Randomized Numerical Linear Algebra

Introduction and basis

Augustin Parret-Fréaud

February 13–14, March 6–7, 2025

Safran SA | DSTI | Safran Tech | Digital Sciences and Technologies
augustin.parret-freaud@safrangroup.com

Slides and materials available at

https://git.sophia.mines-paristech.fr/parret-freaud/hpc-ia-rnla

Table of contents

Introduction

Numerical linear algebra and probabilistic techniques

Major classes of problems for numerical linear algebra:

- solution of dense and sparse linear systems
- orthogonalization, least-squares, Tikhonov regularization
- eigen analysis
- singularvalue decomposition

Numerical linear algebra and probabilistic techniques

Major classes of problems for numerical linear algebra:

- solution of dense and sparse linear systems
- orthogonalization, least-squares, Tikhonov regularization
- eigen analysis
- singularvalue decomposition

Probabilistic techniques that have joined the mainstream of NLA over the last decades allow:

- **accelerating everydays computation** on small/medium usecases
- **enabling (extreme) large-scale computation** far beyond classical methods

Today's challenges

- Multiscale aspect of modern computing architecture (massively distributed, massively multi-core CPUs, GPUs) minimizing data movement
- Extremely high capacity storage ($\sim$PB)
- Various data access : out of core, distributed data, streaming

Various usecases and access patterns

- **Streaming** (single view) computation: **A** to large to be stored, but given as a sequence of simple linear updates

$$\mathbf{A} = \mathbf{H}_1 + \mathbf{H}_2 + \mathbf{H}_3 + \dots$$

  where every $\mathbf{H}_i$ is discarded after processing
- **Dense matrices** stored in RAM: memory latency far beyond CPU's ability, needs for improved blocked hierarchical algorithms
- **Large sparse matrices**: use as much application of **A** matrix to block vectors, sample to maintain sparsity
- **Expensive evaluation** of matrices: some matrix-free context, etc...

Probabilistic algorithms: elements of history

- 1940's: Monte Carlo algorithms (Ulam and von Neumann)
- 1980's: first randomized algorithms in NLA, largest eigenvalue estimation for PSD transform (1983)
- 1990's: trace estimation of large PSD matrix, pivoting optimized Gaussian elimination (1995)
- 2000's: computation on streaming data, low-rank approximation and least-squares problems (2005), RNLA starts to outperform standard NLA
- 2010's: stochastic gradient descent for ML (2010), begin to incorporate some RNLA methods into standard software librairies (2017)
- 2020's: randBLAS, randLAPACK ???

Progress in algorithms complexity through the decades

- Fast Fourier Transform (1960's): $n^2 \longrightarrow n \log n$
- Multigrid (1970's): $n^{4/3} \log n \longrightarrow n$
- Fast Multipole (1980's): $n^2 \longrightarrow n$
- Sparse Grids (1990's): $n^d \longrightarrow n(\log n)^{d-1}$
- $\mathcal{H}$ matrices (2000's): $n^3 \longrightarrow k^2 n(\log n)^2$
- Randomized matrix algorithms (2010's): $n^3 \longrightarrow n^2 \log k$
- ??? (2020's): ??? $\longrightarrow$???

From D. Keyes' talk at SIAM CSE21

Low-rank matrices

Many formally dense matrices have exploitable low-rank structure in ”most” their off-diagonal blocks (if well ordered):

- **covariances** in statistics
- **integral equations** with displacement kernels
- **Schur complements** within discretizations of PDEs
- **Hessians** from PDE-constrained optimization
- **nonlocal operators** from fractional differential equations
- **radial basis functions** from unstructured meshing
- **kernel matrices** from machine learning applications
- …

From D. Keyes’ talk at SIAM CSE21

Low rank approximation

Recall on some linear algebra basis

Vectors: $\mathbf{v} = [v_i]_i$

- 2 and p-norms: $\|\mathbf{v}\|_2 = \sqrt{\mathbf{v}^*\mathbf{v}}$ $\|\mathbf{v}\|_p = \left(\sum_k v_k^p\right)^{1/p}$
- Cauchy-Schwartz inequality: $(\mathbf{u}, \mathbf{v}) = \mathbf{u}^*\mathbf{v} \leq \|\mathbf{u}\|_2 \|\mathbf{v}\|_2$

Matrix norms: $\|\mathbf{AB}\| \leq \|\mathbf{A}\| \|\mathbf{B}\|$

- Frobenius norm: $\|\mathbf{A}\|_F = \sqrt{Tr(\mathbf{A}^*\mathbf{A})} = \sqrt{\sum_{i,j} |a_{i,j}|}$
- Spectral norm: $\|\mathbf{A}\|_2 = \max_{\|\mathbf{u}\|_2=1}(\|\mathbf{Au}\|_2)$

Miscellaneous:

- $\mathcal{R}(\mathbf{A}^*) \oplus^{\perp} \mathcal{K}(\mathbf{A}) = \mathbb{R}^m$
- $\mathbf{A}$ orthonormal verifies: $\mathbf{A}^*\mathbf{A} = \mathbf{AA}^* = \mathbf{I}$
- $\mathbf{P}$ projector verifies: $\mathbf{P}^2 = \mathbf{P}$, $\mathbf{P}^*$ and $(\mathbf{I} - \mathbf{P})$ projectors
- For $\mathbf{U}$ orthonormal: $\mathbf{P} = \mathbf{UU}^*$ denotes the orthonormal projection on $span(\mathbf{U})$

Low-rank approximation

From a given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, we seek to compute an approximation $\mathbf{A}_{approx}$ of rank $k < \min(m, n)$ in the "general" form:

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times n}{\mathbf{A}_{approx}} = \underset{m \times k}{\mathbf{E}} \; \underset{k \times n}{\mathbf{F}}$$

Some type of factored form:

- **SVD**: optimal as one reach minimal error $\|\mathbf{A} - \mathbf{A}_{approx}\|$ whatever Frobenius or Spectral norm is used
- **QR**: much faster (but less optimal)
- **ID, CUR**: may preserve specific strucure of **A**

A has to be stored in RAM to avoid catastrophic performance, of course ! But we'll se later that RNLA offers some interesting alternatives when it doesn't fit.

Low-rank approximation

From a given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, we seek to compute an approximation $\mathbf{A}_{approx}$ of rank $k < \min(m, n)$ in the "general" form:

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times n}{\mathbf{A}_{approx}} = \underset{m \times k}{\mathbf{E}} \; \underset{k \times n}{\mathbf{F}}$$

Some type of factored form:

- **SVD**: optimal as one reach minimal error $\|\mathbf{A} - \mathbf{A}_{approx}\|$ whatever Frobenius or Spectral norm is used
- **QR**: much faster (but less optimal)
- **ID, CUR**: may preserve specific strucure of **A**

Efficiency:

- from the error point of view: SVD $<$ QR, ID $<$ CUR
- from the storage point of view (given **A** dense): ID $<$ SVD $<$ CUR
- from the storage point of view (given **A** sparse): ID, CUR $<$ SVD

Low-rank approximation

From a given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, we seek to compute an approximation $\mathbf{A}_{approx}$ of rank $k < \min(m, n)$ in the "general" form:

$$\underset{m \times n}{\mathbf{A}} \approx \mathbf{A}_{approx} = \underset{m \times k}{\mathbf{E}} \; \underset{k \times n}{\mathbf{F}}$$

Fixed rank approximation problem: given k, find $\mathbf{E} \in \mathbb{R}^{m \times k}$ and $\mathbf{F} \in \mathbb{R}^{k \times n}$ such that

$$\mathbf{A} \approx \mathbf{EF}$$

Fixed precision approximation problem: given ε, find $\mathbf{E} \in \mathbb{R}^{m \times k}$ and $\mathbf{F} \in \mathbb{R}^{k \times n}$ such that

$$\|\mathbf{A} - \mathbf{EF}\| < \varepsilon$$

Singular Values Decomposition (SVD)

From a given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, denoting $l = \min(m, n)$, the "economic" SVD of A is given by:

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times l}{\mathbf{U}} \; \underset{l \times l}{\mathbf{\Sigma}} \; \underset{l \times n}{\mathbf{V}^*} \qquad \mathbf{A} = \sum_{j=1}^{l} \sigma_j \mathbf{u}_j \mathbf{v}_j^*$$

with:

$$\mathbf{U} = [\mathbf{u}_1 \; \mathbf{u}_2 \; \ldots \; \mathbf{u}_l] \qquad \mathbf{V} = [\mathbf{v}_1 \; \mathbf{v}_2 \; \ldots \; \mathbf{v}_l] \qquad \mathbf{\Sigma} = diag\,(\sigma_1 \; \sigma_2 \; \ldots \; \sigma_l)$$

- $\mathbf{u}_i$ (resp. $\mathbf{v}_i$) are the left (resp. right) singular vector of $\mathbf{A}$
- σ_i are the singular values of $\mathbf{A}$

Singular Values Decomposition (SVD)

From a given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, denoting $l = \min(m, n)$, the "economic" SVD of A is given by:

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times l}{\mathbf{U}} \; \underset{l \times l}{\mathbf{\Sigma}} \; \underset{l \times n}{\mathbf{V}^*} \qquad \mathbf{A} = \sum_{j=1}^{l} \sigma_j \mathbf{u}_j \mathbf{v}_j^*$$

with:

$$\mathbf{U} = [\mathbf{u}_1 \; \mathbf{u}_2 \; \dots \; \mathbf{u}_l] \qquad \mathbf{V} = [\mathbf{v}_1 \; \mathbf{v}_2 \; \dots \; \mathbf{v}_l] \qquad \mathbf{\Sigma} = diag\,(\sigma_1 \; \sigma_2 \; \dots \; \sigma_l)$$

- $\mathbf{u}_i$ (resp. $\mathbf{v}_i$) are the left (resp. right) singular vector of $\mathbf{A}$
- σ_i are the singular values of $\mathbf{A}$

For $\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_l = 0$, one has:

- $rk(\mathbf{A}) = rk(\mathbf{A}^*) = r$
- $\mathcal{K}(\mathbf{A}) = span(\mathbf{v}_{r+1}, \dots, \mathbf{v}_n)$
- $\mathcal{R}(\mathbf{A}) = span(\mathbf{u}_1, \dots, \mathbf{u}_r)$

SVD and low rank approximation

A may also be expressed as:

Truncated SVD $\mathbf{A}_k$ of **A**:

$$\mathbf{A} \approx \mathbf{A}_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^* = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^* = \mathbf{U}(:, 1:k)\mathbf{\Sigma}(1:k, 1:k)\mathbf{V}(:, 1:k)^*$$

Eckart-Young theorem
Truncated SVD $\mathbf{A}_k$ ($1 \leq k \leq r$) is the optimal approximation of A:

$$\|\mathbf{A} - \mathbf{A}_k\| = \inf \{\|\mathbf{A} - \mathbf{B}\|, rk(\mathbf{B}) = k\}$$

with the following estimates:

$$\|\mathbf{A} - \mathbf{A}_k\|_2 = \sigma_{k+1} \qquad \|\mathbf{A} - \mathbf{A}_k\|_F = \left(\sum_{j=k+1}^{r} \sigma_j^2 \right)^{1/2}$$

Column pivoted QR (CPQR)

From a given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, denoting $r = \min(m, n)$, the column pivoted QR-factorization of A is given by:

$$\underset{m \times n}{\mathbf{A}} \quad \underset{n \times n}{\mathbf{P}} = \underset{m \times r}{\mathbf{Q}} \quad \underset{r \times n}{\mathbf{S}}$$

- Usually iteratively computed via Gram-Schmidt or Householder QR through rank-1 opdates to the matrix.
- With column pivoting, may be halted after k steps to produce a rank-k approximation.

CPQR and low rank approximation

$$\underset{m\times n}{\mathbf{A}}\ \underset{n\times n}{\mathbf{P}} = \underset{m\times r}{\mathbf{Q}}\ \underset{r\times n}{\mathbf{S}}$$

Halting the factorization after k steps and splitting $\mathbf{Q}$ into $\mathbf{Q}_1$ $(m \times k)$ and $\mathbf{Q}_2$ $(m \times (n-k))$:

$$\mathbf{A} = [\mathbf{Q}_1\ \mathbf{Q}_2] \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{O} & \mathbf{S}_{22} \end{bmatrix} \mathbf{P}^* = \mathbf{Q}_1[\mathbf{S}_{11}\ \mathbf{S}_{12}]\mathbf{P}^* + \mathbf{Q}_2[\mathbf{O}\ \mathbf{S}_{22}]\mathbf{P}^*$$

Approximation error given by:

$$\|\mathbf{A} - \mathbf{A}_{approx}\| = \|\mathbf{Q}_2[\mathbf{O}\ \mathbf{S}_{22}]\mathbf{P}^*\| = \|\mathbf{S}_{22}\|$$

- Much faster to compute rank-k approximation than to compute k-SVD
- ... but approximation error is larger

Krylov methods for low-rank approximation

Coming from so called Krylov spaces, at the basis of the well known iterative solver family (Conjugage Gradient, GMRES and derivatives, Orthodir, Orthomin...)

Idea: pick a starting vector $\mathbf{r}$ (that may be random), restrict the matrix A to the k-dimensional Krylov subspace

$$span\left(\mathbf{r}, \mathbf{A}\mathbf{r}, \mathbf{A}^2\mathbf{r}, \ldots, \mathbf{A}^{k-1}\mathbf{r}\right)$$

and compute the eigendecomposition of the resulting matrix.

- Very simple access to $\mathbf{A}$
- Particulary efficient when $\mathbf{A}$ can be rapidly applied to a given vector (sparse, ...)
- ... but $\mathbf{A}$ has to be accessed $O(k)$ time for a rank-k approximation
- May have numerical stability issues

Randomized methods for low rank approximation

Scope: when **A** is large and target rank k much smaller ($k \ll \min(m, n)$)

- all interactions with **A** are of matrix-matrix multiplication type, very fast on modern CPU and GPUs (BLAS3,...)
- other types of operation involves small matrices of roughly k rows / columns
- allow to goes from asymptotic complexity $O(mn \min(m, n))$ to $O(mnk)$ (even to $O(mn \log(k))$

Use randomisation to cast as much of the computation as possible in terms of highly efficient matrix-matrix multiplications.

In contrast, classical NLA algorithms often requires random access to the data or sequential matrix-vector multiplication.

Randomized low-rank factorizations

Approximate low-rank factorization: two stage approach

$$\underset{m\times n}{\mathbf{A}} \approx \mathbf{A}_{approx} = \underset{m\times k}{\mathbf{E}}\ \underset{k\times n}{\mathbf{F}}$$

1. **Approximate range finder stage**
   Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix approximating the column space of $\mathbf{A}$ such that:

   $$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

   This is the step where randomized process is used.

Approximate low-rank factorization: two stage approach

$$\underset{m\times n}{\mathbf{A}} \approx \underset{m\times n}{\mathbf{A}_{approx}} = \underset{m\times k}{\mathbf{E}}\ \underset{k\times n}{\mathbf{F}}$$

1. **Approximate range finder stage**
   Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix approximating the column space of $\mathbf{A}$ such that:

   $$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

   This is the step where randomized process is used.
2. **Factorisation stage**
   Use classical deterministic methods (SVD, QR, ID...) on $\mathbf{A}_{approx}$
   1. Form a smaller "working" matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$
   2. Compute factorization on $\mathbf{B} \longrightarrow \mathbf{B} = \hat{\mathbf{E}}\mathbf{F}$
   3. Form $\mathbf{E} = \mathbf{Q}\hat{\mathbf{E}}$

Approximate low-rank factorization: two stage approach

$$\underset{m\times n}{\mathbf{A}} \approx \mathbf{A}_{approx} = \underset{m\times k}{\mathbf{E}}\ \underset{k\times n}{\mathbf{F}}$$

1. **Approximate range finder stage**
2. **Factorisation stage**

   Use classical deterministic methods (SVD, QR, ID...) on $\mathbf{A}_{approx}$

   1 Form a smaller "working" matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$
   2 Compute factorization on $\mathbf{B} \longrightarrow \mathbf{B} = \hat{\mathbf{E}}\mathbf{F}$
   3 Form $\mathbf{E} = \mathbf{Q}\hat{\mathbf{E}}$

   Exact computation (up to floating point arithmetic)

$$\mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^*\mathbf{A} = \mathbf{Q}\mathbf{B} = \mathbf{Q}\hat{\mathbf{E}}\mathbf{F} = \mathbf{E}\mathbf{F}$$

   All approximation errors are coming from previous stage (if factorization not truncated)

$$\|\mathbf{A} - \mathbf{E}\mathbf{F}\| = \|\mathbf{A} - \mathbf{A}_{approx}\| \leq \varepsilon$$

Approximate range finder stage

Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix such that:

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

$\mathbf{Q}$ verifies:

- $\mathbf{Q}^*\mathbf{Q} = \mathbf{I}$
- $\mathbf{P}_\mathbf{Q} = \mathbf{Q}\mathbf{Q}^*$ is a projector onto $\mathcal{R}(\mathbf{Q})$
  - $\mathbf{P}_\mathbf{Q}^2 = \mathbf{Q}\mathbf{Q}^*\mathbf{Q}\mathbf{Q}^* = \mathbf{P}_\mathbf{Q}$
  - For $\mathbf{v} \in \mathcal{R}(\mathbf{Q}),\ \exists \boldsymbol{\alpha}$ such that $\mathbf{v} = \mathbf{Q}\boldsymbol{\alpha}$, then $\mathbf{P}_\mathbf{Q}\mathbf{v} = \mathbf{Q}\mathbf{Q}^*\mathbf{v} = \mathbf{Q}\mathbf{Q}^*\mathbf{Q}\boldsymbol{\alpha} = \mathbf{Q}\boldsymbol{\alpha} = \mathbf{v}$

Approximate range finder stage

Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix such that:

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

$\mathbf{Q}$ verifies:

- $\mathbf{Q}^*\mathbf{Q} = \mathbf{I}$
- $\mathbf{P}_\mathbf{Q} = \mathbf{Q}\mathbf{Q}^*$ is a projector onto $\mathcal{R}(\mathbf{Q})$
- The following properties are equivalent:

$$\mathcal{R}(\mathbf{A}) \subseteq \mathcal{R}(\mathbf{Q}) \iff \mathbf{A} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

  - $\implies$: $\exists \mathbf{S}$ such that $\mathbf{A} = \mathbf{Q}\mathbf{S} \implies \mathbf{Q}\mathbf{Q}^*\mathbf{A} = \mathbf{Q}\mathbf{Q}^*\mathbf{Q}\mathbf{S} = \mathbf{Q}\mathbf{S} = \mathbf{A}$
  - $\impliedby$: $\mathbf{A} = \mathbf{Q}\mathbf{Q}^*\mathbf{A} = \mathbf{Q}(\mathbf{Q}^*\mathbf{A}) \implies \mathcal{R}(\mathbf{A}) \subseteq \mathcal{R}(\mathbf{Q})$

Approximate range finder stage

Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix such that:

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

$\mathbf{Q}$ verifies:

- $\mathbf{Q}^*\mathbf{Q} = \mathbf{I}$
- $\mathbf{P}_\mathbf{Q} = \mathbf{Q}\mathbf{Q}^*$ is a projector onto $\mathcal{R}(\mathbf{Q})$
- The following properties are equivalent:

$$\mathcal{R}(\mathbf{A}) \subseteq \mathcal{R}(\mathbf{Q}) \iff \mathbf{A} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

If we succeds in building a matrix $\mathbf{Q}$ such that $\mathcal{R}(\mathbf{Q})$ is "sufficiently close" to $\mathcal{R}(\mathbf{A})$, then $\mathbf{Q}\mathbf{Q}^*\mathbf{A} \approx \mathbf{A}$.

How to efficiently build $\mathbf{Q}$? Surprizingly, randomization can help !

Building Q: first step toward randomization

Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix such that:

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^{*}\mathbf{A}$$

A first way to build $\mathbf{Q}$ through randomization is:

1. Draw k random vector $\{\mathbf{g}_i\}_{i=1\dots k}$ from a Gaussian distribution
2. For each $\mathbf{g}_i$, evaluate $\mathbf{y}_i = \mathbf{A}\mathbf{g}_i$ (inermediate basis)
3. Orthonormalize $\{\mathbf{y}_i\}_{i=1\dots k}$ (typically using Gram-Schmidt algorithm) to obtain the orthonormal basis $\{\mathbf{q}_i\}_{i=1\dots k}$ verifying:

$$\mathcal{R}(\mathbf{q}_1 \dots \mathbf{q}_k) = \mathcal{R}(\mathbf{y}_1 \dots \mathbf{y}_k)$$

$\{\mathbf{g}_i\}_{i=1\dots k}$ will be used as our basis and $\mathbf{Q} = [\mathbf{q}_1 \dots \mathbf{q}_k]$

Building Q: matrix version

Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix such that:

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^{*}\mathbf{A}$$

Building $\mathbf{Q}$ through randomization (matrix version):

1. Draw an $(n \times k)$ Gaussian random matrix $\mathbf{G}$
2. Sample $\mathbf{A}$ through evaluating $\mathbf{Y} = \mathbf{AG}$
3. Orthonormalize $\mathbf{Y}$ to obtain an orthonormal matrix $\mathbf{Q}$ (typically using QR factorization): $\mathbf{Y} = \mathbf{QR}$

Columns of $\mathbf{Q}$ will span our basis

If $rk(\mathbf{A}) = k$, $\mathcal{R}(\mathbf{Q}) = \mathcal{R}(\mathbf{A})$ with probability 1. Has been proven... but that's in theory !

Randomized SVD

Basic RSVD algorithm

Associating previous range finder approach with the use of an SVD factorization gives a first version of Randomized SVD (RSVD).

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank parameter k
Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1. Draw a Gaussian random matrix $\mathbf{G}$ $(n \times k)$
2. Sample input matrix $\mathbf{A}$: $\mathbf{Y} = \mathbf{AG}$ $(n \times k)$
3. Orthonormalize samples $\mathbf{Y}$: $\mathbf{Y} = \mathbf{QR} \longrightarrow \mathbf{Q}$ $(n \times k)$
4. Form the (small) matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$ $(k \times n)$
5. Factorize $\mathbf{B}$ with SVD: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^* \longrightarrow \hat{\mathbf{U}}$ $(k \times k)$, $\mathbf{\Sigma}$ $(k \times k)$, $\mathbf{V}$ $(k \times n)$
6. Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times k)$

Basic RSVD algorithm

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank parameter k

Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1. Draw a Gaussian random matrix $\mathbf{G}$ $(n \times k)$
2. Sample input matrix $\mathbf{A}$: $\mathbf{Y} = \mathbf{AG}$ $(n \times k)$
3. Orthonormalize samples $\mathbf{Y}$: $\mathbf{Y} = \mathbf{QR} \longrightarrow \mathbf{Q}$ $(n \times k)$
4. Form the (small) matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ $(k \times n)$
5. Factorize $\mathbf{B}$ with SVD: $\mathbf{B} = \hat{\mathbf{U}} \mathbf{\Sigma} \mathbf{V}^* \longrightarrow \hat{\mathbf{U}}$ $(k \times k)$, $\mathbf{\Sigma}$ $(k \times k)$, $\mathbf{V}$ $(k \times n)$
6. Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times k)$

Remarks:

- Only 2 interactions with matrix $\mathbf{A}$ through matrix-matrix multiplications to compute $\mathbf{Q}$ ($\mathbf{Y} = \mathbf{AG}$) and $\mathbf{B}$ ($\mathbf{B} = \mathbf{Q}^* \mathbf{A}$)
- Only through matrix-matrix multiplications, which are very efficient on modern CPU and GPU
- Other manipulation steps involves far smaller matrices with (at least) one dimension equal to k

Back to range finder problem

Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix such that:

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

1. Draw an $(n \times k)$ Gaussian random matrix $\mathbf{G}$
2. Sample $\mathbf{A}$ through evaluating $\mathbf{Y} = \mathbf{AG}$
3. Orthonormalize $\mathbf{Y}$ to obtain an orthonormal matrix $\mathbf{Q}$: $\mathbf{Y} = \mathbf{QR}$

If $rk(\mathbf{A}) = k$, $\mathcal{R}(\mathbf{Q}) = \mathcal{R}(\mathbf{A})$ with probability 1. Has been proven... but that's in theory !

Unfortunately, $\mathbf{A}$ is rarely of exact rank k we can afford, thus, extra sigular values above k may shift $\mathcal{R}(\mathbf{Q})$ away from $\mathcal{R}(\mathbf{A})$.

Solution: simply oversample at the first step by drawing a $(n \times (k + p))$ Gaussian matrix $\mathbf{G}$ with p additionnal vectors $\mathbf{g}_k$.

RSVD algorithm (with oversampling)

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling param. p

Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1. Draw a Gaussian random matrix $\mathbf{G}$ $(n \times (k+p))$
2. Sample input matrix $\mathbf{A}$: $\mathbf{Y} = \mathbf{AG}$ $(n \times (k+p))$
3. Orthonormalize samples $\mathbf{Y}$: $\mathbf{Y} = \mathbf{QR} \longrightarrow \mathbf{Q}$ $(n \times (k+p))$
4. Form the (small) matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ $((k+p) \times n)$
5. Factorize $\mathbf{B}$ with SVD:
   $\mathbf{B} = \hat{\mathbf{U}} \mathbf{\Sigma} \mathbf{V}^* \longrightarrow \hat{\mathbf{U}}$ $((k+p) \times (k+p))$, $\mathbf{\Sigma}$ $((k+p) \times k)$, $\mathbf{V}$ $((k+p) \times n)$
6. Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times (k+p))$
7. Truncate the matrices $\mathbf{U}$, $\mathbf{V}$, $\mathbf{\Sigma}$ to k: $\mathbf{U} = \mathbf{U}(:, 1:k)$, $\mathbf{V} = \mathbf{V}(:, 1:k)$, $\mathbf{\Sigma} = \mathbf{\Sigma}(1:k, 1:k)$

RSVD algorithm (with oversampling)

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling param. p

Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1. Draw a Gaussian random matrix $\mathbf{G}$ $(n \times (k+p))$
2. Sample input matrix $\mathbf{A}$: $\mathbf{Y} = \mathbf{AG}$ $(n \times (k+p))$
3. Orthonormalize samples $\mathbf{Y}$: $\mathbf{Y} = \mathbf{QR} \longrightarrow \mathbf{Q}$ $(n \times (k+p))$
4. Form the (small) matrix $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ $((k+p) \times n)$
5. Factorize $\mathbf{B}$ with SVD: $\mathbf{B} = \hat{\mathbf{U}} \mathbf{\Sigma} \mathbf{V}^* \longrightarrow \hat{\mathbf{U}}$ $((k+p) \times (k+p))$, $\mathbf{\Sigma}$ $((k+p) \times k)$, $\mathbf{V}$ $((k+p) \times n)$
6. Form the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times (k+p))$
7. Truncate the matrices $\mathbf{U}$, $\mathbf{V}$, $\mathbf{\Sigma}$ to k: $\mathbf{U} = \mathbf{U}(:, 1:k)$, $\mathbf{V} = \mathbf{V}(:, 1:k)$, $\mathbf{\Sigma} = \mathbf{\Sigma}(1:k, 1:k)$

Remarks:

- Only 1 extra step to truncate the factorized matrices
- Small value for p (e.g. $p = 10$) is usually sufficient

In the following, we set $l = k + p$.

Back to range finder problem: error bounds

Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix from $\mathbf{Y} = \mathbf{AG}$ such that:

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

As we have $\mathbf{Q} = orth(\mathbf{Y}) \Rightarrow \mathcal{R}(\mathbf{Q}) = \mathcal{R}(\mathbf{Y})$ so that $\mathbf{P_Y} = \mathbf{Q}\mathbf{Q}^* = \mathbf{P_Q}$ and

$$\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A} = (\mathbf{I} - \mathbf{P_Y})\mathbf{A}$$

From previous slides : $\mathcal{R}(\mathbf{A}) \subseteq \mathcal{R}(\mathbf{Q}) \iff \mathbf{A} = \mathbf{P_Y}\mathbf{A}$

How to quantify $\|(\mathbf{I} - \mathbf{P_Y})\mathbf{A}\|$ when $\mathcal{R}(\mathbf{A}) \subsetneq \mathcal{R}(\mathbf{Q})$?

Back to range finder problem: error bounds

Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix from $\mathbf{Y} = \mathbf{AG}$ such that:

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

As we have $\mathbf{Q} = orth(\mathbf{Y}) \Rightarrow \mathcal{R}(\mathbf{Q}) = \mathcal{R}(\mathbf{Y})$ so that $\mathbf{P_Y} = \mathbf{Q}\mathbf{Q}^* = \mathbf{P_Q}$ and

$$\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A} = (\mathbf{I} - \mathbf{P_Y})\mathbf{A}$$

From previous slides : $\mathcal{R}(\mathbf{A}) \subseteq \mathcal{R}(\mathbf{Q}) \iff \mathbf{A} = \mathbf{P_Y}\mathbf{A}$

How to quantify $\|(\mathbf{I} - \mathbf{P_Y})\mathbf{A}\|$ when $\mathcal{R}(\mathbf{A}) \subsetneq \mathcal{R}(\mathbf{Q})$?

Let's split the SVD of $\mathbf{A}$ into first k components and remaining:

$$\mathbf{A} = \begin{bmatrix} \\ \mathbf{U} \\ \\ \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{O} \\ \mathbf{O} & \mathbf{\Sigma}_2 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^* \\ \mathbf{V}_2^* \end{bmatrix}, \ \mathbf{\Sigma}_1 \ (k \times k), \ \mathbf{\Sigma}_2 \ ((n-k) \times (n-k))$$

Sampling $\mathbf{A}$: $\mathbf{Y} = \mathbf{AG} = \begin{bmatrix} \\ \mathbf{U} \\ \\ \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_1\mathbf{G}_1 \\ \mathbf{\Sigma}_2\mathbf{G}_2 \end{bmatrix}, \ \mathbf{G}_1 = \mathbf{V}_1^*\mathbf{G}, \ \mathbf{G}_2 = \mathbf{V}_2^*\mathbf{G}$

Back to range finder problem: error bounds

Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix from $\mathbf{Y} = \mathbf{AG}$ such that:

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$$

How to quantify $\|(\mathbf{I} - \mathbf{P}_\mathbf{Y})\mathbf{A}\|$ when $\mathcal{R}(\mathbf{A}) \subsetneq \mathcal{R}(\mathbf{Q})$?

Let's split the SVD of $\mathbf{A}$ into first k components and remaining:

$$\mathbf{A} = \begin{bmatrix} \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{O} \\ \mathbf{O} & \mathbf{\Sigma}_2 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^* \\ \mathbf{V}_2^* \end{bmatrix}, \ \mathbf{\Sigma}_1 \ (k \times k), \ \mathbf{\Sigma}_2 \ ((n-k) \times (n-k))$$

Sampling $\mathbf{A}$: $\mathbf{Y} = \mathbf{AG} = \begin{bmatrix} \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_1\mathbf{G}_1 \\ \mathbf{\Sigma}_2\mathbf{G}_2 \end{bmatrix}, \ \mathbf{G}_1 = \mathbf{V}_1^*\mathbf{G}, \ \mathbf{G}_2 = \mathbf{V}_2^*\mathbf{G}$

If $\mathbf{G}_1$ is full rank, it may be proven that:

$$\|(\mathbf{I} - \mathbf{P}_\mathbf{Y})\mathbf{A}\|^2 \leq \|\mathbf{\Sigma}_2\|^2 + \|\mathbf{\Sigma}_2\mathbf{G}_2\mathbf{G}_1^+\|^2$$

in both Spectral and Frobenius norms

Back to range finder problem: error bounds

Build an orthonormal $(m \times k)$ $\mathbf{Q}$ matrix from $\mathbf{Y} = \mathbf{AG}$ such that:

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{QQ}^*\mathbf{A}$$

How to quantify $\|(\mathbf{I} - \mathbf{P_Y})\mathbf{A}\|$ when $\mathcal{R}(\mathbf{A}) \subsetneq \mathcal{R}(\mathbf{Q})$?

Let's split the SVD of $\mathbf{A}$ into first k components and remaining:

$$\mathbf{A} = \begin{bmatrix} \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{O} \\ \mathbf{O} & \mathbf{\Sigma}_2 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^* \\ \mathbf{V}_2^* \end{bmatrix}, \ \mathbf{\Sigma}_1 \ (k \times k), \ \mathbf{\Sigma}_2 \ ((n-k) \times (n-k))$$

Sampling $\mathbf{A}$: $\mathbf{Y} = \mathbf{AG} = \begin{bmatrix} \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_1\mathbf{G}_1 \\ \mathbf{\Sigma}_2\mathbf{G}_2 \end{bmatrix}, \ \mathbf{G}_1 = \mathbf{V}_1^*\mathbf{G}, \ \mathbf{G}_2 = \mathbf{V}_2^*\mathbf{G}$

If $\mathbf{G}_1$ is full rank, it may be proven that:

$$\|(\mathbf{I} - \mathbf{P_Y})\mathbf{A}\|^2 \leq \|\mathbf{\Sigma}_2\|^2 + \|\mathbf{\Sigma}_2\mathbf{G}_2\mathbf{G}_1^+\|^2$$

If $rk(\mathbf{A}) = k$, then $\mathbf{\Sigma}_2 = \mathbf{O}$ so that:

$$\mathbf{QQ}^*\mathbf{A} = \mathbf{A}, \quad \mathcal{R}(\mathbf{Y}) = \mathcal{R}(\mathbf{A})$$

Error bounds on the expectation

Recall the classical estimates on the truncated SVD:

$$\|\mathbf{A} - \mathbf{A}_k^{svd}\|_2 = \sigma_{k+1} \qquad \|\mathbf{A} - \mathbf{A}_k^{svd}\|_F = \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)^{1/2}$$

Error bounds on the expectation

Recall the classical estimates on the truncated SVD:

$$\|\mathbf{A} - \mathbf{A}_k^{svd}\|_2 = \sigma_{k+1} \qquad \|\mathbf{A} - \mathbf{A}_k^{svd}\|_F = \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)^{1/2}$$

The RSVD has the following bounds in Frobenius norm (for $p \geq 2$):

$$\begin{aligned} \mathbb{E}\left[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_F\right] &\leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)^{1/2} \\ &\leq \left(1 + \frac{k}{p-1}\right)^{1/2} \|\mathbf{A} - \mathbf{A}_k^{svd}\|_F \end{aligned}$$

Error bounds on the expectation

Recall the classical estimates on the truncated SVD:

$$\|\mathbf{A} - \mathbf{A}_k^{svd}\|_2 = \sigma_{k+1} \qquad \|\mathbf{A} - \mathbf{A}_k^{svd}\|_F = \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)^{1/2}$$

The RSVD has the following bounds in Frobenius norm (for $p \geq 2$):

$$\begin{aligned} \mathbb{E}\left[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_F\right] &\leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)^{1/2} \\ &\leq \left(1 + \frac{k}{p-1}\right)^{1/2} \|\mathbf{A} - \mathbf{A}_k^{svd}\|_F \end{aligned}$$

- $p = 10$: $\mathbb{E}\left[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_F\right] \leq \sqrt{1 + \frac{k}{9}} \|\mathbf{A} - \mathbf{A}_k^{svd}\|_F$
- $p = k + 1$: $\mathbb{E}\left[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_F\right] \leq \sqrt{2} \|\mathbf{A} - \mathbf{A}_k^{svd}\|_F$

Not so bad...

Error bounds on the expectation

Recall the classical estimates on the truncated SVD:

$$\|\mathbf{A} - \mathbf{A}_k^{svd}\|_2 = \sigma_{k+1} \qquad \|\mathbf{A} - \mathbf{A}_k^{svd}\|_F = \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)^{1/2}$$

The RSVD has the following bounds in Spectral norm:

$$\begin{aligned} \mathbb{E}\left[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_2\right] &\leq \left(1 + \sqrt{\frac{k}{p-1}}\right) \sigma_{k+1} + \frac{e\sqrt{k+p}}{p} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)^{1/2} \\ &\leq \left(1 + \sqrt{\frac{k}{p-1}}\right) \|\mathbf{A} - \mathbf{A}_k^{svd}\|_2 + \frac{e\sqrt{k+p}}{p} \|\mathbf{A} - \mathbf{A}_k^{svd}\|_F \end{aligned}$$

Worse... depends on the Frobenius norm (sensitive to decay of σ_k)

Error bounds on the expectation

The RSVD has the following bounds in Spectral norm:

$$\mathbb{E}\left[\|\mathbf{A}-\mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_2\right] \leq \left(1+\sqrt{\frac{k}{p-1}}\right)\sigma_{k+1} + \frac{e\sqrt{k+p}}{p}\left(\sum_{j=k+1}^{\min(m,n)}\sigma_j^2\right)^{1/2}$$

Worse... depends on the Frobenius norm (sensitive to decay of σ_k)

- Fast decay: $\sigma_j \simeq \sigma_{k+1}\beta^{j-(k+1)} \Rightarrow \left(\sum_{j=k+1}^{\min(m,n)}\sigma_j^2\right)^{1/2} = (1-\beta^2)^{-1/2}\sigma_{k+1}$

$$\mathbb{E}\left[\|\mathbf{A}-\mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_2\right] \leq \left(1+\sqrt{\frac{k}{p-1}}+\frac{e\sqrt{k+p}}{p\sqrt{1-\beta^2}}\right)\sigma_{k+1}$$

- No decay:

$$\sigma_j = \sigma_{k+1},\ j>k \quad \Longrightarrow \left(\sum_{j=k+1}^{\min(m,n)}\sigma_j^2\right)^{1/2} = \sigma_{k+1}\sqrt{\min(m,n)-k}$$

$$\mathbb{E}\left[\|\mathbf{A}-\mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_2\right] \leq \left(1+\sqrt{\frac{k}{p-1}}+\frac{e\sqrt{k+p}}{p}\sqrt{\min(m,n)-k}\right)\sigma_{k+1}$$

Error bound on the likelihood of large deviations

It can be proven that for $p \geq 4$,

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \left(1 + 17\sqrt{1 + k/p}\right)\sigma_{k+1} + \frac{8\sqrt{k+p}}{p+1}\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}$$

with failure probability at most $3e^{-p}$

Improved randomized SVD algorithms

Improving error bounds: power sampling scheme

We previously notice that error bounds were very sensitive to the decay of singular values σ_k.

Lets replace $\mathbf{Y} = \mathbf{AG}$ by $\mathbf{Y} = ((\mathbf{AA}^*)^q\mathbf{A})$ during the approximate range finder stage.

Using SVD $\mathbf{A} = \mathbf{U\Sigma V}^*$ of $\mathbf{A}$:

$$\mathbf{AA}^* = \mathbf{U\Sigma}^2\mathbf{U} \Longrightarrow (\mathbf{AA}^*)^q = \mathbf{U\Sigma}^{2q}\mathbf{U}^* \Longrightarrow (\mathbf{AA}^*)^q\mathbf{A} = \mathbf{U\Sigma}^{2q+1}\mathbf{V}^*$$

- $\mathbf{A}$ and $(\mathbf{AA}^*)^q\mathbf{A}$ share the same left singular vectors $\mathbf{U}$
- Singular values of $(\mathbf{AA}^*)^q\mathbf{A}$ are given by $\left\{\sigma_j^{2q+1}\right\}_j$
- ... expected to decay faster than σ_j (for small σ_j) !

Accuracy enhanced RSVD

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling p, nb. of power iterations q

Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1 $\mathbf{G}$ $(n \times l)$

2 $\mathbf{Y} = \mathbf{AG}$ $(n \times l)$

3

4

5

6

7 $\mathbf{Y} = \mathbf{QR} \longrightarrow \mathbf{Q}$ $(n \times l)$

8 $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ $(l \times n)$

9 $\mathbf{B} = \hat{\mathbf{U}} \mathbf{\Sigma} \mathbf{V}^* \longrightarrow \hat{\mathbf{U}}$ $(l \times k)$, $\mathbf{\Sigma}$ $(l \times l)$, $\mathbf{V}$ $(l \times n)$

10 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times l)$

11 $\mathbf{U} = \mathbf{U}(:, 1:k)$, $\mathbf{V} = \mathbf{V}(:, 1:k)$, $\mathbf{\Sigma} = \mathbf{\Sigma}(1:k, 1:k)$

Accuracy enhanced RSVD

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling p, nb. of power iterations q

Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1 $\mathbf{G}$ $(n \times l)$

2 $\mathbf{Y} = \mathbf{AG}$ $(n \times l)$

3 **forall** $j = 1 : q$ **do**

4 $\quad \mathbf{Y} \leftarrow \mathbf{A}^*\mathbf{Y}$

5 $\quad \mathbf{Y} \leftarrow \mathbf{AY}$

6 **end**

7 $\mathbf{Y} = \mathbf{QR} \longrightarrow \mathbf{Q}$ $(n \times l)$

8 $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$ $(l \times n)$

9 $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^* \longrightarrow \hat{\mathbf{U}}$ $(l \times k)$, $\mathbf{\Sigma}$ $(l \times l)$, $\mathbf{V}$ $(l \times n)$

10 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times l)$

11 $\mathbf{U} = \mathbf{U}(:, 1 : k)$, $\mathbf{V} = \mathbf{V}(:, 1 : k)$, $\mathbf{\Sigma} = \mathbf{\Sigma}(1 : k, 1 : k)$

Accuracy enhanced RSVD

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling p, nb. of power iterations q

Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1 $\mathbf{G}$ $(n \times l)$

2 $\mathbf{Y} = \mathbf{AG}$ $(n \times l)$

3 **forall** $j = 1 : q$ **do**

4 $\quad \mathbf{Y} \leftarrow orth(\mathbf{A}^*\mathbf{Y})$

5 $\quad \mathbf{Y} \leftarrow orth(\mathbf{AY})$

6 **end**

7 $\mathbf{Y} = \mathbf{QR} \longrightarrow \mathbf{Q}$ $(n \times l)$

8 $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$ $(l \times n)$

9 $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^* \longrightarrow \hat{\mathbf{U}}$ $(l \times k)$, $\mathbf{\Sigma}$ $(l \times l)$, $\mathbf{V}$ $(l \times n)$

10 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times l)$

11 $\mathbf{U} = \mathbf{U}(:, 1 : k)$, $\mathbf{V} = \mathbf{V}(:, 1 : k)$, $\mathbf{\Sigma} = \mathbf{\Sigma}(1 : k, 1 : k)$

Accuracy enhanced RSVD

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling p, nb. of power iterations q

Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1 $\mathbf{G}$ $(n \times l)$
2 $\mathbf{Y} = \mathbf{AG}$ $(n \times l)$
3 **forall** $j = 1 : q$ **do**
4 $\quad \mathbf{Y} \leftarrow orth(\mathbf{A}^*\mathbf{Y})$
5 $\quad \mathbf{Y} \leftarrow orth(\mathbf{AY})$
6 **end**
7 $\mathbf{Y} = \mathbf{QR} \longrightarrow \mathbf{Q}$ $(n \times l)$
8 $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$ $(l \times n)$
9 $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^* \longrightarrow \hat{\mathbf{U}}$ $(l \times k)$, $\mathbf{\Sigma}$ $(l \times l)$, $\mathbf{V}$ $(l \times n)$
10 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times l)$
11 $\mathbf{U} = \mathbf{U}(:, 1 : k)$, $\mathbf{V} = \mathbf{V}(:, 1 : k)$, $\mathbf{\Sigma} = \mathbf{\Sigma}(1 : k, 1 : k)$

- Less accurate for small singular values due to round-off errors
- Mitigate by reorthonormalizing after applying $\mathbf{A}$
- ... can be limited to some specific power iterations
- Much more interaction with matrix $\mathbf{A}$!

Error bounds for the accuracy enhanced RSVD

Recall the classical estimates on the truncated SVD:

$$\|\mathbf{A} - \mathbf{A}_k\|_2 = \sigma_{k+1} \qquad \|\mathbf{A} - \mathbf{A}_k\|_F = \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2 \right)^{1/2}$$

The accuracy enhanced RSVD has the following bound in spectral norm:

$$\mathbb{E}\left[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_2\right] \leq \left[\left(1 + \sqrt{\frac{k}{p-1}} \right) \sigma_{k+1}^{2q+1} + \frac{e\sqrt{k+p}}{p} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^{2(2q+1)} \right)^{1/2} \right]^{1/(2q+1)}$$

Error bounds for the accuracy enhanced RSVD

The accuracy enhanced RSVD has the following bound in spectral norm:

$$\mathbb{E}\left[\|\mathbf{A}-\mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_2\right] \leq \left[\left(1+\sqrt{\frac{k}{p-1}}\right)\sigma_{k+1}^{2q+1} + \frac{e\sqrt{k+p}}{p}\left(\sum_{j=k+1}^{\min(m,n)}\sigma_j^{2(2q+1)}\right)^{1/2}\right]^{1/(2q+1)}$$

- No decay: $\sigma_j = \sigma_{k+1},\ j > k$

$$\mathbb{E}\left[\|\mathbf{A}-\mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_2\right] \leq \left[1+\sqrt{\frac{k}{p-1}}+\frac{e\sqrt{k+p}}{p}\sqrt{\min(m,n)-k}\right]^{1/(2q+1)}\sigma_{k+1}$$

- Comparison with previous bound:

$$\mathbb{E}\left[\|\mathbf{A}-\mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_2\right] \leq \left[1+\sqrt{\frac{k}{p-1}}+\frac{e\sqrt{k+p}}{p}\sqrt{\min(m,n)-k}\right]\sigma_{k+1}$$

Adaptive rank approximation

Up to now, we only considered the fixed rank approximation problem: given k, compute an approximate SVD as $\mathbf{A} \approx \mathbf{U\Sigma V}$

The fixed precision approximation problem is somewhat more useful: given ε, compute an approximate SVD verifying

$$\|\mathbf{A} - \mathbf{U\Sigma V}\| \leq \varepsilon$$

But rank has to be seeked !

2 types of iterative algorithms:

- direct update of the **A** matrix during the range finder process (fit for dense matrices stored in RAM)
- rank determination without updating **A** matrix (out of core storage, sparse matrix, implicit matrix)

Adaptive rank approximation with updating

Compute $\mathbf{Q}_{i+1}$ iteratively by adding one component $\mathbf{q}_i$ from $\mathcal{R}(\mathbf{A}_i)$, where $\mathbf{A}_i = (\mathbf{I} - \mathbf{Q}_i\mathbf{Q}_i^*)\mathbf{A}$

1 $\mathbf{A}_0 = \mathbf{A}$; $\mathbf{Q}_0 = []$; $\mathbf{B}_0 = []$; $i = 0$;
2 **while** $\|\mathbf{A}_i\| \geq \varepsilon$ **do**
3 i=i+1 ;
4 Pick a unit vector $\mathbf{q}_i \in \mathcal{R}(\mathbf{A}_{i-1})$;
5 $\mathbf{b}_i = \mathbf{q}_i^*\mathbf{A}_{i-1}$;
6 $\mathbf{Q}_i = [\mathbf{Q}_{i-1}\ \mathbf{q}]$;
7 $\mathbf{B}_i = [\mathbf{B}_{i-1}^*\ \mathbf{b}^*]^*$;
8 $\mathbf{A}_i = \mathbf{A}_{i-1} - \mathbf{q}_i\mathbf{b}_i$
9 **end**

Key point: at end of iteration i, $\mathbf{A}_i$ and $\mathbf{B}_i$ verify

$$\mathbf{A}_i = (\mathbf{I} - \mathbf{Q}_i\mathbf{Q}_i^*)\mathbf{A}, \quad \mathbf{B}_i = \mathbf{Q}_i^*\mathbf{A}$$

Proof: let as exercise (recurrency).

Adaptive rank approximation with updating

$\mathbf{A}_0 = \mathbf{A}$; $\mathbf{Q}_0 = []$; $\mathbf{B}_0 = []$; $i = 0$;
while $\|\mathbf{A}_i\| \geq \varepsilon$ **do**
    i=i+1 ;
    Pick a unit vector $\mathbf{q}_i \in \mathcal{R}(\mathbf{A}_{i-1})$;
    $\mathbf{b}_i = \mathbf{q}_i^* \mathbf{A}_{i-1}$;
    $\mathbf{Q}_i = [\mathbf{Q}_{i-1}\ \mathbf{q}]$;
    $\mathbf{B}_i = [\mathbf{B}_{i-1}^*\ \mathbf{b}^*]^*$;
    $\mathbf{A}_i = \mathbf{A}_{i-1} - \mathbf{q}_i \mathbf{b}_i$
end

Key point: at end of iteration i, $\mathbf{A}_i$ and $\mathbf{B}_i$ verify

$$\mathbf{A}_i = (\mathbf{I} - \mathbf{Q}_i \mathbf{Q}_i^*)\mathbf{A}, \quad \mathbf{B}_i = \mathbf{Q}_i^* \mathbf{A}$$

Choice of $\mathbf{q}_i$:

- Largest remaining column of $\mathbf{A}$, but this is CPQR
- Use randomization: $\mathbf{y}_i = \mathbf{A}_{i-1}\mathbf{g}_i$ with $\mathbf{g}_i$ gaussian, $\mathbf{q}_i = \mathbf{y}_i / \|\mathbf{y}_i\|$

Dealing with large number of columns

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling p, pow. iter. q
Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1 $\mathbf{G}$ $(n \times l)$
2 $\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q \mathbf{A}\mathbf{G}$ $(n \times l)$
3 $\mathbf{Q}\mathbf{R} = orth(\mathbf{Y}) \longrightarrow \mathbf{Q}$ $(n \times l)$
4 $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ $(l \times n)$
5 $\hat{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^* = svd(\mathbf{B}) \longrightarrow \hat{\mathbf{U}}$ $(l \times k)$, $\mathbf{\Sigma}$ $(l \times l)$, $\mathbf{V}$ $(l \times n)$
6 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times l)$
7 $\mathbf{U} = \mathbf{U}(:, 1:k)$, $\mathbf{V} = \mathbf{V}(:, 1:k)$, $\mathbf{\Sigma} = \mathbf{\Sigma}(1:k, 1:k)$

When n is large, SVD of the $(l \times n)$ matrix $\mathbf{B}$ may be costful.

Dealing with large number of columns

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling p, pow. iter. q
Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1 $\mathbf{G}$ $(n \times l)$
2 $\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q \mathbf{A}\mathbf{G}$ $(n \times l)$
3 $\mathbf{Q}\mathbf{R} = orth(\mathbf{Y}) \longrightarrow \mathbf{Q}$ $(n \times l)$
4 $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ $(l \times n)$
5 $\hat{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^* = svd(\mathbf{B}) \longrightarrow \hat{\mathbf{U}}$ $(l \times k)$, $\mathbf{\Sigma}$ $(l \times l)$, $\mathbf{V}$ $(l \times n)$
6 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times l)$
7 $\mathbf{U} = \mathbf{U}(:, 1:k)$, $\mathbf{V} = \mathbf{V}(:, 1:k)$, $\mathbf{\Sigma} = \mathbf{\Sigma}(1:k, 1:k)$

When n is large, SVD of the $(l \times n)$ matrix $\mathbf{B}$ may be costful. Some mitigations:

- eigen decomposition of the $(l \times l)$ $\mathbf{B}\mathbf{B}^*$ matrix;
- QR factorisation of the $(n \times l)$ $\mathbf{B}^*$ matrix ($\mathbf{B}^* = \tilde{\mathbf{Q}}\tilde{\mathbf{R}}$) followed by an SVD on the $(l \times l)$ triangular ($\tilde{\mathbf{R}}$) matrix

Dealing with large number of columns

Recall that $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^*$

Eigen decomposition of $\mathbf{B}\mathbf{B}^*$:

$$\mathbf{B}\mathbf{B}^* = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^* = \hat{\mathbf{U}}\mathbf{\Sigma}^2\hat{\mathbf{U}}^* \implies \mathbf{\Sigma} = \sqrt{\mathbf{D}}$$

$$\mathbf{B}^*\hat{\mathbf{U}} = \mathbf{V}\mathbf{\Sigma}\hat{\mathbf{U}}^*\hat{\mathbf{U}} = \mathbf{V}\mathbf{\Sigma} \implies \mathbf{V} = \mathbf{B}^*\hat{\mathbf{U}}\mathbf{\Sigma}^{-1}, \ \mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$$

(Note that l has to be lower than rank of A to ensure $\mathbf{\Sigma}$ is invertible.)

Dealing with large number of columns

Recall that $\mathbf{B} = \hat{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^*$

Eigen decomposition of $\mathbf{B}\mathbf{B}^*$:

$$\mathbf{B}\mathbf{B}^* = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^* = \hat{\mathbf{U}}\mathbf{\Sigma}^2\hat{\mathbf{U}}^* \implies \mathbf{\Sigma} = \sqrt{\mathbf{D}}$$

$$\mathbf{B}^*\hat{\mathbf{U}} = \mathbf{V}\mathbf{\Sigma}\hat{\mathbf{U}}^*\hat{\mathbf{U}} = \mathbf{V}\mathbf{\Sigma} \implies \mathbf{V} = \mathbf{B}^*\hat{\mathbf{U}}\mathbf{\Sigma}^{-1},\ \mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$$

(Note that l has to be lower than rank of A to ensure $\mathbf{\Sigma}$ is invertible.)

QR factorisation on $\mathbf{B}^*$ followed by SVD on $\mathbf{R}$:

$$\begin{aligned}\mathbf{B}^* &= \tilde{\mathbf{Q}}\tilde{\mathbf{R}} = \tilde{\mathbf{Q}}\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^* \\ &= \mathbf{V}\mathbf{\Sigma}\hat{\mathbf{U}}^*\end{aligned}$$

$$\implies \mathbf{\Sigma} = \tilde{\mathbf{\Sigma}},\ \hat{\mathbf{U}} = \tilde{\mathbf{V}},\ \mathbf{V} = \tilde{\mathbf{Q}}\tilde{\mathbf{U}},\ \mathbf{U} = \mathbf{Q}\hat{\mathbf{U}} = \mathbf{Q}\tilde{\mathbf{V}}$$

Enhanced RSVD when n is large

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling p, pow. iter. q
Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1 **Algorithm:** Eigen version

2 $\mathbf{G}$ $(n \times l)$;

3 $\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q \mathbf{A}\mathbf{G}$ $(n \times l)$;

4 $\mathbf{Q}\mathbf{R} = orth(\mathbf{Y}) \longrightarrow \mathbf{Q}$ $(n \times l)$;

5 $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ $(l \times n)$;

6 $\mathbf{T} = \mathbf{B}\mathbf{B}^*$ $(l \times l)$;

7 $\hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^* = eig(\mathbf{T})$ $(l \times l) \longrightarrow \hat{\mathbf{U}},\ \mathbf{D}$;

8 $\mathbf{\Sigma} = \sqrt{\mathbf{D}}$;

9 $\mathbf{V} = \mathbf{B}^* \hat{\mathbf{U}} \mathbf{\Sigma}^{-1}$;

10 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times l)$;

11 $\mathbf{U} = \mathbf{U}(:, 1:k)$, $\mathbf{V} = \mathbf{V}(:, 1:k)$,
$\mathbf{\Sigma} = \mathbf{\Sigma}(1:k, 1:k)$

Enhanced RSVD when n is large

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling p, pow. iter. q
Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1 **Algorithm:** Eigen version
2 $\mathbf{G}$ $(n \times l)$;
3 $\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q \mathbf{A}\mathbf{G}$ $(n \times l)$;
4 $\mathbf{Q}\mathbf{R} = orth(\mathbf{Y}) \longrightarrow \mathbf{Q}$ $(n \times l)$;
5 $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ $(l \times n)$;
6 $\mathbf{T} = \mathbf{B}\mathbf{B}^*$ $(l \times l)$;
7 $\hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^* = eig(\mathbf{T})$ $(l \times l) \longrightarrow \hat{\mathbf{U}},\ \mathbf{D}$;
8 $\mathbf{\Sigma} = \sqrt{\mathbf{D}}$;
9 $\mathbf{V} = \mathbf{B}^* \hat{\mathbf{U}} \mathbf{\Sigma}^{-1}$;
10 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times l)$;
11 $\mathbf{U} = \mathbf{U}(:, 1:k), \mathbf{V} = \mathbf{V}(:, 1:k),$
$\mathbf{\Sigma} = \mathbf{\Sigma}(1:k, 1:k)$

1 **Algorithm:** QR version
2 $\mathbf{G}$ $(n \times l)$;
3 $\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q \mathbf{A}\mathbf{G}$ $(n \times l)$;
4 $\mathbf{Q}\mathbf{R} = orth(\mathbf{Y}) \longrightarrow \mathbf{Q}$ $(n \times l)$;
5 $\mathbf{B}^* = \mathbf{A}^* \mathbf{Q}$ $(n \times l)$;
6 $\tilde{\mathbf{Q}}\tilde{\mathbf{R}} = orth(\mathbf{B}^*) \longrightarrow \tilde{\mathbf{R}}$ $(l \times l)$;
7 $\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}} = svd(\tilde{\mathbf{R}})$ $(l \times l) \longrightarrow \tilde{\mathbf{\Sigma}}$;
8 $\mathbf{V} = \tilde{\mathbf{Q}}\tilde{\mathbf{U}}$;
9 $\mathbf{U} = \mathbf{Q}\tilde{\mathbf{V}}$;
10 $\mathbf{U} = \mathbf{U}(:, 1:k), \mathbf{V} = \mathbf{V}(:, 1:k),$
$\mathbf{\Sigma} = \mathbf{\Sigma}(1:k, 1:k)$

Single pass algorithm

Single pass / streaming algorithms

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling p

Output: Approximate SVD matrices $\mathbf{U}$ $(m \times k)$, $\mathbf{V}$ $(k \times n)$, $\mathbf{\Sigma}$ $(k \times k)$

1 $\mathbf{G}$ $(n \times l)$

2 $\mathbf{Y} = \mathbf{AG}$ $(n \times l)$

3 $\mathbf{QR} = orth(\mathbf{Y}) \longrightarrow \mathbf{Q}$ $(n \times l)$

4 $\mathbf{B} = \mathbf{Q}^* \mathbf{A}$ $(l \times n)$

5 $\hat{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^* = svd(\mathbf{B}) \longrightarrow \hat{\mathbf{U}}$ $(l \times k)$, $\mathbf{\Sigma}$ $(l \times l)$, $\mathbf{V}$ $(l \times n)$

6 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(m \times l)$

7 $\mathbf{U} = \mathbf{U}(:, 1:k)$, $\mathbf{V} = \mathbf{V}(:, 1:k)$, $\mathbf{\Sigma} = \mathbf{\Sigma}(1:k, 1:k)$

Recall that one needs 2 interactions with $\mathbf{A}$ to compute $\mathbf{Q}$ and $\mathbf{B}$.

What if A is too large to be stored on RAM (out of core), or if we access it only through a data flow (streaming data) ?

Build a streaming algorithm:

- each entry of A is accessed only once
- order of processing does not matter

Single pass algorithm for symmetric (hermitian) matrices

Suppose $\mathbf{A} = \mathbf{A}^*$, then, we seek to compute an approximate eigenvalues decomposition $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{U}^*$

Given $\mathbf{G}$ gaussian, $\mathbf{Y} = \mathbf{A}\mathbf{G}$ and $\mathbf{Q} = orth(\mathbf{Y})$, we can build $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$ as before, but as $\mathbf{A} = \mathbf{A}^*$, we also have

$$\mathbf{A} \approx \mathbf{A}\mathbf{Q}\mathbf{Q}^* \Longrightarrow \mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{Q}\mathbf{Q}^* = \mathbf{Q}\mathbf{C}\mathbf{Q}^* \text{ with } \mathbf{C} = \mathbf{Q}^*\mathbf{A}\mathbf{Q}$$

If $\mathbf{C}$ is known, $\mathbf{C} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^* \Longrightarrow \mathbf{A} \approx \mathbf{Q}\mathbf{C}\mathbf{Q}^* = \mathbf{Q}\hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^*\mathbf{Q}^* = \mathbf{U}\mathbf{D}\mathbf{U}^*$

The goal now is to evaluate $\mathbf{C}$ with only one pass on $\mathbf{A}$!

One has $\mathbf{C}(\mathbf{Q}^*\mathbf{G}) = \mathbf{Q}^*\mathbf{A}\mathbf{Q}\mathbf{Q}^*\mathbf{G} \approx \mathbf{Q}^*\mathbf{A}\mathbf{G} = \mathbf{Q}\mathbf{Y}$ so that $\mathbf{C}$ is the solution of the $l \times l$ overdetermined system (recall $\mathbf{C}$ is symmetric)

$$\mathbf{C}(\mathbf{Q}^*\mathbf{G}) = (\mathbf{Q}^*\mathbf{Y})$$

Single pass algorithm for symmetric (hermitian) matrices

Input: Matrix $\mathbf{A}$ $(n \times n)$, target rank k, oversampling p

Output: Approximate EVD matrices $\mathbf{U}$ $(n \times k)$, $\mathbf{D}$ $(k \times k)$

1 $\mathbf{G}$ $(n \times l)$

2 $\mathbf{Y} = \mathbf{AG}$ $(n \times l)$

3 $\mathbf{QR} = orth(\mathbf{Y}) \longrightarrow \mathbf{Q}$ $(n \times l)$

4 Form the $(\mathbf{Q}^*\mathbf{G})$ and $(\mathbf{Q}^*\mathbf{Y})$ matrices

5 Solve $\mathbf{C}(\mathbf{Q}^*\mathbf{G}) = (\mathbf{Q}^*\mathbf{Y})$ $(l \times l) \longrightarrow \mathbf{C}$

6 $\hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^* = eig(\mathbf{C}) \longrightarrow \hat{\mathbf{U}}$ $(l \times l)$, $\mathbf{D}$ $(l \times l)$

7 $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$ $(n \times l)$

8 $\mathbf{U} = \mathbf{U}(:, 1:k)$, $\mathbf{D} = \mathbf{D}(1:k, 1:k)$

Unfortunately, $\mathbf{Q}^*\mathbf{G}$ may be ill-conditionned... Using some aggressive extra sampling ($p = k$) together with a k-rank SVD on $\mathbf{Y}$ to select $\mathbf{Q}$ as the leading singular vectors may help.

Due to the ”double” approximation, this algorithms is less accurate than the traditionnal RSVD.

Structure preserving factorization

Structure preserving factorization

A $(m \times n)$ matrix $\mathbf{A}$ of rank k admits an interpolative decomposition (ID) factorisation that may take the following form (I_k and J_k defining indices subsets of the rows/columns):

- Column ID

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{A}(:, J_k)} \quad \underset{k\times n}{\mathbf{Z}}$$

- Row ID

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{X}} \quad \underset{k\times n}{\mathbf{A}(I_k, :)}$$

- Double-sided ID

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{X}} \quad \underset{k\times k}{\mathbf{A}(I_k, J_k)} \quad \underset{k\times n}{\mathbf{Z}}$$

Those specific types of factorisation are called *structure preserving* as they may preserve some properties of the initial matrix:

- sparsity and non-negativity preserved
- requires less memory storage than SVD / QR

However, when $\mathbf{A}$ is not of rank k, we will see that it shares the same approximation error as the QR, larger than the minimal one (SVD).

Column interpolative decomposition (ID)

Standard CPQR routine may be used with some light post-processing to compute the column ID.

Let's start with a partial CPQR up to k steps:

$$\mathbf{AP} = [\mathbf{Q}_1\ \mathbf{Q}_2] \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{O} & \mathbf{S}_{22} \end{bmatrix} = \mathbf{Q}_1[\mathbf{S}_{11}\ \mathbf{S}_{12}] + \mathbf{Q}_2[\mathbf{O}\ \mathbf{S}_{22}]$$

Introducing the pivoting indices $J = [J_k, J_{n-k}]$ such that $\mathbf{P} = \mathbf{I}(:, J)$:

$$\mathbf{AP} = \mathbf{A}(:, J) = \mathbf{QR} \quad \text{and} \quad \mathbf{A}(:, J_k) = \mathbf{Q}_1\mathbf{S}_{11} \text{ (first } k \text{ columns)}$$

On the other hand, k-rank approximation of $\mathbf{A}$ is given similarly to the QR, with same error bound $\|\mathbf{A} - \mathbf{A}_{approx}\| \leq \|\mathbf{S}_{22}\|$, by:

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{Q}_1[\mathbf{S}_{11}\ \mathbf{S}_{12}]\mathbf{P}^* = \mathbf{Q}_1\mathbf{S}_{11}[\mathbf{I}_k\ \mathbf{T}]\mathbf{P}^* \quad \text{with} \quad \mathbf{S}_{11}\mathbf{T} = \mathbf{S}_{12}$$

Then, we get

$$\mathbf{A} \approx \mathbf{A}_{approx} = \mathbf{A}(:, J_k)\mathbf{Z} \quad \text{with} \quad \mathbf{Z} = [\mathbf{I}_k\ \mathbf{T}]\mathbf{P}$$

Low rank column ID decomposition

Input: Matrix $\mathbf{A}$ $_{(m\times n)}$, target rank k

Output: Column set J_k and matrix $\mathbf{Z}$ $_{(k\times n)}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_k)\mathbf{Z}$

1. k-step CPQR $\quad \rightarrow \ \mathbf{AP} \approx \mathbf{Q}_1\mathbf{S}_1$
2. Get $J_k = J(1:k)$ from $\mathbf{P} = \mathbf{I}(:, J)$
3. Extract $\mathbf{S}_{11} = \mathbf{S}_1(:, 1:k)$ and $\mathbf{S}_{12} = \mathbf{S}_1(:, k+1:n)$
4. Solve $\mathbf{S}_{11}\mathbf{T} = \mathbf{S}_{12} \quad \rightarrow \mathbf{T}$
5. $\mathbf{Z} = [\mathbf{I}_k \ \mathbf{T}]\mathbf{P}$

How to build a row ID decomposition ? Simply by processing the column ID decomposition of $\mathbf{A}^*$!

How to build a double-sided ID decomposition ? Simply by chaining a column and a row ID decomposition !

Double-sided ID decomposition

1 **Algorithm:** Double-sided ID decomposition

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k

Output: Column sets I_k, J_k and matrices $\mathbf{X}$ $(m \times k)$, $\mathbf{Z}$ $(k \times n)$ such that

$$\mathbf{A} \approx \mathbf{X}\mathbf{A}(I_k, J_k)\mathbf{Z}$$

2 k-rank column ID on $\mathbf{A}$ $\quad \rightarrow \mathbf{A} \approx \mathbf{A}(:, J_k)\mathbf{Z}$

3 k-rank column ID on $\mathbf{A}(:, J_k)^*$ $\quad \rightarrow \mathbf{A}(:, J_k) \approx \mathbf{X}\mathbf{A}(I_k, J_k)$

with

1 **Algorithm:** Column ID decomposition

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k

Output: Column set J_k and matrix $\mathbf{Z}$ $(k \times n)$ such that $\mathbf{A} \approx \mathbf{A}(:, J_k)\mathbf{Z}$

2 k-step CPQR $\quad \rightarrow \mathbf{A}\mathbf{P} \approx \mathbf{Q}_1\mathbf{S}_1$

3 Get $J_k = J(1:k)$ from $\mathbf{P} = \mathbf{I}(:, J)$

4 Extract $\mathbf{S}_{11} = \mathbf{S}_1(:, 1:k)$ and $\mathbf{S}_{12} = \mathbf{S}_1(:, k+1:n)$

5 Solve $\mathbf{S}_{11}\mathbf{T} = \mathbf{S}_{12}$ $\quad \rightarrow \mathbf{T}$

6 $\mathbf{Z} = [\mathbf{I}_k \ \mathbf{T}]\mathbf{P}$

Back to the low rank column ID decomposition

Input: Matrix $\mathbf{A}_{(m\times n)}$, target rank k

Output: Column set J_k and matrix $\mathbf{Z}_{(k\times n)}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_k)\mathbf{Z}$

1. k-step CPQR $\quad \rightarrow \mathbf{AP} \approx \mathbf{Q}_1\mathbf{S}_1$
2. Get $J_k = J(1:k)$ from $\mathbf{P} = \mathbf{I}(:, J)$
3. Extract $\mathbf{S}_{11} = \mathbf{S}_1(:, 1:k)$ and $\mathbf{S}_{12} = \mathbf{S}_1(:, k+1:n)$
4. Solve $\mathbf{S}_{11}\mathbf{T} = \mathbf{S}_{12} \quad \rightarrow \mathbf{T}$
5. $\mathbf{Z} = [\mathbf{I}_k \ \mathbf{T}]\mathbf{P}$

When $k \ll \min(m, n)$, use a partial k factorisation ($O(mnk)$) instead of a full one ($O(mn\min(m, n))$).

How to further accelerate the algorithm ? Use randomisation !

Let us notice that only $\mathbf{S}_1$ is involved in the ID decomposition, which accounts for the linear dependency of the columns of $\mathbf{A}$. The idea is to replace $(m \times n)$ matrix $\mathbf{A}$ by a smaller $(l \times n)$ matrix representative of those dependencies.

Back to the low rank column ID decomposition

Input: Matrix $\mathbf{A}_{(m\times n)}$, target rank k

Output: Column set J_k and matrix $\mathbf{Z}_{(k\times n)}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_k)\mathbf{Z}$

1. k-step CPQR $\quad \rightarrow \mathbf{AP} \approx \mathbf{Q}_1\mathbf{S}_1$
2. Get $J_k = J(1:k)$ from $\mathbf{P} = \mathbf{I}(:, J)$
3. Extract $\mathbf{S}_{11} = \mathbf{S}_1(:, 1:k)$ and $\mathbf{S}_{12} = \mathbf{S}_1(:, k+1:n)$
4. Solve $\mathbf{S}_{11}\mathbf{T} = \mathbf{S}_{12} \quad \rightarrow \mathbf{T}$
5. $\mathbf{Z} = [\mathbf{I}_k \ \mathbf{T}]\mathbf{P}$

How to further accelerate the algorithm ? Use randomisation !

Let us notice that only $\mathbf{S}_1$ is involved in the ID decomposition, which accounts for the linear dependency of the columns of $\mathbf{A}$. The idea is to replace $(m \times n)$ matrix $\mathbf{A}$ by a smaller $(l \times n)$ matrix representative of those dependencies.

Draw a $(l \times m)$ random matrix $\mathbf{G}$ and use $\mathbf{Y} = \mathbf{GA}$ as input to the CPQR.

Randomized column ID decomposition

Input: Matrix $\mathbf{A}$ $(m \times n)$, target rank k, oversampling p, pow. iter. q

Output: Column set J_k and matrix $\mathbf{Z}$ $(k \times n)$ such that $\mathbf{A} \approx \mathbf{A}(:, J_k)\mathbf{Z}$

1. $l = k + p$
2. $\mathbf{G}$ $(l \times m)$ random Gaussian matrix
3. $\mathbf{Y} = \mathbf{G}\mathbf{A}(\mathbf{A}^*\mathbf{A})^q$
4. k-step CPQR $\rightarrow$ $\mathbf{YP} \approx \mathbf{QS}$
5. $\mathbf{Q}_1 = \mathbf{Q}(:, 1:k)$, $\mathbf{S}_1 = \mathbf{S}(1:k, :)$
6. Get $J_k = J(1:k)$ from $\mathbf{P} = \mathbf{I}(:, J)$
7. Extract $\mathbf{S}_{11} = \mathbf{S}_1(:, 1:k)$ and $\mathbf{S}_{12} = \mathbf{S}_1(:, k+1:n)$
8. Solve $\mathbf{S}_{11}\mathbf{T} = \mathbf{S}_{12}$ $\rightarrow \mathbf{T}$
9. $\mathbf{Z} = [\mathbf{I}_k \ \mathbf{T}]\mathbf{P}$

Questions?