NUMERICAL OPTIMIZATION

BASICS OF NUMERICAL LINEAR ALGEBRA

Alberto Bemporad

http://cse.lab.imtlucca.it/~bemporad/optimization_course.html

IMT SCHOOL FOR ADVANCED STUDIES LUCCA

Numerical Optimization - ©2025 A. Bemporad. All rights reserved.

NUMERICAL LINEAR ALGEBRA

- Before diving into optimization algorithms we need to recall some basic facts of **numerical linear algebra**

- We will review basic numerical techniques of **matrix factorization** and **solution of linear systems**, which are key ingredients for most optimization algorithms

- We also want to keep in mind the **flops** (floating point operations) involved in those numerical techniques, where 1 flop = one operation ($+$, $-$, $*$ or $\div$)[1]

- When counting flops, we usually consider only the largest terms

- Excellent textbook: **Golub-Van Loan, ``Matrix Computations'' (4th ed.), 2012**

[1] We do not consider Single Instruction Multiple Data (SIMD) capabilities of modern processors

BASIC MATRIX OPERATIONS

- The **inner product** $x'y, x, y \in \mathbb{R}^n$, requires $2n-1 \approx 2n$ flops
- The **sum** $x+y$ and **multiplication by a scalar** αx require n flops
- The **matrix-vector** product Ax, $A \in \mathbb{R}^{m\times n}$, requires:
  - $m(2n-1) \approx 2mn$ flops, or
  - $2N$ flops if A is sparse with N nonzero entries, or
  - $2p(n+m)$ flops if $U \in \mathbb{R}^{m\times p}, V \in \mathbb{R}^{n\times p}$ are given such that $A = UV'$ ($Ax = U(V'x)$). This can be useful when $p, n \leq m$
- The **matrix-matrix product** $C = AB$ with $B \in \mathbb{R}^{n\times p}$, requires $mp(2n-1) \approx 2mnp$ flops (or $\approx m^2 n$ if C is symmetric, $m = p$)

Numerical Optimization - ©2025 A. Bemporad. All rights reserved.

SOLVING LINEAR SYSTEMS AX=B

- We want to solve the **square linear system** $Ax = b$, $A \in \mathbb{R}^{n \times n}$, $\det A \neq 0$
- If A is **diagonal**, $A = \mathrm{diag}(a)$, we get $x_i = \frac{b_i}{a_i}$, which requires n flops
- If A is **lower triangular**, $A_{ij} = 0$ for $j > i$, we can compute $x = A^{-1}b$ with $\sum_{k=1}^{n} 2(k-1)+1 = n^2$ flops by **forward substitution**:

$$\begin{aligned} x_1 &= b_1/a_{11} \\ x_2 &= (b_2 - a_{21}x_1)/a_{22} \\ \vdots & \quad \vdots \\ x_n &= (b_n - \textstyle\sum_{i=1}^{n-1} a_{ni}x_i)/a_{nn} \end{aligned} \qquad A = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{12} & a_{22} & \dots 0 & \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

- Similarly, if A is **upper triangular**, $A_{ij} = 0$ for $j < i$, computing $x = A^{-1}b$ requires n^2 flops by **backward substitution**

SOLVING LINEAR SYSTEMS AX=B

- Let A be a **Householder matrix**, $A = I - 2vv'$, $\|v\|_2 = 1$. Since $A^{-1} = A$, $x = A^{-1}b = Ab = b - 2(v'b)v$ requires $4n$ flops

- If A is a **permutation matrix** (=permutation of the columns of I) then $A^{-1} = A'$, and $x = A^{-1}b$ requires 0 flops

  Example:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \; A^{-1} = A' = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \; A^{-1}x = \begin{bmatrix} x_3 \\ x_1 \\ x_4 \\ x_2 \end{bmatrix}$$

LU FACTORIZATION

Definition

A square matrix $A \in \mathbb{R}^{n \times n}$ is **diagonally nonsingular** if all its leading principal submatrices $A_k \in \mathbb{R}^{k \times k}$, $a_{k,ij} = a_{ij}$, are nonsingular, $\forall k = 1, \ldots, n$

- A **diagonally nonsingular matrix** A can be factorized as $A = LU$, where L = lower triangular, U = upper triangular
- The LU factorization requires $\approx \frac{2}{3}n^3$ flops
- Hence we can solve the linear system $Ax = L(Ux) = b$ as follows:
  - Solve $Ly = b$ (n^2 flops)
  - Solve $Ux = y$ (n^2 flops)
- The total cost for solving the linear system is $\frac{2}{3}n^3 + 2n^2 \approx \frac{2}{3}n^3$ flops

LU FACTORIZATION

- Every **nonsingular matrix** A can be factorized as $A = PLU$,
  where P = permutation matrix, L = lower triangular, U = upper triangular

- The permutation matrix P adds no flop in solving $Ax = b$:
  $Pz = b$ (0 flops), $Ly = z$ (n^2 flops), $Ux = y$ (n^2 flops)

- Note that when solving N linear systems $Ax = b^k$, $k = 1, 2, \ldots, N$, we only need to compute the LU factorization once.

  For example in **iterative refinement** (see later) we improve the precision of a solution x_0 of $Ax = b$ by iterating

$$\begin{aligned} r_k &= b - Ax_k \\ Ad_k &= r_k \\ x_{k+1} &= x_k + d_k, \quad k = 0, \ldots \end{aligned}$$

CHOLESKY FACTORIZATION

- Every **symmetric positive definite matrix** A admits the **Cholesky factorization** $A = LL'$. This requires $\approx \frac{1}{3}n^3$ flops and n square roots, where L = lower triangular matrix (Demmel, 1989)

André-Louis Cholesky
(1875–1918)

- Hence the linear system $Ax = L(L'x)) = b$ can be solved by
  - solving $Ly = b$ (n^2 flops)
  - solving $L'x = y$ (n^2 flops)

- The total cost for solving the linear system $Ax = b$ is $\frac{1}{3}n^3 + 2n^2 \approx \frac{1}{3}n^3$ flops

- Again, when solving N linear systems $Ax = b^k$, $k = 1, 2, \ldots, N$ the Cholesky factorization is only computed once

- Efficient sparse versions of the Cholesky factorization algorithm exist

LDL' FACTORIZATION

- Every **diagonally nonsingular and symmetric matrix** A can be factorized as $A = LDL'$ with $\approx \frac{1}{3}n^3$ flops and no square root, where D = **diagonal** and L = **unit lower-triangular** ($L_{ii} = 1$)
- We can solve the linear system $Ax = L(D(L'x)) = b$ as follows:
  - Solve $Lz = b$ (n^2 flops)
  - Solve $Dy = z$ (n flops)
  - Solve $L'x = y$ (n^2 flops)
- The total cost for solving the linear system is $\frac{1}{3}n^3 + 2n^2 + n \approx \frac{1}{3}n^3$ flops
- Every **symmetric positive definite matrix** A is diagonally nonsingular, therefore

$$A = A' \succ 0 \;\Rightarrow\; A = LDL'$$

LDL' FACTORIZATION

- Every **nonsingular symmetric matrix** A can be factorized as

$$PAP' = LDL'$$

with $\approx \frac{1}{3}n^3$ flops and no square root, where P = permutation matrix, L = unit lower triangular, D = block diagonal with 1×1 or 2×2 blocks

- When A is sparse, techniques exist to choose P such that L is sparse, so the solution cost is $\ll \frac{1}{3}n^3$

- Again, when solving N linear systems $Ax = b^k$, $k = 1, 2, \ldots, N$ the LDL' factorization is only computed once

MATRIX INVERSION LEMMA

Lemma

Let $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{m\times m}$, $D \in \mathbb{R}^{m\times n}$, and let A, C, and $C^{-1} + DA^{-1}B$ be nonsingular. Then

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

- Assume the structure of A and C is such that $Ax = b$ is easy to solve (e.g., A block diagonal) and C is easily invertible (e.g., $C = I$ or $m \ll n$)
- Then we can solve $x = (A + BCD)^{-1}b$ as follows:
  - Solve $Ad = b$ with respect to d
  - Get matrix $E = A^{-1}B$ by solving $Az = B_i$ = ith column of B, $i = 1, \ldots, m$
  - Solve $(C^{-1} + DE)y = Dd$ with respect to $y \in \mathbb{R}^m$
  - Solve $Ax = b - By$ with respect to x
- This is very useful when $m \ll n$. For $m = C = 1$ the inversion formula reduces to **Sherman-Morrison's formula** $(A + bd')^{-1} = A^{-1} - \frac{A^{-1}bd'A^{-1}}{1+d'A^{-1}b}$

QR FACTORIZATION

Theorem

Any matrix $A \in \mathbb{R}^{m \times n}$ can be factorized as

$$A = QR$$

where $Q \in \mathbb{R}^{m \times m}$ is orthogonal ($Q'Q = I$) and $R \in \mathbb{R}^{m \times n}$ is upper triangular.

- The number of nonzero diagonal entries of R is equal to $\operatorname{rank} A$
- In case of **overdetermined linear systems** $Ax = b, m > n$, we get

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}, \; Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}, \; \begin{bmatrix} R_1 x \\ 0 \end{bmatrix} = \begin{bmatrix} Q_1' b \\ Q_2' b \end{bmatrix}$$

- If $Q_2' b = 0$ the system is solvable by solving the triangular system

$$R_1 x = Q_1' b$$

- The factorization $A = Q_1 R_1$ is called **economy-size QR factorization** of A

QR FACTORIZATION

- **Q-less factorization**: if we factorize $[A\ b] = Q[R_1\ r_2]$ we can avoid computing and storing Q explicitly in order to solve $Ax = b$:

$$Ax - b = \begin{bmatrix} A & b \end{bmatrix} \begin{bmatrix} x \\ -1 \end{bmatrix} = Q \begin{bmatrix} R_1 & r_2 \end{bmatrix} \begin{bmatrix} x \\ -1 \end{bmatrix} = 0 \Leftrightarrow R_1 x = r_2$$

- There are different algorithms to compute $A = QR$

  (Lawson, Hanson, 1974) (Golub, Van Loan, 2012)

- The QR factorization is useful to solve **least-squares problems**:

$$\|Ax - b\|_2^2 = \|QRx - b\|_2^2 = \left\| \begin{bmatrix} R_1 x - Q_1' b \\ -Q_2' b \end{bmatrix} \right\|_2^2 = \|R_1 x - Q_1' b\|_2^2 + \|Q_2' b\|_2^2$$

  is minimized for $x^* = R_1^{-1} Q_1' b$

- Alternatives: solve the **normal equations** $A'A = A'b$ by factorizing $A'A = L'L$ (Cholesky) or $A'A = LDL'$. Or use SVD decomposition (see next slides)

SINGULAR VALUE DECOMPOSITION (SVD)

- Every matrix $A \in \mathbb{R}^{m \times n}$ can be decomposed as

$$A = U \Sigma V'$$

with $U'U = I, V'V = I, \Sigma_{ii} \geq 0, \Sigma_{ij} = 0, \forall i \neq j$

- The diagonal entries σ_i of Σ are called the **singular values** of A.
They are usually defined in descending order ($\sigma_i \geq \sigma_j$ for $i \leq j$)

- If $\sigma_i = 0$ for $i = r+1, \ldots, \min(n, m)$, then $\text{rank}(A) = r$

- Computing the SVD require $\alpha mn^2 + \beta n^3 + \gamma nm^2$ flops, where α, β, γ depend on the algorithm used and whether only some of Σ, U, V are required

(Golub, Van Loan, 2012)

SINGULAR VALUE DECOMPOSITION (SVD)

- Since $V'v_i = e_i$ (v_i = ith column of V, e_i = ith column of I) and $\Sigma e_i = 0$ for $i > r$ the last $n - r$ columns of V are a basis of the **kernel** (null-space) of A

- Since $y = Ax = U \begin{bmatrix} z_1 & \dots & z_r & 0 & \dots & 0 \end{bmatrix}'$, where x arbitrary and V nonsingular make z_i arbitrary, the first r columns of U are a basis of the **image** (range) of A

- When A **invertible**, $A^{-1} = (U\Sigma V')^{-1} = V\Sigma^{-1}U'$

- When A is **symmetric**, $U = V$ are a basis of eigenvectors of A and $\sigma_i = |\lambda_i|$, with λ_i = eigenvalues of A, $\lambda_i \in \mathbb{R}$

SVD plays a fundamental role in many applications!

Numerical Optimization - ©2025 A. Bemporad. All rights reserved.

LEAST-SQUARES PROBLEMS AND SVD

- Let $A \in \mathbb{R}^{m\times n}, m > n, \operatorname{rank} A = r \leq n$ and factorize $A = U\Sigma V'$

$$\Sigma = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix}, \quad U = [U_1\, U_2], \quad V = [V_1\, V_2]$$
$$\Sigma_1 \in \mathbb{R}^{r\times r}, \quad U_1 \in \mathbb{R}^{m\times r}, \quad V_1 \in \mathbb{R}^{n\times r}$$

- Since U is an orthogonal matrix and Σ_1 invertible, we get

$$\begin{aligned} \arg\min_x \|Ax-b\|_2^2 &= \arg\min_x \|U\Sigma V'x - b\|_2^2 = \arg\min_x \|\Sigma V'x - U'b\|_2^2 \\ &= \arg\min_x \|[\Sigma_1\ 0] \begin{bmatrix} V_1' \\ V_2' \end{bmatrix} x - U_1'b\|_2^2 \end{aligned}$$

- Let $z = V'x = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, z_1 \in \mathbb{R}^r$ and set $[\Sigma_1\ 0]z = U_1'b \Rightarrow z_1 = \Sigma_1^{-1}U_1'b$

- The optimal solutions are given by

$$x^* = [V_1\, V_2] \begin{bmatrix} \Sigma_1^{-1}U_1'b \\ z_2 \end{bmatrix} = V_1\Sigma_1^{-1}U_1'b + V_2z_2, \quad z_2 \text{ free}$$

- For $r = n$, the solution is unique and equal to $x^* = V\Sigma_1^{-1}U_1'b$

PRINCIPAL COMPONENT ANALYSIS (PCA) AND SVD

- Given a dataset of N samples $x_i \in \mathbb{R}^n$, let $A \in \mathbb{R}^{N\times n}$ be the matrix whose row $A_i = (x_i - \bar{x})'$, where $\bar{x} = \sum_{i=1}^{n} x_i$ is the **empirical mean** of the data

- Compute the SVD $A = U\Sigma V'$

- The n columns of $V = [v_1 \ \ldots \ v_n]$ are called **principal components** and form an orthogonal basis of $\mathbb{R}^n$

- Why "principal" components ? Note that the components of $x_i - \bar{x}$ in the new basis are $U_i\Sigma = [U_{i1}\sigma_1 \ \ldots \ U_{in}\sigma_n]$, with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$

- Only the principal components v_i corresponding to "large-enough" singular values σ_i are useful to represent $x_i - \bar{x}$

LOW-RANK MATRIX APPROXIMATION AND SVD

- Let $m \leq n, U = [U_1\ U_2], \Sigma_1 = \text{diag}([\sigma_1\ \dots\ \sigma_m]), V = [V_1\ V_2]$, $\begin{matrix} U_1 \in \mathbb{R}^{N\times m} \\ V_1 \in \mathbb{R}^{n\times m} \end{matrix}$

- **Eckart–Young-Mirsky theorem**:

$$\hat{A}^* = U_1\Sigma_1 V_1' = \arg \min_{\hat{A}\in\mathbb{R}^{N\times n}} \|A - \hat{A}\|_F \quad \text{such that } \text{rank}(\hat{A}) = m$$

  is an optimal **low-rank approximation** of A, and $\|A - \hat{A}^*\|_F^2 = \sum_{i=m+1}^{n} \sigma_i^2$,

  where $\|A\|_F$ is the **Frobenius norm** of A (see next slide).

- Matrix $\hat{A}^*$ also minimizes the **spectral norm** (see next slide) $\|A - \hat{A}\|_2$, and $\|A - \hat{A}^*\|_2 = \sigma_{m+1}$

MATRIX NORMS AND CONDITION NUMBER

- A **matrix norm** is a norm on the vector space $\mathbb{R}^{m\times n}$
- The **Frobenius norm** of matrix a $A \in \mathbb{R}^{m\times n}$ is $\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}^2}$
- A matrix norm can be induced by a vector norm $\|x\|, x \in \mathbb{R}^n$ as

$$\|A\| = \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\| \qquad \textbf{induced norm}$$

- A key role in determining the numerical robustness of an (optimization) algorithm is the **condition number** of an invertible matrix $A \in \mathbb{R}^{n\times n}$

$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\|$$

- As $\|A\| \cdot \|A^{-1}\| \geq \|AA^{-1}\| = \|I\| = 1$, we always have $\text{cond}(A) \geq 1$

MATRIX NORMS AND CONDITION NUMBER

- If we use the **Euclidean norm** $\|x\|_2$ we get the **spectral norm**

$$\|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = \max_{\|x\|_2=1} \|U\Sigma V'x\|_2 = \max_{\|y\|_2=1} \|\Sigma y\|_2 = \sigma_{\max}(A)$$

and

$$\|A^{-1}\|_2 = \max_{\|x\|_2=1} \|V\Sigma^{-1}U'x\|_2 = \max_{\|y\|_2=1} \|\Sigma^{-1}y\|_2 = \frac{1}{\sigma_{\min}(A)}$$

- Therefore

$$\text{cond}(A) = \|A\|_2 \cdot \|A^{-1}\|_2 = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$$

- When A is **symmetric**

$$\text{cond}(A) = \frac{|\lambda_{\max}(A)|}{|\lambda_{\min}(A)|}$$

Roughly speaking, we say that A is **well-conditioned** if $\text{cond}(A) \approx 1$ and **ill-conditioned** if $\text{cond}(A) \gg 1$

NUMERICAL ROBUSTNESS

- Say for numerical errors we are solving $A(x+\delta x) = b + \delta b$ instead of $Ax = b$
- Since $Ax = b$ (exact solution), we get $\delta x = A^{-1}\delta b$
- Therefore, the relative error of the solution is

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\|\|\delta b\|}{\|x\|} \leq \frac{\|A^{-1}\|\|\delta b\|}{\|b\|/\|A\|} = \text{cond}(A)\frac{\|\delta b\|}{\|b\|}$$

- The larger $\text{cond}(A)$ the more an error in computing b propagates into an error in solving $Ax = b$
- Example: say we solve a dual QP and retrieve $x^* = -Q^{-1}(c + A'\lambda^*)$. Small errors in computing λ^* can become large errors in x^* if $\text{cond}(Q) \gg 1$

ITERATIVE REFINEMENT

(Golub and Wilkinson, 1966)

- We want to solve $Ax = b$ but A is ill-conditioned (even singular, but $Ax = b$ is solvable)
- Regularizing A and solving $(A + \epsilon I)x = b, \epsilon > 0$, will provide a different solution $x_0 = (A + \epsilon I)^{-1} b$
- Instead, we factorize $LL' = (A + \epsilon I)$ (any other factorization will work) and iterate the following from x_0 until the residual $b - Ax_k \approx 0$:

$$x_{k+1} = x_k + \underbrace{(A + \epsilon I)^{-1}(b - Ax_k)}_{\text{refinement}}$$

- Theoretically, $(b - Ax_k) \to 0$ for all $\epsilon > 0$
- Usually only a few steps are required if ϵ is properly chosen (large enough to compute L robustly, but not too large otherwise convergence is slow)

CONJUGATE GRADIENT METHOD

(Hestenes, Stiefel, 1952) (Nocedal, Wright, 2012, Algorithm 5.2)

- The conjugate gradient (CG) method is an **iterative method** for solving $Ax = b$ with A **symmetric and positive definite**
- Given an initial guess x_0 and residual $r_0 = Ax_0 - b, p_0 = -r_0$, the CG algorithm iterates the following steps until the residual $r_k \approx 0$:

$$\alpha_k = \frac{r_k' r_k}{p_k' A p_k}, \qquad \begin{bmatrix} x_{k+1} \\ r_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \alpha_k p_k \\ r_k + \alpha_k A p_k \end{bmatrix}$$

$$\beta_{k+1} = \frac{r_{k+1}' r_{k+1}}{r_k' r_k}, \qquad p_{k+1} = -r_{k+1} + \beta_{k+1} p_k$$

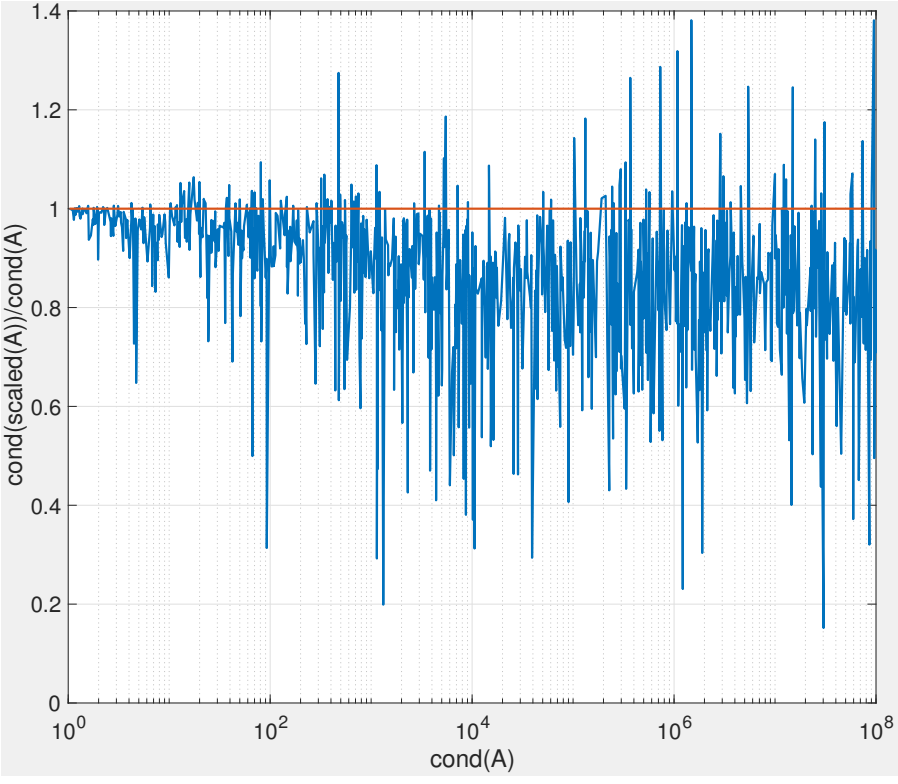
- The method is particularly useful when A is large, as it does not involve any factorization of A
- The method is **matrix-free** as A does not even need to be available, we only need to be able to compute Av
- The convergence speed of CG is sensitive to scaling of A, so it may require preconditioning

PRECONDITIONING

- The speed of convergence of many iterative methods is affected by the choice of coordinate system
- Let $A = A' \succ 0$. Solving $Ax = b$ means minimizing $\frac{1}{2}x'Ax - b'x$
- If we replace $x_s = T^{-1}x$ we get $\frac{1}{2}x_s'T'ATx_s - b'Tx_s \Rightarrow T'ATx_s = T'b$
- Matrix T should be simple to compute and invert, for example diagonal
- **Jacobi scaling** sets $T = \mathrm{diag}(\frac{1}{\sqrt{A_{ii}}})$, so that $T'AT$ has unit diagonal. Usually (but not always) the new condition number gets lower

PRECONDITIONING

- **Example**: Jacobi scaling of random symmetric positive definite matrices with condition number between 1 and 10^8. Ratio $\mathrm{cond}(T'AT)/\mathrm{cond}(A)$



- There are many other techniques for matrix preconditioning and for matrix equilibration (Giselsson, Boyd, 2015)