmatrix}$$

Not all zeros!

Not all zeros!

Case 1: Peeling the Onion (Slightly harder)

- So let us divide the **column** by k.

$$\begin{array}{cc} & \\ M = & \left[\begin{array}{c|cc} k & 4 & 5 \\ \hline 2 & 9 & 17 \\ 3 & 18 & 58 \end{array}\right] \end{array} \quad \textbf{row: } R_1$$

$$\textbf{column: } C_1$$

$$\tilde{C}_1 = \frac{C_1}{k} = \begin{bmatrix} 1 \\ 2/k \\ 3/k \end{bmatrix}$$

Case 1: Peeling the Onion (Slightly harder)

- So let us divide the **column** by k.

$$\begin{array}{rl} M = \left[\begin{array}{c|cc} k & 4 & 5 \\ \hline 2 & 9 & 17 \\ 3 & 18 & 58 \end{array}\right] & \textbf{row: } R_1 \\ \textbf{column: } C_1 & \end{array}$$

$$\tilde{C}_1 = \frac{C_1}{k} = \begin{bmatrix} 1 \\ 2/k \\ 3/k \end{bmatrix} \qquad \tilde{C}_1 R_1 = \begin{bmatrix} k & 4 & 5 \\ 2 & \multicolumn{2}{c}{\boxed{\textbf{stuff}}} \\ 3 & & \end{bmatrix}$$

Case 1: Peeling the Onion (Slightly harder)

- So let us divide the **column** by k. **It works!**

$$M = \begin{array}{cl} \begin{bmatrix} k & 4 & 5 \\ 2 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix} & \textbf{row: } R_1 \\ \textbf{column: } C_1 & \end{array}$$

$$\tilde{C}_1 = \frac{C_1}{k} = \begin{bmatrix} 1 \\ 2/k \\ 3/k \end{bmatrix} \qquad \tilde{C}_1 R_1 = \begin{bmatrix} k & 4 & 5 \\ 2 & \multicolumn{2}{c}{\textbf{stuff}} \\ 3 & & \end{bmatrix} \qquad M - \tilde{C}_1 R_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \multicolumn{2}{c}{\textbf{stuff}} \\ 0 & & \end{bmatrix}$$

Case 1: Peeling the Onion

- In summary, you divide your column C_i by the entry M_{ii} .
    - This also ensures that the resulting lower triangular matrix obtained by combining all the columns are ***uni***-lower triangular (all ones along the diagonal)

Case 2: Peeling the Onion

- What if the entire column is zero? Cannot have division by zero.

$$M = \begin{array}{c|cc|l} 0 & 4 & 5 & \textbf{row: } R_1 \\ \hline 0 & 9 & 17 & \\ 0 & 18 & 58 & \\ \textbf{column: } C_1 & & & \end{array}$$

Case 2: Peeling the Onion

- Choose $\tilde{C}_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$

$$M = \begin{array}{l} \left[\begin{array}{ccc} 0 & 4 & 5 \\ \hline 0 & 9 & 17 \\ 0 & 18 & 58 \end{array}\right] \end{array} \begin{array}{l} \textbf{row: } R_1 \\ \\ \\ \end{array}$$

$$\tilde{C}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad \tilde{C}_1 R_1 = \begin{bmatrix} 0 & 4 & 5 \\ 0 & \multicolumn{2}{c}{\text{stuff}} \\ 0 & & \end{bmatrix} \qquad M - \tilde{C}_1 R_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \multicolumn{2}{c}{\text{stuff}} \\ 0 & & \end{bmatrix}$$

Case 3: Peeling the Onion

- What if the column is ***not*** all zero, but M_{ii} is zero?

$$\begin{array}{rl} M = \left[\begin{array}{c|cc} 0 & 4 & 5 \\ \hline 0 & 9 & 17 \\ 3 & 18 & 58 \end{array}\right] & \begin{array}{l} \textbf{row: } R_1 \\ \\ \\ \end{array} \\ \textbf{column: } C_1 & \end{array}$$

Case 3: Peeling the Onion

- What if the column is ***not*** all zero, but M_{ii} is zero?

$$M = \begin{array}{c} \begin{bmatrix} 0 & 4 & 5 \\ 0 & 9 & 17 \\ 3 & 18 & 58 \end{bmatrix} \\ \textbf{column: } C_1 \end{array} \begin{array}{l} \textbf{row: } R_1 \\ \\ \\ \\ \end{array}$$

- You will need to permute M. Here you swap rows 1 and 3, which results in Case 1.

$$P_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Case 3: Peeling the Onion

- You will need to keep track of all permutations at each step and multiply them all into one big permutation matrix at the end for the LU factorization with permutation

$$PA = LU$$

Peeling the Onion

- There are only three cases to consider:
- **Case 1:** Scale the column
- **Case 2:** Pick column to be
- **Case 3:** Permute and then apply Case 1.
    - (The textbook explains how to pick this permutation uniquely)

LU Factorization: Application

Solving Ax = b via LU Factorization

We seek to solve the system of linear equations $Ax = b$, when A is a real square matrix. Suppose we factor $P \cdot A = L \cdot U$, where P is a permutation matrix, L is lower triangular and U is upper triangular. Would that even be helpful for solving linear equations?

Because $P^\top \cdot P = I$, $\det(P) = \pm 1$ and therefore P is always invertible. Hence,

$$Ax = b \iff P \cdot Ax = P \cdot b \iff L \cdot Ux = P \cdot b.$$

If we define $Ux = y$, then $L \cdot Ux = P \cdot b$ becomes two equations

$$Ly = P \cdot b \tag{4.2}$$

$$Ux = y. \tag{4.3}$$

Furthermore,

$$(P \cdot A = L \cdot U) \implies \det(A) = \pm \det(L) \det(U)$$

and A is invertible if, and only if, both L and U are invertible. Our solution strategy is therefore to solve (4.2) by forward substitution, and then, once we have y in hand, we solve (4.3) by back substitution to find x, the solution to $Ax = b$.

Permutation Matrix for Performance

- In numerical solvers, such as in MATLAB or Julia, the LU factorization algorithm may insert a permutation matrix where it is not strictly necessary to improve numerical accuracy on large problems.

Numerical Methods

- Closed-form solutions to many engineering problems are unknown or do not exist
  - Nevertheless we still need to solve the problem
  - Use iterative procedures!
- Solving ODEs
  - Euler's Method, Heun's Method, Runge-Kutta Methods
- Root-finding
  - Applications: Inverse Kinematics
  - Newton-Raphson Method
- Playing with Taylor polynomials is a common theme for deriving these methods

Euler's Method

- **Covered only in lecture but will *probably* be on the exam.**
- Given first order differential equation of the form:

$$\frac{dy}{dt} = f(t_k, y_k)$$

- Solve for the trajectory $y(t)$ at a set of times (grid points) $t_0, t_1, t_2, \ldots$

$$y_{k+1} = y_k + f(t_k, y_k)(t_{k+1} - t_k)$$

Euler’s Method

- Refer to pages 10-12 of the handout for Euler’s Method for worked-out examples.

Newton-Raphson Method

- Refer to Textbook 6.2 for details
- Given the linear approximation

$$f(x) \approx f(x_k) + \frac{\partial f(x_k)}{\partial x}(x_{k+1} - x_k)$$

- Derive standard form of Newton-Raphson Algorithm

$$x_{k+1} = x_k - \left(\frac{\partial f(x_k)}{\partial x}\right)^{-1} f(x_k)$$

Newton-Raphson Method

- Refer to Textbook 6.2 for details
- Given the linear approximation

$$f(x) \approx f(x_k) + \frac{\partial f(x_k)}{\partial x}(x_{k+1} - x_k)$$

- Derive standard form of Newton-Raphson Algorithm

$$x_{k+1} = x_k - \left(\frac{\partial f(x_k)}{\partial x}\right)^{-1} f(x_k)$$

Inverse of the Jacobian has numerical issues when the Jacobian is (near) singular

Convergence

- How do we know that Newton-Raphson converges quickly? If at all?
    - Quadratic convergence, Contraction Mapping Theorem

Extra Content

- Not on exam

General Optimization

- Maximize/minimize objective/cost function
- Subject to constraint functions
- **Trajectory optimization:**
    - Applications in motion planning
        - Find a walking trajectory that does not fall

Convexity

- Convex sets and convex functions
- Convex optimization problems are “nice”
  - Local minimum is global minimum

Quadratic Programs

Useful Fact about QPs

We consider the QP

$$x^* = \underset{x \in \mathbb{R}^m}{\arg\min} \ \frac{1}{2} x^\top Q x + q x \tag{7.8}$$
$$A_{in} x \preceq b_{in}$$
$$A_{eq} x = b_{eq}$$
$$lb \preceq x \preceq ub$$

and assume that Q is symmetric ($Q^\top = Q$) and **positive definite** ($x \neq 0 \implies x^\top Q x > 0$), and that the subset of $\mathbb{R}^m$ defined by the constraints is non empty, that is

$$S := \{x \in \mathbb{R}^m \mid A_{in} x \preceq b_{in},\ A_{eq} x = b_{eq},\ lb \preceq x \preceq ub\} \neq \emptyset. \tag{7.9}$$

Then x^* exists and is unique.

Linear Programs

Definition 7.14 *A **Linear Program** means to minimize a scalar-valued linear function subject to linear equality and inequality constraints. For $x \in \mathbb{R}^n$, and $f \in \mathbb{R}^n$*

$$\begin{aligned} \textit{minimize} \quad & f^\top x \\ \textit{subject to} \quad & A_{in} x \preceq b_{in} \\ & A_{eq} x = b_{eq} \end{aligned}$$

where $A_{in} x \preceq b_{in}$ means each row of $A_{in} x$ is less than or equal to the corresponding row of b_{in}. The only restrictions on A_{in} and A_{eq} are that the set

$$K = \{x \in \mathbb{R}^n \mid A_{in} x \preceq b_{in}, \; A_{eq} x = b_{eq}\}$$

should be non-empty.

Example: Linear Program for 1-norm

Linear Program for ℓ_1-norm: $||x||_1 = \sum_{i=1}^{n} |x_i|$

Suppose that A is an $m \times n$ real matrix. Minimize $||Ax - b||_1$ is equivalent to the following linear program on $\mathbb{R}^{n+m}$

$$\begin{aligned} &\text{minimize} \quad f^\top X \\ &\text{subject to} \quad A_{in} X \preceq b_{in} \end{aligned} \tag{7.10}$$

with $X = \begin{bmatrix} x \\ s \end{bmatrix}$ ($s \in \mathbb{R}^m$ are called slack variables)

$$f := \begin{bmatrix} 0_{1\times n} & \mathbf{1}_{1\times m} \end{bmatrix}, \ A_{in} := \begin{bmatrix} A & -I_{m\times m} \\ -A & -I_{m\times m} \end{bmatrix} \text{ and } b_{in} := \begin{bmatrix} b \\ -b \end{bmatrix}$$

If $\widehat{X} = [\widehat{x}^\top, \ \widehat{s}^\top]^\top$ is the solution of the linear programming problem, then $\widehat{x}$ solves the 1-norm optimization problem; that is

$$\widehat{x} \in \arg \min_{x \in \mathbb{R}^n} ||Ax - b||_1.$$

Example: Linear Program for Max-Norm

Linear Program for ℓ_∞-norm: $||x||_\infty = \max_{1\leq i\leq n}|x_i|$

Suppose that A is an $m\times n$ real matrix. Minimize $||Ax-b||_\infty$ is equivalent to the following linear program on $\mathbb{R}^{n+1}$

$$\begin{array}{rl} \text{minimize} & f^\top X \\ \text{subject to} & A_{in}X \preceq b_{in} \end{array}$$

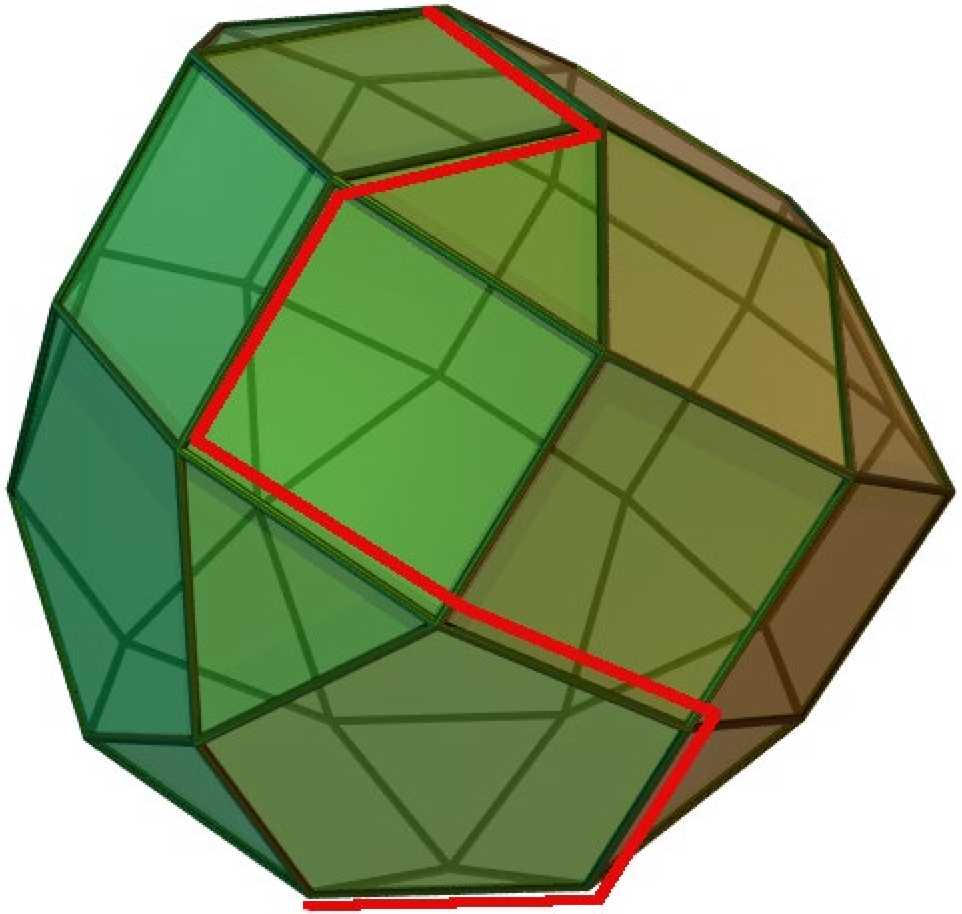
with $X = \begin{bmatrix} x \\ s \end{bmatrix}$ ($s \in \mathbb{R}$ is called a slack variable)

$$f := \begin{bmatrix} 0_{1\times n} & 1 \end{bmatrix},\ A_{in} := \begin{bmatrix} A & -\mathbf{1}_{m\times 1} \\ -A & -\mathbf{1}_{m\times 1} \end{bmatrix} \text{ and } b_{in} := \begin{bmatrix} b \\ -b \end{bmatrix}$$

If $\widehat{X} = [\widehat{x}^\top,\ \widehat{s}\,]^\top$ solves the linear programming problem, then $\widehat{x}$ solves the max-norm optimization problem; that is

$$\widehat{x} \in \arg\min_{x\in\mathbb{R}^n} ||Ax-b||_\infty.$$

Simplex Algorithm



The End