10 Lecture 10, Feb 4

Announcements

- HW3 posted. Due ~~Feb 11~~ Feb 16 @ 11:59PM. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/biostat_m280_2016_hw3.pdf`
- Quiz 2 ~~this Thu Feb 4~~ next Tue Feb 9. In class, closed book.
- TA office hour location: common area of Gonda 6th floor or Gonda 6545 (Max's office).

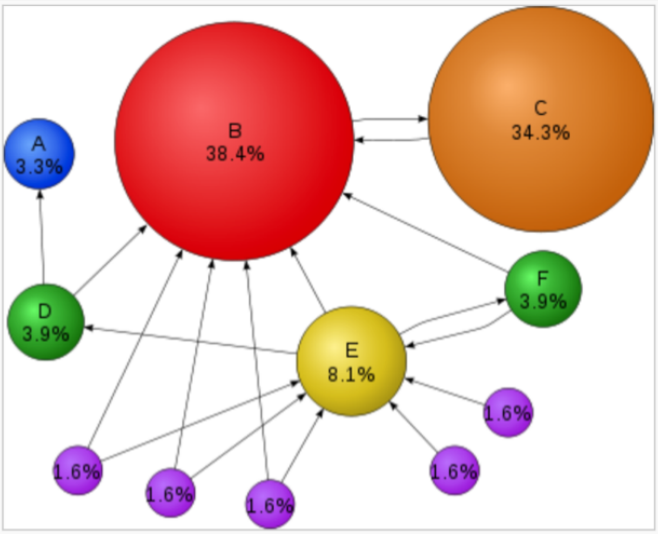
Last time

- Sweep operator.
- Summary of numerical methods for linear regression.
- Summary of solving linear equations.
- Condition number.

Today

- Iterative methods for solving linear equations.
- A catalog of easy linear systems.

Iterative method for solving linear equations: introduction



Mathematical **PageRanks** for a simple network, expressed as percentages. (Google uses a logarithmic scale.) Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value. If web surfers who start on a random page have an 85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web, they will reach Page E 8.1% of the time. (The 15% likelihood of jumping to an arbitrary page corresponds to a damping factor of 85%.) Without damping, all web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presence of damping, Page A effectively links to all pages in the web, even though it has no outgoing links of its own.

- Direct method (flops fixed *a priori*) vs iterative methods:
  - Direct method (GE/LU, Cholesky, QR, SVD): good for dense, small or moderate sized, unstructured $\boldsymbol{A}$
  - Iterative methods (Jacobi, Gauss-Seidal, SOR, conjugate-gradient, GMRES): good for large, sparse, or structured linear system, parallel computing, warm start
- PageRank (HW3):
  - $\boldsymbol{A} \in \{0,1\}^{n\times n}$ the connectivity matrix with entries

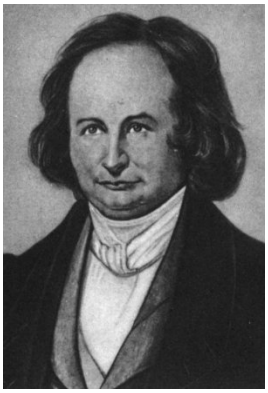
$$a_{ij} = \begin{cases} 1 & \text{if page } i \text{ links to page } j \\ 0 & \text{otherwise} \end{cases}.$$

    $n \approx 10^9$ in Feb 2016.

- $r_i = \sum_j a_{ij}$ is the out-degree of page i.
- Larry Page: Imagine a random surfer wandering on internet according to following rules:
  - From a page i with $r_i > 0$
    - with probability p, (s)he randomly chooses a link on page i (uniformly) and follows that link to the next page
    - with probability $1 - p$, (s)he randomly chooses one page from the set of all n pages (uniformly) and proceeds to that page
  - From a page i with $r_i = 0$ (a dangling page), (s)he randomly chooses one page from the set of all n pages (uniformly) and proceeds to that page

  The process defines a Markov chain on the space of n pages. Stationary distribution of this Markov chain gives the ranks (probabilities) of each page.
- Stationary distribution is the top left eigenvector of the transition matrix $\boldsymbol{P}$ corresponding to eigenvalue 1. Equivalently it can be cast as a linear equation.
- Largest matrix computation in world (?).
- GE/LU will take $2 \times (10^9)^3/3/10^{12} \approx 6.66 \times 10^{14}$ seconds $\approx$ 20 million years on a tera-flop supercomputer!
- Iterative methods come to the rescue.

Jacobi method



Solve linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

- Jacobi iteration:

$$x_i^{(t+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(t)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(t)}}{a_{ii}}.$$

- Requires non-zero diagonal element!
- One round costs $2n^2$ flops with an unstructured $\boldsymbol{A}$. Gain over GE/LU if converges in $o(n)$ iterations. Saving is huge for *sparse* or *structured* $\boldsymbol{A}$. By structured, we mean the matrix-vector multiplication $\boldsymbol{A}\boldsymbol{v}$ is fast.
- Splitting: $\boldsymbol{A} = \boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U}$.
- Jacobi: $\boldsymbol{D}\boldsymbol{x}^{(t+1)} = -(\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^{(t)} + \boldsymbol{b}$, i.e.,

$$\boldsymbol{x}^{(t+1)} = -\boldsymbol{D}^{-1}(\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^{(t)} + \boldsymbol{D}^{-1}\boldsymbol{b}.$$

Gauss-Seidel


- Gauss-Seidel iteration:

$$x_i^{(t+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(t+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(t)}}{a_{ii}}.$$

- With splitting, $(\boldsymbol{D} + \boldsymbol{L})\boldsymbol{x}^{(t+1)} = -\boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{b}$, i.e.,

$$\boldsymbol{x}^{(t+1)} = -(\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{U}\boldsymbol{x}^{(t)} + (\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{b}.$$

- GS converges for any $\boldsymbol{x}^{(0)}$ for symmetric and pd $\boldsymbol{A}$.
- Convergence rate of Gauss-Seidel is the spectral radius of the $(\boldsymbol{D}+\boldsymbol{L})^{-1}\boldsymbol{U}$.
- Comparing Jacobi and GS, Jacobi is particularly attractive for parallel computing.

Successive over-relaxation (SOR)

- SOR: $x_i^{(t+1)} = \omega(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(t+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(t)})/a_{ii} + (1-\omega)x_i^{(t)}$, i.e.,

$$\boldsymbol{x}^{(t+1)} = (\boldsymbol{D}+\omega\boldsymbol{L})^{-1}[(1-\omega)\boldsymbol{D} - \omega\boldsymbol{U}]\boldsymbol{x}^{(t)} + (\boldsymbol{D}+\omega\boldsymbol{L})^{-1}(\boldsymbol{D}+\boldsymbol{L})^{-1}\omega\boldsymbol{b}.$$

- Need to pick $\omega \in [0,1]$ beforehand, with the goal of improving convergence rate.

Conjugate gradient method

Solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is equivalent to minimizing the quadratic function $\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$. To do later, when we study optimization. Conjugate gradient and its variants are the top-notch iterative methods for solving huge, structured linear systems.

Table 1. Kershaw's results for a fusion problem.

Method	Number of iterations
Gauss Seidel	208,000
Block successive overrelaxation methods	765
Incomplete Cholesky conjugate gradients	25

A list of "easy" linear systems

Consider $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, $\boldsymbol{A} \in \mathbb{R}^{n\times n}$. Or, consider matrix inverse (if you want). $\boldsymbol{A}$ can be huge. Keep massive data in mind: 1000 Genome Project, NetFlix, Google PageRank, finance, spatial statistics, ... We should be alert to many easy linear systems. *Don't waste computing resources by bad choices of algorithms!*

- Diagonal: n flops.

- Tridiagonal/banded: Band LU, band Cholesky, ... roughly $O(n)$ flops
- Triangular: n^2 flops
- Block diagonal: Suppose $n = \sum_i n_i$. $(\sum_i n_i)^3$ vs $\sum_i n_i^3$.
- Kronecker product: $(\boldsymbol{A} \otimes \boldsymbol{B})^{-1} = \boldsymbol{A}^{-1} \otimes \boldsymbol{B}^{-1}$, $(\boldsymbol{C}^\top \otimes \boldsymbol{A})\text{vec}\boldsymbol{B} = \text{vec}(\boldsymbol{ABC})$ fits iterative method.
- Sparsity: iterative method, or sparse matrix decomposition.
  Remark: Probably the easiest recognizable structure. Familiarize yourself with the sparse matrix computation tools in JULIA, MATLAB, R (`Matrix` package), MKL (sparse BLAS), ... as much as possible.
- Easy plus low rank: $\boldsymbol{U} \in \mathbb{R}^{n\times m}$, $\boldsymbol{V} \in \mathbb{R}^{n\times m}$, $m \ll n$. Woodbury formula

$$(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{V}^\top)^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{I}_m + \boldsymbol{V}^\top\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}^\top\boldsymbol{A}^{-1}.$$

  Keep HW2 Q5 in mind.
- Easy plus border: For $\boldsymbol{A}$ pd and $\boldsymbol{V}$ full row rank,

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{V}^\top \\ \boldsymbol{V} & \boldsymbol{0} \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{V}^\top(\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{V}^\top)^{-1}\boldsymbol{V}\boldsymbol{A}^{-1} & \boldsymbol{A}^{-1}\boldsymbol{V}^\top(\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{V}^\top)^{-1} \\ (\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{V}^\top)^{-1}\boldsymbol{V}\boldsymbol{A}^{-1} & -(\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{V}^\top)^{-1} \end{pmatrix}.$$

- Orthogonal: n^2 flops *at most*. Permutation matrix, Householder matrix, Jacobi matrix, ... take less.
- Toeplitz systems:

$$\boldsymbol{T} = \begin{pmatrix} r_0 & r_1 & r_2 & r_3 \\ r_{-1} & r_0 & r_1 & r_2 \\ r_{-2} & r_{-1} & r_0 & r_1 \\ r_{-3} & r_{-2} & r_{-1} & r_0 \end{pmatrix}.$$

  $\boldsymbol{Tx} = \boldsymbol{b}$, where $\boldsymbol{T}$ is pd and Toeplitz, can be solved in $O(n^2)$ flops. Durbin algorithm (Yule-Walker equation), Levinson algorithm (general $\boldsymbol{b}$), Trench algorithm (inverse). These matrices occur in auto-regressive models and econometrics.

- Circulant systems: Toeplitz matrix with wraparound

$$C(\boldsymbol{z}) = \begin{pmatrix} z_0 & z_4 & z_3 & z_2 & z_1 \\ z_1 & z_0 & z_4 & z_3 & z_2 \\ z_2 & z_1 & z_0 & z_4 & z_3 \\ z_3 & z_2 & z_1 & z_0 & z_4 \\ z_4 & z_3 & z_2 & z_1 & z_0 \end{pmatrix},$$

  FFT type algorithms: DCT (discrete cosine transform) and DST (discrete sine transform).

- Vandermonde matrix: such as in interpolation and approximation problems

$$\boldsymbol{V}(x_0, \ldots, x_n) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_0^n & x_1^n & \cdots & x_n^n \end{pmatrix}.$$

  $\boldsymbol{V}\boldsymbol{x} = \boldsymbol{b}$ or $\boldsymbol{V}^T\boldsymbol{x} = \boldsymbol{b}$ can be solved in $O(n^2)$ flops.

- Cauchy-like matrices:

$$\boldsymbol{\Omega}\boldsymbol{A} - \boldsymbol{A}\boldsymbol{\Lambda} = \boldsymbol{R}\boldsymbol{S}^T,$$

  where $\boldsymbol{\Omega} = \text{diag}(\omega_1, \ldots, \omega_n)$ and $\boldsymbol{\Lambda} = (\lambda_1, \ldots, \lambda_n)$. $O(n)$ flops for LU and QR.

- Structured-rank problems: semiseparable matrices (LU and QR takes $O(n)$ flops), quasiseparable matrices, ...

- Fast multiple method (FMM) for kernel matrix.

- ...

Other computations such as matrix-vector multiplication with these "easy" matrices are typically fast too.
Bottom line: Don't blindly use `solve()`.

Linear algebra review: eigen-decomposition

Our last topic on numerical linear algebra is eigen-decomposition and singular value decomposition (SVD). We already saw the wide applications of QR decomposition in least squares problem and solving square and under-determined linear equations. Eigen-decomposition and SVD can be deemed as more thorough orthogonalization of a matrix. We start with a brief review of the related linear algebra.

- *Eigenvalues* are defined as roots of the characteristic equation $\det(\lambda \boldsymbol{I}_n - \boldsymbol{A}) = 0$.
- If λ is an eigenvalue of $\boldsymbol{A}$, then there exist non-zero $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ such that $\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$ and $\boldsymbol{y}^T \boldsymbol{A} = \lambda \boldsymbol{y}^T$. $\boldsymbol{x}$ and $\boldsymbol{y}$ are called the (column) *eigenvector* and *row eigenvector* of $\boldsymbol{A}$ associated with the eigenvalue λ.
- $\boldsymbol{A}$ is singular if and only if it has at least one 0 eigenvalue.
- Eigenvectors associated with distinct eigenvalues are linearly independent.
- Eigenvalues of an upper or lower triangular matrix are its diagonal entries: $\lambda_i = a_{ii}$.
- Eigenvalues of an idempotent matrix are either 0 or 1.
- Eigenvalues of an orthogonal matrix have complex modulus 1.
- In most statistical applications, we deal with eigenvalues/eigenvectors of symmetric matrices. The eigenvalues and eigenvectors of a real *symmetric* matrix are real.
- Eigenvectors associated with distinct eigenvalues of a symmetry matrix are orthogonal.
- Eigen-decompostion of a symmetric matrix: $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\intercal$, where
  - $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_n)$
  - columns of $\boldsymbol{U}$ are the eigenvectors, which are (or can be chosen to be) mutually orthonormal. Thus $\boldsymbol{U}$ is an orthogonal matrix.
- A real symmetric matrix is positive semidefinite (positive definite) if and only if all eigenvalues are nonnegative (positive).

- *Spectral radius* $\rho(\boldsymbol{A}) = \max_i |\lambda_i|$.
- $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ a square matrix (not required to be symmetric), then $\mathrm{tr}(\boldsymbol{A}) = \sum_i \lambda_i$ and $\det(\boldsymbol{A}) = \prod_i \lambda_i$.

Linear algebra review: SVD

- Singular value decomposition (SVD): For a rectangular matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, let $p = \min\{m, n\}$, then we have the SVD

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal,$$

  where

  - $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m) \in \mathbb{R}^{m\times m}$ is orthogonal
  - $\boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n) \in \mathbb{R}^{n\times n}$ is orthgonal
  - $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$.

  σ_i are called the *singular values*, $\boldsymbol{u}_i$ are the *left singular vectors*, and $\boldsymbol{v}_i$ are the *right singular vectors*.

- Thin SVD. Assume $m \geq n$. $\boldsymbol{A}$ can be factored as

$$\boldsymbol{A} = \boldsymbol{U}_n\boldsymbol{\Sigma}_n\boldsymbol{V}^\intercal = \sum_{i=1}^{n} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T,$$

  where

  - $\boldsymbol{U}_n \in \mathbb{R}^{m\times n}$, $\boldsymbol{U}_n^\intercal\boldsymbol{U}_n = \boldsymbol{I}_n$
  - $\boldsymbol{V} \in \mathbb{R}^{n\times n}$, $\boldsymbol{V}^\intercal\boldsymbol{V} = \boldsymbol{I}_n$
  - $\boldsymbol{\Sigma}_n = \mathrm{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{n\times n}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$

- Denote $\sigma(\boldsymbol{A}) = (\sigma_1, \ldots, \sigma_p)^T$. Then

  - $r = \mathrm{rank}(\boldsymbol{A}) = \#$ nonzero singular values $= \|\sigma(\boldsymbol{A})\|_0$
  - $\boldsymbol{A} = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r^T = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$
  - $\|\boldsymbol{A}\|_{\mathrm{F}} = (\sum_{i=1}^{p} \sigma_i^2)^{1/2} = \|\sigma(\boldsymbol{A})\|_2$
  - $\|\boldsymbol{A}\|_2 = \sigma_1 = \|\sigma(\boldsymbol{A})\|_\infty$

- Assume $\text{rank}(\boldsymbol{A}) = r$ and partition $\boldsymbol{U} = (\boldsymbol{U}_r, \tilde{\boldsymbol{U}}_r) \in \mathbb{R}^{m\times m}$ and $\boldsymbol{V} = (\boldsymbol{V}_r, \tilde{\boldsymbol{V}}_r) \in \mathbb{R}^{n\times n}$, then
  - $\mathcal{C}(\boldsymbol{A}) = \mathcal{C}(\boldsymbol{U}_r)$, $\mathcal{N}(\boldsymbol{A}^T) = \mathcal{C}(\tilde{\boldsymbol{U}}_r)$
  - $\mathcal{N}(\boldsymbol{A}) = \mathcal{C}(\tilde{\boldsymbol{V}}_r)$, $\mathcal{C}(\boldsymbol{A}^T) = \mathcal{C}(\boldsymbol{V}_r)$
  - $\boldsymbol{U}_r\boldsymbol{U}_r^T$ is the orthogonal projection onto $\mathcal{C}(\boldsymbol{A})$
  - $\tilde{\boldsymbol{U}}_r\tilde{\boldsymbol{U}}_r^T$ is the orthogonal projection onto $\mathcal{N}(\boldsymbol{A}^T)$
  - $\boldsymbol{V}_r\boldsymbol{V}_r^T$ is the orthogonal projection onto $\mathcal{C}(\boldsymbol{A}^T)$
  - $\tilde{\boldsymbol{V}}_r\tilde{\boldsymbol{V}}_r^T$ is the orthogonal projection onto $\mathcal{N}(\boldsymbol{A})$
- Relation to eigen-decomposition. Using thin SVD,

$$\begin{aligned}\boldsymbol{A}^\intercal\boldsymbol{A} &= \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\intercal\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal = \boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^\intercal \\ \boldsymbol{A}\boldsymbol{A}^\intercal &= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\intercal = \boldsymbol{U}\boldsymbol{\Sigma}^2\boldsymbol{U}^\intercal.\end{aligned}$$

  In principle we can obtain singular triplets of $\boldsymbol{A}$ by doing two eigen-decompositions.
- Another relation to eigen-decomposition. Using thin SVD,

$$\begin{pmatrix}\boldsymbol{0}_{n\times n} & \boldsymbol{A}^\intercal \\ \boldsymbol{A} & \boldsymbol{0}_{m\times m}\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}\boldsymbol{V} & \boldsymbol{V} \\ \boldsymbol{U} & -\boldsymbol{U}\end{pmatrix}\begin{pmatrix}\boldsymbol{\Sigma} & \boldsymbol{0}_{n\times n} \\ \boldsymbol{0}_{n\times n} & -\boldsymbol{\Sigma}\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}\boldsymbol{V}^\intercal & \boldsymbol{U}^\intercal \\ \boldsymbol{V}^\intercal & -\boldsymbol{U}^\intercal\end{pmatrix}.$$

  Hence any symmetric eigen-solver can produce the SVD of a matrix $\boldsymbol{A}$ without forming $\boldsymbol{A}\boldsymbol{A}^\intercal$ or $\boldsymbol{A}^\intercal\boldsymbol{A}$.
- Yet another relation to eigen-decomposition: If the eigendecomposition of a real symmetric matrix is $\boldsymbol{A} = \boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^T = \boldsymbol{W}\text{diag}(\lambda_1, \ldots, \lambda_n)\boldsymbol{W}^T$, then

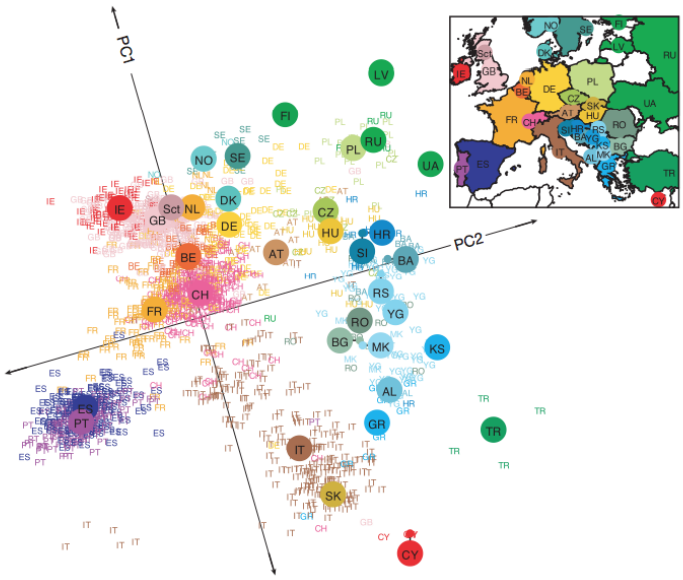
$$\boldsymbol{A} = \boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^T = \boldsymbol{W}\begin{pmatrix}|\lambda_1| & & \\ & \ddots & \\ & & |\lambda_n|\end{pmatrix}\begin{pmatrix}\text{sgn}(\lambda_1) & & \\ & \ddots & \\ & & \text{sgn}(\lambda_n)\end{pmatrix}\boldsymbol{W}^T$$

  is the SVD of $\boldsymbol{A}$.

Applications of eigen-decomposition and SVD

- Principal components analysis (PCA). $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ is a centered data matrix. Perform SVD $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal$ or equivalently $\boldsymbol{X}^\intercal\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^\intercal$. The linear combinations $\tilde{\boldsymbol{x}}_i = \boldsymbol{X}\boldsymbol{v}_i$ are the principal components and have variance σ_i^2. Usages:

1. Dimension reduction: reduce dimensionality p to $q \ll p$. Use top PCs $\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_q$ in downstream analysis.
2. Use PCs to adjust for confounding – a serious issue in association studies in large data sets.



- Low rank approximation, e..g, image/data compression.
  Eckart-Young theorem:

$$\min_{\text{rank}(\boldsymbol{Y})=r} \|\boldsymbol{X} - \boldsymbol{Y}\|_{\text{F}}^2$$

  is achieved by $\boldsymbol{Y} = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^{\text{T}}$ with optimal value $\sum_{i=1}^{r-1} \sigma_i^2$, where $(\sigma_i, \boldsymbol{u}_i, \boldsymbol{v}_i)$ are singular values and vectors of $\boldsymbol{X}$.

  Gene Golub's 2691×598 picture requires $2691 \times 598 \times 6 = 9,655,308$ bytes (RGB 16 bit per channel). Rank 120 approximation requires $120 \times (2691 + 598) \times 6 = 2,368,080$ bytes. Rank 50 approximation requires $50 \times (2691 + 598) \times 6 = 986,700$ bytes. Rank 12 approximation requires $12 \times (2691 + 598) \times 8 = 236,808$ bytes.

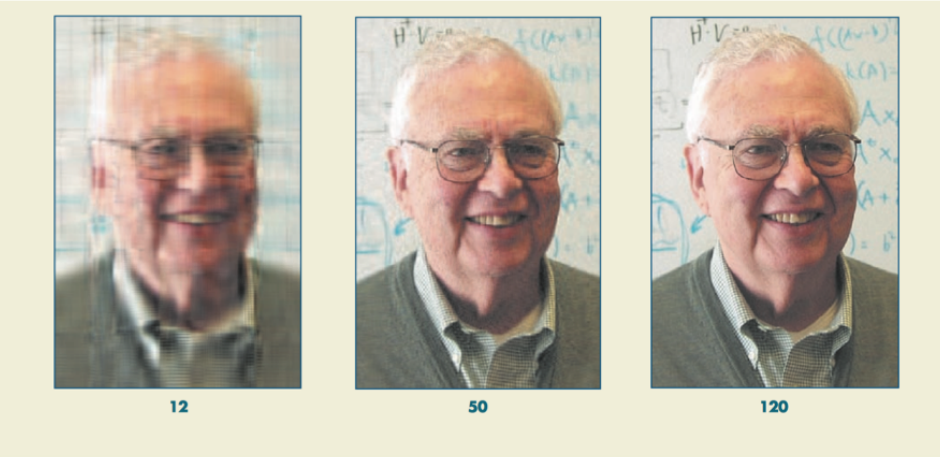


Figure 2. Rank 12, 50, and 120 approximations to a rank 598 color photo of Gene Golub.

- Least squares, ridge regression (HW3), least squares over a sphere, ...
- Moore-Penrose (MP) inverse: Using thin SVD,

$$\boldsymbol{A}^{+} = \boldsymbol{V}\boldsymbol{\Sigma}^{+}\boldsymbol{U}^{\mathsf{T}},$$

  where $\boldsymbol{\Sigma}^{+} = \mathrm{diag}(\sigma_1^{-1}, \ldots, \sigma_r^{-1}, 0, \ldots, 0)$, $r = \mathrm{rank}(\boldsymbol{A})$. This is how the `ginv()` function is implemented in `MASS` package.
- Read KL Chapters 8 and 9 and do HW4 for some more applications.

One eigen-pair - power method

Assume $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ symmetric.

- *Power method* iterates according to

$$\begin{aligned} \boldsymbol{x}^{(t)} &\leftarrow \frac{1}{\|\boldsymbol{A}\boldsymbol{x}^{(t-1)}\|_2}\boldsymbol{A}\boldsymbol{x}^{(t-1)} \\ \lambda_1^{(t)} &\leftarrow \boldsymbol{x}^{(t)\,\mathsf{T}}\boldsymbol{A}\boldsymbol{x}^{(t)} \end{aligned}$$

  from an initial guess $\boldsymbol{x}^{(0)}$ of unit norm.
- Suppose we arrange $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$ (the first inequality strict) with

corresponding eigenvectors $\boldsymbol{u}_i$, and expand $\boldsymbol{x}^{(0)} = c_1\boldsymbol{u}_1 + \cdots + c_n\boldsymbol{u}_n$, then

$$\begin{aligned}
\boldsymbol{x}^{(t)} &= \frac{\left(\sum_i \lambda_i^t \boldsymbol{u}_i\boldsymbol{u}_i^\top\right)\left(\sum_i c_i\boldsymbol{u}_i\right)}{\left\|\left(\sum_i \lambda_i^t \boldsymbol{u}_i\boldsymbol{u}_i^\top\right)\left(\sum_i c_i\boldsymbol{u}_i\right)\right\|_2} \\
&= \frac{\sum_i c_i\lambda_i^t\boldsymbol{u}_i}{\|\sum_i c_i\lambda_i^t\boldsymbol{u}_i\|_2} \\
&= \frac{c_1\boldsymbol{u}_1 + c_2(\lambda_2/\lambda_1)^t\boldsymbol{u}_2 + \cdots + c_n(\lambda_n/\lambda_1)^t\boldsymbol{u}_n}{\|c_1\boldsymbol{u}_1 + c_2(\lambda_2/\lambda_1)^t\boldsymbol{u}_2 + \cdots + c_n(\lambda_n/\lambda_1)^t\boldsymbol{u}_n\|_2}\left(\frac{\lambda_1}{|\lambda_1|}\right)^t.
\end{aligned}$$

Thus $\boldsymbol{x}^{(t)} - \frac{c_1\boldsymbol{u}_1}{\|c_1\boldsymbol{u}_1\|_2}\left(\frac{\lambda_1}{|\lambda_1|}\right)^t \to 0$ as $t \to \infty$. The convergence rate is $|\lambda_2|/|\lambda_1|$.

- $\boldsymbol{x}^{(t)\top}\boldsymbol{A}\boldsymbol{x}^{(t)}$ converges to λ_1.
- *Inverse power method* for finding the eigenvalue of smallest absolute value: Substitute $\boldsymbol{A}$ by $\boldsymbol{A}^{-1}$ in the power method. (E.g., pre-compute LU or Cholesky of $\boldsymbol{A}$).
- *Shifted inverse power*: Substitute $(\boldsymbol{A}-\mu\boldsymbol{I})^{-1}$ in the power method. It converges to an eigenvalue close to the given μ.
- Initial guess of the desired eigenvalue can be obtain by Gerschgorin's circle theorem and so on.
- Power method also applies to asymmetric $\boldsymbol{A}$, e.g., PageRank problem costs $O(n)$ per iteration.

Top r eigen-pairs - orthogonal iteration

Generalization of power method to higher dimensional invariant subspace.

- *Orthogonal iteration*: Initialize $\boldsymbol{Q}^{(0)} \in \mathbb{R}^{n\times r}$ with orthonormal columns. For $t = 1, 2, \ldots,$

$$\begin{aligned}
\boldsymbol{Z}^{(t)} &\leftarrow \boldsymbol{A}\boldsymbol{Q}^{(t-1)} \quad (2n^2r \text{ flops}) \\
\boldsymbol{Q}^{(t)}\boldsymbol{R}^{(t)} &\leftarrow \boldsymbol{Z}^{(t)} \quad (\text{QR factorization})
\end{aligned}$$

- $\boldsymbol{Z}^{(t)}$ converges to the eigenspace of the largest r eigenvalues if they are real and separated from remaining spectrum. The convergence rate is $|\lambda_{r+1}|/|\lambda_r|$.

(Impractical) full eigen-decomposition - QR iteration

Assume $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ symmetric.

- take $r = n$ in the orthogonal iteration. Then $\boldsymbol{Q}^{(t)}$ converges to the eigenspace $\boldsymbol{U}$ of $\boldsymbol{A}$. This implies that

$$\boldsymbol{T}^{(t)} := \boldsymbol{Q}^{(t)\,T} \boldsymbol{A} \boldsymbol{Q}^{(t)}$$

  converges to a diagonal form $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.

- Note how to compute $\boldsymbol{T}^{(t)}$ from $\boldsymbol{T}^{(t-1)}$

$$\begin{aligned} \boldsymbol{T}^{(t-1)} &= \boldsymbol{Q}^{(t-1)\,T} \boldsymbol{A} \boldsymbol{Q}^{(t-1)} = \boldsymbol{Q}^{(t-1)\,T} (\boldsymbol{A} \boldsymbol{Q}^{(t-1)}) = (\boldsymbol{Q}^{(t-1)\,T} \boldsymbol{Q}^{(t)}) \boldsymbol{R}^{(t)} \\ \boldsymbol{T}^{(t)} &= \boldsymbol{Q}^{(t)\,T} \boldsymbol{A} \boldsymbol{Q}^{(t)} = \boldsymbol{Q}^{(t)\,T} \boldsymbol{A} \boldsymbol{Q}^{(t-1)} \boldsymbol{Q}^{(t-1)\,\intercal} \boldsymbol{Q}^{(t)} = \boldsymbol{R}^{(t)} (\boldsymbol{Q}^{(t-1)\,\intercal} \boldsymbol{Q}^{(t)}). \end{aligned}$$

- *QR iteration*: Initialize $\boldsymbol{U}^{(0)} \in \mathbb{R}^{n \times n}$ orthogonal and set $\boldsymbol{T}^{(0)} = \boldsymbol{U}^{(0)\,T} \boldsymbol{A} \boldsymbol{U}^{(0)}$. For $t = 1, 2, \ldots$

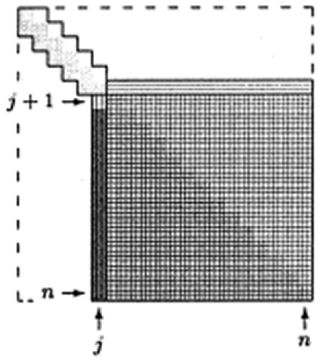
$$\begin{aligned} \boldsymbol{U}^{(t)} \boldsymbol{R}^{(t)} &\leftarrow \boldsymbol{T}^{(t-1)} \quad \text{(QR factorization)} \\ \boldsymbol{T}^{(t)} &\leftarrow \boldsymbol{R}^{(t)} \boldsymbol{U}^{(t)} \end{aligned}$$

- QR iteration is expensive: $O(n^3)$ per iteration and linear convergence rate.

QR algorithm for symmetric eigen-decomposition

Assume $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ symmetric.

- As done in LAPACK: `eigen()` in `R`, `eig()` in `Matlab` and `Julia`.
- Idea: Tri-diagonalization (by Householder) + QR iteration on the tri-diagonal system with implicit shift
  - Step 1: Householder tri-diagonalization: $4n^3/3$ for eigenvalues only, $8n^3/3$ for both eigenvalues and eigenvectors. (Why can't we apply Householder to make it diagonal directly?)

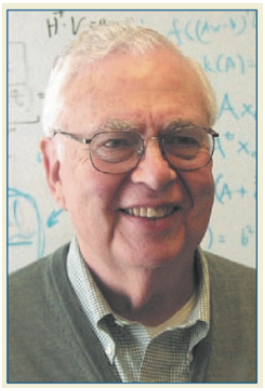


– Step 2: QR iteration on the tridiagonal matrix. Implicit shift accelerates convergence rate. On average 1.3-1.6 QR iteration per eigenvalue, $\sim 20n$ flops per QR iteration. So total operation count is about $30n^2$. Eigenvectors need an extra of about $6n^3$ flops.

	Eigenvalue	Eigenvector
Householder reduction	$4n^3/3$	$4n^3/3$
QR with implicit shift	$\sim 30n^2$	$\sim 6n^3$

- Message: Don't request eigenvectors unless necessary: set `only.values = TRUE` when calling `eigen()` in `R`.
- The *unsymmetric QR algorithm* obtains the real Schur decomposition of an asymmetric matrix $\boldsymbol{A}$.

Algorithm for singular value decomposition (SVD)

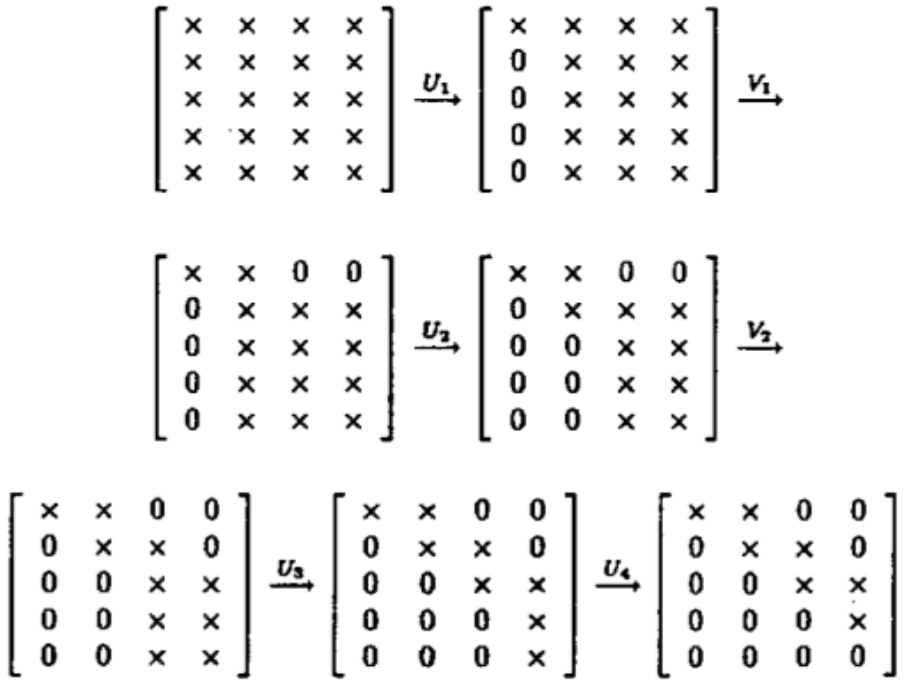


Gene Golub's license plate, photographed by Professor P. M. Kroonenberg of Leiden University.

Assume $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and we seek the SVD $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathsf{T}}$.

- "Golub-Kahan-Reinsch" algorithm:
  - Stage 1: Transform $\boldsymbol{A}$ to an upper bidiagonal form $\boldsymbol{B}$ (by Householder).

$$U_B^T A V_B = \begin{bmatrix} B \\ 0 \end{bmatrix} \qquad B = \begin{bmatrix} d_1 & f_1 & & \cdots & 0 \\ 0 & d_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & f_{n-1} \\ 0 & \cdots & & 0 & d_n \end{bmatrix} \in \mathbb{R}^{n \times n}$$



  - Stage 2: Apply implicit-shift QR step to the tridiagonal matrix $\boldsymbol{B}^\intercal \boldsymbol{B}$ *implicitly*.

- See Golub and Van Loan (1996, Section 8.6) for more details.
- $4m^2n + 8mn^2 + 9n^3$ flops for a tall ($m \geq n$) matrix.
- `svd()` in `R` and `Matlab`: wrapper of the `_GESVD` subroutine in LAPACK.

Lanczos/Arnoldi iterative method for top eigen-pairs

- Motivation
  - Consider the Google PageRank problem. We want to find the top left eigenvector of the transition matrix

$$\boldsymbol{P} = p\boldsymbol{R}^+\boldsymbol{A} + \boldsymbol{z}\boldsymbol{1}_n^\intercal,$$

    where $\boldsymbol{R} = \text{diag}(r_1, \ldots, r_n)$ and $z_j = (1-p)/n$ if $r_i > 0$ and $1/n$ if $r_i = 0$. Suppose there are $n \approx$ billion web pages.

The (unsymmetric) QR algorithm will take order

$$\frac{(1 \times 10^9)^3}{10^{12}} \approx 3.33 \times 10^{14} \text{ seconds } \approx 1 \times 10^7 \text{ years}$$

on a tera-flop supercomputer!

- Consider adjusting for confounding by PCA in modern GWAS (genome-wide association studies). We want to find the top singular values/vectors of a genotype matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, where $n \sim 10^3$ and $p \sim 10^6$.

- Krylov subspace methods are the state-of-art iterative method for obtaining the top eigen-values/vectors or singular values/vectors of large sparse or structured matrices.

- Lanczos method: top eigen-pairs of a large *symmetric* matrix.

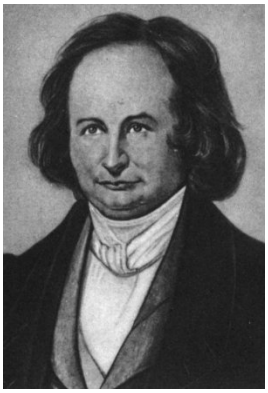
- Arnoldi method: top eigen-pairs of a large *asymmetric* matrix.

- Both methods are also adapted to obtain top singular values/vectors of large sparse or structured matrices.

- We will give a brief overview of these methods together with the conjugate gradient method for solving large linear system.

- `eigs()` and `svds()` in `Matlab` and `Julia` are wrappers of the `ARPACK` package. No native functions in R (?).

Jacobi method for symmetric eigen-decomposition (KL 8.2)



Assume $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is symmetric and we seek the eigen-decomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$.

- Idea: Systematically reduce off-diagonal entries

$$\mathrm{off}(\boldsymbol{A}) = \sum_i \sum_{j\neq i} a_{ij}^2$$

by Jacobi rotations.

- Jacobi/Givens rotations:

$$\boldsymbol{J}(p,q,\theta) = \begin{pmatrix} 1 & & 0 & & 0 & & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & & \cos(\theta) & & \sin(\theta) & & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & & -\sin(\theta) & & \cos(\theta) & & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & & 0 & & 0 & & 1 \end{pmatrix},$$

$\boldsymbol{J}(p,q,\theta)$ is orthogonal.

- Consider $\boldsymbol{B} = \boldsymbol{J}^\intercal \boldsymbol{A}\boldsymbol{J}$. $\boldsymbol{B}$ preserves the symmetry and eigenvalues of $\boldsymbol{A}$.

Taking

$$\begin{cases} \tan(2\theta) = 2a_{pq}/(a_{qq}-a_{pp}) & \text{if } a_{pp} \neq a_{qq} \\ \theta = \pi/4 & \text{if } a_{pp} = a_{qq} \end{cases}$$

forces $b_{pq} = 0$.

- Since orthogonal transform preserves Frobenius norm, we have

$$b_{pp}^2 + b_{qq}^2 = a_{pp}^2 + a_{qq}^2 + 2a_{pq}^2.$$

(Just check the 2-by-2 block)

- Since $\|\boldsymbol{A}\|_{\mathrm{F}} = \|\boldsymbol{B}\|_{\mathrm{F}}$, this implies that the off-diagonal part

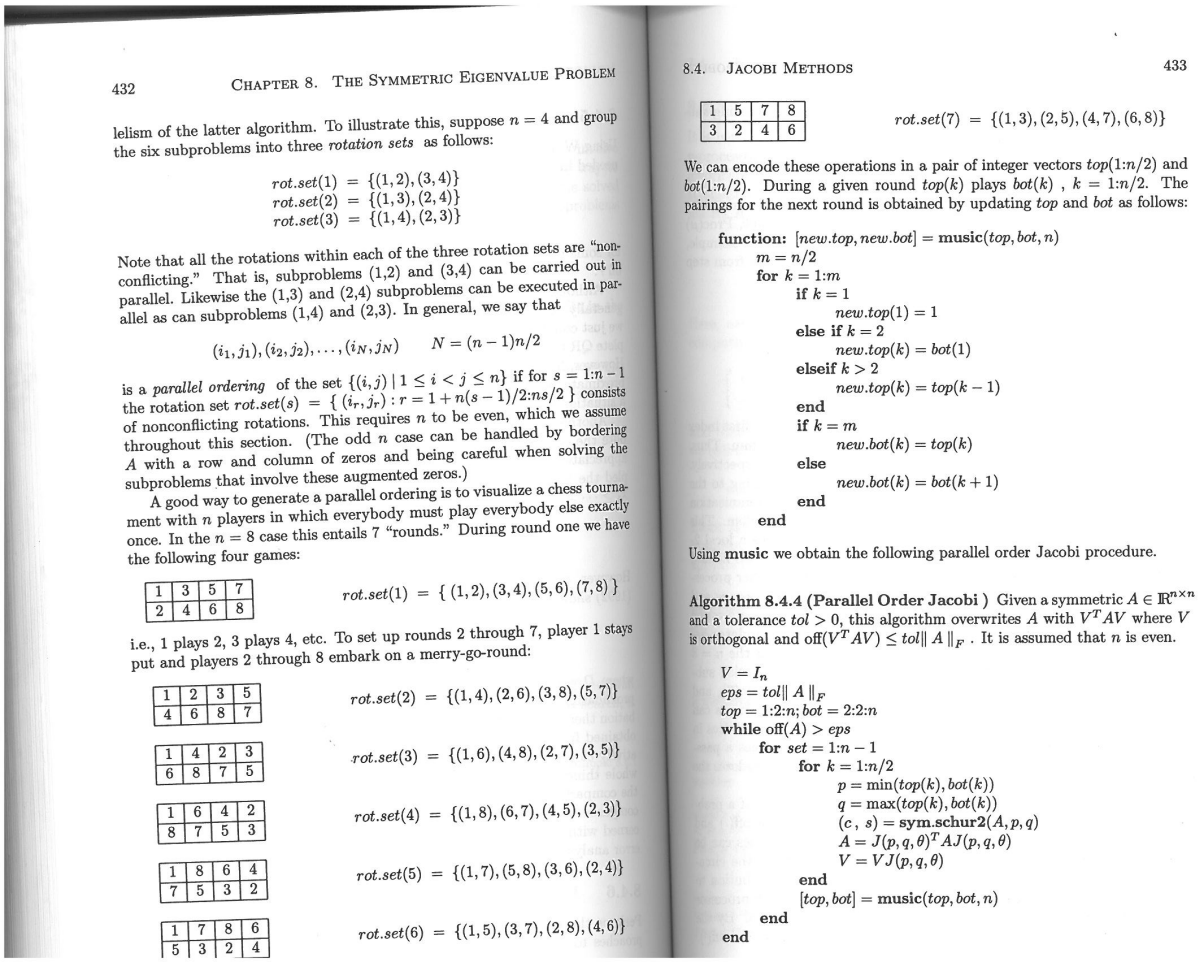
$$\mathrm{off}(\boldsymbol{B}) = \mathrm{off}(\boldsymbol{A}) - 2a_{pq}^2$$

is decreased whenever $a_{pq} \neq 0$.

- One Jacobi rotation costs $O(n)$ flops.
- *Classical Jacobi*: search for the largest $|a_{ij}|$ at each iteration.
- $\text{off}(\boldsymbol{A}) \leq n(n-1)a_{ij}^2$ and $\text{off}(\boldsymbol{B}) = \text{off}(\boldsymbol{A}) - 2a_{ij}^2$ together implies

$$\text{off}(\boldsymbol{B}) \leq \left(1 - \frac{2}{n(n-1)}\right) \text{off}(\boldsymbol{A}).$$

- In practice, cyclic-by-row implementation, to avoid the costly $O(n^2)$ search in the classical Jacobi.
- Jacobi method attracts a lot recent attention because of its rich inherent parallelism.
- *Parallel Jacobi*: "merry-go-round" to generate parallel ordering.



432 CHAPTER 8. THE SYMMETRIC EIGENVALUE PROBLEM

lelism of the latter algorithm. To illustrate this, suppose $n = 4$ and group the six subproblems into three *rotation sets* as follows:

$$\begin{aligned} rot.set(1) &= \{(1,2),(3,4)\} \\ rot.set(2) &= \{(1,3),(2,4)\} \\ rot.set(3) &= \{(1,4),(2,3)\} \end{aligned}$$

Note that all the rotations within each of the three rotation sets are "nonconflicting." That is, subproblems (1,2) and (3,4) can be carried out in parallel. Likewise the (1,3) and (2,4) subproblems can be executed in parallel as can subproblems (1,4) and (2,3). In general, we say that

$$(i_1,j_1),(i_2,j_2),\ldots,(i_N,j_N) \qquad N = (n-1)n/2$$

is a *parallel ordering* of the set $\{(i,j) \mid 1 \leq i < j \leq n\}$ if for $s = 1{:}n-1$ the rotation set $rot.set(s) = \{\,(i_r,j_r) : r = 1 + n(s-1)/2{:}ns/2\,\}$ consists of nonconflicting rotations. This requires n to be even, which we assume throughout this section. (The odd n case can be handled by bordering A with a row and column of zeros and being careful when solving the subproblems that involve these augmented zeros.)

A good way to generate a parallel ordering is to visualize a chess tournament with n players in which everybody must play everybody else exactly once. In the $n = 8$ case this entails 7 "rounds." During round one we have the following four games:

1	3	5	7
2	4	6	8

$rot.set(1) = \{\,(1,2),(3,4),(5,6),(7,8)\,\}$

i.e., 1 plays 2, 3 plays 4, etc. To set up rounds 2 through 7, player 1 stays put and players 2 through 8 embark on a merry-go-round:

1	2	3	5
4	6	8	7

$rot.set(2) = \{(1,4),(2,6),(3,8),(5,7)\}$

1	4	2	3
6	8	7	5

$rot.set(3) = \{(1,6),(4,8),(2,7),(3,5)\}$

1	6	4	2
8	7	5	3

$rot.set(4) = \{(1,8),(6,7),(4,5),(2,3)\}$

1	8	6	4
7	5	3	2

$rot.set(5) = \{(1,7),(5,8),(3,6),(2,4)\}$

1	7	8	6
5	3	2	4

$rot.set(6) = \{(1,5),(3,7),(2,8),(4,6)\}$

8.4. JACOBI METHODS 433

1	5	7	8
3	2	4	6

$rot.set(7) = \{(1,3),(2,5),(4,7),(6,8)\}$

We can encode these operations in a pair of integer vectors $top(1{:}n/2)$ and $bot(1{:}n/2)$. During a given round $top(k)$ plays $bot(k)$, $k = 1{:}n/2$. The pairings for the next round is obtained by updating top and bot as follows:

```
function: [new.top, new.bot] = music(top, bot, n)
    m = n/2
    for k = 1:m
        if k = 1
            new.top(1) = 1
        else if k = 2
            new.top(k) = bot(1)
        elseif k > 2
            new.top(k) = top(k − 1)
        end
        if k = m
            new.bot(k) = top(k)
        else
            new.bot(k) = bot(k + 1)
        end
    end
```

Using **music** we obtain the following parallel order Jacobi procedure.

Algorithm 8.4.4 (Parallel Order Jacobi) Given a symmetric $A \in \mathbb{R}^{n\times n}$ and a tolerance $tol > 0$, this algorithm overwrites A with V^TAV where V is orthogonal and $\text{off}(V^TAV) \leq tol \| A \|_F$. It is assumed that n is even.

```
V = I_n
eps = tol‖ A ‖_F
top = 1:2:n; bot = 2:2:n
while off(A) > eps
    for set = 1:n − 1
        for k = 1:n/2
            p = min(top(k), bot(k))
            q = max(top(k), bot(k))
            (c , s) = sym.schur2(A, p, q)
            A = J(p, q, θ)^T AJ(p, q, θ)
            V = VJ(p, q, θ)
        end
        [top, bot] = music(top, bot, n)
    end
end
```

Generalized eigen-problem

- Generalized eigen-problem: $\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{B}\boldsymbol{x}$, where $\boldsymbol{A}$ psd and $\boldsymbol{B}$ pd.
- Applications: partial least squares (PLS), sliced inverse regression (SIR), canonical correlation analysis (CCA).
- Method 1: $\boldsymbol{B}^{-1}\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$. Non-symmetric eigen-problem ☹.
- Method 2: Cholesky $\boldsymbol{B} = \boldsymbol{L}\boldsymbol{L}^\intercal$. Then $\boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{L}^{-T}\boldsymbol{y} = \lambda \boldsymbol{y}$ where $\boldsymbol{y} = \boldsymbol{L}^\intercal \boldsymbol{x}$.
- Method 3 (most numerically stable, $\boldsymbol{B}$ can be rank deficient): QZ algorithm.
- `eig()` and `qz()` in `Matlab` and `Julia` implement QZ. No native function in `R`?

Generalized singular value decomposition

- $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{p\times n}$. Then there exists orthogonal $\boldsymbol{U} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{V} \in \mathbb{R}^{p\times p}$ and an invertible $\boldsymbol{X} \in \mathbb{R}^{n\times n}$ such that

$$\begin{aligned} \boldsymbol{U}^T\boldsymbol{A}\boldsymbol{X} &= \boldsymbol{C} = \text{diag}(c_1, \ldots, c_n), \quad c_i \geq 0 \\ \boldsymbol{V}^T\boldsymbol{B}\boldsymbol{X} &= \boldsymbol{S} = \text{diag}(s_1, \ldots, s_q), \quad s_i \geq 0, \end{aligned}$$

  where $q = \min\{p, n\}$.
- Applications: quadratically inequality-constrained least squares problem (LSQI).
- `gsvd()` in `Matlab` implements generalized SVD. No native function in `R`?

In the zoo of least squares (self-study)

Weighted least squares

- In weighted least squares, we minimize $\sum_{i=1}^n w_i(y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2$, where $w_i > 0$ are observation weights.
- Let $\boldsymbol{W} = \text{diag}(w_1, \ldots, w_n)$. Then the criterion is $\|\boldsymbol{W}^{1/2}\boldsymbol{y} - \boldsymbol{W}^{1/2}\boldsymbol{X}\boldsymbol{\beta}\|_2^2$, which can be solved by standard methods for least squares with $\tilde{\boldsymbol{y}} = \boldsymbol{W}^{1/2}\boldsymbol{y}$ and $\tilde{\boldsymbol{X}} = \boldsymbol{W}^{1/2}\boldsymbol{X}$.

General least squares

- In Aitken model: $E(\boldsymbol{y}) = \boldsymbol{X}\boldsymbol{\beta}$, $\mathrm{Cov}(\boldsymbol{y}) = \sigma^2\boldsymbol{V}$, where $\boldsymbol{V}$ is a positive semidefinite matrix. We minimize the generalized least squares criterion

$$(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T \boldsymbol{M}^- (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}),$$

  where $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ is some positive semidefinite matrix, e.g., $\boldsymbol{M} = \boldsymbol{V}$ for nonsingular $\boldsymbol{V}$ or $\boldsymbol{M} = \boldsymbol{V} + \boldsymbol{X}\boldsymbol{X}^T$ for singular $\boldsymbol{V}$.

- Let $\boldsymbol{M} = \boldsymbol{B}\boldsymbol{B}^T$ for some $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ (e.g., the Cholesky factor). One approach is to minimize

$$\|\boldsymbol{B}^{-1}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\|_2^2.$$

  Unfortunately, when $\boldsymbol{B}$ is poorly conditioned (or even not invertible), the procedure produces a poor solution.

- Paige's method. The generalized least squares problem is equivalent to

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{v}^T\boldsymbol{v} \\ \text{subject to} \quad & \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{B}\boldsymbol{v} = \boldsymbol{y}. \end{aligned}$$

  To solve this problem, first compute the QR of $\boldsymbol{X}$

$$\boldsymbol{X} = (\boldsymbol{Q}_1, \boldsymbol{Q}_2)\begin{pmatrix}\boldsymbol{R}_1 \\ \boldsymbol{0}\end{pmatrix}.$$

  Compute another QR for the (flat) matrix $\boldsymbol{Q}_2^T\boldsymbol{B}$ such that

$$\boldsymbol{Q}_2^T\boldsymbol{B} = (\boldsymbol{0}, \boldsymbol{S})\begin{pmatrix}\boldsymbol{Z}_1^T \\ \boldsymbol{Z}_2^T\end{pmatrix},$$

  where $\boldsymbol{S}$ is upper triangular and $(\boldsymbol{Z}_1, \boldsymbol{Z}_2) \in \mathbb{R}^{n\times n}$ is orthogonal. Then the constraint becomes

$$\begin{pmatrix}\boldsymbol{R}_1 \\ \boldsymbol{0}\end{pmatrix}\boldsymbol{\beta} + \begin{pmatrix}\boldsymbol{Q}_1^T\boldsymbol{B}\boldsymbol{Z}_1 & \boldsymbol{Q}_1^T\boldsymbol{B}\boldsymbol{Z}_2 \\ \boldsymbol{0} & \boldsymbol{S}\end{pmatrix}\begin{pmatrix}\boldsymbol{Z}_1^T\boldsymbol{v} \\ \boldsymbol{Z}_2^T\boldsymbol{v}\end{pmatrix} = \begin{pmatrix}\boldsymbol{Q}_1^T\boldsymbol{y} \\ \boldsymbol{Q}_2^T\boldsymbol{y}\end{pmatrix}.$$

  From the bottom half we can solve for $\boldsymbol{v}$ from the equation (how?)

$$\boldsymbol{S}\boldsymbol{Z}_2^T\boldsymbol{v} = \boldsymbol{Q}_2^T\boldsymbol{y}.$$

  Then we solve for $\boldsymbol{\beta}$ from the equation

$$\boldsymbol{R}_1\boldsymbol{\beta} = \boldsymbol{Q}_1^T\boldsymbol{y} - (\boldsymbol{Q}_1^T\boldsymbol{B}\boldsymbol{Z}_1\boldsymbol{Z}_1^T + \boldsymbol{Q}_1^T\boldsymbol{B}\boldsymbol{Z}_2\boldsymbol{Z}_2^T)\boldsymbol{v} = \boldsymbol{Q}_1^T\boldsymbol{y} - \boldsymbol{Q}_1^T\boldsymbol{B}\boldsymbol{Z}_2(\boldsymbol{Z}_2^T\boldsymbol{v}).$$

- Paige's method also works for singular $\boldsymbol{X}$ and $\boldsymbol{B}$ (using QR with column pivoting).
- MATLAB's `lscov()` function implements Paige's method for singular covariance $\boldsymbol{V}$. No R implementation (?)

Ridge regression

- In ridge regression, we minimize

$$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2,$$

  where λ is a tuning parameter.

- Ridge regression by augmented linear regression. Ridge regression problem is equivalent to

$$\left\| \begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{0}_p \end{pmatrix} - \begin{pmatrix} \boldsymbol{X} \\ \sqrt{\lambda}\boldsymbol{I}_p \end{pmatrix} \boldsymbol{\beta} \right\|_2^2.$$

  Therefore any methods for linear regression can be applied.

- Ridge regression by method of normal equation. The normal equation for the ridge problem is

$$(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}_p)\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{y}.$$

  Therefore Cholesky or sweep can be used.

- Ridge regression by SVD. If we obtain the (thin) SVD of $\boldsymbol{X}$

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}_{p\times p}\boldsymbol{V}^T.$$

  Then the normal equation reads

$$(\boldsymbol{\Sigma}^2 + \lambda\boldsymbol{I}_p)\boldsymbol{V}^T\boldsymbol{\beta} = \boldsymbol{\Sigma}\boldsymbol{U}^T\boldsymbol{y}$$

  and we get

$$\widehat{\boldsymbol{\beta}}(\lambda) = \sum_{i=1}^{p} \frac{\sigma_i \boldsymbol{u}_i^T\boldsymbol{y}}{\sigma_i^2 + \lambda}\boldsymbol{v}_i = \sum_{i=1}^{r} \frac{\sigma_i \boldsymbol{u}_i^T\boldsymbol{y}}{\sigma_i^2 + \lambda}\boldsymbol{v}_i, \quad r = \text{rank}(\boldsymbol{X}).$$

It is clear that

$$\lim_{\lambda \to 0} \widehat{\boldsymbol{\beta}}(\lambda) = \widehat{\boldsymbol{\beta}}_{\text{OLS}}$$

and $\|\widehat{\boldsymbol{\beta}}(\lambda)\|_2$ is monotone decreasing as λ increases.

- Only one SVD is needed for all λ (!), in contrast to the method of augmented linear regression, Cholesky, or sweep.

Least squares over a sphere

- Ridge regression "shrinks" the solution via penalty. Alternatively we can simply fit a least squares problem subject to the constraint that the solution lives in a sphere

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 \\ \text{subject to} \quad & \|\boldsymbol{\beta}\|_2 \leq \alpha. \end{aligned}$$

- Suppose we obtain the (thin) SVD $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}_{p\times p}\boldsymbol{V}^T$. If the ordinary least squares solution

$$\widehat{\boldsymbol{\beta}}_{\text{OLS}} = \sum_{i=1}^{r} \frac{\boldsymbol{u}_i^T \boldsymbol{y}}{\sigma_i} \boldsymbol{v}_i$$

has ℓ_2 norm less than α, then we are done. If not, we use the method of Lagrangian multipliers

$$\psi(\boldsymbol{\beta}, \lambda) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \frac{\lambda}{2}(\|\boldsymbol{\beta}\|_2^2 - \alpha^2).$$

Setting the gradient to 0, we have the shifted normal equation

$$(\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}_p)\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{y},$$

which has solution

$$\widehat{\boldsymbol{\beta}}(\lambda) = \sum_{i=1}^{r} \frac{\sigma_i \boldsymbol{u}_i^T \boldsymbol{y}}{\sigma_i^2 + \lambda} \boldsymbol{v}_i.$$

We need to choose the λ such that $\|\widehat{\boldsymbol{\beta}}(\lambda)\|_2 = \alpha$. That is we need to find the (unique) zero of the function

$$f(\lambda) = \|\widehat{\boldsymbol{\beta}}(\lambda)\|_2^2 - \alpha^2 = \sum_{i=1}^{r} \left(\frac{\sigma_i \boldsymbol{u}_i^T \boldsymbol{y}}{\sigma_i^2 + \lambda} \right)^2 - \alpha^2.$$

This is easily achieved by Newton's or other methods.

Least squares with equality constraints

- In many applications, there are *a priori* constraints on the regression parameters. Let's consider how to solve linear regression with equality constraints (LSE)

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 \\ \text{subject to} \quad & \boldsymbol{B}\boldsymbol{\beta} = \boldsymbol{d}. \end{aligned}$$

- LSE by QR. First compute QR of $\boldsymbol{B}^T \in \mathbb{R}^{p\times m}$

$$\boldsymbol{B}^T = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix}$$

and set

$$\boldsymbol{X}\boldsymbol{Q} = (\boldsymbol{X}_1, \boldsymbol{X}_2) \quad \text{and} \quad \boldsymbol{Q}^T\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \end{pmatrix}.$$

Then the original minimization problem becomes

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{y} - \boldsymbol{X}_1\boldsymbol{\beta}_1 - \boldsymbol{X}_2\boldsymbol{\beta}_2\|_2^2 \\ \text{subject to} \quad & \boldsymbol{R}^T\boldsymbol{\beta}_1 = \boldsymbol{d}. \end{aligned}$$

Now $\boldsymbol{\beta}_1$ is determined from the constraint $\boldsymbol{R}^T\boldsymbol{\beta}_1 = \boldsymbol{d}$ and $\boldsymbol{\beta}_2$ is solved from the unconstrained least squares problem

$$\text{minimize} \quad \|(\boldsymbol{y} - \boldsymbol{X}_1\boldsymbol{\beta}_1) - \boldsymbol{X}_2\boldsymbol{\beta}_2\|_2^2.$$

Finally we recover the solution from

$$\boldsymbol{\beta} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \end{pmatrix}.$$

- LSE by augmented system. Define the Lagrangian function

$$\phi(\boldsymbol{\beta}, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 - \boldsymbol{\lambda}^T(\boldsymbol{B}\boldsymbol{\beta} - \boldsymbol{d}).$$

Setting gradient to zero yields

$$\begin{aligned} \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{B}^T\boldsymbol{\lambda} &= \boldsymbol{X}^T\boldsymbol{y} \\ \boldsymbol{B}\boldsymbol{\beta} &= \boldsymbol{d}, \end{aligned}$$

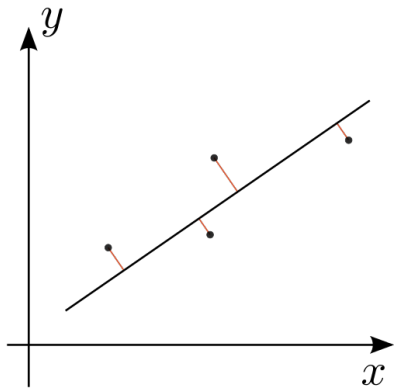
suggesting the augmented system

$$\begin{pmatrix} \boldsymbol{X}^T\boldsymbol{X} & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta} \\ -\boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}^T\boldsymbol{y} \\ \boldsymbol{d} \end{pmatrix}.$$

This linear system is non-singular when $\boldsymbol{X}$ and $\boldsymbol{B}$ have full rank and can be solved by Cholesky, sweep, and so on.

- LSE by generalized SVD.

Total least squares (TLS)



TLS considers the case both predictors and observations are subject to errors. It is solved by SVD. Read KL 9.3.6 if interested.

Tikhonov regularization

Tikhonov regularization is an extension of the ridge regression

$$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{B}\boldsymbol{\beta}\|_2^2,$$

where $\boldsymbol{B} \in \mathbb{R}^{m\times p}$ is a fixed regularization matrix and λ is a tuning parameter. It is solved by the generalized singular value decomposition (GSVD).

Least squares with quadratic inequality constraint (LSQI)

Least squares with quadratic inequality constraint (LSQI) minimizes the least squares criterion over a hyper-ellipsoid:

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 \\ \text{subject to} \quad & \|\boldsymbol{B}\boldsymbol{\beta}\|_2 \leq \alpha, \end{aligned}$$

where $\boldsymbol{B} \in \mathbb{R}^{m \times p}$ is a fixed regularization matrix. It is solved by the generalized singular value decomposition (GSVD). See Golub and Van Loan (1996, Section 2.1.1).

Concluding remarks on numerical linear algebra

- Numerical linear algebra forms the building blocks of most computation we do. Most lines of our computational code are numerical linear algebra.
- Be flop and memory aware.

  > The form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.

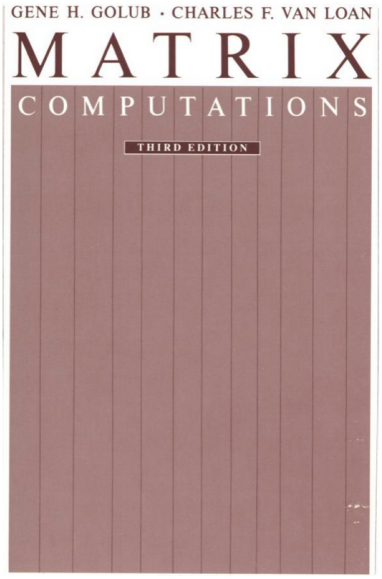
- Be alert to problem structure and make educated choice of software/algorithm.

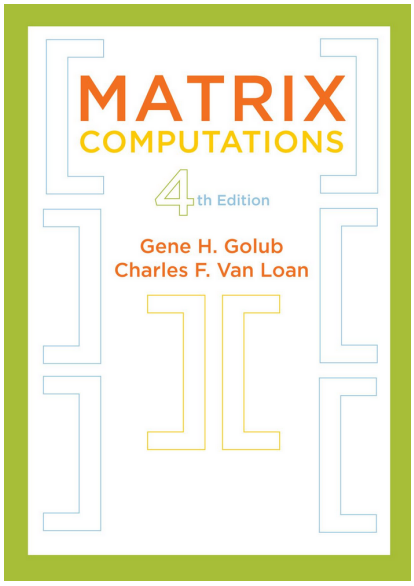
  > The structure should be exploited whenever solving a problem.

- Do not write your own matrix computation routines unless for good reason. Utilize `BLAS` and `LAPACK` as much as possible!
- In contrast, for optimization, often we need to devise problem specific optimization routines, or even "mix and match" them.

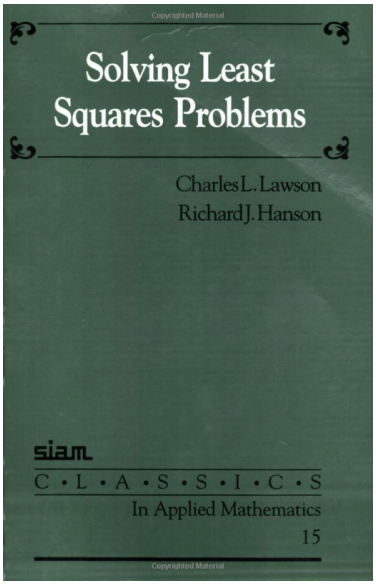
Some useful reference books on (numerical) linear algebra

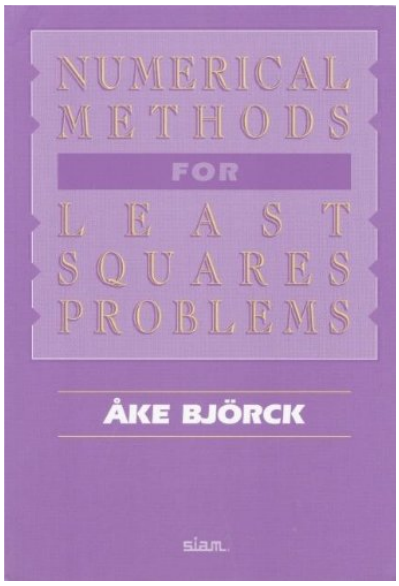
- Golub and Van Loan (1996): "Bible" in numerical linear algebra. Good for reference.



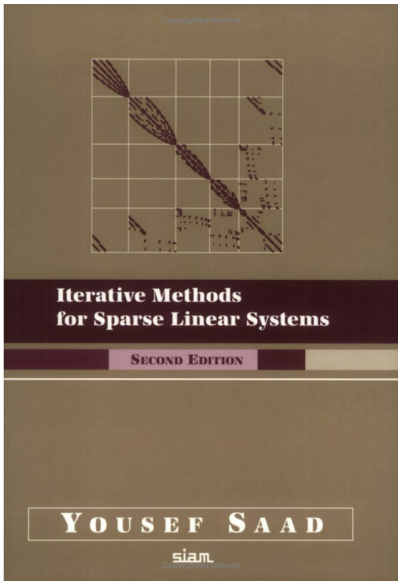


- Lawson and Hanson (1987) and Björck (1996): classical monographs on solving least squares problems.





- Saad (2003): standard reference for iterative methods


Iterative Methods
for Sparse Linear Systems
Second Edition
Yousef Saad
siam