Eigenvalues and Singular Values
Power Method and MATLAB use

Aleksandar Donev
Courant Institute, NYU[1]
donev@courant.nyu.edu

[1]Course MATH-UA.0252/MA-UY_4424, Spring 2021

Spring 2021

Outline

Outline

The need for iterative algorithms

- The eigenvalues are roots of the **characteristic polynomial** of **A**, which is generally of order n.
- According to Abel's theorem, there is no closed-form (rational) solution for $n \geq 5$.
  All eigenvalue algorithms must be iterative!
- There is an important distinction between iterative methods to:
  - Compute **all eigenvalues** (similarity transformations). These are based on dense-matrix factorizations such as the QR factorization, with total cost $O(n^3)$.
  - Compute **only one or a few eigenvalues**, typically the smallest or the largest one (e.g., power method). These are similar to iterative methods for solving linear systems.

Sparse Matrices

- Recall that for a diagonalizable matrix
$$\mathbf{A}^n = \mathbf{X}\boldsymbol{\Lambda}^n\mathbf{X}^{-1}$$
and **assume well-separated eigenvalues** $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \cdots |\lambda_n|$, and that the columns of $\mathbf{X}$ are normalized, $\|\mathbf{x}_j\| = 1$.
- For sparse matrices we sometimes only need to know a **few of the eigenvalues**/vectors, not all of them.
- Notably, knowing the eigenvector corresponding to the **smallest and largest (in magnitude) eigenvalues** is often most important (see Google Page Rank algorithm).

Iterative Method

- Any **initial guess** vector $\mathbf{q}_0$ can be represented in the linear basis formed by the eigenvectors

$$\mathbf{q}_0 = \mathbf{X}\mathbf{a}$$

- Recall iterative methods for linear systems: **Multiply a vector with the matrix A** many times:

$$\mathbf{q}_{k+1} = \mathbf{A}\mathbf{q}_k$$

$$\mathbf{q}_n = \mathbf{A}^n\mathbf{q}_0 = \left(\mathbf{X}\mathbf{\Lambda}^n\mathbf{X}^{-1}\right)\mathbf{X}\mathbf{a} = \mathbf{X}\left(\mathbf{\Lambda}^n\mathbf{a}\right)$$

Power Method

- As $n \to \infty$, the **eigenvalue of largest modulus** λ_0 will dominate,

$$\boldsymbol{\Lambda}^n = \lambda_1^n \text{Diag}\left\{1, \left(\frac{\lambda_2}{\lambda_1}\right)^n, \dots\right\} \to \text{Diag}\left\{\lambda_1^n, 0, \dots, 0\right\}$$

$$\mathbf{q}_n = \mathbf{X}\left(\boldsymbol{\Lambda}^n \mathbf{a}\right) \to \lambda_1^n \mathbf{X} \begin{bmatrix} a_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \lambda_1^n \mathbf{x}_1$$

- Therefore the **normalized iterates** converge to the eigenvector:

$$\tilde{\mathbf{q}}_n = \frac{\mathbf{q}_n}{\|\mathbf{q}_n\|} \to \mathbf{x}_1$$

- The **Rayleigh quotient** converges to the eigenvalue:

$$r_A\left(\mathbf{q}_n\right) = \frac{\mathbf{q}_n^\star \mathbf{A} \mathbf{q}_n}{\mathbf{q}_n \cdot \mathbf{q}_n} = \tilde{\mathbf{q}}_n^\star \mathbf{A} \tilde{\mathbf{q}}_n \to \lambda_1$$

Power Iteration

Start with an initial guess $\mathbf{q}_0$, and then iterate:

1. Compute **matrix-vector product** and normalize it:

$$\mathbf{q}_k = \frac{\mathbf{A}\mathbf{q}_{k-1}}{\left\|\mathbf{A}\mathbf{q}_{k-1}\right\|}$$

2. Use Raleigh quotient to obtain **eigenvalue estimate**:

$$\hat{\lambda}_k = \mathbf{q}_k^\star \mathbf{A}\mathbf{q}_k$$

3. **Test for convergence**: Evaluate the residual

$$\mathbf{r}_k = \mathbf{A}\mathbf{q}_k - \hat{\lambda}_k \mathbf{q}_k$$

and terminate if the residual norm is smaller than some tolerance, e.g., for tolerance $\epsilon \ll 1$,

$$\|\mathbf{r}_k\| \approx \left|\lambda_1 - \hat{\lambda}_k\right| \leq \epsilon \hat{\lambda}_k .$$

Eigenvalues in MATLAB

- The **Schur decomposition** is provided by $[U, T] = schur(A)$.
- In MATLAB, sophisticated variants of the **QR algorithm** (LAPACK library) are implemented in the function *eig*:

$$\Lambda = eig(A)$$

$$[X, \Lambda] = eig(A)$$

- For large or sparse matrices, iterative methods based on the **Arnoldi iteration** (ARPACK library), can be used to obtain a few of the largest eigenvalues:

$$\Lambda = eigs(A, n_{eigs})$$

$$[X, \Lambda] = eigs(A, n_{eigs})$$

Outline

Sensitivity (conditioning) of the SVD

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\star}$$

- Since unitary matrices have unit 2-norm,

$$\|\delta\Sigma\|_2 \approx \|\delta A\|_2 \,.$$

- The SVD computation is always **perfectly well-conditioned**!
- However, this refers to absolute errors: The **relative error** of small singular values will be large.
- The **power of the SVD** lies in the fact that it always exists and can be computed stably...but it is somewhat **expensive to compute**.

Computing the SVD

- The SVD can be computed by performing an eigenvalue computation for the **normal matrix $\mathbf{A}^\star\mathbf{A}$** (a positive-semidefinite matrix).
- This squares the condition number for small singular values and is **not numerically-stable**.
- Instead, modern algorithms use an algorithm based on computing eigenvalues / eigenvectors using the QR factorization.
- The cost of the calculation is $\sim O(mn^2)$, of the same order as eigenvalue calculation if $m \sim n$.

Reduced SVD

The **full (standard) SVD**

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\star} = \sum_{i=1}^{p} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\star}$$
$$[m \times n] = [m \times m]\,[m \times n]\,[n \times n]\,,$$

is in practice often computed in **reduced (economy) SVD** form, where $\boldsymbol{\Sigma}$ is $[p \times p]$:

$$[m \times n] = [m \times n]\,[n \times n]\,[n \times n] \quad \text{for} \quad m > n$$
$$[m \times n] = [m \times m]\,[m \times m]\,[m \times n] \quad \text{for} \quad n > m$$

This contains all the information as the full SVD but can be **cheaper to compute** if $m \gg n$ or $m \ll n$.

In MATLAB

- $[U, \Sigma, V] = svd(A)$ for **full SVD**, computed using a QR-like method.
- $[U, \Sigma, V] = svd(A,' econ')$ for **economy SVD**.
- The **least-squares solution** for square, overdetermined, underdetermined, or even rank-defficient systems can be computed using *svd* or *pinv* (pseudo-inverse, see homework).
- The q largest singular values and corresponding approximation can be computed efficiently for **sparse matrices** using

$$[U, \Sigma, V] = svds(A, q).$$

Outline

Low-rank approximations

- The SVD is a decomposition into **rank-1 outer product matrices**:

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\star} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\star} = \sum_{i=1}^{r} \mathbf{A}_i$$

- The rank-1 components $\mathbf{A}_i$ are called **principal components**, the most important ones corresponding to the larger σ_i.
- Ignoring all singular values/vectors except the first q, we get a **low-rank approximation**:

$$\mathbf{A} \approx \hat{\mathbf{A}}_q = \mathbf{U}_q \boldsymbol{\Sigma}_q \mathbf{V}_q^{\star} = \sum_{i=1}^{q} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\star}.$$

- Theorem: This is the **best approximation** of rank-q in the Euclidian and Frobenius norm:

$$\left\| \mathbf{A} - \hat{\mathbf{A}}_q \right\|_2 = \sigma_{q+1}$$

Applications of SVD/PCA

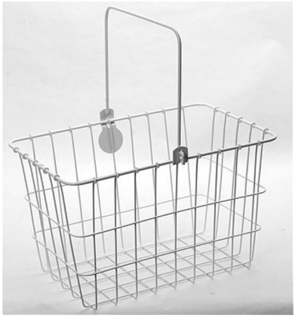
- **Statistical analysis** (e.g., DNA microarray analysis, clustering).
- Data **compression** (e.g., image compression, explained next).
- **Feature extraction**, e.g., face or character recognition (see Eigenfaces on Wikipedia).
- **Latent semantic indexing** for context-sensitive searching (see Wikipedia).
- **Noise reduction** (e.g., weather prediction).
- One example concerning language analysis given in homework.

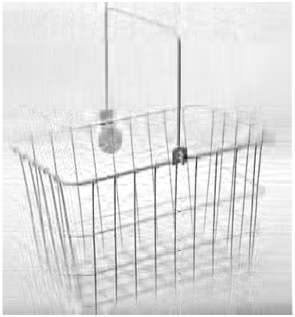
Image Compression

```
>> A=rgb2gray(imread('basket.jpg'));
>> imshow(A);
>> [U,S,V]=svd(double(A));
>> r=25; % Rank-r approximation
>> Acomp=U(:,1:r)*S(1:r,1:r)*(V(:,1:r))';
>> imshow(uint8(Acomp));
```

Compressing an image of a basket

We used only 25 out of the ~ 400 singular values to construct a rank 25 approximation:

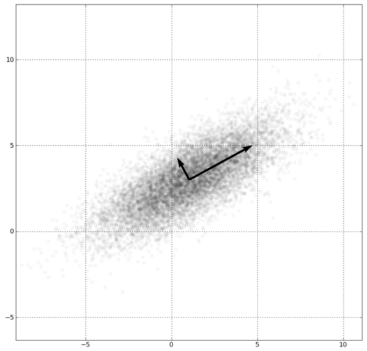




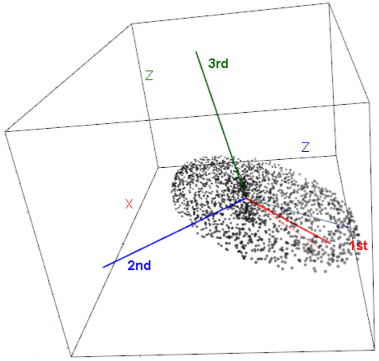
Principal Component Analysis

- **Principal Component Analysis** (PCA) is a term used for low-rank approximations in statistical analysis of data.
- Consider having m empirical data points or **observations** (e.g., daily reports) of n **variables** (e.g., stock prices), and put them in a **data matrix** $\mathbf{A} = [m \times n]$.
- Assume that each of the variables has **zero mean**, that is, the empirical mean has been subtracted out.
- It is also useful to choose the units of each variable (normalization) so that the **variance is unity**.
- We would like to find an **orthogonal transformation** of the original variables that accounts for as much of the variability of the data as possible.
- Specifically, the first principal component is the direction along which the variance of the data is largest.

PCA and Variance



PCA applied to an ellipsoidically shaped point cloud



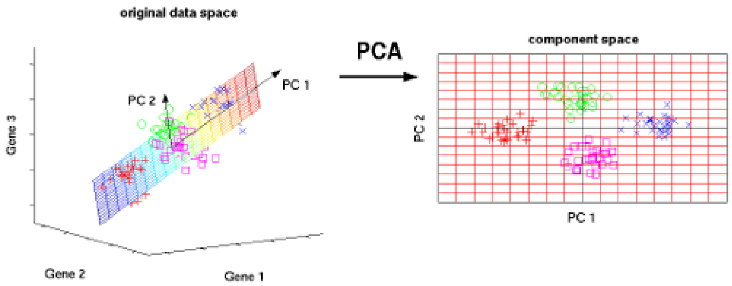
more information: www.joyofdata.de/blog/illustration-of-principal-component-analysis-pca

PCA and SVD

- The **covariance matrix** of the data tells how correlated different pairs of variables are:
$$\mathbf{C} = \mathbf{A}^T\mathbf{A} = [n \times n]$$
- The largest eigenvalue of $\mathbf{C}$ is the direction (line) that minimizes the sum of squares of the distances from the points to the line, or equivalently, **maximizes the variance** of the data projected onto that line.
- The SVD of the data matrix is $\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\star$.
- The eigenvectors of $\mathbf{C}$ are in fact the columns of $\mathbf{V}$, and the eigenvalues of $\mathbf{C}$ are the squares of the singular values,
$$\mathbf{C} = \mathbf{A}^T\mathbf{A} = \mathbf{V}\boldsymbol{\Sigma}\left(\mathbf{U}^\star\mathbf{U}\right)\boldsymbol{\Sigma}\mathbf{V}^\star = \mathbf{V}\boldsymbol{\Sigma}^2\mathbf{V}^\star.$$

  Note: the eigenvalues values are necessarily real and positive since $\mathbf{C}$ is positive semi-definite.

Dimensionality reduction via PCA


original data space
PC 2
PC 1
Gene 3
Gene 2
Gene 1
PCA
component space
PC 2
PC 1