

ECS-230 App Num Linear Algebra

Notes By: Shuxian Zhang

Lecture 2 - Vector Spaces	6
Convention	6
Vector	6
Vector operations	6
Field	7
Subspaces	7
Span	7
Spanning Sets	8
Linear Independence	8
Basis	8
Dimension	8
Lecture 3 - Linear Transformations and Matrices	9
Linear Transformation	9
Image and Kernel (Let T be a linear transformation)	9
Representing Linear Transformation	9
Interpretation of Matrix-Vector Multiplication	10
Lecture 4 - Matrix	10
Matrix Multiplication	10
Range and Nullspace	11
Matrix Rank	11
Matrix Inversion	12
Matrix Operation	12
Lecture 5 - Block Matrices and Determinants	12
Block Matrices	12
Determinants	13
Lecture 6 - Inner Product	15
Geometric Dot Product	15
Geometric Dot Product in a Basis	15

General Form of Inner Product	16
Decomposition into Components using the Inner Product	17
Unitary Matrices	17
Rank-One Matrices	18
Inner Product Properties	18
 Lecture 7 - Norm	18
Norms	18
p-norms	18
Weighted Norms	19
Induced Norms	19
 Lecture 8 & 9 - Singular Value Decomposition	20
Induction	20
SVD	21
SVD's Power	21
Proof of Existence & Uniqueness of SVD	22
SVD vs. Eigendecomposition	22
 Lecture 10 - Conditioning of Computational Problems	23
Generic Computation Problem	23
Conditioning	23
Relative and Absolute Error Measures	23
Absolute Condition Numbers	24
Relative Condition Numbers	24
III-Conditioned Problems	25
 Lecture 11 - Matrix Condition Numbers	25
Condition Number of a Matrix	25
 Lecture 12 & 13 - Floating Point	26
Floating Point Representation	26
Unit in the last place and Machine	26
IEEE 754 Floating Point: Single Precision & Double Precision	27
Fundamental Axiom of Floating Point Arithmetic	27

Floating Point Peculiarities	28
Error Propagation	28
Floating Point “Exceptions”	30
IEEE 754 Rounding Modes	30
Lecture 14 - C++ Types	31
Types in C++	31
Type Conversions in C++	31
Casting in C++	31
Lecture 15 - Stability	32
Accuracy of Algorithms	32
Stability	32
Backward Stability	33
Meaning of	33
What Norm?	34
Stability Summary	34
Accuracy vs. Backward Stability	34
Lecture 16 - Triangular Linear Systems	36
Triangular Linear Systems	36
Forward / Backward Substitution	36
Lecture 16 - LU Decomposition	37
LU Decomposition	37
Gaussian Elimination	37
Lecture 17 - LU Decomposition with Pivoting	38
Pivots	38
Choosing Pivots: Complete Pivoting	39
Choosing Pivots: Partial Pivoting	39
Permutation Matrix in Partial Pivoting	39
Partial Pivoting Example	40
Lecture 18 - Stability of LU	41
Without Pivoting	41

Complete Pivoting	41
Partial Pivoting	42
Lecture 19 - Cholesky Factorization	44
Motivation	44
Cholesky Factorization	44
Hermitian Positive Definite Matrices	44
Symmetric Gaussian Elimination	45
Cholesky Factorization	45
Factorization	46
Lecture 20 - Least Squares	46
Least Squares	46
Solution: Cholesky Factorization	47
Other Approaches	48
Lecture 21 - QR Factorization	49
Motivation	49
QR Factorization	49
Reduced vs. Full QR	50
Gram-Schmidt	50
Existence and Uniqueness	51
Solving using QR Factorization	51
Lecture 22 - QR Factorization Pt2	51
Projector Matrices	51
Complementary Projectors	52
Orthogonal Projectors	52
QR Factorization in Projector Form	52
Lecture 23 - QR Householder Triangularization	53
Lecture 24 - Eigenvalues and Eigenvectors	54
Eigenvalue and Eigenvector	54
Eigenvalue Decomposition	54
Eigenvector Basis	54

Characteristic Polynomial	55
Algebraic Multiplicity	55
Eigenspaces and Geometric Multiplicity	55
Similarity Transformation	56
Diagonalizability	56
Determinant and Trace	56
Unitary Diagonalizability	56
Schur Factorization	57
Lecture 25~26 - Eigenvalue Solvers	57
Eigenvalue Algorithms	57
Iterative Eigenvalue Solvers	58
Phase 1: Reduce to Hessenberg matrix	59
Phase 2: Power Iteration	59
Inverse Iteration & Rayleigh Quotient Iteration	60
Lecture 27 - QR Algorithm for Eigenvalues	61
QR Algorithm	61
Accelerating the QR Algorithm (Inverse & Rayleigh Quotient Iteration)	62
Review	63
Factorization	63
Solving Linear System	65
Solving Least Squares	65
Lemma	67
Triangular Matrix	67
Orthogonal Square Matrix	67
Rank	67
Determinant	67
Inverse Matrix	68
Positive Definite	68

Lecture 2 - Vector Spaces

Convention

- Vectors
 - denoted as bold letters
 - All column vector (row vectors are expressed as \mathbf{x}^T)
- 1-based indexing
- Usually suppress *: $\alpha * \mathbf{v} = \alpha \mathbf{v}$

Vector

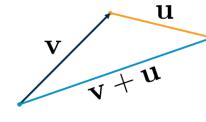
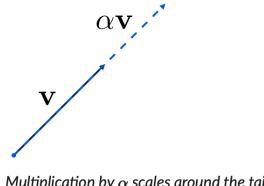
- Geometric/spatial vectors: oriented line segments defined by a direction and magnitude
- N-tuples (arrays) of real numbers, $\mathbf{x} \in R^n$
- There is a $\mathbf{0}$ vector such that $\mathbf{v} + \mathbf{0} = \mathbf{v}$
- There is a $-\mathbf{v}$ vector such that $\mathbf{v} + (-\mathbf{v}) = \mathbf{0}$
- $1\mathbf{v} = \mathbf{v}$

Vector operations

- Scale them by factor α
 - $\alpha(\mathbf{v} + \mathbf{u}) = \alpha \mathbf{v} + \alpha \mathbf{u}$
 - Associative: $(\alpha\beta)\mathbf{v} = \alpha(\beta\mathbf{v})$
- Addition
 - Commutative: $\mathbf{v} + \mathbf{u} = \mathbf{u} + \mathbf{v}$
 - Associative:

$$(\mathbf{v} + \mathbf{u}) + \mathbf{w} = \mathbf{v} + (\mathbf{u} + \mathbf{w})$$

- Geometric vectors



- n-tuples scale and add component-wise:

$$\alpha \mathbf{x} = \begin{bmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_n \end{bmatrix}, \quad \mathbf{x} + \mathbf{y} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix}$$

A *vector space* over a *field* \mathbb{F} is a set of vectors V together with two binary operations, $+ : V \times V \rightarrow V$ and $* : \mathbb{F} \times V \rightarrow V$, satisfying the *axioms*:

1. Addition is commutative: $\mathbf{v} + \mathbf{u} = \mathbf{u} + \mathbf{v} \quad \forall \mathbf{u}, \mathbf{v} \in V$
2. Addition is associative: $(\mathbf{v} + \mathbf{u}) + \mathbf{w} = \mathbf{v} + (\mathbf{u} + \mathbf{w}) \quad \forall \mathbf{u}, \mathbf{v}, \mathbf{w} \in V$
3. Scaling "distributes over" addition:
 - $\alpha * (\mathbf{v} + \mathbf{u}) = \alpha * \mathbf{v} + \alpha * \mathbf{u} \quad \forall \alpha \in \mathbb{F} \quad \forall \mathbf{u}, \mathbf{v} \in V$
 - $(\alpha + \beta) * \mathbf{v} = \alpha * \mathbf{v} + \beta * \mathbf{v} \quad \forall \alpha, \beta \in \mathbb{F} \quad \forall \mathbf{v} \in V$
4. $(\alpha\beta) * \mathbf{v} = \alpha(\beta * \mathbf{v}) \quad \forall \alpha, \beta \in \mathbb{F} \quad \forall \mathbf{v} \in V$
5. Zero vector: $\exists \mathbf{0} \in V$ such that $\mathbf{v} + \mathbf{0} = \mathbf{v} \quad \forall \mathbf{v} \in V$
6. Vector negation: $\forall \mathbf{v} \in V, \exists (-\mathbf{v}) \in V$ such that $\mathbf{v} + (-\mathbf{v}) = \mathbf{0}$
7. $1\mathbf{v} = \mathbf{v} \quad \forall \mathbf{v} \in V$, where 1 is the multiplicative identity of \mathbb{F}

Field \mathbb{F}

- Informally: a set of scalars obeying the familiar properties of \mathbb{R} .

A field is a set \mathbb{F} along with binary operations $+, * : \mathbb{F} \times \mathbb{F} \rightarrow \mathbb{F}$ satisfying the *axioms*:

1. Addition is commutative: $\alpha + \beta = \beta + \alpha \quad \forall \alpha, \beta \in \mathbb{F}$
 2. Addition is associative: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma \quad \forall \alpha, \beta, \gamma \in \mathbb{F}$
 3. Additive identity: $\exists 0 \in \mathbb{F}$ such that $\alpha + 0 = \alpha \quad \forall \alpha \in \mathbb{F}$
 4. Additive inverse: $\forall \alpha \in \mathbb{F}, \exists (-\alpha) \in \mathbb{F}$ such that $(-\alpha) + \alpha = 0$
 5. Multiplicative is commutative: $\alpha * \beta = \beta * \alpha \quad \forall \alpha, \beta \in \mathbb{F}$
 6. Multiplicative is associative: $\alpha * (\beta * \gamma) = (\alpha * \beta) * \gamma \quad \forall \alpha, \beta, \gamma \in \mathbb{F}$
 7. Multiplicative identity: $\exists 1 \in \mathbb{F}$ such that $\alpha * 1 = \alpha \quad \forall \alpha \in \mathbb{F}$
 8. Multiplicative inverse: $\forall \alpha \neq 0 \in \mathbb{F}, \exists \alpha^{-1} \in \mathbb{F}$ such that $\alpha^{-1} * \alpha = 1$
 9. Multiplication distributes over addition: $\alpha * (\beta + \gamma) = \alpha * \beta + \alpha * \gamma \quad \forall \alpha, \beta, \gamma \in \mathbb{F}$
-

Subspaces S

- A subspace is a nonempty subset $S \subseteq V$ of a vector space V over field \mathbb{F} that satisfies the condition
 - $\alpha \mathbf{v} \in S, \forall \mathbf{v} \in S, \forall \alpha \in \mathbb{F}$
 - $\mathbf{v} + \mathbf{u} \in S, \forall \mathbf{v}, \mathbf{u} \in S$
 - In other words S is closed scalar multiplication and under addition.
 - A common and convenient way of building a subspace S is to pick some set of vectors $\tilde{S} \subseteq V$ and then compute their closure under $+$ and $*$.
 - In other words, for every $\alpha \in \mathbb{F}$ and $\mathbf{v}, \mathbf{u} \in \tilde{S}$, include $\alpha \mathbf{v}$ and $\mathbf{v} + \mathbf{u}$ in S .
 - At the end of this (infinite) process, S is guaranteed to be a subspace by definition.
-

Span

- Computing the span of the vectors in \tilde{S}
 - Given $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\} \subseteq V$, we can form **linear combinations** $\alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \dots + \alpha_n \mathbf{v}_n$ for $\alpha_i \in \mathbb{F}$ using the vector space operations.
 - The set of all linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ is called the span of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ and is denoted: $\text{span}(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n)$
- Theorem: $S := \text{span}(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n)$ is a subspace of V .
- Intuition: all linear combinations of vectors

Spanning Sets

- The set $\{v_1, v_2, \dots, v_n\} \subseteq V$ is called a spanning set if $\text{span}(v_1, v_2, \dots, v_n) = V$. (i.e. every vector in V can be written as a linear combination of v_1, v_2, \dots, v_n)

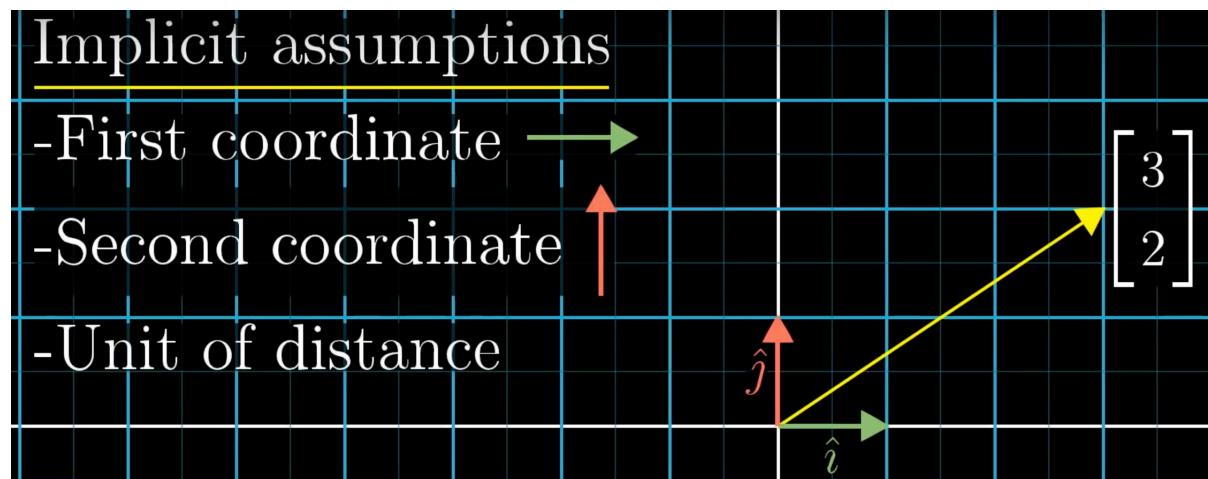
Linear Independence

- Vector v_1, v_2, \dots, v_n are called linearly independent if $c_1v_1 + c_2v_2 + \dots + c_nv_n = \mathbf{0}$ implies that $c_1 = \dots = c_n = 0$.
- On the contrary, if there exists scalars c_1, \dots, c_n not all zero such that $c_1v_1 + c_2v_2 + \dots + c_nv_n = \mathbf{0}$, we call the vectors linearly dependent.
 - In this case, at least one vector is redundant and we can form a smaller spanning set.
 - At least one vector can be a linear transformation of other vectors

Basis

- A set of vectors $\{v_1, v_2, \dots, v_n\}$ that are linearly independent and span the full vector space V , (i.e. $\text{span}(v_1, v_2, \dots, v_n) = V$), are called a **basis** for V .
- Bases are a central concept in linear algebra
 - For a given vector space V there are usually an infinite number of bases to choose from
 - For many problems, the key solution step is transforming into the right basis
- Given a basis $\{v_1, v_2, \dots, v_n\}$ for V , there is a unique way to write any $\tilde{v} \in V$ as:

$$\tilde{v} = \alpha_1v_1 + \alpha_2v_2 + \dots + \alpha_nv_n$$



Dimension

- If a vector space V has a basis consisting of n vectors, then V is said to have **dimension n**.

Lecture 3 - Linear Transformations and Matrices

Linear Transformation

- Intuition: <https://www.youtube.com/watch?v=kYB8IZa5AuE>
- Def: a mapping from vector space (V, \mathbb{F}) to vector space (W, \mathbb{F}) such that $L(\alpha \mathbf{v} + \beta \mathbf{u}) = \alpha L(\mathbf{v}) + \beta L(\mathbf{u}), \forall \alpha, \beta \in \mathbb{F}, \forall \mathbf{v}, \mathbf{u} \in V$
- Some basic properties of a linear map $L : V \rightarrow W$
 - $L(\mathbf{0}_V) = \mathbf{0}_W$ (the zero vector of V maps to the zero vector of W)
 - $L(c_1 \mathbf{v}_1 + c_2 \mathbf{v}_2 + \dots + c_n \mathbf{v}_n) = c_1 L(\mathbf{v}_1) + c_2 L(\mathbf{v}_2) + \dots + c_n L(\mathbf{v}_n)$

Image and Kernel (Let $L : V \rightarrow W$ be a linear transformation)

In the following definitions, let $\mathcal{L} : V \rightarrow W$ be a linear transformation.

The **image** of a subspace $S \subseteq V$ under \mathcal{L} is:

$$\mathcal{L}(S) := \{\mathcal{L}(\mathbf{v}) | \mathbf{v} \in S\}$$

All vectors in W that can be produced by applying \mathcal{L} to a $\mathbf{v} \in S$

The **image** of \mathcal{L} is just the image of V under \mathcal{L} :

$$\text{Im}(\mathcal{L}) := \mathcal{L}(V) = \{\mathcal{L}(\mathbf{v}) | \mathbf{v} \in V\}.$$

The **nullspace** or **kernel** of \mathcal{L} is the set of vectors mapping to $\mathbf{0}_W$:

$$\mathcal{N}(\mathcal{L}) := \{\mathbf{v} \in V | \mathcal{L}(\mathbf{v}) = \mathbf{0}_W\}.$$

Representing Linear Transformation

- Evaluating the coordinates of $\mathcal{L}(\mathbf{v})$ in basis C for an arbitrary \mathbf{v} :

$$\begin{aligned} [\mathcal{L}(\mathbf{v})]_C &= [\mathcal{L}(x_1 \mathbf{v}_1 + \dots + x_n \mathbf{v}_n)]_C \\ &= \sum_{j=1}^n x_j [\mathcal{L}(\mathbf{v}_j)]_C = \sum_{j=1}^n \begin{bmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{bmatrix} x_j \\ &= \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \\ &:= A[\mathbf{v}]_B, \end{aligned}$$

◦ $A \in \mathbb{F}^{m \times n}$ is the **matrix** representation of \mathcal{L} relative to the bases B and C .

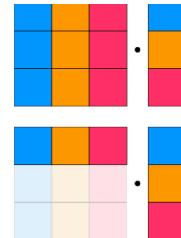
◦ We have **defined** matrix-vector multiplication $\mathbf{y} = Ax$ to be the operation:

$$y_i = \sum_{j=1}^n a_{ij} x_j \quad 1 \leq i \leq m$$

Interpretation of Matrix-Vector Multiplication

- $\mathbf{a} = A\mathbf{b}$, assume $\hat{\mathbf{e}}_1, \dots, \hat{\mathbf{e}}_n$ is the basis of vector space of \mathbf{b} , then each column of A is where the transformed vectors of $\hat{\mathbf{e}}_1, \dots, \hat{\mathbf{e}}_n$ landed in the vector space of \mathbf{a} (i.e. forming a basis)

- A linear combination of the columns of A
 - This is why we can use matrix multiplication to represent linear maps!
 - It's also the *most useful interpretation* in this course.
- A "dot product" of each row of A with \mathbf{x} to produce an entry of \mathbf{y} .
 - Each row represents a scalar-valued linear function of \mathbf{x} .
 - We'll discuss dot products later this week.



$$\begin{array}{ccc} \mathbf{v} \in V & \xrightarrow{\mathcal{L}} & \mathbf{w} = \mathcal{L}(\mathbf{v}) \in W \\ \downarrow & & \downarrow \\ \mathbf{x} = [\mathbf{v}]_B \in \mathbb{F}^n & \xrightarrow{A} & A\mathbf{x} = [\mathbf{w}]_C \in \mathbb{F}^m \end{array}$$

Lecture 4 - Matrix

Matrix Multiplication

- Let $A \in \mathbb{F}^{p \times m}$ and $B \in \mathbb{F}^{m \times n}$. The matrix multiplication operation $C = AB$ is defined as:
- $$c_{ij} = \sum_{k=1}^m a_{ik} b_{kj}$$
- The matrix dimension should match!
 - Each column of C is a linear combination of the columns of A

- Each row of C is a linear combination of the rows of B
- Associative: $A(BC) = (AB)C$

$$\begin{pmatrix}
 a_{11} & a_{12} & a_{13} \\
 a_{21} & a_{22} & a_{23} \\
 a_{31} & a_{32} & a_{33}
 \end{pmatrix}
 \begin{pmatrix}
 b_{11} & b_{12} \\
 b_{21} & b_{22} \\
 b_{31} & b_{32}
 \end{pmatrix}
 = \begin{pmatrix}
 c_{11} & c_{12} \\
 c_{21} & c_{22} \\
 c_{31} & c_{32}
 \end{pmatrix}$$

- Not Commutative: $AB \neq BA$
- Interpretation: $\mathbf{u} = AB\mathbf{v} = C\mathbf{v}$ ($C = AB$), matrix composition is the composition of linear transformation, i.e. the effect of first applying linear transformation A, then applying linear transformation B is the same as applying linear transformation C.

Range and Nullspace

- The range or column space of $A \in \mathbb{F}^{m \times n}$ is the image of \mathbb{F}^n under A
 - $\text{range}(A) := \{Ax \mid x \in \mathbb{F}^n\}$, i.e. the set of all possible outputs of Ax
 - Given our interpretation of matrix-vector multiplication as a linear combination of the columns, it is clear that $\text{range}(A) = \text{span}(a_1, \dots, a_n)$ where a_j are the columns of A.
(Column space = the span of columns)
- The nullspace of $A \in \mathbb{F}^{m \times n}$ is the set of vectors mapping to origin $\mathbf{0} \in \mathbb{F}^m$
 - $\mathbb{N}(A) := \{x \in \mathbb{F}^n \mid Ax = \mathbf{0}\}$
 - Another interpretation: the set of vectors *orthogonal* to each row of A

Matrix Rank

- The **column rank** of $A \in \mathbb{F}^{m \times n}$ is the dimension of the column space $\text{range}(A)$
- The **row rank** of $A \in \mathbb{F}^{m \times n}$ is the dimension of the row space (the span of A's rows)
 - Column and row ranks always equal, so we can simply refer to the rank of a matrix
- **Full rank:** Matrix $A \in \mathbb{F}^{m \times n}$ is full rank if it has the greatest possible rank, $\min(m, n)$
 - A **full-rank** matrix $A \in \mathbb{F}^{m \times n}$ with $m \geq n$ must have n linearly independent columns
 - Theorem: a matrix $A \in \mathbb{F}^{m \times n}$ with $m \geq n$ has **full rank** n iff $Ax \neq Ay, \forall x \neq y \in \mathbb{F}^n$
(no distinct vectors get mapped to the same vector)
- Intuition: rank = the number of dimensions in the output / column space
 - Rank 1: the transformation becomes a line

- Rank2: the transformation becomes a plane

Matrix Inversion

- A^{-1} is called the inverse of A , it inverses the transformation of A
- A **nonsingular** or **invertible** matrix is a square matrix $A \in \mathbb{F}^{m \times m}$ of **full rank**
- A^{-1} is both a left and right inverse: $AA^{-1} = A^{-1}A = I$
- **Theorem:** the following conditions for $A \in \mathbb{F}^{m \times m}$ are equivalent
 - A has an inverse A^{-1}
 - A is a nonsingular matrix
 - A is of full rank m
 - $\text{range}(A) = \mathbb{F}^m$
 - $\mathbb{N}(A) = \{\mathbf{0}\}$
 - 0 is not an eigenvalue of A
 - 0 is not a singular value of A
 - $\det(A) \neq 0$ (if $\det(A)=0$, then the transformation squishes into lower dimension)
- Interpretation of Linear Systems: see slides 04-11

Matrix Operation

- Multiplication $C = AB$
- Inversion $B = A^{-1}$
- Addition $C = A + B$
- Scalar multiplication $B = \alpha A$
- Transposition $B = A^T$

◦ Scalar multiplication

$$\alpha \begin{bmatrix} A_{11} & A_{12} & \dots \\ A_{21} & A_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} = \begin{bmatrix} \alpha A_{11} & \alpha A_{12} & \dots \\ \alpha A_{21} & \alpha A_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

◦ Addition (assuming compatible block sizes)

$$\begin{bmatrix} A_{11} & A_{12} & \dots \\ A_{21} & A_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} + \begin{bmatrix} B_{11} & B_{12} & \dots \\ B_{21} & B_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} = \begin{bmatrix} A_{11} + B_{11} & A_{12} + B_{12} & \dots \\ A_{21} + B_{21} & A_{22} + B_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

◦ Transposition

$$\begin{bmatrix} A_{11} & A_{12} & \dots \\ A_{21} & A_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}^T = \begin{bmatrix} A_{11}^T & A_{21}^T & \dots \\ A_{12}^T & A_{22}^T & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

Lecture 5 - Block Matrices and Determinants

Block Matrices

- Decompose A into a block matrix, the following operations respect the block structure
 - Scalar multiplication
 - Addition
 - Transposition

- Matrix multiplication

Determinants $A \in \mathbb{F}^{m \times m}$

- The determinant of a square matrix, , measuring the “volume change” produced by the corresponding linear map.
- Intuition: change in volume of the unit hypercube when it is transformed by A
- Three axioms

- **Axiom1:** $\det(A)$ is a multilinear function of the columns of A

► Intuition: in 3D, scale the length by α then add it by b , the volume changes to $\alpha V + V_b$

$$\det \left(\begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_j & \cdots & \mathbf{a}_m \end{bmatrix} \right) = \alpha \det \left(\begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_j & \cdots & \mathbf{a}_m \end{bmatrix} \right) + \det \left(\begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{b} & \cdots & \mathbf{a}_m \end{bmatrix} \right)$$

- **Axiom2:** $\det(A)$ vanishes if any columns are repeated

► Intuition: in 3D, when a dimension collapse, the cube falls into a plane.

$$\mathbf{a}_j = \mathbf{a}_k \text{ for any } j \neq k \implies \det \left(\begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_m \end{bmatrix} \right) = 0$$

- **Axiom3:** $\det(I)$, the determinant of the identity matrix, is 1.

- Properties

- **Property1:** Adding a multiple of another column doesn't change the determinant

$$\det(e_1, \dots, e_k + \alpha e_j, \dots, e_m) = \det(e_1, \dots, e_k, \dots, e_m) + \underbrace{\alpha \det(e_1, \dots, e_j, \dots, e_j, \dots, e_m)}_{=0}$$

- **Property2:** Swapping two adjacent columns negates

$$\begin{aligned} 0 &= \det(\mathbf{a}_1, \dots, (\mathbf{a}_j + \mathbf{a}_{j+1}), (\mathbf{a}_j + \mathbf{a}_{j+1}), \dots, \mathbf{a}_m) \\ &= \underbrace{\det(\mathbf{a}_1, \dots, \mathbf{a}_j, \mathbf{a}_j, \dots, \mathbf{a}_m)}_0 + \det(\mathbf{a}_1, \dots, \mathbf{a}_j, \mathbf{a}_{j+1}, \dots, \mathbf{a}_m) + \det(\mathbf{a}_1, \dots, \mathbf{a}_{j+1}, \mathbf{a}_j, \dots, \mathbf{a}_m) + \underbrace{\det(\mathbf{a}_1, \dots, \mathbf{a}_{j+1}, \mathbf{a}_{j+1}, \dots, \mathbf{a}_m)}_0 \\ &\implies \det(\mathbf{a}_1, \dots, \mathbf{a}_j, \mathbf{a}_{j+1}, \dots, \mathbf{a}_m) = -\det(\mathbf{a}_1, \dots, \mathbf{a}_{j+1}, \mathbf{a}_j, \dots, \mathbf{a}_m) \end{aligned}$$

- **Property3:** $\det(A) = 0$ if A's columns are linearly dependent

$$\det(e_1, \dots, e_{m-1}, \sum_{i=1}^{m-1} c_i e_i) = \sum_{i=1}^{m-1} c_i \det(e_1, \dots, e_{m-1}, e_i) = 0$$

$$\det \left(\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{mm} \end{bmatrix} \right) = (a_{11}a_{22} \cdots a_{mm}) \det \left(\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \right) = \prod_{i=1}^m a_{ii}$$

- Property4: $\det(AB) = \det(A)\det(B)$

- Property5: $\det(A) = \det(A^T)$

- Computing using the Axioms

- Diagonal matrix (using Axiom1)

- Upper/lower triangular (using Property1)

$$\det \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ 0 & a_{22} & a_{23} & \dots & a_{2m} \\ 0 & 0 & a_{33} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & a_{mm} \end{pmatrix} = a_{11} \det \begin{pmatrix} 1 & a_{12} & a_{13} & \dots & a_{1m} \\ 0 & a_{22} & a_{23} & \dots & a_{2m} \\ 0 & 0 & a_{33} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & a_{mm} \end{pmatrix} = a_{11} \det \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & a_{22} & a_{23} & \dots & a_{2m} \\ 0 & 0 & a_{33} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & a_{mm} \end{pmatrix}$$

You can do the same procedure for upper-triangular matrices, just "bottom-up."

$$= a_{11} a_{22} \det \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & a_{23} & \dots & a_{2m} \\ 0 & 0 & a_{33} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & a_{mm} \end{pmatrix} = \dots = \prod_{i=1}^m a_{ii}$$

9

- Calculate: Leibniz Formula

Decomposing the columns of $\det(A)$ in the standard basis $\{\mathbf{e}_i\}$: $\mathbf{a}_j = \sum_{i=1}^m a_{ij} \mathbf{e}_i$
then apply linearity one column at a time:

$$\begin{aligned} \det(A) &= \det \left(\sum_{i_1=1}^m a_{i_1 1} \mathbf{e}_{i_1}, \sum_{i_2=1}^m a_{i_2 2} \mathbf{e}_{i_2}, \dots, \sum_{i_m=1}^m a_{i_m m} \mathbf{e}_{i_m} \right) \\ &= \sum_{i_1=1}^m a_{i_1 1} \det \left(\mathbf{e}_{i_1}, \sum_{i_2=1}^m a_{i_2 2} \mathbf{e}_{i_2}, \dots, \sum_{i_m=1}^m a_{i_m m} \mathbf{e}_{i_m} \right) \\ &= \sum_{i_1=1}^m a_{i_1 1} \sum_{i_2=1}^m a_{i_2 2} \cdots \sum_{i_m=1}^m a_{i_m m} \det(\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \dots, \mathbf{e}_{i_m}) \\ &= \sum_{i_1, \dots, i_m=1}^m \left(\prod_{j=1}^m a_{i_j j} \right) \det(\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \dots, \mathbf{e}_{i_m}) \\ &= \sum_{i_1, \dots, i_m=1}^m \det(\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \dots, \mathbf{e}_{i_m}) \prod_{j=1}^m a_{i_j j} \end{aligned}$$

Lecture 6 - Inner Product

Geometric Dot Product

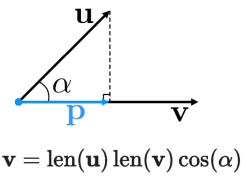
- $\mathbf{u} \cdot \mathbf{v} = \text{len}(\mathbf{u})\text{len}(\mathbf{v})\cos(\alpha)$
- Contains everything we need to measure length, angle, and projections

$$\text{- } \text{len}(\mathbf{u}) = \sqrt{\mathbf{u} \cdot \mathbf{u}}, \alpha = \cos^{-1}\left(\frac{\mathbf{u} \cdot \mathbf{v}}{\text{len}(\mathbf{u})\text{len}(\mathbf{v})}\right), \mathbf{p} = \frac{\mathbf{u} \cdot \mathbf{v}}{\text{len}(\mathbf{v})^2}\mathbf{v}$$

- Properties

- Commutative: Symmetric with respect to \mathbf{u} and \mathbf{v}

- Linear in each argument: $(\alpha\mathbf{u} + \mathbf{w}) \cdot \mathbf{v} = \alpha\mathbf{u} \cdot \mathbf{v} + \mathbf{w} \cdot \mathbf{v}$



Geometric Dot Product in a Basis

Consider a basis $\{\mathbf{b}_1, \dots, \mathbf{b}_m\}$ for the m -dimensional geometric vectors. Then we can find the scalar components v_i and u_j in \mathbb{R} :

$$\begin{aligned} \mathbf{v} &= \sum_i v_i \mathbf{b}_i, \quad \mathbf{u} = \sum_j u_j \mathbf{b}_j \\ \implies \mathbf{v} \cdot \mathbf{u} &= \left(\sum_{i=1}^m v_i \mathbf{b}_i \right) \cdot \left(\sum_{j=1}^m u_j \mathbf{b}_j \right) \\ &= \sum_{i=1}^m v_i \sum_{j=1}^m (\mathbf{b}_i \cdot \mathbf{b}_j) u_j \quad \text{According to linearity} \\ &= [v_1 \ \dots \ v_m] B \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \quad \text{where } B = \begin{bmatrix} (\mathbf{b}_1 \cdot \mathbf{b}_1) & \dots & (\mathbf{b}_1 \cdot \mathbf{b}_m) \\ \vdots & & \vdots \\ (\mathbf{b}_m \cdot \mathbf{b}_1) & \dots & (\mathbf{b}_m \cdot \mathbf{b}_m) \end{bmatrix} \end{aligned}$$

- B is a symmetric positive definitive matrix.
- Property 1 of B : $B^T = B$ because $b_{ij} = b_i \cdot b_j = b_j \cdot b_i = b_{ji}$
- Property 2 of B : Any nonzero $\mathbf{c} \in \mathbb{R}^m$ corresponds to a nonzero geometric vector \mathbf{v} in the basis $\{\mathbf{b}_j\}$. So $\mathbf{c}^T B \mathbf{c} = \text{len}(\mathbf{v})^2 > 0$
- It turns out that, for $x, y \in \mathbb{R}^m$ the expression $\mathbf{x}^T B \mathbf{y}$ corresponds to a “geometric dot product” iff B is a symmetric positive definitive matrix.
- Very special case (orthonormal case)
 - We call two geometric vector orthogonal if $\mathbf{u} \cdot \mathbf{v} = 0$ which means $\alpha = 90$ degree
 - If the basis vectors are all unit length, $b_i \cdot b_i = 1$, and mutually orthogonal $b_i \cdot b_j = 0, \forall i \neq j$ and $B = I$, so we have $\mathbf{x}^T B \mathbf{y} = \mathbf{x}^T I \mathbf{y} = \mathbf{x}^T \mathbf{y} = \sum_{i=1}^m x_i y_i$

- ▷ The standard dot product

General Form of Inner Product

- Definitions

An **inner product** on a vector space (V, \mathbb{F}) is an operation $\langle \cdot, \cdot \rangle : V \times V \rightarrow \mathbb{F}$ satisfying the following axioms:

1. Linearity in the second argument

$$\langle \mathbf{v}, \alpha \mathbf{u} + \beta \mathbf{w} \rangle = \alpha \langle \mathbf{v}, \mathbf{u} \rangle + \beta \langle \mathbf{v}, \mathbf{w} \rangle$$

WARNING: the inner product is *antilinear* in the first argument:

$$\begin{aligned} \langle \alpha \mathbf{v} + \beta \mathbf{w}, \mathbf{u} \rangle &= \overline{\langle \mathbf{u}, \alpha \mathbf{v} + \beta \mathbf{w} \rangle} = \overline{\alpha \langle \mathbf{u}, \mathbf{v} \rangle + \beta \langle \mathbf{u}, \mathbf{w} \rangle} \\ &= \overline{\alpha} \langle \mathbf{v}, \mathbf{u} \rangle + \overline{\beta} \langle \mathbf{w}, \mathbf{u} \rangle. \end{aligned}$$

2. Conjugate symmetry

$$\langle \mathbf{v}, \mathbf{u} \rangle = \overline{\langle \mathbf{u}, \mathbf{v} \rangle} \quad (\text{this is the same as symmetry } \langle \mathbf{v}, \mathbf{u} \rangle = \langle \mathbf{u}, \mathbf{v} \rangle \text{ when } \mathbb{F} = \mathbb{R})$$

Note: conventions differ on which argument is linear/antilinear.

3. Positive definiteness

$$\langle \mathbf{v}, \mathbf{v} \rangle > 0 \quad \forall \mathbf{v} \neq \mathbf{0} \quad (\text{by conjugate symmetry: } \langle \mathbf{v}, \mathbf{v} \rangle = \overline{\langle \mathbf{v}, \mathbf{v} \rangle} \in \mathbb{R})$$

- Inner Product in terms of scalar components in arbitrary basis

$$\begin{aligned} \mathbf{v} &= \sum_i v_i \mathbf{b}_i, \quad \mathbf{u} = \sum_j u_j \mathbf{b}_j \quad v_i, u_j \in \mathbb{F} \\ \implies \langle \mathbf{v}, \mathbf{u} \rangle &= \left\langle \sum_{i=1}^m v_i \mathbf{b}_i, \sum_{j=1}^m u_j \mathbf{b}_j \right\rangle = \sum_{i=1}^m \overline{v_i} \left\langle \mathbf{b}_i, \sum_{j=1}^m \mathbf{b}_j u_j \right\rangle \\ &= \sum_{i=1}^m \overline{v_i} \sum_{j=1}^m \langle \mathbf{b}_i, \mathbf{b}_j \rangle u_j \\ &= [\overline{v_1} \quad \cdots \quad \overline{v_m}] B \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \quad \text{where } B = \begin{bmatrix} \langle \mathbf{b}_1, \mathbf{b}_1 \rangle & \cdots & \langle \mathbf{b}_1, \mathbf{b}_m \rangle \\ \vdots & & \vdots \\ \langle \mathbf{b}_m, \mathbf{b}_1 \rangle & \cdots & \langle \mathbf{b}_m, \mathbf{b}_m \rangle \end{bmatrix} \end{aligned}$$

- B 's properties

- ▷ $B = \overline{B^T} := B^H$, the adjoint of B . In other word, B is **hermitian** or **self-adjoint**.
- ▷ $\forall \mathbf{v} \neq \mathbf{0}$, the corresponding coordinate vector $\mathbf{c} \in \mathbb{F}^m$ must satisfy $\langle \mathbf{v}, \mathbf{v} \rangle = \mathbf{c}^H B \mathbf{c} > 0$. So B must be a positive definite hermitian matrix.
- ▷ Any inner product can be expressed in this form, and any positive definite B defines an inner product!

- Very special case: orthonormal basis

- We call two vectors (\mathbf{u}, \mathbf{v}) **orthogonal** if $\langle \mathbf{v}, \mathbf{u} \rangle = 0$.
- What if basis vectors are unit length, $\langle \mathbf{b}_i, \mathbf{b}_i \rangle = 1$, and mutually orthogonal $\langle \mathbf{b}_i, \mathbf{b}_j \rangle = 0 \forall i \neq j$?

- $B = I$ and so: $\mathbf{x}^H B \mathbf{y} = \mathbf{x}^H I \mathbf{y} = \boxed{\mathbf{x}^H \mathbf{y} = \sum_{i=1}^m \overline{x_i} y_i}$

This is the complex dot product formula you probably know. When $\mathbb{F} = \mathbb{R}$ it is just the ordinary dot product.

- Complex Dot Product: $\mathbf{x}^H \mathbf{y} = \sum_{i=1}^m \bar{x}_i y_i$
 - Why conjugate? Since $x = a + bi, y = c + di \in \mathbb{C}, \bar{x}y = (a - bi)(c + di) = (ac + bd) + (ad - bc)i$
 - Specifically, $\langle \mathbf{x}, \mathbf{x} \rangle = \mathbf{x}^H \mathbf{x}$ computes the total squared magnitude.

Decomposition into Components using the Inner Product

- Supposing we are given a vector $\mathbf{v} \in V$ and want to find its coordinates in a basis $\{\mathbf{b}_j\}$
 - Since $\{\mathbf{b}_j\}$ form a basis, we know there exists scalar values $\{x_{ij}\}$ (**the coordinate**) such that $\mathbf{v} = \sum_{j=1}^m x_j \mathbf{b}_j = [\mathbf{b}_1 \dots \mathbf{b}_m] \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$
 - Compute each basis vector with vector \mathbf{v} , we have $\langle \mathbf{b}_j, \mathbf{v} \rangle = \langle \mathbf{b}_j, \sum_{i=1}^m x_i \mathbf{b}_i \rangle = \sum_{i=1}^m x_i \langle \mathbf{b}_j, \mathbf{b}_i \rangle \forall i$
 - If we set $\mathbf{A} = \begin{bmatrix} \langle \mathbf{b}_1, \mathbf{b}_1 \rangle & \dots & \langle \mathbf{b}_1, \mathbf{b}_m \rangle \\ \vdots & \ddots & \vdots \\ \langle \mathbf{b}_m, \mathbf{b}_1 \rangle & \dots & \langle \mathbf{b}_m, \mathbf{b}_m \rangle \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}, \mathbf{y} = \begin{bmatrix} \sum_{i=1}^m x_i \langle \mathbf{b}_1, \mathbf{b}_i \rangle \\ \vdots \\ \sum_{i=1}^m x_i \langle \mathbf{b}_m, \mathbf{b}_i \rangle \end{bmatrix} = \begin{bmatrix} \langle \mathbf{b}_1, \mathbf{v} \rangle \\ \vdots \\ \langle \mathbf{b}_m, \mathbf{v} \rangle \end{bmatrix} \implies \mathbf{Ax} = \mathbf{y}$
 - Thus, our coordinates $\mathbf{x} = \mathbf{A}^{-1} \mathbf{y}$
 - Special case: if $\{\mathbf{b}_j\}$ are orthonormal with respect to $\langle \cdot, \cdot \rangle$

$$\triangleright \mathbf{A} = I \implies \mathbf{x} = \mathbf{y} = \begin{bmatrix} \langle \mathbf{b}_1, \mathbf{v} \rangle \\ \vdots \\ \langle \mathbf{b}_m, \mathbf{v} \rangle \end{bmatrix}$$

- Inner products can directly decompose vectors into an orthonormal basis.

Unitary Matrices

- A matrix $Q \in \mathbb{C}^{m \times m}$ is called **unitary** if $Q^H Q = I$, in other words, $Q^{-1} = Q^H$
- Corresponding terminology for real matrices is:
 - A matrix $Q \in \mathbb{R}^{m \times m}$ is called **orthogonal** if $Q^T Q = I$, in other words, $Q^{-1} = Q^T$
- In both, the columns of Q are orthogonal with respect to the standard dot product.

Rank-One Matrices

- A matrix $A \in \mathbb{C}^m$ has rank one if and only if it can be written as $\mathbf{x}\mathbf{y}^H$
- A matrix $A \in \mathbb{R}^m$ has rank one if and only if it can be written as $\mathbf{x}\mathbf{y}^T$
- Note the action of a rank one matrix on a vector is in general: it takes an inner product with \mathbf{v} and uses the result to scale \mathbf{u}

$$(\mathbf{u} \otimes \mathbf{v})\mathbf{x} = (\mathbf{u}\mathbf{v}^H)\mathbf{x} = \mathbf{u}(\mathbf{v}^H\mathbf{x}) = \mathbf{u} \langle \mathbf{v}, \mathbf{x} \rangle$$

Inner Product Properties

- Pythagorean Theorem: If $\langle \mathbf{v}, \mathbf{u} \rangle = 0$, then $\underbrace{\langle \mathbf{v}, \mathbf{v} \rangle}_{a^2} + \underbrace{\langle \mathbf{u}, \mathbf{u} \rangle}_{b^2} = \underbrace{\langle \mathbf{v} + \mathbf{u}, \mathbf{v} + \mathbf{u} \rangle}_{c^2}$
- Cauchy-Schwarz inequality: $|\langle \mathbf{u}, \mathbf{v} \rangle|^2 \leq \langle \mathbf{u}, \mathbf{u} \rangle \langle \mathbf{v}, \mathbf{v} \rangle$
 - For geometric vectors, $(\mathbf{u} \cdot \mathbf{v})^2 = \text{len}^2(\mathbf{u})\text{len}^2(\mathbf{v})\cos^2(\alpha) \leq \text{len}^2(\mathbf{u})\text{len}^2(\mathbf{v})$

Lecture 7 - Norm

Norms

- A **norm** for vector space V is a function $\| \cdot \| : V \rightarrow \mathbb{R}$ satisfying the following properties:
 - Positivity: $\|\mathbf{v}\| \geq 0$, with $\|\mathbf{v}\| = 0$ if and only if $\mathbf{v} = 0$
 - Homogeneity: $\|\alpha \mathbf{v}\| = |\alpha| \|\mathbf{v}\|$
 - Triangle inequality: $\|\mathbf{v} + \mathbf{u}\| \leq \|\mathbf{v}\| + \|\mathbf{u}\|$

p-norms

- A very important family of norms for \mathbb{C}^m is called the p or l_p norms.

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^m |x_i|^p \right)^{\frac{1}{p}} \quad 1 \leq p \leq \infty$$

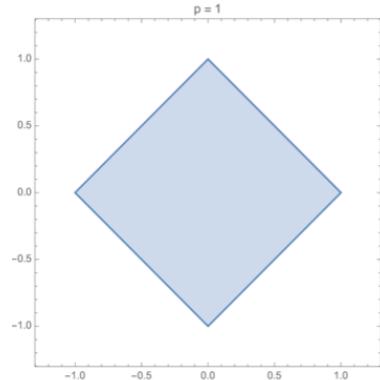
- Common cases

- “Taxicab” or “Manhattan distance”: $p = 1 \implies \|\mathbf{x}\|_1 = \sum_{i=1}^m |x_i|$

- “Euclidean length”: $p = 2 \implies \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^m |x_i|^2} = \sqrt{\mathbf{x}^H \mathbf{x}}$

- Taking the limit $p \rightarrow \infty$, the largest value dominates. $p = \infty \implies \|x\|_\infty = \max_{1 \leq i \leq m} |x_i|$
- p-norm visualizations in \mathbb{R}^2

- One way to visualize a norm is to plot its “unit ball,” the set of points within distance of the origin according to the norm.
- When $p = 1$, this is the diamond shape bounded by lines $|x| + |y| = 1$
- When $p = 2$, this is just a circular disk
- When $p = \infty$, this is the square bounded by $\max(|x|, |y|) = 1$



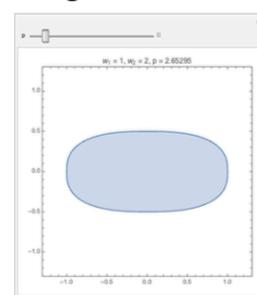
1-norm

Weighted Norms

- Given a nonsingular matrix W , we can define the norm: $\|\mathbf{x}\|_W = \|W\mathbf{x}\|_p$ using any of the p-norms for the right-hand side.
- In the special case where $W = \text{diag}(w_1, \dots, w_m)$ is a diagonal matrix and using 2-norm:

$$\|\mathbf{x}\|_W = \left(\sum_{i=1}^m |w_i x_i|^2 \right)^{\frac{1}{2}} = \sqrt{\mathbf{x}^H W^H W \mathbf{x}}$$

This is actually norm induced by the inner product whose “B” matrix is (hermitian!) $W^H W$.



- Intuition: entries of \mathbf{x} are interpreted as components in the column basis of W

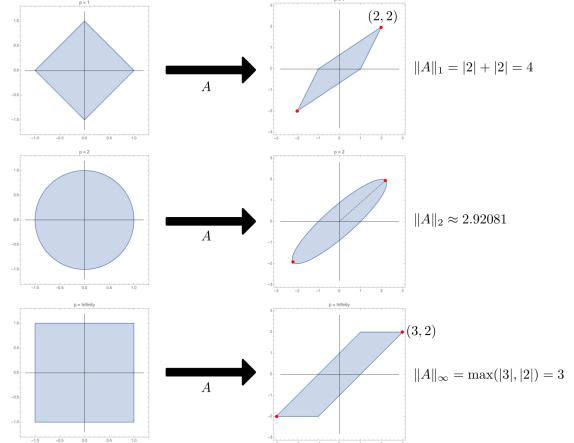
Induced Norms

- Vector norms provide a very useful way to define the norm of a matrix that respects its role as representing a linear transformation.
 - The “size” of a matrix will be the most it can stretch a vector it is applied to.
- Given a pair of vector norms $\|\cdot\|_{(n)}, \|\cdot\|_{(m)}$ for the input and output spaces of $A \in \mathbb{C}^{m \times n}$, the induced matrix norm $\|A\|_{m,n}$ is defined as:

$$\|A\|_{(m,n)} := \max_{\substack{\mathbf{x} \in \mathbb{C}^n \\ \mathbf{x} \neq \mathbf{0}}} \frac{\|A\mathbf{x}\|_{(m)}}{\|\mathbf{x}\|_{(n)}} = \max_{\substack{\mathbf{x} \in \mathbb{C}^n \\ \|\mathbf{x}\|_{(n)}=1}} \|A\mathbf{x}\|_{(m)}$$

- The second expression is most intuitive: how large can an input vector on the unit circle for $\|\cdot\|_{(n)}$ become?

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

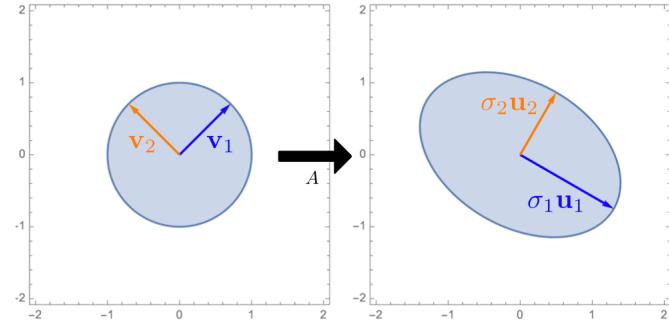


- If the input and output norms are the same (e.g. $(m)=(n)=p$), we denote the induced norm just as $\|A\|_p$
- For a diagonal matrix, $\|A\|_p = \max_i a_i$ (proof see picture in the right)

Lecture 8 & 9 - Singular Value Decomposition

Induction

- $\mathbf{u}_1, \mathbf{u}_2$ are the **unit** axis vectors for the ellipse
 - Must be orthogonal ($\mathbf{u}_1^H \mathbf{u}_2 = \mathbf{0}$)
- $\sigma_1 \mathbf{u}_1, \sigma_2 \mathbf{u}_2$ are the the ellipse's semi-major and semi-minor axes
- $\mathbf{v}_1, \mathbf{v}_2$ are the **unit** pre-images of $\sigma_1 \mathbf{u}_1, \sigma_2 \mathbf{u}_2$: $A \mathbf{v}_1 = \sigma_1 \mathbf{u}_1, A \mathbf{v}_2 = \sigma_2 \mathbf{u}_2$
 - Must be orthogonal ($\mathbf{v}_1^H \mathbf{v}_2 = \mathbf{0}$)



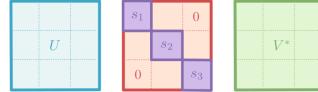
- Organizing these facts into a matrix equation:

$$\underbrace{[\mathbf{u}_1 | \mathbf{u}_2]}_U \underbrace{\begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}}_{\Sigma} = [\sigma_1 \mathbf{u}_1 | \sigma_2 \mathbf{u}_2] = A \underbrace{[\mathbf{v}_1 | \mathbf{v}_2]}_V$$
- Because both U and V are **unitary** $U^H U = V^H V = I_{2 \times 2}$ since \mathbf{u}_j and \mathbf{v}_j are orthonormal
- We have $A = U \Sigma V^H$, it implies applying A can be decomposed into following steps
 - Rotate/reflects V^H
 - Applies axis-aligned scale Σ
 - Rotate/reflects again U

SVD

- Singular values are $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$, $p=\min(m, n)$, diagonal entries in Σ

The Singular Value Decomposition



Given $A \in \mathbb{C}^{m \times n}$, a **singular value decomposition (SVD)** of A is a factorization:

$$A = U\Sigma V^H \quad \text{where} \quad \begin{cases} U \in \mathbb{C}^{m \times m} & \text{is unitary,} \\ \Sigma \in \mathbb{C}^{m \times n} & \text{is diagonal,} \\ V \in \mathbb{C}^{n \times n} & \text{is unitary.} \end{cases}$$

Following the standard convention, we assume the **nonnegative** diagonal entries σ_j of Σ are ordered $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$, where $p = \min(m, n)$.

- Note: the columns of V and U form an *orthonormal basis* for the input and output spaces \mathbb{C}^n and \mathbb{C}^m , respectively.
 - Columns of U are called "left singular vectors" u_j and columns of V are called "right singular vectors" v_j .

- The columns of V and U form an orthonormal basis for the input and output space \mathbb{C}^n and \mathbb{C}^m
 - Columns of U are called "left singular vectors" u_j
 - Columns of V are called "right singular vectors" v_j

SVD's Power

- Easily compute determinants when $m=n$

$$|det(A)| = |det(U\Sigma V^H)| = |det(U)||det(\Sigma)||det(V^H)| = 1 * |det(\Sigma)| * 1 = \sum_{i=1}^m \sigma_{ii}$$

- Easily computer inverse: $A^{-1} = (U\Sigma V^H)^{-1} = (V^H)^{-1}\Sigma^{-1}U^{-1} = V\Sigma^{-1}U^H$, with $\Sigma^{-1} = diag(1/\sigma_1, \dots, 1/\sigma_m)$

- Prove row rank=column rank

- Reveals normal bases for the **nullspace** and **range** of A

- $\text{span}(u_1, \dots, u_r) = \text{range}(A)$
- $\text{span}(v_{r+1}, \dots, v_n) = \mathcal{N}(A)$

- Reveals that row rank = column rank! Suppose the singular values are $\sigma_1 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_p = 0$.

$$A = \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_r & \mathbf{u}_{r+1} & \cdots \end{bmatrix} \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & 0 & \\ & & & & \ddots \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^H \\ \vdots \\ \mathbf{v}_r^H \\ \hline \mathbf{v}_{r+1}^H \\ \vdots \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_r \end{bmatrix}}_{\hat{U}} \underbrace{\begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & & \end{bmatrix}}_{\hat{\Sigma}} \underbrace{\begin{bmatrix} \mathbf{v}_1^H \\ \vdots \\ \mathbf{v}_r^H \end{bmatrix}}_{\hat{V}^H}$$

Because SVD lets us determine rank by counting the nonzero singular values, it is called "rank-revealing."

- *A's column rank is r*
 - A' 's columns are a linear combination of $\{\mathbf{u}_1, \dots, \mathbf{u}_r\} \implies \text{column rank} \leq r$
 - Any $\mathbf{u} \in \text{span}(\mathbf{u}_1, \dots, \mathbf{u}_r)$ can be written as $A\mathbf{x}$ with $\mathbf{x} = \hat{V}\hat{\Sigma}^{-1}\hat{U}^H\mathbf{u} \implies \text{column rank} \geq r$
- *A's row rank is r*
 - A' 's rows are a linear combination of $\{\mathbf{v}_1^H, \dots, \mathbf{v}_r^H\}, \implies \text{row rank} \leq r$
 - Any vector $\mathbf{v} \in \text{span}(\mathbf{v}_1^H, \dots, \mathbf{v}_r^H)$ can be written as $A^H\mathbf{x}$ with $\mathbf{x} = \hat{U}\hat{\Sigma}^{-1}\hat{V}^H\mathbf{v} \implies \text{row rank} \geq r$

Proof of Existence & Uniqueness of SVD

- An unique SVD exists for any $A \in \mathbb{C}^{m \times n}$ (Proof See slides 09-03)

SVD vs. Eigendecomposition

SVD vs. Eigendecomposition

$$A = \underbrace{U\Sigma V^H}_{\text{SVD}} \quad \text{vs.} \quad A = \underbrace{X\Lambda X^{-1}}_{\text{Eigendecomposition}}$$

- You may remember that eigenvalue decompositions also diagonalize matrices.
- Some key differences:
 - The eigendecomposition insists on using the *same* change of basis for the input and output space.
 - Consequently, the eigendecomposition only exists for *square matrices*—and *not even all square matrices!*
 - X is not a unitary matrix in general.
- However, you can compute the SVD via an eigendecomposition:
 - Consider the “square” of the singular value decomposition, $A^H A = V \Sigma^\top U^H U \Sigma V^H = V \Sigma^\top \Sigma V^H$
 - The right-hand side is actually an eigendecomposition with $\Lambda = \Sigma^\top \Sigma$, $X = V$
 - The singular values are the square root of these eigenvalues.
 - The right-singular vectors are the eigenvectors.
 - The left singular vectors can be found from AV .

Lecture 10 - Conditioning of Computational Problems

Generic Computation Problem

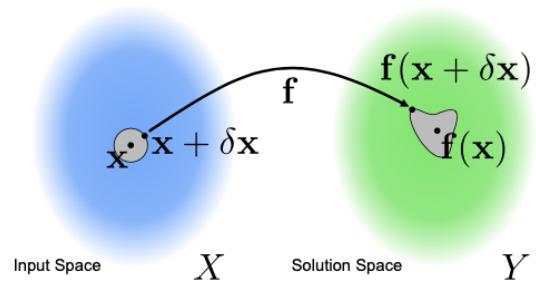
- Abstract viewpoint: all computational problems can be interpreted as a ***nonlinear*** function f mapping from inputs to outputs
 - Assume inputs and outputs live in vector spaces X and Y
 - Assume we have norms for both X and Y
- We will call computing $f(\mathbf{x})$ “solving a problem instance.”
- Central question: how accurately can we evaluate $f(\mathbf{x})$ on a computer?

Conditioning

- How does an inaccuracy/roundoff error in \mathbf{x} affect the computed result $f(\mathbf{x})$ assuming all results are computed exactly?
 - Given a “small” perturbation to the input $\delta\mathbf{x}$, how large can the perturbation to the output $\delta f := f(\mathbf{x} + \delta\mathbf{x}) - f(\mathbf{x})$ be?
 - Stated differently: “How much does f magnify errors/uncertainties?”
 - This will give us a bound on how accurately we can expect to solve a problem instance.

Relative and Absolute Error Measures

- How do we define the “size” of perturbations?
 - We assumed X and Y are normed vector spaces, so we can use each space’s norm, $\|\cdot\|_X$ and $\|\cdot\|_Y$
 - **Absolute** perturbation/error magnitude:
 - ▷ $\|(x + \delta x) - x\|_X = \|\delta x\|_X$
 - ▷ $\|f(x + \delta x) - f(x)\|_Y = \|\delta f\|_Y$
 - **Relative** perturbation/error magnitude:
 - ▷
$$\frac{\|(x + \delta x) - x\|_X}{\|x\|_X} = \frac{\|\delta x\|_X}{\|x\|_X}$$
 - ▷
$$\frac{\|f(x + \delta x) - f(x)\|_Y}{\|f(x)\|_Y} = \frac{\|\delta f\|_Y}{\|f(x)\|_Y}$$

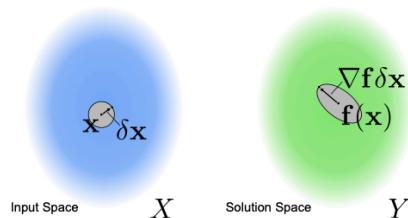


Absolute Condition Numbers

- Definition: $\hat{\kappa} := \lim_{h \rightarrow 0} \max_{\substack{\delta x \\ 0 < \|\delta x\| \leq h}} \frac{\|\delta f\|}{\|\delta x\|}$
- Using **absolute** error measures: Given a ‘small’ perturbation to the input δx , how large can the perturbation to the output $\delta f := f(x + \delta x) - f(x)$ be?
- Worst-case analysis: meaning we must maximize over all input possible perturbations.
- When f is differentiable, we have $\hat{\kappa} = \|\nabla f\|$ (proof see below)

Relative Condition Numbers

- Definition: $\kappa := \lim_{h \rightarrow 0} \max_{\substack{\delta x \\ 0 < \|\delta x\| \leq h}} \frac{\|\delta f\| / \|\delta x\|}{\|f(x)\| / \|x\|}$
- Using **relative** error measures: Given a ‘small’ perturbation to the input δx , how large can the perturbation to the output $\delta f := f(x + \delta x) - f(x)$ be?
- When f is differentiable, we have $\kappa = \frac{\|\nabla f\|}{\|f(x)\| / \|x\|}$ (proof see below)



$$\hat{\kappa} := \lim_{h \rightarrow 0} \max_{\substack{\delta x \\ 0 < \|\delta x\| \leq h}} \frac{\|\delta f\|}{\|\delta x\|}$$

- If $\mathbf{f}(x)$ is differentiable, we can construct a first-order Taylor approximation:

$$\mathbf{f}(x + \delta x) = \mathbf{f}(x) + \nabla \mathbf{f}(x) \delta x + O(\|\delta x\|^2) \implies \delta \mathbf{f}(x) = \nabla \mathbf{f}(x) \delta x + O(\|\delta x\|^2),$$

where $\nabla \mathbf{f}$ is the **Jacobian** matrix whose $(i, j)^{\text{th}}$ entry holds $\frac{\partial f_i}{\partial x_j}$.

- Note: after linearizing, the map from δx to $\delta \mathbf{f}$ is just the matrix-vector multiplication

$$\delta \mathbf{f} \approx \nabla \mathbf{f} \delta x = \sum_j \frac{\partial f_i}{\partial x_j} \delta x_j$$

[Absolute Condition Numbers Proof \(1\)](#)

III-Conditioned Problems

$$\hat{\kappa} = \underbrace{\|\nabla \mathbf{f}\|}_{\text{Absolute}}, \quad \kappa = \underbrace{\frac{\|\nabla \mathbf{f}\|}{\|\mathbf{f}(\mathbf{x})\| / \|\mathbf{x}\|}}_{\text{Relative}}$$

- We will use the **relative** condition number κ in this course.
It is a dimensionless quantity and compatible with the relative error introduced by floating point.
- Given a relative uncertainty $\|\delta \mathbf{x}\| / \|\mathbf{x}\| \leq \epsilon$, we can only expect to compute $\mathbf{f}(\mathbf{x})$ with accuracy:

$$\frac{\|\delta \mathbf{f}\|}{\|\mathbf{f}(\mathbf{x})\|} \leq \kappa \epsilon.$$
- If $\kappa \gg 1$ (e.g., 10^6 or 10^{16}), we may not be able to meaningfully solve the problem! Such problems are called ill-conditioned. Problems with $= 1, 10^2$, etc. are called well-conditioned.

Lecture 11 - Matrix Condition Numbers

Condition Number of a Matrix

- The condition number of a square matrix $A \in \mathbb{C}^{m \times m}$ relative to a norm $\|\cdot\|$ is defined as $\kappa(A) := \|A\| \|A^{-1}\|$ (Proof see below)
 - This condition number is attached to matrix A itself, not to a particular problem
 - If $\kappa(A)$ is small, A is “well-conditioned”
 - If $\kappa(A)$ is large, A is “ill-conditioned”
 - If A is singular, we write $\kappa(A) = \infty$
- When using the 2-norm for $\|\cdot\|$, we have $\|A\|_2 = \sigma_1$, $\|A^{-1}\|_2 = \sigma_m$ and can conclude

$$\kappa(A) = \frac{\sigma_1}{\sigma_m} \quad (\text{for } \|\cdot\| = \|\cdot\|_2)$$

Condition Numbers: Matrix Multiplication

- Important example: consider matrix multiplication $\mathbf{f}(\mathbf{x}) = A\mathbf{x}$ for $A \in \mathbb{C}^{m \times n}$
 - In this case $\nabla \mathbf{f} = A$
 - Let $\|\mathbf{x}\|$, $\|A\mathbf{x}\|$ denote arbitrary norms for \mathbb{C}^n and \mathbb{C}^m , and $\|A\|$ the corresponding induced matrix norm.
 - Then for a given \mathbf{x} :

$$\begin{aligned} \kappa &= \frac{\|\nabla \mathbf{f}\|}{\|\mathbf{f}(\mathbf{x})\| / \|\mathbf{x}\|} = \frac{\|A\|}{\|A\mathbf{x}\| / \|\mathbf{x}\|} \\ &= \|A\| \frac{\|\mathbf{x}\|}{\|A\mathbf{x}\|}. \end{aligned}$$

- When A is square and nonsingular, we can use the fact that $\frac{\|\mathbf{x}\|}{\|A\mathbf{x}\|} = \frac{\|A^{-1}\mathbf{b}\|}{\|\mathbf{b}\|} \leq \|A^{-1}\|$ to say that **for any \mathbf{x} :**

$$\kappa \leq \|A\| \|A^{-1}\|,$$

and there exists some \mathbf{x} for which this upper bound is attained ($\kappa = \|A\| \|A^{-1}\|$).

Lecture 12 & 13 - Floating Point

Floating Point Representation

$$3. \frac{\overbrace{14159}^{\text{significand}}}{\overbrace{10}^{\text{base}}} \times \underbrace{10^{-5}}_{\text{exponent}}$$

- A floating point representation is defined by its:

- Base β (there are standards for $\beta \in \{2, 10, 16\}$, but essentially all computation uses binary $\beta = 2$)
- Precision $p \in \mathbb{Z}^+$, the number of bits/digits for storing the significand; and
- Range for the integer exponent, $[e_{\min}, e_{\max}]$

- To ensure a unique encoding, we use “**normalized**” or “**normal**” floating point numbers wherever possible. This means shifting the significand and adjusting the exponent so that the significand looks like $1.f$.
- In binary ($\beta = 2$) the “**normal**” numbers are in the form: (where f is a $(p-1)$ bits integer) $(-1)^s 1.f \times 2^e$, $f \in \{0, \dots, 2^{p-1} - 1\}$, $s \in \{0, 1\}$, $e \in [e_{\min}, e_{\max}]$
- ± 0 and the “**subnormal**” or “**denormal**” numbers are in the form: $(-1)^s 0.f \times 2^{e_{\min}}$, $f \in \{0, \dots, 2^{p-1} - 1\}$, $s \in \{0, 1\}$, $e = e_{\min} - 1$

- Let's consider the space of floating point numbers in the system

$$\beta = 2, p = 3, e_{\min} = -1, e_{\max} = 1$$

- The “normal” numbers are in the form:

$$(-1)^s 1.f \times 2^e \quad f \in \{00, \dots, 11\}, \quad s \in \{0, 1\}, \quad e \in \{-1, 0, 1\},$$

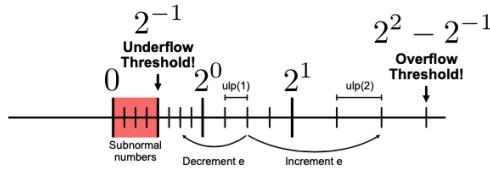
here s is the “sign bit.”

- When $s = 0$, the normal numbers look like

$$(1 + \underbrace{0 \times 2^{-2}}_{.00}) \times 2^e, (1 + \underbrace{1 \times 2^{-2}}_{.01}) \times 2^e, (1 + \underbrace{2 \times 2^{-2}}_{.10}) \times 2^e, (1 + \underbrace{3 \times 2^{-2}}_{.11}) \times 2^e$$

- The “gap” between numbers is $2^{-2} \times 2^e = 2^{e-2}$. In general, this is $2^{e-(p-1)}$. This quantity is called the **unit in the last place** or $\text{ulp}(2^e)$.

- Increasing e doubles the gaps between numbers, decreasing e halves them:



- We can represent “**subnormal**” or “**denormal**” numbers by setting $e = e_{\min} - 1$.

- In this mode, the number is interpreted as $(-1)^s 0.f \times 2^{e_{\min}}$ (note the leading 0).
- “ ± 0 ” can then be represented as $f = 0, e = e_{\min} - 1$.
- “ ± 0 ” is always represented like this, but denormal numbers *can be disabled* for performance reasons.

Unit in the last place and Machine ϵ

$$\hat{x} = (-1)^s 1. \underbrace{[f_{p-1} \dots f_1]}_{p-1 \text{ bit integer } f} \times \beta^e$$

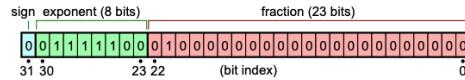
- **Unit in the last place (“Gap”)**

- The next floating number after \hat{x} is found by incrementing f .
- The “gap” between these numbers is the “**unit in the last place**” $ulp(\hat{x})$
- This is the “place value” of f_1 : $ulp(\hat{x}) = ulp(2^e) = \beta^{-(p-1)}\beta^e = \beta^{e-(p-1)}$
- The maximum *absolute error* in rounding a real number in $[\hat{x}, \hat{x} + ulp(\hat{x})]$ is $\frac{ulp(\hat{x})}{2}$
- The maximum *relative error* occurs around $f = 0 \implies \hat{x} = \beta^e$ (the smallest magnitude with exponent e), when rounding $\hat{x} + \overline{0.4999}ulp(\hat{x})$

$$\frac{|x - fl(x)|}{|x|} \leq \frac{|(\beta^e + ulp(\hat{x})/2) - \beta^e|}{|\beta^e + ulp(\hat{x})/2|} < \frac{|ulp(\hat{x})/2|}{|\beta^e|} = \frac{|\beta^{e-(p-1)}/2|}{|\beta^e|} = \beta^{1-p}/2 = ulp(1)/2$$
- ▷ This is our definition of **machine epsilon** $\epsilon = \beta^{1-p}/2 = ulp(1)/2$
- ▷ When $\beta = 2$, this simplifies to $\epsilon = 2^{-p}$

IEEE 754 Floating Point: Single Precision & Double Precision

IEEE 754 Floating Point: Single Precision



- IEEE 754 single precision floating point numbers (`float`) are defined by :
 - $\beta = 2$
 - $p = 24$ (so that 23 bits are used to represent the *fraction* of the significand $1.f$)
 - $e_{\min} = -126, e_{\max} = 127$ ($\approx 10^{-38}$ to 10^{38})
- To represent both negative and positive exponents as a bit string, IEEE 754 uses a somewhat unusual “bias” approach (instead of “sign + magnitude” or two’s complement).
 - For single precision, store $\hat{e} \in \{0, \dots, 255\}$ and define: $e = \hat{e} - 127$.
 - $\{e_{\min}, \dots, e_{\max}\}$ correspond to $\hat{e} \in \{1, \dots, 254\}$
 - $\hat{e} = 0$ indicates zeros/denormals.
 - $\hat{e} = 255$ signals $\pm\infty$ when $f = 0$ (e.g., result of $\frac{1}{\pm 0}$), or NaN (not a number!) when $f \neq 0$ (e.g., result of $\frac{0}{0}, \infty - \infty$).
 - This is so positive floats sort in the correct order if we reinterpret the bitstring as an integer (simplifies comparison)
- The gap between floating point numbers with exponent e is $ulp(2^e) = 2^{e-(p-1)} = 2^{e-23}$
 - The *relative rounding error* is thus bounded by $\epsilon = ulp(1)/2 = 2^{-23}/2 = 2^{-24} \approx 6 \times 10^{-8}$.
 - We call this the “**machine epsilon**.” Note: some references define $\epsilon = ulp(1)$ instead...
 -

Single Precision

Fundamental Axiom of Floating Point Arithmetic

- For any $x \in \pm [2^{e_{\min}}, 2^{e_{\max}}] \subset \mathbb{R}$, the nearest floating point representation satisfies:
 $fl(x) = x(1 + \delta) \quad |\delta| \leq \epsilon$, where $|\delta|$ is the **relative error** $|fl(x) - x|/|x|$.

- Single precision represents numbers with 7-8 digits of accuracy, Double precision gives 16 digits
- IEEE 754 guarantees that the floating point result of, e.g., $x + y$ of two floating point numbers x and y is the exact result rounded to the nearest floating point number.
 - **Fundamental Axiom of Floating Point Arithmetic:** For all floating point numbers $fl(x), fl(y)$, there exists δ with $|\delta| \leq \epsilon$ such that:
 $fl(x) \oplus fl(y) = (fl(x) * fl(y))(1 + \delta)$, where $*$ stands for any of the arithmetic operators $+, -, /, \times$, and \oplus is the floating point operator.

Floating Point Peculiarities

- Floating point numbers do not behave exactly like real numbers
 - Additive inverses ($\pm n$) are always exactly representable (flip s), but multiplicative inverses ($n, \frac{1}{n}$) are not!
 - Operations are commutative:
 - ▷ $a \oplus b = b \oplus a$
 - ▷ $a \otimes b = b \otimes a$
 - Operations are not associative (use parentheses carefully)
 - ▷ $a \oplus (b \oplus c) \neq (a \oplus b) \oplus c$
 - ▷ $a \otimes (b \otimes c) \neq (a \otimes b) \otimes c$
 - Does the following statement hold?

$$x \neq y \implies (x - y) \neq 0$$
 - Actually, yes! But only with denormals/subnormals enabled...
 - Example: $1.1 \times 2^{e_{\min}} - 1.0 \times 2^{e_{\min}} = 0.1 \times 2^{e_{\min}}$
Underflows to 0 if denormals are disabled/unsupported.

Because basic identities like $(x * (1/x)) = 1$ fail to hold in floating point, you should **almost never** test “ $x = y$ ” for two floating point numbers.

Prefer tests like “ $|x - y| < \text{tol}$ ” for some tolerance parameter tol .

Error Propagation

- Multiplication

$$z = (x(1 + \delta_x)) \otimes (y(1 + \delta_y)) = (x(1 + \delta_x) * y(1 + \delta_y)) * (1 + \delta) \quad |\delta| \leq \epsilon$$
 - The **relative error** in this approximation is

$$\frac{|z - xy|}{|xy|} = \frac{|xy(1 + \delta_x)(1 + \delta_y)(1 + \delta) - xy|}{|xy|} = |(1 + \delta_x)(1 + \delta_y)(1 + \delta) - 1| = \delta_x + \delta_y + \delta + \text{higher order terms}$$

- Therefore, when multiplying and dividing, the **relative** errors **add**.
- Consequently, multiplication and division are safe since accurate inputs give similarly accurate outputs.
- Addition $z = (x(1 + \delta x)) \oplus (y(1 + \delta y)) = (x(1 + \delta x) + y(1 + \delta y)) * (1 + \delta)$ $|\delta| \leq \epsilon$
 - The **absolute** error in this approximation is
 $|z - (x + y)| = |(x + x\delta x + y + y\delta y) + O(\delta) - x - y| = |x\delta x + y\delta y + O(\delta)|$
 - Here, the **absolute** errors of the input $x\delta x$ and $y\delta y$ **add**.
 - This is dangerous since adding two inputs with low relative error can produce an output with high relative error when $|x + y| \ll \max(|x|, |y|)$
 - This is the so called **catastrophic cancellation** phenomenon
 - *Example: calculate $1 - \sqrt{1 - c}$ (see picture below)*
- **Loss of Digits**
 - We also run into problems when adding numbers of widely-differing magnitude
 - $1.0010 + 0.00013000 = 1.0011$
 - Only registers one of the digits of the smaller number!
 - *Example: harmonic numbers (see picture below)*

Catastrophic Cancellation

- Catastrophic cancellation example:

$$1.23456 \pm 5 \times 10^{-6} - 1.23455 \pm 5 \times 10^{-6} = 0.00001 \pm 5 \times 10^{-6}$$

- Relative error is now $5 \times 10^{-6} / 1 \times 10^{-5} = 50\%$ despite good accuracy in the inputs.

- Consider trying to solve $x^2 - 2x + c = 0$ for $|c| \ll 1$

- One of the roots is $1 - \sqrt{1 - c}$
- What's dangerous about this expression?

Floating Point “Exceptions”

- By default, calculations proceed silently, even with unusual inputs:
 - Dividing $x \neq 0$ by zero: $x/0$ gives $\text{sign}(x) * \infty$.
 - $1/\infty$ gives 0
 - Operations like $\sqrt{-1}$, $0/0$, ∞/∞ , $\infty * 0$ produce NaN
 - Any operation involving NaN produces a NaN, ensuring if a NaN ever originates in a computation, it propagates to the final result.
 - IEEE 754 comparison rules for NaNs may look strange at first glance:
NaN cmp NaN is false for any comparison cmp in $\{=, \neq, <, >, \leq, \geq\}$
 - Despite associativity, $a + b == b + a$ can be false if a or b is NaN !
- Usually verifying the final result is not NaN suffices, but IEEE standard provides mechanisms for notifying us immediately when things go wrong.
 - Exception flags FE_DIVBYZERO, FE_INVALID, FE_INEXACT, FE_OVERFLOW, FE_UNDERFLOW can be checked after a calculation.
 - On many platforms, exception handling can be enabled so that some or all of these errors “trap” and invoke a custom handler function. (e.g., `feenableexcept` on Linux)

IEEE 754 Rounding Modes

- By default, numbers that cannot be represented exactly are rounded to the nearest floating point number.
- However, IEEE 754 requires that three other rounding modes can be selected:
 - Round toward 0
 - Round toward $+\infty$ (analogous to “ceiling”)
 - Round toward $-\infty$ (analogous to “floor”)
- These modes also modify overflow behavior:
 - Round toward 0 or $-\infty$: overflowing **positive** values round down to the largest representable value.
 - Round toward 0 or $+\infty$: overflowing **negative** values round up to the most negative representable value.
- Application: *interval arithmetic*
 - Compute lower and upper bounds l, u for the exact result $x \in [l, u]$ of every operation
 - At the end of a computation, you know the exact answer must be within the computed bounds.
 - Small interval \Rightarrow answer known precisely. However, bounds are often overly pessimistic/conservative.

Lecture 14 - C++ Types

Types in C++

- Every variable and expression in C/C++ has a **static** (known at compile-time) type.
 - Floating point:

float	IEEE 754 single-precision (32-bit)
double	IEEE 754 double-precision (64-bit)
 - Signed/Unsigned integers:

short, unsigned short	usually 16-bit
int, unsigned int	usually 32-bit
long, unsigned long	64-bit on modern Linux machines, 32-bit on Windows...
 - Fixed-width integer types: #include <cstdint>

int{8,16,32,64}_t	signed integer with exactly 8, 16, ... bits.
uint{8,16,32,64}_t	unsigned integer with exactly 8, 16, ... bits.

Type Conversions in C++

- See more rules [here](#).
- For binary operators with mismatched types, one operand converts to the other's type.
- What are the following expressions' types?
 - 1.0f / 2 float
 - 1.0f / 2.0 double
 - 1u + (-2) unsigned int (value is 4294967295)
 - 1u + (-2ll) long long (value is -1)
- Conversion rules:
 - Integers are *always* promoted to floating point.
 - Narrower types are converted to wider types.
 - Signed integers of the “same width” as unsigned are converted to unsigned.
 - [See more rules here](#).
- These implicit conversions can be counterintuitive/unexpected and lead to bugs!
 - It is best to be **explicit** when it matters (using **casts**)!

Casting in C++

- Values can be explicitly converted to another type using casts.
- In C++ there are several different kinds of casts with varying degrees of safety:

(float)(1)	C-style cast: most powerful/dangerous; attempt C++ casts below in increasing order of danger.
float(1)	Functional cast: “syntactic sugar” for C-style cast (equivalent)
const_cast<T>(expr)	Add/remove const qualifier
static_cast<T>(expr)	Apply any available implicit conversion rules, cast base pointer to derived pointer, etc.
reinterpret_cast<T>(expr)	Reinterpret bits as if they were of type T. This can be used to inspect bits of a float .

- More on [this](#)
- For numerical expressions, we can generally use C-style or functional casts.

- How to check if a value v stored in a *double* is representable by a *float*?
 - Cast to float and then back to double: test if $\text{double}(\text{float}(v)) == v$.
- Functional Argument Type
 - For function calls, the compiler tries to implicitly convert arguments to the accepted types (see right)
- Templates in C++
 - Templates enable us to write generic code that can be instantiated for different types (and much more...)
 - The compiler automatically generates a distinct functions for each T used.

```
#include<iostream>
int f(int i) {
    return i;
}

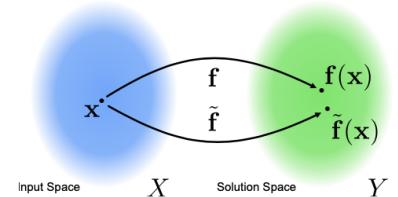
int main() {
    std::cout << f(1.98) << std::endl;
    return 0;           print 1
}
```

Lecture 15 - Stability

How to analyze error accumulated when computing $f(x + \delta x)$?

Accuracy of Algorithms

- We view an algorithm for problem f as another nonlinear function $\tilde{f} : X \rightarrow Y$
- Evaluating $\tilde{f}(x)$ to solve a problem instance $f(x)$ entails:
 - Rounding true input x to the computer's number representation; and then
 - Executing a finite sequence of arithmetic operations (a computer program implementing some numerical method), accumulating roundoff error in every step.
- Def: Ideally \tilde{f} is “close” to f , achieving a low relative error (high accuracy) for all inputs $x \in X$: $\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = O(\epsilon)$

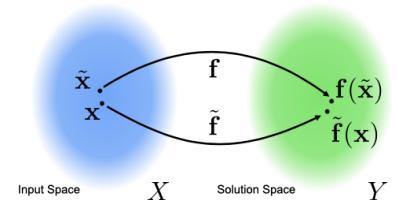


Stability

- Accuracy is too much to ask for. Stability is a weaker, more realistic goal.
- Def: An algorithm \tilde{f} is **stable** if for each $x \in X$:

$$\frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = O(\epsilon)$$
 for some \tilde{x} such that

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon)$$

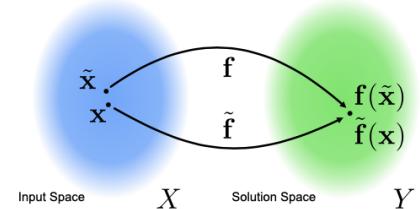


- It computes almost the right solution to almost the right problem.
- It's sometimes called "mixed stability" since both the problem instance and the solution are allowed to perturb.

- Accurate algorithms are clearly stable since $\frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(x)\|} = O(\epsilon)$ holds for $\tilde{x} = x$

Backward Stability

- Stronger but simpler condition than stability.
- Def: Algorithm \tilde{f} is **backward stable** if $\forall x \in X$:
 $\tilde{f}(x) = f(\tilde{x})$ for some \tilde{x} such that $\frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon)$.
- It computes the exact solution to almost the right problem.
- $\frac{\|\tilde{x} - x\|}{\|x\|}$ is called the (relative) backward error. (How much the input must change to make our approximate solution exact)



Meaning of $O(\epsilon)$

- The statement that an expression is $O(\epsilon)$ refers to the limit $\epsilon \rightarrow 0$. That is, we consider a sequence of higher- and higher-precision number systems.

Example: Floating Point Addition

- We saw earlier that floating point addition \oplus is **dangerous** because *absolute* errors add and relative errors are not kept under control.

- In other words, there are inputs for which the result of addition will not be accurate when cancellation occurs.
 - This is due to the conditioning of the problem $f(x_1, x_2) = x_1 + x_2$. We shouldn't blame the algorithm!

- The algorithm $\tilde{f}(x_1, x_2) = \text{fl}(x_1) \oplus \text{fl}(x_2)$ is still **backward stable** (and thus **stable**):

$$\begin{aligned}
 \tilde{f}(x_1, x_2) &= \text{fl}(x_1) \oplus \text{fl}(x_2) \\
 &= x_1(1 + \delta_1) \oplus x_2(1 + \delta_2) & |\delta_1|, |\delta_2| \leq \epsilon \\
 &= (x_1(1 + \delta_1) + x_2(1 + \delta_2))(1 + \delta_3) & |\delta_1| \leq \epsilon \quad (\text{Fundamental axiom of fp arithmetic}) \\
 &= x_1(1 + \delta_4) + x_2(1 + \delta_5) \\
 &= f(x_1(1 + \delta_4), x_2(1 + \delta_5)) \\
 \text{with } \delta_4 &= (1 + \delta_1)(1 + \delta_3) - 1 = \delta_1 + \delta_3 + \delta_1\delta_3 & |\delta_4| \leq 2\epsilon + \epsilon^2 \\
 \text{with } \delta_5 &= (1 + \delta_2)(1 + \delta_3) - 1 = \delta_2 + \delta_3 + \delta_2\delta_3 & |\delta_5| \leq 2\epsilon + \epsilon^2.
 \end{aligned}$$

$\tilde{f}(x_1, x_2)$ computes the exact result for perturbed inputs $\tilde{x} = \begin{bmatrix} x_1(1 + \delta_4) \\ x_2(1 + \delta_5) \end{bmatrix}$, with $\|\tilde{x} - x\| = \left\| \begin{bmatrix} x_1\delta_4 \\ x_2\delta_5 \end{bmatrix} \right\| \leq \|x\|(2\epsilon + \epsilon^2) \leq C\epsilon\|x\|$.

- $\frac{\|x - \tilde{x}\|}{\|x\|} = O(\epsilon)$ really means that for all ϵ small enough: $\|x - \tilde{x}\| = C\epsilon\|x\|$
 - constant C independent of the input x
 - moving $\|x\|$ to right side handles $\|x\| = 0$
 - In other words, the backward error for even the worst x must eventually shrink proportionally to ϵ (or faster).

What Norm?

- It doesn't matter.

Algorithm \tilde{f} is **backward stable** if $\forall x \in X$:

$$\tilde{f}(x) = f(\tilde{x}) \quad \text{for some } \tilde{x} \quad \text{such that } \|x - \tilde{x}\| \leq C\epsilon\|x\|.$$

- Which vector norm should we use to measure stability (or accuracy)?

- It does not matter. All finite-dimensional vector norms are equivalent!

- By this we mean for any two norms $\|\cdot\|_a$ and $\|\cdot\|_b$, there are positive constants C_1 and C_2 such that:

$$C_1\|x\|_a \leq \|x\|_b \leq C_2\|x\|_a \quad \forall x$$

- Therefore, after proving backward stability using $\|\cdot\|_a$:

$$\|x - \tilde{x}\|_a \leq C\epsilon\|x\|_a,$$

we can conclude:

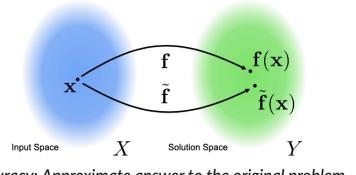
$$\|x - \tilde{x}\|_b \leq C_2\|x - \tilde{x}\|_a \leq C_2C\epsilon\|x\|_a \leq C_2C\epsilon \frac{\|x\|_b}{C_1},$$

proving backward stability under $\|\cdot\|_b$ just with a different constant $\frac{C_2}{C_1}C$.

Stability Summary

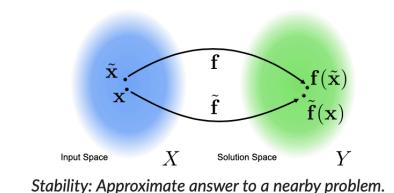
An algorithm \tilde{f} is **accurate** if $\forall x \in X$:

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = O(\epsilon).$$



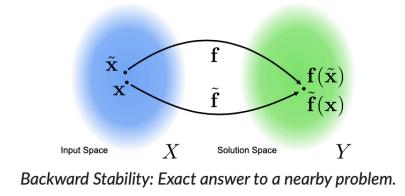
An algorithm \tilde{f} is **stable** if $\forall x \in X$:

$$\frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = O(\epsilon) \quad \text{for some } \tilde{x} \text{ such that } \frac{\|x - \tilde{x}\|}{\|x\|} = O(\epsilon).$$



Algorithm \tilde{f} is **backward stable** if $\forall x \in X$:

$$\tilde{f}(x) = f(\tilde{x}) \quad \text{for some } \tilde{x} \text{ such that } \frac{\|x - \tilde{x}\|}{\|x\|} = O(\epsilon).$$



Accuracy vs. Backward Stability

- Algorithms mapping from lower-dimensional input space X to higher-dimensional input space Y generally cannot be backward stable.
 - Arbitrarily small roundoff error can perturb the result outside the image $f(X) \subset Y$

- For $\kappa \ll 1$ we can hope for accuracy but not backward stability.
- For $\kappa \gg 1$ we can hope for backward stability but not accuracy.
- In both cases, we can hope for stable algorithms.
- When an algorithm is stable, its accuracy reflects the condition number.

Accuracy of a Backward-Stable Algorithm

This leads us to consider the connection between backward stability and accuracy: how accurate will a backward stable algorithm $\tilde{f}(x)$ be?

By definition of backward stability, for all x there exists some \tilde{x} such that:

$$\tilde{f}(x) = f(\tilde{x}), \quad \text{and} \quad \frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon).$$

Recall the definition of the condition number for a problem instance $f(x)$:

$$\begin{aligned} \kappa(x) &= \lim_{h \rightarrow 0} \max_{\|\delta x\| \leq h} \frac{\|f(x + \delta x) - f(x)\|}{\|f(x)\|} / \frac{\|\delta x\|}{\|x\|}. \\ \implies \max_{\|\delta x\| \leq h} \frac{\|f(x + \delta x) - f(x)\|}{\|f(x)\|} / \frac{\|\delta x\|}{\|x\|} &= \kappa(x) + o(1) \quad o(1) \text{ denotes terms shrinking to zero as } \|\delta x\| \rightarrow 0. \end{aligned}$$

Perturbation $\delta x = \tilde{x} - x$ can be no worse than the maximizer for $h = \|\tilde{x} - x\|$:

$$\frac{\|f(x + (\tilde{x} - x)) - f(x)\|}{\|f(x)\|} \leq (\kappa(x) + o(1)) \frac{\|\tilde{x} - x\|}{\|x\|}.$$

But $f(x + (\tilde{x} - x)) = f(\tilde{x}) = \tilde{f}(x)$, and we conclude $\tilde{f}(x)$ has accuracy:

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} \leq (\kappa(x) + o(1)) \frac{\|\tilde{x} - x\|}{\|x\|} = \kappa(x)O(\epsilon).$$

When an algorithm is stable, its accuracy reflects the condition number.

Lecture 16 - Triangular Linear Systems

Triangular Linear Systems

- Triangular systems are a special type of linear system that can be solved with a very simple, efficient, and stable algorithm.
 - Efficient approaches for solving general linear systems factor a matrix into a product of matrices where at least one is triangular and then apply a triangular system solver.
 - ▶ LU decomposition factors into the product of lower and upper triangular matrices.
 - ▶ QR decomposition factors into the product of a unitary matrix and an upper triangular matrix.
- Example: Lower Triangular(or “Left Triangular”)

$$\begin{aligned}
 l_{11}x_1 &= b_1 \\
 l_{21}x_1 + l_{22}x_2 &= b_2 \\
 l_{31}x_1 + l_{32}x_2 + l_{33}x_3 &= b_3 \\
 &\vdots \\
 l_{m1}x_1 + l_{m2}x_2 + l_{m3}x_3 + \cdots + l_{mm}x_m &= b_m
 \end{aligned}
 \implies \underbrace{\begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ l_{31} & l_{32} & l_{33} & \\ \vdots & & & \ddots \\ l_{m1} & l_{m2} & l_{m3} & \cdots & l_{mm} \end{bmatrix}}_L \mathbf{x} = \mathbf{b}$$

Lower/Left Triangular

Forward / Backward Substitution

- Both are backward stable (Proof see Slides 15b-6)

Forward Substitution

- We can efficiently solve a *lower* triangular system by rearranging:

$$\begin{aligned}
 l_{11}x_1 &= b_1 & x_1 &= b_1/l_{11} \\
 l_{21}x_1 + l_{22}x_2 &= b_2 & x_2 &= (b_2 - l_{21}x_1)/l_{22} \\
 l_{31}x_1 + l_{32}x_2 + l_{33}x_3 &= b_3 & x_3 &= (b_3 - l_{31}x_1 - l_{32}x_2)/l_{33} \\
 &\vdots & &\vdots \\
 l_{m1}x_1 + l_{m2}x_2 + l_{m3}x_3 + \cdots + l_{mm}x_m &= b_m & x_i &= \left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j \right) / l_{ii}
 \end{aligned}$$

- Solving for x_1, x_2, \dots, x_m in sequence, the RHS expression contains known quantities at every step.
- This process is called “**forward substitution**” since solved values are substituted into the next equation.
- Rearranged equations are still “lower triangular” in shape, needing a \odot and \ominus at each position.
Thus roughly m^2 floating point operations must be performed.

Lecture 16 - LU Decomposition

LU Decomposition

- Factor a matrix $A \in \mathbb{C}^{m \times m}$ into the form: $A = LU$, where L is lower triangular and U is upper triangular.
 - The algorithm is Gaussian elimination with careful bookkeeping

- Motivation

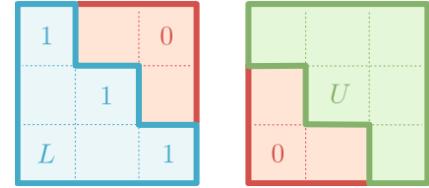
- Motivation: computing determinants

$$\det(A) = \det(LU) = \det(L) \det(U) = \left(\prod_{i=1}^m l_{ii} \right) \left(\prod_{i=1}^m u_{ii} \right) = \prod_{i=1}^m u_{ii} \quad (L \text{ is *unit* lower triangular})$$

- Motivation: solving linear systems

$$Ax = b \implies L \underbrace{Ux}_y = b$$

- Solve $Ly = b$ (forward substitution: fast ($O(m^2)$) and stable)
- Solve $Ux = y$ (back substitution: ($O(m^2)$) and stable)
- LU with pivoting (next lecture) is the standard approach for solving linear systems.



Gaussian Elimination

- General definition (pic 1)
- Example (pic 2-4)
- General Case (pic 5-7)

- Zero out entries below the diagonal, one column at a time:

$$\underbrace{\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}}_A \xrightarrow{L_1} \underbrace{\begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix}}_{L_1 A} \xrightarrow{L_2} \underbrace{\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix}}_{L_2 L_1 A} \xrightarrow{L_3} \underbrace{\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \end{bmatrix}}_{U=L_3 L_2 L_1 A}$$

- This is accomplished by multiplying by a sequence of lower triangular matrices:

$$L_{m-1} \cdots L_1 A = U,$$

where L_k subtracts multiples of row k from the rows below.

- Therefore: $A = (L_{m-1} \cdots L_1)^{-1} U = LU \quad L = L_1^{-1} \cdots L_{m-1}^{-1}$
 - Facts used from homework 0:
 - The inverse of a lower triangular matrix is lower triangular.
 - Product of lower triangular matrices $L_{m-1} \cdots L_1$ is lower triangular.

Gaussian Elimination

- Complexity

Complexity of Gaussian Elimination

The LU decomposition algorithm operating on an input matrix $A \in \mathbb{C}^{m \times m}$ is:

```
algorithm lu(A):
    L, U = I, A
    for k=1 to m - 1:
        for i=k + 1 to m:
            L(i, k) = U(i, k) / U(k, k)
            for j=k + 1 to m:
                U(i, j) -= L(i, k) * U(k, j) # Subtract multiple of row k.
    return L, U
```

- Work is dominated by the multiplication and subtraction in the innermost loop.

- For each outermost iteration k , we do a “double-for” loop over the $m - k$ indices $\{k + 1, \dots, m\}$. Thus, the innermost line runs $(m - k)^2$ times, performing two operations each time:

$$\begin{aligned} \sum_{k=1}^{m-1} 2(m-k)^2 &= \sum_{k'=1}^{m-1} 2(k')^2, \quad \text{with } k' = m - k \\ &= 2 \frac{(m-1)m(2m-1)}{6} = 2 \frac{m^3}{3} - m^2 + \frac{m}{6} \sim \frac{2}{3}m^3. \end{aligned}$$

- Therefore roughly $\frac{2}{3}m^3$ floating point operations are needed.

Lecture 17 - LU Decomposition with Pivoting

Pivots

- The version of Gaussian elimination presented is neither backward stable nor stable — very dangerous
- The backward stability issue is closely related to zeros on the diagonal leading to $l_{ik} = \frac{a_{ik}}{0}$
 - Solution: swap the rows/columns (pivoting)
 - Pivoting brings backward stability
- Step k of Gaussian elimination looks like
 - The k^{th} row is used to eliminate the entries below it in column k .
 - Entry x_{kk} is called the “pivot” in this operation
 - We subtract x_{ik}/x_{kk} times row k from each row $i > k$. (Dividing by $x_{kk} \approx 0$ will be problematic)
 - It’s not necessary to choose x_{kk} as the pivot.
 - For stability, pick largest value as pivot, then exchanging rows/columns.
 - This process of exchanging rows/columns can be interpreted as reordering the variables/equations and is called pivoting.

$$\left[\begin{array}{cccc} * & * & * & * \\ x_{kk} & * & * \\ * & * & * \\ * & * & * \end{array} \right] \rightarrow \left[\begin{array}{cccc} * & * & * & * \\ x_{kk} & * & * \\ 0 & * & * \\ 0 & * & * \end{array} \right]$$

$$\left[\begin{array}{cccc} * & * & * & * \\ * & * & * & * \\ x_{ij} & * & * & * \\ * & * & * & * \end{array} \right] \rightarrow \left[\begin{array}{cccc} * & * & * & * \\ * & * & * & * \\ x_{ij} & * & * & * \\ * & * & * & * \end{array} \right] \rightarrow \left[\begin{array}{cccc} * & * & * & * \\ x_{ij} & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{array} \right]$$

Pivoting

The **LU decomposition algorithm** operating on an input matrix $A \in \mathbb{C}^{m \times m}$ is:

```
algorithm lu(A):
    L, U = I, A
    for k=1 to m - 1:
        for i=k + 1 to m:
            L(i, k) = U(i, k) / U(k, k)
            for j=k + 1 to m:
                U(i, j) -= L(i, k) * U(k, j) # Subtract multiple of row k.
    return L, U
```

Choosing Pivots: Complete Pivoting

- At step k , we ideally pick the largest magnitude value in block $i, j \geq k$ as the pivot.
- Require searching through the $(m - k + 1)^2$ entries for each step $1 \leq k < m$, it means $O(m^3)$ work for selecting pivots, a nontrivial cost on top of $\frac{2}{3}m^3$ operations of plain lu.

Choosing Pivots: Partial Pivoting

- Only reorder the **rows**. Choose the largest entry from column k as the pivot: x_{ik} ($i \geq k$)
- Require searching through the $(m - k + 1)$ entries for each step $1 \leq k < m$: only $O(m^2)$ work
- Almost as stable as complete pivoting in practice.

$$\begin{bmatrix} * & * & * & * \\ & * & * & * \\ \hline & & \textcolor{green}{x_{ik}} & * \\ \hline & * & * & * \end{bmatrix}$$

Permutation Matrix in Partial Pivoting

- The row exchange operation can be written as a multiplication by a simple permutation matrix $P_k = P_k I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- Permutation matrices P are orthogonal: $P^{-1} = P^T$ and “binary”
- Because P_k just exchanges two rows: P_k is an **involution**: $P_k P_k = I$ and **symmetric**: $P_k = P_k^{-1} = P_k^T$
- After all steps $k \in \{1, \dots, m - 1\}$ we have: $L_{m-1} P_{m-1} \dots L_2 P_2 L_1 P_1 A = U$
- To make $L_{m-1} P_{m-1} \dots L_2 P_2 L_1 P_1$ a lower triangular, we can “shift” later row exchanges P_k past earlier elimination matrices L_j ($j < k$) to collect them on the right.
 - This simply reorders the rows modified by L_j but does not change the typical lower triangular “elimination step” instructor of L_j
 - The multipliers l_{ij} in column j are just reordered by the permutation $P_k \dots P_{j+1}$ to obtain entries l'_{ij} of L'_j
 - At the end of this process: $L'_{m-1} \dots L'_1 P_{m-1} \dots P_1 A = U$

- Finally, $L'_{m-1} \dots L'_1 P_{m-1} \dots P_1 A = U \implies P_{m-1} \dots P_1 A = L'^{-1}_{m-1} \dots L'^{-1}_1 U \implies PA = LU$
 - This can be interpreted as applying P to A and running the ordinary LU algorithm. (But it determines P computes the LU decomposition of PA all at once).
 - LU with partial pivoting even works for all square matrices—**even singular ones!**
 - L is always unit lower triangular ($\det(L)=1$).
 - But when A is singular, U has zeros on its diagonal ($\det(U)=\det(A)=0$).

Gaussian elimination with partial pivoting operating $A \in \mathbb{C}^{m \times m}$ is the algorithm:

```
algorithm lu_partial_pivot(A):
    L, U, P = I, A, I
    for k=1 to m - 1:
        imax = argmax_{i ≥ k}(|U(i, k)|)
        swap(U(k, k:m), U(imax, k:m))
        swap(L(k, 1:k-1), L(imax, 1:k-1))
        swap(P(k, :), P(imax, :))
        for i=k + 1 to m:
            L(i, k) = U(i, k) / U(k, k)
            for j=k + 1 to m:
                U(i, j) -= L(i, k) * U(k, j)
    return L, U, P
```

Partial Pivoting Example

Partial Pivoting Example

- We run LU with partial pivoting on the same example matrix from last lecture:

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}$$

- Step 1a: choose pivot entry (3, 1) and exchange

$$\underbrace{\begin{bmatrix} & 1 \\ & 1 \\ 1 & & \\ & & 1 \end{bmatrix}}_{P_1} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{bmatrix}$$

- Step 1b: eliminate

$$\underbrace{\begin{bmatrix} 1 & & \\ -\frac{1}{2} & 1 & \\ -\frac{1}{4} & & 1 \\ -\frac{3}{4} & & \end{bmatrix}}_{L_1} \begin{bmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 8 & 7 & 9 & 5 \\ -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} & \\ -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} & \\ \frac{7}{4} & \frac{9}{4} & \frac{17}{4} & \end{bmatrix}$$

Lecture 18 - Stability of LU

Without Pivoting

- In our **pathological example** for LU without pivoting:

$$A = \begin{bmatrix} 10^{-30} & 1 \\ 1 & 1 \end{bmatrix} \implies L = \begin{bmatrix} 1 & 0 \\ 10^{30} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 10^{-30} & 1 \\ 0 & 1 - 10^{30} \end{bmatrix}$$

$$\implies \tilde{L} = \begin{bmatrix} 1 & 0 \\ 10^{30} & 1 \end{bmatrix}, \quad \tilde{U} = \begin{bmatrix} 10^{-30} & 1 \\ 0 & -10^{30} \end{bmatrix}$$

we observed a large relative backward error $\frac{\|A - \tilde{L}\tilde{U}\|}{\|A\|} = \frac{\left\| \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \right\|}{\|A\|}$ that caused a completely wrong answer to " $Ax = b$ " via forward/back substitution.

- LU* without pivoting is **not** backward stable
 - We said this was linked to the huge entry 10^{30} in the *L* factor, which we could avoid with pivoting.
 - It turns out large backward errors are *always* due to large entries in either *U* or *L*

Backward Stability Theorem

Suppose $A = LU$ for a nonsingular $A \in \mathbb{C}^{m \times m}$ is computed by Gaussian elimination **without pivoting** to obtain \tilde{L} and \tilde{U} . If the *LU* factorization exists, and the computer obeys the fundamental axiom of FPA:

$$\tilde{L}\tilde{U} = A + \delta A \quad \frac{\|\delta A\|}{\|L\|\|U\|} = O(\epsilon) \quad \text{Warning: } \|L\|\|U\| \text{ in the denominator, not } \|A\|!!!$$

for some $\delta A \in \mathbb{C}^{m \times m}$.

- This also applies to Gaussian elimination **with pivoting** if we first apply *P* to *A*.
- Recall, for backward stability, we require $\frac{\|\delta A\|}{\|A\|} = O(\epsilon)$.
 - This is only true if $\|L\|\|U\| = O(\|A\|)$.
 - If $\|L\|\|U\| > O(\|A\|)$ we expect backward instability.
 - Without pivoting, entries in *L* and *U* can become unbounded in magnitude.
 - With partial pivoting, we know *L* has entries of magnitude ≤ 1 (formed by dividing a smaller column *k* entry by the pivot x_{kk}) and thus $\|L\| = O(1)$.
 - The condition for stability is thus $\frac{\|\delta A\|}{\|U\|} = O(\epsilon)$ or $\|U\| = O(\|A\|)$.

Complete Pivoting

- Complete pivoting can avoid backward instability. But in practice, complete pivoting is almost never done since choosing pivots takes $O(m^3)$ work, and partial pivoting nearly always achieves low backward error.
- Nevertheless, it can still be written in matrix form:

$$\begin{aligned}
 & (L_{m-1}P_{m-1}\dots L_2P_2L_1P_1)AQ_1Q_2\dots Q_{m-1} = U \\
 \implies & \underbrace{(L'_{m-1}\dots L'_2L'_1)}_{L^{-1}} \underbrace{P_{m-1}\dots P_2P_1}_{P} \underbrace{A Q_1Q_2\dots Q_{m-1}}_{Q} = U \\
 \implies & PAQ = LU
 \end{aligned}$$

Partial Pivoting

- With partial pivoting, L has entries of magnitude ≤ 1 , thus $\|L\| = O(1)$
- The condition for stability is $\frac{\|\delta A\|}{\|U\|} = O(\epsilon)$ or $\|U\| = O(\|A\|)$
- Growth factor

Backward Stability Theorem Partial Pivoting

- With partial pivoting, the condition for stability is $\frac{\|\delta A\|}{\|U\|} = O(\epsilon)$ or $\|U\| = O(\|A\|)$.

 - To reason about this, we introduce the **growth factor**:

$$\rho := \frac{\max_{i,j} |u_{ij}|}{\max_{i,j} |a_{ij}|},$$

so that $\|U\| = O(\rho \|A\|) = \|A\| O(\rho)$.

 - Therefore, we can state the following corollary to the general backward stability condition:

Suppose $A = LU$ for a nonsingular $A \in \mathbb{C}^{m \times m}$ is computed by Gaussian elimination **with partial pivoting** to obtain \tilde{P} , \tilde{L} , and \tilde{U} . If the computer obeys the fund. axiom of FPA:

$$\tilde{L}\tilde{U} = \tilde{P}A + \delta A \quad \frac{\|\delta A\|}{\|A\|} = O(\rho\epsilon)$$

for some $\delta A \in \mathbb{C}^{m \times m}$, where ρ is the growth factor for A . If $|l_{ij}|$ for each $i > j$ (so that there are no ties in pivot selection in exact arithmetic) then $\tilde{P} = P$ for sufficiently small ϵ .

- Gaussian elimination with partial pivoting is thus stable if $\rho = O(1)$ for all A of dimension m .

Wednesday, September 22, 2021

- Using an informal, more practical definition of the stability as the growth of ρ as we increase matrix size m , we see that in the worst case, Gaussian elimination is horribly unstable. However, the worst case seldom happens in real applications.
- Conclusion: Partial Pivoting is truly helpful and appears to be sufficient for acceptable (non-exponential) growth factors except in vanishingly rare pathological cases.

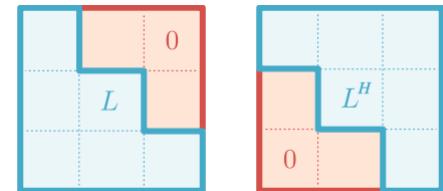
Lecture 19 - Cholesky Factorization

Motivation

- LU decomposition without pivoting $A = LU$, not backward stable
- LU decomposition with partial pivoting $PA = LU$, technically backward stable, though theoretically with backward error growing exponentially as we enlarge the matrix $\rho = 2^{m-1}$
- LU decomposition with complete pivoting $PAQ = LU$, keep backward error under control at an extra cost
- What if A is symmetric or hermitian? Can we come up with a faster algorithm?

Cholesky Factorization

- $A = LL^H$ or $A = RR^H$ with positive real diagonal entries $L_{ii} > 0$
- If $A \in \mathbb{C}^{m \times m}$ is hermitian/symmetric, it seems natural for LU to respect this symmetry.
 - The product of two upper/lower triangular is upper/lower triangular
 - By this, we mean $U = L^H$
 - No extra time and space needed to calculate U in LU decomposition
- A is called **positive definite** (正定矩阵) if $A \in \mathbb{R}^{m \times m}$ we have $x^T Ax > 0 \quad \forall x \neq 0$
- A is called **hermitian positive definite** (hermitian 正定矩阵) if $A \in \mathbb{C}^{m \times m}$ we have $A = A^H, \quad x^H Ax > 0 \quad \forall x \neq 0$
- Is this always possible? Consider the vector $y = L^H x$ for an arbitrary nonzero $x \in \mathbb{C}^m$.
 - We can compute the ℓ_2 (Euclidean) norm of y : $0 \leq \|y\|_2^2 = y^H y = x^H LL^H x = x^H Ax$.
 - Furthermore, if we are hoping to solve linear systems, $\det(A) \neq 0$, meaning L^H has linearly independent columns and $y \neq 0 \implies \|y\|_2^2 > 0 \iff x^H Ax > 0 \quad \forall x \neq 0$.



In other words, A must be hermitian **positive definite**.

Hermitian Positive Definite Matrices

$$A = A^H, \quad x^H Ax > 0 \quad \forall x \neq 0$$

- If $X \in \mathbb{C}^{m \times n}$ is full rank with $m \geq n$, then A is hermitian positive definite $\implies X^H A X$ is hermitian positive definite. (Proof see Slide 19-5)
- Every diagonal of A must be strictly positive ($a_{ii} = e_i^H A e_i > 0$)
- Every “principal submatrix” is positive definite

Symmetric Gaussian Elimination

- Suppose that a hermitian positive definite matrix A happens to be in the block form:

$$A = \begin{bmatrix} 1 & \mathbf{w}^H \\ \mathbf{w} & K \end{bmatrix}.$$

- Consider the first step of Gaussian elimination:

$$A = \underbrace{\begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{w} & I \end{bmatrix}}_{L_1} \begin{bmatrix} 1 & \mathbf{w}^H \\ \mathbf{0} & K - \mathbf{w}\mathbf{w}^H \end{bmatrix}$$

- To maintain conjugate symmetry, we can use the leftmost column to eliminate \mathbf{w}^H in the top row:

$$A = \underbrace{\begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{w} & I \end{bmatrix}}_{L_1} \underbrace{\begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & K - \mathbf{w}\mathbf{w}^H \end{bmatrix}}_{L_k^H} \underbrace{\begin{bmatrix} 1 & \mathbf{w}^H \\ \mathbf{0} & I \end{bmatrix}}_{L_k^H}$$

- Now ideally we repeat, multiplying by L_k and L_k^H on the left/right until the working matrix becomes I .

Cholesky Factorization

- After m steps of symmetric Gaussian elimination, the working matrix becomes I and:

$$\underbrace{A = L_1 L_2 \dots L_m}_{L} \underbrace{L_m^H \dots L_2^H L_1^H}_{L^H}$$
 with positive real diagonal entries $L_{ii} > 0$
 - This can also be written as $A = R^H R$ with $R = L^H$
 - This process can never break down! Diagonal pivots are always strictly positive by positive definiteness.
 - Furthermore, each L_k is uniquely determined by the algorithm, making the entire factorization unique.
- Every** hermitian positive definite matrix $A \in \mathbb{C}^{m \times m}$ has a **unique** Cholesky factorization.
 - Formal proof see Slide 19-9
- Algorithm
 - Time complexity: $\frac{m^3}{3}$ operations
- Cholesky Factorization is backward stable

$$A = \begin{bmatrix} a_{11} & \mathbf{w}^H \\ \mathbf{w} & K \end{bmatrix} = \begin{bmatrix} \sqrt{a_{11}} & \mathbf{0} \\ \mathbf{w}/\sqrt{a_{11}} & I \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & K - \mathbf{w}\mathbf{w}^H/a_{11} \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & \mathbf{w}^H/\sqrt{a_{11}} \\ \mathbf{0} & I \end{bmatrix}$$

- Using Cholesky factorization, we only need to look at/operate on “half” of the input (either upper or lower triangle).
 - The other triangle is just a conjugate transpose.
 - We can avoid half the arithmetic and potentially half the storage (using a compressed representation).

Cholesky factorization for any Hermitian positive definite matrix A . [Compare to LU](#).

```
algorithm cholesky(A):
    L = A                                # We operate in-place, replacing the k'th column of A with the k'th column of L
    for k = 1 to m:
        alpha = sqrt(L(k, k))
        for i = k + 1 to m:
            beta = L(i, k) / L(k, k)      # Multiplier used for row i.
            for j = i to m:              # (Conceptually) eliminate "w" by subtracting beta * row k from row i. But only
                L(i, j) -= beta * L(k, j) #   actually modify the upper tri of the "K" block; keep "w" in the leftmost column.
            L(i, k) /= alpha           # k'th column of L is just the k'th column of the working matrix divided by alpha
            L(k, i) = 0                # Zero out k'th row of upper tri (optional if upper tri of result is ignored)
        L(k, k) = alpha
    return L, U, P
```

Algorithm

LDL^H Factorization

- What if we want to solve $A\mathbf{x} = \mathbf{b}$ for a symmetric non-positive-definite matrix A ?
 - Can we still use Cholesky?
 - No: we proved $A = LL^H$ implies A is positive definite.
 - Practically: one of the a_{ii} will become nonpositive.
 - Can we still exploit symmetry?
 - Sometimes. We can avoid the square root by leaving the a_{ii} factors on the diagonal to obtain

$$A = LDL^H$$

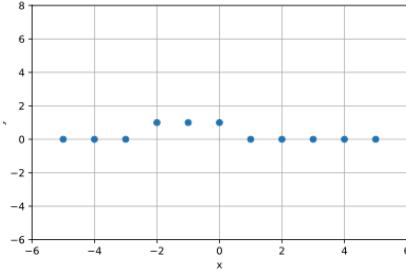
with L unit lower triangular and some entries D being zero or negative. (This of course can be done for positive definite matrices too if you want to avoid the square root.)

- However this doesn't always work for indefinite matrices (with positive and negative eigenvalues). One sufficient condition: all “leading principal minors” are nonsingular.

Lecture 20 - Least Squares

Least Squares

- Suppose we have n points $\{(x_1, y_1), \dots, (x_n, y_n)\}$ and wish to fit the polynomial



$$p(x) = a_0 + a_1x + \cdots + a_dx^d = \sum_{j=0}^d a_jx^j$$

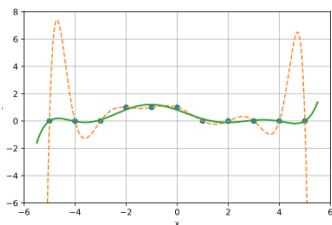
- We can find a polynomial passing through each point by solving a linear system, but this is usually a bad idea!
 - Notice that $p(x)$ is a linear combination of the “basis monomials” x^j . Evaluating at each point x_i :

$$p(x_i) = \sum_{j=0}^d a_jx_i^j = [1 \ x_i \ \dots \ x_i^d] \begin{bmatrix} a_0 \\ \vdots \\ a_d \end{bmatrix} \implies \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & \dots & x_1^d \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \dots & x_n^d \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_d \end{bmatrix}$$

- When $d = n - 1$, This is the square, nonsingular Vandermonde matrix, meaning we can solve for unique coefficients.
 - ▷ But this usually becomes a badly ill-conditioned system and produces a highly oscillatory polynomial.

Solution: Cholesky Factorization

- Cholesky is generally fastest (but not most robust) method for solving least-squares problems.
- It's much better to use a smaller degree $d < n - 1$ and **minimize the residual** of the

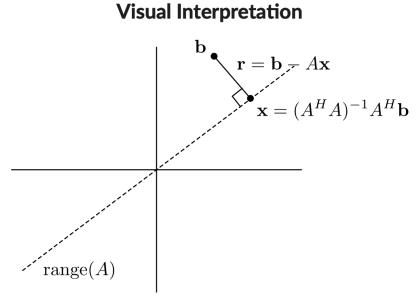


$$\min_{\mathbf{a}} \left\| \begin{bmatrix} 1 & x_1 & \dots & x_1^d \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \dots & x_n^d \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_d \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2$$

overdetermined system

- **General Statement**
 - In general, the least-squares problem seeks to solve: $Ax = b$ with rectangular $A \in \mathbb{C}^{m \times n}$, $b \in \mathbb{C}^m$, $x \in \mathbb{C}^n$, and $m > n$
 - Because there are more equations than unknowns (overdetermined), we cannot expect an exact solution.
 - Instead, we seek the best solution by minimizing the **residual** (or “error vector”): $r := b - Ax \in \mathbb{C}^m$

- The simplest, most efficient formulation is to measure \mathbf{r} using the 2-norm: $\min_x \|\mathbf{b} - A\mathbf{x}\|_2^2$
 - We seek the vector \mathbf{x} such that $A\mathbf{x}$ is the closest point in $\text{range}(A)$ to \mathbf{b} in Euclidean distance
- Solve $\min_x \|\mathbf{b} - A\mathbf{x}\|_2^2 = \min_x (\mathbf{b} - A\mathbf{x})^H(\mathbf{b} - A\mathbf{x})$ with Calculus
 - Using some matrix calculus:
$$0 = \frac{\partial (\mathbf{b} - A\mathbf{x})^H(\mathbf{b} - A\mathbf{x})}{\partial \mathbf{x}} = 2 \frac{\partial (\mathbf{b} - A\mathbf{x})^H}{\partial \mathbf{x}}(\mathbf{b} - A\mathbf{x}) = 2A^H(\mathbf{b} - A\mathbf{x}) \iff A^H A \mathbf{x} = A^H \mathbf{b}$$
- Thus, when A has **full rank** $\min(m, n) = n$, we have $A^H A$ is nonsingular
- Because A has **full rank**, we have $\forall \mathbf{x} \neq 0$, we have $A\mathbf{x} \neq 0 \implies 0 < \|A\mathbf{x}\|_2^2 = \mathbf{x}^H(A^H A)\mathbf{x}$ (why), thus $A^H A$ is hermitian positive definite
- In conclusion, when A has **full rank**, we have unique optimal $\mathbf{x} = (A^H A)^{-1} A^H \mathbf{b}$ to get $\min_x \|\mathbf{b} - A\mathbf{x}\|_2^2$
- Detail proof of uniqueness and pseudoinverse, see Slide 20.6-7
- Because the normal equations $A^H A \mathbf{x} = A^H \mathbf{b}$ involve a **positive definite** hermitian matrix $A^H A$, we can use the Cholesky factorization.



Cholesky Factorization Approach to Least Squares Problem $\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2$

1. Construct the normal equations matrix $A^H A$ and right-hand side vector $A^H \mathbf{b}$.
2. Compute the Cholesky factorization $A^H A = LL^H$.
3. Solve $L\mathbf{y} = \mathbf{b}$ with forward-substitution for \mathbf{y} .
4. Solve $L^H \mathbf{x} = \mathbf{y}$ with back-substitution for \mathbf{x} .

- Floating point operations: (dominated by building $A^H A$ and Cholesky factorization)
 - mn^2 operations to construct $A^H A$ leveraging symmetry (half the work of generic matrix-matrix mult.).
 - $\sim n^3/3$ operations to compute Cholesky factorization.
 - Total work is thus $\sim mn^2 + \frac{1}{3}n^3$ arithmetic operations.
- This is the most efficient algorithm (and the classical choice), but suffers from robustness problems.
 - Condition number $\kappa(A^H A) = \frac{\sigma_{\max}(A^H A)}{\sigma_{\min}(A^H A)} = \frac{\sigma_{\max}(A)^2}{\sigma_{\min}(A)^2} = \kappa(A)^2$ (why?) is squared, impacting accuracy.

Other Approaches

- Cholesky Factorization: the most efficient algorithm (and the classical choice), but suffers from robustness problems

- QR Factorization: more robust, and the “modern classical” approach
- SVD: even more robust than QR

Lecture 21 - QR Factorization

Motivation

- Least squares for *overdetermined systems*: “ $A\mathbf{x} = \mathbf{b}$ ” for $A \in \mathbb{C}^{m \times n}$, $m > n$.
 - Solving $\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2^2$ with the Cholesky factorization means solving the **normal equations**:

$$A^H A \mathbf{x} = A^H \mathbf{b},$$

whose condition number is as bad as $\kappa(A^H A) = \kappa(A)^2$.

◦ “Squaring” A impacts accuracy, and the QR factorization avoids this.

- *Underdetermined systems*: solving $A\mathbf{x} = \mathbf{b}$ for $A \in \mathbb{C}^{m \times n}$, $m < n$.
 - The full QR decomposition of A^H will look like:

$$A^H = [\hat{Q} \quad Q_N] \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}$$

More generally:
constructing an
orthonormal basis for
 $\text{range}(A)$ and $\mathcal{N}(A^H)$

where \hat{R} is square upper tri. and \hat{Q} and Q_N hold orthonormal bases for the rowspace and nullspace of A .

◦ The underdetermined linear system then looks like:

$$\mathbf{b} = [\hat{R}^H \quad 0] \begin{bmatrix} \hat{Q}^H \\ Q_N^H \end{bmatrix} \mathbf{x} = \hat{R}^H \tilde{\mathbf{x}} + 0 \mathbf{x}_N \quad \text{with } \tilde{\mathbf{x}} = \hat{Q}^H \mathbf{x}, \quad \mathbf{x}_N = Q_N^H \mathbf{x}$$

◦ All vectors of the form $\hat{Q} \hat{R}^{-H} \mathbf{b} + Q_N \mathbf{x}_N$ are solutions, and $\hat{Q} \hat{R}^{-H} \mathbf{b}$ has the smallest norm.

QR Factorization

- Starting points
 - Successive spaces spanned by the columns a_j of $A \in \mathbb{C}^{m \times n}$, $m \geq n$:
 $\text{span}(a_1) \subseteq \text{span}(a_1, a_2) \subseteq \text{span}(a_1, a_2, a_3) \subseteq \dots$
 - Idea: construct a sequence of orthonormal vectors q_1, q_2, \dots, q_n whose prefixes span these spaces.
 - In other words, if A has full rank n , the first j vectors should satisfy:
 $\text{span}(q_1, q_2, \dots, q_j) = \text{span}(a_1, a_2, \dots, a_j)$, $j = 1, \dots, n$, and the first j columns of A can be written as a linear combination of q_1, q_2, \dots, q_n
 - Expressing this fact in matrix form (diagram in the right)
 - When A isn't full rank, some of the r_{jj} will be 0 ($\text{span}(q_1, \dots, q_j) \supset \text{span}(a_1, \dots, a_j)$)
- Reduced QR Factorization
 - We have just shown $A = \hat{Q} \hat{R}$ with

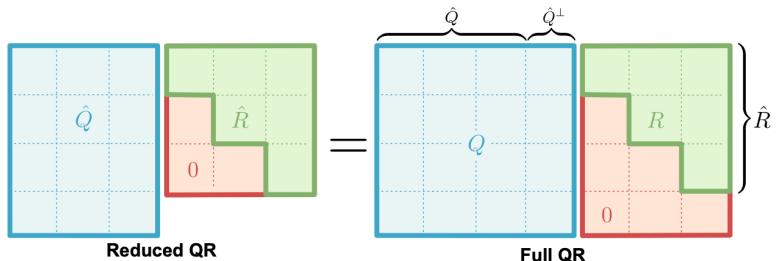
$$\underbrace{\begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{bmatrix}}_A = \underbrace{\begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_n \end{bmatrix}}_{\hat{Q}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ r_{n1} & r_{n2} & \cdots & r_{nn} \end{bmatrix}}_{\hat{R}}$$

$$\hat{Q} \in \mathbb{C}^{m \times n} \text{ and } \hat{R} \in \mathbb{C}^{n \times n}$$

- Where \hat{R} is a square upper triangular matrix and the columns of \hat{Q} hold an orthonormal basis for $\text{range}(A)$: $q_i^H q_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$
- This is called the **reduced QR** factorization, since \hat{Q} contains only a basis for $\text{range}(A)$ rather than \mathbb{C}^m . (It is rectangular rather than square)

Reduced vs. Full QR

- The reduced QR decomposition of a “tall” matrix $A \in \mathbb{C}^{m \times n}$ has “tall” \hat{Q} and square \hat{R}
- We can augment the columns of \hat{Q} with additional orthonormal vectors to form the full QR decomposition $A = QR$ with a square unitary matrix Q and a “tall” R .
- The reduced decomposition is recovered from the full decomposition by slicing off the zero rows at the bottom of R and the corresponding columns \hat{Q}^\perp
- When A is of full rank $\min(m, n) = n$
 - Columns of \hat{Q} form an orthonormal basis for $\text{range}(A)$
 - Columns of \hat{Q}^\perp form an orthonormal basis for $\text{range}(A^\perp)$ (the space of vectors **orthogonal** to all columns of A). This also the null space of A^H : $\mathcal{N}(A^H)$



Gram-Schmidt

$$\underbrace{\left[\mathbf{a}_1 \mid \mathbf{a}_2 \mid \cdots \mid \mathbf{a}_n \right]}_A = \underbrace{\left[\mathbf{q}_1 \mid \mathbf{q}_2 \mid \cdots \mid \mathbf{q}_n \right]}_{\hat{Q}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{22} & \cdots & r_{2n} \\ \vdots & & \vdots \\ r_{nn} \end{bmatrix}}_{\hat{R}}$$

- Inspecting the individual equations:

$$\begin{array}{ll} \mathbf{a}_1 = r_{11}\mathbf{q}_1 & \mathbf{q}_1 = \mathbf{a}_1 / r_{11} \\ \mathbf{a}_2 = r_{12}\mathbf{q}_1 + r_{22}\mathbf{q}_2 & \mathbf{q}_2 = (\mathbf{a}_2 - r_{12}\mathbf{q}_1) / r_{22} \\ \vdots & \vdots \\ \mathbf{a}_n = r_{1n}\mathbf{q}_1 + r_{2n}\mathbf{q}_2 + \cdots + r_{nn}\mathbf{q}_n & \mathbf{q}_n = (\mathbf{a}_n - r_{1n}\mathbf{q}_1 - r_{2n}\mathbf{q}_2 - \cdots - r_{n-1,n}\mathbf{q}_{n-1}) / r_{nn} \end{array}$$

- We can find unit vector \mathbf{q}_1 by normalizing \mathbf{a}_1 ($r_{11} = \|\mathbf{a}_1\|$).
- We can find orthogonal unit vector \mathbf{q}_2 by normalizing $\mathbf{a}_2 - r_{12}\mathbf{q}_1$, picking r_{12} to make \mathbf{q}_2 orthogonal.
- We can find orthogonal unit vector \mathbf{q}_j by making \mathbf{a}_j orthogonal to $\mathbf{q}_1, \dots, \mathbf{q}_{j-1}$:

$$\mathbf{a}_j - \underbrace{(\mathbf{q}_1^H \mathbf{a}_j) \mathbf{q}_1}_{r_{1j}} - \underbrace{(\mathbf{q}_2^H \mathbf{a}_j) \mathbf{q}_2}_{r_{2j}} - \cdots - \underbrace{(\mathbf{q}_{j-1}^H \mathbf{a}_j) \mathbf{q}_{j-1}}_{r_{j-1,j}}$$

and then normalizing. It is easy to verify that the complex dot product with $\mathbf{q}_1, \dots, \mathbf{q}_{j-1}$ is zero.

- This process is a very old idea and is known as Gram-Schmidt orthogonalization.

Existence and Uniqueness

- **Theorem:** Every $A \in \mathbb{C}^{m \times n}$ ($m \geq n$) has a full QR factorization $A = QR$ (and hence also a reduced QR factorization).
- **Theorem:** Every $A \in \mathbb{C}^{m \times n}$ ($m \geq n$) of full rank has a unique reduced QR factorization $A = \hat{Q}\hat{R}$ with positive real diagonal entries $r_{jj} > 0$.

Solving using QR Factorization

Solving $Ax = b$

- The QR factorization is a stable and easy method for solving $Ax = b$ with a nonsingular square matrix $A \in \mathbb{C}^{m \times m}$:

QR Factorization Approach to solve $Ax = b$:

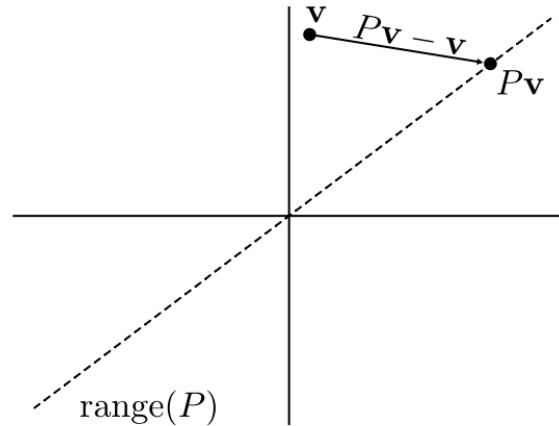
1. Compute the QR factorization $A = QR$.
2. Compute right-hand-side vector $y = Q^H b$.
3. Solve $Rx = y$ for x with back substitution.

- Generally not used because LU takes about half the work.

Lecture 22 - QR Factorization Pt2

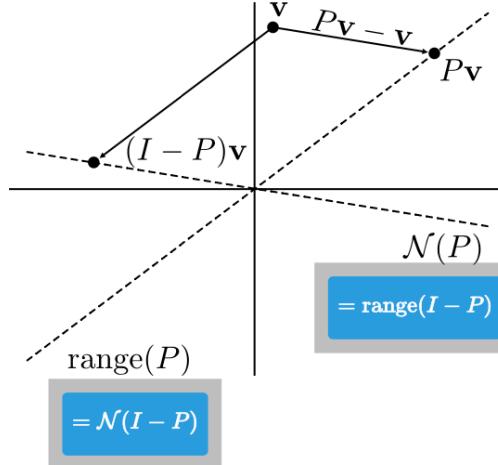
Projector Matrices

- A **projector** is a square matrix P that satisfies: $PP = P$
 - Another term: P is an idempotent matrix
 - After projecting a vector v to $x = Pv \in \text{range}(P)$, subsequent projection do nothing: $Px = PPv = Pv = x$
 - We say that P projects onto $\text{range}(P)$
 - Visually: Pv is the “shadow” cast by v when illuminated by a directional light pointing along $Pv - v$.
 - ▷ The light direction may depend on v but lies in P ’s nullspace: $P(Pv - v) = \underline{PPv} - Pv = 0$



Complementary Projectors

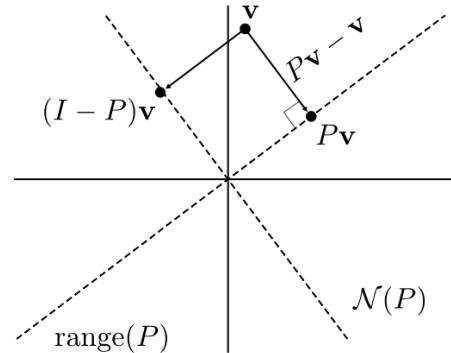
- If P is a projector, then $I-P$ is a projector as well: $(I-P)(I-P)=I-2P+PP=I-2P+P=I-P$
 - $I-P$ is called the **complementary projector** to P .
 - It projects onto the nullspace of P . Specifically, $\text{range}(I-P)=\mathcal{N}(P)$
 - Because $P=I-(I-P)$, P is the complementary projector to $(I-P)$ and thus $\text{range}(P)=\mathcal{N}(I-P)$
- Complementary subspaces: A projector partitions \mathbb{C}^m into two “disjoint”, complementary subspaces:
 - $\text{range}(P) + \text{range}(I-P) = \mathbb{C}^m$
 - $\text{range}(P) \cap \text{range}(I-P) = \{0\}$



Orthogonal Projectors

- An orthogonal projector P projects onto a subspace $\text{range}(P)=S_1$ along a subspace $\text{range}(I-P)=S_2$ where S_1 and S_2 are orthogonal.
- A projector P is orthogonal if and only if $P = P^H$
 - First direction: $P = P^H \implies$ any $\mathbf{u} \in S_1$ and $\mathbf{v} \in S_2$ are orthogonal. Note: $\mathbf{u} = P\mathbf{x}, \mathbf{v} = (I-P)\mathbf{y}$ for some \mathbf{x}, \mathbf{y} .

$$\mathbf{u}^H \mathbf{v} = \mathbf{x}^H P^H (I-P) \mathbf{y} = \mathbf{x}^H (P - PP) \mathbf{y} = 0.$$
 - Second direction: $S_1 \perp S_2 \implies P = P^H$.
 - Construct orthonormal bases $\{\mathbf{q}_1, \dots, \mathbf{q}_n\}$ for range S_1 (n -dim) and $\{\mathbf{q}_{n+1}, \dots, \mathbf{q}_m\}$ for nullspace S_2 ($(m-n)$ -dim).
 - Concatenate them into a full orthonormal basis for \mathbb{C}^m in Q :
$$Q^H P Q = Q^H \left[\begin{array}{c|c|c|c|c} \mathbf{q}_1 & \cdots & \mathbf{q}_n & \mathbf{0} & \cdots & \mathbf{0} \end{array} \right] = \left[\begin{array}{cc} I_{n \times n} & 0 \\ 0 & 0 \end{array} \right] := \Sigma$$
 - Hence $P = Q\Sigma Q^H$, which is clearly Hermitian.



QR Factorization in Projector Form

- Details see slide 22-7~11

Orthogonal Projectors with an Orthonormal Basis

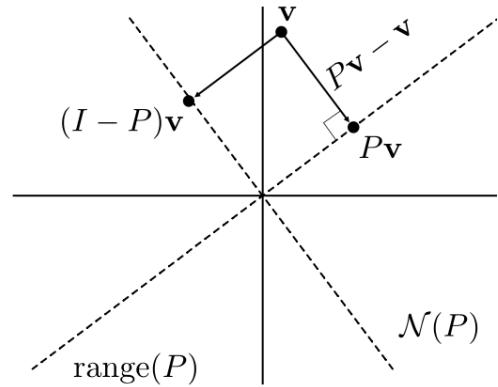
We have shown that an orthogonal projector P can be written using orthonormal bases \hat{Q} for $\text{range}(P)$ and \hat{Q}^\perp for $\mathcal{N}(P)$ simply as:

$$P = Q\Sigma Q^H = [\hat{Q} \quad \hat{Q}^\perp] \begin{bmatrix} I_{n \times n} & 0 \\ 0 & 0 \end{bmatrix} [\hat{Q} \quad \hat{Q}^\perp]^H,$$

- The \hat{Q}^\perp blocks do not contribute to P (multiplied by 0). We can drop them and conclude: $P = \hat{Q}\hat{Q}^H$.

Given any n orthonormal vectors $\{\mathbf{q}_1, \dots, \mathbf{q}_n\}$ collected in the columns of \hat{Q} , the matrix $\hat{Q}\hat{Q}^H$ represents an orthogonal projection onto $\text{range}(\hat{Q})$.

- In sum form: $\hat{Q}\hat{Q}^H = \sum_{i=1}^n \mathbf{q}_i \mathbf{q}_i^H$
- Complementary projector onto $\text{range}(\hat{Q}^\perp)$ is $I - \hat{Q}\hat{Q}^H$.
- Special case: Rank-1 orthogonal projection $P_{\mathbf{q}} = \mathbf{q}\mathbf{q}^H$ for unit-norm \mathbf{q} and its complement $P_{\perp\mathbf{q}} = I - \mathbf{q}\mathbf{q}^H$.



Lecture 23 - QR Householder Triangularization

- Gram-Schmidt vs. Householder
 - Gram-Schmidt is a “triangular orthogonalization” process: it transforms A ’s columns into \hat{Q} ’s orthonormal columns by multiplying a sequence of upper triangular matrices.
 - $\triangleright AR_1R_2\dots R_n = \hat{Q} \implies A = \hat{Q}\hat{R}$
- Householder is a “orthogonal triangularization” process: it reduces A to upper triangular form by multiplying a sequence of unitary matrices

$$\triangleright \underbrace{Q_n \dots Q_2 Q_1}_Q A = R \implies A = QR$$

- Key idea: pick **Householder reflector** matrices Q_k to zero out entries below the diagonal.
 - Q_1 operates on all rows, introducing zeros in the first column.
 - Q_2 operates on all rows after the **first**, introducing zeros in the second column.
 - Since entries 2, 3, 4 of the first column are all zeros, they **remain zero** when a linear combination is formed.

- Q_k operates on rows k to m , introducing zeros in the k -th column.

- Details see slide 23

$$\underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_A \xrightarrow{Q_1} \underbrace{\begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \\ 0 & * & * \end{bmatrix}}_{Q_1 A} \xrightarrow{Q_2} \underbrace{\begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & * \end{bmatrix}}_{Q_2 Q_1 A} \xrightarrow{Q_3} \underbrace{\begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & 0 \end{bmatrix}}_{R=Q_3 Q_2 Q_1 A}$$

Lecture 24 - Eigenvalues and Eigenvectors

Eigenvalue and Eigenvector

- Definition: Given a square matrix $A \in \mathbb{C}^{m \times m}$ a nonzero vector $\mathbf{x} \in \mathbb{C}^m$ is an **eigenvector**, and $\lambda \in \mathbb{C}$ is its corresponding **eigenvalue**, if: $A\mathbf{x} = \lambda\mathbf{x}$
 - Why only for square matrices? $A\mathbf{x}$ must live in the same space as \mathbf{x} .
 - Intuitively: \mathbf{A} acts just like scalar multiplication on certain subspaces $S \subseteq \mathbb{C}^m$
 - Intuitive video
 - These are called eigenspaces of \mathbf{A}
 - Any nonzero vector $\mathbf{x} \in S$ is an eigenvector with the same eigenvalue.
 - These S are generally 1-dimensional, but they can be higher dimension (example: αI)
 - The set of all eigenvalues of \mathbf{A} is called the matrix **spectrum** $\Lambda(A)$
-

Eigenvalue Decomposition

- Given a square matrix $A \in \mathbb{C}^{m \times m}$, an eigenvalue decomposition is a factorization: $A = X\Lambda X^{-1} \iff AX = X\Lambda$ where X is a nonsingular matrix.

$$A \begin{bmatrix} \mathbf{x}_1 & | & \mathbf{x}_2 & | & \cdots & | & \mathbf{x}_m \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 & | & \mathbf{x}_2 & | & \cdots & | & \mathbf{x}_m \end{bmatrix} \begin{bmatrix} \lambda_1 & & & & & & \\ & \lambda_2 & & & & & \\ & & \ddots & & & & \\ & & & & & & \lambda_m \end{bmatrix}$$

- Does not always exist!
 - Comparison of Eigenvalue Decomposition and Singular Value Decomposition:
 - SVD always exists!
 - Singular values are positive and real, while eigenvalues can be general complex numbers.
 - Singular vectors are orthogonal, while eigenvectors generally are not.
 - Left and right singular vectors can be completely decoupled; Right eigenvectors (rows of X^{-1}) are orthogonal to corresponding left eigenvector (columns of X).
-

Eigenvector Basis

- If $A\mathbf{x} = b$ and $A = X\Lambda X^{-1}$ then:

$$b = X\Lambda X^{-1}\mathbf{x} \iff (X^{-1}b) = \Lambda(X^{-1}\mathbf{x})$$

- $X^{-1}b$ and $X^{-1}\mathbf{x}$ are the coefficients of \mathbf{b} and \mathbf{x} in the eigenvector basis.
- They express \mathbf{b} and \mathbf{x} as linear combinations of the columns of \mathbf{X}
- After this change of basis, the multiplication by A simply scales each coefficient by the corresponding λ_i
- An **eigenbasis** is a basis of \mathbb{R}^n consisting of eigenvectors of A .
- A 's inverse (if it exists) has the same eigenvectors and reciprocal eigenvalues:

$$A^{-1} = (X\Lambda X^{-1})^{-1} = X\Lambda^{-1}X^{-1}$$

- We can solve the linear system $Ax = b$ easily in the eigenvector basis:
 $(X^{-1}\mathbf{x}) = \Lambda^{-1}(X^{-1}b)$, just dividing each coefficient of B by the corresponding λ_i

Characteristic Polynomial

- By definition: λ is an eigenvalue of $A \in \mathbb{C}^{m \times m}$ if and only if:
 $A\mathbf{x} = \lambda\mathbf{x}$ for some nonzero $\mathbf{x} \iff (\lambda I - A)\mathbf{x} = 0 \iff \det(\lambda I - A) = 0$
- The **characteristic polynomial** of A is the degree m polynomial defined by:
 $p_A(z) = \det(zI - A)$, where the sign was chosen so that the leading term is always z^m rather than $(-z)^m$.
- In fact, you can show that $p_A(z) = z^m - \text{tr}(A)z^{m-1} + \dots$
- λ is an eigenvalue if and only if it is a root of the characteristic polynomial: $p_A(\lambda) = 0$
 $\det(\begin{bmatrix} \lambda - a_{11} & a_{12} \\ a_{21} & \lambda - a_{22} \end{bmatrix}) = 0 \iff (\lambda - a_{11})(\lambda - a_{22}) - a_{12}a_{21} = \lambda^2 - (a_{11} + a_{22})\lambda + a_{11}a_{22} - a_{12}a_{21} = 0$

Algebraic Multiplicity

- Eigenvalues (roots of p_A) are not necessarily distinct. An eigenvalue λ_j may appear more than once in $\Lambda(A)$.
- The **algebraic multiplicity** of an eigenvalue λ of A is its multiplicity as a root of p_A . An eigenvalue is called **simple** if it is of algebraic multiplicity 1.
 - $A \in \mathbb{C}^{m \times m}$ has exactly m eigenvalues when counted with multiplicity; it will have distinct eigenvalues iff all eigenvalues are simple.
 - Every matrix has at least one eigenvalue.
 Some matrices have only one (distinct) eigenvalue: e.g. αI

Eigenspaces and Geometric Multiplicity

- The set of eigenvectors corresponding to a single eigenvalue λ , together with the zero vector, forms a subspace $E_\lambda \subseteq \mathbb{C}^m$ called an **eigenspace**.

- The dimension of the eigenspace, $\dim(E_\lambda)$, is the number of linearly independent eigenvectors with eigenvalue λ .
 - The dimension is known as the **geometric multiplicity** of λ .
 - Geometric multiplicity is always **less than or equal to** algebraic multiplicity. (Proof see slide 24-14)
 - **Defective matrix** has a **defective eigenvalue** whose algebraic multiplicity exceeds its geometric multiplicity.
-

Similarity Transformation

- Let $A, B, Y \in \mathbb{C}^{m \times m}$ and suppose Y is nonsingular. Then A and B are said to be **similar** if:

$$B = Y^{-1}AY,$$

And transformation $A \mapsto Y^{-1}AY$ is called a **similarity transformation**.

- Furthermore, A and B have the same characteristic polynomial, eigenvalues, and algebraic/geometric multiplicity. (Proof see slide 24-13)
-

Diagonalizability

- A matrix $A \in \mathbb{C}^{m \times m}$ has an eigenvalue decomposition $A = X\Lambda X^{-1}$ if and only if it is non defective (geometric multiplicity=algebraic multiplicity for all eigenvalues)
-

Determinant and Trace

- $\det(A) = \prod_{j=1}^m \lambda_j$
 - $\text{tr}(A) = \sum_{j=1}^m \lambda_j$
 - Proof see slide 24-16
-

Unitary Diagonalizability

- A matrix $A \in \mathbb{C}^{m \times m}$ is called **unitarily diagonalizable** if there exists a unitary matrix Q such that: $A = Q\Lambda Q^H$
- These special eigenvalue decompositions are also SVDs (up to signs/ordering of Λ).
- Hermitian matrices are unitarily diagonalizable and have all real eigenvalues!
- A is unitarily **diagonalizable** if and only if $A^H A = A A^H$. Matrices satisfying this are called **normal**.

Schur Factorization

- A very important factorization forming the basis of eigenvalue algorithms:

A **Shur factorization** of a matrix $A \in \mathbb{C}^{m \times m}$ is a factorization:

$$A = QTQ^H$$

where Q is unitary and T is upper-triangular.

- Because A and T are similar, $\Lambda(A) = \Lambda(T) = t_{ii}$.
- We therefore call the Shur Factorization an **eigenvalue revealing** factorization.
- If A is Hermitian:

$$A^H = (QTQ^H)^H = QT^HQ^H = A = QTQ^H$$

- Thus $T = T^H$ and T must actually be a diagonal matrix with **real** entries.
- In other words A is unitarily diagonalizable and has real eigenvalues!

- **Every** square matrix has a Schur factorization!

(Even defective ones—this is why the factorization is so powerful.)

Schur Factorization

Lecture 25~26 - Eigenvalue Solvers

Eigenvalue Algorithms

- Algorithms for getting all eigenvalues are generally based on computing a Schur factorization $A = QTQ^H$.
 - When A is Hermitian, the Schur factorization is really the eigenvalue decomposition (eigenvectors are the columns of Q).
 - When A is not Hermitian, the eigenvalues are still the diagonal entries t_{ii} .
 - ▷ We don't get the eigenvectors directly in this case.
 - ▷ However, we will find there are fast algorithms for finding the eigenvectors once you know the eigenvalues.
- A fundamental difficulty: uncomputability!
 - It is actually impossible to compute the eigenvalues (or the Schur factorization) of a general matrix of size in a finite number of arithmetic steps.

- Proof: a reduction from computing the roots of degree polynomials, which is known to be impossible for certain polynomials of degree in a finite number of arithmetic steps (Galois theory). *Proof see slide 25-7*
- Thus, eigenvalue solvers fundamentally must be **iterative**.

Iterative Eigenvalue Solvers

- We can still develop extremely effective, fast eigenvalue solvers.
 - They simply must be iterative: produce a sequence of better and better approximations to the solution.
 - ▷ This is fine since we cannot represent the exact solutions in floating point anyway!
 - ▷ Stop when we determine the error is on the order of the machine epsilon (or sooner if less accuracy is needed).
 - The best algorithms for individual eigenvalues converge **cubically** in most cases, **quadratically** worst-case.
 - ▷ Cubically: the number of correct digits in the eigenvalue estimates triples with each iteration.
 - ▷ Quadratically: the number of correct digits doubles with each iteration.
- Most eigenvalue solvers used today approximate a Schur factorization by iteratively applying a sequence of unitary similarity transformations $X \rightarrow Q_j^H X Q_j$ to obtain:

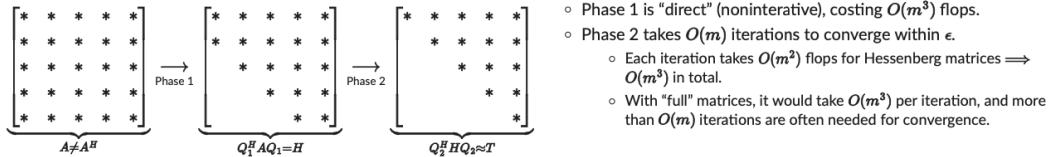
$$A_j := \underbrace{Q_j^H \cdots Q_2^H Q_1^H}_Q A \underbrace{Q_1 Q_2 \cdots Q_j}_Q$$

where A_j approaches an upper-triangular matrix T as $j \rightarrow \infty$.

- Remember: even if A is real, these Schur factors can be complex!
- If $A = A^H$, then T_j converges to a real diagonal matrix.
- Two Phases Iterative Eigenvalue Solvers

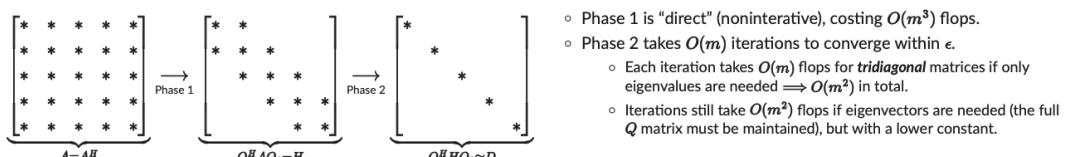
- The computations needed in each iteration will be expensive for general “full” matrices.
 - It is worthwhile to first transform A into a nearly-triangular matrix, an **upper Hessenberg matrix**.
 - This “first phase” of the algorithm is similar to Householder QR, but it must apply a similarity transform.

- In the non-Hermitian case:



- Phase 1 is “direct” (noniterative), costing $O(m^3)$ flops.
- Phase 2 takes $O(m)$ iterations to converge within ϵ .
 - Each iteration takes $O(m^2)$ flops for Hessenberg matrices $\Rightarrow O(m^3)$ in total.
 - With “full” matrices, it would take $O(m^3)$ per iteration, and more than $O(m)$ iterations are often needed for convergence.

- In the Hermitian case:



- Phase 1 is “direct” (noniterative), costing $O(m^3)$ flops.
- Phase 2 takes $O(m)$ iterations to converge within ϵ .
 - Each iteration takes $O(m)$ flops for **tridiagonal** matrices if only eigenvalues are needed $\Rightarrow O(m^2)$ in total.
 - Iterations still take $O(m^2)$ flops if eigenvectors are needed (the full Q matrix must be maintained), but with a lower constant.

Phase 1: Reduce to Hessenberg matrix

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \xrightarrow{\text{Phase 1}} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}$$

$A \neq A^H$ $Q^H A Q = H$

- Idea: use Householder reflections to zero out entries below **sub**-diagonal via similarity transforms.
- Why can't we just zero out all elements below the diagonal and obtain T directly?
 - The uncomputability theorem says this is impossible!
 - If we try, applying the Householder transformation also on the right (needed for similarity transform) undoes progress:
 - Multiplying by Q_1 on the right replaces all columns by a linear combination of all columns, destroying the newly inserted zeros.
 - However, the new entries (red) are typically smaller than before. This will form the basis of the QR algorithm to be studied later.
- Detail see slide 25. 10~15

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \xrightarrow{Q_1^H \cdot} \begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{bmatrix} \xrightarrow{\cdot Q_1} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}$$

A $Q_1^H A$ $Q_1^H A Q_1$

Phase 2: Power Iteration

- In a word: 不停地用矩阵A乘以向量v, $A^k v$ 会收敛到对应最大特征值的特征向量。
- Let $A = Q \Lambda Q^T$ with $\Lambda = \text{diag}(\lambda_i)$ sorted $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_m|$
- Consider an arbitrary vector $v \in \mathbb{R}^m$ decomposed in A's eigenbasis:
 $v = a_1 q_1 + a_2 q_2 + \dots + a_m q_m$
- we have

$$A^k v = \lambda_1^k (a_1 q_1 + a_2 (\frac{\lambda_2}{\lambda_1})^k q_2 + \dots + a_m (\frac{\lambda_m}{\lambda_1})^k q_m)$$

- If $a_1 \neq 0$, $v_k := \frac{A^k v}{\|A^k v\|}$ approaches $\pm q_1$ since $\frac{\lambda_i}{\lambda_1}$ approaches 0 as $k \rightarrow \infty$

- ▷ Thus, repeatedly applying A gives us an **estimate of the eigenvector** corresponding to the largest magnitude eigenvalue
- ▷ What if $a_1 = 0$? Highly unlikely if \mathbf{v} is chosen randomly!
- What about an eigenvalue estimate?

$$v_k := \frac{A^k v}{\|A^k v\|} \approx \pm q_1 \implies \lambda_1 = q_1^T A q_1 \approx v_k^T A v_k \text{ (Rayleigh Quotient)}$$

- Convergence:

- The relative eigenvector error is: $\|(\pm v_k) - q_1\| := \|\delta q_1\| = O(|\frac{\lambda_1}{\lambda_2}|)$ as $k \rightarrow \infty$
- The component orthogonal to q_1 gets shrunk by factor $|\frac{\lambda_1}{\lambda_2}|$ at each iteration. (Linear convergence)
- In a word: too slow

Inverse Iteration & Rayleigh Quotient Iteration

- In a word
 - Inverse Iteration: 对 $(A - \mu I)^{-1}$ 使用power iteration可以更快地收敛到 q_j 和 λ_j
 - Rayleigh Quotient Iteration: 对 $(A - \lambda I)^{-1}$ 使用power iteration, 更快收敛

Inverse Iteration

- Suppose A has eigenvalues $\{\lambda_j\}$
- If $\mu \notin \{\lambda_j\}$ is not exactly an eigenvalue of A , then

$$\begin{aligned} (A - \mu I)^{-1} &= (Q \Lambda Q^\top - \mu I)^{-1} = (Q \Lambda Q^\top - \mu Q Q^\top)^{-1} \\ &= (Q (\Lambda - \mu I) Q^\top)^{-1} = (Q \operatorname{diag}(\lambda_j - \mu) Q^\top)^{-1} \\ &= Q \operatorname{diag}(\lambda_j - \mu)^{-1} Q^\top, \end{aligned}$$

has the same eigenvectors as A and eigenvalues $\left\{ \frac{1}{\lambda_j - \mu} \right\}$.

- By picking μ very close to eigenvalue λ_J , we expect $|(\lambda_J - \mu)^{-1}|$ to be much larger than $|(\lambda_j - \mu)^{-1}|$ for all $\lambda_j \neq \lambda_J$.
- Hence, if we apply power iteration to $(A - \mu I)^{-1}$ we expect it to converge quickly to q_J .

```
algorithm inverse_power_iteration(A, μ):
    v = arbitrary unit-norm vector
    for k = 1, 2, ...
        v = normalize((A - μ I) \ v)
        λ = dot(v, A * v)
```

- The operation $(A - \mu I) \setminus v$ means solve the linear system $(A - \mu I)x = v$.
 - Since the matrix doesn't change, we can factorize $(A - \mu I)$ once and reuse it.
 - If $\mu \approx \lambda_J$, $A - \mu I$ is ill-conditioned (nearly singular), but actually this is fine!

- Quicker convergence

Lecture 27 - QR Algorithm for Eigenvalues

QR Algorithm

- An algorithm to compute the **entire** eigenvalue decomposition of a symmetric real matrix.
- The most basic form

```
algorithm pure_qr_algorithm(A):
    for k = 1, 2, ...:
        Q R = A      # Compute the QR factorization of A
        A = R Q      # Recombine the factors in the reverse order.
```

- For it to work, we must have two properties:

1. The working matrix remains similar to the input matrix with every iteration.

$$\begin{aligned} \triangleright Q_k R_k &= A_{k-1} \\ A_k &= R_k Q_k \end{aligned} \implies Q_k A_k = Q_k (R_k Q_k) = A_{k-1} Q_k \implies A_k = Q_k^{-1} A_{k-1} Q_k$$

2. The working matrix converges to a diagonal matrix.

$$\begin{aligned} \triangleright Q_k R_k &= A_{k-1} \\ A_k &= R_k Q_k \end{aligned} \implies Q_k R_k = A_k = R_{k-1} Q_{k-1}$$

$$\begin{aligned} A &= Q_1 R_1 \\ A^2 &= (Q_1 \textcolor{red}{R}_1)(\textcolor{red}{Q}_1 R_1) = Q_1 \textcolor{green}{Q}_2 \textcolor{red}{R}_2 R_1 \\ A^3 &= (Q_1 \textcolor{red}{R}_1)(\textcolor{red}{Q}_1 Q_2 R_2 R_1) = Q_1 \textcolor{green}{Q}_2 \textcolor{red}{R}_2 \textcolor{green}{Q}_2 R_2 R_1 = Q_1 Q_2 \textcolor{green}{Q}_3 \textcolor{red}{R}_3 R_2 R_1 \\ &\vdots \\ A^k &= Q_1 \textcolor{red}{R}_1 (\textcolor{red}{Q}_1 \cdots Q_{k-1} R_{k-1} \cdots R_1) = \cdots = Q_1 Q_2 \cdots \textcolor{green}{Q}_k \textcolor{red}{R}_k \cdots R_1 \end{aligned}$$

- Conclusion: It's a QR factorization of A^k where $\underline{Q}_k := Q_1 Q_2 \dots Q_k$, $\underline{R}_k := R_k \cdots R_1$

Block Power Iteration

- To better understand how the QR factorization $A^k = \underline{Q}_k \underline{R}_k$ is helpful, consider applying power iteration to several vectors at once:

- Suppose we start with a set of m linearly independent vectors $\mathbf{v}_1, \dots, \mathbf{v}_m$ and apply A^k
- We have argued that the ordinary power iteration $A^k \mathbf{v}$ converges to a vector parallel to \mathbf{q}_1 :

$$\begin{aligned} A^k \mathbf{v}_1 &= \lambda_1^k (a_{11} \mathbf{q}_1 + a_{21}(\lambda_2/\lambda_1)^k \mathbf{q}_2 + \dots + a_{m1}(\lambda_m/\lambda_1)^k \mathbf{q}_m) \\ A^k \mathbf{v}_2 &= \lambda_1^k (a_{12} \mathbf{q}_1 + a_{22}(\lambda_2/\lambda_1)^k \mathbf{q}_2 + \dots + a_{m2}(\lambda_m/\lambda_1)^k \mathbf{q}_m) \\ &\dots \end{aligned}$$

as long as the initial component along \mathbf{q}_1 , $a_{1*} \neq 0$. Apparently all vectors will converge to be parallel to \mathbf{q}_1 \circledast .

- But what if we **force** $a_{12} = 0$? Then the \mathbf{q}_2 term must dominate (assuming $|\lambda_2| > |\lambda_3|$):

$$A^k \mathbf{v}_2 = \lambda_2^k (a_{22} \mathbf{q}_2 + a_{32}(\lambda_3/\lambda_2)^k \mathbf{q}_3 + \dots + a_{m2}(\lambda_m/\lambda_2)^k \mathbf{q}_m),$$

so that $\frac{A^k \mathbf{v}_2}{\|A^k \mathbf{v}_2\|} \rightarrow \pm \mathbf{q}_2$ linearly as $k \rightarrow \infty$, with error $O\left(\frac{|\lambda_3|}{|\lambda_2|}\right)$.

We can apply this same projection process to make

$$\frac{\|A^k \mathbf{v}_3\|}{\|A^k \mathbf{v}_2\|} \rightarrow \pm \mathbf{q}_3, \dots, \frac{\|A^k \mathbf{v}_m\|}{\|A^k \mathbf{v}_2\|} \rightarrow \pm \mathbf{q}_m \text{ as } k \rightarrow \infty.$$

- We can force this by projecting out the component of \mathbf{v}_2 along \mathbf{q}_1 $\approx \frac{A^k \mathbf{v}_2}{\|A^k \mathbf{v}_2\|}$ at the start.

- In exact arithmetic, this is equivalent to projecting out $A^k \mathbf{v}_2$'s component along \mathbf{q}_1 (both simply zero out the a_{12} term).

This is exactly what the QR factorization does: it projects out the second column's component along the first!

5

- ▶ According to above proof, the column of \underline{Q}_k in $A^k = \underline{Q}_k \underline{R}_k$ will converge linearly to the eigenvectors, and A^k converges to Λ .

Accelerating the QR Algorithm (Inverse & Rayleigh Quotient Iteration)

- Details see slide 27.8~11

Review

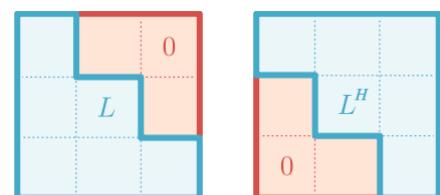
Factorization

LU Factorization

- Definition: Factor a matrix A into the form: $A=LU$, where L is lower triangular and U is upper triangular.
- Existence:
 - Without pivoting, might not exist
 - With partial pivoting, we can always construct LUP decomposition
- What do they reveal?
- What can they be used for?
- What algorithms exits for computing them?
 - Gaussian elimination, $A=LU$
 - ▷ neither backward stable nor stable, why: 0 on the diagonal)
 - Complete pivoting, $PAQ=LU$ (P, Q are permutation matrix)
 - ▷ pick the largest x_{ij} in submatrix $i, j \geq k$, reordering rows and columns
 - ▷ Backward stable
 - Partial pivoting, $PA=LU$ (P are permutation matrix)
 - ▷ pick the largest x_{ik} in column k ($i \geq k$), only reorder rows
 - ▷ Backward stable

Cholesky Factorization

- $A = LL^H$ or $A = RR^H$ with positive real diagonal entries $L_{ii} > 0$
- Existence: A must be hermitian positive definite matrix
- Uniqueness: **Every** hermitian positive definite matrix $A \in \mathbb{C}^{m \times m}$ has a **unique** Cholesky factorization.
- What do they reveal?
- What can they be used for?
- What algorithms exits for computing them?
 - Symmetric Gaussian Elimination

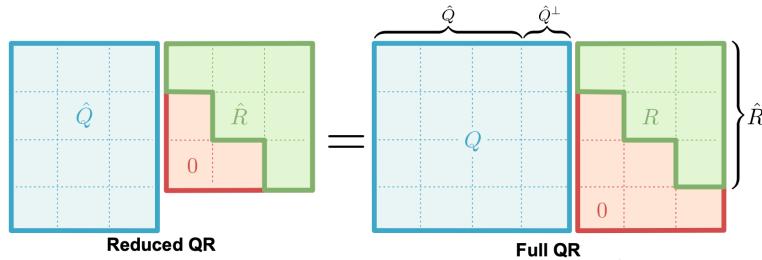


- ▶ Backward stable (No pivoting needed)

QR Factorization

- Reduced QR: $A = \hat{Q}\hat{R}$ with $\hat{Q} \in \mathbb{C}^{m \times n}$ and $\hat{R} \in \mathbb{C}^{n \times n}$, where \hat{R} is a square upper triangular matrix and the columns of \hat{Q} hold an orthonormal basis for $\text{range}(A)$:

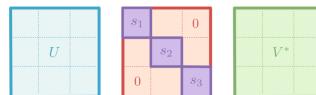
$$q_i^H q_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$
- Full QR: We can augment the columns of \hat{Q} with additional orthonormal vectors to form the full QR decomposition with a square unitary matrix Q and a “tall” R .



- Existence: *always exists*
- Uniqueness: **Reduced QR** is unique if we require R 's diagonal to be real and positive.
- What do they reveal?
- What can they be used for?
- What algorithms exist for computing them?
 - Gram-Schmidt (unstable, q_j quickly loses orthogonality due to round off error)
 - Householder (backward stable)

Singular Value Decomposition (SVD)

The Singular Value Decomposition



Given $A \in \mathbb{C}^{m \times n}$, a **singular value decomposition (SVD)** of A is a factorization:

$$A = U\Sigma V^H \quad \text{where} \quad \begin{cases} U \in \mathbb{C}^{m \times m} & \text{is unitary,} \\ \Sigma \in \mathbb{C}^{m \times n} & \text{is diagonal,} \\ V \in \mathbb{C}^{n \times n} & \text{is unitary.} \end{cases}$$

Following the standard convention, we assume the **nonnegative** diagonal entries σ_j of Σ are ordered $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$, where $p = \min(m, n)$.

- Note: the columns of V and U form an *orthonormal basis* for the input and output spaces \mathbb{C}^n and \mathbb{C}^m , respectively.
 - Columns of U are called “left singular vectors” \mathbf{u}_j and columns of V are called “right singular vectors” \mathbf{v}_j .

- Given $A \in \mathbb{C}^{m \times n}$, a singular value decomposition of A is a factorization: $A = U \Sigma V^H$, where $U \in \mathbb{C}^{m \times m}$ is unitary, $\Sigma \in \mathbb{C}^{m \times n}$ is diagonal, and $V \in \mathbb{C}^{n \times n}$ is unitary.
- Existence: *always exists*
- What do they reveal?
 - determinants when $m=n$
 - Easily compute the inverse
 - Prove row rank = column rank
 - Reveal normal bases for the nullspace and range of A
- Not backward stable. SVD algorithms are doomed to fail backward stability since an arbitrarily small roundoff error can make \hat{U} and \hat{V} non-unitary: in this case, they cannot be an exact SVD of any matrix.

Schur Factorization

- Given a square matrix $A \in \mathbb{C}^{m \times m}$, a Schur Factorization is a factorization: $A = QTQ^H$ where Q is a unitary matrix and T is an upper triangular.
- Existence: always exists

Eigenvalue Decomposition

- Given a square matrix $A \in \mathbb{C}^{m \times m}$, an eigenvalue decomposition is a factorization: $A = X\Lambda X^{-1} \iff AX = X\Lambda$ where X is a nonsingular matrix.
- Existence: *not always exists*
- Computing Eigenvalues?
 - “Direct” reduction to upper Hessenberg (similar to householder)
 - Iterative reduction to upper triangular (power iteration)
- Computing full eigenvalue decomposition of a symmetric real matrix A
 - QR algorithm

Solving Linear System

$$Ax=b$$

Solving Least Squares

$$\min_x \|Ax - b\|_2 \text{ for } A \in \mathbb{C}^{m \times n}, m \geq n$$

- Cholesky (normal equations)

- $A^H A x = A^H b, \quad A^H A = LL^H$

- Reduced QR

$$A = \hat{Q} \hat{R}, Ax = b \implies \underbrace{\hat{R}^H \hat{Q}^H}_{A^H} \underbrace{\hat{Q} \hat{R}}_A x = \underbrace{\hat{R}^H \hat{Q}^H}_{A^H} b \implies \hat{R}x = \hat{Q}^H b$$

- Reduced SVD

$$A = \hat{U} \hat{\Sigma} V^H, Ax = b \implies \underbrace{V \hat{\Sigma} \hat{U}^H}_{A^H} \underbrace{\hat{U} \hat{\Sigma} V^H}_A x = \underbrace{V \hat{\Sigma} \hat{U}^H}_{A^H} b \implies x = V \hat{\Sigma}^{-1} \hat{U}^H b$$

Lemma

Triangular Matrix

- The product of two upper/lower triangular matrix is still upper/lower triangular. (HW0 - Prob 5)
 - The inverse of upper/lower triangular matrix, if it exists, is still upper/lower triangular. (HW0 - Prob 6)
-

Orthogonal Square Matrix

- $A^T = A^{-1}$
 - $AA^T = I$
 - A^T is orthogonal
 - $\det(A) = \pm 1$ (Proof: $\det(I) = \det(A^T A) = \det(A^T)\det(A) = 1$)
-

Rank

- Rank-One: The rank of a matrix $A \in \mathbb{R}^{m \times n}$ equals 1 if and only if it can be written in the form xy^T . (HW0 - Prob 3)
 - Full-Rank: A matrix $A \in \mathbb{F}^{m \times n}$ with $m \geq n$ has full rank n iff $Ax \neq Ay, \forall x \neq y \in \mathbb{F}^n$
 - $\implies \forall x \neq 0$, we have $Ax \neq 0$
 - $\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B))$
-

Determinant

- $\det(A) = 0$ if A's columns are linearly dependent
- $\det(I) = 1$
- $\det(\text{triangular matrix}) = \prod_{i=1}^m a_{ii}$
- $\det(\alpha A) = \alpha^m \det(A)$
- $\det(A^T) = \det(A)$
- $\det(AB) = \det(A)\det(B)$
- $\det(A^{-1}) = \frac{1}{\det(A)}$

Inverse Matrix

- **Theorem:** the following conditions for square matrix $A \in \mathbb{F}^{m \times m}$ are equivalent
 - A has an inverse A^{-1}
 - A is a nonsingular matrix
 - $\det(A) \neq 0$ (if $\det(A)=0$, then the transformation squishes into lower dimension)
 - A is of full rank m
 - $\text{range}(A) = \mathbb{F}^m$
 - $\mathbb{N}(A) = \{\mathbf{0}\}$
 - 0 is not an eigenvalue of A
 - 0 is not a singular value of A
-

Positive Definite

- A hermitian matrix (i.e. $A = A^H$) is positive definite if $x^H A x > 0 \quad \forall x \neq 0$