Lectures on Numerical Linear Algebra

Yunkai Zhou

Department of Mathematics
Southern Methodist University
Dallas, Texas 75075

yzhou@smu.edu

Spring, 2011

Acknowledgements

The lecture slides benefit from the following resources

- Prof. Per-Olof Persson's course materials for MIT 18.335 (Thanks Per-Olof for the tex files of slides)
- Prof. Yousef Saad's book on iterative methods (Thanks Yousef for the tex files of course materials)
- Several well-known textbooks on NLA by J. Demmel, Trefethen and Bau, G. W. Stewart, C. D. Meyer, Golub and Van Loan
- Several other books on matrix analysis and matrix computations
- Several books/papers on theory and applications of LA
- The Wikipedia website
- The open source software: Linux, LaTeX, beamer, pstricks

The instructor greatly acknowledges NSF for grants CMMI-0727194 and OCI-0749074.

Basic Linear Algebra

- Spaces: $\mathbb{R}^n, \mathbb{C}^n$, $\mathbb{R}^{n\times n}, \mathbb{C}^{n\times n}$, $\mathbb{R}^{m\times n}, \mathbb{C}^{m\times n}$
  (by default, $\mathbb{R}^n = \mathbb{R}^{n\times 1}$, $\mathbb{C}^n = \mathbb{C}^{n\times 1}$)

  (Real: $\mathbb{R}^n, \mathbb{R}^{n\times n}, \mathbb{R}^{m\times n}$; Complex: $\mathbb{C}^n, \mathbb{C}^{n\times n}, \mathbb{C}^{m\times n}$)

Basic Linear Algebra

- Spaces: $\mathbb{R}^n, \mathbb{C}^n$, $\mathbb{R}^{n\times n}, \mathbb{C}^{n\times n}$, $\mathbb{R}^{m\times n}, \mathbb{C}^{m\times n}$
  (by default, $\mathbb{R}^n = \mathbb{R}^{n\times 1}$, $\mathbb{C}^n = \mathbb{C}^{n\times 1}$)

  (Real: $\mathbb{R}^n, \mathbb{R}^{n\times n}, \mathbb{R}^{m\times n}$; Complex: $\mathbb{C}^n, \mathbb{C}^{n\times n}, \mathbb{C}^{m\times n}$)
- Vectors:
  $v \in \mathbb{R}^n$ (length-n column real vector)
  $v \in \mathbb{C}^n$ (length-n column complex vector)
  $w \in \mathbb{R}^{1\times n}$ (length-n row real vector)
  $w \in \mathbb{C}^{1\times n}$ (length-n row complex vector)

  (We use column vector as the default, so a vector means a column vector)

Basic Linear Algebra

- Spaces: $\mathbb{R}^n, \mathbb{C}^n$, $\mathbb{R}^{n\times n}, \mathbb{C}^{n\times n}$, $\mathbb{R}^{m\times n}, \mathbb{C}^{m\times n}$
  (by default, $\mathbb{R}^n = \mathbb{R}^{n\times 1}$, $\mathbb{C}^n = \mathbb{C}^{n\times 1}$)

  (Real: $\mathbb{R}^n, \mathbb{R}^{n\times n}, \mathbb{R}^{m\times n}$; Complex: $\mathbb{C}^n, \mathbb{C}^{n\times n}, \mathbb{C}^{m\times n}$)
- Vectors:
  $v \in \mathbb{R}^n$ (length-n column real vector)
  $v \in \mathbb{C}^n$ (length-n column complex vector)
  $w \in \mathbb{R}^{1\times n}$ (length-n row real vector)
  $w \in \mathbb{C}^{1\times n}$ (length-n row complex vector)

  (We use column vector as the default, so a vector means a column vector)
- Special vectors:
  Length-n basis vector: e_i
  e_i: (all elements equal to 0 except the i-th element equals to 1)

  Length-n vector of all-ones: $\mathbf{1} = \underbrace{[1, 1, \cdots, 1]^\mathsf{T}}_{n} = \sum_{i=1}^{n} e_i$

Basic Linear Algebra

- Matrices: (element-wise)
  An $m \times n$ matrix $A \in \mathbb{R}^{m\times n}$ (or $A \in \mathbb{C}^{m\times n}$)

$$A = \left[a_{i,j}\right]$$

where $\quad a_{i,j} \in \mathbb{R}$ (or $\mathbb{C}$), $\;i = 1, 2, \ldots, m, \;\; j = 1, 2, \ldots, n.$

Basic Linear Algebra

- Matrices: (element-wise)
  An $m \times n$ matrix $A \in \mathbb{R}^{m\times n}$ (or $A \in \mathbb{C}^{m\times n}$)

$$A = \left[a_{i,j}\right]$$

  where $\quad a_{i,j} \in \mathbb{R}$ (or $\mathbb{C}$), $\;i = 1, 2, \ldots, m, \;\; j = 1, 2, \ldots, n.$

- Matrices: (vector-wise)
  An $m \times n$ matrix $A \in \mathbb{R}^{m\times n}$ (or $A \in \mathbb{C}^{m\times n}$)

$$A = \left[a_1, a_2, \cdots, a_n\right]$$

  where $\quad a_i \in \mathbb{R}^m$ (or $\mathbb{C}^m$), $\quad i = 1, 2, \ldots, n.$

Basic Linear Algebra

- Transpose:

$$A = [a_{i,j}]_{m\times n} \iff A^{\mathsf{T}} = [a_{j,i}]_{n\times m}$$

Example:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \iff A^{\mathsf{T}} = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \end{bmatrix}$$

Basic Linear Algebra

- Transpose:

$$A = [a_{i,j}]_{m\times n} \iff A^{\mathsf{T}} = [a_{j,i}]_{n\times m}$$

Example:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \iff A^{\mathsf{T}} = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \end{bmatrix}$$

- Adjoint (conjugate transpose) :

$$A = [a_{i,j}]_{m\times n} \iff A^{\mathsf{H}} = A^{*} = [\bar{a}_{j,i}]_{n\times m}$$

Example:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \iff A^{\mathsf{H}} = \begin{bmatrix} \bar{a}_{11} & \bar{a}_{21} & \bar{a}_{31} \\ \bar{a}_{12} & \bar{a}_{22} & \bar{a}_{32} \end{bmatrix}$$

Basic Linear Algebra

- A is symmetric: if $A = A^{\mathsf{T}}$
  (usually it refers to "real" symmetric,
  it can also be "complex" symmetric)
- A is hermitian: if $A = A^{\mathsf{H}}$ (or $A = A^*$)
- Vector-wise notation:

$$a \in \mathbb{C}^m \iff a^{\mathsf{T}} \in \mathbb{C}^{1\times m}$$

$$A = \left[a_1, a_2, \cdots, a_n\right] \in \mathbb{C}^{m\times n} \iff A^{\mathsf{T}} = \begin{bmatrix} a_1^{\mathsf{T}} \\ a_2^{\mathsf{T}} \\ \vdots \\ a_n^{\mathsf{T}} \end{bmatrix} \in \mathbb{C}^{n\times m}$$

Basic Linear Algebra

Let $b = (b_i) \in \mathbb{R}^m, \quad A = (a_{i,j}) \in \mathbb{R}^{m \times n}, \quad x = (x_i) \in \mathbb{R}^n$

- Matrix-vector product $b = Ax$
  - Element-wise $b_i = \sum_{j=1}^{n} a_{i,j} x_j, \quad i = 1, 2, \ldots, m$
  - Vector-wise $b = \sum_{j=1}^{n} a_j x_j$
- Any $A \in \mathbb{C}^{m \times n}$ is a linear mapping from $\mathbb{C}^n$ to $\mathbb{C}^m$, meaning that

$$\begin{aligned} A(x+y) &= Ax + Ay, \qquad \forall\, x, y \in \mathbb{C}^n \\ A(\alpha x) &= \alpha A x, \qquad \forall\, \alpha \in \mathbb{C} \end{aligned}$$

Conversely, any linear mapping in finite dimensional space can be expressed as a matrix-vector product

Basic Linear Algebra

Let $b = (b_i) \in \mathbb{R}^m, \quad A = (a_j) \in \mathbb{R}^{m \times n}, \quad x = (x_i) \in \mathbb{R}^n$

- Matrix-vector product $\boxed{b = Ax}$
  - Vector-wise

$$\begin{aligned} b &= \sum_{j=1}^{n} a_j x_j \\ &= x_1[a_1] + x_2[a_2] + \cdots x_n[a_n] \end{aligned}$$

  - b is a linear combination of the columns of A
  - Any column of A can be picked out by choosing a specific x, e.g.

$$a_j = A(:, j) = Ae_j$$

  - Any row of of A can be picked out by matrix-vector product, e.g.

$$A(i, :) = e_i^{\mathsf{T}} A$$

Basic Linear Algebra

Let $A = (a_j) \in \mathbb{R}^{m\times n}, \quad B = (b_j) \in \mathbb{R}^{n\times k}, \quad C = (c_j) \in \mathbb{R}^{m\times k}$

- Matrix-matrix product $\boxed{C = AB}$
  - Vector-wise (compare columns in $C = AB$)

$$[c_1, c_2, \ldots, c_k] = A[b_1, b_2, \ldots, b_k]$$

$$\Longrightarrow \quad \boxed{c_j = Ab_j = \sum_{k=1}^{n} a_k b_{k,j}}$$

  - Each c_j is a linear combination of the columns of A

Basic Linear Algebra Subroutines (BLAS)

– Standardized interface for simple vector and matrix operations
– The building block of LAPACK (as the one used in Matlab)
– Optimized implementations for specific machines provided by manufacturers

- History:
  – BLAS1 (1970s) Vector operations: $\beta = x^{\mathsf{T}} y, y = \beta x + y$
  – BLAS2 (mid 1980s) Matrix-vector operations: $y = Ax + y$
  – BLAS3 (late 1980s) Matrix-matrix operations: $C = AB + C$
- Careful cache-aware implementations give close to peak performance for BLAS3 operations
- High level algorithms (Gaussian elimination, etc) use BLAS but no other machine dependent code
  – Performance and portability

Memory Hierarchy and (BLAS)

- Modern computers use a memory hierarchy:
  From fast/expensive to cheap/slow:
  Registers, L1 cache, L2 cache, (L3 cache ...)
  local memory, remote memory, secondary memory
- Fast algorithms perform many operations on each memory block to minimize memory access (cache reuse)
- Only BLAS3 has potential for very high performance

BLAS	Memory Refs	Flops	Flops/Memory Ref
Level 1 ($y = \beta x + y$)	$3n$	$2n$	$2/3$
Level 2 ($y = Ax + y$)	n^2	$2n^2$	2
Level 3 ($C = AB + C$)	$4n^2$	$2n^3$	$n/2$

Flop — floating points operations, here each $+$, $-$, $*$, $/$, $\sqrt{\ }$ counts as one flop, with no distinction between real and complex.

BLAS implementations

- Vendor provided:
  — Intel Math Kernel Library (MKL)
  — AMD Core Math Library (ACML)
  — Sun Performance Library
  — SGI Scientific Computing Software Library
- Automatically Tuned Linear Algebra Software (ATLAS)
  — Analyzes hardware to produce BLAS libraries for any platform
  — Used in MATLAB, precompiled libraries freely available
  — Sometimes outperforms vendor libraries
- GOTO BLAS (mainly for Intel processors)
  — Manually optimized assembly code,
  (fastest implementation for Intel processors)

Basic Linear Algebra

Examples of matrix-matrix product:

- Outer product: (rank-1)
  For $a = (a_i) \in \mathbb{C}^m,\ b = (b_i) \in \mathbb{C}^n, \quad (a_i, b_i \in \mathbb{C})$

$$ab^{\mathsf{H}} = [a\bar{b}_1, a\bar{b}_2, \cdots, a\bar{b}_n] = (a_i b_j) \in \mathbb{C}^{m\times n}$$

- Outer product: (rank $\leq k$)
  For $U = [u_j] \in \mathbb{C}^{m\times k},\ V = [v_j] \in \mathbb{C}^{n\times k}, \quad (u_j \in \mathbb{C}^m, v_j \in \mathbb{C}^n)$

$$UV^{\mathsf{H}} = [u_1, u_2, \cdots, u_k] \begin{bmatrix} v_1^{\mathsf{H}} \\ v_2^{\mathsf{H}} \\ \vdots \\ v_k^{\mathsf{H}} \end{bmatrix} = \sum_{j=1}^{k} u_j v_j^{\mathsf{H}} \in \mathbb{C}^{m\times n}$$

- Rank-k SVD is a representative rank-k outer product.

$$\boxed{A = U\Sigma V^{\mathsf{H}} = \textstyle\sum_{j=1}^{k} \sigma_j u_j v_j^{\mathsf{H}}}$$

Basic Linear Algebra

Examples of matrix-matrix product: $A \in \mathbb{C}^{m\times n}$

- Right multiply by an upper triangular matrix: $B = AR$
  Let $R = (r_{ij}) \in \mathbb{C}^{n\times n}$ be upper triangular,

$$B = AR = [a_1, a_2, \cdots, a_n] \begin{bmatrix} r_{11} & \cdots & r_{1n} \\ & \ddots & \vdots \\ & & r_{nn} \end{bmatrix} \implies \boxed{b_j = \sum_{\ell=1}^{j} a_\ell r_{\ell j}}$$

  (b_j is a linear combination of only the first j columns of A)
- Right multiply by a lower triangular matrix: $B = AL, L \in \mathbb{C}^{n\times n}$
  (b_j is a linear combination of only the last $n-j+1$ columns of A
- Left multiply by an upper triangular matrix: $B = RA, R \in \mathbb{C}^{m\times m}$
  (i-th row of B is a linear combination of last $m-i+1$ rows of A)
- Left multiply by a lower triangular matrix: $B = LA, L \in \mathbb{C}^{m\times m}$
  (i-th row of B is a linear combination of only the first i rows of A)

Basic Linear Algebra: Range, Nullspace

- The range or column space of $A = [a_1, a_2, \ldots, a_n] \in \mathbb{C}^{m\times n}$:

$$\begin{aligned} \mathrm{range}(A) &= \mathrm{span}\{a_1, a_2, \ldots, a_n\} \\ &= \text{All linear combinations of the columns of } A \\ &= \{Ax \mid \forall\, x \in \mathbb{C}^n\} \end{aligned}$$

- The nullspace of $A \in \mathbb{C}^{m\times n}$: (also written as kernel space $\ker(A)$)

$$\mathrm{null}(A) = \{x \;\mid\; Ax = 0\}$$

- Relation between $\mathrm{range}(A^{\mathsf{H}})$ and $\mathrm{null}(A)$

$$\mathrm{null}(A) = (\mathrm{range}(A^{\mathsf{H}}))^{\perp}$$

Equivalently,

Rank-nullity theorem: $\mathrm{rank}(A) + \dim(\mathrm{null}(A)) = n$

Basic Linear Algebra: Rank

- The column rank of $A = [a_1, a_2, \ldots, a_n] \in \mathbb{C}^{m\times n}$ is the dimension of $\mathrm{range}(A)$, it is the same as the number of linearly independent columns in $[a_1, a_2, \ldots, a_n]$.
- Similar definition for row rank
- For any $m \times n$ matrix A :

$$\mathrm{rank}(A) = \text{column rank of } A = \text{row rank of } A$$

Question: How to determine the rank of a given A?

Theorem

An $m \times n$ matrix A ($m \geq n$) is full rank iff $\mathrm{null}(A) = \{0\}$.

In other words, a full rank matrix never maps two different vectors to a same vector.

Basic Linear Algebra: Rank

Theorem: *Let $A \in \mathbb{C}^{m\times n}$, (assume operation compatibility)*

- $\mathrm{rank}(A) \le \min(m,n)$; $\quad \mathrm{rank}(A) = \dim(\mathrm{range}(A))$
- $\mathrm{rank}(AB) \le \min(\mathrm{rank}(A), \mathrm{rank}(B))$
- $\mathrm{rank}(AB) = \mathrm{rank}(A)$ if B has full row-rank
- $\mathrm{rank}(CA) = \mathrm{rank}(A)$ if C has full column-rank
- Subadditivity: $\mathrm{rank}(A+B) \le \mathrm{rank}(A) + \mathrm{rank}(B)$
  (Implication: A rank-k matrix can be the sum of k rank-1 matrices, but not fewer)
- $\mathrm{rank}(A^{\mathsf{H}}A) = \mathrm{rank}(AA^{\mathsf{H}}) = \mathrm{rank}(A) = \mathrm{rank}(A^{\mathsf{H}}) = \mathrm{rank}(A^{\mathsf{T}})$
- Rank-nullity theorem: $\boxed{\mathrm{rank}(A) + \dim(\mathrm{null}(A)) = n}$
- Frobenius' rank-inequality:
  $\boxed{\mathrm{rank}(AB) + \mathrm{rank}(BC) \le \mathrm{rank}(B) + \mathrm{rank}(ABC)}$
  Special case (Sylvester's rank-inequality):
  $\boxed{\mathrm{rank}(A) + \mathrm{rank}(B) \le n + \mathrm{rank}(AB)}$

Basic Linear Algebra: Inverse

- A square (size-n) matrix A is called nonsingular (or invertible or non-degenerate) if $\exists B$ s.t. $AB = BA = I_n$,
  in this case B is called the inverse of A: $A^{-1} = B$
- If A is nonsingular, then
  - $\left(A^{-1}\right)^{-1} = A$
  - $(A^{\mathsf{T}})^{-1} = (A^{-1})^{\mathsf{T}}, \qquad (A^{\mathsf{H}})^{-1} = (A^{-1})^{\mathsf{H}}$
  - $(AB)^{-1} = B^{-1}A^{-1}$
  - $\det(A^{-1}) = \det(A)^{-1}$
- Change of basis (view):

$$x = A^{-1}b \iff x \text{ is the solution to } Ax = b$$

  - x is the linear combination of the columns of A^{-1} with coefficients b
  - x is the vector of coefficients of the expansion of b in the basis of columns of A

Basic Linear Algebra: Inverse

Theorem: *For $A \in \mathbb{C}^{n\times n}$, the following statements are equivalent:*

- A is invertible
- $\mathrm{rank}\,(A) = n$
- $\ker\,(A) = \{0\}$ (or, $Ax = b$ has a unique solution)
- $\mathrm{range}\,(A) = \mathbb{C}^n$
- $\det\,(A) \neq 0$
- Eigenvalues of A are all non-zero
- Singular values of A are all non-zero
- The linear mapping $x \mapsto Ax$ is a bijection from $\mathbb{C}^n \to \mathbb{C}^n$
- A can be expressed as a product of a finite number of elementary matrices

Basic Linear Algebra: Elementary matrices

An elementary matrix is a matrix obtained by only one row operation (permutation, scaling, addition) of the identity matrix.
There are three types of elementary matrices.

(I) Permutation: $E(i,j)$ exchanges the i-th row with the j-th row of I_n,

$$E(i,j) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 0 & & 1 & & \\ & & & \ddots & & & \\ & & 1 & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix} = I_n - e_i e_i^{\mathsf{T}} - e_j e_j^{\mathsf{T}} + e_i e_j^{\mathsf{T}} + e_j e_i^{\mathsf{T}} \ .$$

Properties and applications:

- $[E(i,j)]^{-1} = E(i,j)$ (self-inverse)
- $E(i,j)A$ exchanges the i-th row with the j-th row of A
- $AE(i,j)$ exchanges the i-th column with the j-th column of A
- $\det(E(i,j)) \equiv -1, \quad \det(E(i,j)A) = \det(AE(i,j)) = -\det(A)$

Basic Linear Algebra: Elementary matrices

(II) Scaling: $E_s(i,c)$ scales the i-th row of I_n by c,

$$E_s(i,c) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & c & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix} = I_n + (c-1)e_i e_i^{\mathsf{T}}.$$

Properties and applications:

- If $c \neq 0$, then $E_s(i,c)^{-1} = E_s(i, \frac{1}{c})$
- $E_s(i,c)A$ scales only the i-th row of A by c
- $AE_s(i,c)$ scales only the i-th column of A by c
- $\det(E_s(i,c)) = c, \quad \det(E_s(i,c)A) = \det(AE_s(i,c)) = c\det(A)$

Basic Linear Algebra: Elementary matrices

(III) Addition: $E_a(i,j,c)$ scales the i-th row of I_n by c, and adds it to the j-th row of I_n,

$$E_a(i,j,c) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \ddots & & & \\ & & c & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix} = I_n + ce_j e_i^{\mathsf{T}}.$$

Properties and applications:

- $[E_a(i,j,c)]^{-1} = E_a(i,j,-c)$
- $E_a(i,j,c)A$ scales i-th row of A by c, and adds it to j-th row
- $AE_a(i,j,c)$ scales i-th column of A by c, and adds it to j-th column
- $\det(E_a(i,j,c)) \equiv 1$, $\det(E_a(i,j,c)A) = \det(AE_a(i,j,c)) = \det(A)$

Basic Linear Algebra: Elementary matrices

A general definition of size-n elementary matrices:

Size-n matrices of the form $I_n - uv^\top$, where $u, v \in \mathbb{R}^n$ with $v^\top u \neq 1$, are called elementary matrices.

It is easy to select u and v for the E, E_s, E_a just discussed. E.g.,

- For $E(i,j)$, $\quad u = v = e_i - e_j$
- For $E_s(i,c)$, $\quad u = (1-c)e_i$, $v = e_i$
- For $E_a(i,j,c)$, $\quad u = ce_j$, $v = -e_i$

Theorem:

An elementary matrix $I - uv^\top$ is always invertible, its inverse is

$$(I - uv^\top)^{-1} = I - \frac{uv^\top}{v^\top u - 1},$$

which is also an elementary matrix.

Inverse (block-wise elementary matrix operation)

Triangular nonsingular block matrix: $\begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix}$ and $\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}$

$$\begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix}^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{22}^{-1}A_{21}A_{11}^{-1} & A_{22}^{-1} \end{bmatrix}$$

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}^{-1} = \begin{bmatrix} A_{11}^{-1} & -A_{11}^{-1}A_{12}A_{22}^{-1} \\ 0 & A_{22}^{-1} \end{bmatrix}$$

In particular,

$$\begin{bmatrix} I & 0 \\ A_{21} & I \end{bmatrix}^{-1} = \begin{bmatrix} I & 0 \\ -A_{21} & I \end{bmatrix}, \qquad \begin{bmatrix} I & A_{12} \\ 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} I & -A_{12} \\ 0 & I \end{bmatrix}$$

Inverse (block-wise view)

General block matrix: $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$

- If A_{11} is invertible,

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}$$

$\boxed{S = A_{22} - A_{21}A_{11}^{-1}A_{12}}$ is called the Schur complement of A_{11} in A.

- If A_{22} is invertible,

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I & A_{12}A_{22}^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} \hat{S} & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} I & 0 \\ A_{22}^{-1}A_{21} & I \end{bmatrix}$$

$\boxed{\hat{S} = A_{11} - A_{12}A_{22}^{-1}A_{21}}$ is called the Schur complement of A_{22} in A.

- The above decompositions prove a Theorem:
  A is nonsingular iff its Schur complement is nonsingular.

Inverse (block-wise view)

If A is nonsingular, then

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}^{-1} = \begin{bmatrix} I & -A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix}$$

$$= \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}S^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}S^{-1} \\ -S^{-1}A_{21}A_{11}^{-1} & S^{-1} \end{bmatrix}$$

Similarly,

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}^{-1} = \begin{bmatrix} I & 0 \\ -A_{22}^{-1}A_{21} & I \end{bmatrix} \begin{bmatrix} \hat{S}^{-1} & 0 \\ 0 & A_{22}^{-1} \end{bmatrix} \begin{bmatrix} I & -A_{12}A_{22}^{-1} \\ 0 & I \end{bmatrix}$$

$$= \begin{bmatrix} \hat{S}^{-1} & -\hat{S}^{-1}A_{12}A_{22}^{-1} \\ -A_{22}^{-1}A_{21}\hat{S}^{-1} & A_{22}^{-1} + A_{22}^{-1}A_{21}\hat{S}^{-1}A_{12}A_{22}^{-1} \end{bmatrix}$$

Comparing the 1-1 block of $A^{-1} \Longrightarrow \hat{S}^{-1} = A_{11}^{-1} + A_{11}^{-1}A_{12}S^{-1}A_{21}A_{11}^{-1}$, the Binomial Inverse Theorem:

$$\left(A_{11} - A_{12}A_{22}^{-1}A_{21}\right)^{-1} = A_{11}^{-1} + A_{11}^{-1}A_{12}\left(A_{22} - A_{21}A_{11}^{-1}A_{12}\right)^{-1}A_{21}A_{11}^{-1}$$

Sherman-Morrison-Woodbury (SMW) formula

Binomial Inverse Theorem: (or SMW)

$$\left(A + UCV^{\mathsf{H}}\right)^{-1} = A^{-1} - A^{-1}U\left(C^{-1} + V^{\mathsf{H}}A^{-1}U\right)^{-1}V^{\mathsf{H}}A^{-1},$$

where A, U, C, V are matrices with compatible dimensions, A and $\left(C^{-1} + V^{\mathsf{H}}A^{-1}U\right)$ are nonsingular.

Special cases:

- (Sherman-Morrison) If A is nonsingular, $u, v \in \mathbb{C}^n$, and $1 + v^{\mathsf{H}}A^{-1}u \neq 0$, then

$$\left(A + uv^{\mathsf{H}}\right)^{-1} = A^{-1} - \frac{A^{-1}uv^{\mathsf{H}}A^{-1}}{1 + v^{\mathsf{H}}A^{-1}u}$$

- (Sherman-Morrison-Woodbury) If A is nonsingular, $U, V \in \mathbb{C}^{n\times k}$, and $I_k + V^{\mathsf{H}}A^{-1}U$ is invertible, then

$$\left(A + UV^{\mathsf{H}}\right)^{-1} = A^{-1} - A^{-1}U\left(I + V^{\mathsf{H}}A^{-1}U\right)^{-1}V^{\mathsf{H}}A^{-1}$$

Basic Linear Algebra: Vector Norms

Definition:

A vector norm $\|\cdot\|$ on a vector space $\mathbb{X}$ is a real-valued function on $\mathbb{X}$, which satisfies the following three conditions:

1. $\|x\| \geq 0, \quad \forall\, x \in \mathbb{X}, \quad$ and $\quad \|x\| = 0$ iff $x = 0$.
2. $\|\alpha x\| = |\alpha|\|x\|, \quad \forall\, x \in \mathbb{X}, \quad \forall \alpha \in \mathbb{C}$.
3. (Triangle inequality) $\|x + y\| \leq \|x\| + \|y\|, \quad \forall\, x, y \in \mathbb{X}$.

Common vector norms on $\mathbb{C}^n$

- $\|x\|_1 := |x_1| + |x_2| + \cdots + |x_n|$. (Manhattan norm or taxicab norm)
- $\|x\|_2 = \left(|x_1|^2 + |x_2|^2 + \cdots + |x_n|^2\right)^{1/2}$. (Euclidean norm)
- $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$. (Chebyshev norm or maximum norm)

All these three norms are special cases of the L_p norm

$$\boxed{\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}, \quad p \geq 1.}$$

(if $p < 1$, does $\|x\|_p$ define a norm?)

Verification of Norm Conditions

Example 1: Show that $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$ defines a norm.

Verification of Norm Conditions

Example 1: Show that $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$ defines a norm.

1. $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i| \geq 0$, and $\|x\|_\infty = 0$ iff $x = 0$
2. $\|\alpha x\|_\infty = \max_{i=1,\ldots,n} |\alpha x_i| = |\alpha| \max_{i=1,\ldots,n} |x_i| = |\alpha| \|x\|_\infty$
3. $\|x+y\|_\infty = \max_{i=1,\ldots,n} |x_i+y_i| \leq \max_{i=1,\ldots,n} |x_i| + \max_{i=1,\ldots,n} |y_i| = \|x\|_\infty + \|y\|_\infty$

Verification of Norm Conditions

Example 1: Show that $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$ defines a norm.

1. $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i| \geq 0$, and $\|x\|_\infty = 0$ iff $x = 0$
2. $\|\alpha x\|_\infty = \max_{i=1,\ldots,n} |\alpha x_i| = |\alpha| \max_{i=1,\ldots,n} |x_i| = |\alpha| \|x\|_\infty$
3. $\|x+y\|_\infty = \max_{i=1,\ldots,n} |x_i+y_i| \leq \max_{i=1,\ldots,n} |x_i| + \max_{i=1,\ldots,n} |y_i| = \|x\|_\infty + \|y\|_\infty$

Example 2: Show that $\|x\|_M = \sqrt{x^{\mathrm{H}} M x}$, where M is (hermitian) PD, defines a norm on $\mathbb{C}^n$. (This is called a weighted 2-norm.)

Verification of Norm Conditions

Example 1: Show that $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$ defines a norm.

1. $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i| \geq 0$, and $\|x\|_\infty = 0$ iff $x = 0$
2. $\|\alpha x\|_\infty = \max_{i=1,\ldots,n} |\alpha x_i| = |\alpha| \max_{i=1,\ldots,n} |x_i| = |\alpha| \|x\|_\infty$
3. $\|x+y\|_\infty = \max_{i=1,\ldots,n} |x_i+y_i| \leq \max_{i=1,\ldots,n} |x_i| + \max_{i=1,\ldots,n} |y_i| = \|x\|_\infty + \|y\|_\infty$

Example 2: Show that $\|x\|_M = \sqrt{x^{\mathsf{H}} M x}$, where M is (hermitian) PD, defines a norm on $\mathbb{C}^n$. (This is called a weighted 2-norm.)

1. Since M is PD, $\|x\|_M = \sqrt{x^{\mathsf{H}} M x} \geq 0$, and $\|x\|_M = 0$ iff $x = 0$
2. $\|\alpha x\|_M = \sqrt{\bar{\alpha} x^{\mathsf{H}} M \alpha x} = |\alpha| \|x\|_M$
3. $\|x+y\|_M^2 = (x+y)^{\mathsf{H}} M (x+y) = x^{\mathsf{H}} M x + x^{\mathsf{H}} M y + y^{\mathsf{H}} M x + y^{\mathsf{H}} M y$, Since M is PD, let $M = W^{\mathsf{H}} W$ for some W nonsingular, then $x^{\mathsf{H}} M y + y^{\mathsf{H}} M x = (Wx)^{\mathsf{H}}(Wy) + (Wy)^{\mathsf{H}}(Wx) \leq 2\|Wx\|_2 \|Wy\|_2 = 2\|x\|_M \|y\|_M$, therefore $\|x+y\|_M^2 \leq (\|x\|_M + \|y\|_M)^2$.

Basic Linear Algebra: Matrix norms

A matrix norm ($\|\cdot\|$) is a vector norm on $\mathbb{F}^{m\times n}$ (where $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$). That is,

- $\|A\| \geq 0$, and $\|A\| = 0$ iff $A = 0$
- $\|\alpha A\| = |\alpha|\|A\|, \quad \forall\, \alpha \in \mathbb{F}$ and $\forall A \in \mathbb{F}^{m\times n}$
- (Triangle inequality) $\|A + B\| \leq \|A\| + \|B\|, \quad \forall\, A, B \in \mathbb{F}^{m\times n}$
- In the case of square matrices, if $\|\cdot\|$ also satisfies $\|AB\| \leq \|A\|\|B\|, \quad \forall\, A, B \in \mathbb{F}^{n\times n}$, then $\|\cdot\|$ is called a sub-multiplicative norm (also called a *consistent* norm)

Example: Show that $\|A\| = \max\limits_{ij} |a_{ij}|$ defines a matrix norm. (This is called the max-norm.) Is it sub-multiplicative ?

Induced Matrix norms

Consider $A \in \mathbb{F}^{m\times n}$ as an operator from $\mathbb{F}^n \to \mathbb{F}^m$. Define the subordinate matrix norm on $\mathbb{F}^{m\times n}$ induced by $\|\cdot\|_\alpha$ on $\mathbb{F}^n$ and $\|\cdot\|_\beta$ on $\mathbb{F}^m$ as:

$$\|A\|_{\alpha,\beta} = \max_{x\neq 0} \frac{\|Ax\|_\beta}{\|x\|_\alpha} = \max_{\|x\|_\alpha=1} \|Ax\|_\beta \ .$$

When $\alpha = \beta$, it defines the induced matrix norm by vector norm $\|\cdot\|_\alpha$, this norm is also called the operator norm,

$$\|A\|_\alpha = \max_{x\neq 0} \frac{\|Ax\|_\alpha}{\|x\|_\alpha} = \max_{\|x\|_\alpha=1} \|Ax\|_\alpha \ .$$

Clearly,

$$\|Ax\|_\alpha \leq \|A\|_\alpha \|x\|_\alpha \, .$$

Property: Every induced matrix norm is sub-multiplicative.

Proof: For any compatible A, B and an induced matrix norm $\|\cdot\|_\alpha$,

$$\|AB\|_\alpha = \max_{\|x\|_\alpha=1} \|ABx\|_\alpha \leq \max_{\|x\|_\alpha=1} \|A\|_\alpha \|Bx\|_\alpha \leq \|A\|_\alpha \|B\|_\alpha \ .$$

Examples of induced matrix norms

- 1-norm:

$$\|A\|_1 = \max_{x \neq 0} \frac{\|Ax\|_1}{\|x\|_1} = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$$

- 2-norm:

$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sqrt{\lambda_{\max}(A^{\mathsf{H}}A)} = \sigma_{\max}(A)$$

(2-norm is also called the spectral norm)

- ∞-norm:

$$\|A\|_\infty = \max_{x \neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|.$$

Frobenius norm

For $A \in \mathbb{F}^{m\times n}$, treat A as a length mn vector and define the L_p vector norm: $(p \geq 1)$

$$\|A\|_{L_p} = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^p\right)^{1/p}.$$

- $p = 2$ gives the Frobenius norm
  $\|A\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2\right)^{1/2} = \sqrt{\mathrm{trace}\,(A^*A)} = \sqrt{\mathrm{trace}\,(AA^*)}$
- Frobenius norm is sub-multiplicative, but it is not an induced norm (why?)

Frobenius norm

For $A \in \mathbb{F}^{m\times n}$, treat A as a length mn vector and define the L_p vector norm: $(p \geq 1)$

$$\|A\|_{L_p} = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^p\right)^{1/p}.$$

- $p = 2$ gives the Frobenius norm
  $\|A\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2\right)^{1/2} = \sqrt{\mathrm{trace}\,(A^*A)} = \sqrt{\mathrm{trace}\,(AA^*)}$
- Frobenius norm is sub-multiplicative, but it is not an induced norm (why?) (if it is induced, one would have $\|I\|_F = 1$)

Frobenius norm

For $A \in \mathbb{F}^{m\times n}$, treat A as a length mn vector and define the L_p vector norm: $(p \geq 1)$

$$\|A\|_{L_p} = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^p\right)^{1/p}.$$

- $p = 2$ gives the Frobenius norm
  $\|A\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2\right)^{1/2} = \sqrt{\mathrm{trace}\,(A^*A)} = \sqrt{\mathrm{trace}\,(AA^*)}$
- Frobenius norm is sub-multiplicative, but it is not an induced norm (why?) (if it is induced, one would have $\|I\|_F = 1$)
- Both 2-norm and Frobenius norm are unitarily invariant: Given $A \in \mathbb{C}^{m\times n}$, then for any unitary $Q_m \in \mathbb{C}^{m\times m}, Q_n \in \mathbb{C}^{n\times n}$, (later)

$$\|A\|_\gamma = \|Q_m A\|_\gamma = \|AQ_n\|_\gamma = \|Q_m A Q_n\|_\gamma \quad \text{where } \gamma = 2, F.$$

- $p = \infty$ yields the max-norm $\left(\max_{ij} |a_{ij}|\right)$, also called the uniform norm.

Matrix norms defined by singular values

Let $X \in \mathbb{C}^{m\times n}$ with $m \geq n$, denote the singular values of X as $\{\sigma_i(X)\}, i = 1, \ldots, n$. The Schatten p-norm ($p \geq 1$) of X is defined as

$$\|X\|_{S_p} := \left(\sum_{i=1}^{n} \sigma_i(X)^p\right)^{1/p}.$$

Special cases:

- Nuclear norm ($p = 1$), also called the *trace norm* or *Ky-Fan norm*:

$$\|X\|_* = \|X\|_{tr} := \sum_{i=1}^{n} \sigma_i(X)$$

- Frobenius norm ($p = 2$): $\|X\|_F = \|X\|_{S_2}$.
- Spectral norm ($p = \infty$): $\|X\|_2 = \|X\|_{S_\infty}$.

Basic Linear Algebra: Inner Products

- For $x, y \in \mathbb{R}^n$,

$$\langle x \ , \ y \rangle := y^{\mathsf{T}} x = x^{\mathsf{T}} y = \sum_{i=1}^{n} x_i y_i$$

- For $x, y \in \mathbb{C}^n$,

$$\langle x \ , \ y \rangle := y^{\mathsf{H}} x = \overline{x^{\mathsf{H}} y} = \sum_{i=1}^{n} x_i \bar{y}_i$$

- (An important property) Given $A \in \mathbb{C}^{m \times n}$,

$$\boxed{\langle Ax \ , \ y \rangle = \langle x \ , \ A^{\mathsf{H}} y \rangle, \quad \forall\, x \in \mathbb{C}^n, \ \ y \in \mathbb{C}^m}$$

- $\boxed{\langle x \ , \ x \rangle \geq 0 \quad \forall\, x}$
- Cauchy inequality: (Cauchy-Bunyakowski-Schwarz)

$$\boxed{\langle x \ , \ y \rangle \leq \|x\|_2 \, \|y\|_2}$$

- Let α be the angle between two vectors $x, y \in \mathbb{C}^n$, then

$$\cos(\alpha) = \frac{\langle x \ , \ y \rangle}{\|x\|_2 \, \|y\|_2}$$

Inner Product (general definition)

Definition: (inner product on a linear space V)

A mapping $\langle \cdot\ ,\ \cdot \rangle : V \times V \to \mathbb{F}$ ($\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$) is called an inner product if it satisfies

1. Positive-definiteness:
   $\langle u\ ,\ u \rangle \geq 0,\ \forall\, u \in V;\qquad \langle u\ ,\ u \rangle = 0$ if and only if $u = 0$,
2. Conjugate symmetry:
   $$\langle u\ ,\ v \rangle = \overline{\langle v\ ,\ u \rangle}, \quad \forall\, u, v \in V$$
3. Linearity in the first argument:
   I.e., the mapping $u \to \langle u\ ,\ v \rangle$ is linear for each $v \in V$:
   $$\langle \alpha u\ ,\ v \rangle = \alpha \langle u\ ,\ v \rangle, \qquad \forall \alpha \in \mathbb{F}$$
   $$\langle u_1 + u_2\ ,\ v \rangle = \langle u_1\ ,\ v \rangle + \langle u_2\ ,\ v \rangle, \qquad \forall u_1, u_2, v \in V$$

If $\mathbb{F} = \mathbb{R}$, then the conjugate symmetry reduces to symmetry.

Examples of matrix inner products

A very common inner product on the vector space $\mathbb{R}^{n\times n}$ is defined as

$$\langle X,\ Y\rangle = \mathrm{trace}\left(X^{\mathsf{T}}Y\right) = \mathrm{trace}\left(Y^{\mathsf{T}}X\right), \qquad \forall\, X, Y \in \mathbb{R}^{n\times n}.$$

The corresponding inner product on $\mathbb{C}^{n\times n}$ is defined as

$$\langle X,\ Y\rangle = \mathrm{trace}\left(Y^{\mathsf{H}}X\right), \qquad \forall\, X, Y \in \mathbb{C}^{n\times n}.$$

- The above defined $\langle\cdot\ ,\ \cdot\rangle$ is known as the Hilbert-Schmidt inner product.
- Frobenius norm is the same as the Hilbert-Schmidt norm: $\|A\|_F = \sqrt{\langle X,\ X\rangle} = \sqrt{\mathrm{trace}\left(X^{\mathsf{H}}X\right)}$.

Orthogonality; Orthonormality

- Two vectors x, y in an inner product space (say $\mathbb{R}^n$ or $\mathbb{C}^n$) are orthogonal if
$$\langle x \ , \ y \rangle = 0$$
- Two sets of vectors X, Y are orthogonal if
$$\langle x \ , \ y \rangle = 0, \qquad \forall x \in X, \quad \forall y \in Y$$
- Pairwise orthogonal set of vectors S is defined as a set of nonzero vectors orthogonal to each other. I.e.,
$$\langle x \ , \ y \rangle = 0, \qquad \forall x, y \in S, \quad x \neq y$$
- Pairwise orthonormal set of vectors S is defined as a set of unit length (in 2-norm) vectors orthogonal to each other.

Orthogonal matrices; Unitary matrices

- A matrix $Q \in \mathbb{R}^{n\times n}$ is orthogonal if

$$Q^{-1} = Q^{\mathsf{T}}$$

- A matrix $Q \in \mathbb{C}^{n\times n}$ is unitary if

$$Q^{-1} = Q^{\mathsf{H}}$$

- A set of column vectors of a unitary (or orthogonal) matrix is pairwise orthonormal
- A set of row vectors of a unitary (or orthogonal) matrix is pairwise orthonormal
- Inverse reduced to (conjugate) transpose !

$$Qx = b \quad \Longleftrightarrow \quad x = Q^{\mathsf{H}} b$$

- Important class of normal matrices (defined as $A^*A = AA^*$)

Preservation of length and angle

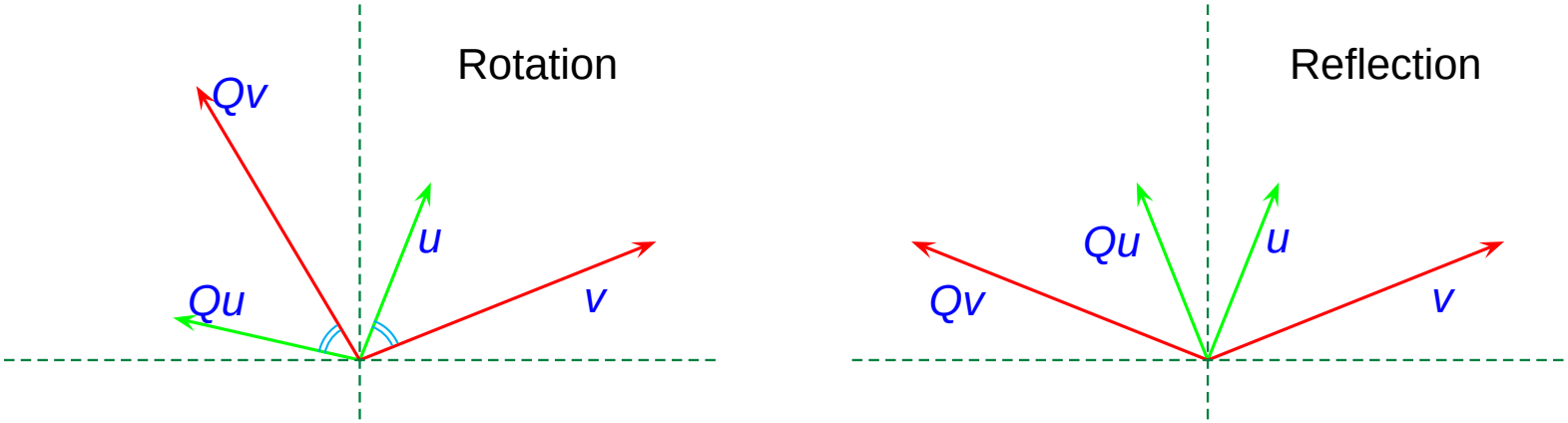
- $Q^{\mathsf{H}}Q = QQ^{\mathsf{H}} = I \implies \boxed{|\det(Q)| = 1}$, $\det(Q) = \pm 1$ when Q is real
- Preserves inner product

$$\langle Qx\ ,\ Qy\rangle = \langle x\ ,\ Q^{\mathsf{H}}Qy\rangle = \langle x\ ,\ y\rangle$$

  Therefore, unitary matrix multiplication preserves length of vector ($\|Qx\|_2 = \|x\|_2$) and angle between vectors

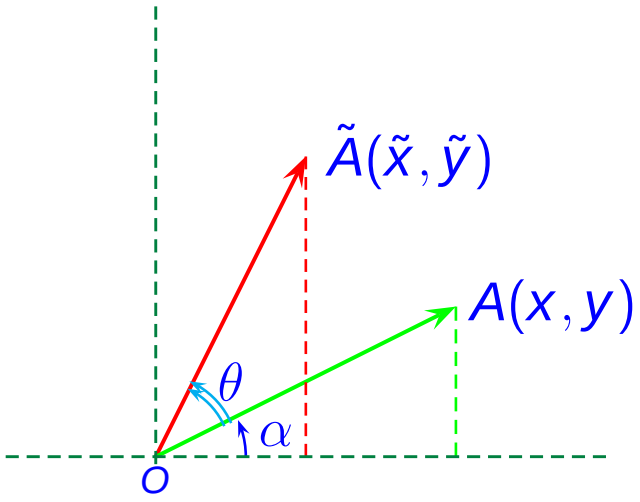
$$\cos\angle(Qx, Qy) = \cos\angle(x, y)$$

- A (real) orthogonal Q can only be a rigid rotation ($\det(Q) = 1$) or reflection ($\det(Q) = -1$)



Givens rotation in 2-D

Rotating $\overrightarrow{OA}$ anti-clockwise by θ to $\overrightarrow{O\tilde{A}}$. Denote $L = \|\overrightarrow{OA}\| = \|\overrightarrow{O\tilde{A}}\|$.



$$
\begin{aligned}
x &= L\cos(\alpha), \quad y = L\sin(\alpha); \\
\tilde{x} &= L\cos(\alpha+\theta) \\
&= x\cos(\theta) - y\sin(\theta), \\
\tilde{y} &= L\sin(\alpha+\theta) \\
&= y\cos(\theta) + x\sin(\theta).
\end{aligned}
$$

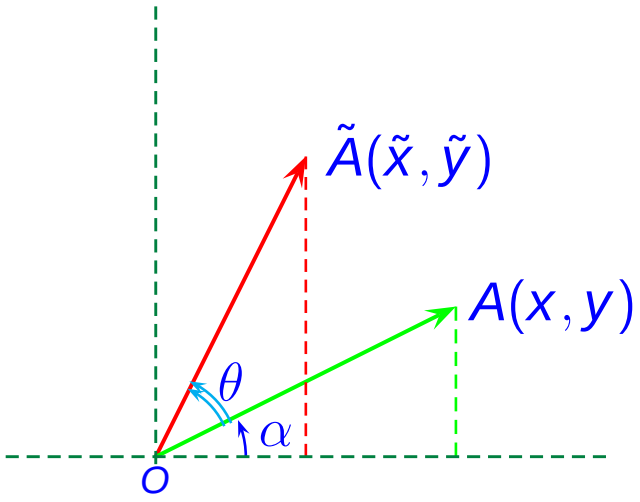
$$
\Longrightarrow \quad \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = G(\theta) \begin{bmatrix} x \\ y \end{bmatrix} := \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
$$

If rotate clockwise by θ, then the Givens rotation matrix is

$$
G(-\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} .
$$

Givens rotation in 2-D

Rotating $\overrightarrow{OA}$ anti-clockwise by θ to $\overrightarrow{O\tilde{A}}$. Denote $L = \|\overrightarrow{OA}\| = \|\overrightarrow{O\tilde{A}}\|$.



$$
\begin{aligned}
x &= L\cos(\alpha), \quad y = L\sin(\alpha); \\
\tilde{x} &= L\cos(\alpha+\theta) \\
&= x\cos(\theta) - y\sin(\theta), \\
\tilde{y} &= L\sin(\alpha+\theta) \\
&= y\cos(\theta) + x\sin(\theta).
\end{aligned}
$$

$$
\Longrightarrow \quad \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = G(\theta) \begin{bmatrix} x \\ y \end{bmatrix} := \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
$$

If rotate clockwise by θ, then the Givens rotation matrix is

$$
G(-\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}. \qquad \boxed{G^{-1}(\theta) = G(-\theta)}
$$

Givens rotation to zero out an element

$$\begin{bmatrix}\tilde{x}\\ \tilde{y}\end{bmatrix} = G(\theta)\begin{bmatrix}x\\ y\end{bmatrix} := \begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix} = \begin{bmatrix}x\cos(\theta) - y\sin(\theta)\\ y\cos(\theta) + x\sin(\theta)\end{bmatrix}$$

- To zero out the 2nd element in $\begin{bmatrix}x\\ y\end{bmatrix}$, simply choose a θ s.t. $\tilde{y} = 0$, i.e., $\cot(\theta) = \frac{-x}{y}$
- There are more numerically stable ways to compute the $\sin(\theta), \cos(\theta)$ from x, y
- To selectively zero out k elements in a length-n vector, apply corresponding Givens rotation k times sequentially

Givens rotation in n-D

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos(\theta) & \cdots & -\sin(\theta) & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \sin(\theta) & \cdots & \cos(\theta) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

That is, $G(i,j,\theta) = I_n$ except at the ii, jj, ij, ji positions.

- Effect: $G(i,j,\theta)x$ rotates x counterclockwise in (i,j) plane by θ
- Main use: To introduce zeros in vectors or matrices. E.g., for computing QR decompositions
- Advantage: Stable (it is unitary!)
  Lower operation count for very sparse matrices (only need to zero out a few nonzero elements)

Householder reflection

Givens rotation targets to introduce one zero per rotation. Householder reflection introduces $n-1$ zeros to a length-n vector per reflection: by requiring at that the reflected vector has only one nonzero (i.e., parallel only to some e_i).

Let $x \in \mathbb{C}^{n\times n}$, denote the Householder reflector as H, want Hx to be parallel to some e_i, say e_1:

$$Hx = \alpha e_1$$

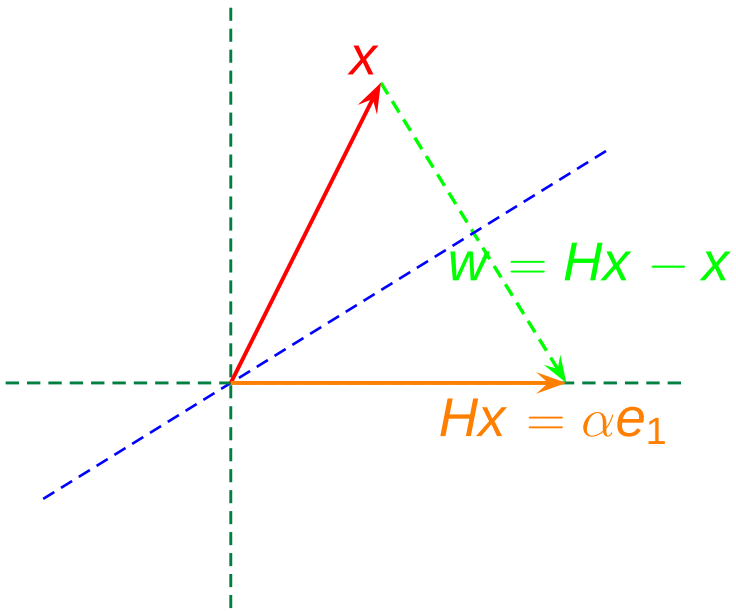
$$H \text{ is unitary} \Longrightarrow \quad \|Hx\|_2 = \|\alpha e_1\|_2 = |\alpha| = \|x\|_2$$

Question: How to construct H, which clearly only depends on x, s.t. the above two requirements are met ?

Essentially there is only one requirement: Construct H to be unitary such that $Hx = \alpha e_1$. (The $|\alpha| = \|x\|_2$ will hold automatically.)

Constructing a Householder reflector

The hyper-plane to reflect on should be orthogonal to $w = Hx - x$.



Orthogonal projection of x on w is

$$\frac{w(w^{\mathsf{H}}x)}{w^{\mathsf{H}}w}.$$

From x, need to go twice the length of this projection to reach Hx:

$$Hx = x - 2\frac{ww^{\mathsf{H}}x}{w^{\mathsf{H}}w}$$

The desired Householder reflector is

$$\boxed{H = I - 2\frac{ww^{\mathsf{H}}}{w^{\mathsf{H}}w}}$$

where $w = \alpha e_1 - x$, $|\alpha| = \|x\|_2$. Choose the sign of α s.t. least cancellation of $\alpha e_1 - x$ is involved (i.e., $\alpha = -\mathrm{sign}(x_1)\,\|x\|_2$)

More on Householder reflector

H can be compactly written as

$$H = I - 2vv^{\mathsf{H}}, \qquad \text{where } \|v\|_2 = 1 \,.$$

Question: What is $\det(H)$? What are the eigenvalues of H?

Exercise: For a given nonzero $v \in \mathbb{C}^n$, construct an H such that $Hv = \|v\|_2 \, e_n$. Use the constructed H to directly calculate Hv and verify that it indeed equals to $\|v\|_2 \, e_n$.

Summary of six major matrix decompositions:

- LU decomposition

$$A = LU$$

where L is unit lower triangular, U is upper triangular

- Cholesky decomposition (for hermitian PSD matrices) :

$$A = R^{\mathsf{H}} R = L D L^{\mathsf{H}}$$

where R is upper triangular, and L is unit upper triangular

- QR decomposition (for $A \in \mathbb{C}^{m \times n}, m \geq n$)

$$A = \tilde{Q} \begin{bmatrix} R \\ 0 \end{bmatrix} := [Q, Q_{\perp}] \begin{bmatrix} R \\ 0 \end{bmatrix} = QR,$$

where $R \in \mathbb{C}^{n \times n}$ is upper triangular, $Q \in \mathbb{C}^{m \times n}$, and $\tilde{Q} = [Q, Q_{\perp}] \in \mathbb{C}^{m \times m}$ is unitary

Summary of six major matrix decompositions:

- Spectral decomposition (for diagonalizable $A \in \mathbb{C}^{n\times n}$)

$$A = X\Lambda X^{-1}, \qquad \Lambda = \mathrm{diag}(\lambda_1, \cdots, \lambda_n),$$

where X contains the eigenvectors.
If A is symmetric/hermitian, then

$$\boxed{A = Q\Lambda Q^{\mathsf{H}}\ ,}$$

where Q is unitary and contains the eigenvectors.

- Schur decomposition (for $A \in \mathbb{C}^{n\times n}$)

$$A = USU^{\mathsf{H}},$$

where U is unitary, and S is upper triangular. (Questions: What are on the $\mathrm{diag}(S)$? Can one choose the order of the diagonal elements?)

- Singular value decomposition (SVD) — next few slides

Some history of SVD

- Originally developed independently by differential geometers: Eugenio Beltrami (1873), Camille Jordan (1874)
- Rediscovered independently: James J. Sylvester (1889)
- Analog of singular values for compact integral operators: Erhard Schmidt (1907), Hermann Weyl (1912)
  Émile Picard in 1910 seems the first to use the term *singular values*
- SVD of complex matrices: Léon Autonne (1913)
  SVD of rectangular matrices: Carl Eckart and Gale Young (1936), L. Mirsky (1960)
- Computation: Gene Golub and William Kahan (1965), Gene Golub and Christian Reinsch (1970)

SVD is also known as principal component analysis (PCA), proper orthogonal decomposition (POD), Hotelling transform, or (discrete) Karhunen-Loève (KL) transformation.

Some applications of SVD

- Information retrieval and data mining
- Data compression; Noise filtering
  (Noises tend to correspond to small singular values)
- Solving least squares;
  Regularization of ill-conditioned (inverse) problems
- Image and signal processing: e.g., Image deblurring;
  Seismology; Tomography
- Graph partition; graph coloring
- Bioinformatics and computational biology: Immunology;
  Molecular dynamics; Microarray data analysis
- Weather prediction
- Quantum information, in which SVD is known as the Schmidt decomposition

Geometrical motivation of SVD

Fact: Image of a unit sphere S^n in $\mathbb{R}^n$ under any real $m \times n$ matrix is a hyperellipse AS^n in $\mathbb{R}^m$.

For example:

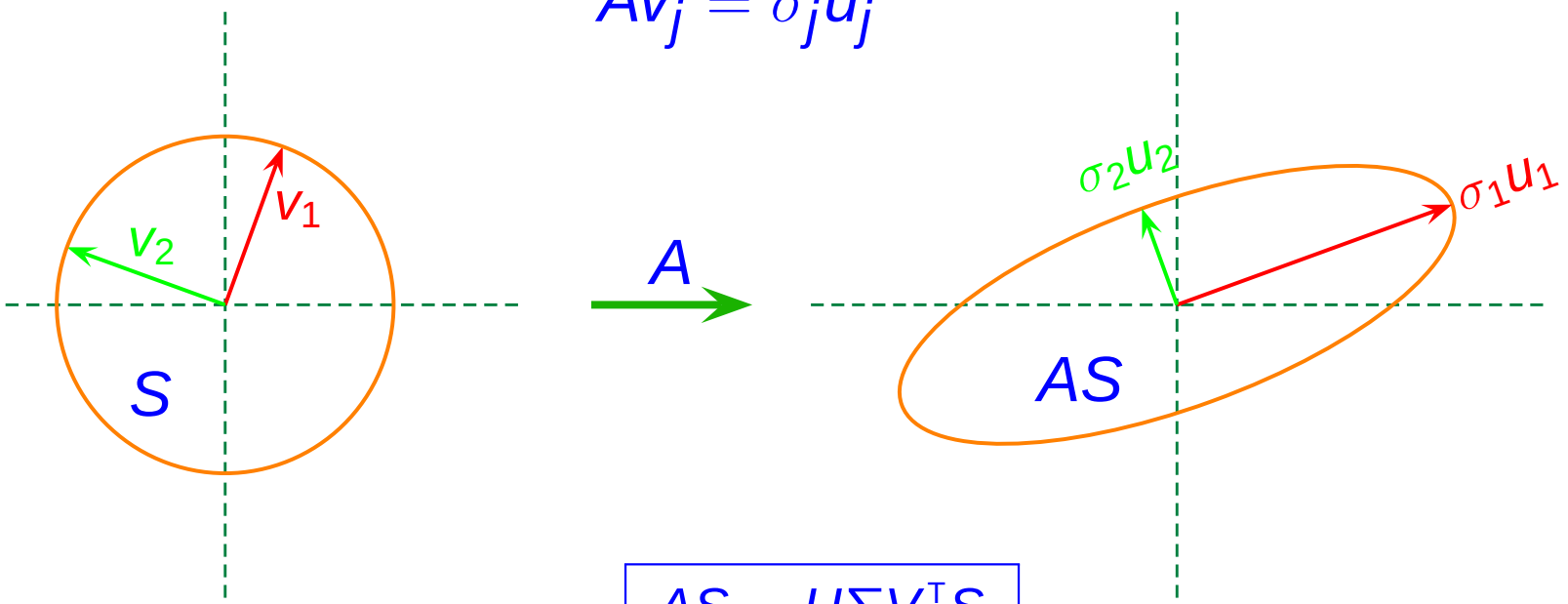
$$S^2 = \left\{ (x_1, x_2) \,\middle|\, x_1^2 + x_2^2 = 1 \right\}$$

- If $A = \begin{bmatrix} \sigma_1 & \\ & \sigma_2 \end{bmatrix}$, then $AS^2 = \left\{ (y_1, y_2) \,\middle|\, \frac{y_1^2}{\sigma_1^2} + \frac{y_2^2}{\sigma_2^2} = 1 \right\}$ is an ellipse in $\mathbb{R}^2$
- If $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, then $AS^2 = \left\{ (y_1, y_2) \,\middle|\, y_i = \sum_j a_{ij} x_j,\ x_1^2 + x_2^2 = 1 \right\}$ is an ellipse in $\mathbb{R}^2$
- If $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$, then AS^2 is a (reduced) ellipsoid in $\mathbb{R}^3$ (essentially it is still a 2-d ellipse)

Geometrical interpretation of SVD

Fact: Image of a unit sphere S in $\mathbb{R}^n$ under any $A \in \mathbb{R}^{m\times n}$ is a hyperellipse AS in $\mathbb{R}^m$.

$$Av_j = \sigma_j u_j$$



$$\boxed{AS = U\Sigma V^{\mathsf{T}} S}$$

$V^{\mathsf{T}}S$ contains rotations/reflections of S, it is still a unit sphere; $\Sigma(V^{\mathsf{T}}S)$ contains scaling of the new unit sphere, resulting in a hyperellipse; and $U(\Sigma V^{\mathsf{T}}S)$ contains rotations/reflections of the hyperellipse, without changing its shape.

Geometrical interpretation of SVD

Fact: Image of $S = \left\{ x \;\middle|\; \|x\|_2 = 1,\ x \in \mathbb{R}^n \right\}$ under any $A = U\Sigma V^\top \in \mathbb{R}^{m\times n}$ is a hyperellipse AS in $\mathbb{R}^m$.

The $\sigma_i(A)$'s measure how much distortion A applies to S:
$U^\top AS$ is a hyperellipse in standard position, with k-th semiaxis equal to $\sigma_k(A)$.

Note $U^\top AS = \left\{ y \;\middle|\; y = U^\top Ax,\ x \in S \right\}$, (assume $\sigma_i > 0, i = 1, \ldots, n$)

$$y := U^\top Ax = U^\top U\Sigma V^\top x = \Sigma V^\top x, \quad \forall x \in S$$

$$\|x\|_2 = \left\|V^\top x\right\|_2 = \left\|\Sigma^{-1} y\right\|_2 = 1, \quad \Longrightarrow \quad \boxed{\frac{y_1^2}{\sigma_1^2} + \frac{y_2^2}{\sigma_2^2} + \cdots + \frac{y_n^2}{\sigma_n^2} = 1}$$

Since U is unitary, $U^\top AS$ only applies rotation/reflection to AS without changing its shape. $\Longrightarrow$ AS is a (reduced) hyperellipse in $\mathbb{R}^m$, with its k-th semiaxis equal to $\sigma_k(A)$.

Singular value decomposition (main idea)

Let $A \in \mathbb{C}^{m\times n}$, assume that $m \geq n$.

The idea of SVD may be summarized as to find two sets of orthonormal bases of A s.t. A appears to be a simple diagonal matrix:

- $U = [u_1, \ldots, u_n]$ for the column space, i.e., $\mathrm{range}\,(A) \subseteq \mathrm{span}(U)$
- $V = [v_1, \ldots, v_n]$ for the row space, i.e., $\mathrm{range}\,(A^{\mathsf{H}}) \subseteq \mathrm{span}(V)$
- such that Av_i is in the direction of u_i: $\boxed{Av_i = \sigma_i u_i}$ $(\sigma_i \geq 0)$

In matrix notation,

$$A \left[v_1 \middle| v_2 \middle| \cdots \middle| v_n \right] = \left[u_1 \middle| u_2 \middle| \cdots \middle| u_n \right] \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix} \implies \boxed{AV = U\Sigma}$$

The σ_i's are called *singular values* of A and usually ordered non-increasingly: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$.

Singular value decomposition (main structure)

Singular value decomposition (for $A \in \mathbb{C}^{m\times n}$, $m \geq n$):

$$A = \tilde{U} \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^{\mathsf{H}}$$

where $\tilde{U} \in \mathbb{C}^{m\times m}, V \in \mathbb{C}^{n\times n}$ are unitary, Σ is diagonal.
Let $\tilde{U} := [U, U_\perp], U \in \mathbb{C}^{m\times n}$, then

$$A = [U, U_\perp] \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^{\mathsf{H}} = U\Sigma V^{\mathsf{H}}$$

Furthermore, if $\Sigma = \begin{bmatrix} \Sigma_k & \\ & 0 \end{bmatrix}$ with $k < n$, then

$$A = U\Sigma V^{\mathsf{H}} = [U_k, U_{k\perp}] \begin{bmatrix} \Sigma_k & \\ & 0 \end{bmatrix} \begin{bmatrix} V_k^{\mathsf{H}} \\ (V_{k\perp})^{\mathsf{H}} \end{bmatrix} = U_k \Sigma_k V_k^{\mathsf{H}}$$

Another proof of the rank-nullity theorem

Rank-nullity theorem: $\forall\, A \in \mathbb{C}^{m\times n}$, $\operatorname{rank}(A) + \dim(\ker(A)) = n$.

This result is a corollary of a stronger result:

$$\operatorname{range}(A^*) = \ker(A)^{\perp}.$$

This result is straightforward from SVD: Let $A = U\Sigma V^{\mathsf{H}}$, where $\Sigma = \begin{bmatrix} \Sigma_k & \\ & 0 \end{bmatrix}$ with $\sigma_k > 0, k \le n$, $U = [U_k, U_{k\perp}], V = [V_k, V_{k\perp}]$. Then

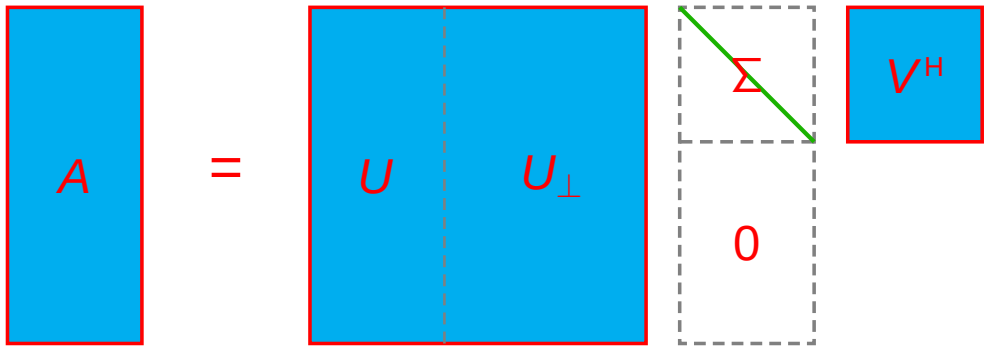
$$A[V_k, V_{k\perp}] = [U_k, U_{k\perp}] \begin{bmatrix} \Sigma_k & \\ & 0 \end{bmatrix}, \qquad A^* = V_k \Sigma_k U_k^*.$$

Therefore, $\ker(A) = \operatorname{span}(V_{k\perp})$, $\operatorname{range}(A^*) = \operatorname{span}(V_k)$, from which it follows

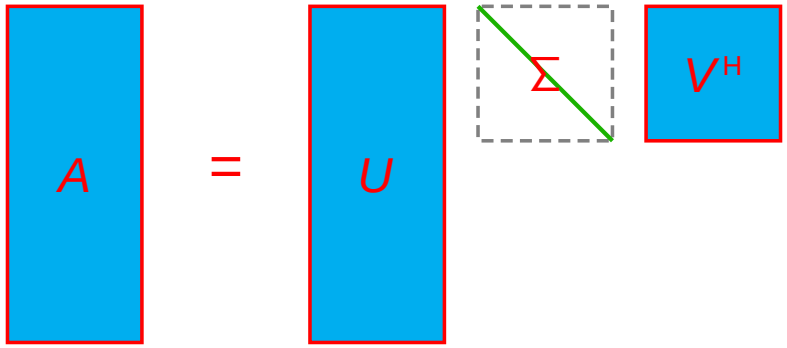
$$\operatorname{range}(A^*) = \ker(A)^{\perp}.$$

SVD (main structure)

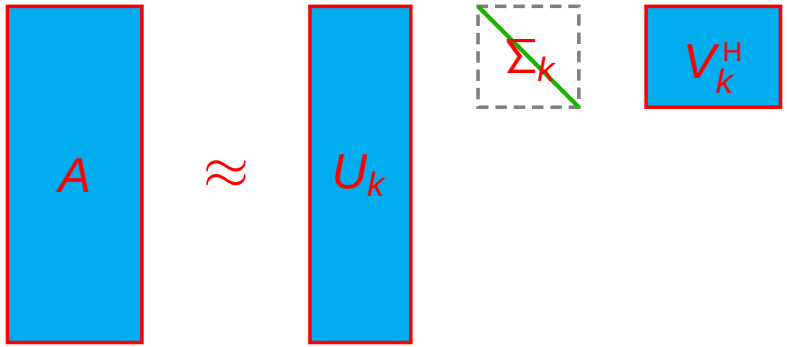
Full SVD:
$A = \tilde{U}\tilde{\Sigma}V^{\mathsf{H}}$



(Thin) SVD:
$A = U\Sigma V^{\mathsf{H}}$



Truncated SVD:
$A \approx U_k \Sigma_k V_k^{\mathsf{H}}$



SVD and the Eigenvalue Decomposition (EVD)

Assume that A is square and diagonalizable, the *eigenvalue decomposition* is

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

- EVD uses the same basis X for row and column space;
  SVD uses two different bases V, U
- EVD generally does not maintain an orthonormal basis in X, unless A is normal;
  SVD always has two orthonormal bases
- EVD is defined only for square matrices;
  SVD exists for all matrices
- For *hermitian/symmetric positive definite* matrices A, EVD and SVD are the same (assuming same order in Λ and Σ)
- For *hermitian/symmetric* matrices A, EVD and SVD are the same except that $\sigma_i = |\lambda_i|$ (assuming same order in Λ and Σ)

Matrix properties revealed by SVD

For general matrix $A \in \mathbb{C}^{m\times n}$,

$$A = U\Sigma V^{\mathsf{H}} \implies \begin{cases} AA^{\mathsf{H}}U &= U\Sigma^2 \\ A^{\mathsf{H}}AV &= V\Sigma^2 \end{cases}$$

- Nonzero eigenvalues of $A^{\mathsf{H}}A$ are nonzero σ_i^2, eigenvectors are v_i
- Nonzero eigenvalues of AA^{H} are nonzero σ_i^2, eigenvectors are u_i
- The rank of $A =$ the number of nonzero singular values
- $\mathrm{range(A)} = \langle u_1, \ldots, u_r \rangle$ and $\mathrm{null(A)} = \langle v_{r+1}, \ldots, v_n \rangle$, ($r = \mathrm{rank}\,(A)$)
- $\|A\|_2 = \sigma_1$ and $\|A\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \ldots + \sigma_r^2}$
- If $A = A^{\mathsf{H}}$, then $\sigma_i = |\lambda_i|$ where λ_i are eigenvalues of A
- For square A, $|\det(A)| = \prod_{i=1}^m \sigma_i$, (compare $\det(A) = \prod_{i=1}^m \lambda_i$)
- Condition number of A: $\mathrm{cond}\,(A) = \frac{\sigma_{\max}}{\sigma_{\min}}$

Low-Rank Approximations

The SVD of a rank r matrix $A \in \mathbb{C}^{m\times n}$ ($r \le \min(m,n)$) can be written as a sum of r rank-one matrices

$$A = U\Sigma V^* = \sum_{j=1}^{r} \sigma_j u_j v_j^* \ .$$

Theorem: (Schmidt-Weyl / Eckart-Young-Mirsky)

The best rank k approximation of a rank r A in the 2- and F-norm is

$$A_k = \textstyle\sum_{j=1}^{k} \sigma_j u_j v_j^* \ .$$

The errors are $\|A - A_k\|_2 = \sigma_{k+1}$ and $\|A - A_k\|_F = \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_r^2}$.

In other words,

$$\sigma_{k+1} = \min_{\mathrm{rank}(B)=k} \|A - B\|_2 \ , \qquad \sum_{i=k+1}^{r} \sigma_i^2 = \min_{\mathrm{rank}(B)=k} \|A - B\|_F^2 \ .$$

Proof of the Schmidt-Weyl Theorem

We prove the general result: For any $A \in \mathbb{C}^{m\times n}$,

$$\min_{\substack{B\in\mathbb{C}^{m\times n}\\ \mathrm{rank}(B)\le k}} \|A-B\|_2 = \sigma_{k+1}(A)\ .$$

The proof uses a standard technique in linear algebra which may be called *dimensionality argument*.

Proof. By contradiction, if $\exists B \in \mathbb{C}^{m\times n}, \mathrm{rank}\,(B) \le k$ s.t. $\|A-B\|_2 < \sigma_{k+1}(A)$. Then $\forall w \in \ker(B), w \neq 0$, $\|Aw\|_2 = \|(A-B)w\|_2 \le \|A-B\|_2 \|w\|_2 < \sigma_{k+1}(A)\|w\|_2$.
Note that $\dim(\ker(B)) \ge n-k$, and $\dim(\mathrm{span}\{v_1, v_2, \ldots, v_{k+1}\}) = k+1$, therefore $\exists w_0 \in \ker(B) \cap \mathrm{span}\{v_1, v_2, \ldots, v_{k+1}\}$, where $w_0 = \sum_{i=1}^{k+1} c_i v_i \neq 0$, for which it must be true that $\|Aw_0\|_2 = \left\|\sum_{i=1}^{k+1} c_i \sigma_i(A) u_i\right\|_2 \ge \sigma_{k+1}(A)\|w_0\|_2$. A contradiction. ■

Another interpretation of SVD

The SVD of a rank r matrix $A \in \mathbb{C}^{m\times n}$ ($r \le \min(m,n)$) can be written as a sum of r rank-one matrices

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^* = \sum_{j=1}^{r} \sigma_j Z_j, \quad \text{where} \quad Z_j := u_j v_j^* \ .$$

The $\{Z_j\}_{j=1}^r$ construct part of an orthonormal basis of the $\mathbb{C}^{m\times n}$ space:

$$\langle Z_i \ , \ Z_j \rangle = \mathrm{trace}\left(Z_j^* Z_i\right) = \delta_{ij}$$

Therefore, SVD can be considered as a (partial) Fourier expansion of A in the partial orthonormal basis $\{Z_j\}_{j=1}^r$,

$$\sigma_j = \langle A \ , \ Z_j \rangle$$

can be interpreted as the Fourier coefficient of A in the Z_j "direction".

Why SVD is so fundamental

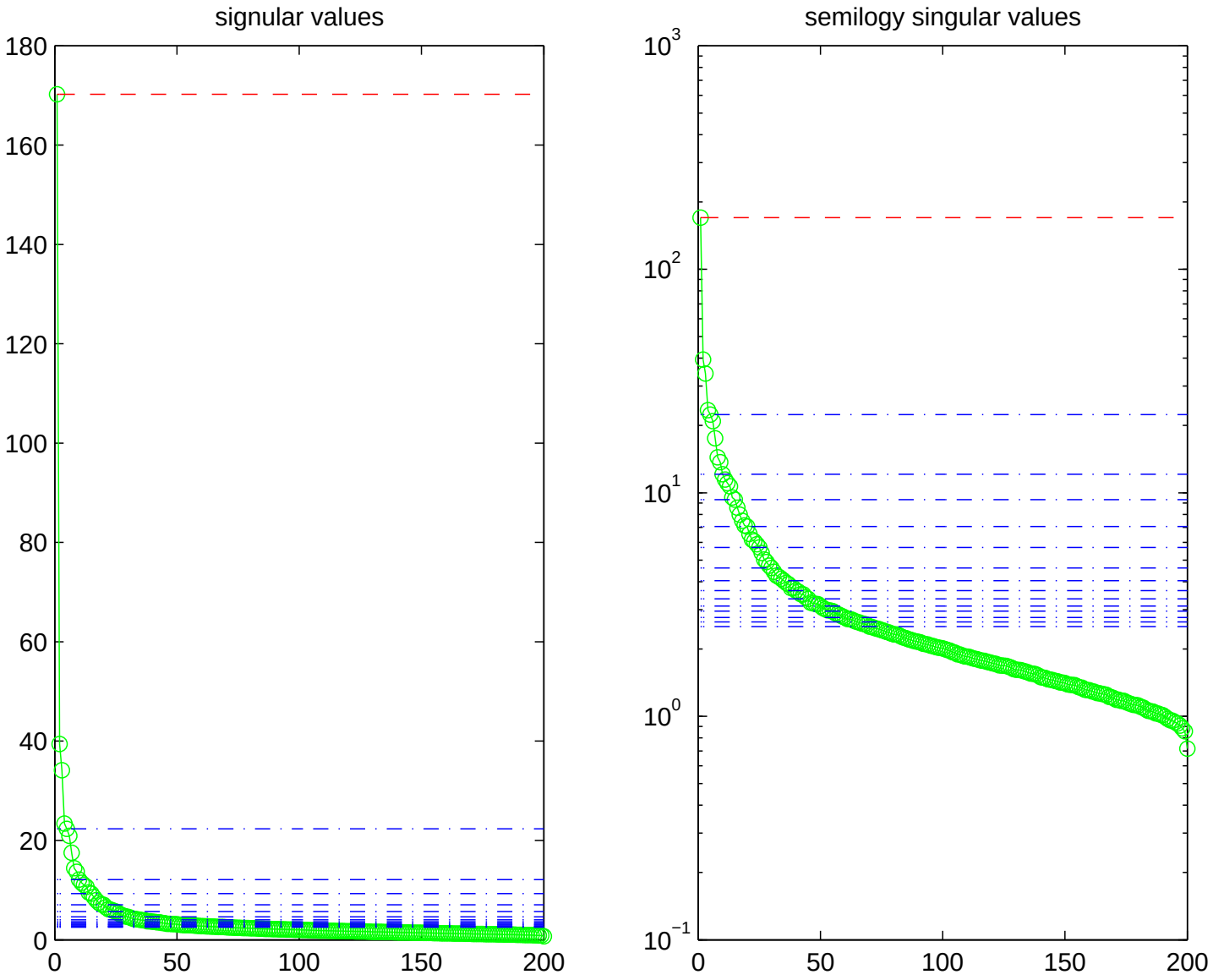
- Provide fundamental matrix properties:
  - Numerical Rank of matrix (counting $\frac{\sigma_j}{\sigma_1}$'s $>$ tolerance)
  - Bases for range and nullspace (in U and V)
  - Define matrix norms (e.g., $\|\cdot\|_2$, $\|\cdot\|_*$, $\|\cdot\|_{S_p}$)
- U and V are unitary — best numerical stability (best conditioning)
  - Least squares fitting; Regularization of ill-conditioned problems
  - U and V unitary/orthogonal provide useful geometric insight
  - Very stable — small changes in A causes only small changes in the SVD
- Large singular values correspond to the principal components; Small singular values correspond to noises (can be truncated)
  - Optimal low-rank approximations (in $\|\cdot\|_{S_p}$ such as $\|\cdot\|_2, \|\cdot\|_F$) conveniently obtained via truncated SVD
  - In most applications, the principal components are essential and noise better be discarded

Why SVD does denoising well

- Random noise (non-directional, e.g., Gaussian white noise) exist almost universally
- Non-directional noise distributes more or less uniformly across each orthonormal basis Z_i
  - Each $\sigma_i Z_i$ contains approximately the same level of noise
  - Signal-to-noise ratio (SNR) in $\sigma_i Z_i$ improves with larger σ_i
  - For σ_i's below some threshold, the noise level basically dominate the signal level in $\sigma_i Z_i$ (i.e., SNR($\sigma_i Z_i$) is too small). In this case, truncating $\sigma_i Z_i$ loses only a small amount of signal, but removes disproportionately large amount of noise.

Application of SVD in Image Compression

- View $m \times n$ image as a (real) matrix A, find best rank k approximation by SVD
- Storage $k(m+n)$ instead of mn
- (When m, n are really large, more economical methods than SVD are needed)



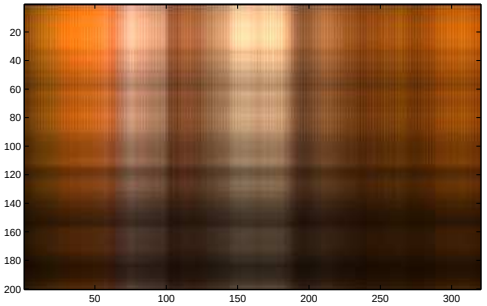
Singular values of the `clown` image, the horizontal lines plot the 1st, 5th, 10th, 15th, 20th, $\cdots$, 65th, 70th singular values.

Original (Rank 200)

[200 x 320] original image

Rank 1

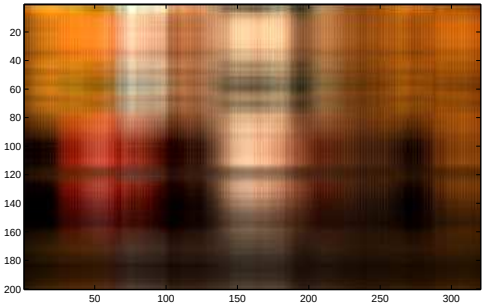
[200 x 320] image, svd rank = 1



Truncation error: sigma(2)/sigma(1)=2.315e−01

Rank 2

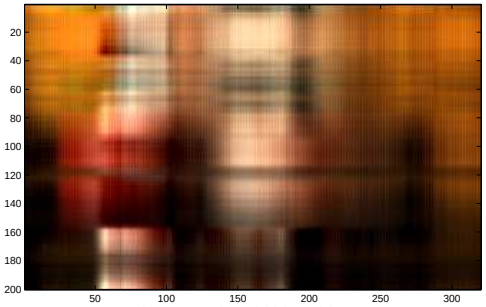
[200 x 320] image, svd rank = 2



Truncation error: sigma(3)/sigma(1)=2.006e−01

Rank 3

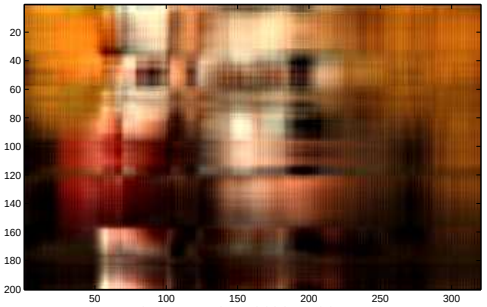
[200 x 320] image, svd rank = 3



Truncation error: sigma(4)/sigma(1)=1.375e−01

Rank 5

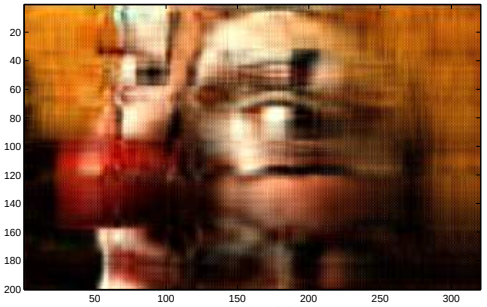
[200 x 320] image, svd rank = 5



Truncation error: sigma(6)/sigma(1)=1.229e−01

Rank 10

[200 x 320] image, svd rank = 10



Truncation error: sigma(11)/sigma(1)=6.738e−02

Rank 15

[200 x 320] image, svd rank = 15

Truncation error: sigma(16)/sigma(1)=5.042e−02

Rank 25

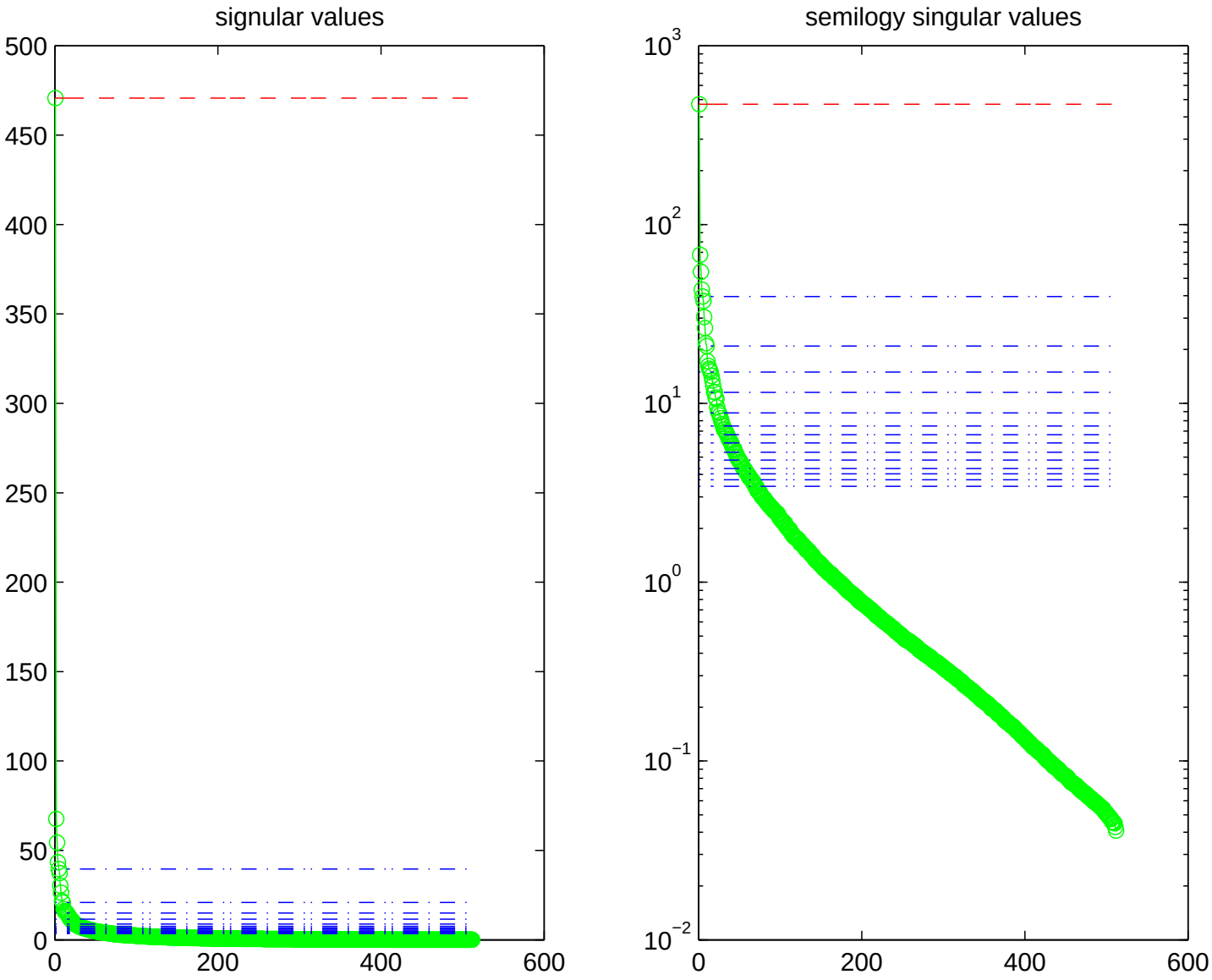
[200 x 320] image, svd rank = 25

Truncation error: sigma(26)/sigma(1)=3.160e−02

Rank 50

[200 x 320] image, svd rank = 50

Truncation error: sigma(51)/sigma(1)=1.795e−02



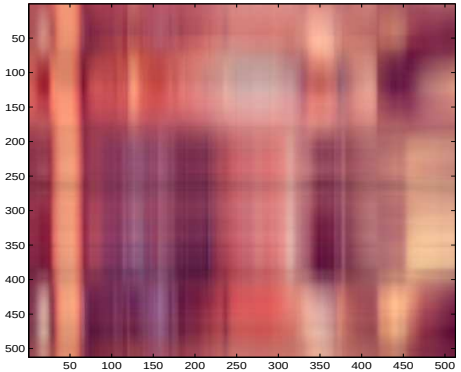
Singular values of the `lena` image, the horizontal lines plot the 1st, 5th, 10th, 15th, 20th, $\cdots$, 65th, 70th singular values.

Original (Rank 512)

[512 x 512] original image

Rank 3

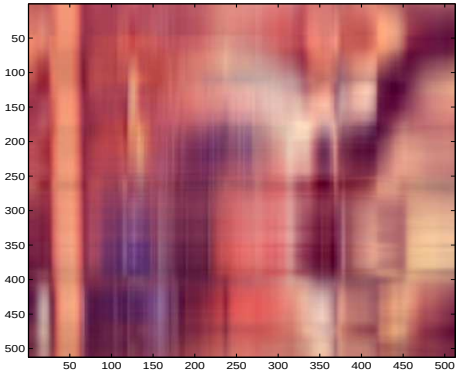
[512 x 512] image, svd rank = 3



Truncation error: sigma(4)/sigma(1)=9.206e−02

Rank 5

[512 x 512] image, svd rank = 5



Truncation error: sigma(6)/sigma(1)=7.938e−02

Rank 10

[512 x 512] image, svd rank = 10

Truncation error: sigma(11)/sigma(1)=3.655e−02

Rank 15

[512 x 512] image, svd rank = 15

Truncation error: sigma(16)/sigma(1)=3.030e−02

Rank 20

[512 x 512] image, svd rank = 20

Truncation error: sigma(21)/sigma(1)=2.292e−02

Rank 30

[512 x 512] image, svd rank = 30

Truncation error: sigma(31)/sigma(1)=1.519e−02

Rank 50

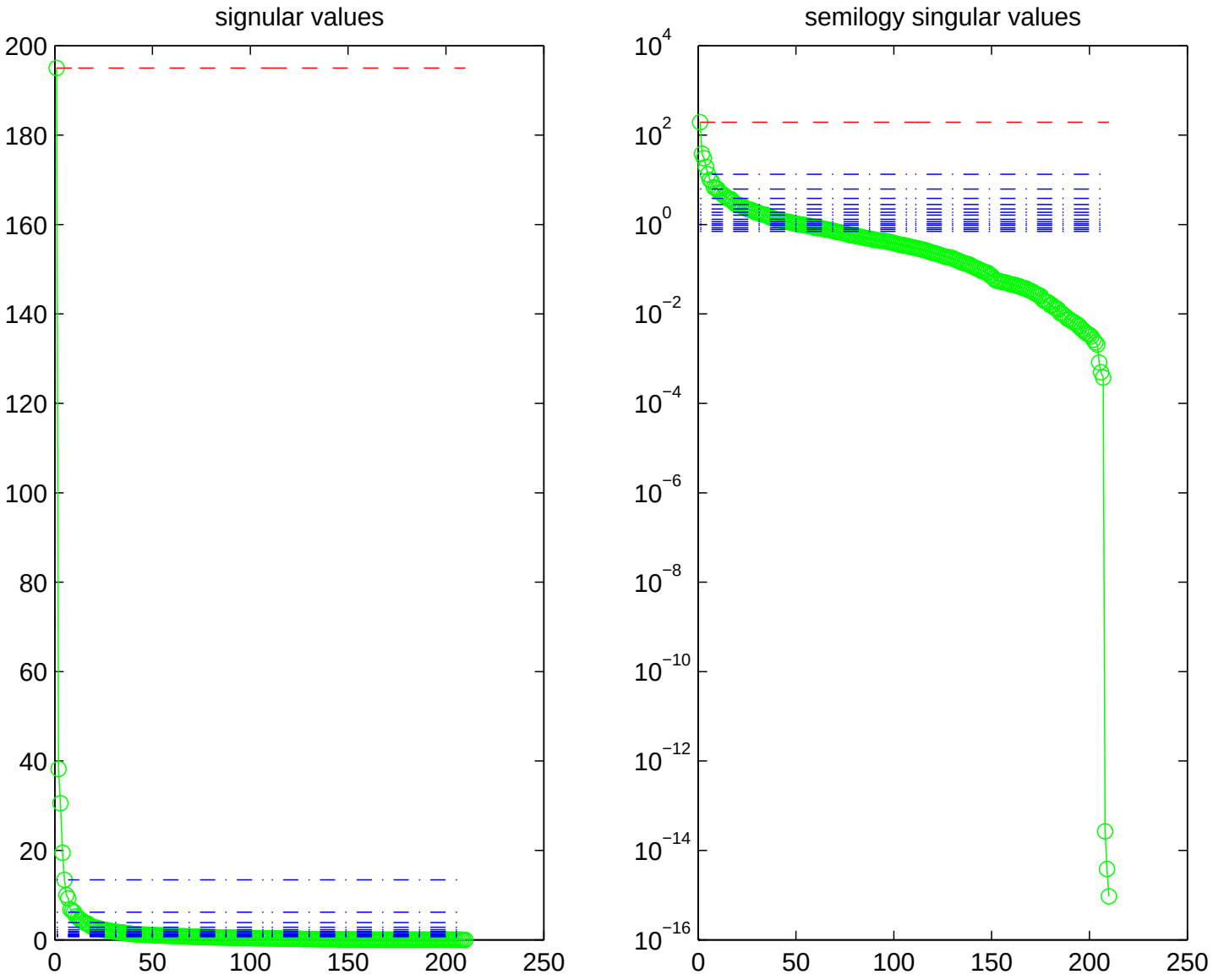
[512 x 512] image, svd rank = 50

Truncation error: sigma(51)/sigma(1)=9.987e−03

Rank 70

[512 x 512] image, svd rank = 70

Truncation error: sigma(71)/sigma(1)=7.237e−03



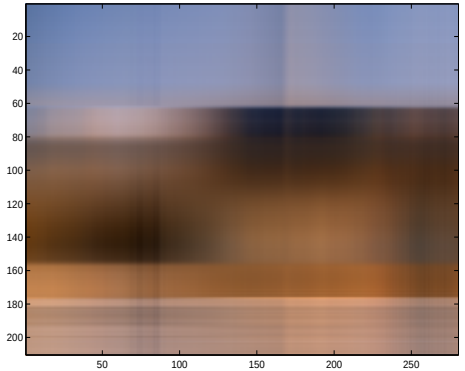
Singular values of the `sand` image (from webshots), the horizontal lines plot the 1st, 5th, 10th, 15th, 20th, $\cdots$, 65th, 70th singular values.

Original (Rank 512)

[210 x 280] original image

Rank 3

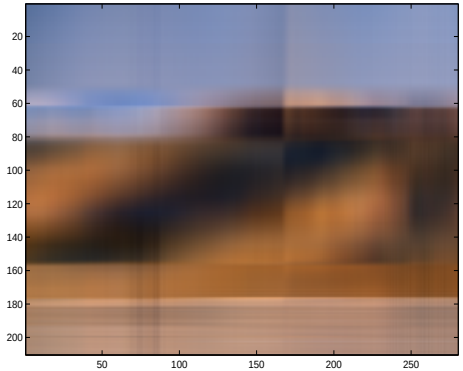
[210 x 280] image, svd rank = 3



Truncation error: sigma(4)/sigma(1)=9.982e−02

Rank 5

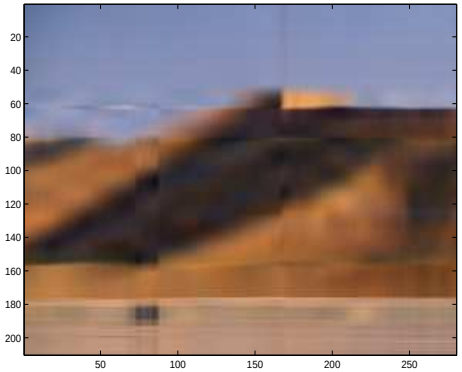
[210 x 280] image, svd rank = 5



Truncation error: sigma(6)/sigma(1)=5.124e−02

Rank 10

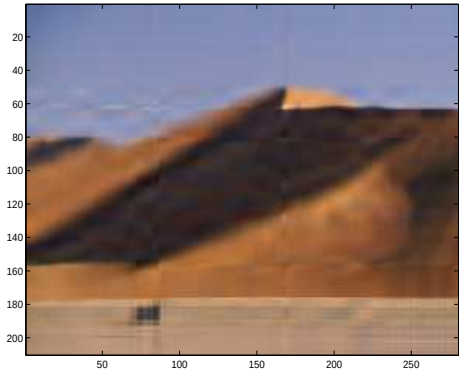
[210 x 280] image, svd rank = 10



Truncation error: sigma(11)/sigma(1)=2.690e−02

Rank 15

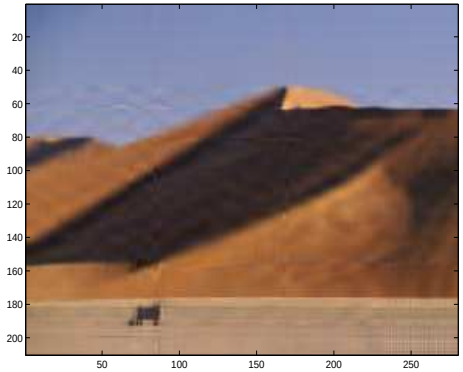
[210 x 280] image, svd rank = 15



Truncation error: sigma(16)/sigma(1)=1.869e−02

Rank 20

[210 x 280] image, svd rank = 20



Truncation error: sigma(21)/sigma(1)=1.390e−02

Rank 30

[210 x 280] image, svd rank = 30

Truncation error: sigma(31)/sigma(1)=9.384e−03

Rank 40

[210 x 280] image, svd rank = 40

Truncation error: sigma(41)/sigma(1)=6.457e−03

Rank 50

[210 x 280] image, svd rank = 50

Truncation error: sigma(51)/sigma(1)=5.271e−03

SVD: Proof of existence

Theorem: (Any matrix has a SVD decomposition)

For any $A \in \mathbb{C}^{m\times n}$, there exist unitary matrices $U \in \mathbb{C}^{m\times m}, V \in \mathbb{C}^{n\times n}$, and a nonnegative diagonal matrix $\Sigma \in \mathbb{C}^{m\times n}$ such that $A = U\Sigma V^{\mathsf{H}}$.

SVD: Proof of existence

Theorem: (Any matrix has a SVD decomposition)

For any $A \in \mathbb{C}^{m\times n}$, there exist unitary matrices $U \in \mathbb{C}^{m\times m}, V \in \mathbb{C}^{n\times n}$, and a nonnegative diagonal matrix $\Sigma \in \mathbb{C}^{m\times n}$ such that $A = U\Sigma V^{\mathsf{H}}$.

Outline of proof: Let $v_1 = \arg\max_{\|x\|_2=1} \|Ax\|_2$. Let $Av_1 = \sigma_1 u_1$ with $\sigma_1 \geq 0, \|u_1\|_2 = 1$ Then clearly $\sigma_1 = \|A\|_2$.

Extend u_1 and v_1 into unitary matrices $\hat{U} = [u_1, U_2]$, $\hat{V} = [v_1, V_2]$, then

$$\hat{U}^{\mathsf{H}} A \hat{V} = \begin{bmatrix} \sigma_1 & w \\ & A_2 \end{bmatrix}, \quad \text{where } A_2 = U_2^{\mathsf{H}} A V_2.$$

Show that $w = 0$. Then apply induction to A_2. ■

SVD: Proof of existence

Theorem: (Any matrix has a SVD decomposition)

For any $A \in \mathbb{C}^{m\times n}$, there exist unitary matrices $U \in \mathbb{C}^{m\times m}, V \in \mathbb{C}^{n\times n}$, and a nonnegative diagonal matrix $\Sigma \in \mathbb{C}^{m\times n}$ such that $A = U\Sigma V^{\mathsf{H}}$.

Outline of proof: Let $v_1 = \arg\max_{\|x\|_2=1} \|Ax\|_2$. Let $Av_1 = \sigma_1 u_1$ with $\sigma_1 \geq 0, \|u_1\|_2 = 1$ Then clearly $\sigma_1 = \|A\|_2$.

Extend u_1 and v_1 into unitary matrices $\hat{U} = [u_1, U_2]$, $\hat{V} = [v_1, V_2]$, then

$$\hat{U}^{\mathsf{H}} A \hat{V} = \begin{bmatrix} \sigma_1 & w \\ & A_2 \end{bmatrix}, \quad \text{where } A_2 = U_2^{\mathsf{H}} A V_2.$$

Show that $w = 0$. Then apply induction to A_2. ■

(Uniqueness: Assume σ_i's are in nonincreasing order. If A is square and σ_j are distinct, then left/right singular vectors u_j, v_j are uniquely determined up to complex signs.)

Proof of Schur Decomposition

Theorem: (Schur decomposition)

Any $A \in \mathbb{C}^{n \times n}$ can be decomposed as $A = QSQ^*$, where Q is unitary and S is upper triangular.

Proof of Schur Decomposition

Theorem: (Schur decomposition)

Any $A \in \mathbb{C}^{n\times n}$ can be decomposed as $A = QSQ^*$, where Q is unitary and S is upper triangular.

Proof. Pick an eigenpair (λ_1, x) of A, with $\|x\|_2 = 1$. Augment x into a unitary $U_1 := [x, U_2]$, then

$$U_1^* A U_1 = \begin{bmatrix} x^*Ax & x^*AU_2 \\ U_2^*Ax & U_2^*AU_2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & x^*AU_2 \\ 0 & U_2^*AU_2 \end{bmatrix}.$$

Apply induction: Assume that $U_2^*AU_2$ has Schur decomposition $Q_2S_2Q_2^*$. Then

$$U_1^* A U_1 = \begin{bmatrix} \lambda_1 & x^*AU_2 \\ 0 & Q_2S_2Q_2^* \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & Q_2 \end{bmatrix} \begin{bmatrix} \lambda_1 & x^*AU_2Q_2 \\ 0 & S_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & Q_2^* \end{bmatrix}.$$

Multiply U_1, U_1^* on both sizes to obtain the Q, S as in $A = QSQ^*$. ■

A Corollary of Schur Decomposition

Corollary: Any normal matrix is unitarily diagonalizable.

A Corollary of Schur Decomposition

Corollary: Any normal matrix is unitarily diagonalizable.

Proof. Let A be a normal matrix with $A = QSQ^*$. Since $AA^* = A^*A$, one must have $SS^* = S^*S$. It now remains to show that a triangular normal matrix must be diagonal. Let

$$S = \begin{bmatrix} s_{11} & t^* \\ & S_2 \end{bmatrix}.$$

Then $|s_{11}|^2 = |s_{11}|^2 + t^*t \Longrightarrow t = 0$. Since S_2 is also normal and upper triangular, one can use the same trick to show that S_2 must be diagonal. ■

Jordan-Wielandt Theorem

This theorem can be stated as an exercise: Let the SVD of $A \in \mathbb{C}^{m\times n}$ be $A = U\Sigma V^{\mathsf{H}}$. Find the eigendecomposition of $\begin{bmatrix} 0 & A^{\mathsf{H}} \\ A & 0 \end{bmatrix}$.

Variational Principle (VP) for σ_i's

Characterization of σ_i's (based on the VP of eigenvalues of hermitian matrices: notice that $\|Ax\|_2 = \sqrt{x^*A^*Ax}$) . Let $A \in \mathbb{C}^{m\times n}$ with SVD

$$A = U\Sigma V^*, \quad \Sigma = \mathrm{diag}(\sigma_i), \; V = [v_1, v_2, \cdots, v_n],$$

with σ_i's in nonincreasing order. Let $\boxed{\mathcal{V}_k = \mathrm{span}\{v_1, \cdots, v_k\}.}$

Then

- $\sigma_1 = \max\{\|Ax\|_2 \;:\; \|x\|_2 = 1, x \in \mathbb{C}^n\}$
- $\sigma_2 = \max\{\|Ax\|_2 \;:\; \|x\|_2 = 1, x \in \mathbb{C}^n, x \perp \mathcal{V}_1\}$
- $\cdots$

$$\boxed{\sigma_{k+1} = \max\left\{\|Ax\|_2 \;:\; \|x\|_2 = 1,\; x \in \mathbb{C}^n,\; x \perp \mathcal{V}_k\right\}.}$$

More generally,

$$\sigma_{k+1} = \min_{\substack{\mathcal{W}_k \subset \mathbb{C}^n \\ \dim(\mathcal{W}_k)=k}} \; \max_{\substack{x\in\mathbb{C}^n,\; \|x\|_2=1 \\ x\perp\mathcal{W}_k}} \left\{\|Ax\|_2\right\}.$$

for $k = 0, 1, \cdots, n-1$.

(semi-) Variational Principle

Theorem: For any $A \in \mathbb{C}^{m\times n}, u \in \mathbb{C}^m, v \in \mathbb{C}^n$,

$$\sigma_{\max}(A) = \max_{u\neq 0}\max_{v\neq 0}\frac{|u^*Av|}{\|u\|_2\,\|v\|_2} = \max_{\|u\|_2=1}\max_{\|v\|_2=1}|u^*Av|.$$

The following generalization is often used to prove the triangular inequalities for the norms defined by various sum of σ_i's.

Theorem: For any $A \in \mathbb{C}^{m\times n}$, with nonincreasing σ_i's,

$$\sum_{i=1}^{k}\sigma_i(A) = \max_{\substack{U\in\mathbb{C}^{m\times k}\\ U^*U=I_k}}\ \max_{\substack{V\in\mathbb{C}^{n\times k}\\ V^*V=I_k}}|\mathrm{trace}\,(U^*AV)|.$$

Proof. Apply SVD and Cauchy inequality. ■

Pseudo-inverse (Generalized Inverse)

Given $A \in \mathbb{C}^{m\times n}$, $A^+ \in \mathbb{C}^{n\times m}$ is called pseudo-inverse of A if

1. $AA^+A = A$
2. $A^+AA^+ = A^+$

Such A^+ always exists, but uniqueness is not guaranteed. A^+ is called the *Moore-Penrose pseudoinverse* of A if a further condition is added

3. Both AA^+ and A^+A are hermitian

This condition guarantees uniqueness. In practice pseu-doinverse A^+ mainly refers to the *Moore-Penrose pseudoinverse*.

If the full SVD of $A \in \mathbb{C}^{m\times n}$ is $A = U\Sigma V^*$, where $\begin{bmatrix} \Sigma_k & \\ & 0 \end{bmatrix} \in \mathbb{R}^{m\times n}$, with $\sigma_k > 0$ and $\sigma_j = 0, j > k$, then A^+ can be easily obtained by

$$A^+ = V\Sigma^+U^*, \qquad \text{where } \Sigma^+ = \begin{bmatrix} \Sigma_k^{-1} & \\ & 0 \end{bmatrix} \in \mathbb{R}^{n\times m}.$$

Some properties of pseudo-inverse

- $\ker(A^+) = \ker(A^*)$, $\quad \mathrm{range}(A^+) = \mathrm{range}(A^*)$
- $(A^+)^+ = A$
- $(A^\intercal)^+ = (A^+)^\intercal$, $\overline{A}^+ = \overline{A^+}$, $(A^*)^+ = (A^+)^*$
- $$\begin{aligned} A &= AA^*A^{*+} = A^{*+}A^*A \\ A^+ &= A^+A^{+*}A^* = A^*A^{+*}A^+ \end{aligned}$$
- $A^+ = (A^*A)^+A^*$, $\quad$ If A has full column-rank, $A^+ = (A^*A)^{-1}A^*$
- $A^+ = A^*(AA^*)^+$, $\quad$ If A has full row-rank, $A^+ = A^*(AA^*)^{-1}$
- $$A^+ = \lim_{\delta \searrow 0} (A^*A + \delta I)^{-1}A^* = \lim_{\delta \searrow 0} A^*(AA^* + \delta I)^{-1}$$
- $(AA^+)^2 = AA^+$, $\quad (A^+A)^2 = A^+A$
  (important, related to orthogonal-projectors (later))

On QR decompositions

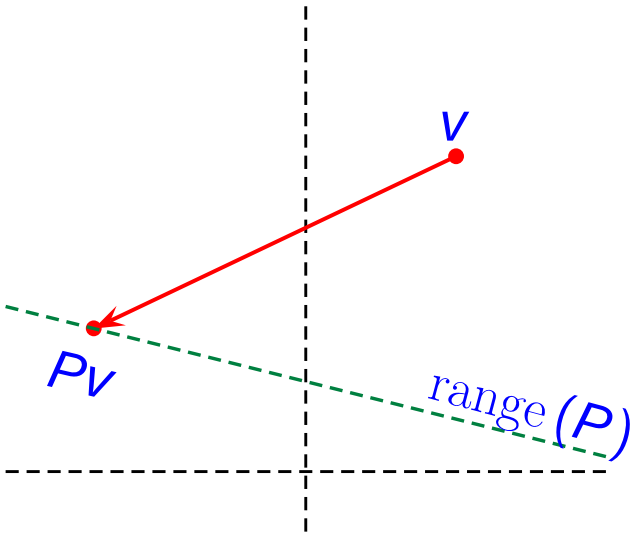
- Projectors, orthogonal projectors, reflectors
- Computing QR factorization (GS, MGS, Householder, Givens)
- Solving least squares by QR and SVD

Projectors

- A *projector* is a square matrix P that satisfies

$$P^2 = P$$

- If $v \in \mathrm{range}(P)$, then $Pv = v$
  - Since with $v = Px$, $Pv = P^2x = Px = v$
- Projection along the line $Pv - v \in \mathrm{null}(P)$
  - Since $P(Pv - v) = P^2v - Pv = 0$



- $P^2 = P$ is not enough to introduce an *orthogonal projector* (later)

Complementary Projectors, Complementary Subspaces

- For projector P, the matrix $I - P$ is its *complementary projector*
- $I - P$ projects on the nullspace of P:
  - If $Pv = 0$, then $(I - P)v = v$, so $\mathrm{null}(P) \subseteq \mathrm{range}(I - P)$
  - For any $y \in \mathrm{range}(I - P)$, $y = (I - P)v$, then $Py = (P - P^2)v = 0$; so $\mathrm{range}(I - P) \subseteq \mathrm{null}(P)$
  - Therefore

$$\mathrm{range}(I - P) = \mathrm{null}(P), \qquad \mathrm{null}(I - P) = \mathrm{range}(P)$$

  That is,

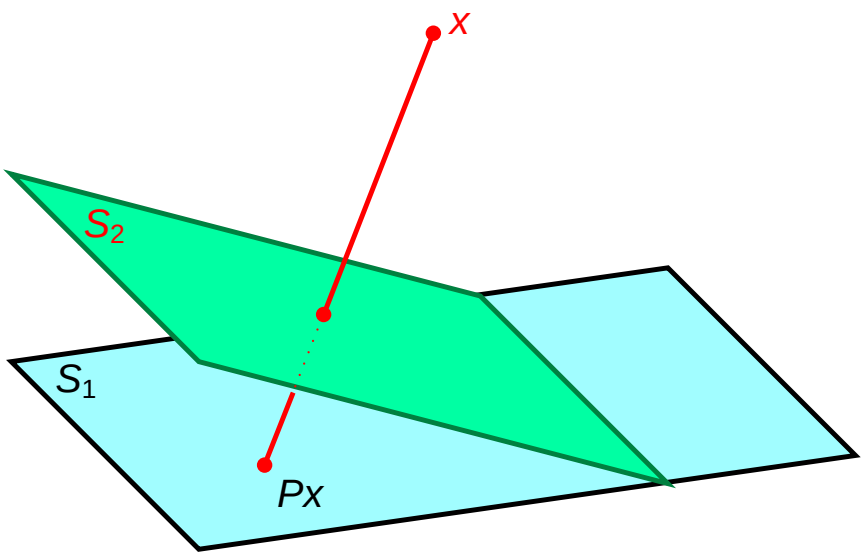
$$\mathrm{null}(I - P) \cap \mathrm{null}(P) = \{0\}$$

$$\text{or,} \qquad \mathrm{range}(P) \cap \mathrm{null}(P) = \{0\}$$

- A projector separates $\mathbb{C}^m$ into two spaces S_1, $S_2^\perp$, with $\mathrm{range}(P) = S_1$ and $\mathrm{null}(P) = S_2^\perp$.
  That is, P is the projector *along* $\mathrm{null}(P)$ *onto* $\mathrm{range}(P)$.
- Any $x \in \mathbb{C}^m$ can be decomposed as $x = x_1 + x_2$, where $x_1 \in \mathrm{range}(P), x_2 \in \mathrm{null}(P)$: $\boxed{x = Px + (I - P)x.}$

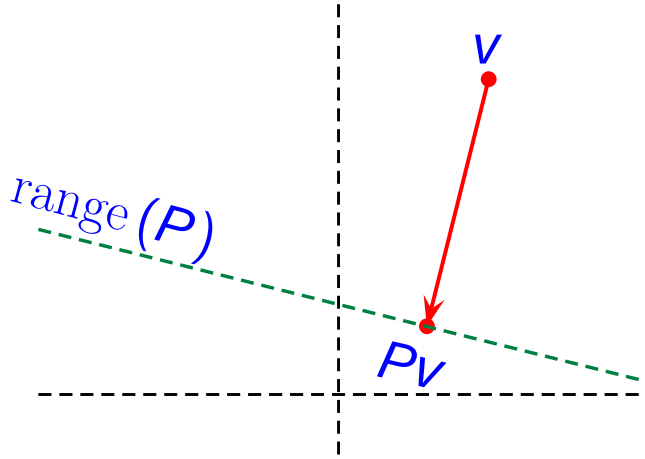
A more general view of a projector

Lemma: Given any two dimension n ($n < m$) subspaces S_1 and S_2, if S_1 and S_2 are not orthogonal (i.e., $S_1 \cap S_2^{\perp} = \{0\}$), then for any $x \in \mathbb{C}^m$, there exists a projector P such that $Px \in S_1, \;\; x - Px \perp S_2$.
(Px is the unique projection of x onto S_1 along $S_2^{\perp}$.
And the projector is called a projector onto S_1 along $S_2^{\perp}$.)



When $S_1 = S_2$, the projector is called an orthogonal projector, otherwise it is called an oblique projector.

Orthogonal Projectors



- Definition (geometric): A projector P is orthogonal if

$$\boxed{\mathrm{range}(P) = (\mathrm{null}(P))^{\perp}}$$

- (More generally, an *orthogonal projector* projects onto a subspace S_1 along a subspace $S_2^{\perp}$ which is orthogonal to S_1.)
- Definition (algebraic): A projector P is orthogonal if $\boxed{P^* = P}$
- Definition (analytic): A projector P is orthogonal if $\boxed{\|P\|_2 = 1}$

Equivalence of the definitions

Theorem: For any projector P,

$$\mathrm{range}(P) = (\mathrm{null}(P))^{\perp} \quad \Longleftrightarrow \quad P = P^*.$$

Proof. The ($\Longleftarrow$) part is straightforward by the known fact (related to the Rank-nullity theorem) that

$$\mathrm{range}(P^*) = (\mathrm{null}(P))^{\perp}.$$

For the ($\Longrightarrow$) part: Given any $x \in \mathbb{C}^m$, let $y = Px \in \mathrm{range}\,(P)$. Since $\mathrm{range}(P) = (\mathrm{null}(P))^{\perp} = \mathrm{range}(P^*)$, $y \in \mathrm{range}\,(P^*)$. Now apply the properties of a projector,

$$y = Py = P^2x = Px$$
$$y = P^*y = P^*Px,$$

which lead to $Px = P^*Px$, or $(P - P^*P)x = 0$, for all $x \in \mathbb{C}^m$. This is only possible when $P = P^*P$, taking conjugate transpose gives

$$P = P^* = P^*P. \quad \blacksquare$$

Equivalence of the definitions

Theorem: For any projector P,

$$\|P\|_2 = 1 \quad \Longleftrightarrow \quad P = P^*.$$

Proof. The ($\Longleftarrow$) part is straightforward and can be proved in several different ways, we list two here:

(1) $P = P^* \implies P$ is unitarily diagonalizable, let $P = Q\Lambda Q^*$, then $P^2 = P \implies \Lambda^2 = \Lambda \implies \Lambda$ can only have 1 or 0 on its diagonal $\implies \|P\|_2 = 1$.

(2) $\langle Px\ ,\ Px\rangle = \langle x\ ,\ P^*Px\rangle = \left\langle x\ ,\ P^2x\right\rangle = \langle x\ ,\ Px\rangle$
$\implies \|Px\|_2^2 \le \|x\|_2\,\|Px\|_2 \implies \|Px\|_2 \le \|x\|_2 \implies \|P\|_2 \le 1.$
But since $\|P\|_2 \ge 1$ for all $P^2 = P$, it must be $\|P\|_2 = 1$.

The ($\Longrightarrow$) part is more involved but can also be proved in several different ways. One of them is to use the fact that $\sin(\theta) = \frac{1}{\|P\|_2}$, where θ is the angle between $\mathrm{range}\,(P)$ and $\mathrm{null}(P)$. Therefore $\|P\|_2 = 1$ implies that $\mathrm{range}\,(P) \perp \mathrm{null}(P)$, which is equivalent to P being orthogonal, from previous equivalence proof we get $P = P^*$. ■

Two other proofs based on matrix decompositions are listed below.

Prove that $P^2 = P, \|P\|_2 = 1 \implies P = P^*$

- By SVD of P: Assume $\operatorname{rank}(P) = k \le m$. Let $P = U_k \Sigma_k V_k^*$ be the TSVD of P, with Σ_k nonsingular. Then

$$P^2 = P \implies \Sigma_k V_k^* U_k \Sigma_k = \Sigma_k \implies V_k^* U_k = \Sigma_k^{-1}.$$

  Therefore $V_k^* U_k$ is diagonal. In addition, since U_k, V_k are columns of unitary matrices, the diagonal elements of $V_k^* U_k$ are all ≤ 1 by Cauchy inequality. But since $\|P\|_2 = 1$, we have $\sigma_i(P) \le 1$. Hence it must be that $V_k^* U_k = \Sigma_k^{-1} = I_k$, therefore $U_k = V_k$, and $P = U_k \Sigma_k U_k^* = P^*$. ■

  (Comment: This proof shows that the singular values, as well as eigenvalues, of an orthogonal projector must be 1 or 0.)

- By Schur-decomposition of P: Let $P = QSQ^*$, then $P^2 = P \implies S^2 = S$. Let $\operatorname{diag}(S) = (s_{ii})$, comparing diagonal elements of $S^2 = S$ we have $s_{ii} = 1$ or 0 for all i. Assume S is ordered as $S = \begin{bmatrix} S_{11} & S_{12} \\ & S_{22} \end{bmatrix}$, where $\operatorname{diag}(S_{11}) = I_k$, $\operatorname{diag}(S_{22}) = 0_{m-k}$. Then clearly $S_{22}^2 = S_{22} \implies S_{22} = (0)_{m-k}$. Now use the condition $\|S\|_2 = 1$ to show that $S_{12} = (0)$ and $S_{11} = I_k$: Let $s_{i:} = S(i,:), i = 1:k$, by variational principal, $\sigma_1(S) = 1 \ge \frac{e_i^* S s_{i:}^*}{\|e_i\|_2 \|s_{i:}\|_2} = \frac{s_{i:} s_{i:}^*}{\|s_{i:}\|_2} = \|s_{i:}\|_2$. ■

Equivalence of the definitions

Theorem: For any projector P,

$$\mathrm{range}(P) = (\mathrm{null}(P))^{\perp} \iff \|P\|_2 = 1.$$

(straightforward from the previous two equivalences, however, it is a good exercise to show the equivalence directly)

Equivalence of the definitions

Theorem: For any projector P,

$$\mathrm{range}(P) = (\mathrm{null}(P))^{\perp} \iff \|P\|_2 = 1.$$

Proof. For the ($\Longleftarrow$) part, $\|P\|_2 \geq 1$ easily follows from $P^2 = P$. Now show $\|P\|_2 \leq 1$: Since $\mathrm{range}(P) \perp \mathrm{null}(P)$, and $(I - P)x \in \mathrm{null}(P)$ for any x, $x = Px + (I - P)x$ is an orthogonal decomposition, by the Pythagorean theorem $\|x\|_2 \geq \|Px\|_2$, hence $\|P\|_2 \leq 1$.

For the ($\Longrightarrow$) part: Given any nonzero x, y, with $x \in \mathrm{range}(P)$, $y \in \mathrm{null}(P)$, need to show $x \perp y$:

Decompose x as $x = \alpha y + r$ where $r \perp y$ and $\alpha \in \mathbb{C}$, then by the Pythagorean theorem, $\|x\|_2^2 = |\alpha|^2 \|y\|_2^2 + \|r\|_2^2$. However, P is a projector with $\|P\|_2 = 1$,

$$x = Px = \alpha Py + Pr = Pr \implies \|x\|_2 = \|Pr\|_2 \leq \|P\|_2 \|r\|_2 = \|r\|_2 .$$

This is only possible when $\alpha = 0$, i.e., $x = r \implies x \perp y$. Therefore $\mathrm{range}(P) \perp \mathrm{null}(P) \implies P$ is orthogonal. ■

Projection with orthonormal basis

Given $V \in \mathbb{C}^{m\times k}$ with orthonormal columns, (i.e., $V^*V = I_k$), find the orthogonal projectors P_V and $P_{V_\perp}$ that projects onto $\mathrm{range}\,(V)$ and $(\mathrm{range}\,(V))^\perp$ respectively.

Projection with orthonormal basis

Given $V \in \mathbb{C}^{m\times k}$ with orthonormal columns, (i.e., $V^*V = I_k$), find the orthogonal projectors P_V and $P_{V_\perp}$ that projects onto $\mathrm{range}\,(V)$ and $(\mathrm{range}\,(V))^\perp$ respectively.

- Note that an orthogonal P needs to satisfy $P^2 = P = P^*$
- Since $\mathrm{range}\,(P_V) = \mathrm{range}\,(V)$,

$$P_V = VV^*.$$

- The complement $I - P_V$ is the $P_{V_\perp}$, (note $[V, V_\perp]$ is unitary)

$$P_{V_\perp} = V_\perp {V_\perp}^* = I - VV^* \ .$$

- Special cases
  - Rank-1 orthogonal projector (project onto a unit direction q)

$$P_q = qq^*$$

  - Rank $m-1$ orthogonal projector (eliminate component in a unit direction q)

$$P_{q_\perp} = I - qq^* \qquad \text{(also written as } P_{\perp q}\text{)}$$

Projection with arbitrary basis

Given $A \in \mathbb{C}^{m\times k}$ with $\mathrm{rank}\,(A) = k$, for the orthogonal projectors P_A and $P_{A_\perp}$ that projects onto $\mathrm{range}\,(A)$ and $(\mathrm{range}\,(A))^\perp$ respectively.

Projection with arbitrary basis

Given $A \in \mathbb{C}^{m\times k}$ with $\mathrm{rank}\,(A) = k$, for the orthogonal projectors P_A and $P_{A_\perp}$ that projects onto $\mathrm{range}\,(A)$ and $(\mathrm{range}\,(A))^\perp$ respectively.

- Easily done if QR decomposition of A is available.
- Can do without QR of A:

Projection with arbitrary basis

Given $A \in \mathbb{C}^{m\times k}$ with $\mathrm{rank}\,(A) = k$, for the orthogonal projectors P_A and $P_{A_\perp}$ that projects onto $\mathrm{range}\,(A)$ and $(\mathrm{range}\,(A))^\perp$ respectively.

- Easily done if QR decomposition of A is available.
- Can do without QR of A:
-
  - For any $v \in \mathbb{C}^m$, $P_A v \in \mathrm{range}(A)$. Then
  $$P_A v - v \perp \mathrm{range}(A), \quad \text{or} \quad A^*(P_A v - v) = 0,$$
  - Set $P_A v = Ax$, then
  $$A^*(Ax - v) = 0 \iff A^*Ax = A^*v$$
  - Since A^*A is nonsingular,
  $$x = (A^*A)^{-1}A^*v$$
  - Finally, $P_A v = Ax = A(A^*A)^{-1}A^*v$, giving the orthogonal projector
  $$P_A = A(A^*A)^{-1}A^*; \quad \text{by complement} \quad P_{A_\perp} = I - P_A\ .$$

Projection with arbitrary basis

Given $A \in \mathbb{C}^{m\times k}$ with $\mathrm{rank}\,(A) = k$, for the orthogonal projectors P_A and $P_{A_\perp}$ that projects onto $\mathrm{range}\,(A)$ and $(\mathrm{range}\,(A))^\perp$ respectively.

- Easily done if QR decomposition of A is available.
- Can do without QR of A:

- Another way to look at it:
  - Since $\mathrm{range}(P_A) \subseteq \mathrm{range}(A)$ and $P^* = P$, we have $P_A = AMA^*$ for some $M = M^* \in \mathbb{C}^{k\times k}$
  - Since $P^2 = P$, we have $AMA^*AMA^* = AMA^*$
  - Notice that A^*A is nonsingular, we pick $M = (A^*A)^{-1}$, which readily makes $P_A = AMA^* = A(A^*A)^{-1}A^*$ an orthogonal projector (since $P^2 = P = P^*$) to $\mathrm{range}(A)$.

Relation to pseudo-inverse

Recall that

$$A^+ = (A^*A)^+A^* = A^*(AA^*)^+$$

If A has full column rank,

$$A^+ = (A^*A)^{-1}A^*$$

So the orthogonal projector that projects onto $\mathrm{range}\,(A)$ (column space of A) is

$$P_A = A(A^*A)^+A^* = AA^+ \ .$$

Similarly, the orthogonal projector that projects onto $\mathrm{range}\,(A^*)$ (row space of A) is

$$P_{A^*} = A^*(AA^*)^+A = A^+A \ .$$

The QR Factorization (main idea)

- Find orthonormal vectors that span the successive spaces spanned by the columns of A:

$$\langle a_1 \rangle \subseteq \langle a_1, a_2 \rangle \subseteq \langle a_1, a_2, a_3 \rangle \subseteq \ldots$$

- This means that (for full rank A),

$$\left\langle q_1, q_2, \ldots, q_j \right\rangle = \left\langle a_1, a_2, \ldots, a_j \right\rangle, \quad \text{for } j = 1, \ldots, n$$

The QR Factorization (matrix structure)

- In matrix form, $\langle q_1, q_2, \ldots, q_j \rangle = \langle a_1, a_2, \ldots, a_j \rangle$ becomes

$$\left[\begin{array}{c|c|c|c} a_1 & a_2 & \cdots & a_n \end{array}\right] = \left[\begin{array}{c|c|c|c} q_1 & q_2 & \cdots & q_n \end{array}\right] \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & & \vdots \\ & & \ddots & \vdots \\ & & & r_{nn} \end{bmatrix}$$

or

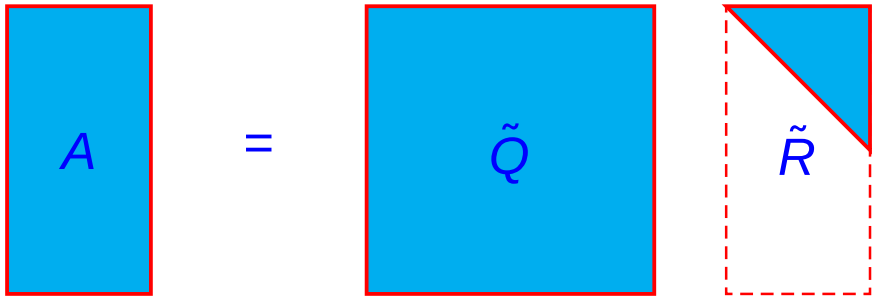
$$A = QR$$

- This is the *thin QR* factorization (also called *reduced QR*)
- Orthogonal extension from $Q \in \mathbb{C}^{m \times n}$ to $\tilde{Q} = [Q, Q_\perp] \in \mathbb{C}^{m \times m}$, and adding zero rows to R gives the *full QR* factorization .

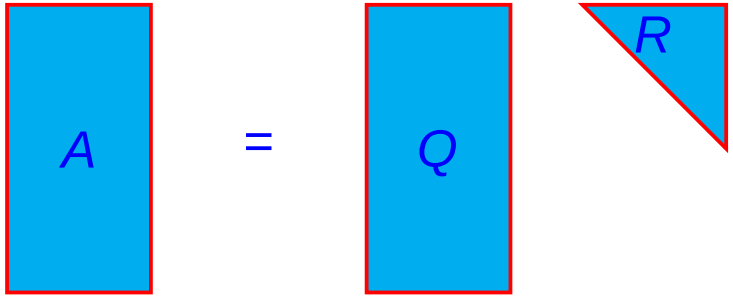
The structure of full and thin QR Factorizations

Let A be an $m \times n$ matrix (for $m \geq n$)

- The full QR factorization is $A = \tilde{Q}\tilde{R}$, where $\tilde{Q}$ is $m \times m$ unitary, $\tilde{R}$ is $m \times n$ upper-triangular.



- The *thin* QR Factorization is more compact, $A = QR$, where Q is the first $m \times n$ part of $\tilde{Q}$, R is the top $n \times n$ upper-triangular part of $\tilde{R}$



Gram-Schmidt Orthogonalization

- Find new q_j orthogonal to $q_1, \ldots, q_{j-1}$ by subtracting components along previous vectors

$$v_j = a_j - (q_1^* a_j) q_1 - (q_2^* a_j) q_2 - \cdots - (q_{j-1}^* a_j) q_{j-1}$$

- Normalize to get $q_j = v_j / \|v_j\|$
- We then obtain a reduced QR factorization $A = QR$, with

$$r_{ij} = q_i^* a_j, \quad (i \neq j)$$

and

$$|r_{jj}| = \|a_j - \sum_{i=1}^{j-1} r_{ij} q_i\|_2$$

Classical Gram-Schmidt

- Straight-forward application of Gram-Schmidt orthogonalization
- Numerically unstable

Algorithm: Classical Gram-Schmidt
for $j = 1$ **to** n
  $v_j = a_j$
  for $i = 1$ **to** $j - 1$
    $r_{ij} = q_i^* a_j$
    $v_j = v_j - r_{ij} q_i$
  $r_{jj} = \|v_j\|_2$
  $q_j = v_j / r_{jj}$

Matlab implementation (uses BLAS-2)

```
R(1,1) = norm(A(:,1));  Q(:,1) = A(:,1)/R(1,1);
for j = 2 : n,
    R(1:j-1,j) = Q(:,1:j-1)'*A(:,j);
    Q(:,j) = A(:,j) - Q(:,1:j-1) * R(1:j-1,j);
    R(j,j) = norm(Q(:,j));
    if ( R(j,j) == 0 ),  error(['columns linearly dependent']);  end
    Q(:,j) = Q(:,j)/R(j,j);
end
```

Existence and Uniqueness

- Every $A \in \mathbb{C}^{m\times n}$ ($m \geq n$) has a full QR factorization and a thin QR factorization.
  Proof. For full rank A, Gram-Schmidt process proves existence of thin $A = QR$. Otherwise, when $v_j = 0$ choose arbitrary vector orthogonal to previous q_i.
  For full QR, add orthogonal extension to Q and zero rows to R.

- Each $A \in \mathbb{C}^{m\times n}$ ($m \geq n$) of full rank has a unique thin QR decomposition $A = QR$, with $r_{jj} > 0$.
  Proof. Again Gram-Schmidt, $r_{jj} > 0$ uniquely determines the sign.

Gram-Schmidt Projections

- The orthogonal vectors produced by Gram-Schmidt can be written in terms of orthogonal projectors

$$q_1 = \frac{P_1 a_1}{\|P_1 a_1\|}, \quad q_2 = \frac{P_2 a_2}{\|P_2 a_2\|}, \quad \ldots, \quad q_n = \frac{P_n a_n}{\|P_n a_n\|}$$

where

$$P_j = I - \hat{Q}_{j-1}\hat{Q}_{j-1}^* \text{ with } \hat{Q}_{j-1} = \left[\begin{array}{c|c|c|c} q_1 & q_2 & \cdots & q_{j-1} \end{array}\right]$$

- P_j projects orthogonally onto the space orthogonal to $\langle q_1, \ldots, q_{j-1} \rangle$, and $\mathrm{rank}(P_j) = m - (j-1)$

The Modified Gram-Schmidt (MGS) Algorithm

- The projection P_j can equivalently be written as

$$P_j = P_{\perp q_{j-1}} \cdots P_{\perp q_2} P_{\perp q_1}$$

where

$$P_{\perp q} = I - qq^*$$

- $P_{\perp q}$ projects orthogonally onto the space orthogonal to q, and $\mathrm{rank}(P_{\perp q}) = m - 1$
- The *Classical Gram-Schmidt* algorithm computes an orthogonal vector by

$$v_j = P_j a_j$$

while the *Modified Gram-Schmidt* algorithm uses

$$v_j = P_{\perp q_{j-1}} \cdots P_{\perp q_2} P_{\perp q_1} a_j$$

Classical vs. Modified Gram-Schmidt

- MGS is only a simple modification of CGS: use the most current vector for projection (e.g., orth. $A = [a_1, \cdots, a_n] \in \mathbb{C}^{m\times n}$)

Classical GS (CGS)

1. For $j = 1, ..., n$ Do:
2. $q_j := a_j$
3. For $i = 1, \ldots, j-1$ Do
   $r_{ij} = \langle a_j \ , \ q_i \rangle$
   $q_j := q_j - r_{ij} q_i$
4. EndDo
5. $r_{jj} = \|q_j\|_2$. If $r_{jj} = 0$ exit
6. $q_j := q_j / r_{jj}$
7. EndDo

Modified GS (MGS)

1. For $j = 1, ..., n$ Do:
2. $q_j := a_j$
3. For $i = 1, \ldots, j-1$ Do
   $r_{ij} = \langle q_j \ , \ q_i \rangle$
   $q_j := q_j - r_{ij} q_i$
4. EndDo
5. $r_{jj} = \|q_j\|_2$. If $r_{jj} = 0$ exit
6. $q_j := q_j / r_{jj}$
7. EndDo

- The above MGS partially uses $P^2 = P$ for any orthogonal projector P (theoretically equivalent, numerically not equivalent)

Question: For this version of MGS, is there a BLAS-2 implementation of steps 3–4, such as that for CGS?

MGS (BLAS-2 version)

Can reorganize MGS s.t. inner loops use BLAS-2 operations, as in CGS. Compute R row by row instead of column by column.

Modified GS (MGS2)

For $j = 1, ..., n$ Do:
  $q_j := a_j$
EndDo
For $j = 1, \ldots, n$ Do
  $r_{jj} = \|q_i\|_2$
  If $r_{jj} = 0$ exit
  $q_j := q_j / r_{jj}$
  For $i = j + 1, ..., n$ Do:
    $r_{ji} = \langle q_i \, , \, q_j \rangle$
    $q_i := q_i - r_{ji} q_j$
  EndDo
EndDo

```
Q = A;  R = zeros(n,n);
for j = 1 : n,
  R(j,j) = norm(Q(:,j));
    if ( R(j,j) == 0 ),
       error('linearly dependent columns');
    end
    Q(:,j) = Q(:,j) / R(j,j);
    R(j,j+1:n)=Q(:,j)'*Q(:,j+1:n);
    Q(:,j+1:n)=Q(:,j+1:n)-Q(:,j)*R(j,j+1:n);
end
```

This version of MGS essentially uses the relation

$P_j = P_{\perp q_{j-1}} \cdots P_{\perp q_2} P_{\perp q_1}$ and do individual projection one by one, while CGS apply P_j at once.

- Compare CGS with MGS for the vectors (choose ϵ s.t. $1+\epsilon^2 \approx 1$)

$$a_1 = (1,\epsilon,0,0)^T, \quad a_2 = (1,0,\epsilon,0)^T, \quad a_3 = (1,0,0,\epsilon)^T$$

- Classical:

$$\begin{aligned}
v_1 &\leftarrow (1,\epsilon,0,0)^T, \quad r_{11} = \sqrt{1+\epsilon^2} \approx 1, \quad q_1 = v_1/1 = (1,\epsilon,0,0)^T \\
v_2 &\leftarrow (1,0,\epsilon,0)^T, \quad r_{12} = q_1^T a_2 = 1, \quad v_2 \leftarrow v_2 - 1q_1 = (0,-\epsilon,\epsilon,0)^T \\
&\quad r_{22} = \sqrt{2}\epsilon, \quad q_2 = v_2/r_{22} = (0,-1,1,0)^T/\sqrt{2} \\
v_3 &\leftarrow (1,0,0,\epsilon)^T, \quad r_{13} = q_1^T a_3 = 1, \quad v_3 \leftarrow v_3 - 1q_1 = (0,-\epsilon,0,\epsilon)^T \\
&\quad r_{23} = q_2^T a_3 = 0, \quad v_3 \leftarrow v_3 - 0q_2 = (0,-\epsilon,0,\epsilon)^T \\
&\quad r_{33} = \sqrt{2}\epsilon, \quad q_3 = v_3/r_{33} = (0,-1,0,1)^T/\sqrt{2}
\end{aligned}$$

- Modified:

$$\begin{aligned}
v_1 &\leftarrow (1,\epsilon,0,0)^T, \quad r_{11} = \sqrt{1+\epsilon^2} \approx 1, \quad q_1 = v_1/1 = (1,\epsilon,0,0)^T \\
v_2 &\leftarrow (1,0,\epsilon,0)^T, \quad r_{12} = q_1^T v_2 = 1, \quad v_2 \leftarrow v_2 - 1q_1 = (0,-\epsilon,\epsilon,0)^T \\
&\quad r_{22} = \sqrt{2}\epsilon, \quad q_2 = v_2/r_{22} = (0,-1,1,0)^T/\sqrt{2} \\
v_3 &\leftarrow (1,0,0,\epsilon)^T, \quad r_{13} = q_1^T v_3 = 1, \quad v_3 \leftarrow v_3 - 1q_1 = (0,-\epsilon,0,\epsilon)^T \\
&\quad r_{23} = q_2^T v_3 = \epsilon/\sqrt{2}, \quad v_3 \leftarrow v_3 - r_{23}q_2 = (0,-\epsilon/2,-\epsilon/2,\epsilon)^T \\
&\quad r_{33} = \sqrt{6}\epsilon/2, \quad q_3 = v_3/r_{33} = (0,-1,-1,2)^T/\sqrt{6}
\end{aligned}$$

- Check Orthogonality:
  - Classical: $q_2^T q_3 = (0,-1,1,0)(0,-1,0,1)^T/2 = 1/2$
  - Modified: $q_2^T q_3 = (0,-1,1,0)(0,-1,-1,2)^T/\sqrt{12} = 0$
- MGS is numerically stable (less sensitive to rounding errors)

Flops counts of Gram-Schmidt QR

- Count each $+, -, *, /, \sqrt{\cdot}$ as one flop, only look at the higher order terms
- Orthonormalize $A \in \mathbb{R}^{m\times n}, (m \geq n)$

Modified GS (MGS)

1. For $j = 1, ..., n$ Do:
2. $\quad q_j := a_j$
3. $\quad$ For $i = 1, \ldots, j-1$ Do
   $\quad\quad r_{ij} = \langle q_j \ , \ q_i \rangle$
   $\quad\quad q_j := q_j - r_{ij} q_i$
4. $\quad$ EndDo
5. $\quad r_{jj} = \|q_j\|_2$. If $r_{jj} = 0$ exit
6. $\quad q_j := q_j / r_{jj}$
7. EndDo

- Each $r_{ij} = \langle q_j \ , \ q_i \rangle, q_j := q_j - r_{ij} q_i$ step needs about $4m$ flops
- Need to do it approximately this many times

$$\sum_{j=1}^{n} \sum_{i=1}^{j-1} 1 \approx \int_1^n \int_1^{j-1} 1 di dj \approx \frac{n^2}{2}$$

- Approximate total flops for MGS (same for CGS)

$$4m \frac{n^2}{2} = 2mn^2$$

Gram-Schmidt as Triangular Orthogonalization

- Gram-Schmidt can be considered as multiplying with triangular matrices to make orthogonal columns. E.g., at 1st step:

$$\left[\begin{array}{c|c|c|c} a_1 & a_2 & \cdots & a_n \end{array}\right] \begin{bmatrix} \frac{1}{r_{11}} & \frac{-r_{12}}{r_{11}} & \frac{-r_{13}}{r_{11}} & \cdots \\ & 1 & & \\ & & 1 & \\ & & & \ddots \end{bmatrix} = \left[\begin{array}{c|c|c|c} q_1 & q_2^{(2)} & \cdots & q_n^{(2)} \end{array}\right]$$

- After n steps we get a product of triangular matrices

$$A \underbrace{R_1 R_2 \cdots R_n}_{R^{-1}} = Q$$

- "Triangular orthogonalization"

Householder Orthogonal Triangularization

- The Householder method multiplies by unitary matrices to make a triangular matrix. E.g., at 1st step:

$$Q_1 A = \begin{bmatrix} r_{11} & \mathbf{X} & \cdots & \mathbf{X} \\ 0 & \mathbf{X} & \cdots & \mathbf{X} \\ 0 & \mathbf{X} & \cdots & \mathbf{X} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \mathbf{X} & \cdots & \mathbf{X} \end{bmatrix}$$

- After all the steps we get a product of orthogonal matrices

$$\underbrace{Q_n \cdots Q_2 Q_1}_{Q^*} A = R$$

- "Orthogonal triangularization"

Introducing Zeros by Householder Reflectors

- Q_k introduces zeros below the diagonal in column k
- Preserves all the zeros previously introduced

$$\underset{A^{(0)} := A}{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}} \xrightarrow{Q_1} \underset{A^{(1)} := Q_1 A}{\begin{bmatrix} \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \end{bmatrix}} \xrightarrow{Q_2} \underset{A^{(2)} := Q_2 Q_1 A}{\begin{bmatrix} \times & \times & \times \\ & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} \\ & 0 & \mathbf{X} \\ & 0 & \mathbf{X} \end{bmatrix}} \xrightarrow{Q_3} \underset{A^{(3)} := Q_3 Q_2 Q_1 A}{\begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & & \mathbf{X} \\ & & 0 \\ & & 0 \end{bmatrix}}$$

Question: what shape is Q_k?

Introducing Zeros by Householder Reflectors

- Q_k introduces zeros below the diagonal in column k
- Preserves all the zeros previously introduced

$$\underset{A^{(0)} := A}{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}} \xrightarrow{Q_1} \underset{A^{(1)} := Q_1 A}{\begin{bmatrix} \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \end{bmatrix}} \xrightarrow{Q_2} \underset{A^{(2)} := Q_2 Q_1 A}{\begin{bmatrix} \times & \times & \times \\ & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} \\ & 0 & \mathbf{X} \\ & 0 & \mathbf{X} \end{bmatrix}} \xrightarrow{Q_3} \underset{A^{(3)} := Q_3 Q_2 Q_1 A}{\begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & & \mathbf{X} \\ & & 0 \\ & & 0 \end{bmatrix}}$$

Question: what shape is Q_k?

$$Q_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & H_k \end{bmatrix} \in \mathbb{C}^{m \times m}, \qquad H_k = I_{m-k+1} - 2 v_k v_k^*, \quad v_k \in \mathbb{C}^{m-k+1}.$$

Introducing Zeros by Householder Reflectors

- Q_k introduces zeros below the diagonal in column k
- Preserves all the zeros previously introduced

$$\underset{A^{(0)} := A}{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}} \xrightarrow{Q_1} \underset{A^{(1)} := Q_1 A}{\begin{bmatrix} \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \end{bmatrix}} \xrightarrow{Q_2} \underset{A^{(2)} := Q_2 Q_1 A}{\begin{bmatrix} \times & \times & \times \\ & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} \\ & 0 & \mathbf{X} \\ & 0 & \mathbf{X} \end{bmatrix}} \xrightarrow{Q_3} \underset{A^{(3)} := Q_3 Q_2 Q_1 A}{\begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & & \mathbf{X} \\ & & 0 \\ & & 0 \end{bmatrix}}$$

Question: what shape is Q_k?

$$Q_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & H_k \end{bmatrix} \in \mathbb{C}^{m \times m}, \qquad H_k = I_{m-k+1} - 2 v_k v_k^*, \quad v_k \in \mathbb{C}^{m-k+1}.$$

Question: what is v_k ?

Introducing Zeros by Householder Reflectors

- Q_k introduces zeros below the diagonal in column k
- Preserves all the zeros previously introduced

$$\underset{A^{(0)} := A}{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}} \xrightarrow{Q_1} \underset{A^{(1)} := Q_1 A}{\begin{bmatrix} \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \end{bmatrix}} \xrightarrow{Q_2} \underset{A^{(2)} := Q_2 Q_1 A}{\begin{bmatrix} \times & \times & \times \\ & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} \\ & 0 & \mathbf{X} \\ & 0 & \mathbf{X} \end{bmatrix}} \xrightarrow{Q_3} \underset{A^{(3)} := Q_3 Q_2 Q_1 A}{\begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & & \mathbf{X} \\ & & 0 \\ & & 0 \end{bmatrix}}$$

Question: what shape is Q_k?

$$Q_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & H_k \end{bmatrix} \in \mathbb{C}^{m\times m}, \qquad H_k = I_{m-k+1} - 2v_k v_k^*, \quad v_k \in \mathbb{C}^{m-k+1}.$$

Question: what is v_k ?

$$\tilde{v}_k = A^{(k-1)}(k:m,\, k), \quad v_k \leftarrow \alpha \|\tilde{v}_k\|_2 e_1 - \tilde{v}_k, \ (\alpha = ?), \qquad v_k \leftarrow \frac{v_k}{\|v_k\|_2}$$

The Householder Algorithm

- Choice of reflector: $v_k = \alpha \|\tilde{v}_k\|_2 e_1 - \tilde{v}_k$,
  To minimize cancellation error, choose $\alpha = -sign(\tilde{v}_k(1))$.
  Equivalently, $v_k = sign(\tilde{v}_k(1)) \|\tilde{v}_k\|_2 e_1 + \tilde{v}_k$.
- Compute the factor R of a QR factorization of $A \in \mathbb{C}^{m\times n}$, $(m \geq n)$
- Leave result in place of A, (i.e., overwrite A by R).
- Store reflection vectors v_k for later use.

Algorithm: (QR by Householder reflectors)

For $k = 1$ **to** n

$x = A_{k:m,k}$

$v_k = \mathrm{sign}(x(1))\|x\|_2 e_1 + x$

$v_k = v_k / \|v_k\|_2$

$A_{k:m,k:n} = A_{k:m,k:n} - 2v_k(v_k^* A_{k:m,k:n})$

Implicit application of Q

- The idea is that $Q_k w$ for any $w \in \mathbb{C}^m$ is only about $4(m-k+1)$ operation due to the special structure of Q_k
- Compute $Q^* b = Q_n \cdots Q_2 Q_1 b$ implicitly:

Algorithm: Implicit Calculation of $Q^* b$

For $k = 1$ **to** n

$$b_{k:m} = b_{k:m} - 2v_k(v_k^* b_{k:m})$$

- Compute $Qx = Q_1 Q_2 \cdots Q_n x$ implicitly:

Algorithm: Implicit Calculation of Qx

For $k = n$ **downto** 1

$$x_{k:m} = x_{k:m} - 2v_k(v_k^* x_{k:m})$$

- To create Q explicitly, apply the calculation of Qx to $x = I$

Flop counts of Householder QR

Algorithm: (QR by Householder reflectors)

For $k = 1$ **to** n

$x = A_{k:m,k}$

$v_k = \mathrm{sign}(x(1))\|x\|_2 e_1 + x$

$v_k = v_k / \|v_k\|_2$

$A_{k:m,k:n} = A_{k:m,k:n} - 2v_k(v_k^* A_{k:m,k:n})$

- Look at the highest order: Most work done by

$$A_{k:m,k:n} = A_{k:m,k:n} - 2v_k(v_k^* A_{k:m,k:n})$$

  - $2(m-k)(n-k)$ for the dot products $v_k^* A_{k:m,k:n}$
  - $(m-k)(n-k)$ for the outer product $2v_k(\cdots)$
  - $(m-k)(n-k)$ for the subtraction $A_{k:m,k:n} - \cdots$
- $4(m-k)(n-k)$ major work per iteration
- Including the outer loop, the total becomes

$$\sum_{k=1}^{n} 4(m-k)(n-k) = 4\sum_{k=1}^{n}(mn - k(m+n) + k^2)$$
$$\approx 4(mn^2 - (m+n)n^2/2 + n^3/3) = 2mn^2 - 2n^3/3$$

QR via Givens Rotations

- Recall Givens rotation $G(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ rotates $x \in \mathbb{R}^2$ anticlockwisely by θ
- To set an element to zero, choose $\cos\theta$ and $\sin\theta$ so that

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_i \\ x_j \end{bmatrix} = \begin{bmatrix} \sqrt{x_i^2 + x_j^2} \\ 0 \end{bmatrix}$$

  or

$$\cos\theta = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \qquad \sin\theta = \frac{-x_j}{\sqrt{x_i^2 + x_j^2}}$$

- "Orthogonal Triangularization"

QR via Givens Rotations

- Introduce zeros in column from bottom and up

$$\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{(3,4)} \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \end{bmatrix} \xrightarrow{(2,3)} \begin{bmatrix} \times & \times & \times \\ \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ & \times & \times \end{bmatrix} \xrightarrow{(1,2)}$$

$$\begin{bmatrix} \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} \\ & \times & \times \\ & \times & \times \end{bmatrix} \xrightarrow{(3,4)} \begin{bmatrix} \times & \times & \times \\ & \times & \times \\ & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} \end{bmatrix} \xrightarrow{(2,3)} \begin{bmatrix} \times & \times & \times \\ & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} \\ & & \times \end{bmatrix} \xrightarrow{(3,4)} R$$

- Flop count $3mn^2 - n^3$ (or 50% more than Householder QR)

Linear Least Squares Problems

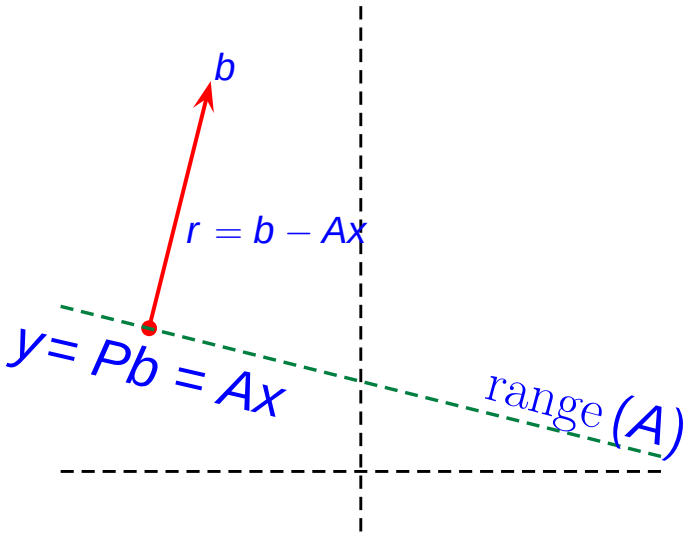
- In general, an over-determined system such as $Ax = b$, ($A \in \mathbb{C}^{m\times n}$, with $m > n$), has no solution
- A least square solution tries to minimize the 2-norm of the *residual* $r = b - Ax$:

Least Square problem:

Given $A \in \mathbb{C}^{m\times n}, m \geq n, b \in \mathbb{C}^m$, solve $\min_{x\in\mathbb{C}^n} \|Ax - b\|_2$.

- Geometric Interpretation

- For any $x \in \mathbb{C}^n$, $Ax \in \mathrm{range}(A)$
- Minimizing $\|b - Ax\|_2$ means finding the shortest distance from b to $\mathrm{range}(A)$
- Need $Ax = Pb$ where P is an orthogonal projector onto $\mathrm{range}(A)$, i.e., $r \perp \mathrm{range}(A)$



Solving Least Squares Problems

Essentially, we solve $Ax = Pb$, which always has a solution.

Different ways representing P leads to different methods.

- If $A = QR$, then $P = QQ^*$

$$Ax = Pb \implies QRx = QQ^*b \implies Rx = Q^*b$$

- If $A = U\Sigma V^*$, than $P = UU^*$

$$Ax = Pb \implies U\Sigma V^*x = UU^*b \implies \Sigma V^*x = U^*b$$

  (Most stable but also most expensive among the three)

- If A is full rank, then $P = A(A^*A)^{-1}A^*$ (note $PP^* = P^*P, P^2 = P$)

$$Ax = Pb \implies Ax = A(A^*A)^{-1}A^*b \implies A^*Ax = A^*b.$$

  This is called the *normal equations*. (Least expensive, but also least accurate among the three if A has close to linearly dependent columns.)

Solving LS: via thin QR decomposition

- Using thin QR: $A = QR$, $Q \in \mathbb{C}^{m\times n}, R \in \mathbb{C}^{n\times n}$.
  Project b onto $\mathrm{range}(A)$ as $Pb = QQ^*b$
- Insert into $Ax = Pb$ to get $QRx = QQ^*b$, or $Rx = Q^*b$

Algorithm: Least Squares via QR Factorization

1. Compute the thin QR factorization $A = QR$
2. Compute the vector Q^*b (without forming Q)
3. Solve the upper-triangular system $Rx = Q^*b$ for x

- Major cost: thin QR Factorization $\sim 2mn^2 - 2n^3/3$ flops
- Good stability, relatively fast. (Used in MATLAB's "backslash" \)

Solving LS: via SVD

- Compute SVD of A: $A = U\Sigma V^*$, $Q \in \mathbb{C}^{m\times n}, \Sigma \in \mathbb{R}^{n\times n}, V \in \mathbb{C}^{n\times n}$. Project b onto $\mathrm{range}(A)$ as $Pb = UU^*b$
- Insert into $Ax = Pb$ to get $U\Sigma V^*x = UU^*b$, or $\Sigma V^*x = U^*b$

Algorithm: Least Squares via SVD

1. Compute the reduced SVD $A = U\Sigma V^*$
2. Compute the vector U^*b
3. Solve the diagonal system $\Sigma w = U^*b$ for w
4. Set $x = Vw$

- Major cost: SVD of A $\sim 2mn^2 + 11n^3$ flops
- Very good stability properties, use if A is close to rank-deficient

Solving LS: via Normal Equations

- If A has full rank, A^*A is square, (hermitian) positive definite
- Solve by *Cholesky factorization* (Gaussian elimination)

Algorithm: Least Squares via Normal Equations

1. Form the matrix A^*A and the vector A^*b
2. Compute the Cholesky factorization $A^*A = R^*R$
3. Solve the lower-triangular system $R^*w = A^*b$ for w
4. Solve the upper-triangular system $Rx = w$ for x

- Major cost: Forming A^*A and Cholesky $\sim mn^2 + n^3/3$ flops
- Fast, but sensitive to rounding errors (particularly so when A is close to rank deficient)

LS by normal equations

In fact, the normal equation $A^*Ax = A^*b$ is a necessary condition for $\min_{x\in\mathbb{C}^n} \|Ax - b\|_2$ (no need to assume A full rank).

The is readily seen from the geometric view:

$$r \perp \mathrm{range}\,(A) \quad \Longrightarrow \quad A^*r = A^*(Ax - b) = 0 \quad \Longrightarrow \quad A^*Ax = A^*b.$$

It can also be obtained by expanding $\min_{x\in\mathbb{C}^n} \|Ax - b\|_2^2$ as

$$f(x) = x^*A^*Ax - b^*Ax - x^*A^*b + b^*b,$$

then set the first order derivative of $f(x)$ w.r.t. x to 0. This also leads to the normal equation $A^*Ax = A^*b$.

On solving linear equations $Ax = b$, $A \in \mathbb{C}^{m \times n}$ (with $m = n$)

- Gaussian Elimination via LU and pivoted LU
- Cholesky decomposition for A SPD or (H)PD
- Conditioning and stability

The LU Factorization

- Compute $A = LU$, where $L, U \in \mathbb{C}^{m\times m}$, L is unit lower triangular, U is upper triangular
- Obtain U by sequentially subtracting multiples of rows:
  Left multiply by elementary matrices, each L_i introduces zeros below diagonal of column i.

$$\underset{A}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}} \overset{L_1}{\to} \underset{L_1A}{\begin{bmatrix} \times & \times & \times & \times \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}} \overset{L_2}{\to} \underset{L_2L_1A}{\begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & 0 & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} & \mathbf{X} \end{bmatrix}} \overset{L_3}{\to} \underset{L_3L_2L_1A}{\begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & & \times & \times \\ & & 0 & \mathbf{X} \end{bmatrix}}$$

$$\underbrace{L_{m-1}\cdots L_2L_1}_{L^{-1}}A = U \quad \Longrightarrow \quad A = LU \quad \text{where } L = L_1^{-1}L_2^{-1}\cdots L_{m-1}^{-1}$$

- "Triangular triangularization"

The Matrices L_k

- At step k, eliminate elements below A_{kk}:

$$x_k = \begin{bmatrix} x_{1k} & \cdots & x_{kk} & x_{k+1,k} & \cdots & x_{mk} \end{bmatrix}^*$$
$$L_k x_k = \begin{bmatrix} x_{1k} & \cdots & x_{kk} & 0 & \cdots\cdots\cdots & 0 \end{bmatrix}^*$$

- The *multipliers* $\ell_{jk} = \frac{x_{jk}}{x_{kk}}$ appear in L_k:

$$L_k = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -\ell_{k+1,k} & 1 & & \\ & & \vdots & & \ddots & \\ & & -\ell_{mk} & & & 1 \end{bmatrix} = \prod_{j=k+1}^{m} E_a(k,j,-\ell_{jk})$$

Recall: $E_a(k,j,c) = I + ce_j e_k^{\mathsf{T}}, \quad E_a^{-1}(k,j,c) = E_a(k,j,-c)$

Structure of L in $A = LU$

- Each L_k is an elementary matrix:
  Let $\ell_k = [0, \cdots, 0, \ell_{k+1,k}, \cdots, \ell_{m,n}]^*$, then $L_k = I - \ell_k e_k^*$.
  By Sherman-Morison, (or direct verification)

$$L_k^{-1} = I - \frac{\ell_k e_k^*}{e_k^* \ell_k - 1} = I + \ell_k e_k^*$$

- $L_k^{-1} L_{k+1}^{-1} = (I + \ell_k e_k^*)(I + \ell_{k+1} e_{k+1}^*) = I + \ell_k e_k^* + \ell_{k+1} e_{k+1}^*$
- The product $L = L_1^{-1} L_2^{-1} \cdots L_{m-1}^{-1}$ is obtained by inserting ℓ_k into the k-th column of I

$$L = L_1^{-1} L_2^{-1} \cdots L_{m-1}^{-1} = \begin{bmatrix} 1 & & & & \\ \ell_{21} & 1 & & & \\ \ell_{31} & \ell_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{m1} & \ell_{m2} & \cdots & \ell_{m,m-1} & 1 \end{bmatrix}$$

Gaussian Elimination (without pivoting)

Algorithm: Factorize $A \in \mathbb{C}^{m\times m}$ into $A = LU$, (no pivoting)

$L = I$, $U = A$ (can overwrite A by L and U to avoid using L, U)

For $k = 1$ **to** $m - 1$

  for $j = k + 1$ **to** m

    $\ell_{jk} = u_{jk}/u_{kk}$

    $u_{j,k:m} = u_{j,k:m} - \ell_{jk} u_{k,k:m}$

- Inner loop can use matrix operations, e.g., (overwrite A)

```
for k = 1 : m-1
    if (A(k,k) == 0),
       error(' without pivoting, LU decomposition fails ')
    else
       A(k+1:m,k) = A(k+1:m,k)/A(k,k);
       A(k+1:m,k+1:m) = A(k+1:m,k+1:m)-A(k+1:m,k)*A(k,k+1:m);
    end
end
```

- Operation count $\sim \sum_{k=1}^{m} 2(m-k)(m-k) \sim 2\sum_{k=1}^{m} k^2 \sim 2m^3/3$

Pivoting

- At step k of no pivoting LU, the (k,k) element is used (as pivot) to introduce zeros in k-column below the (k,k) element

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ & \mathbf{x}_{kk} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \times & \times & \times \\ & x_{kk} & \times & \times & \times \\ & 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}$$

- But any nonzero element $i \geq k$ in column k can be used as pivot:

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \mathbf{x}_{ik} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \times & \times & \times & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \times & \times & \times \\ & 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & x_{ik} & \times & \times & \times \\ & 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}$$

Pivoting

- Also, any nonzero row element $j \geq k$ can be used as pivot:

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \mathbf{X} & \mathbf{x}_{ij} & \mathbf{X} & \mathbf{X} \\ & \times & \times & \times & \times \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \times & \times & \times \\ & \mathbf{X} & 0 & \mathbf{X} & \mathbf{X} \\ & \mathbf{X} & 0 & \mathbf{X} & \mathbf{X} \\ & \times & x_{ij} & \times & \times \\ & \mathbf{X} & 0 & \mathbf{X} & \mathbf{X} \end{bmatrix}$$

- Choose different pivots to avoid zero or very small pivots (reduce instability) !
- *Pivoting* means first exchanging rows (or columns) s.t. the diagonal pivot has larger magnitude, then applying the standard (no-pivot) LU
- A computer code might account for the pivoting indirectly instead of actually moving the data

Partial (row) Pivoting

- Full pivoting searches among all valid pivots, i.e., at k-th step, choose $\max_{i\geq k, j\geq k} |a_{ij}|$ as pivot, (interchange rows and columns), expensive
- *Partial pivoting* considers a pivot in column k only, i.e., choose $\max_{i\geq k} |a_{ik}|$ as pivot, (interchange rows)

$$\underset{\text{Pivot selection}}{\begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \mathbf{x}_{ik} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \times & \times & \times & \times \end{bmatrix}} \xrightarrow{P_1} \underset{\text{Row interchange}}{\begin{bmatrix} \times & \times & \times & \times & \times \\ & \mathbf{x}_{ik} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \times & \times & \times & \times \\ & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \times & \times & \times & \times \end{bmatrix}} \xrightarrow{L_1} \underset{\text{Elimination}}{\begin{bmatrix} \times & \times & \times & \times & \times \\ & x_{ik} & \times & \times & \times \\ & 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}}$$

- In terms of matrices:

$$L_{m-1}P_{m-1}\cdots L_2P_2L_1P_1A = U,$$

where P_i's are the elementary matrices, each used to switch two rows when a pivot is necessary.

The $PA = LU$ Factorization

- To combine all L_k and all P_k into matrices, rewrite as

$$L_{m-1}P_{m-1}\cdots L_2P_2L_1P_1A = U$$
$$(L'_{m-1}\cdots L'_2L'_1)(P_{m-1}\cdots P_2P_1)A = U$$

where

$$L'_k = P_{m-1}\cdots P_{k+1}L_kP_{k+1}^{-1}\cdots P_{m-1}^{-1}$$

- This gives the LU factorization of A

$$PA = LU$$

Gaussian Elimination with Partial Pivoting

Algorithm: Gaussian Elimination for $PA = LU$

$U = A, L = I, P = I$
for $k = 1$ **to** $m - 1$
    Select $i \geq k$ to maximize $|u_{ik}|$
    $u_{k,k:m} \leftrightarrow u_{i,k:m}$ (interchange two rows)
    $\ell_{k,1:k-1} \leftrightarrow \ell_{i,1:k-1}$
    $p_{k,:} \leftrightarrow p_{i,:}$
    for $j = k + 1$ **to** m
        $\ell_{jk} = u_{jk}/u_{kk}$
        $u_{j,k:m} = u_{j,k:m} - \ell_{jk} u_{k,k:m}$

- Can overwrite A by L and U (saves the memory for storing L, U)
- When used to solve $Ax = b$, no need to store P either, can apply P directly to b and solve $PAx = Pb \Longrightarrow LUx = Pb$.
- Flops: similar to no pivoting, $\sim 2m^3/3$.

Gaussian Elimination with Partial Pivoting

Matlab code using PPGE to solve $Ax = b$. Overwrite A by L and U, P is not stored but directly applied to A and b.

```
for j = 1 : n-1
  % choose the one with largest magnitude from A(j:n,j) as pivot
  [amax, ip] = max( abs(A(j:n,j)) );
  % ip from above is in [1:n-j+1], point it to true row number in A
  ip = ip + j-1;
  if ( ip ~= j ),
     % apply Pj to both A and b, this is nothing but row swamping
     tmp=A(ip, j:n); A(ip,j:n)=A(j,j:n); A(j,j:n)=tmp;
     tmp = b(ip);  b(ip) = b(j);  b(j) = tmp;
  end
  if (A(j,j)~=0),
     % apply the standard gauss elimination
     A(j+1:n,j) = A(j+1:n,j)/A(j,j);
     A(j+1:n,j+1:n) = A(j+1:n,j+1:n) - A(j+1:n,j)*A(j,j+1:n);
     b(j+1:n) = b(j+1:n) - A(j+1:n,j)*b(j);
  else
     error(' singular matrix ');
  end
end
x = triu(A)\b;
```

Full Pivoting

- If pivots are selected from a different column, permutation matrices Q_k for the columns are required:

$$L_{m-1}P_{m-1}\cdots L_2P_2L_1P_1AQ_1Q_2\cdots Q_{m-1} = U$$
$$(L'_{m-1}\cdots L'_2L'_1)(P_{m-1}\cdots P_2P_1)A(Q_1Q_2\cdots Q_{m-1}) = U$$

- Set

$$L = (L'_{m-1}\cdots L'_2L'_1)^{-1}$$
$$P = P_{m-1}\cdots P_2P_1$$
$$Q = Q_1Q_2\cdots Q_{m-1}$$

to obtain

$$PAQ = LU$$

Cholesky Factorization

- Compute with R upper triangular;
  or $A = LDL^*$ for L unit lower triangular
- Need A to be symmetric/hermitian;
  need positive definiteness[1] of A for $A = R^*R$
- Utilize symmetry, complexity is $\sim m^3/3$
  (reduced by half that of general LU)
- Some applications: Solve $Ax = b$ when A is SPD, such as in the Hessian matrices (quasi-Newton methods for nonlinear optimization), and covariance matrices (Monte Carlo simulation, and Kalman filters)

[1] For $A \in \mathbb{C}^{n\times n}$, A is PD if $x^*Ax > 0, \forall x \in \mathbb{C}^n \neq 0$; this condition implicitly guarantees $A^* = A$. While for $A \in \mathbb{R}^{n\times n}$, A is PD if $x^\top Ax > 0, \forall x \in \mathbb{R}^n \neq 0$; but this does not guarantee $A^\top = A$, hence one needs A to be SPD to guarantee existence of $A = R^\top R$.

Computing Cholesky Factorization $A = R^*R$

Let $\alpha = \sqrt{a_{11}}$. The first step for $A = R^*R$ is

$$A := \begin{bmatrix} a_{11} & w^* \\ w & A^{(1)} \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ w/\alpha & I \end{bmatrix} \begin{bmatrix} \alpha & w^*/\alpha \\ 0 & A^{(1)} - ww^*/a_{11} \end{bmatrix}$$

$$= \begin{bmatrix} \alpha & 0 \\ w/\alpha & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & A^{(1)} - ww^*/a_{11} \end{bmatrix} \begin{bmatrix} \alpha & w^*/\alpha \\ 0 & I \end{bmatrix} =: R_1^* A_1 R_1$$

That is, $R_{(1,1)} = \sqrt{A_{(1,1)}}$, $\quad R_{(1,2:n)} = A^*_{(2:n,1)}/R_{(1,1)}$.

Can apply the same to $\boxed{A^{(2)} := A^{(1)} - ww^*/a_{11}}$ (also PD, why?)

$$A = R_1^* \begin{bmatrix} 1 & 0 \\ 0 & \tilde{R}_2^* \tilde{A}_2 \tilde{R}_2 \end{bmatrix} R_1 = R_1^* R_2^* A_2 R_2 R_1, \quad R_2 = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{R}_2 \end{bmatrix}, A_2 = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{A}_2 \end{bmatrix}$$

Note $R_{(2,2)} = \sqrt{A^{(2)}_{(1,1)}}$, $\quad R_{(2,2:n)} = {A^{(2)}_{(2:n,1)}}^*/R_{(2,2)}$.

Apply the same recursively to diagonal block of $A^{(k)}$

Computing $A = R^*R$ (A is PD, two versions)

```
R = A;
for k = 1 : n
    for j = k+1 : n  % only update upper triangular part (symmetry)
        R(j,j:n) = R(j,j:n) - R(k,j:n)*R(k,j)'/R(k,k);
    end
    if ( R(k,k) <= 0 ),
        error('A is not HPD, try ''A=R^*DR'' instead'),
    end
    R(k,k:n) = R(k,k:n)/sqrt(R(k,k));
end
R = triu(R);
```

```
R = zeros(n,n);
for i = 1 : n,
   tmp = A(i,i) - R(1:i-1,i)'*R(1:i-1,i);
   if ( tmp <= 0 ),
      error('A is not HPD, try ''A=R^*DR'' instead'),
   end
   R(i,i) = sqrt(tmp);
   for  j = i+1 : n
       R(i,j) = (A(i,j) - R(1:i-1,i)'*R(1:i-1,j))/R(i,i);
   end
end
```

Computing $A = R^*DR$ (two of several versions)

```
R = eye(n);       % the returned R is unit upper triangular
for j = 1 : n-1,
    dv(j)=real(A(j,j));
    R(j,j+1:n) = A(j,j+1:n)/dv(j);
    for i = j+1 : n  % only update upper triangular row elements
        A(i,i:n) = A(i,i:n) - R(j,i)'*dv(j)*R(j,i:n);
    end
end
dv(n) = A(n,n);  % D=diag(dv(1:n))
```

```
R = eye(n);
for j = 1 : n-1,
    dv(j)=real(A(j,j));
    for i = j+1:n
        R(j,i) = A(j,i)/dv(j);
        for k = j+1 : i  %only update lower triangular column elements
            A(k,i) = A(k,i) - R(j,i)*dv(j)*R(j,k)' ;
        end
    end
end
dv(n) = A(n,n);
```

Solving nonsingular triangular systems

Solving $Ux = b$: (backward substitution)

$$\sum_{k=i}^{n} u_{ik} x_k = b_i, \quad i = 1, \cdots, n$$

$$\Longrightarrow \quad x_i = \frac{b_i - \sum_{k=i+1}^{n} u_{ik} x_k}{u_{ii}}, \quad i = n, \cdots, 1$$

Solving $Lx = b$: (forward substitution)

$$\sum_{k=1}^{i} l_{ik} x_k = b_i, \quad i = 1, \cdots, n$$

$$\Longrightarrow \quad x_i = \frac{b_i - \sum_{k=1}^{i-1} l_{ik} x_k}{l_{ii}}, \quad i = 1, \cdots, n$$

Complexity for triangular solves: $\sim O(n^2)$

On Conditioning and Stabilities

- General definition of Condition Numbers
- Accuracy of (numerical) solutions
- Stability
  - Forward stability
  - Backward stability
  - Mixed stability
- Main picture: Accuracy depend on two things
  1. Conditioning of the underlying problem
  2. Stability of the algorithm used to solve the problem

Conditioning, Condition number

- *Absolute Condition Number* of a function $f : X \to Y$ at x:

$$\hat{\kappa}(f,x) = \sup_{\delta x \neq 0} \frac{\|f(x+\delta x) - f(x)\|}{\|\delta x\|}$$

- If f is differentiable,

$$\hat{\kappa}(f,x) = \|J_f(x)\|$$

  where the Jacobian $(J_f)_{ij} = \partial f_i / \partial x_j$, and the matrix norm is induced by the norms on X and Y.

- *Relative Condition Number*

$$\kappa(f,x) = \frac{\hat{\kappa}}{\|f(x)\| \,/\, \|x\|} = \sup_{\delta x \neq 0} \left(\frac{\|f(x+\delta x) - f(x)\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right)$$

- If f is differentiable,

$$\kappa(f,x) = \frac{\|J_f(x)\|}{\|f(x)\| / \|x\|}$$

Conditioning, Condition number

- **Example**: The function $f(x) = \alpha x$
  - Absolute condition number $\hat{\kappa} = \|J_f\| = \alpha$
  - Relative condition number $\kappa = \frac{\|J_f\|}{\|f(x)\|/\|x\|} = \frac{\alpha}{\alpha x/x} = 1$
- **Example**: The function $f(x) = \sqrt{x}$
  - Absolute condition number $\hat{\kappa} = \|J_f\| = \frac{1}{2\sqrt{x}}$
  - Relative condition number $\kappa = \frac{\|J_f\|}{\|f(x)\|/\|x\|} = \frac{1/(2\sqrt{x})}{\sqrt{x}/x} = \frac{1}{2}$
- **Example**: The function $f(x) = x_1 - x_2$ (with ∞-norms)
  - Absolute condition number $\hat{\kappa} = \|J_f\| = \|(1, -1)\| = 2$
  - Relative condition number

$$\kappa = \frac{\|J_f\|}{\|f(x)\|/\|x\|} = \frac{2}{|x_1 - x_2|/\max\{|x_1|, |x_2|\}}$$

  - Ill-conditioned (in the relative sense) when $x_1 \approx x_2$
    (This is the well-known cancellation problem when subtracting two close numbers)

Conditioning, Condition number

- Consider $f(x) = Ax$, with $A \in \mathbb{C}^{m\times n}$

$$\kappa = \frac{\|J_f\|}{\|f(x)\|/\|x\|} = \|A\|\frac{\|x\|}{\|Ax\|}$$

- For A square and nonsingular, use $\|x\|/\|Ax\| \le \|A^{-1}\|$:

$$\kappa \le \|A\|\|A^{-1}\|$$

  (equality achieved for the last right singular vector $x = v_n$)
- $\kappa = \|A\|\|A^{-1}\|$ is also the condition number for $f(b) = A^{-1}b$ (solution of linear system)
- *Condition number of matrix A*:

$$\kappa(A) := \|A\|_2\|A^{-1}\|_2 = \frac{\sigma_1}{\sigma_n}$$

Condition of System of Equations

- For fixed b, consider $f(A) = A^{-1}b$
- Perturb A by δA and find perturbation δx:

$$(A + \delta A)(x + \delta x) = b$$

- Use $Ax = b$ and assume $(\delta A)(\delta x) \approx 0$:

$$(\delta A)x + A(\delta x) = 0 \quad \Longrightarrow \quad \delta x = -A^{-1}(\delta A)x$$

- Condition number of problem f:

$$\kappa = \frac{\|\delta x\|}{\|x\|} \Big/ \frac{\|\delta A\|}{\|A\|} \leq \frac{\|A^{-1}\|\|\delta A\|\|x\|}{\|x\|} \Big/ \frac{\|\delta A\|}{\|A\|} = \|A^{-1}\|\|A\| = \kappa(A)$$

$O(\epsilon_{\text{machine}})$ notation

- The notation $\varphi(t) = O(\psi(t))$ means there is a constant C such that, for t close to a limit (often 0 or ∞), $|\varphi(t)| \le C\psi(t)$
- **Example**: $\sin^2 t = O(t^2)$ as $t \to 0$ means $|\sin^2 t| \le Ct^2$ for some C
- If φ depends on additional variables, the notation

$$\varphi(s,t) = O(\psi(t)) \qquad \text{uniformly in } s$$

  means there is a constant C such that $|\varphi(s,t)| \le C\psi(t)$ for any s
- **Example**: $(\sin^2 t)(\sin^2 s) = O(t^2)$ uniformly as $t \to 0$, but not if $\sin^2 s$ is replaced by s^2
- In bounds such as $\|\tilde{x} - x\| \le C\kappa(A)\epsilon_{\text{machine}}\|x\|$, C does not depend on A or b, but it might depend on the dimension m

Accuracy of an algorithm

- For a problem described as $f : X \to Y$, which is assumed differentiable,
  Apply (discrete) approximation and solve by an algorithm, described as $\tilde{f} : X \to Y$. ($\tilde{f}(x)$ is the computed value of $f(x)$)
- $\tilde{f}(x)$ has *absolute error* $\|\tilde{f}(x) - f(x)\|$ and *relative error*

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|}$$

- Algorithm is *accurate* if (for all $x \in X$)

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = O(\epsilon_{\text{machine}})$$

  where $O(\epsilon_{\text{machine}})$ is "on the order of $\epsilon_{\text{machine}}$"
- Constant in $O(\epsilon_{\text{machine}})$ is likely to be large in many problems (rounding error exists for x)
- More realistic to compare $\tilde{f}(x)$ with $f(\tilde{x})$, where $\tilde{x}$ is an approximation of the exact x

Stability of an algorithm

- An algorithm $\tilde{f} : X \to Y$ for a problem $f : X \to Y$ is stable if (for all $x \in X$)

$$\frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = O(\epsilon_{\text{machine}})$$

for some $\tilde{x}$ with

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon_{\text{machine}})$$

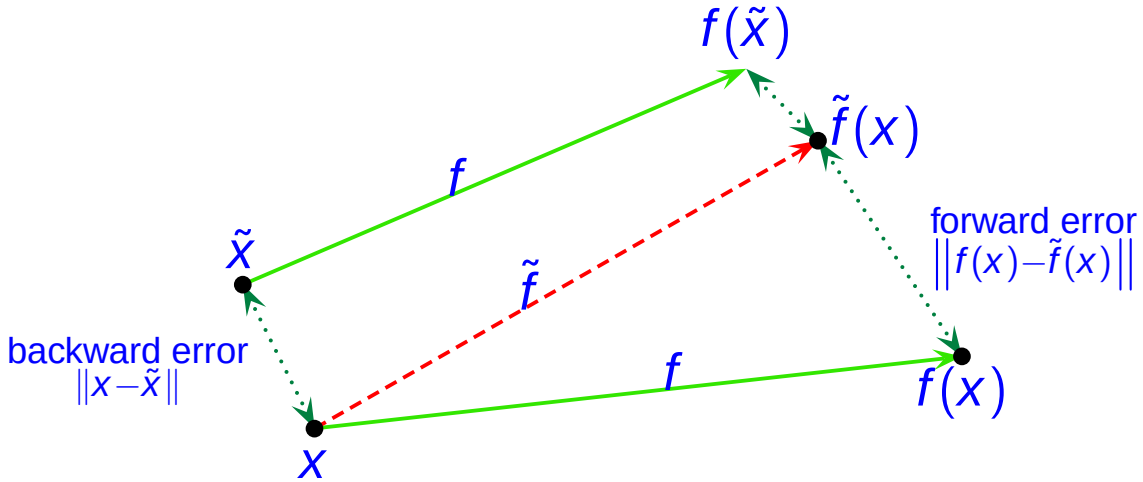
"Nearly the right answer to nearly right data/problem"

- An algorithm $\tilde{f}$ for a problem f is backward stable if (for all $x \in X$)

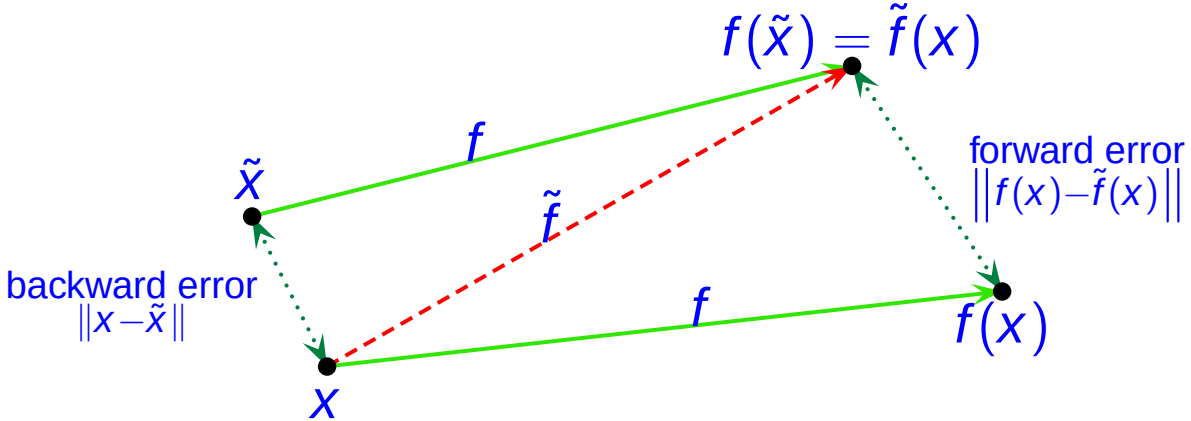
$$\tilde{f}(x) = f(\tilde{x}) \quad \text{for some } \tilde{x} \text{ with} \quad \frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon_{\text{machine}})$$

"Exactly the right answer to nearly the right data/problem"

Stability, Backward Stability



$\tilde{f}$ is stable (in the mixed forward-backward sense): Nearly right solution to a nearly right problem.



$\tilde{f}$ is backward stable: Exactly right solution to a nearly right problem.

Linking forward error with backward error

Assume that forward error, backward error, and condition number are defined mutually consistently, then a rule of thumb in error analysis is

$$\boxed{\text{(forward error)} \leq \text{C} * \text{ (condition number) } * \text{ (backward error)}}$$

That is,

$$\left\| f(x) - \tilde{f}(x) \right\| \leq C\hat{\kappa}(f, x) \left\| x - \tilde{x} \right\|,$$

which may be considered as an approximation of the 1st order Taylor expansion.

If f is backward stable, then by the definition of $\hat{\kappa}(f, x)$ we see the constant C can be set to 1.

- Idea of backward error analysis: Backward error reveals the stability of the algorithm, isolated from the conditioning of the underlying problem. (While forward error depends on both stability of algorithm and conditioning of problem.)

Three types of stability

- Small $\frac{\text{forward error}}{\text{condition number}}$, i.e., $\left(\frac{\|\tilde{f}(x)-f(x)\|}{\kappa(f,x)}\right)$
  $\Longrightarrow$ forward stable algorithm
- Small mixed error $\left(\left\|\tilde{f}(x) - f(\tilde{x})\right\|\right)$
  $\Longrightarrow$ stable algorithm (in mixed forward-backward sense)
- Small backward error $(\|\tilde{x} - x\|)$
  $\Longrightarrow$ backward stable algorithm

Backward stability is the strongest among the three:

Backward stable $\Longrightarrow$ stable

Backward stable $\Longrightarrow$ forward stable

Comment: However, the above definition for forward stability is not universally accepted. It is also possible to require small "forward error" for forward stability. In this case "backward stability" does not imply "forward stability". An example is the QR factorization by GS, which may be considered "forward unstable" (Q factor may not be orthogonal), though it is backward stable.

Accuracy of a Backward Stable Algorithm

Theorem: If a backward stable algorithm is used to solve a problem f with condition number κ, then the relative errors satisfy

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = O(\kappa(f,x)\epsilon_{\text{machine}}) \ .$$

Proof. The definition of condition number gives

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

where $o(1) \to 0$ as $\epsilon_{\text{machine}} \to 0$.

Backward stability of $\tilde{f}$ means $\tilde{f}(x) = f(\tilde{x})$ for $\tilde{x}$ such that

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\epsilon_{\text{machine}})$$

Combining these gives the desired result. ■

Backward Stability of Householder QR

- For a QR factorization $A = QR$ computed by Householder triangularization, the factors $\tilde{Q}$ and $\tilde{R}$ satisfy

$$\tilde{Q}\tilde{R} = A + \delta A, \qquad \frac{\|\delta A\|}{\|A\|} = O(\epsilon_{\text{machine}})$$

- Exactly the right QR factorization of a slightly perturbed A
- Here $\tilde{R}$ is the R computed by the algorithm using floating points
- However, $\tilde{Q}$ is a product of *exactly unitary* reflectors:

$$\tilde{Q} = \tilde{Q}_1 \tilde{Q}_2 \cdots \tilde{Q}_n$$

where $\tilde{Q}_k$ is implicitly given by the computed $\tilde{v}_k$ (since Q is generally not formed explicitly)

Backward Stability of Solving $Ax = b$ with QR

Algorithm: Solving $Ax = b$ by QR Factorization

1. $QR = A$ using Householder, represent Q by reflectors
2. $y = Q^* b$ implicitly using reflectors
3. $x = R^{-1} y$ by back substitution

- Step 1 is backward stable (from previous slide)
- Step 2 can be shown to be backward stable:

$$(\tilde{Q} + \delta Q)\tilde{y} = b, \qquad \|\delta Q\| = O(\epsilon_{\text{machine}})$$

- Step 3 is backward stable (will be shown later):

$$(\tilde{R} + \delta R)\tilde{x} = \tilde{y}, \qquad \frac{\|\delta R\|}{\|\tilde{R}\|} = O(\epsilon_{\text{machine}})$$

Backward Stability of Solving $Ax = b$ with QR

- Put the three steps together to show backward stability of the algorithm:

$$(A + \Delta A)\tilde{x} = b, \qquad \frac{\|\Delta A\|}{\|A\|} = O(\epsilon_{\text{machine}})$$

- *Proof*. Steps 2 and 3 give

$$b = (\tilde{Q} + \delta Q)(\tilde{R} + \delta R)\tilde{x} = \left[\tilde{Q}\tilde{R} + (\delta Q)\tilde{R} + \tilde{Q}(\delta R) + (\delta Q)(\delta R)\right]\tilde{x}$$

Step 1 (backward stability of Householder) gives

$$\begin{aligned} b &= \left[A + \delta A + (\delta Q)\tilde{R} + \tilde{Q}(\delta R) + (\delta Q)(\delta R)\right]\tilde{x} \\ &= (A + \Delta A)\tilde{x} \end{aligned}$$

Backward Stability of Solving $Ax = b$ with QR

δA is small compared to A, therefore

$$\frac{\|\tilde{R}\|}{\|A\|} \leq \|\tilde{Q}^*\| \frac{\|A + \delta A\|}{\|A\|} = O(1)$$

Now show that each term in ΔA is small:

$$\frac{\|(\delta Q)\tilde{R}\|}{\|A\|} \leq \|\delta Q\| \frac{\|\tilde{R}\|}{\|A\|} = O(\epsilon_{\text{machine}})$$

$$\frac{\|\tilde{Q}(\delta R)\|}{\|A\|} \leq \|\tilde{Q}\| \frac{\|\delta R\|}{\|\tilde{R}\|} \frac{\|\tilde{R}\|}{\|A\|} = O(\epsilon_{\text{machine}})$$

$$\frac{\|(\delta Q)(\delta R)\|}{\|A\|} \leq \|\delta Q\| \frac{\|\delta R\|}{\|A\|} = O(\epsilon_{\text{machine}}^2)$$

Backward Stability of Solving $Ax = b$ with QR

Add the terms to show that ΔA is small:

$$\begin{aligned}\frac{\|\Delta A\|}{\|A\|} &\le \frac{\|\delta A\|}{\|A\|} + \frac{\|(\delta Q)\tilde{R}\|}{\|A\|} + \frac{\|\tilde{Q}(\delta R)\|}{\|A\|} + \frac{\|(\delta Q)(\delta R)\|}{\|A\|} \\ &= O(\epsilon_{\text{machine}})\end{aligned}$$

- Since the algorithm is backward stable, it is also accurate:

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\kappa(A)\epsilon_{\text{machine}})$$

On Floating Points

- Representation
  - Precision (or size of Significand, or significant digits):
    - an integer $p \geq 1$
  - Exponent size:
    - two bounds $e_{\min}$ and $e_{\max}$, with an integer $e \in [e_{\min}, e_{\max}]$
  - Base (or Radix):
    - an integer $\beta \geq 2$
      $\beta = 2$ — binary format (most common in computers)
      $\beta = 10$ — decimal format
      $\beta = 16$ — hexadecimal
- IEEE single and double precision floating point data type
- Floating point arithmetic

Floating Point Representations

A floating point (number) system is a subset of the real numbers $\mathbb{R}$, with elements represented by

$$\boxed{\pm m\beta^{e-p} = \pm\frac{m}{\beta^p}\beta^e}$$

- The β is the base (also called *radix*)
- The p is the precision
- The e is the exponent — an integer bounded by $[e_{\min}, e_{\max}]$
- The m is the significand — an integer satisfying $0 \le m \le \beta^p - 1$

An equivalent form of the floating point (number) is

$$\pm 0.d_1 d_2 \cdots d_p \times \beta^e = \pm\left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_p}{\beta^p}\right)\beta^e,$$

$0 \le d_i \le \beta - 1$, and $d_1 \ne 0$ for *normalized* numbers.

Floating Point Representations (continued)

- Two advantages of normalized representation:
  - Uniqueness of representation
  - For $\beta = 2$, $d_1 \equiv 1$, which does not need to be stored (saved one extra bit for a longer significand (also called *mantissa*))
- For *normalized* floating points: To represent 0, use $e = e_{\min} - 1$.
- For nonzero normalized floating points, $\boxed{\beta^{p-1} \le m \le \beta^p - 1}$ (uniqueness of representation)
- Range of nonzero normalized floating points (symmetric w.r.t. 0)

$$\boxed{\beta^{e_{\min}-1} \le |\mathrm{fl}(y)| \le \beta^{e_{\max}}(1 - \beta^{-p})}$$

  - Minimum when $d_1 = 1, d_i = 0\ (i > 1), e = e_{\min}$, i.e., $\frac{1}{\beta}\beta^{e_{\min}}$.
  - Maximum when $d_i = \beta - 1\ (i \ge 1), e = e_{\max}$, i.e., $\left(\sum_{i=1}^{p} \frac{\beta - 1}{\beta^i}\right)\beta^{e_{\max}} = \beta^{e_{\max}}(1 - \beta^{-p})$.
  - Or, by using $m\beta^{e-p}$: $\beta^{p-1}\beta^{e_{\min}-p} \le |\mathrm{fl}(y)| \le (\beta^p - 1)\beta^{e_{\max}-p}$.

Machine epsilon and unit roundoff

- Machine epsilon ($\epsilon_{\text{machine}}$), is sometimes called *unit roundoff* (μ), (while some authors uses $\mu = \epsilon_{\text{machine}}/2$ for a good reason)
  - The IEEE standard does not define the terms "machine epsilon" and unit roundoff
  - $\epsilon_{\text{machine}}$ provides an upper bound on the relative error due to rounding. That is, for any non-zero real number y within the normalized range of a floating point system,

$$\left| \frac{\mathrm{fl}(y) - y}{y} \right| \leq \epsilon_{\text{machine}}$$

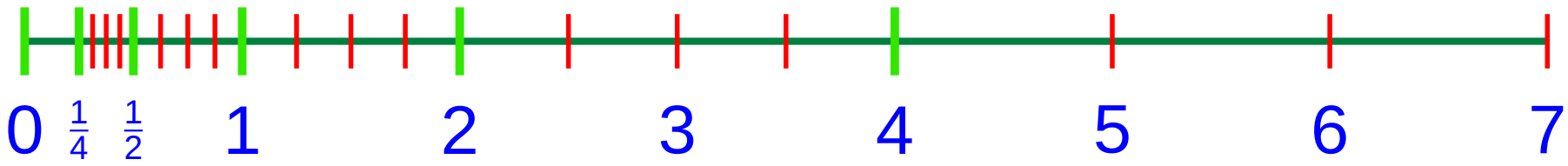
  - A few (essentially) equivalent (but slightly different) definitions exist: E.g., $\epsilon_{\text{machine}}$ measures the distance from 1 to the adjacent larger floating point, i.e., from $\frac{1}{\beta}\beta$ to $(\frac{1}{\beta} + \frac{1}{\beta^p})\beta$, $\boxed{\epsilon_{\text{machine}} = \beta^{1-p}}$
  - The definition $\epsilon_{\text{machine}} = \beta^{1-p}$ assumes "rounding to zero" (i.e., *directed rounding* towards zero with truncation)
  - If "rounding to nearest" is used, then $\epsilon_{\text{machine}} = \frac{1}{2}\beta^{1-p}$, which is the unit roundoff as is (quite often) used

Floating Point Numbers

- The gaps between adjacent numbers scale with the size of the numbers
- For all $x \in \mathbb{R}$ in the range of a floating point system, there exists a floating point number $\mathrm{fl}(x)$ such that $|x - \mathrm{fl}(x)| \le \epsilon_{\mathrm{machine}}|x|$
- Example: $\beta = 2, p = 3, e_{\min} = -1, e_{\max} = 3$, normalized

$$\left(\frac{d_1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3}\right) 2^e, \qquad e \in \{-1, 0, 1, 2, 3\},$$

$d_1 \equiv 1, \;\; d_2, d_3 \in \{0, 1\}, \quad$ (essentially only need two bits for $p = 3$)



Number of floating points between adjacent powers of 2: $\;2^{p-1} - 1$.
(# of floating points between adjacent powers of β: $\;(\beta - 1)\beta^{p-1} - 1$)

Denormalized (or Subnormal) Numbers

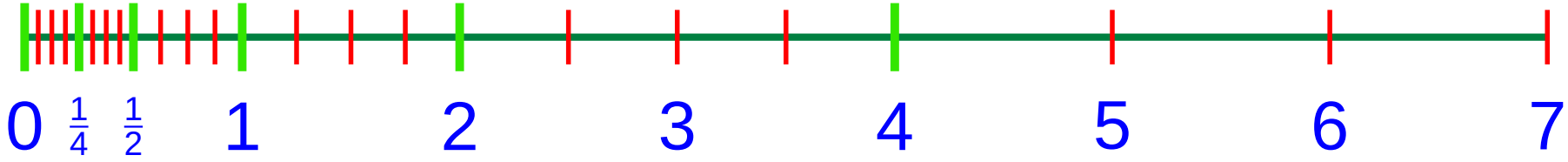
- With normalized significand, $\exists$ a "gap" between 0 and $\beta^{e_{\min}-1}$
- This can result in $x - y = 0$ even though $x \neq y$, and code fragments like **if** $x \neq y$ **then** $z = 1/(x - y)$ might break
- Solution: Allow non-normalized significand when the exponent is $e_{\min}$ (i.e, d_1 can be 0 when $e = e_{\min}$)
- This *gradual underflow* garantees that

$$x = y \iff x - y = 0$$

- Subnormal numbers have lower relative precision than normalized numbers

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

$$\left(d_1/2 + d_2 2^{-2} + d_3 2^{-3}\right) 2^e, \qquad e \in \{-1, 0, 1, 2, 3\}, \qquad d_i \in \{0, 1\}.$$



Two equivalent floating point representations

- The (normalized) representation just discussed uses

$$\pm m\beta^{e-p} = \pm\frac{m}{\beta^p}\beta^e, \quad \text{where } \beta^{p-1} \le m \le \beta^p - 1$$

  The range of m implies that this representation is essentially

$$\pm 0.d_1 d_2 \cdots d_p \times \beta^e = \pm\left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_p}{\beta^p}\right)\beta^e,$$

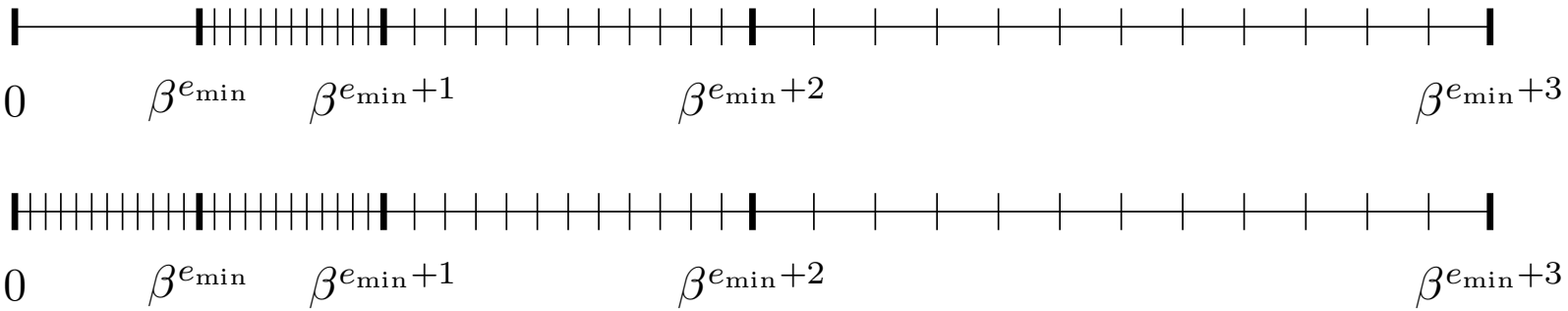
  where $0 \le d_i \le \beta - 1$, and $d_1 \ne 0$.
- Another equivalent representation (more often used, as used in IEEE) is

$$\pm d_1.d_2 \cdots d_p \times \beta^{e-1} = \pm\left(d_1 + \frac{d_2}{\beta^1} + \cdots + \frac{d_p}{\beta^{p-1}}\right)\beta^{e-1},$$

  where $0 \le d_i \le \beta - 1$, and $d_1 \ne 0$.
- No essential difference at all, except that in order to represent the same floating point numbers, the $e_{\min}$ and $e_{\max}$ of the first representation should be 1 greater than that of the second representation. (which can cause some confusion.)
  For example, the previous example using the second representation should be $\beta = 2, p = 3, e_{\min} = -2, e_{\max} = 2$.

An exercise

The following shows a portion of a floating point system

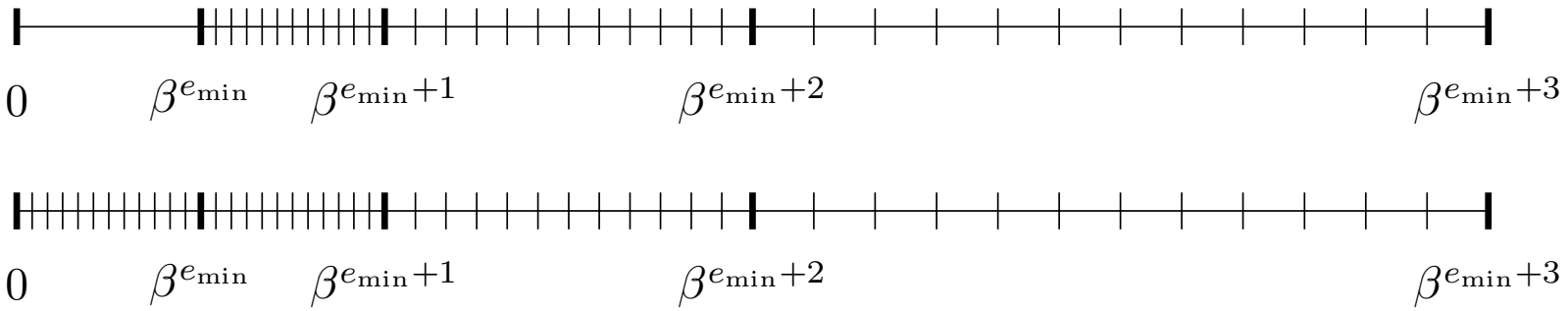


The top one contains the normalized, while the bottom one contains both the normalized and the subnormal, floating points.

1. Which representation is the system using,
   the $0.d_1 d_2 \cdots d_p \times \beta^e$ or the $d_1.d_2 \cdots d_p \times \beta^e$?
2. Determine the possible values of β and p for this system.

An exercise

The following shows a portion of a floating point system



The top one contains the normalized, while the bottom one contains both the normalized and the subnormal, floating points.

1. Which representation is the system using,
   the $0.d_1 d_2 \cdots d_p \times \beta^e$ or the $d_1.d_2 \cdots d_p \times \beta^e$?
2. Determine the possible values of β and p for this system.
   Answer: To solve this problem, apply the formula that determines the number of floating points between adjacent powers of β, which is $(\beta - 1)\beta^{p-1} - 1$. (This formula can be obtained in several ways.)
   Here, since $(\beta - 1)\beta^{p-1} - 1 = 11$, the only two integer solution pairs are $(\beta, p) = (4, 2)$ and $(13, 1)$. (Note the proportion of gap is not drawn correctly to reveal the value of β.)

Special Quantities

- $\pm\infty$ is returned when an operation overflows
- $x/\pm\infty = 0$ for any number x, $x/0 = \pm\infty$ for any nonzero number x
- Operations with infinity are defined as limits, e.g.

$$4 - \infty = \lim_{x\to\infty} 4 - x = -\infty$$

- NaN (Not a Number) is returned when the an operation has no well-defined finite or infinite result .
  — Examples: $\infty - \infty$, ∞/∞, $0/0$, $\mathrm{NaN} \odot x$

IEEE 754 binary formats

Floating-point numbers are stored in computer data as three parts (from left to right): 1. the sign bit, 2. the exponents, 3. the significand.

Sign bit (S)	Exponent bits (E)	Significand bits (M)

The IEEE 754 standard was created in the early 1980s (published version IEEE 754-1985), which has been followed by almost all modern machines. Current version is IEEE 754-2008, which is a revision of IEEE 754-1985 and adds the half-precision type.

IEEE 754 standard represent floating point data using bit sizes as

Precision Type	Sign	Exponent bits, $[e_{\min}, e_{\max}]$	Significand bits, (bits precision)	Total bits	Exponent bias
Half	1	5, [-14,15]	10, (11)	16	15
Single	1	8, [-126,127]	23, (24)	32	127
Double	1	11, [-1022,1023]	52, (53)	64	1023
Quadruple	1	15, [-16382, 16383]	112, (113)	128	16383

- In binary formats the exponent is stored as an unsigned number, with a fixed "bias" to account for the $\pm$ sign of an exponent.
- The listed $[e_{\min}, e_{\max}]$ assume the $1.d_1d_2d_3\cdots d_p \times 2^e$ format.

IEEE Single Precision

- 1 sign bit, 8 exponent bits, 23 significand bits:

0	00000000	00000000000000000000000
S	E (8 bits)	M (23 physical bits, effective 24 bits)
0/1	$e_{\min}=1-127=-126$ $e_{\max}=2^8-2-127=127$	$2^{23}-1$ # of floating reals in $(2^e,\ 2^{e+1})$, for every integer $e\in[e_{\min},e_{\max}]$

- Represented number:

$$(-1)^S \times 1.M \times 2^{E-127}$$

- Special cases:

	$E=0$	$0<E<255$	$E=255$
$M=0$	± 0	Powers of 2	$\pm\infty$
$M\neq 0$	Denormalized	Ordinary numbers	NaN

IEEE Single Precision

- 1 sign bit, 8 exponent bits, 23 significand bits:

0	00000000	00000000000000000000000
S	E (8 bits)	M (23 physical bits, effective 24 bits)
0/1	$e_{\min}=1-127=-126$ $e_{\max}=2^8-2-127=127$	$2^{23}-1$ # of floating reals in $(2^e,\ 2^{e+1})$, for every integer $e\in[e_{\min},e_{\max}]$

- Represented number:

$$(-1)^S \times 1.M \times 2^{E-127}$$

- Special cases:

	$E=0$	$0<E<255$	$E=255$
$M=0$	± 0	Powers of 2	$\pm\infty$
$M\neq 0$	Denormalized	Ordinary numbers	NaN

Comment: Giving up two strings for exponents (representing $E=0$ and $E=255$) to store the special 0 and ∞.

That is why $e_{\min}=-126$ and $e_{\max}=127$.

IEEE Single Precision, Examples

S	E	M	Quantity
0	11111111	00000100000000000000000	NaN
1	11111111	00100010001001010101010	NaN
0	11111111	00000000000000000000000	∞
0	10000001	10100000000000000000000	$+1 \cdot 2^{129-127} \cdot 1.101 = 6.5$
0	10000000	00000000000000000000000	$+1 \cdot 2^{128-127} \cdot 1.0 = 2$
0	00000001	00000000000000000000000	$+1 \cdot 2^{1-127} \cdot 1.0 = 2^{-126}$
0	00000000	10000000000000000000000	$+1 \cdot 2^{-126} \cdot 0.1 = 2^{-127}$
0	00000000	00000000000000000000001	$+1 \cdot 2^{-126} \cdot 2^{-23} = 2^{-149}$
0	00000000	00000000000000000000000	0
1	00000000	00000000000000000000000	-0
1	10000001	10100000000000000000000	$-1 \cdot 2^{129-127} \cdot 1.101 = -6.5$
1	11111111	00000000000000000000000	$-\infty$

IEEE Single and Double Precision binary data type

	Single precision	Double precision
Significand size (p)	24 bits	53 bits
Exponent size	8 bits	11
Exponent bias	$2^7 - 1 = 127$	$2^{10} - 1 = 1023$
Total size	32 bits	64 bits
$e_{\max}$	+127	+1023
$e_{\min}$	-126	-1022
Smallest normalized	$2^{-126} \approx 10^{-38}$	$2^{-1022} \approx 10^{-308}$
Largest normalized	$2^{127} \approx 10^{38}$	$2^{1023} \approx 10^{308}$
unit roundoff (β^{-p})	$2^{-24} \approx 6 \cdot 10^{-8}$	$2^{-53} \approx 10^{-16}$

Floating Point Arithmetic

- Define $\mathrm{fl}(x)$ as the closest floating point approximation to x
- By the definition of $\epsilon_{\text{machine}}$, we have for the relative error:

  For all $x \in \mathbb{R}$ in the range of a floating point system,
  there exists ϵ with $|\epsilon| \leq \epsilon_{\text{machine}}$ such that $\mathrm{fl}(x) = x(1+\epsilon)$
- The result of an operation $\circledast$ using floating point numbers is $\mathrm{fl}(a \circledast b)$
- The arithmetic is said to *rounds correctly* if $\mathrm{fl}(a \circledast b)$ is the nearest floating point number to $a \circledast b$. In a floating point system that rounds correctly (IEEE standard does), the following property holds:
  For all floating point x, y, there exists ϵ with $|\epsilon| \leq \epsilon_{\text{machine}}$ such that $\boxed{x \circledast y = (x * y)(1+\epsilon)}$
- Tie-breaking rule: Round to nearest even (i.e., set the least significant bit to 0)

A few examples *(In Matlab, with IEEE single precision)*

```
>> single(2^23 + [1:22]) - single(2^23)
ans = 1     2     3     4     5     6     7     8     9    10    11    ↩
     12    13    14    15    16    17    18    19    20    21    22
>> single(2^24 + [1:22]) - single(2^24)
ans = 0     2     4     4     4     6     8     8     8    10    12    ↩
     12    12    14    16    16    16    18    20    20    20    22
>> single(2^25 + [1:22]) - single(2^25)
ans = 0     0     4     4     4     8     8     8     8     8    12    ↩
     12    12    16    16    16    16    16    20    20    20    24
>> single(2^26 + [1:22]) - single(2^26)
ans = 0     0     0     0     8     8     8     8     8     8     8    ↩
     16    16    16    16    16    16    16    16    16    24    24
>> single(2^27) + [1:22] - single(2^27)
ans = 0     0     0     0     0     0     0     0    16    16    16    ↩
     16    16    16    16    16    16    16    16    16    16    16
>> single(2^28) + [1:22] - single(2^28)
ans = 0     0     0     0     0     0     0     0     0     0     0 ↩
         0     0     0     0     0    32    32    32    32    32    32
>> single(2^29) + [1:22] - single(2^29)
ans = 0     0     0     0     0     0     0     0     0     0     0 ↩
         0     0     0     0     0     0     0     0     0     0     0
```

A few examples *(In Matlab, with IEEE double precision)*

```
>> 2^52 + [1:22] - 2^52
ans = 1     2     3     4     5     6     7     8     9    10    11    ↩
     12    13    14    15    16    17    18    19    20    21    22
>> 2^53 + [1:22] - 2^53
ans = 0     2     4     4     4     6     8     8     8    10    12    ↩
     12    12    14    16    16    16    18    20    20    20    22
>> 2^54 + [1:22] - 2^54
ans = 0     0     4     4     4     8     8     8     8     8    12    ↩
     12    12    16    16    16    16    16    20    20    20    24
>> 2^55 + [1:22] - 2^55
ans = 0     0     0     0     8     8     8     8     8     8     8    ↩
     16    16    16    16    16    16    16    16    16    24    24
>> 2^56 + [1:22] - 2^56
ans = 0     0     0     0     0     0     0     0    16    16    16    ↩
     16    16    16    16    16    16    16    16    16    16    16
>> 2^57 + [1:22] - 2^57
ans = 0     0     0     0     0     0     0     0     0     0     0 ↩
          0     0     0     0     0    32    32    32    32    32    32
>> 2^58 + [1:22] - 2^58
ans = 0     0     0     0     0     0     0     0     0     0     0 ↩
          0     0     0     0     0     0     0     0     0     0     0
```

A few examples *(In Matlab, with IEEE double precision)*

```
>> format long e

>> eps/2
ans =     1.110223024625157e-16
>> 1. + eps/2 - 1.
ans =     0

>> eps/1.5
ans =     1.480297366166875e-16
>> 1. + eps/1.5 - 1.
ans =     2.220446049250313e-16

>> 2. + eps - 2.
ans =     0
>> 2. + 1.1*eps - 2.
ans =     4.440892098500626e-16
>> 2. + 2*eps - 2.
ans =     4.440892098500626e-16

>> 4. + 2*eps - 4.
ans =     0
>> 4. + 3*eps - 4.
ans =     8.881784197001252e-16
>> 4. + 4*eps - 4.
ans =     8.881784197001252e-16
```

A few examples *(In Matlab, with IEEE double precision)*

```
>> 2^9*eps
ans =      1.136868377216160e-13
>> 1024. + 2^9*eps - 1024.
ans =      0
>> 1024. + (1+1.e-16)*2^9*eps - 1024.
ans =      0
>> 1024. + (1+eps)*2^9*eps - 1024.
ans =      2.273736754432321e-13
>> 1024. + 2^10*eps - 1024.
ans =      2.273736754432321e-13

>> 2^11. + 2^10*eps - 2^11.
ans =      0
>>  3*2^10*eps
ans =      6.821210263296962e-13
>> [ 2^11 + 3*2^10*eps - 2^11,    2^11 + 5*2^10*eps - 2^11 ]
ans =      9.094947017729282e-13     9.094947017729282e-13

>> 2^1000*eps
ans =    2.379227053564453e+285
>> 2^1001+ 2^1000*eps - 2^1001
ans =      0

>> [      2^1022*eps,              2^1023 + 2^1022*eps - 2^1023 ]
ans =    9.979201547673599e+291                          0
```

On eigenvalue problems and related algorithms

- Properties related to eigen-problems
- A few representative algorithms
  - Power method, inverse iteration, shift-inverse iteration
  - RQI
  - The QR algorithm
  - Jacobi iteration, Divide-and-Conquer
- Computing SVD

The Eigenvalue Problem

- The standard eigenvalue problem for $m \times m$ matrix A is

$$Ax = \lambda x$$

  with *eigenvalues* λ and *eigenvectors* x ($x \neq 0$)
- In the direction of an eigenvector, A is "condensed" into a scalar λ
- *Eigenvalue decomposition* of A: (assume A has complete eigenvectors)

$$A = X\Lambda X^{-1} \quad \text{or} \quad AX = X\Lambda$$

  Columns of X are eigenvectors, with corresponding eigenvalues on diagonal of Λ
- In "eigenvector coordinates", A is diagonal:

$$Ax = b \quad \rightarrow \quad (X^{-1}b) = \Lambda(X^{-1}x)$$

Eigen-subspace, invariant subspace, multiplicity

- The span of eigenvectors corresponding to an eigenvalue λ form an *eigen-subspace* E_λ
- Dimension of E_λ = $\dim(\mathrm{null}(A - \lambda I))$ = *geometric multiplicity* of λ
- The span of k linearly independent eigenvectors (corresponding to eigenvalues) form a dimension-k *eigen-subspace* Y_k, which is invariant under A

$$AY_k = Y_k S_k, \qquad \text{with } S_k \in \mathbb{C}^{k\times k}$$

- The *characteristic polynomial* of A is

$$p_A(z) = \det(zI - A) = (z - \lambda_1)(z - \lambda_2)\cdots(z - \lambda_m)$$

- λ is eigenvalue of $A \iff p_A(\lambda) = 0$
  - If λ is eigenvalue, then $\exists x \neq 0, \lambda x - Ax = 0$. Hence $\lambda I - A$ is singular, $\det(\lambda I - A) = 0$.
- Multiplicity of a root λ to p_A = *algebraic multiplicity* of λ
- Any $A \in \mathbb{C}^{m\times m}$ has m eigenvalues, counted with algebraic multiplicity

Similarity Transformations

- The map $A \mapsto X^{-1}AX$ is a *similarity transformation* of A
- $A, B \in \mathbb{C}^{m\times m}$ are called *similar* if there is a similarity transformation $B = X^{-1}AX$
- A and $X^{-1}AX$ have the same characteristic polynomials, eigenvalues, and multiplicities:
  - The characteristic polynomials are the same:

$$\begin{aligned} p_{X^{-1}AX}(z) &= \det(zI - X^{-1}AX) = \det(X^{-1}(zI - A)X) \\ &= \det(X^{-1})\det(zI - A)\det(X) = \det(zI - A) = p_A(z) \end{aligned}$$

  - Therefore, the algebraic multiplicities are the same
  - If E_λ is eigenspace for A, then $X^{-1}E_\lambda$ is eigenspace for $X^{-1}AX$, so geometric multiplicities are the same

Algebraic Multiplicity $\geq$ *Geometric Multiplicity*

- Let n first columns of $\hat{V}$ be orthonormal basis of the eigenspace for λ
- Extend $\hat{V}$ to square unitary V, and form

$$B = V^* A V = \begin{bmatrix} \lambda I & C \\ 0 & D \end{bmatrix}$$

- Since

$$\det(zI - B) = \det(zI - \lambda I)\det(zI - D) = (z - \lambda)^n \det(zI - D)$$

  the algebraic multiplicity of λ (as eigenvalue of B) is $\geq n$
- A and B are similar; so the same is true for λ of A

Defective and Diagonalizable Matrices

- An eigenvalue is called *defective* if its algebraic multiplicity $>$ its geometric multiplicity
- A *defective matrix* is any matrix with at least one defective eigenvalue
- A *nondefective* or *diagonalizable* matrix has equal algebraic and geometric multiplicities for all eigenvalues
- A is nondefective $\iff$
  A is diagonalizable (i.e., $\exists X$ nonsingular, s.t. $A = X\Lambda X^{-1}$)
  - ($\Longleftarrow$) If $A = X\Lambda X^{-1}$, A is similar to Λ and has the same eigenvalues and multiplicities. But Λ is diagonal and thus nondefective.
  - ($\Longrightarrow$) Nondefective A has m linearly independent eigenvectors. Take these as the columns of X, then $A = X\Lambda X^{-1}$.

Eigenvalue-Revealing Factorizations

Three common Eigenvalue-Revealing Factorizations:

- Diagonalization $A = X\Lambda X^{-1}$ (any nondefective A)
- Unitary diagonalization $A = Q\Lambda Q^*$ (any normal A)
- Unitary triangularization (Schur factorization) $A = QSQ^*$ (any A)

A few direct consequences of these decompositions:

- $\mathrm{trace}\,(A) = \mathrm{trace}\,(QSQ^*) = \mathrm{trace}\,(S) = \sum_{j=1}^{m} \lambda_j$
- $\det\,(A) = \det\,(QSQ^*) = \det\,(S) = \prod_{j=1}^{m} \lambda_j$
- Since it is known (by SVD) that $|\det\,(A)\,| = \prod_{j=1}^{m} \sigma_j$, we get

$$\prod_{j=1}^{m} |\lambda_j| = \prod_{j=1}^{m} \sigma_j$$

Eigenvalues and roots of polynomials

- Well-known: Roots of a polynomial lead to eigenvalues: Eigenvalues of A are the roots of $p_A(\lambda) = 0$
- Conversely: Eigenvalues lead to roots of a given polynomial. For any given $p(z) = z^m + a_{m-1}z^{m-1} + \cdots + a_1 z + a_0$, it can be shown that the roots of p are the eigenvalues of its *companion matrix*

$$A = \begin{bmatrix} 0 & & & & & -a_0 \\ 1 & 0 & & & & -a_1 \\ & 1 & 0 & & & -a_2 \\ & & 1 & \ddots & & \vdots \\ & & & \ddots & 0 & -a_{m-2} \\ & & & & 1 & -a_{m-1} \end{bmatrix}$$

- Conclusion: Finding eigenvalues of a matrix is equivalent to solving for roots of a polynomial

Eigenvalue Algorithms

- The obvious method: Find roots of $p_A(\lambda)$, is ill-conditioned
- Instead, compute Schur factorization $A = QSQ^*$ by introducing zeros
- This can not be done in a finite number of steps. In fact

Any eigenvalue solver for $A \in \mathbb{C}^{m\times m}$ with $m \geq 5$ must be iterative

  - Reason: Consider a general polynomial of degree m

$$p(z) = z^m + a_{m-1}z^{m-1} + \cdots + a_1 z + a_0$$

  - There is no closed-form expression for the roots of p: (Abel, 1842) **In general, the roots of polynomial equations higher than fourth degree cannot be written in terms of a finite number of operations**

- Schur factorization is utilized for computing all eigenvalues
- Next we first look at iterative algorithms for computing only one eigenvalue

Eigenvalue Algorithms (*compute 1 eigenvalue*)

- The Power Iteration (Power method)
  - Arguably "the mother of most eigenvalue algorithms"
  - Reveals the "essential ratio" that determines convergence rate
  - The QR algorithm, as well as sparse eigen-algorithms such as Arnoldi/Lanczos/Davidson are all variations of power method (including its block and shift-inverse versions)
- The Shift-Inverse Iteration
  - Essentially "power iteration", but applied to a shift-inverse matrix
- The Rayleigh-Quotient Iteration (RQI)
  - Essentially "power iteration", but applied to a shift-inverse matrix, where the shift is the current Rayleigh-quotient

$$r(x) = \frac{x^* A x}{x^* x}, \qquad x \in \mathbb{C}^m,\ x \neq 0$$

The Power Iteration

Algorithm: The simple Power Iteration

Choose $v^{(0)} =$ a unit length (random) vector

for $k = 0,\ 1,\ 2, \ldots$

$w = Av^{(k)}$	(apply A)
$\lambda^{(k)} = (v^{(k)})^* w$	(Rayleigh quotient, note $\left\|v^{(k)}\right\|_2 \equiv 1$)
$v^{(k+1)} = w/\|w\|_2$	(normalize)

Questions:

1. Under what condition does it converge?

The Power Iteration

Algorithm: The simple Power Iteration

Choose $v^{(0)} =$ a unit length (random) vector

for $k = 0,\ 1,\ 2, \ldots$

$w = Av^{(k)}$	(apply A)
$\lambda^{(k)} = (v^{(k)})^* w$	(Rayleigh quotient, note $\left\|v^{(k)}\right\|_2 \equiv 1$)
$v^{(k+1)} = w/\|w\|_2$	(normalize)

Questions:

1. Under what condition does it converge?
2. How to determine convergence?

The Power Iteration

Algorithm: The simple Power Iteration

Choose $v^{(0)} =$ a unit length (random) vector
for $k = 0,\ 1,\ 2, \ldots$

$w = Av^{(k)}$	(apply A)
$\lambda^{(k)} = (v^{(k)})^* w$	(Rayleigh quotient, note $\left\|v^{(k)}\right\|_2 \equiv 1$)
$v^{(k+1)} = w/\|w\|_2$	(normalize)

Questions:

1. Under what condition does it converge?
2. How to determine convergence?
   - Convergence may be determined from $|\lambda^{(k+1)} - \lambda^{(k)}|$, or from the angle between $v^{(k+1)}$ and $v^{(k)}$, or by the residual norm $\left\|Av^{(k)} - \lambda^{(k)} v^{(k)}\right\|$
3. If it converges, what does it converge to?

Convergence of Power Iteration

- Assume diagonalizable. Expand initial $v^{(0)}$ in the eigenvector basis $\{q_i\}$, and apply A^k:

$$\begin{aligned} v^{(0)} &= a_1 q_1 + a_2 q_2 + \cdots + a_m q_m \\ v^{(k)} &= c_k A^k v^{(0)} = c_k (a_1 \lambda_1^k q_1 + a_2 \lambda_2^k q_2 + \cdots + a_m \lambda_m^k q_m) \\ &= c_k \lambda_1^k (a_1 q_1 + a_2 (\lambda_2/\lambda_1)^k q_2 + \cdots + a_m (\lambda_m/\lambda_1)^k q_m) \end{aligned}$$

- If $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_m|$ and $q_1^T v^{(0)} \neq 0$, then

$$\|v^{(k)} - (\pm q_1)\| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right), \qquad |\lambda^{(k)} - \lambda_1| = \begin{cases} O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right), & \text{if } A = A^* \\ O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{k}\right), & \text{if } A \neq A^* \end{cases}$$

- Converges to the largest eigen-pair, unless eigenvector $q_1 \perp v^{(0)}$, which is unlikely if $v^{(0)}$ is (uniformly/Gaussian) random
- Linear convergence, factor $\approx |\lambda_2/\lambda_1|$ (the gap-ratio), at each iteration

The Shift-Inverse Iteration

- Power method converges to $\max_i |\lambda_i|$ only, and if gap-ratio $|\lambda_2/\lambda_1| \approx 1^-$, then very slow convergence
- Apply power iteration on $(A - \mu I)^{-1}$, (eigenvalues $(\lambda_j - \mu)^{-1}$, converges to a λ closest to μ, with potentially much improved gap-ratio)

Algorithm: Shift-Inverse Iteration

Choose a shift μ, and set $v^{(0)} =$ some unit length (random) vector

for $k = 1, 2, \ldots$

Solve $(A - \mu I)w = v^{(k-1)}$ for w	apply $(A - \mu I)^{-1}$
$v^{(k)} = w/\|w\|$	normalize
$\lambda^{(k)} = (v^{(k)})^* A v^{(k)}$	Rayleigh quotient

- Converges to eigenvector q_J if the shift μ is closest to a simple λ_J (and second closest to $\lambda_L \neq \lambda_J$):

$$\|v^{(k)} - (\pm q_j)\| = O\left(\left|\frac{\mu - \lambda_J}{\mu - \lambda_L}\right|^k\right); \qquad |\lambda^{(k)} - \lambda_J| = O\left(\left|\frac{\mu - \lambda_J}{\mu - \lambda_L}\right|^{\hat{k}}\right), \hat{k} = \begin{cases} 2k & \text{if } A = A^* \\ k & \text{if } A \neq A^* \end{cases}$$

The Rayleigh-Quotient Iteration (RQI)

- The shift μ is constant in shift-inverse iteration, (better convergence if μ is updated to be closer to an eigenvalue)
- Improvement: Set μ as the most current Rayleigh quotient

Algorithm: RQI

Choose $v^{(0)} =$ some unit length (random) vector
Compute $\lambda^{(0)} = (v^{(0)})^* A v^{(0)}$
for $k = 1, 2, \ldots$
  Solve $(A - \lambda^{(k-1)} I) w = v^{(k-1)}$ for w (shift-inverse)
  $v^{(k)} = w / \|w\|$ (normalize)
  $\lambda^{(k)} = (v^{(k)})^* A v^{(k)}$ (current Rayleigh quotient)

Convergence rate:

- (locally) Square in v and λ when A is not hermitian
- (locally) Cubic in v and 6th order in λ when A is hermitian

Block Power Method

- Also called *simultaneous iteration*, or *subspace iteration*, or *orthogonal iteration*
- Can be used to compute more than 1 eigenpairs
- Simultaneously apply Power method to a block of linearly independent vectors

$$V^{(0)} = [v_1^{(0)}, v_2^{(0)}, \cdots, v_n^{(0)}],$$
$$V^{(k)} = A^k V^{(0)} = [A^k v_1^{(0)}, A^k v_2^{(0)}, \cdots, A^k v_n^{(0)}]$$

- Intrinsically ill-conditioned, since from the Power method we know all $A^k v_i^{(0)}$ will converge to the dominant eigenvector
- Rescue: Find an orthonormal basis of $V^{(k)}$ at each step of iteration to enforce linear independence of columns

Block Power Method

Algorithm: The simple Block Power Iteration

Choose $V^{(0)} \in \mathbb{C}^{m\times n}$ with n orthonormal column vectors

for $k = 0,\ 1,\ 2, \ldots$

$W = AV^{(k)}$	(apply A)
$\Lambda^{(k)} = (V^{(k)})^* W$	(block Rayleigh quotient, for convergence test)
$V^{(k+1)}R = W$	(compute QR of W, orthonormalization)

- Under suitable conditions, $V(k)$ converges to an orthonormal basis of the invariant subspace of A spanned by the first n dominant eigenvectors
- Assume $|\lambda_1| \geq \cdots \geq |\lambda_n| > |\lambda_{n+1}| \geq \cdots \geq |\lambda_m|$, then the rate of convergence is linear with factor $|\lambda_{n+1}/\lambda_n|$. With an acceleration scheme by Stewart (1976),

$$|\lambda_i^{(k)} - \lambda_i| = O\left(\left|\frac{\lambda_{n+1}}{\lambda_i}\right|^k\right), \qquad i = 1 : n.$$

Computing all eigenvalues

- The previously discussed methods compute only partial eigenvalues, and they only require matrix-vector products, i.e., A need not be explicitly available, only a subroutine that generates Ax for any x is necessary (the basic requirement of many sparse eigen algorithms)
- Now we turn to eigen algorithms that compute all eigenvalues, they are based on matrix decompositions and usually require A to be explicitly available
  - Based on unitary similarity transformation
  - Based on QR decomposition
  - In essence, they are variants of (shift-inverse) power method, the choice of shift is quite important

Schur Factorization and Diagonalization

- Compute Schur factorization $A = QSQ^*$ by "unitary triangularization":
  Transforming A with similarity transformations

$$\underbrace{Q_j^* \cdots Q_2^* Q_1^*}_{Q^*} A \underbrace{Q_1 Q_2 \cdots Q_j}_{Q}$$

  which converge to S as $j \to \infty$
- For practical reason, an eigen algorithm should converge with a reasonably small j
- For hermitian A, the sequence converges to a diagonal matrix
- Since a real matrix may have complex eigenvalues (and they always appear in conjugate pairs), the Q and S in its Schur form can be complex.
  When only real Q and S are desired, then one uses a *real Schur factorization*, in which S may have 2×2 blocks on its diagonal.

Unitary similarity triangularization

- Goal: Compute a Schur factorization $A = QSQ^*$. Can apply Householder reflectors from left and right to introduce zeros. But directly targeting at upper-triangular form is too ambitious

$$\underset{A}{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}} \xrightarrow{\text{left mult. } Q_1^*} \underset{Q_1^*A}{\begin{bmatrix} \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}} \xrightarrow{\text{right mult. } Q_1} \underset{Q_1^*AQ_1}{\begin{bmatrix} \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}}$$

- The right multiplication destroys the zeros previously introduced
- We already knew similarity transformation to triangular form in finite steps would not work (because of Abel's theorem)
- Will need iteration to reach the goal ($A = QSQ^*$)
- Need two phases, so that the iterative phase can be done as inexpensive (per iteration) as possible

Two Phases of (dense) Eigenvalues Computations

- General A: First to *upper-Hessenberg* form, then to upper-triangular

$$\underset{A \neq A^*}{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}} \xrightarrow[\text{(finite steps)}]{\text{Phase 1}} \underset{H}{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}} \xrightarrow[\text{(iterative)}]{\text{Phase 2}} \underset{S}{\begin{bmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ & & & & \times \end{bmatrix}}$$

- Hermitian A: First to *tridiagonal* form, then to diagonal (both because of symmetry)

$$\underset{A = A^*}{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}} \xrightarrow[\text{(finite steps)}]{\text{Phase 1}} \underset{S}{\begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}} \xrightarrow[\text{(iterative)}]{\text{Phase 2}} \underset{D}{\begin{bmatrix} \times & & & & \\ & \times & & & \\ & & \times & & \\ & & & \times & \\ & & & & \times \end{bmatrix}}$$

First phase: To Hessenberg form

- Try to introduce as many zeros in the (finite steps) first phase
- Need similarity transform: An (upper) Hessenberg form is the best possible form without destroying zeros previously introduced
- First step unitary similarity transform:

$$\underset{A}{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}} \xrightarrow{Q_1^*} \underset{Q_1^*A}{\begin{bmatrix} \times & \times & \times & \times & \times \\ \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}} \xrightarrow{Q_1} \underset{Q_1^*AQ_1}{\begin{bmatrix} \times & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ \times & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}}$$

(zeros introduced by left-mult-Q^* are kept after right-mult-Q)

- Continue in a similar way with column 2:

$$\underset{Q_1^*AQ_1}{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \end{bmatrix}} \xrightarrow{Q_2^*} \underset{Q_2^*Q_1^*AQ_1}{\begin{bmatrix} \times & \times & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ \times & \times & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \times & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & & 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & & 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}} \xrightarrow{Q_2} \underset{Q_2^*Q_1^*AQ_1Q_2}{\begin{bmatrix} \times & \times & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ \times & \times & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \times & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}}$$

First phase: To Hessenberg form

- Reach the (upper) Hessenberg form in $m-2$ (finite) steps:

$$\underbrace{Q_{m-2}^* \cdots Q_2^* Q_1^*}_{Q^*} A \underbrace{Q_1 Q_2 \cdots Q_{m-2}}_{Q} = H = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}$$

- For hermitian A, Hessenberg reduces to tridiagonal (due to symmetry)

$$\underset{A}{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}} \xrightarrow{Q_1^*} \underset{Q_1^* A}{\begin{bmatrix} \times & \times & \times & \times & \times \\ \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ 0 & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}} \xrightarrow{Q_1} \underset{Q_1^* A Q_1}{\begin{bmatrix} \times & \mathbf{X} & 0 & 0 & 0 \\ \times & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \\ & \mathbf{X} & \mathbf{X} & \mathbf{X} & \mathbf{X} \end{bmatrix}} \xrightarrow{\cdots} \underset{T}{\begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}}$$

Producing a hermitian tridiagonal matrix T after $m-2$ steps

$$\underbrace{Q_{m-2}^* \cdots Q_2^* Q_1^*}_{Q^*} A \underbrace{Q_1 Q_2 \cdots Q_{m-2}}_{Q} = T$$

Reduction to Hessenberg by Householder reflectors

Algorithm: Hessenberg by Householder reflectors

for $k = 1$ **to** $m - 2$

$$x = A_{k+1:m,k}$$
$$v_k = \mathrm{sign}(x_1)\|x\|_2 e_1 + x$$
$$v_k = v_k / \|v_k\|_2$$
$$A_{k+1:m,k:m} = A_{k+1:m,k:m} - 2v_k(v_k^* A_{k+1:m,k:m})$$
$$A_{1:m,k+1:m} = A_{1:m,k+1:m} - 2(A_{1:m,k+1:m} v_k) v_k^*$$

Matlab code:

```
function  [H, Q] = hessen(A)
   [m,n]=size(A);  H = A;
   if (nargout>1), Q = eye(n); end
   for k = 1 : n-2
       u = H(k+1:n,k);
       u(1) = sign(u(1))*norm(u)+u(1);  u = u / norm(u);
       H(k+1:n,k:n) = H(k+1:n,k:n) - 2*u*(u'*H(k+1:n,k:n));
       H(1:n,k+1:n) = H(1:n,k+1:n) - 2*(H(1:n,k+1:n)*u)*u';
       if (nargout>1),  % accumulate Q  s.t.  A = QHQ';
          % forward accumulation (backward would use less flops)
          Q(1:n,k+1:n) = Q(1:n,k+1:n) - 2*(Q(1:n,k+1:n)*u)*u';
       end
   end
```

Reduction to Hessenberg (*Another implementation*)

```
function  [u, tau] = house_gen(x)
% generates a householder reflector H = I - uu' st H*x = tau*e_1,
% where |tau|= norm(x),  (note here norm(u,2) = sqrt(2))
  u = x;   tau = norm(x);  if tau == 0,  u(1)=sqrt(2);  return,  end
  u = x/tau;
  if u(1) >= 0,  u(1)= u(1)+1;  tau = -tau;  else,  u(1)= u(1)-1;  end
  u = u/sqrt(abs(u(1)));

function  [H, Q] = hessen2(A)
  [m,n]=size(A);  H=A;
  Q = eye(n);
  for k = 1 : n-2
     [Q(k+1:n,k), H(k+1,k)] = house_gen( H(k+1:n,k) );
     % premultiply by (I - uu'),  u = Q(k+1:n,k);
     H(k+1:n,k+1:n)=H(k+1:n,k+1:n) - ...
                    Q(k+1:n,k)*(Q(k+1:n,k)'*H(k+1:n,k+1:n));
     H(k+2:n,k) = zeros(n-k-1,1);
     % postmultiply by (I - uu')
     H(1:n,k+1:n)=H(1:n,k+1:n)-(H(1:n,k+1:n)*Q(k+1:n,k))*Q(k+1:n,k)';
  end
  % accumulate Q, use backward accumulation (less flops)
  for k = n-2 : -1 : 1
      u = Q(k+1:n,k);
      Q(k+1:n,k+1:n) = Q(k+1:n,k+1:n) - u*(u'*Q(k+1:n,k+1:n));
      Q(:,k)=zeros(n,1);  Q(k,k)=1;
  end
```

Operation counts and stability

- Operation count (*not* twice Householder QR):
  Main operations:

$$A_{k+1:m,k:m} = A_{k+1:m,k:m} - 2v_k(v_k^* A_{k+1:m,k:m})$$
$$A_{1:m,k+1:m} = A_{1:m,k+1:m} - 2(A_{1:m,k+1:m}v_k)v_k^*$$

$$\sum_{k=1}^{m} 4(m-k)^2 + 4m(m-k) = \underbrace{4m^3/3}_{QR} + 4m^3 - 4m^3/2 = 10m^3/3$$

- For hermitian A, flop count is twice QR divided by two $= 4m^3/3$
- The Householder Hessenberg reduction algorithm is backward stable:

$$\tilde{Q}\tilde{H}\tilde{Q}^* = A + \delta A, \qquad \frac{\|\delta A\|}{\|A\|} = O(\epsilon_{\text{machine}})$$

where $\tilde{Q}$ is an exactly unitary matrix based on $\tilde{v}_k$

Main picture of the QR algorithm

Change notation a bit, use V to denote the unitary matrix that transforms A into H, i.e., $V^*AV = H$

i. reduce A to upper Hessenberg form: $AV = VH$

ii. while not convergent `Do` :

1. select a shift μ
2. QR factorization of the shifted H: $QR = H - \mu I$
3. update V: $V \leftarrow VQ$
4. update H: $H \leftarrow RQ + \mu I \;\; (= Q^*HQ)$

Denote $V^+ = VQ$ the updated matrix with columns $[v_1^+, v_2^+, \cdots, v_m^+]$:

$$AV = VH = V(QR + \mu I) \;\; \Rightarrow \;\; (A - \mu I)V = VQR$$
$$\Rightarrow \;\; (A - \mu I)v_1 = v_1^+ r_{11}$$

(shifted A power iteration on the first column of V)

$$V^*(A - \mu I)^{-1} = R^{-1}(VQ)^* \;\; \Rightarrow \;\; RV^* = (VQ)^*(A - \mu I)$$
$$\Rightarrow \;\; VR^* = (A - \mu I)^*(VQ) \;\; \Rightarrow \;\; v_m r_{mm}^* = (A - \mu I)^* v_m^+$$

(shifted A^* inverse iteration on the last column of V)

Understanding the QR algorithm

A step further: If we look at a block of V (e.g., the full V) instead of just one single vector, then

$$(A - \mu I)V = VQR = V^{+}R$$

$\Rightarrow$ At each iteration, QR is block power iteration with shift μ

$\Rightarrow$ In total, QR is subspace iteration with variable shifts

$$VR^{*} = (A - \mu I)^{*}(VQ) = (A - \mu I)^{*}V^{+}$$

$\Rightarrow$ At each iteration, QR is inverse block power iteration with shift μ

$\Rightarrow$ In total, QR is inverse subspace iteration with variable shifts

(guaranteed convergence with suitably chosen shifts)

That is, QR algorithm does both subspace iteration and shift-inverse subspace iteration on each column of V at the same time.

Second phase: From Hessenberg to Triangular

This iterative phase essentially contains two steps

- QR factorization of a shifted H: $\quad QR = H - \mu I$
- Reverse multiplication of the QR factors, plus shift: $H \leftarrow RQ + \mu I$

A is pre-processed into a Hessenberg form ($V^*AV = H$) because QR decomposition of H is only of $O(m^2)$ complexity, instead of the $O(m^3)$ for a general A. Can use either of two approaches to reduce H to R:

- By Givens rotator (only 1 non-zero to zero out per step)
- By Householder reflector of length-2 (instead of length-m) per step . (for real A using real arithmetic, use length-3 reflectors)

The other two key properties:

- Each update of $H^+ = RQ + \mu I$ is a similarity transform of the previous H: $\quad H^+ = Q^*HQ$
- Each updated H still maintains upper Hessenberg form (why?)

Choices of shifts

At a certain iteration step, obtain a shift from a 2×2 diagonal block of the current H, say, $H_2(k) := \begin{bmatrix} h_{k-1,k-1} & h_{k-1,k} \\ h_{k-1,k} & h_{k,k} \end{bmatrix}$. Usually obtain a shift from H in a bottom-to-top manner. That is, k from m downto 2.

- Rayleigh-quotient shift: (mainly for hermitian matrix)
  Set $\mu = h_{k,k}$, note that $h_{k,k} = v_k^* A v_k$ is a readily available RQ.
  Questions: Why RQ shift can fail to converge for real nonsymmetric matrix with complex eigenvalues?
- Wilkinson shift: Set μ as the eigenvalue of the 2×2 matrix $H_2(k)$ that is closer to $h_{k,k}$.
  Convergence rate: Quadratic for $A \neq A^*$, cubic for $A = A^*$.
  Needs on average two QR iterations to converge an eigenvalue, which makes the QR algorithm behave like a "direct" method.
- Francis double shifts: Use (implicitly) both of the eigenvalues of the 2×2 matrix $H_2(k)$ as the double shifts. (For real A using real arithmetic)

Choices of shifts

- Need “exceptional shift” for a small set of matrices, e.g.

$$\begin{bmatrix} 0 & 0 & 0 & h \\ h & 0 & 0 & 0 \\ 0 & h & 0 & 0 \\ 0 & 0 & h & 0 \end{bmatrix}, \qquad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 10^{-13} & 0 \\ 0 & -10^{-13} & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Deflation of converged eigenvalues in QR algorithm

- Mainly utilize the "shift-inverse power" property of the QR algorithm for fast convergence:
  Recall that QR algorithm performs "shift-inverse" iteration on the last column of V
- With Wilkinson shift, the convergence rate is at least quadratic, and the last column in V typically converges first
- Therefore, deflate converged columns in V from the last column to the first
- That is, check convergence in H from bottom up. Typically, the last subdiagonal elements in H decreases to 0 fastest

A very simplified sample code of QR algorithm

```
function [H, V] = qrschur(A, tol);
 % compute A=VHV', where H converges to upper triangular

  [m, n] = size(A);  H = zeros(n,n);
  if (nargout > 2), [H, V] = hessen2(A);  else, [H]=hessen(A); end
  k = n;  it = 1;  itmax = n^2;
  while (k > 1  &  it <=itmax)
     % compute the Wilkinson shift
     mu = eig(H(k-1:k,k-1:k));
     if abs(mu(1)-H(k,k))<=abs(mu(2)-H(k,k)),  mu = mu(1);
     else, mu = mu(2); end

     % compute QR (should use Givens or length-2 Householder instead,
     % should use implicit shift instead of explicit shift)
     [Q, R] = qr(H(1:k,1:k) - mu*eye(k));
     H(:,1:k) = H(:,1:k)*Q;   H(1:k,:) = Q'*H(1:k,:);

     if (nargout > 2), V(:,1:k) = V(:,1:k)*Q;  end  %update V

     % deflate if a subdiagonal is small enough
     if abs(H(k,k-1)) < tol*(abs(H(k-1,k-1))+abs(H(k,k))),
        H(k,k-1) = 0;  k = k-1;
     end
     it = it + 1;
  end
```

A few demos of the convergence of the QR algorithm

Click the following links for some online demos

- A symmetric, with Wilkinson shift [2]
- A symmetric, with Rayleigh quotient shift [3]
- A nonsymmetric, with Wilkinson shift [4]

[2] http://faculty.smu.edu/yzhou/Teach/demo/sym_wilks.gif
[3] http://faculty.smu.edu/yzhou/Teach/demo/sym_RQshifts.gif
[4] http://faculty.smu.edu/yzhou/Teach/demo/nonsym_wilks.gif

Quite a few details left out

- When $A \in \mathbb{R}^{m \times m}$, do not want to use complex arithmetic, instead, using real arithmetic, perform implicit double shifts to compute "real" Schur form.
- For the iterative 2nd phase, exploit the "implicit Q theorem" to get the QR decomposition of the shifted matrix (either $H - \mu I$ or a double shifted $H^2 - sH + tI$) without using explicit shift
- Using Givens rotator or length-2/3 Householder reflectors for the iterative process to go from Hessenberg to triangular
- Details in Golub and Van Loan's "matrix computations",
  or, J. Demmel's "Applied Numerical Linear Algebra",
  or, G. W. Stewart's "Matrix algorithms, Vol 2".

The implicit Q theorem

Theorem: Given $A \in \mathbb{C}^{m\times m}$. Let U and V be two unitary matrices, with $U^*AU = H$, $V^*AV = G$, where H, G are of unreduced upper Hessenberg form. If $u_1 = v_1$, then $u_i = c_i v_i$ with $|c_i| = 1$, and $|h_{ij}| = |g_{ij}|$.

In words, if A is unitarily transformed into unreduced Hessenberg form by similarity transformation, and the first columns of the unitary matrices are identical, then the remaining columns are identical upto a complex sign.

Proof: Quite straightforward noting that u_k only depends on $u_1, \cdots u_{k-1}$ in $AU = UH$ when H is unreduced upper Hessenberg. Comparing columns in $AU = UH$ and $AV = VG$, it becomes apparent that $u_1 = v_1$ is enough to guarantee that u_i is parallel to v_i for all i. ■

Other Eigen Algorithms 1: Jacobi iteration

- *Jacobi rotator* $J = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$, looks very much like a Givens rotator, but with an intrinsic difference:
  The need to keep a similarity transformation.
  E.g., Diagonalize a 2×2 real symmetric matrix using J

$$J^T \begin{bmatrix} a & d \\ d & b \end{bmatrix} J = \begin{bmatrix} \mathbf{X} & 0 \\ 0 & \mathbf{X} \end{bmatrix} \quad \Longrightarrow \quad \tan(2\theta) = \frac{2d}{b-a}$$

- Iteratively apply transformation to 2 rows and 2 columns of $A \in \mathbb{R}^{m \times m}$
- Loop over all pairs of rows/columns, quadratic convergence
- $O(m^2)$ steps, $O(m)$ operations per step $\Longrightarrow O(m^3)$ operation count

Other Eigen Algorithms 2: Divide-and-Conquer

- Assume T is symmetric tridiagonal, split T into submatrices:

$$T = \left[\begin{array}{c|c} T_1 & \beta \\ \hline \beta & T_2 \end{array}\right] = \left[\begin{array}{c|c} \hat{T}_1 & \\ \hline & \hat{T}_2 \end{array}\right] + \left[\begin{array}{c|c} & \beta \\ \hline \beta & \beta \end{array}\right]$$

- The sum of a 2×2 block-diagonal matrix and a rank-one correction
- Split T and compute eigenvalues of $\hat{T}_1, \hat{T}_2$ recursively
- Assume diagonalizations $\hat{T}_1 = Q_1 D_1 Q_1^T$ and $\hat{T}_2 = Q_2 D_2 Q_2^T$ have been computed, then

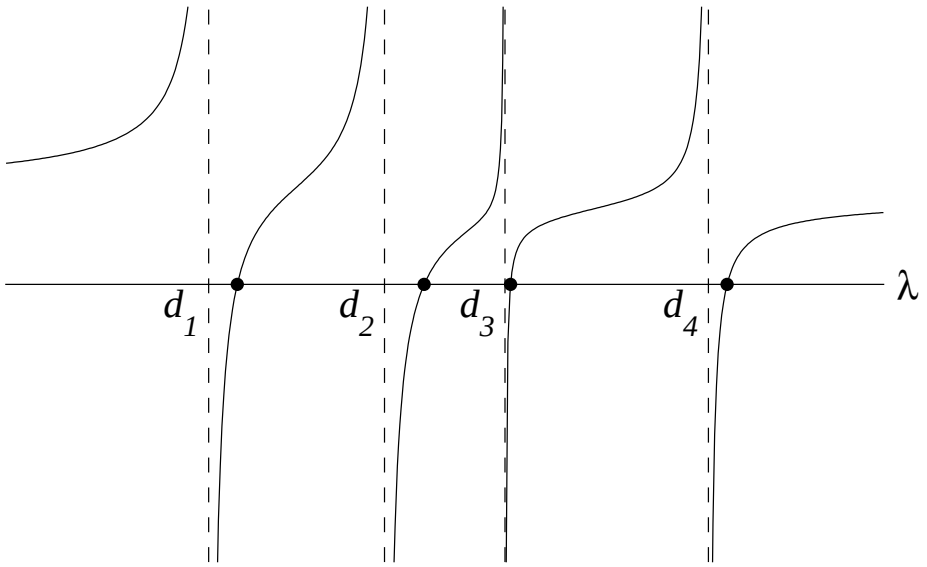
$$T = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \left(\begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} + \beta z z^T \right) \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix}$$

with $z^T = (q_1^T, q_2^T)$, where q_1^T is last row of Q_1 and q_2^T is first row of Q_2

Secular equation of Divide-and-Conquer

- Eigenvalues of T are the eigenvalues of $\begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} + \beta z z^T$
- Solve a (nonlinear) secular equation to get eigenvalues of T from those of $\hat{T}_1, \hat{T}_2$
- In general, eigenvalues of $D + ww^T$ are the roots of the secular equation

$$f(\lambda) := 1 + \sum_{j=1}^{m} \frac{w_j^2}{d_j - \lambda} = 0, \qquad \text{where } w^T = [w_1, w_2, \cdots, w_m]$$



Cost of Divide-and-Conquer

- Solve the *secular equation* $f(\lambda) = 0$ with a nonlinear solver, such as Newton's method on each interval (d_i, d_{i+1})
- Very fast convergence, typically $O(m)$ flops per root, $O(m^2)$ flops for all roots
- Total cost for divide-and-conquer algorithm (for computing eigenvalues only):

$$O\left(m^2 + 2\frac{m^2}{2^2} + 4\frac{m^2}{4^2} + 8\frac{m^2}{8^2} + \cdots + m\frac{m^2}{m^2}\right) = O(m^2)$$

- Most of the operations are spent in reducing A into the tridiagonal T, and the constant in "Phase 2" is not important
- However, for computing eigenvectors, divide-and-conquer reduces Phase 2 to $4m^3/3$ flops compared to $6m^3$ for the QR algorithm [5]

[5] Stable algorithm for computing eigenvectors within DC developed one decade later since the 1st DC algorithm was proposed

Computing SVD

Two phases for dense SVD

- Phases 1: (direct finite steps)
  Unitary bi-diagonalization
  $A \to \tilde{U}^* A \tilde{V} = B$ where B is bi-diagonal
- Phases 2: (iterative) Iterate from bi-diagonal to diagonal.
  Essentially performing QR algorithm on the tridiagonal Hermitian B^*B, but without forming B^*B explicitly
- Most of the important details of computing SVD can be found in these matlab files:
  - Phase 1 bi-diagonalization by Householder reflectors (bidiag.m)
  - Phase 2 iteration to diagonal form (svdbiqr.m) ,
    this code calls the implicit shifted QR using Given rotators (biqr.m)
  - Although mostly coded from scratch, (for small/medium sized matrices) these codes compute SVD with comparable performance to the Matlab built-in function `svd` which calls Lapack