9 Lecture 9, Feb 2

Announcements

- HW1 graded. Feedback:
  - Sketch of solution: `http://hua-zhou.github.io/teaching/biostatm280-2016winter/hw01sol.html`. Please compare to your code carefully and understand why.
  - Q2: *Rounding to even rule* in R.
  - Q5: be familiar with R functions: `crossprod`, `tcrossprod`, `outer`, `row`, `col`, `rowSums`, `colSums`, ...
  - Q6: vectorize code (no looping necessary).
  - Code style. Be professional!
  - Frequent commits in git. I want to see how you developed the solutions.
- HW2 due today @ 11:59PM.
- HW3 posted. Due Feb 11 @ 11:59PM. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/biostat_m280_2016_hw3.pdf`
- Quiz 2 this Thu Feb 4. In class, closed book.

Last time

- Linear regression by Cholesky. $np^2 + p^3/3$ flops for $\widehat{\boldsymbol{\beta}}$ or more for s.e.
- Linear regression by QR $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$.
- QR by MGS. $2np^2$ flops to obtain $\boldsymbol{Q}_1$ and $\boldsymbol{R}$.
- QR by Householder. $2np^2 - 2p^3/3$ flops to overwrite $\boldsymbol{X}$ by $\boldsymbol{R}$ and Householder vectors. It is the algorithm R uses for linear regression.
- QR by Givens. $3np^2 - p^3$ to overwrite $\boldsymbol{X}$ by $\boldsymbol{R}$ and Givens rotations.

Today

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

Sweep operator

Assume $\boldsymbol{A} \succeq \mathbf{0}_{n\times n}$.

- The popular statistical software SAS uses sweep operator for linear regression and matrix inversion.
- Read KL 7.4-7.6 for introduction.

  Also see "*A tutorial on the SWEEP operator*" by James H. Goodnight. `http://www.jstor.org/stable/2683825`
- *Sweep* on the k-th diagonal entry $a_{kk} \neq 0$ yields $\hat{\boldsymbol{A}}$ with entries

$$\begin{aligned}
\hat{a}_{kk} &= -\frac{1}{a_{kk}} \\
\hat{a}_{ik} &= \frac{a_{ik}}{a_{kk}} \\
\hat{a}_{kj} &= \frac{a_{kj}}{a_{kk}} \\
\hat{a}_{ij} &= a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}}, \quad i \neq k, j \neq k.
\end{aligned}$$

  n^2 flops (taking into account of symmetry).
- *Inverse sweep* sends $\boldsymbol{A}$ to $\check{\boldsymbol{A}}$ with entries

$$\begin{aligned}
\check{a}_{kk} &= -\frac{1}{a_{kk}} \\
\check{a}_{ik} &= -\frac{a_{ik}}{a_{kk}} \\
\check{a}_{kj} &= -\frac{a_{kj}}{a_{kk}} \\
\check{a}_{ij} &= a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}}, \quad i \neq k, j \neq k.
\end{aligned}$$

n^2 flops (taking into account of symmetry).

- $\check{\hat{\boldsymbol{A}}} = \boldsymbol{A}$.
- Successively sweeping all diagonal entries of $\boldsymbol{A}$ yields $-\boldsymbol{A}^{-1}$.
- Exercise: invert a 2×2 matrix, say $\boldsymbol{A} = \begin{pmatrix} 4 & 3 \\ 3 & 2 \end{pmatrix}$ on paper using sweep operator.
- Block form of sweep: Let the symmetric matrix $\boldsymbol{A}$ be partitioned as $\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{pmatrix}$. If possible, sweep on the diagonal entries of $\boldsymbol{A}_{11}$ yields

$$\hat{\boldsymbol{A}} = \begin{pmatrix} -\boldsymbol{A}_{11}^{-1} & \boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12} \\ \boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1} & \boldsymbol{A}_{22} - \boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12} \end{pmatrix}.$$

  Order dose *not* matter. The block $\boldsymbol{A}_{22} - \boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12}$ is recognized as the *Schur complement* of $\boldsymbol{A}_{11}$.

- Pd and determinant: $\boldsymbol{A}$ is pd if and only if each diagonal entry can be swept in succession and is positive until it is swept. When a diagonal entry of a pd matrix $\boldsymbol{A}$ is swept, it becomes negative and remains negative thereafter. Taking the product of diagonal entries just before each is swept yields the determinant of $\boldsymbol{A}$.
- Linear regression by sweep. Sweep on $\begin{pmatrix} \boldsymbol{X}^\top\boldsymbol{X} & \boldsymbol{X}^\top\boldsymbol{y} \\ \boldsymbol{y}^\top\boldsymbol{X} & \boldsymbol{y}^\top\boldsymbol{y} \end{pmatrix}$ yields

$$\begin{pmatrix} -(\boldsymbol{X}^\top\boldsymbol{X})^{-1} & (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \\ \boldsymbol{y}^\top\boldsymbol{X}(\boldsymbol{X}\boldsymbol{X})^{-1} & \boldsymbol{y}^\top\boldsymbol{y} - \boldsymbol{y}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \end{pmatrix} = \begin{pmatrix} -\frac{1}{\sigma^2}\mathrm{Var}(\hat{\boldsymbol{\beta}}) & \hat{\boldsymbol{\beta}} \\ \hat{\boldsymbol{\beta}}^\top & \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 \end{pmatrix}.$$

  In total $np^2 + p^3$ flops.

- Sweep is useful for stepwise regression, (conditional) multivariate normal density calculation, MANOVA, ...
- LAPACK does *not* implement sweep operator. It is statisticians' secret tool.
- Warning: the `sweep()` function in R has nothing do to with the sweep operator here.
- Demo code: `http://hua-zhou.github.io/teaching/biostatm280-2016winter/sweep.html`

Summary of linear regression: Table on KL p105

Method	Flops	Remarks	Software	Stability
Sweep	$np^2 + p^3$	$(\boldsymbol{X}^\intercal \boldsymbol{X})^{-1}$ available	SAS	less stable
Cholesky	$np^2 + p^3/3$			less stable
QR by Householder	$2np^2 - 2p^3/3$		R	stable
QR by MGS	$2np^2$	$\boldsymbol{Q}_1$ available		most stable

Table 1: Numerical methods for linear regression. In order of stability.

Remarks:

- When $n \gg p$, sweep and Cholesky are twice faster than QR and need less space.
- Sweep and Cholesky is based on the Gram matrix $\boldsymbol{X}^T\boldsymbol{X}$, which can be dynamically updated with incoming data. They can easily handle huge n, moderate p data sets that cannot fit into memory.
- QR methods are more stable and produce numerically more accurate solution.
- Although sweep is slower than Cholesky, it yields standard errors and so on.
- Sweep is useful for stepwise regression, multivariate normal calculation, and numerous other statistical applications.
- MGS appears slower than Householder, but it yields $\boldsymbol{Q}_1$.

> There is simply no such thing as a universal 'gold standard' when it comes to algorithms.
>
> anonymous reviewer

Summary of solving linear equations

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

- We now know some good numerical methods for the least squares problem, which is essentially "solving" an overdetermined system (a tall $\boldsymbol{A}$).

- Table 2 compares the flops of some methods (in order of stability) for solving a square (unstructured) $\boldsymbol{A} \in \mathbb{R}^{n\times n}$.

Method	Flops	Stability
Gaussian elimination	$2n^3/3$	less stable
QR by Householder	$4n^3/3$	
QR by MGS	n^3	
SVD	$6n^3$	most stable

Table 2: Flops of different numerical methods for $n\times n$ square linear systems, assuming availability of the right hand side at time of decomposition.

- Solve an underdetermined system (a flat $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ of full row rank) by QR – version 1. First compute QR on $\boldsymbol{A}^T$

$$\boldsymbol{A}^T = \boldsymbol{Q}\boldsymbol{R} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0}_{(n-m)\times m} \end{pmatrix}.$$

  Then $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ becomes

$$(\boldsymbol{Q}\boldsymbol{R})^T\boldsymbol{x} = \begin{pmatrix} \boldsymbol{R}_1^T & \boldsymbol{0}_{m\times(n-m)} \end{pmatrix}\begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix} = \boldsymbol{b},$$

  where

$$\boldsymbol{Q}^T\boldsymbol{x} = \begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix}.$$

  $\boldsymbol{z}_1$ is determined from $\boldsymbol{R}_1^T\boldsymbol{z}_1 = \boldsymbol{b}$. If we take $\boldsymbol{z}_2 = \boldsymbol{0}_{n-m}$, then we obtain the minimum norm solution (why?).

- Solve an underdetermined system (a flat $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ of full row rank) by QR – version 2. First compute QR with column pivoting on $\boldsymbol{A}$

$$\boldsymbol{A}\boldsymbol{\Pi} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{R}_1 & \boldsymbol{R}_2 \end{pmatrix},$$

  where $\boldsymbol{R}_1 \in \mathbb{R}^{m\times m}$ is upper triangular and $\boldsymbol{R}_2 \in \mathbb{R}^{m\times(n-m)}$. Thus $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ transforms to

$$\begin{pmatrix} \boldsymbol{R}_1 & \boldsymbol{R}_2 \end{pmatrix}\begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix} = \boldsymbol{Q}^T\boldsymbol{b},$$

where

$$\boldsymbol{\Pi}^T \boldsymbol{x} = \begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix}.$$

One solution is obtained by $\boldsymbol{z}_1 = \boldsymbol{R}_1^{-1}\boldsymbol{Q}^T\boldsymbol{b}$ and $\boldsymbol{z}_2 = \boldsymbol{0}_{n-m}$. It is not guaranteed to be of minimum norm.

Condition number for linear equations (matrix inversion)

- Assume $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is nonsingular and consider the system of linear equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. The solution is $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$. We want to know how the solution changes with a small perturbation of the input $\boldsymbol{b}$ (or $\boldsymbol{A}$).
- Let $\tilde{\boldsymbol{b}} = \boldsymbol{b} + \Delta\boldsymbol{b}$. Then $\tilde{\boldsymbol{x}} = \boldsymbol{A}^{-1}(\boldsymbol{b} + \Delta\boldsymbol{b}) = \boldsymbol{x} + \Delta\boldsymbol{x}$. Thus

  $$\|\Delta\boldsymbol{x}\| = \|\boldsymbol{A}^{-1}\Delta\boldsymbol{b}\| \le \|\boldsymbol{A}^{-1}\|\|\Delta\boldsymbol{b}\|.$$

  Because $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}$, $\frac{1}{\|\boldsymbol{x}\|} \le \|\boldsymbol{A}\|\frac{1}{\|\boldsymbol{b}\|}$. This results

  $$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \le \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|\frac{\|\Delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|}.$$

- $\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$ is called the *condition number for inversion*. It depends on the matrix norm being used. κ_p means condition number defined from matrix-p norm.
- Large condition number means "bad".
- Some useful facts

  $$\begin{aligned}
  \kappa(\boldsymbol{A}) &= \kappa(\boldsymbol{A}^{-1}) \\
  \kappa(c\boldsymbol{A}) &= \kappa(\boldsymbol{A}) \\
  \kappa(\boldsymbol{A}) &\ge 1 \\
  \kappa_1(\boldsymbol{A}) &= \kappa_\infty(\boldsymbol{A}^\intercal) \\
  \kappa_2(\boldsymbol{A}) &= \kappa_2(\boldsymbol{A}^\intercal) = \frac{\sigma_1(\boldsymbol{A})}{\sigma_n(\boldsymbol{A})} \\
  \kappa_2(\boldsymbol{A}^\intercal\boldsymbol{A}) &= \frac{\lambda_1(\boldsymbol{A}^\intercal\boldsymbol{A})}{\lambda_n(\boldsymbol{A}^\intercal\boldsymbol{A})} = \kappa_2^2(\boldsymbol{A}) \ge \kappa_2(\boldsymbol{A}).
  \end{aligned}$$

  The last fact says that the condition number of $\boldsymbol{A}^\intercal\boldsymbol{A}$ can be much larger than that of $\boldsymbol{A}$.

- The smallest singular value σ_n indicates the "distance to the trouble".
- Condition number for the least squares problem is more complicated. Roughly speaking, the method based on normal equation (Cholesky, sweep) has a condition depending on $\kappa_2(\boldsymbol{X})^2$. QR for a "small residuals" least squares problem has a condition depending on $\kappa_2(\boldsymbol{X})$.
- Numerically, consider the simple case

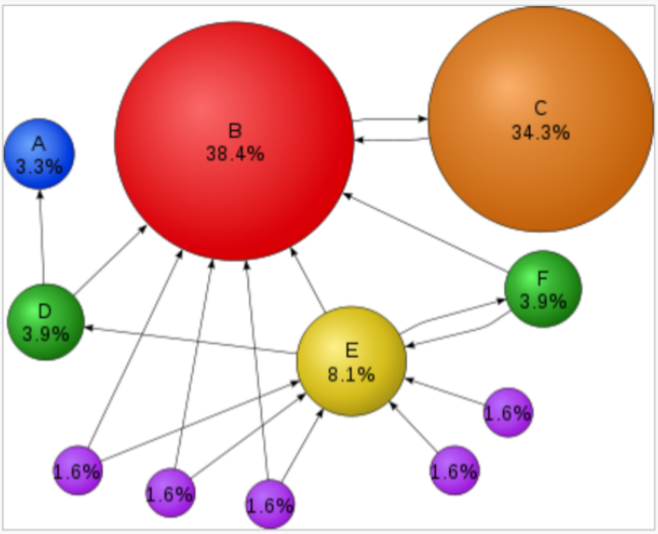
$$\boldsymbol{X} = \begin{pmatrix} 1 & 1 \\ 10^{-3} & 0 \\ 0 & 10^{-3} \end{pmatrix}.$$

  Forming normal equation yields a singular Gramian matrix

$$\boldsymbol{X}^\top \boldsymbol{X} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

  if executed with a precision of 6 digits.
- In `R`, the `kappa()` function (wrapper of `DTRCON` in LAPACK and `DTRCO` in LINPACK) computes or approximates (default) the condition number of a matrix.
- In regression problems, standardizing the predictors could improve the condition. See demo on the Longley data `http://hua-zhou.github.io/teaching/biostatm280-2016winter/longleycond.html`.
- In design of experiments (DoE), people favor orthogonal design. Why?

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

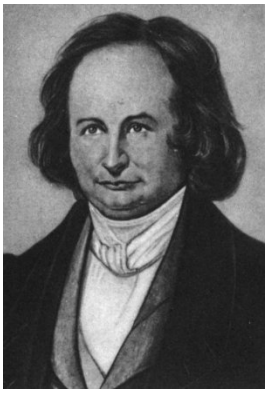
$$a_{ij} = \begin{cases} 1 & \text{if page } i \text{ links to page } j \\ 0 & \text{otherwise} \end{cases}.$$

    $n \approx 10^9$ 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 $\approx 2 \times 10^7$ 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()`.