Orthogonal and Orthonormal Bases, Orthogonal Matrix, Gram-Schmidt

Orthogonal and Orthonormal Bases

A collection of nonzero vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{R}^m$ is said to be

- orthogonal if $\mathbf{a}_i^T \mathbf{a}_j = 0$ for all i, j with $i \neq j$
- orthonormal if $\|\mathbf{a}_i\|_2 = 1$ for all i and $\mathbf{a}_i^T \mathbf{a}_j = 0$ for all i, j with $i \neq j$.

The same definition applies to complex $\mathbf{a}_i$'s, but we need to replace "T" with "H".

Examples:

- $\{\mathbf{e}_1, \ldots, \mathbf{e}_m\} \subset \mathbb{R}^m$ is orthonormal; in fact, it's an orthonormal basis for $\mathbb{R}^m$
- any subset of $\{\mathbf{e}_1, \ldots, \mathbf{e}_m\}$ is orthornormal
- (to be learnt) discrete Fourier transform (DFT), Haar transform, etc., form orthonormal bases

Orthogonal and Orthonormal Bases

Some immediate facts:

- an orthonormal set of vectors is also linearly independent.
- let $\{\mathbf{a}_1, \ldots, \mathbf{a}_n\} \subset \mathbb{R}^m$ be an orthonormal set of vectors. Suppose $\mathbf{y} \in \text{span}\{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$. Then the coefficient $\boldsymbol{\alpha}$ for the representation

$$\mathbf{y} = \sum_{i=1}^{n} \alpha_i \mathbf{a}_i$$

  is uniquely given by $\alpha_i = \mathbf{a}_i^T \mathbf{y}$, $i = 1, \ldots, n$.

A not so immediate fact:

- (important) every subspace $\mathcal{S}$ with $\mathcal{S} \neq \{\mathbf{0}\}$ has an orthonormal basis.
  - this will be clear when we consider Gram-Schmidt later

Orthogonal Matrices

A real matrix $\mathbf{Q}$ is said to be

- orthogonal if it is square and its columns are orthonormal
  (note: we often call it an orthogonal matrix, but not an orthonormal matrix)
- semi-orthogonal if its columns are orthonormal
  - a semi-orthogonal $\mathbf{Q}$ must be tall or square

A complex matrix $\mathbf{Q}$ is said to be

- unitary if it is square and its columns are orthonormal,
- semi-unitary if its columns are orthonormal.

Orthogonal Matrices

Facts:

- $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ and $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$ for orthogonal $\mathbf{Q}$
- $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ (but *not* necessarily $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$) for semi-orthogonal $\mathbf{Q}$
- $\mathbf{Q}^{-1} = \mathbf{Q}^T$ for orthogonal $\mathbf{Q}$
- $\mathbf{Q}^T$ is orthogonal if $\mathbf{Q}$ is orthogonal
- $|\det(\mathbf{Q})| = 1$ for orthogonal $\mathbf{Q}$
- the set of columns of semi-orthogonal $\mathbf{Q}$ is a basis for $\mathcal{R}(\mathbf{Q})$
- (isometry property) $\|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$ for semi-orthogonal $\mathbf{Q}$
  - physical meaning for square $\mathbf{Q}$: rotation and reflection (to be learnt next) do not affect the vector length
- for every tall and semi-orthogonal matrix $\mathbf{Q}_1 \in \mathbb{R}^{n\times k}$, there exists a matrix $\mathbf{Q}_2 \in \mathbb{R}^{n\times(n-k)}$ such that $[\ \mathbf{Q}_1\ \mathbf{Q}_2\ ]$ is orthogonal

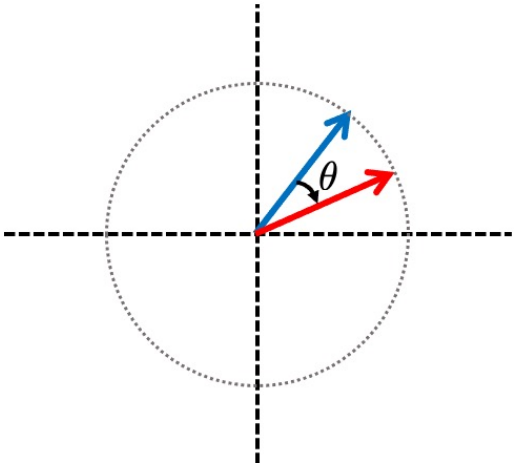
note: similar results hold for unitary and semi-unitary matrices

Orthogonal Matrices

- a transformation $\mathbf{y} = \mathbf{Q}\mathbf{x}$ with orthogonal $\mathbf{Q}$ performs rotations and/or reflections

Rotation in $\mathbb{R}^2$: Consider a rotation matrix

$$\mathbf{Q} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix},$$



where $\theta \in [0, 2\pi)$. It describes a rotation by θ.

Rotation in a coordinate plane in $\mathbb{R}^n$. For example,

$$\mathbf{Q} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

describes a rotation in the (x_1, x_3) plane in $\mathbb{R}^3$.

Orthogonal Matrices

- Reflection in $\mathbb{R}^2$: Consider a reflection matrix

$$\mathbf{Q} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{bmatrix},$$

  where $\theta \in [0, 2\pi)$. It describes a reflection across a line at an angle of $\frac{\theta}{2}$.

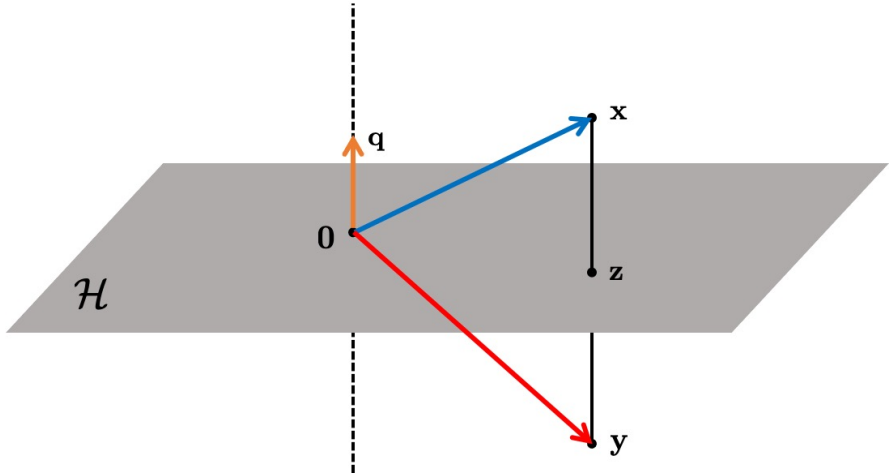
A Householder reflection: a matrix of the form

$$\mathbf{Q} = \mathbf{I} - 2\mathbf{q}\mathbf{q}^T$$

with $\mathbf{q}$ a unit-norm vector (i.e., $\|\mathbf{q}\|_2 = 1$)

- Properties: a reflection matrix is orthogonal and symmetric

Orthogonal Matrices



- $\mathcal{H}= \{\mathbf{u} \mid \mathbf{q}^T\mathbf{u} = \mathbf{0}\}$ is the (hyper-)plane of vectors orthogonal to $\mathbf{q}$
- if $\|\mathbf{q}\|_2 = 1$, the projection of $\mathbf{x}$ on $\mathcal{H}$ is given by

$$\mathbf{z} = \mathbf{x} - (\mathbf{q}^T\mathbf{x})\mathbf{q} = \mathbf{x} - \mathbf{q}(\mathbf{q}^T\mathbf{x}) = (\mathbf{I} - \mathbf{q}\mathbf{q}^T)\mathbf{x}$$

- reflection of $\mathbf{x}$ through the hyperplane is given by product with reflection matrix:

$$\mathbf{y} = \mathbf{z} + (\mathbf{z} - \mathbf{x}) = (\mathbf{I} - 2\mathbf{q}\mathbf{q}^T)\mathbf{x} = \mathbf{Q}\mathbf{x}$$

Orthogonal Matrices

A permutation matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is defined as

$$q_{ij} = \begin{cases} 1 & j = \pi_i \\ 0 & \text{otherwise} \end{cases}$$

where $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_n]^T$ is a permutation of $[1, \ldots, n]^T$

- interpretation: $\mathbf{Qx} = [x_{\pi_1}, ..., x_{\pi_n}]^T$; $\mathbf{QX}$ ($\mathbf{XQ}$) permutation of rows (columns)
- $\mathbf{Q}$ has exactly one element equal to 1 in each row and each column
- $\mathbf{Q}$ can be obtained by reordering the columns/rows of $\mathbf{I}_n$ or arranging $\mathbf{e}_i$'s
- for permutation matrices $\mathbf{Q}_1, \ldots, \mathbf{Q}_n$, $\mathbf{Q}_1 \cdots \mathbf{Q}_n$ is a permutation matrix
- a (general) permutation matrix can be decomposed into a product of elementary permutation matrices (only interchange two elements for $\mathbf{Qx}$)
- permutation matrices are orthogonal
  - $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ because
    $$[\mathbf{Q}^T\mathbf{Q}]_{ij} = \sum_{k=1}^{n} [\mathbf{Q}^T]_{ik}[\mathbf{Q}]_{kj} = \sum_{k=1}^{n} [\mathbf{Q}]_{ki}[\mathbf{Q}]_{kj} = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases}$$
  - $\mathbf{Q}^T = \mathbf{Q}^{-1}$ is the inverse permutation matrix

Orthogonal Bases and Matrices

Question: given a subspace $\mathcal{S}$, how do we know that it has an orthonormal basis?

- we know that every subspace has a basis, c.f. Theorem 1
- but the theorem doesn't say if that basis is orthonormal
- we can construct an orthonormal basis from a basis—and one way to do it is the Gram-Schmidt procedure

Gram-Schmidt Procedure

Algorithm: Gram-Schmidt
input: a collection of vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$, presumably linearly independent
$\tilde{\mathbf{q}}_1 = \mathbf{a}_1$, $\mathbf{q}_1 = \tilde{\mathbf{q}}_1 / \|\tilde{\mathbf{q}}_1\|_2$
for $i = 2, \ldots, n$
$\quad \tilde{\mathbf{q}}_i = \mathbf{a}_i - \sum_{j=1}^{i-1} (\mathbf{q}_j^T \mathbf{a}_i) \mathbf{q}_j$
$\quad \mathbf{q}_i = \tilde{\mathbf{q}}_i / \|\tilde{\mathbf{q}}_i\|_2$
end
output: $\mathbf{q}_1, \ldots, \mathbf{q}_n$

- Fact: Suppose that $\mathbf{a}_1, \ldots, \mathbf{a}_n$ are linearly independent. The collection of vectors $\mathbf{q}_1, \ldots, \mathbf{q}_n$ produced by the Gram-Schmidt procedure is orthonormal and satisfies

$$\text{span}\{\mathbf{a}_1, \ldots, \mathbf{a}_n\} = \text{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}.$$

- here we use Gram-Schmidt to identify the existence of an orthonormal basis for a subspace, but it is a numerical algorithm

Gram-Schmidt Procedure

Proof of the fact on the last page:

- assume linearly independent $\mathbf{a}_1, \ldots, \mathbf{a}_n$
- consider $i = 2$.
  - $\tilde{\mathbf{q}}_2$ is a linear combination of $\mathbf{a}_1, \mathbf{a}_2$ and is nonzero:

$$\tilde{\mathbf{q}}_2 = \mathbf{a}_2 - (\mathbf{q}_1^T \mathbf{a}_2)\mathbf{q}_1 = \mathbf{a}_2 - (\mathbf{q}_1^T \mathbf{a}_2 / \|\mathbf{a}_1\|_2)\mathbf{a}_1; \qquad (\dagger)$$

    the linear independence of $\mathbf{a}_1, \mathbf{a}_2$ implies $\tilde{\mathbf{q}}_2 \neq \mathbf{0}$.
  - $\mathbf{a}_2$ is a linear combination of $\mathbf{q}_1, \mathbf{q}_2$: seen from $(\dagger)$
  - consequence: $\mathrm{span}\{\mathbf{a}_1, \mathbf{a}_2\} = \mathrm{span}\{\mathbf{q}_1, \mathbf{q}_2\}$ (why?)
  - $\tilde{\mathbf{q}}_2$ is orthogonal to $\mathbf{q}_1$:

$$\mathbf{q}_1^T \tilde{\mathbf{q}}_2 = \mathbf{q}_1^T (\mathbf{a}_2 - (\mathbf{q}_1^T \mathbf{a}_2)\mathbf{q}_1) = \mathbf{q}_1^T \mathbf{a}_2 - \mathbf{q}_1^T \mathbf{a}_2 = 0.$$

Gram-Schmidt Procedure

- consider $i \geq 2$.
  - $\tilde{\mathbf{q}}_i$ is a linear combination of $\mathbf{a}_1, \ldots, \mathbf{a}_{i-1}$ and is nonzero: by induction, $\mathbf{q}_1, \ldots \mathbf{q}_{i-1}$ are linear combinations of $\mathbf{a}_1, \ldots, \mathbf{a}_{i-1}$. So,

$$\tilde{\mathbf{q}}_i = \mathbf{a}_i - \textstyle\sum_{j=1}^{i-1} (\mathbf{q}_j^T \mathbf{a}_i) \mathbf{q}_j \qquad (\ddagger)$$

    is a linear combination of $\mathbf{a}_1, \ldots, \mathbf{a}_i$. The linear independence of $\mathbf{a}_1, \ldots, \mathbf{a}_i$ implies $\tilde{\mathbf{q}}_i \neq \mathbf{0}$.
  - $\mathbf{a}_i$ is a linear combination of $\mathbf{q}_1, \ldots, \mathbf{q}_i$: seen from $(\ddagger)$
  - consequence: $\operatorname{span}\{\mathbf{a}_1, \ldots, \mathbf{a}_i\} = \operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_i\}$
  - $\tilde{\mathbf{q}}_i$ is orthogonal to $\mathbf{q}_1, \ldots, \mathbf{q}_{i-1}$: by induction, $\mathbf{q}_1, \ldots, \mathbf{q}_{i-1}$ are orthonormal. For any $k \in \{1, \ldots, i-1\}$,

$$\mathbf{q}_k^T \tilde{\mathbf{q}}_i = \mathbf{q}_k^T (\mathbf{a}_i - \textstyle\sum_{j=1}^{i-1} (\mathbf{q}_j^T \mathbf{a}_i) \mathbf{q}_j) = \mathbf{q}_k^T \mathbf{a}_i - \mathbf{q}_k^T \mathbf{a}_i = 0.$$

Gram-Schmidt Procedure

More comments:

- the step

  $$\tilde{\mathbf{q}}_i = \mathbf{a}_i - \sum_{j=1}^{i-1} (\mathbf{q}_j^T \mathbf{a}_i) \mathbf{q}_j$$

  can be shown to be equivalent to

  $$\tilde{\mathbf{q}}_i = \Pi_{\text{span}\{\mathbf{q}_1,\ldots,\mathbf{q}_{i-1}\}^\perp}(\mathbf{a}_i) = \Pi_{\text{span}\{\mathbf{a}_1,\ldots,\mathbf{a}_{i-1}\}^\perp}(\mathbf{a}_i);$$

  this will be seen in the Least Squares Topic.

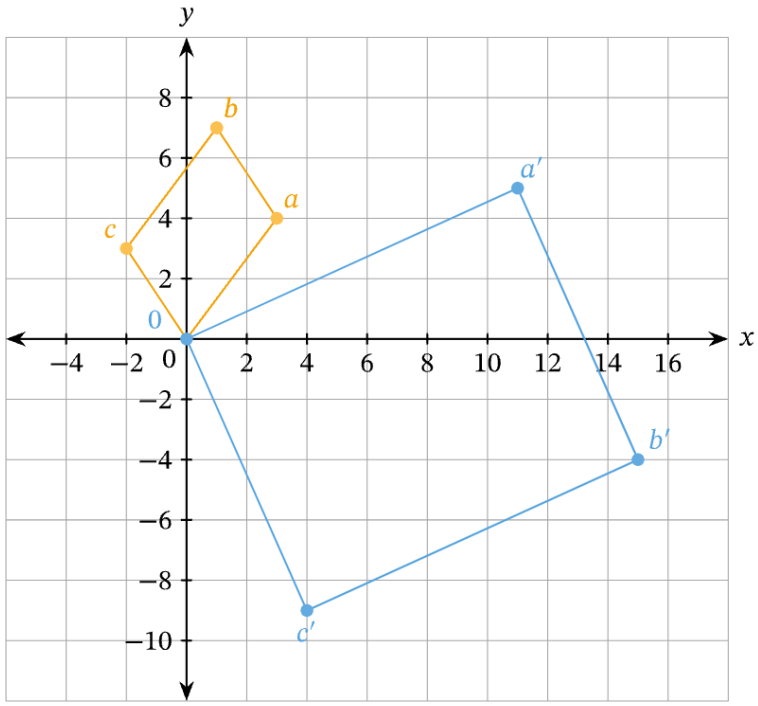
- the Gram-Schmidt procedure can be modified in various ways
  - e.g., it can be modified to do linear independence test, or to find a maximal linearly independent vector subset

Matrix Multiplications and Representations, Block Matrix Manipulations

Linear Transformations of Matrix

A matrix, $\mathbf{A} \in \mathbb{R}^{m \times n}$, is a 2-dimensional array of numbers that represents a linear transformation, $L : \mathbb{R}^n \to \mathbb{R}^m$, such that for all $\mathbf{x} \in \mathbb{R}^n$ the matrix-vector multiplication $\mathbf{Ax}$ yields the same result as does $L(\mathbf{x})$.

Let us take examples in $\mathbb{R}^2$



Matrix Product Representations

Let $\mathbf{A} \in \mathbb{R}^{m \times k}$, $\mathbf{B} \in \mathbb{R}^{k \times n}$, and consider

$$\mathbf{C} = \mathbf{AB}.$$

- column representation:

$$\mathbf{c}_i = \mathbf{A}\mathbf{b}_i, \quad i = 1, \ldots, n$$

- row representation: redefine $\mathbf{c}_i \in \mathbb{R}^n, \mathbf{a}_i \in \mathbb{R}^k$ as the ith row of $\mathbf{C}$, $\mathbf{A}$, respectively.

$$\mathbf{c}_i^T = \mathbf{a}_i^T \mathbf{B}, \quad i = 1, \ldots, n$$

Matrix Product Representations

- inner-product representation: redefine $\mathbf{a}_i \in \mathbb{R}^k$ as the ith row of $\mathbf{A}$.

$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_m^T \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 & \cdots & \mathbf{b}_n \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^T\mathbf{b}_1 & \cdots & \mathbf{a}_1^T\mathbf{b}_n \\ \vdots & & \vdots \\ \mathbf{a}_m^T\mathbf{b}_1 & \cdots & \mathbf{b}_m^T\mathbf{b}_n \end{bmatrix}$$

Thus,

$$c_{ij} = \mathbf{a}_i^T\mathbf{b}_j, \quad \text{for any } i, j.$$

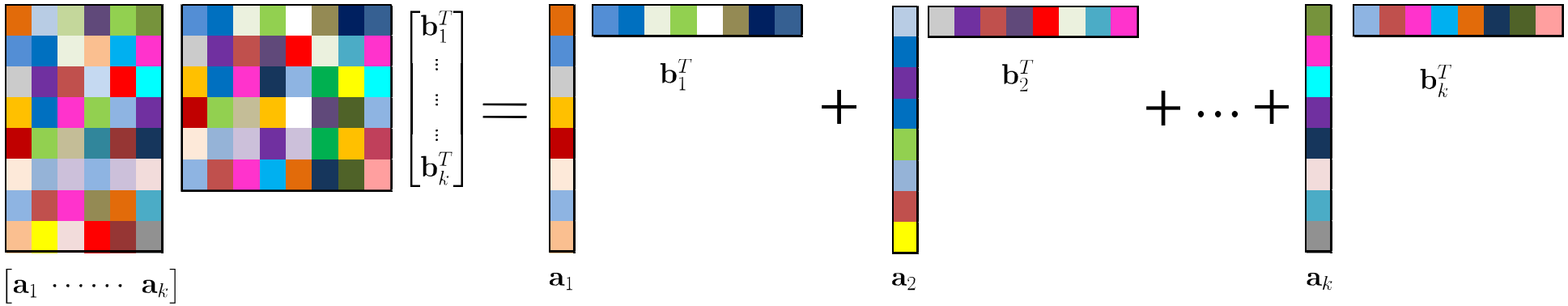
Matrix Product Representations

- outer-product representation: redefine $\mathbf{b}_i \in \mathbb{R}^k$ as the ith row of $\mathbf{B}$.

$$\mathbf{C} = \mathbf{A}(\mathbf{I})\mathbf{B} = \mathbf{A}\left(\sum_{i=1}^{k} \mathbf{e}_i\mathbf{e}_i^T\right)\mathbf{B} = \sum_{i=1}^{k} \mathbf{A}\mathbf{e}_i\mathbf{e}_i^T\mathbf{B}$$

Thus,

$$\mathbf{C} = \sum_{i=1}^{k} \mathbf{a}_i\mathbf{b}_i^T$$



Matrix Product Representations

- a matrix of the form $\mathbf{X} = \mathbf{a}\mathbf{b}^T$ for some $\mathbf{a}, \mathbf{b}$ is called a rank-one outer product. It can be verified that $\mathrm{rank}(\mathbf{X}) \leq 1$, and $\mathrm{rank}(\mathbf{X}) = 1$ iff $\mathbf{a} \neq \mathbf{0}, \mathbf{b} \neq \mathbf{0}$.
- the outer-product representation $\mathbf{C} = \mathbf{A}\mathbf{B} = \sum_{i=1}^{k} \mathbf{a}_i \mathbf{b}_i^T$ is a sum of k rank-one outer products
- does it mean that $\mathrm{rank}(\mathbf{C}) = k$?
  - $\mathrm{rank}(\mathbf{C}) \leq \sum_{i=1}^{k} \mathrm{rank}(\mathbf{a}_i \mathbf{b}_i^T) \leq k$ is true [2]
  - but the above equality is generally not attained; e.g., $k = 2$, $\mathbf{a}_1 = \mathbf{a}_2$, $\mathbf{b}_1 = -\mathbf{b}_2$ leads to $\mathbf{C} = \mathbf{0}$
  - $\mathrm{rank}(\mathbf{C}) = k$ only when $\mathbf{A}$ has full column rank *and* $\mathbf{B}$ has full row rank (in this case $\mathbf{C} = \mathbf{A}\mathbf{B}$ is sometimes called a (full-)rank factorization of $\mathbf{C}$) (proof as an exercise)
- a rank-k matrix can be written as the sum of k rank-1 matrices, but not fewer

[2] use the rank inequality $\mathrm{rank}(\mathbf{A} + \mathbf{B}) \leq \mathrm{rank}(\mathbf{A}) + \mathrm{rank}(\mathbf{B})$.

Block Matrix Manipulations

Sometimes it may be useful to manipulate matrices in a block form.

- let $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$. By partitioning

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}$$

  where $\mathbf{A}_1 \in \mathbb{R}^{m \times n_1}, \mathbf{A}_2 \in \mathbb{R}^{m \times n_2}, \mathbf{x}_1 \in \mathbb{R}^{n_1}, \mathbf{x}_2 \in \mathbb{R}^{n_2}$, with $n_1 + n_2 = n$, we can write

$$\mathbf{A}\mathbf{x} = \mathbf{A}_1\mathbf{x}_1 + \mathbf{A}_2\mathbf{x}_2$$

- similarly, by partitioning

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix},$$

  we can write

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} \mathbf{A}_{11}\mathbf{x}_1 + \mathbf{A}_{12}\mathbf{x}_2 \\ \mathbf{A}_{21}\mathbf{x}_1 + \mathbf{A}_{22}\mathbf{x}_2 \end{bmatrix}$$

Block Matrix Manipulations

- consider $\mathbf{AB}$. By an appropriate partitioning,

$$\mathbf{AB} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 \end{bmatrix} \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix} = \mathbf{A}_1\mathbf{B}_1 + \mathbf{A}_2\mathbf{B}_2$$

- similarly, by an appropriate partitioning,

$$\mathbf{AB} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix} \begin{bmatrix} \mathbf{B}_1 & \mathbf{B}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1\mathbf{B}_1 & \mathbf{A}_1\mathbf{B}_2 \\ \mathbf{A}_2\mathbf{B}_1 & \mathbf{A}_2\mathbf{B}_2 \end{bmatrix}$$

- we showcase two-block partitioning only, but the same manipulations apply to multi-block partitioning like

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \cdots & \mathbf{A}_{1q} \\ \vdots & & \vdots \\ \mathbf{A}_{p1} & \cdots & \mathbf{A}_{pq} \end{bmatrix}$$

Gram matrices

- Gram matrices (Gramian matrices): Consider m real vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$. The Gram matrix of the collection is the matrix $\mathbf{G} \in \mathbb{R}^{m \times m}$ with elements $g_{ij} = \mathbf{x}_i^T \mathbf{x}_j$. It can be expressed compactly in terms of the matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m]$, as

$$\mathbf{G} = \mathbf{X}^T \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_m^T \end{bmatrix} \begin{bmatrix} \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \end{bmatrix}.$$

  By construction, a Gram matrix is always symmetric, i.e., $g_{ij} = g_{ji}$ for all i, j.

- Assume that each vector $\mathbf{x}_i$ is normalized: $\|\mathbf{x}_i\|_2 = 1$. Then the coefficient g_{ij} can be expressed as

$$g_{ij} = \cos \theta_{ij},$$

  where θ_{ij} is the angle between the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. Thus g_{ij} is a measure of how similar $\mathbf{x}_i$ and $\mathbf{x}_j$ are. Like in text document classification and information retrieval, $\mathbf{x}_i$, $\mathbf{x}_j$ could represent the bag-of-words representation of the i-th and j-th document.

Schur complement

- the Schur complement: let

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix},$$

where $\mathbf{A} \in \mathbb{R}^{m\times m}$, $\mathbf{B} \in \mathbb{R}^{m\times n}$, $\mathbf{C} \in \mathbb{R}^{n\times m}$, and $\mathbf{D} \in^{n\times n}$.

If $\mathbf{A}$ is invertible, then the Schur complement of $\mathbf{A}$ of $\mathbf{M}$ is defined by

$$\mathbf{S}_A = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B};$$

If $\mathbf{D}$ is invertible, then the Schur complement of $\mathbf{D}$ of $\mathbf{M}$ is defined by

$$\mathbf{S}_D = \mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}.$$

- For square $\mathbf{A}$ and $\mathbf{D}$, we have $\mathrm{tr}(\mathbf{M}) = \mathrm{tr}(\mathbf{A}) + \mathrm{tr}(\mathbf{D})$. This result generalizes the trace formula for 2×2 matrices.

Schur complement

let

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix},$$

where $\mathbf{A} \in \mathbb{R}^{m \times m}$, $\mathbf{B} \in \mathbb{R}^{m \times n}$, $\mathbf{C} \in \mathbb{R}^{n \times m}$, and $\mathbf{D} \in^{n \times n}$.

- If $\mathbf{A}$ is invertible, then

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\mathbf{S}_A^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{S}_A^{-1} \\ -\mathbf{S}_A^{-1}\mathbf{C}\mathbf{A}^{-1} & \mathbf{S}_A^{-1} \end{bmatrix}$$

- If $\mathbf{D}$ is invertible, then

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{S}_D^{-1} & -\mathbf{S}_D^{-1}\mathbf{B}\mathbf{D}^{-1} \\ -\mathbf{D}^{-1}\mathbf{C}\mathbf{S}_D^{-1} & \mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{C}\mathbf{S}_D^{-1}\mathbf{B}\mathbf{D}^{-1} \end{bmatrix}$$

The results generalize the inversion formula for 2×2 matrices.

Schur complement

let

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix},$$

where $\mathbf{A} \in \mathbb{R}^{m \times m}$, $\mathbf{B} \in \mathbb{R}^{m \times n}$, $\mathbf{C} \in \mathbb{R}^{n \times m}$, and $\mathbf{D} \in^{n \times n}$.

- If $\mathbf{A}$ is invertible, then

$$\mathbf{M} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{C}\mathbf{A}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$

  we have

$$\det(\mathbf{M}) = \det(\mathbf{A}) \det(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}) = \det(\mathbf{A}) \det(\mathbf{S}_A).$$

- If $\mathbf{D}$ is invertible, then

$$\mathbf{M} = \begin{bmatrix} \mathbf{I} & \mathbf{B}\mathbf{D}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C} & \mathbf{0} \\ \mathbf{0} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{D}^{-1}\mathbf{C} & \mathbf{I} \end{bmatrix}$$

  we have

$$\det(\mathbf{M}) = \det(\mathbf{D}) \det(\mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}) = \det(\mathbf{D}) \det(\mathbf{S}_D).$$

The results generalize the determinant formula for 2×2 matrices.

Extension to $\mathbb{C}^n$

- all the concepts described above apply to the complex case
- we only need to replace every "$\mathbb{R}$" with "$\mathbb{C}$", and every "T" with "H"; e.g.,

$$\text{span}\{\mathbf{a}_1, \ldots, \mathbf{a}_n\} = \{\mathbf{y} \in \mathbb{C}^m \mid \mathbf{y} = \textstyle\sum_{i=1}^n \alpha_i \mathbf{a}_i,\ \boldsymbol{\alpha} \in \mathbb{C}^n\},$$

$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{y}^H \mathbf{x}$, $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^H \mathbf{x}}$, and so forth.

Extension to $\mathbb{R}^{m\times n}$

- the concepts also apply to the matrix case
  - e.g., we may write

$$\text{span}\{\mathbf{A}_1,\ldots,\mathbf{A}_k\} = \{\mathbf{Y}\in\mathbb{R}^{m\times n} \mid \mathbf{Y} = \textstyle\sum_{i=1}^k \alpha_i\mathbf{A}_i,\ \boldsymbol{\alpha}\in\mathbb{R}^k\}.$$

  - sometimes it is more convenient to *vectorize* $\mathbf{X}$ as a vector $\mathbf{x}\in\mathbb{R}^{mn}$, and use the same treatment as in the $\mathbb{R}^n$ case
  - inner product for $\mathbb{R}^{m\times n}$:

$$\langle\mathbf{X},\mathbf{Y}\rangle = \textstyle\sum_{i=1}^m\sum_{j=1}^n x_{ij}y_{ij} = \text{tr}(\mathbf{Y}^T\mathbf{X}),$$

  - the matrix version of the Euclidean norm is called the Frobenius norm:

$$\|\mathbf{X}\|_F = \sqrt{\textstyle\sum_{i=1}^m\sum_{j=1}^n |x_{ij}|^2} = \sqrt{\text{tr}(\mathbf{X}^T\mathbf{X})}$$

- extension to $\mathbb{C}^{m\times n}$ is just as straightforward as in that to $\mathbb{C}^n$

Complexity, Floating Point Operations (flops)

Complexities of Matrix Computations

- every vector/matrix operation such as $\mathbf{x} + \mathbf{y}$, $\mathbf{y}^T\mathbf{x}$, $\mathbf{A}\mathbf{x}$, ... incurs computational costs, and they cost more as the vector and matrix sizes get bigger

- we typically look at floating point (arithmetic) operations (flops), such as add, subtract, multiply, and divide

Complexities of Matrix Computations

- flop: one flop means one floating point operation, i.e., one addition, subtraction, multiplication, or division of two floating-point numbers.

- to estimate complexity of an algorithm: express number of flops as a (polynomial) function of the problem dimensions, and simplify by keeping only the leading terms

- flop counts of some standard vector/matrix operations:

  for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times p}$,

  - $\mathbf{x} + \mathbf{y}$: n adds, so n flops
  - $\mathbf{y}^T \mathbf{x}$: n multiplies and $n - 1$ adds, so $2n - 1$ flops
  - $\mathbf{A}\mathbf{x}$: m inner products, so $m(2n - 1)$ flops
  - $\mathbf{A}\mathbf{B}$: do "$\mathbf{A}\mathbf{x}$" above p times, so $pm(2n - 1)$ flops

Complexities of Matrix Computations

- we are often interested in the *order* of the complexity
- big O notation: given two functions $f(n), g(n)$, the notation

$$f(n) = \mathcal{O}(g(n))$$

  means that there exists a constant $C > 0$ and n_0 such that $|f(n)| \leq C|g(n)|$ for all $n \geq n_0$.

- big O complexities of standard vector/matrix operations:
  - $\mathbf{x} + \mathbf{y}$: $\mathcal{O}(n)$ flops
  - $\mathbf{y}^T\mathbf{x}$: $\mathcal{O}(n)$ flops
  - $\mathbf{A}\mathbf{x}$: $\mathcal{O}(mn)$ flops
  - $\mathbf{A}\mathbf{B}$: $\mathcal{O}(mnp)$ flops
  - (we'll learn it later) solve $\mathbf{y} = \mathbf{A}\mathbf{x}$ for $\mathbf{x}$, with $\mathbf{A} \in \mathbb{R}^{n\times n}$: $\mathcal{O}(n^3)$ flops

Complexities of Matrix Computations

- big O complexities are commonly used, although we should be careful sometimes
- example: suppose you have an algorithm whose exact flop count is

$$f(n) = 3n^3 + 8n^2 + 2n + 1234.$$

  - $\mathcal{O}(n^3)$ flops
  - big O makes sense for large n; n^3 dominates as n is large
  - but be careful: for small n, it's 1234 that consumes more

- example: suppose you have two algorithms for the same problem. Their exact flop counts are

$$f_1(n) = n^3, \quad f_2(n) = \frac{1}{2}n^3.$$

  - their big O complexities are the same: $\mathcal{O}(n^3)$
  - but two times faster is two times faster!

Complexities of Matrix Computations

- example: suppose our algorithm deals with complex vector and matrix operations. Define one flop as one real flop.
  - one complex add = 2 real adds = 2 flops
  - one complex multiply = 4 real multiplies + 2 real adds = 6 flops
  - ⋮

  When we report big O complexity, the scaling factors above are not seen

Exercise: Count the Complexity of Gram-Schmidt

- recall the Gram-Schmidt procedure recursively computes

$$\tilde{\mathbf{q}}_i = \mathbf{a}_i - \sum_{j=1}^{i-1} (\mathbf{q}_j^T \mathbf{a}_i)\mathbf{q}_j, \quad \mathbf{q}_i = \tilde{\mathbf{q}}_i / \|\tilde{\mathbf{q}}_i\|_2, \quad i = 1, \ldots, n.$$

- consider iteration i.
  - every $\mathbf{q}_j^T \mathbf{a}_i$, $j = 1, \ldots, i = 1$, takes $\mathcal{O}(m)$
  - then, computing $\tilde{\mathbf{q}}_i = \mathbf{a}_i - \sum_{j=1}^{i-1} (\mathbf{q}_j^T \mathbf{a}_i)\mathbf{q}_j$ is almost the same as the operation "$\mathbf{Ax}$"; it takes $\mathcal{O}(mi)$
  - $\tilde{\mathbf{q}}_i = \tilde{\mathbf{q}}_i / \|\tilde{\mathbf{q}}_i\|_2$ requires $\mathcal{O}(m)$ (one divide, one $\sqrt{\cdot}$, one inner product $\tilde{\mathbf{q}}_i^T \tilde{\mathbf{q}}_i$)
  - total complexity for iteration i: $(i-1) \times \mathcal{O}(m) + \mathcal{O}(mi) + \mathcal{O}(m) = \mathcal{O}(mi)$
- total complexity of the whole algorithm:

$$\mathcal{O}(m \sum_{i=1}^{n} i) = \mathcal{O}(m \frac{n(n+1)}{2}) = \mathcal{O}(mn^2)$$

Complexities of Matrix Computations

- Discussion: flop counts do not always translate into the actual efficiency of the execution of an algorithm, say, in terms of actual running time.

- things like pipelining, FPGA, parallel computing (multiple GPUs, multiple servers, cloud computing), etc., can make the story different.

- flop counts also ignore memory usage and other overheads...

- that said, we need at least a crude measure of how computationally costly an algorithm would be, and counting the flops serves that purpose.

How to Save Computations

- computational complexities depend much on how we design and write an algorithm
- generally, it is about
  - top-down, analysis-guided, designs: often seen in class, often look elegant
  - street-smart, possibly bottom-up, tricks: usually *not* taught much in class, also not commonplace in papers (unless you download and read somebody's code), subtly depends on your problem at hand, but a bunch of small differences can make a big difference, say in actual running time
- here we give several, but by no means all, tips for saving computations

How to Save Computations

- apply matrix operations wisely

- example: try this on MATLAB

```
>> A=randn(5000,2); B=randn(2,10000); C=randn(10000,10000);
>>
>> tic; D= A*B*C; toc
Elapsed time is 12.238567 seconds.
>> tic; D= (A*B)*C; toc      % ask MATLAB to do AB first
Elapsed time is 12.640961 seconds.
>> tic; D= A*(B*C); toc      % ask MATLAB to do BC first
Elapsed time is 0.222270 seconds.
```

How to Save Computations

- let us analyze the complexities in the last example
  - $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{B} \in \mathbb{R}^{n\times p}, \mathbf{C} \in \mathbb{R}^{p\times p}$, with $n \ll \min\{m, p\}$. We want to compute $\mathbf{D} = \mathbf{ABC}$.
  - if we compute $\mathbf{AB}$ first, and then $\mathbf{D} = (\mathbf{AB})\mathbf{C}$, the flop count will be

$$\mathcal{O}(mnp) + \mathcal{O}(mp^2) = \mathcal{O}(m(n+p)p) \approx \mathcal{O}(mp^2)$$

  - if we compute $\mathbf{BC}$ first, and then $\mathbf{D} = \mathbf{A}(\mathbf{BC})$, the flop count will be

$$\mathcal{O}(np^2) + \mathcal{O}(mnp) = \mathcal{O}((m+p)np).$$

  - the 2nd option is preferable if n is much smaller than m, p

How to Save Computations

- use structures, if available

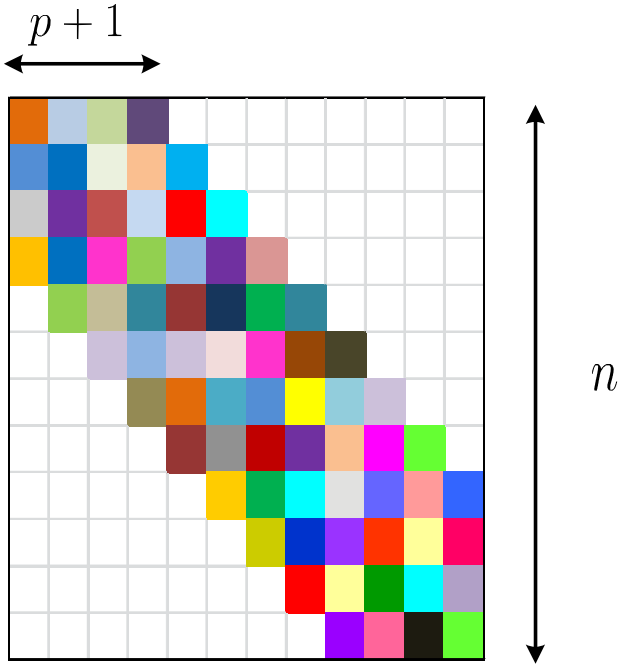
- example: let $\mathbf{A} \in \mathbb{R}^{n \times n}$ and suppose that

  $$a_{ij} = 0 \quad \text{for all } i, j \text{ such that } |i - j| > p,$$

  for some integer $p > 0$.

  – such a structured $\mathbf{A}$ is a band matrix

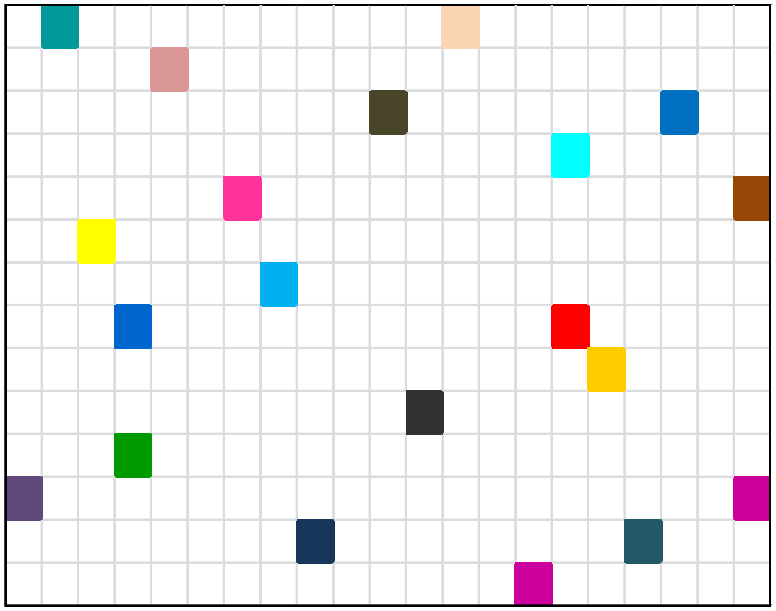
  – if we don't use structures, computing $\mathbf{Ax}$ requires $\mathcal{O}(n^2)$



  – if we use the band diagonal structures, we can compute $\mathbf{Ax}$ with $\mathcal{O}(pn)$

How to Save Computations

- use sparsity, if available
- a vector or matrix is said to be sparse if it contains many zero elements
  - we assume unstructured sparsity



- computations involving sparse matrices are important tasks to be investigated

How to Save Computations

- let $\mathrm{nnz}(\mathbf{x})$ denote the number of nonzero elements of a vector $\mathbf{x}$; the same notation applies to matrices
- flop counts: for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times p}$,
  - $\mathbf{x} + \mathbf{y}$: from 0 and $\min\{\mathrm{nnz}(\mathbf{x}), \mathrm{nnz}(\mathbf{y})\}$ flops $\Longrightarrow \mathcal{O}(\min\{\mathrm{nnz}(\mathbf{x}), \mathrm{nnz}(\mathbf{y})\})$
  - $\mathbf{y}^T\mathbf{x}$: from 0 to $2\min\{\mathrm{nnz}(\mathbf{x}), \mathrm{nnz}(\mathbf{y})\}$ flops $\Longrightarrow \mathcal{O}(\min\{\mathrm{nnz}(\mathbf{x}), \mathrm{nnz}(\mathbf{y})\})$
  - $\mathbf{A}\mathbf{x}$, $\mathbf{x}$ being dense: from $\mathrm{nnz}(\mathbf{A})$ to $2\mathrm{nnz}(\mathbf{A})$ flops $\Longrightarrow \mathcal{O}(\mathrm{nnz}(\mathbf{A}))$
  - $\mathbf{A}\mathbf{B}$: no simple expression for the flops, but at most $2\min\{\mathrm{nnz}(\mathbf{A})p, \mathrm{nnz}(\mathbf{B})m\}$ flops $\Longrightarrow \mathcal{O}(\min\{\mathrm{nnz}(\mathbf{A})p, \mathrm{nnz}(\mathbf{B})m\})$
- reference: S. Boyd and L. Vandenberghe, *Introduction to Applied Linear Algebra – Vectors, Matrices, and Least Squares*, 2018. Available online at https://web.stanford.edu/~boyd/vmls/vmls.pdf.