Deterministic algorithms for the low rank approximation of matrices

ANF "Réduction de la dimension dans la fouille de données massives : enjeux, méthodes et outils pour le calcul "
(Ile d'Oléron)

Xavier VASSEUR (xavier.vasseur@isae.fr)

September 27 2017

Institut Supérieur de l'Aéronautique et de l'Espace

Outline

Objectives

Objectives

- Given $A \in \mathbb{C}^{m \times n}$ with $p = \min(m, n)$ we seek to compute a rank-k approximation, typically with $k \ll p$ (say $m, n \sim 10^4, 10^6, 10^8, \cdots$ and $k \approx 10$ or 10^2) as

$$A \approx E\, F^H, \quad E \in \mathbb{C}^{m \times k}, \quad F \in \mathbb{C}^{n \times k}.$$

- Solving this problem usually requires algorithms for computing the **Singular Value Decomposition** (SVD) or an **eigendecomposition** corresponding to dominant eigenvalues.
- **Goal of the lecture** : review standard **deterministic** approaches for the low rank approximation of matrices (**sparse and dense**).
- Those methods are **building blocks** in more recent advanced strategies (e.g. randomized methods, see lecture of Pierre Blanchard).
- Focus on the analysis of the standard algorithms with respect to **parallelism**.

Current challenges in algorithmic design

Current challenges in algorithmic design

- **State-of-the-art** deterministic methods of numerical linear algebra were designed for an environment where the matrix **fits into memory** (RAM) and the key to performance was to minimize the number of **floating point operations** (flops) required.
- Currently, **communication** is the real bottleneck
  - Moving data from a hard drive
  - Moving data between nodes of a parallel machine
  - Moving data between nodes of a cloud computer.
- Ideally we should target for efficient algorithms scaling **linearly** with the problem size and with **minimal data movement**.
- This is required due to the **increasingly large amount of data** in current applications (web search, machine learning, social networks, genomics/proteomics data, $\cdots$).

Outline

Material and goals of the lecture

Material

- Lecture notes with references to original papers and additional suggested readings.
- Interactive julia notebook during the afternoon session.
- **"Real-life"** applications during the afternoon session.

Goals of the lecture

- Briefly review **fundamental matrix decompositions** (Section 2).
- Provide a brief review on **deterministic methods** for low rank approximations with an emphasis on parallel aspects (Sections 3 and 4).
- Shortly describe related **parallel software** (Section 5).
- Give numerical illustrations in julia during the afternoon session.

Outline

Preliminaries (Linear algebra) - Basic definitions

Norms

- **Euclidean inner product** : $x \in \mathbb{C}^n, y \in \mathbb{C}^n, < x, y >= y^H x = \sum_{j=1}^n x_j \bar{y}_j$.
- **Euclidean norm** : $x \in \mathbb{C}^n, \quad \|x\|_2 = (\sum_{j=1}^n |x_j|^2)^{1/2}$.
- ℓ^p **norm** : $x \in \mathbb{C}^n, \quad \|x\|_p = (\sum_{j=1}^n |x_j|^p)^{1/p}$.
- **Spectral norm** of $A \in \mathbb{C}^{m \times n}$:

$$\|A\|_2 = \sup_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}, x \in \mathbb{C}^n.$$

- **Frobenius norm** of $A \in \mathbb{C}^{m \times n}$:

$$\|A\|_F = (\sum_{i=1}^m \sum_{j=1}^n |A_{ij}|^2)^{1/2}.$$

Preliminaries (Linear algebra) - Basic definitions

Transpose and adjoint

- Given $A \in \mathbb{C}^{m \times n}$, the **transpose** $B = A^T \in \mathbb{C}^{n \times m}$ is defined as :

$$B_{ij} = A_{ji}.$$

- Given $A \in \mathbb{C}^{m \times n}$, the **adjoint** $A^H \in \mathbb{C}^{n \times m}$ is defined as :

$$A^H = \overline{A^T}.$$

- Useful norm relation :

$$\|A\|_2^2 = \|AA^H\|_2 = \|A^H A\|_2.$$

Preliminaries (Linear algebra) - Basic definitions

Special classes of matrices

- A $m \times n$ matrix A is **orthonormal** if its columns form an orthonormal basis, i.e., $A^H A = I_n$.
- A $n \times n$ matrix A is **normal** if $A^H A = AA^H$.
- A $n \times n$ real matrix A is **symmetric** if $A^T = A$.
- A $n \times n$ matrix A is **self-adjoint (Hermitian)** if $A^H = A$.
- A $n \times n$ matrix A is **skew-adjoint** if $A^H = -A$.
- A $n \times n$ matrix A is **unitary** if it is invertible and $A^H = A^{-1}$.
- A $n \times n$ self-adjoint matrix A is said to be **positive definite** if :

$$\forall x \in \mathbb{C}^n, x \neq 0, x^H A x > 0.$$

- A $n \times n$ self-adjoint matrix A is said to be **positive semi-definite** if :

$$\forall x \in \mathbb{C}^n, x \neq 0, x^H A x \geq 0.$$

Preliminaries (Linear algebra) - Basic definitions

Spectral decomposition

- λ is an **eigenvalue** and v an **eigenvector** of $A \in \mathbb{C}^{n\times n}$ if :

$$v \neq 0,\ Av = \lambda v.$$

- $A \in \mathbb{C}^{n\times n}$ is **normal** if and only if A admits a factorization of the form :

$$A = VDV^H,$$

where $V = [v_1\, v_2 \cdots v_n] \in \mathbb{C}^{n\times n}$ is unitary and $D \in \mathbb{C}^{n\times n}$ is diagonal with entries $\lambda_j, (j = 1, \cdots, n)$.

- The previous relation can alternatively be written

$$A = \sum_{j=1}^{n} \lambda_j v_j v_j^H,$$

where (λ_j, v_j) are the **eigenpairs** of A.

Preliminaries (Linear algebra) - Basic definitions

Partial spectral decomposition and Krylov subspace methods

- If $A \in \mathbb{C}^{n \times n}$ can be applied **rapidly** to vectors as happens when A is sparse or structured, then **Krylov subspace methods** can accurately and effectively compute a **partial spectral decomposition**.
- Given $v \in \mathbb{C}^n$ such as $\|v\|_2 = 1$ and $A \in \mathbb{C}^{n \times n}$, the Krylov subspace of dimension at most k generated by A and v is defined as :

$$\mathcal{K}_k(A, v) = span\{v, Av, \cdots, A^{k-1}v\}.$$

- The idea is to seek approximations of eigenvectors **within this particular subspace**.
- **Caveat** : the most basic versions of Krylov subspace methods are **numerically unstable** !
- **Block Krylov subspace methods** consider the case where the starting vector v is replaced by a starting matrix V of appropriate dimension. A **richer** subspace is then expected.

Preliminaries (Linear algebra) - Basic definitions

Householder reflection (Householder matrix, Householder transformation)

- Given $v \in \mathbb{C}^m$ such as $\|v\|_2 = 1$, the Householder matrix H_v is defined as :
$$H_v = I_m - 2v\,v^H.$$
- H_v is Hermitian (self-adjoint).
- H_v is unitary and $H_v^2 = I_m$.
- $H_v v = -v$, $\forall w \in v^\perp, H_v w = w$.
- **Application** : Given $u \in \mathbb{C}^m$ with $u_1 = e^{i\theta_1}\|u\|_2$, $e_1 \in \mathbb{C}^m$ (first unit vector), and $v = \dfrac{u - e^{i\theta_1}\|u\|_2 e_1}{\|u - e^{i\theta_1}\|u\|_2 e_1\|_2}$,
$$H_v u = u_1 e_1.$$
- This is a basic step in the **Householder QR** factorization.

Preliminaries (Linear algebra) - Basic definitions

Householder-QR factorization

- Reduce $A \in \mathbb{C}^{m \times m}$ to triangular form $H_L\ A = R$

$$A = \begin{bmatrix} \star & \star & \star \\ \star & \star & \star \\ \star & \star & \star \end{bmatrix} \xrightarrow{H_{L,1}} \begin{bmatrix} \square & \square & \square \\ & \square & \square \\ & \square & \square \end{bmatrix} \xrightarrow{H_{L,2}} \begin{bmatrix} \square & \square & \square \\ & \circ & \circ \\ & & \circ \end{bmatrix}$$

  where $H_{L,i}$ indicates a left-multiplication by a **Householder** reflection. At the end of this step we have :

$$H_{L,m-1} \cdots H_{L,1}\ A = R.$$

- **Final step** : $A = QR$ with $Q = H_{L,1} \cdots H_{L,m-1}$.
- **Complexity** : $O(m^2 p)$ with $p = min(m, n)$.
- **Parallel performance** : **low** since heavily based on sequence of matrix-vector operations due to Householder reflections.

Preliminaries (Parallel computing)

Main features to analyze

- **Data distribution**.
- **Load balancing** property of the algorithm.
- **Weak and strong** scalability properties of the algorithm.
- **Resiliency** and **fault-tolerant** properties of the algorithm.

Distributed data analysis and scientific computing

- Apache **Hadoop Map/Reduce** (**RDD** : Resilient Data Distribution).
- Spark Apache **MLlib** (Dimensionality Reduction : SVD, PCA).
- **Message Passing Interface** (MPI).
- **R and Distributed R** (Rmpi, RHadoop).

Preliminaries (Parallel computing)

Map/Reduce algorithms

- **Framework** for processing parallelizable problems across large datasets using a large number of nodes on a cluster.
- **Current methodology** :
  - **Map** : Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of redundant input data is processed.
  - **Shuffle** : Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
  - **Reduce** : Worker nodes now process each group of output data, per key, in parallel.
- Widely used in **Big data** applications.
- An efficient **distributed file system** is usely required.

Suggested reading

Suggested reading

- **A. Gittens, A. Devarakonda, E. Racah, M. Ringenburg, L. Gerhardt, J. Kottalam, J. Liu, K. Maschhoff, S. Canon, J. Chhugani, P. Sharma, J. Yang, J. Demmel, J. Harrell, V. Krishnamurthy and M. Mahoney** *Matrix factorizations at scale : A comparison of scientific data analytics in Spark and C+ MPI using three case studies*, IEEE International Conference on Big Data, pp. 204-213, 2016.
- **G. Golub, and C. Van Loan**. *Matrix Computations*, Johns Hopkins University, fourth edition, 2012.
- **G. Hager and G. Wellein**. *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, 2010.
- **V. Miele and V. Louvet**. *Calcul parallèle avec R*, EDP Sciences, 2016.

Outline

Objectives and key idea

Objectives

- Review **fundamental matrix decompositions** that are **useful** for low rank approximations.
- Focus on **parallel** properties of the algorithms and discuss parallel performance on modern computing platforms.
- Provide first numerical illustrations in julia.

Key idea

- Exploit the **optimality** property of the Singular Value Decomposition in terms of approximation to provide a rank-k approximation of a given matrix.

Outline

Full SVD

Full SVD [Beltrami, 1873], [Jordan, 1874], [Sylvester, 1889], [Picard, 1910]

- Given $A \in \mathbb{C}^{m \times n}$ with $p = \min(m, n)$, the **full** singular value decomposition of A reads :

$$A = U \, \Sigma \, V^H,$$

  with $U \in \mathbb{C}^{m \times m}$, $V \in \mathbb{C}^{n \times n}$ unitary ($U^H U = I_m$, $V^H V = I_n$) and $\Sigma \in \mathbb{R}^{n \times m}$.
- $\Sigma = diag(\sigma_1, \cdots, \sigma_p)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$.
- $\sigma_i, (i = 1, p)$ are called **singular values** of A.
- The columns of $U = [u_1, u_2, \cdots, u_p]$ are called **left singular vectors** : $AV = U \, \Sigma$.
- The columns of $V = [v_1, v_2, \cdots, v_p]$ are called **right singular vectors** : $A^H U = V \, \Sigma^H$.
- (U, Σ, V) is called a **singular triplet** of A (non unique !)

Thin SVD and truncated SVD

Thin, compact, thresholded and truncated SVD

- Given $A \in \mathbb{C}^{m \times n}$ with $p = \min(m, n)$, the **thin** singular value decomposition of A reads :

$$A = U \, \Sigma \, V^H,$$

with $U \in \mathbb{C}^{m \times p}$, $V \in \mathbb{C}^{n \times p}$ with orthonormal columns ($U^H U = I_p$, $V^H V = I_p$) and $\Sigma \in \mathbb{R}^{p \times p}$ a diagonal matrix.

- A **compact** SVD only keeps the r singular triplets corresponding to nonzero singular values ($r = \text{rank}(A)$).

Thin SVD and truncated SVD

Thin, compact, thresholded and truncated SVD

- Given $A \in \mathbb{C}^{m \times n}$ with $p = \min(m, n)$, the **thin** singular value decomposition of A reads :

$$A = U \, \Sigma \, V^H,$$

with $U \in \mathbb{C}^{m \times p}$, $V \in \mathbb{C}^{n \times p}$ with orthonormal columns ($U^H U = I_p$, $V^H V = I_p$) and $\Sigma \in \mathbb{R}^{p \times p}$ a diagonal matrix.

- A **thresholded** SVD only keeps the singular triplets with singular values larger than a given positive threshold τ.

Thin SVD and truncated SVD

Thin, compact, thresholded and truncated SVD

- Given $A \in \mathbb{C}^{m \times n}$ with $p = \min(m, n)$, the **thin** singular value decomposition of A reads :

$$A = U \, \Sigma \, V^H,$$

with $U \in \mathbb{C}^{m \times p}$, $V \in \mathbb{C}^{n \times p}$ with orthonormal columns ($U^H U = I_p$, $V^H V = I_p$) and $\Sigma \in \mathbb{R}^{p \times p}$ a diagonal matrix.

- A **truncated** SVD (to rank k) corresponds to the approximation :

$$A_k = \sum_{j=1}^{k} \sigma_j u_j v_j^H = U(:, 1:k) \Sigma(1:k, 1:k) V(:, 1:k)^H.$$

Thin SVD and truncated SVD

Thin, compact, thresholded and truncated SVD

- Given $A \in \mathbb{C}^{m \times n}$ with $p = \min(m, n)$, the **thin** singular value decomposition of A reads :

$$A = U \, \Sigma \, V^H,$$

with $U \in \mathbb{C}^{m \times p}$, $V \in \mathbb{C}^{n \times p}$ with orthonormal columns ($U^H U = I_p$, $V^H V = I_p$) and $\Sigma \in \mathbb{R}^{p \times p}$ a diagonal matrix.

- A **truncated** SVD (to rank k) corresponds to the approximation :

$$A_k = \sum_{j=1}^{k} \sigma_j u_j v_j^H = U(:, 1:k) \Sigma(1:k, 1:k) V(:, 1:k)^H.$$

- **Approximation property [Eckart-Young-Mirsky theorem, 1936]** :

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

Thin SVD and truncated SVD

Thin, compact, thresholded and truncated SVD

- Given $A \in \mathbb{C}^{m \times n}$ with $p = \min(m, n)$, the **thin** singular value decomposition of A reads :

$$A = U \, \Sigma \, V^H,$$

with $U \in \mathbb{C}^{m \times p}$, $V \in \mathbb{C}^{n \times p}$ with orthonormal columns ($U^H U = I_p$, $V^H V = I_p$) and $\Sigma \in \mathbb{R}^{p \times p}$ a diagonal matrix.

- A **truncated** SVD (to rank k) corresponds to the approximation :

$$A_k = \sum_{j=1}^{k} \sigma_j u_j v_j^H = U(:, 1:k) \Sigma(1:k, 1:k) V(:, 1:k)^H.$$

- **Approximation property [Eckart-Young-Mirsky theorem, 1936]** :

$$\|A - A_k\|_F = \min_{\text{rank(B)=k}} \|A - B\|_F = \sum_{i=k+1}^{p} \sigma_i, B \in \mathbb{C}^{m \times n}.$$

Sketch of the standard SVD algorithm [Golub and Kahan, 1965]

- **First step** : Reduce $A \in \mathbb{C}^{m \times n}$ to upper bidiagonal form $H_L \; A \; H_R = B$

$$A \xrightarrow{H_{L,1}} \begin{bmatrix} \star & \star & \star \\ & \star & \star \\ & \star & \star \\ & \star & \star \end{bmatrix} \xrightarrow{H_{R,1}} \begin{bmatrix} \star & \star & \\ & \star & \star \\ & \star & \star \\ & \star & \star \end{bmatrix} \xrightarrow{H_{L,2}} \begin{bmatrix} \star & \star & \\ & \star & \star \\ & & \star \\ & & \star \end{bmatrix} \xrightarrow{H_{L,3}} B$$

  where $H_{L,i}$ indicates a left-multiplication by a **Householder** reflection and $H_{R,i}$ a right-multiplication. At the end of this step we have :

$$H_{L,m-1} \cdots H_{L,1} \; A \; H_{R,1} \cdots H_{R,n-2} = B.$$

- **Second step** : Perform a bidiagonal SVD of B as $B = U_B \Sigma V_B^H$.
- **Final step** : $A = U \; \Sigma \; V^H$ with $U = H_L^H U_B$ and $V = H_R V_B$.
- **Complexity** : $O(mnp)$ with $p = min(m, n)$.
- **Parallel performance** : **low** since heavily based on sequence of matrix-vector operations due to Householder reflections.

Outline

R-SVD

R-SVD [Chan, 1982]

- **Idea** : Perform an initial QR decomposition if the matrix is sufficiently tall relative to its width (i.e. $m \geq n$ with at least by a factor of 1.5).
- **First step** : QR factorization of $A \in \mathbb{C}^{m \times n}$ as $A = QR$ where $Q \in \mathbb{C}^{m \times n}$ has orthonormal columns ($Q^H Q = I_n$ and $R \in \mathbb{C}^{n \times n}$ is a triangular matrix).
- **Second step** : SVD decomposition of R as $R = U_R \Sigma_R V_R^H$.
- **Final step** : $A = U \, \Sigma_R \, V^H$ with $U = QU_R$ and $V = V_R$.
- **Complexity** : $4mn^2 + 22n^3$.
- **Parallel performance** : Tall and Skinny QR ($TSQR$) algorithm [Demmel et al, 2012] to be favored for the first step to obtain parallel performance.

Outline

Polar decomposition

Polar decomposition [Autonne, 1902]

- Given $A \in \mathbb{C}^{m \times n}$ with $p = \min(m, n)$, the polar decomposition of A reads :

$$A = W\ H,$$

  where $W \in \mathbb{C}^{m \times n}$ has orthonormal columns and $H \in \mathbb{R}^{n \times n}$ is Hermitian positive semidefinite.
- Relation with the SVD decomposition of A :

$$A = W\ H = W(V_H\ \Sigma_H\ V_H^H) = (WV_H)\ \Sigma_H\ V_H^H.$$

- **Interest** : efficient **parallel iterative** methods are available to first compute the polar decomposition and then deduce the SVD decomposition of A.

QR-based Dynamically Weighted Halley (QDWH)

QDWH [Nakatsukasa et al, 2010] [Householder prize, 2014]

- Given $A \in \mathbb{C}^{m\times n}$ with $p = \min(m,n)$, the QR-based Dynamically Weighted Halley ℓ-th iteration ($\ell \geq 1$) reads (with $X_\ell \in \mathbb{C}^{m\times n}$) :

$$\begin{bmatrix} \sqrt{c_\ell}\, X_\ell \\ I_n \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \quad X_{\ell+1} = \frac{b_\ell}{c_\ell} X_\ell + \frac{1}{\sqrt{c_\ell}}(a_\ell - \frac{b_\ell}{c_\ell}) Q_1 Q_2^H, \ell \geq 0.$$

  with $Q_1 \in \mathbb{C}^{m\times m}$, $Q_2 \in \mathbb{C}^{n\times m}$ ($Q_1^H Q_1 = I_m$, $Q_2^H Q_2 = I_m$), $R \in \mathbb{R}^{m\times n}$ upper triangular and a_ℓ, b_ℓ, c_ℓ parameters chosen dynamically to optimise the convergence rate. $X_0 = \frac{A}{\alpha}$ with $\alpha = \|A\|_2$.
- The polar factor W is obtained as the limit of the sequence X_ℓ. H is deduced as $H = \frac{1}{2}(W^H A + (W^H A)^H)$.
- The sequence is usually converging very fast in practice in double precision arithmetic for any matrix A with $\kappa_2(A) \leq 10^{16}$).
- **Complexity** : $O(mnp)$

Outline

UTV

UTV [Stewart, 1992]

- Given $A \in \mathbb{C}^{m \times n}$ with $p = \min(m, n)$, the UTV factorization of A reads :

$$A = U\ T\ V^H,$$

  with $U \in \mathbb{C}^{m \times m}$, $V \in \mathbb{C}^{n \times n}$ both unitary ($U^H U = I_m$, $V^H V = I_n$) and $T \in \mathbb{R}^{m \times n}$ is (lower or upper) triangular.
- Algorithm that is possible to stop once a specified tolerance has been met.
- Provides close to optimal low rank approximations in practice.
- **Complexity** : $O(mnk)$ for a rank-k approximation of A.
- **Parallel performance** : **good** due to blocking (matrix-matrix operations).

Outline

Comparison of matrix decomposition algorithms

Synopsis

	SVD	R-SVD	QDWH	UTV
Arithmetic cost		$O(mnp)$		
Backward stability	✓	✓	✓	✓
Ease of parallelization	Hard	Fairly easy	Easy	Easy
Min. communication ?	×	×	✓	×
Partial factorization	Not easily	Yes but not useful	×	✓
Useful for low rank approximation	✓	✓	✓	✓

Outline

Afternoon session

- Implement in julia a simple communication-minimizing factorization algorithm **Cholesky-QR** to be used when the matrix is tall and skinny.
- This algorithm provides a **QR** factorization.
- An **SVD** factorization can be then deduced easily (as in the **R-SVD** algorithm).

- **Interest** : discover how to use of **Map**/**Reduce** strategies.
- Study **robustness** for synthetic problems with variable singular gap.
- Study **performance** on your dataset if time permits.
- Conclusions to be shared !

Suggested reading

Suggested reading

- **G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz.** *Communication lower bounds and optimal algorithms for numerical linear algebra* , Acta Numerica. Cambridge University Press, 23, 1-155, 2014.
- **J. Demmel, L. Grigori, M. F. Hoemmen, and J. Langou**. *Communication-optimal parallel and sequential QR and LU factorizations*, SIAM Journal on Scientific Computing, Vol. 34, No 1, 2012.
- **N. Higham**, *Accuracy and Stability of Numerical Algorithms*, SIAM, Second edition, 2002.
- Talk of **L. Grigori** at Collège de France : `http://www.college-de-france.fr/site/pierre-louis-lions/seminar-2014-01-10-11h15.htm`

Outline

Your notes

Outline

Objectives and key idea

Objectives

- Review **the Lanczos bidiagonalization method** that is **useful** for low rank approximations of A when A is either sparse or structured or when only the action of A and of A^H on a vector is available.
- Put emphasis on specific important features of the algorithm.
- Focus on **parallel** properties of the algorithms.

Key idea

- Implicitly construct a rank-k approximation of A by solving a projected problem of reduced dimension on a particularly relevant subspace, called the Krylov subspace $\mathcal{K}_{k+1}(A, v_1)$.

Outline

Partial Lanczos bidiagonalization

Partial Lanczos bidiagonalization [Björck, 1996] - decomposition

- Given $v_1 \in \mathbb{C}^n$ with $\|v_1\|_2 = 1$ and $A \in \mathbb{C}^{m\times n}$, the Lanczos bidiagonalization algorithm **implicitly** leads to the decomposition ($k \geq 1$) :

$$\begin{aligned} AV_k &= U_k B_k, \\ A^H U_k &= V_k B_k^T + \beta_k v_{k+1} e_k^T \end{aligned}$$

with $U_k \in \mathbb{C}^{m\times k}$, $V_k \in \mathbb{C}^{n\times k}$ with orthonormal columns ($U_k^H U_k = I_k$, $V_k^H V_k = I_k$), $B_k \in \mathbb{R}^{k\times k}$ being upper bidiagonal and $v_{k+1} \in \mathbb{C}^n$

$$B_k = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ & \alpha_2 & \ddots & \\ & & \ddots & \beta_{k-1} \\ & & & \alpha_k \end{bmatrix}.$$

Partial Lanczos bidiagonalization method - algorithm (basic version)

Input : $A \in \mathbb{C}^{m \times n}$, $v_1 \in \mathbb{C}^n$ with $\|v_1\|_2 = 1$
Output : Partial Lanczos bidiagonal decomposition with $U_k = [u_1, \cdots, u_k] \in \mathbb{C}^{m \times k}$, $V_{k+1} = [v_1, \cdots, v_{k+1}] \in \mathbb{C}^{n \times k}$ with orthonormal columns and $B_k \in \mathbb{R}^{k \times k}$ upper bidiagonal

$\beta_0 = 0, u_0 = 0$
for $j = 1, k$ **do**
  $u_j = Av_j - \beta_{j-1} u_{j-1}$
  $\alpha_j = \|u_j\|_2$
  $u_j = u_j / \alpha_j$
  $v_{j+1} = A^H u_j - \alpha_j v_j$
  $\beta_j = \|v_{j+1}\|_2$
  $v_{j+1} = v_{j+1} / \beta_j$
end for

Simple algorithm ...

Partial Lanczos bidiagonalization method - algorithm (basic version)

Input : $A \in \mathbb{C}^{m \times n}$, $v_1 \in \mathbb{C}^n$ with $\|v_1\|_2 = 1$
Output : Partial Lanczos bidiagonal decomposition with $U_k = [u_1, \cdots, u_k] \in \mathbb{C}^{m \times k}$, $V_{k+1} = [v_1, \cdots, v_{k+1}] \in \mathbb{C}^{n \times k}$ with orthonormal columns and $B_k \in \mathbb{R}^{k \times k}$ upper bidiagonal

$\beta_0 = 0, u_0 = 0$
for $j = 1, k$ **do**
  $u_j = Av_j - \beta_{j-1} u_{j-1}$
  $\alpha_j = \|u_j\|_2$
  $u_j = u_j / \alpha_j$
  $v_{j+1} = A^H u_j - \alpha_j v_j$
  $\beta_j = \|v_{j+1}\|_2$
  $v_{j+1} = v_{j+1} / \beta_j$
end for

Simple algorithm ... that is prone to roundoff error propagation due to loss of orthogonality in U_k and V_k.

Outline

Main features of the Lanczos bidiagonalization method

- The Lanczos bidiagonalization method delivers a low rank approximation with a complexity that linearly depends on the number of iterations k.
- Only products with A or A^H and matrix-vector operations are required, which leads to a fairly easy parallel implementation.
- The storage is quite reduced.

Drawbacks of the Lanczos bidiagonalization method

- The Lanczos bidiagonalization method is prone to **roundoff error propagation**. This leads to a loss of orthogonality in U_k and V_k.
- A first cure is to use **selective or complete reorthogonalization** techniques to limit the roundoff error propagation during the algorithm.
- A second cure consists of stopping the algorithm after a certain number of iterations and of restarting by exploiting meaningful information (**thick restarting**).

Partial Lanczos bidiagonalization method with FULL orthogonalization [changes with respect to the basic version are highlighted in color]

for $j = 1, k$ **do**
  $u_j = A v_j$
  for $i = 1, j-1$ **do**
    $\gamma = u_i^H u_j$
    $u_j = u_j - \gamma\, u_i$
  end for
  $\alpha_j = \|u_j\|_2$
  $u_j = u_j / \alpha_j$
  $v_{j+1} = A^H u_j$
  for $i = 1, j$ **do**
    $\gamma = v_i^H v_{j+1}$
    $v_{j+1} = v_{j+1} - \gamma\, v_i$
  end for
  $\beta_j = \|v_{j+1}\|_2$
  $v_{j+1} = v_{j+1} / \beta_j$
end for

Partial Lanczos bidiagonalization method with ONE-SIDED orthogonalization [changes with respect to the basic version are highlighted in color]

$\beta_0 = 0, u_0 = 0$
for $j = 1, k$ **do**
  $u_j = Av_j - \beta_{j-1} u_{j-1}$
  $\alpha_j = \|u_j\|_2$
  $u_j = u_j / \alpha_j$
  $v_{j+1} = A^H u_j$
  for $i = 1, j$ **do**
    $\gamma = v_i^H v_{j+1}$
    $v_{j+1} = v_{j+1} - \gamma v_i$
  end for
  $\beta_j = \|v_{j+1}\|_2$
  $v_{j+1} = v_{j+1} / \beta_j$
end for

Outline

Partial Lanczos bidiagonalization method with thick restarting and reorthogonalization

- They must be **favored** with respect to standard Lanczos bidiagonalization methods !
- Their **parallel performance** is however limited by the reorthogonalization procedure, which can be costly in a massively parallel environment.
- Their **complexity** heavily depends on the repartition of the leading singular values of A.
- **Software** : SLEPc.

Outline

Krylov Golub-Kahan decomposition

Krylov Golub-Kahan decomposition [Stoll, 2012]

- Given $v_1 \in \mathbb{C}^n$ of unit Euclidean norm, we consider the **Lanczos bidiagonalization** decomposition :

$$\begin{aligned} AV_k &= U_k B_k, \\ A^H U_k &= V_k B_k^T + \beta_k v_{k+1} e_k^T. \end{aligned}$$

- We perform the **SVD** decomposition of the small bidiagonal matrix $B_k \in \mathbb{R}^{k \times k}$ as $B_k = P_k \Sigma_k Q_K^T$.
- This leads to the **Krylov Golub-Kahan** decomposition :

$$\begin{aligned} A\widetilde{V}_k &= \widetilde{U}_k \Sigma_k, \\ A^H \widetilde{U}_k &= \widetilde{V}_k \Sigma_k + \beta_k v_{k+1} p_k^T \end{aligned}$$

with $\widetilde{U}_k = U_k P_k, \widetilde{U}_k^H \widetilde{U}_k = I_k$, $\tilde{V}_k = V_k Q_k, \widetilde{V}_k^H \widetilde{V}_k = I_k$, $p_k^T = e_k^T P_k$ and $\Sigma_k \in \mathbb{R}^{k \times k}$ being **diagonal**.

Krylov Golub-Kahan decomposition

Krylov Golub-Kahan decomposition [Stoll, 2012]

- The **Krylov Golub-Kahan** decomposition is **extremely** convenient for implementing **deflation**, an important feature to improve convergence that can be tedious to implement in other decompositions.
- **Deflation** : if l singular values are of interest, we decide to **lock** (**keep**) them, otherwise we **purge** them. This corresponds to a simple permutation matrix Π of Σ_k as :

$$\widehat{\Sigma}_k = \Pi^T \Sigma_k \Pi = diag(\sigma_1, \cdots, \sigma_l, \cdots, \sigma_k), \ (l \leq k)$$

- This then leads to the **Krylov Golub-Kahan** shrinked decomposition :

$$\begin{aligned} A\widehat{V}_l &= \widehat{U}_l \widehat{\Sigma}_l, \\ A^H \widehat{U}_l &= \widehat{V}_l \widehat{\Sigma}_l + \beta_l v_{l+1} \widehat{p}_l^T \end{aligned}$$

- The factorization can then be expanded from dimension l to k.

Suggested reading

Suggested reading

- **A. Björck**. *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.
- **V. Hernandez, J. Roman, A. Tomas**. *Restarted Lanczos Bidiagonalization for the SVD in SLEPc*, SLEPc Technical Report STR-8, 2007. Available at `http://slepc.upv.es/documentation/reports/str8.pdf`.

Outline

Your notes

Outline

Objectives and key idea

Objectives

- Why selecting this approach ? : **Much better performance** in terms of parallelization for the approaches based on the symmetric eigenvalue decomposition can be expected with respect to standard factorization methods.
- Discuss both **sparse and dense** aspects of these methods.
- Focus on parallel properties of the algorithms.

Key idea

- Deduce all or selected singular values/vectors of A as the result of a standard eigenproblem to be detailed.

Outline

Formulation of the matrix eigenvalue problem

Cross-product formulation

- **Idea** : to retrieve a low rank approximation by solving a standard Hermitian eigenvalue problem for which efficient deterministic parallel methods are available.
- Given $A \in \mathbb{C}^{m \times n}$ with $m \geq n$, the **cross-product eigenvalue formulation** reads :
$$A^H A\, x = \lambda x,$$
where $x \in \mathbb{C}^n$ is an eigenvector of $A^H A$ associated with the eigenvalue $\lambda \in \mathbb{R}^+$. $A^H A$ is called the Gram matrix.
- This yields $A^H A V = V\, \Lambda$ with $\Lambda \in \mathbb{R}^{k \times k}$ diagonal.
- Singular values are given by $\lambda_i = \sigma_i^2$.
- Case of $m \leq n$: $A A^H\, x = \lambda x$.

Formulation of the matrix eigenvalue problem

Cyclic formulation

- **Idea** : to retrieve a low rank approximation by solving a Hermitian eigenvalue problem for which efficient deterministic parallel methods are available.
- Given $A \in \mathbb{C}^{m \times n}$ with $m \geq n$, the **cyclic eigenvalue formulation** reads :

$$\begin{bmatrix} 0_{m\times m} & A \\ A^H & 0_{n\times n} \end{bmatrix} \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \lambda_i \begin{bmatrix} u_i \\ v_i \end{bmatrix},$$

  where $\begin{bmatrix} u_i \\ v_i \end{bmatrix} \in \mathbb{C}^{m+n}$ is an eigenvector of the augmented matrix C associated with the eigenvalue $\lambda_i \in \mathbb{R}$.
- This yields $CV_C = V_C\Lambda$ with $\Lambda \in \mathbb{R}^{k\times k}$ diagonal.
- $\lambda_i(C) = \sigma_i(A) = -\lambda_{n+m-i+1}(C)$ and 0.

Outline

Standard algorithm for the Hermitian eigendecomposition : Householder tridiagonalization

- **First step** : Reduction of $C \in \mathbb{C}^{n \times n}$ to **tridiagonal** form $H_L \; C \; H_R = T$

$$C \xrightarrow{H_1} \begin{bmatrix} \star & \star & & \\ \star & \star & \star & \star \\ & \star & \star & \star \\ & \star & \star & \star \end{bmatrix} \xrightarrow{H_2} \begin{bmatrix} \star & \star & & \\ \star & \star & \star & \\ & \star & \star & \star \\ & & \star & \star \end{bmatrix} \equiv \quad T$$

  where H_i indicates a **two-sided Householder** transformation. At the end of this step we have $C = HTH^H$ with :

$$H = H_{n-2} \cdots H_1 .$$

- **Second step** : Eigendecomposition of T as $T = Q_T \Lambda Q_T^H$
- **Final step** : $C = (HQ_T) \; \Lambda \; (HQ_T)^H$.
- **Complexity** : $O(n^3)$.
- **Parallel performance** : relatively low in the first step and **high** in the second step (Divide and conquer [Cuppen, 1981], MRRR [Dhillon et al, 2006]).

Outline

Subspace iteration method

Subspace iteration method

Input : $\ell > 1$, $C \in \mathbb{C}^{n\times n}$, $V_1 \in \mathbb{C}^{n\times k}$ with $V_1^H V_1 = I_k$
Output : Orthonormal basis $V_\ell \in \mathbb{C}^{n\times k}$
for $j = 1, \ell - 1$ **do**
  $W_j = CV_j$
  Compute the QR decomposition of W_j as $W_j = V_{j+1} R_{j+1}$
end for

- Eigenvalue extraction from Galerkin condition $Cv - \mu v \perp \mathcal{V}_\ell, v \in \mathcal{V}_\ell$
- μ is an eigenvalue of the $k \times k$ matrix $V_\ell^H C V_\ell$
- $v = V_\ell w$ with $\|w\|_2 = 1$, eigenvector of $V_\ell^H C V_\ell$ associated with μ.
- μ and v are called Ritz value and Ritz vector, respectively.
- The basic subspace iteration extracts k Ritz pairs which are close to the dominant eigenvalues/eigenvectors of C.

Outline

Lanczos tridiagonalization

Lanczos method (basic version)

Input : $C \in \mathbb{C}^{n \times n}$ or the action of C on a vector, $v_1 \in \mathbb{C}^n$ with $\|v_1\|_2 = 1$
Output : orthonormal basis $V_{k+1} = [v_1, \cdots, v_{k+1}]$ of $\mathcal{K}_{k+1}(C, v_1)$

for $j = 1, k$ **do**
  $z_j = Cv_j$
  $\alpha_j = v_j^H z_j$
  $\tilde{v}_{j+1} = z_j - \alpha_j v_j$
  if $j > 1$ **then**
    $\tilde{v}_{j+1} = \tilde{v}_{j+1} - \beta_{j-1} v_{j-1}$
  end if
  $\beta_j = \|\tilde{v}_{j+1}\|_2$
  $v_{j+1} = \tilde{v}_{j+1} / \beta_j$
end for

Lanczos tridiagonalization

Lanczos decomposition

- The Lanczos algorithm leads to the decomposition :

$$CV_k = V_k H_k + \widehat{\beta}_k v_{k+1} e_k^T$$

with H_k being symmetric and tridiagonal :

$$H_k = \begin{bmatrix} \widehat{\alpha}_1 & \widehat{\beta}_1 & & \\ \widehat{\beta}_1 & \widehat{\alpha}_2 & \ddots & \\ & \ddots & \ddots & \widehat{\beta}_{k-1} \\ & & \widehat{\beta}_{k-1} & \widehat{\alpha}_k \end{bmatrix}.$$

- The eigenpairs (μ_j, w_j) of $H_k = V_k^H C V_k$ are called **Ritz pairs** (Ritz values and Ritz vectors, respectively). They provide approximate spectral information of C.
- Convergence of Ritz pairs (which part of the spectrum ?) + difficulties

Lanczos tridiagonalization

Lanczos decomposition with complete/selective reorthogonalization

- Complete reorthogonalization is an effective but **expensive** cure
- Require to store the complete basis V_k (i.e. k vectors)
- The computational cost grows from $O(mvp + nk)$ to $O(mvp + nk^2)$
- Paige (1990) : Consider a k-order Lanczos decomposition computed in floating point arithmetic with machine precision ε_{mach}. The Ritz pairs $(\mu_1, w_1), \cdots, (\mu_k, w_k)$ satisfy

$$w_i^H w_{k+1} = \frac{O(\varepsilon_{mach} \|C\|_2)}{\|r_i\|_2}, \quad i = 1, \cdots, k$$

  with $r_i = Cw_i - \mu_i w_i$
- This suggests the use of **selective reorthogonalization** only versus converged Ritz pairs (within $\sqrt{\varepsilon}$).

Outline

Contour integration spectrum slicing (CISS) methods

CISS [Sakurai and Sugiura [2003], Polizzi [2009]]

- **Locate** and **compute** the **eigenvalues** within a given region of interest with contour $\mathcal{C}$.
- Indicator function of $\mathcal{C}$: $f(z) = -\frac{1}{2i\pi}\int_{\mathcal{C}}(\mu - z)^{-1}d\mu, \quad z \notin \mathcal{C}$.

$$f(z) = 1, z \in \mathcal{C}, \qquad f(z) = 0, \quad \text{otherwise}.$$

- Numerical approximation of the spectral projector $D = -\frac{1}{2i\pi}\int_{\mathcal{C}}(zI_n - C)^{-1}dz$ by Gauss quadrature

$$\widehat{D} = \sum_{j=0}^{N} w_j\,(z_j\,I_n - C)^{-1},$$

where $N+1$ is the number of contour points, z_j the quadrature points on $\mathcal{C}$ and w_j the quadrature weights, respectively.

Contour integration spectrum slicing (CISS) methods

Algorithm (Hermitian case) formulated as a filtered subspace iteration

- We define the density matrix (spectral projector) as : $D = -\frac{1}{2\pi i}\int_C G(z)dz$ with $G(z) = (zI_n - C)^{-1}$.
- (a) Pick $Y_{n\times M} = [y_1, \cdots, y_M]$ M **random** vectors of $\mathbb{C}^n$.
- (b) Compute Q an approximation of $D\ Y_{n\times M}$ by numerical integration :

$$Q = \sum_{j=0}^{N} w_j\ (z_j I_n - C)^{-1} Y_{n\times M}.$$

- (c) Solve the **projected** generalized Hermitian eigenproblem (of size $M \times M$) :

$$Q^H C Q\ p_i = \lambda_i (Q^H Q)\ p_i$$

  with $(\lambda_i, x_i = Qp_i)$ is a putative eigenpair of C.
- Check if $\lambda_i \in \mathcal{C}$, and go back to step (b) using $Y = X = [x_1, \cdots, x_M]$ if needed.

Contour integration spectrum slicing (CISS) methods

Main properties

- **Fast and systematic convergence** in the Hermitian case : using 8 to 16 contour points, the algorithm converges in $2-3$ iterations only to obtain up to **thousands** of eigenpairs (if exist) with machine accuracy.
- Naturally captures **all** multiplicities.
- **No** (explicit) orthogonalization procedure required.
- **Natural** parallelism at various levels ($\mathcal{C}$, $N+1$ contour points, solution of linear systems).
- Allow the use of (**parallel**) iterative methods for solving the complex linear systems.
- **Drawback** : use of complex arithmetic for solving a symmetric real-valued eigenproblem (cross-product formulation).

Outline

Afternoon session

- Experiment the two different strategies for the eigendecomposition (**Lanczos** based or **CISS** based) with two different codes that are publicly available.
- Synthetic test matrices are provided for easy testing.
- Study **robustness** for synthetic problems with variable singular gap.
- Study **performance** on your dataset if time permits.
- **Requirements** : your matrix must be stored in **HDF5** or **MatrixMarket** formats.
- Conclusions to be shared !

Suggested reading

Suggested reading

- **Z. Bai et al.** *Templates for the Solution of Algebraic Eigenvalue Problems : A Practical Guide.*, SIAM, 2000.
- **Y. Saad**. *Numerical Methods for Large Eigenvalue Problems*, SIAM, 2011.
- **E. Polizzi**. *Density-matrix-based algorithms for solving eigenvalue problems*, Phys. Rev. B, 79 :115112, 2009.
- **T. Sakurai and H. Sugiura**. *A projection method for generalized eigenvalue problems*, J. Comput. Appl. Math., Vol. 159, pp. 119-128, 2003.
- **P. T. P. Tang and E. Polizzi**. *FEAST as a subspace iteration eigensolver accelerated by approximate spectral projection*. SIAM J. Matrix Anal. Appl., 35(2) :354–390, 2014

Outline

Your notes

Outline

CANDMC

CANDMC (Communication Avoiding Numerical Dense Matrix Computations)

- `https://github.com/solomonik/CANDMC`
- E. Solomonik (Univerity of Illinois, USA)
- **Dense linear algebra** software
- Special focus on **communication avoiding** algorithms (LU, QR and symmetric eigendecomposition)
- Implementation of TSQR algorithm
- Written in C++
- Last version : 2016, BSD licence
- **Householder prize in 2017**

Chameleon

Chameleon

- https://project.inria.fr/chameleon/
- Joint project : ICL (University of Tenessee), INRIA, KAUST, University of Colorado
- **Dense linear algebra** software relying on sequential task-based algorithms where subtasks of the overall algorithms are submitted to a runtime system
- **General paradigm** (Direct Acyclic Graph (DAG)) used on very different type of architectures : laptop, many-core nodes, CPUs-GPUs, multiple nodes
- Chameleon is able to perform a Cholesky factorization (double-precision) at 80 TFlop/s on a dense matrix of order 400000
- Written in C++, C and Python
- Last version : 0.9.0 in June 2016, Cecill-C licence

DPLASMA

DPLASMA

- `https://www.icl.utk.edu/dplasma`
- ICL (University of Tenessee)
- **Dense linear algebra** software relying on sequential task-based algorithms
- **General paradigm** (Direct Acyclic Graph (DAG)) used on very different type of architectures
- Cholesky, QR and TSQR factorizations
- Written in Fortran, C, C++
- Last version : 1.2.0 in May 2014

Outline

Elemental

Elemental

- `https://www.libelemental.org`
- Elemental is open-source, openly-developed, software for distributed-memory **dense and sparse-direct** linear algebra and optimization which supports a wide range of functionality not available elsewhere.
- Support for “double-double”, “quad-double”, quad-precision, and arbitrary-precision floating-point arithmetic.
- **Research oriented** software with a focus on recent algorithms.
- General software (decomposition, SVD and eigendecomposition).
- Written in C++.
- Last version : 0.87.7 in February 2017.

IRLBA

IRLBA (Implicitly Restarted Lanczos Bidiagonalization Algorithm)

- `https://cran.r-project.org/web/packages/irlba/`
- Implementation of the algorithm proposed in [Baglama and Reichel, 2005]
- **Dense and sparse** matrices are considered
- Lanczos bidiagonalization with selective reorthogonalization and thick restarting
- Distributed memory implementation
- Written in R.
- Python version available at `https://github.com/bwlewis/irlbpy`
- Last version : 2.2.1 in May 2017, GPL3 license.

QR_MUMPS

QR_MUMPS

- `http://buttari.perso.enseeiht.fr/qr_mumps/`
- A. Buttari (IRIT, Toulouse)
- Applicable to **sparse** matrices.
- Parallel, multithreaded software based on the StarPU runtime engine.
- Asynchronous and dynamic data-flow programming model which provides high efficiency and scalability.
- Written in Fortran.
- Last version : 2.0 in June 2016.

Outline

ARPACK

ARPACK (Arnoldi Package)

- `http://www.caam.rice.edu/software/ARPACK/`
- R. Lehoucq, K. Maschhoff, D. Sorensen and C. Yang, Rice University, USA.
- **Sparse and dense linear algebra**. Include routines for the SVD.
- Based on reverse communication interface.
- Written in Fortran 77.
- Last version : BSD license.
- Current suported library : https ://github.com/opencollab/arpack-ng

ELPA

ELPA (Eigenvalue SoLvers for Petaflop-Applications)

- `https://elpa.mpcdf.mpg.de/`
- Joint project in Germany (Max Planck Gesellschaft and several universities).
- **Dense linear algebra**.
- Provide **highly efficient and highly scalable** direct eigensolvers for symmetric matrices based on standard algorithms.
- Target **massively parallel** architectures.
- Written in Fortran (C/C++ interface available).
- Last version : 2017.05 in May 2017, LGPL license.

FEAST

FEAST

- http://www.feast-solver.org/
- University of Amherst, USA
- Contour Integral Spectrum slicing method
- **Dense and sparse linear algebra**
- Free high-performance numerical library for solving the Hermitian and non-Hermitian eigenvalue problems, and obtaining all the eigenvalues and (right/left) eigenvectors within a given search interval or arbitrary domain in the complex plane
- It includes flexible reverse communication interfaces and ready to use predefined interfaces for dense, banded and sparse systems.
- Versions for shared and distributed memory platforms
- Written in Fortran
- Last version : 3.0 in June 2015, BSD license.

SLEPc

SLEPc (Scalable Library for Eigenvalue Problem Computations)

- `http://slepc.upv.es/`
- University Politècnica de València, Spain
- **Sparse linear algebra**
- Software library for the solution of large scale sparse eigenvalue problems on parallel computers. It can also be used for computing a partial SVD of a large, sparse, rectangular matrix.
- Extension of PETSc `http://www.mcs.anl.gov/petsc/`.
- Versions based on the PETSc data structures which employs the MPI standard for message-passing communication.
- Main language : C.
- Last version : slepc-3.7.4 in May 2017, LGPL license.

Spectra and RSpectra

Spectra (Sparse Eigenvalue Computation Toolkit as a Redesigned ARPACK)

- `http://spectralib.org/`
- C++ library for large scale eigenvalue problems, built on top of Eigen `http://eigen.tuxfamily.org`, an open source linear algebra library.
- Appropriate for the computation of few eigenvalues and corresponding eigenvectors of large and sparse matrices based on the Implicitly Restarted Arnoldi Method
- **Dense and sparse linear algebra**
- Available in R as RSpectra `https://cran.r-project.org/web/packages/RSpectra/index.html`
- Partial SVD is also provided ('svds' function) in RSpectra
- `https://bwlewis.github.io/irlba/comparison.html`
- Last version : 0.12-0 in June 2016, MPL2 license.

z-Pares

z-Pares

- `http://zpares.cs.tsukuba.ac.jp/`
- University of Tsukuba, Japan.
- Contour Integral Spectrum slicing method.
- z-Pares is designed to compute eigenvalues and eigenvectors of sparse or dense matrices.
- Single precision and double precision are supported.
- Versions for distributed memory platforms.
- Written in Fortran 90/95.
- Last version : v0.9.6a in October 2014.

Outline

Your notes

Outline

Conclusions

Summary

- We have first reviewed popular **fundamental rank-revealing matrix decompositions** (Section 2).
- We have focused on **deterministic methods** for low rank approximations with an emphasis on parallel methods (Sections 3 and 4).
- We have shortly described a few related **parallel software libraries** (Section 5).
- We have mostly focused on the **SVD** due to its optimal approximation property. Other deterministic **close to optimal** algorithms have been proposed (two popular examples follow).

Deterministic algorithms for dimensionality reduction

Nonnegative matrix factorization (NMF) [Lee et al, 1999]

- **Goal** : retain both **sparseness** and **interpretability** in the factorization, contrary to the SVD which leads to dense factors.
- **Idea** : Impose a particular constraint in the factored form of A (e.g. nonnegativity, sparsity, weights, regularization or restriction to nonzero entries).
- **Example** : Find $C \in \mathbb{C}^{m\times k}$ and $H \in \mathbb{C}^{k\times n}$ such that :

$$\min_{C,H} \|A - CH\|_F, \quad C, H \geq 0,$$

  i.e. C, H are entry-wise nonnegative matrices.
- This leads to a **non-convex** optimization problem.
- C is usually selected as a subset of the columns of A, if A has nonnegative entries.
- Applications in image processing, medical imaging, astronomy.

Deterministic algorithms for dimensionality reduction

Skeleton decomposition : CUR/CX factorizations [Mahoney et al, 2009]

- **Idea** : Impose a particular structure for the factored form of A expressed in terms of a small number of actual columns/rows of A. If A is sparse, this keeps sparseness !
- **CZ factorization** : Find $C \in \mathbb{C}^{m \times k}$ and $Z \in \mathbb{C}^{k \times n}$ such that :

$$\min_{C,Z} \|A - CZ\|_F.$$

- C is usually selected as a relevant subset of the columns of A.
- Current algorithms lead to the upper bound : $\|A - CZ\|_F \leq (1+\varepsilon)\|A - A_k\|_F$, where A_k is the best rank-k approximation of A and ε positive.
- Applications in astronomy, genetics, mass spectrometry imaging.

Outline

Algorithms for dimensionality reduction

This is an active research area from different perspectives

- **Linear algebra** : Randomized algorithms (see the lecture of Pierre Blanchard).

Algorithms for dimensionality reduction

This is an active research area from different perspectives

- **Linear algebra** : Randomized algorithms (see the lecture of Pierre Blanchard).
- **Multilinear algebra** : Low-rank tensor approximations (canonical polyadic factorization and Tucker decomposition).

Algorithms for dimensionality reduction

This is an active research area from different perspectives

- **Linear algebra** : Randomized algorithms (see the lecture of Pierre Blanchard).
- **Multilinear algebra** : Low-rank tensor approximations (canonical polyadic factorization and Tucker decomposition).
- **Numerical optimization** : Solve the minimization problem in large dimension with deterministic or stochastic gradient descent formulations.

Algorithms for dimensionality reduction

This is an active research area from different perspectives

- **Linear algebra** : Randomized algorithms (see the lecture of Pierre Blanchard).
- **Multilinear algebra** : Low-rank tensor approximations (canonical polyadic factorization and Tucker decomposition).
- **Numerical optimization** : Solve the minimization problem in large dimension with deterministic or stochastic gradient descent formulations.
- **Machine learning** : Current strong activity on **deep learning** algorithms for dimensionality reduction.

Algorithms for dimensionality reduction

This is an active research area from different perspectives

- **Linear algebra** : Randomized algorithms (see the lecture of Pierre Blanchard).
- **Multilinear algebra** : Low-rank tensor approximations (canonical polyadic factorization and Tucker decomposition).
- **Numerical optimization** : Solve the minimization problem in large dimension with deterministic or stochastic gradient descent formulations.
- **Machine learning** : Current strong activity on **deep learning** algorithms for dimensionality reduction.
- **Statistical** learning algorithms.

Algorithms for dimensionality reduction

This is an active research area from different perspectives

- **Linear algebra** : Randomized algorithms (see the lecture of Pierre Blanchard).
- **Multilinear algebra** : Low-rank tensor approximations (canonical polyadic factorization and Tucker decomposition).
- **Numerical optimization** : Solve the minimization problem in large dimension with deterministic or stochastic gradient descent formulations.
- **Machine learning** : Current strong activity on **deep learning** algorithms for dimensionality reduction.
- **Statistical** learning algorithms.

Thank you for your attention !

Bibliography I

J. Baglama and L. Reichel.
Augmented implicitly restarted Lanczos bidiagonalization methods.
SIAM J. Sci. Comput., 27-1 :19-42, 2005.

A. Björck.
Numerical Methods for Least Squares Problems.
SIAM, Philadelphia, 1996.

T. Chan.
An improved algorithm for computing the Singular Value Decomposition.
ACM Trans. Math. Softw., 8 :1, 84-88, 1982.

J. Chiu and L. Demanet.
Sublinear randomized algorithms for skeleton decomposition.
SIAM J. Matrix Anal. Appl., 34 :3, 1361-1383, 2013.

J. Cuppen.
A divide and conquer method for the symmetric tridiagonal eigenproblem.
Numerische Mathematik, 36 :177-195, 1981.

Bibliography II

I. Dhillon and B. Parlett.
The Design and Implementation of the MRRR Algorithm.
ACM Transactions on Mathematical Software, 32 :4, 533–560, 2006.

C. Eckart and G. Young.
The approximation of one matrix by another of lower rank.
Psychometrika, 1 :211-8, 1936.

A. Gittens et al.
Matrix factorizations at scale : A comparison of scientific data analytics in Spark and C+ MPI using three case studies.
IEEE International Conference on Big Data, pp. 204-213, 2016.

G. Golub and W. Kahan.
Calculating the singular values and pseudo-inverse of a matrix.
Journal of the Society for Industrial and Applied Mathematics : Series B, Numerical Analysis. 2 (2) : 205–224, 1965.

Bibliography III

G. Golub, and C. Van Loan.
Matrix Computations.
Johns Hopkins University, fourth edition, 2012.

G. Golub and W. Kahan.
Calculating the singular values and pseudo-inverse of a matrix.
Journal of the Society for Industrial and Applied Mathematics : Series B, Numerical Analysis. 2 (2) : 205–224, 1965.

V. Hernandez, J. Roman and A. Tomas.
A robust and efficient parallel SVD solver based on restarted Lanczos bidiagonalization.
ETNA, 31, 68-85, 2008.

D. Lee and H. Seung.
Learning the parts of objects by non-negative matrix factorizations.
Nature, 401(6755), 788-791, 1999.

Bibliography IV

M. Mahoney and P. Drineas.
CUR matrix decompositions for improved data analysis.
Proc. Nat. Acad. Sci. USA, 106, 697-702, 2009.

Y. Nakatsukasa and R. W. Freund.
Computing Fundamental Matrix Decompositions Accurately via the Matrix Sign Function in Two Iterations : The Power of Zolotarev's Functions.
SIAM Review, 58 :3, 461-493, 2016.

E. Polizzi.
Density-Matrix-Based algorithms for solving eigenvalue problems.
Phys. Rev. B., 79 :115112, 2009.

H. Simon and H. Zha.
Low-rank matrix approximation using the Lanczos bidiagonalization process with applications.
SIAM J. Sci. Comput., 21-6 :2257-2274, 2000.

Bibliography V

G. W. Stewart.
An updating algorithm for subspace tracking.
IEEE Transactions on Signal Processing, 40 :6, 1535, 1992.

G. W. Stewart.
On the early history of the Singular Value Decomposition.
SIAM Review, 35 (4) : 551–566, 1993.

M. Stoll.
A Krylov–Schur approach to the truncated SVD.
Linear Algebra and its Applications, 436 :2795-2806, 2012.

L. N. Trefethen and D. Bau III.
Numerical linear algebra..
SIAM, Philadelphia, 1997.