Emory University

MATH 517 Iterative Methods

Learning Notes

Jiuru Lyu

December 16, 2025

Contents

List of Algorithms

1 Simple Iterations

1.1 Introduction

Suppose we want to solve the linear system $A\mathbf{x} = \mathbf{b}$, where $A \in \mathbb{R}^{n\times n}$ is nonsingular, and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.

- We can use LU factorization. But if A is large, LU factorization takes a long time.
- In such cases, we should consider using an *iterative method*.
- When A is large and sparse, doing matrix vector multiplication is much cheaper than directly factorize A.
- So, in iterative method, we should aim to only do matrix vector multiplications.

1.1.1 Simple Iteration Algorithm

Suppose $M \in \mathbb{R}^{n\times n}$ is a nonsingular matrix. Let $\mathbf{x}_0$ be an initial approximation of $\mathbf{x}$. Suppose $M^{-1}A \approx I$. That is, $M \approx A$ in some sense. Then,

$$\begin{aligned}\mathbf{x} - \mathbf{x}_0 &= A^{-1}\mathbf{b} - \mathbf{x}_0 \\ &= A^{-1}(\mathbf{b} - A\mathbf{x}_0) \approx M^{-1}(\mathbf{b} - A\mathbf{x}_0).\end{aligned}$$

Hence, a better approximation is given by

$$\mathbf{x}_1 = \mathbf{x}_0 + M^{-1}(\mathbf{b} - A\mathbf{x}_0).$$

In general, we can iterate:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k), \quad \text{for } k = 0, 1, 2, \ldots.$$

Algorithm 1: Simple Iteration for $A\mathbf{x} = \mathbf{b}$

Input: Initial guess, $\mathbf{x}_0$

1 **begin**
2 **for** $k = 0, 1, 2, \ldots$ **do**
3 $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$;
4 Solve $M\mathbf{d}_k = \mathbf{r}_k$;
5 $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$;

Remark. We should assume:

- Inverting M, or solving $M\mathbf{d} = \mathbf{r}$ (Line 4), is easier than it would be for A.
- Computing matrix vector multiplications $A\mathbf{x}_k$ are relativelly cheap.

1.1.2 Stopping Criterion

For now, we stop the iteration when

1. $\frac{\|\mathbf{r}_k\|}{\|\mathbf{b}_k\|} \leq \texttt{tol}$.

   For example, $\texttt{tol} = 10^{-8}$.

   We might need to worry about conditioning number, but let's ignore it for now.

2. $k > \texttt{max \# of iteration}$.

   Otherwise, if we set `tol` to small, the algorithm might never stop.

1.1.3 Convergence

Definition 1.1.1 (Spectral Radius). Suppose $A \in \mathbb{R}^{n\times n}$. Then, the *spectral radius* of A, denoted as $\rho(A)$ is defined as

$$\rho(A) = \max\{|\lambda_1|, |\lambda_2|, \ldots, |\lambda_n|\},$$

where $\lambda_i \in \mathbb{C}$ is the eigenvalue of A.

Theorem 1.1.2 Convergence of Simple Iteration
Suppose $A \in \mathbb{R}^{n\times n}$ and $M \in \mathbb{R}^{n\times n}$ are nonsingular, and

$$\rho(I - M^{-1}A) < 1.$$

Then, the simple iteration converges linearly to $\mathbf{x} = A^{-1}\mathbf{b}$.

Proof 1. Let $\mathbf{e}_{k+1} = \mathbf{x} - \mathbf{x}_{k+1}$ be the error at iteration k. [*If converging,* $\mathbf{e}_{k+1} \to 0$.] Note that $\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k)$ is the iteration, we have

$$\begin{aligned}
\mathbf{e}_{k+1} = \mathbf{x} - \mathbf{x}_{k+1} &= \mathbf{x} - \left(\mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k)\right) \\
&= \mathbf{x} - \mathbf{x}_k - M^{-1}(A\mathbf{x} - A\mathbf{x}_k) && [A\mathbf{x} = \mathbf{b}] \\
&= \mathbf{x} - \mathbf{x}_k - M^{-1}A(\mathbf{x} - \mathbf{x}_k) \\
&= \mathbf{e}_k - M^{-1}A\mathbf{e}_k \\
&= (I - M^{-1}A)\mathbf{e}_k.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathbf{e}_{k+1} = (I - M^{-1}A)\mathbf{e}_k &= (I - M^{-1}A)(I - M^{-1}A)\mathbf{e}_{k-1} \\
&= (I - M^{-1}A)^{k+1}\mathbf{e}_0.
\end{aligned}$$

[*This error will go to zero, i.e., the iteration converges, if* $(I - M^1A)^{k+1} \to 0$.]

Now, assume $I - M^{-1}A$ is *diagonalizable*, i.e.,

$$I - M^{-1}A = V\Lambda V^{-1},$$

where $V = \left[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\right]$ is consists of eigenvectors of $I - M^{-1}A$ and $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is formed by the eigenvalues of $I - M^{-1}A$. Then,

$$\begin{aligned}(I - M^{-1}A)^{k+1} &= (V\Lambda V^{-1})^{k+1} \\ &= (V\Lambda V^{-1})(V\Lambda V^{-1})\cdots(V\Lambda V^{-1}) \\ &= V\Lambda^{k+1}V^{-1}.\end{aligned}$$

Since Λ is a diagonal matrix, to have $(I - M^{-1}A)^{k+1} = V\Lambda^{k+1}V^{-1} \to 0$, we want each $\lambda_i^{k+1} \to 0$. To achieve so, we require $|\lambda_i| < 1 \quad \forall\, i = 1, \ldots, n$. That is,

$$\rho(I - M^{-1}A) = \max_{i=1,\ldots,n} \{|\lambda_i|\} < 1.$$

Q.E.D. ■

Remark.

- We can relax the assumption that $I - M^{-1}A$ is diagonalizable. Just take the Jordan form.
- If $\left\|I - M^{-1}A\right\| < 1$, the error decays monotonically:

$$\begin{aligned}\|\mathbf{e}_{k+1}\| &= \left\|(I - M^{-1}A)\mathbf{e}_k\right\| \\ &\le \left\|I - M^{-1}A\right\| \cdot \|\mathbf{e}_k\|.\end{aligned}$$

  This is the *ideal situation.*

- However, $\rho(I - M^{-1}A) < 1$ does not imply $\left\|I - M^{-1}A\right\| < 1$.

Example 1.1.3

Let $A = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}$. This is not a *normal matrix.*

[*A matrix A is a normal matrix if $AA^\top = A^\top A$, or A is orthogonally diagonalizable.*]

Note that $\rho(A) = 0$ (with $\lambda_1 = \lambda_2 = \lambda_3 = 0$). But $\|A\|_1 = \|A\|_\infty = \|A\|_2 = 2 > 1$.

So, it could happen that the error increases in early iterations before decreasing.

- If we have the ideal situation, $\left\|I - M^{-1}A\right\| < 1$, then

$$\|\mathbf{e}_{k+1}\| \le \left\|I - M^{-1}A\right\|^{k+1} \|\mathbf{e}_k\|$$

This gives us an representation of relative error:

$$\frac{\|\mathbf{e}_{k+1}\|}{\|\mathbf{e}_0\|} \leq \left\|I - M^{-1}A\right\|^{k+1}.$$

Example 1.1.4 Number of Iterations

Suppose we want $\frac{\|\mathbf{e}_{k+1}\|}{\|\mathbf{e}_0\|} < \delta$. How many iterations do we need?

That is, we want $\left\|I - M^{-1}A\right\|^{k+1} \leq \delta$. So,

$$\begin{aligned}
\left\|I - M^{-1}A\right\|^{k+1} &\leq \delta \\
(k+1)\log\left\|I - M^{-1}A\right\| &\leq \log\delta \\
k+1 &\geq \frac{\log\delta}{\log\|I - M^{-1}A\|} \\
k &\geq \frac{\log\delta}{\log\|I - M^{-1}A\|} - 1.
\end{aligned}$$

[*Under the ideal situation,* $\left\|I - M^{-1}A\right\| < 1 \implies \log\left\|I - M^{-1}A\right\| < 0$.]

1.2 Stationary Methods

Stationary iterative methods are based on "splitting" A as

$$A = M - N, \quad \text{where } M \text{ is nonsingular.}$$

So, from the system $A\mathbf{x} = \mathbf{b}$, we get

$$\begin{aligned}
A\mathbf{x} = \mathbf{b} &\implies (M-N)\mathbf{x} = \mathbf{b} \\
&\quad M\mathbf{x} - N\mathbf{x} = \mathbf{b} \\
&\quad \mathbf{x} = M^{-1}N\mathbf{x} + M^{-1}\mathbf{b} \qquad \text{(Fixed Point Iteration)}
\end{aligned}$$

Hence, the natural iteration to use is the Fixed Point Iteration:

$$\mathbf{x}_{k+1} = M^{-1}N\mathbf{x}_k + M^{-1}\mathbf{b}.$$

1.2.1 Well-Known Splitting Methods

Let's partition A as:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = D + L + U,$$

where D is the diagonal entries of A, L is the strictly lower triangular part of A, and U is the strictly upper triangular part of A.

- *Jacobi*: Take $M = D$ (diagonal) amd $N = -(L+U)$
- *Gauss-Seidel*: Take $M = D + L$ (lower triangular) and $N = -U$.
- *SOR (Successive OverRelaxation)*: Let ω be the *relaxation parameter*. Then,

$$M = \frac{1}{\omega}D + L \quad \text{and} \quad N = \frac{1-\omega}{\omega}D - U.$$

1. If $\omega = 1$, we are back to Gauss-Seidel.
2. If $\omega > 1$, we have *over relaxation*.
3. If $\omega < 1$, we have *under relaxation*.

Remark. Stationary iterations are simple iterations. Here's a simple proof.

Proof 1.

$$\begin{aligned} \mathbf{x}_{k+1} &= M^{-1}N\mathbf{x}_k + M^{-1}\mathbf{b} \\ &= M^{-1}(M - M + N)\mathbf{x}_k + M^{-1}\mathbf{b} && [\textit{Add and subtract } M] \\ &= \mathbf{x}_k - M^{-1}A\mathbf{x}_k + M^{-1}\mathbf{b} && [A = M - N] \\ &= \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k) && [\textit{Exactly simple iteration}] \end{aligned}$$

Q.E.D. ■

1.2.2 Convergence

Recall that the convergence of simple iteration depends on $I - M^{-1}A$. For stationary iteration methods, it means

$$I - M^{-1}A = I - M^{-1}(M - N) = M^{-1}N.$$

So, we will be looking at $\rho(M^{-1}N)$ or $\left\|M^{-1}N\right\|$.

Theorem 1.2.1 Jacobi Convergence
If A is strictly diagonally dominant by rows or by columns, then the Jacobi iteration converges for every $\mathbf{x}_0$.

Proof 2. For Jacobi, assume $A = D + L + U$ and take $M = D$ and $N = -(L + U)$. Then,

$$\begin{aligned} M^{-1}N &= -D^{-1}(L+U) \\ &= -\begin{bmatrix} 1/a_{11} & & & \\ & 1/a_{22} & & \\ & & \ddots & \\ & & & 1/a_{nn} \end{bmatrix} \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix} \end{aligned}$$

Recall that multiplying a diagonal matrix from the left is equivalent to scaling the row vectors. So,

$$M^{-1}N = -\begin{bmatrix} 0 & a_{12}/a_{11} & \cdots & a_{1n}/a_{11} \\ a_{21}/a_{22} & 0 & \cdots & a_{2n}/a_{22} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}/a_{nn} & a_{n2}/a_{nn} & \cdots & 0 \end{bmatrix}$$

Note that if we take the ∞-norm of $M^{-1}N$, we get

$$\left\|M^{-1}N\right\|_\infty = \max_i \sum_{j\neq i} \left|\frac{a_{ij}}{a_{ii}}\right| = \max_i \frac{1}{|a_{ii}|} \sum_{j\neq i} |a_{ij}|.$$

If $|a_{ii}| > \sum_{j\neq i} |a_{ij}|$ for $i = 1, \ldots, n$, (i.e., A is diagonally dominant by rows), then $\left\|M^{-1}N\right\|_\infty < 1$. Hence, Jacobi converges.

[If we want to get results for diagonally dominant by columns, we need to use $\|\cdot\|_1$ instead.] Q.E.D. ■

Theorem 1.2.2 Gauss-Seidel Convergence
If A is symmetric positive definite (SPD), then the Gauss-Seidel iteration will converge for any $\mathbf{x}_0$.

Proof 3. For Gauss-Seidel, $A = D + L + U$, and take $M = D + L$ and $N = -U$. Suppose A is SPD.

Remark 4. (Properties of SPD). Here, we review some properties of SPD:

- Symmetry: $U = L^\top$.
- Positive-definite: $\mathbf{x}^\top A\mathbf{x} > 0 \quad \forall\, \mathbf{x} \neq 0$.

  Then, $\mathbf{e}_i^\top A\mathbf{e}_i = a_{ii} > 0$. Hence, the diagonal entries of A are positive. That is, $d_{ii} > 0$.

We will use these properties later in the proof.

Let λ be an eigenvalue of $M^{-1}N$ and $\mathbf{v}$ be one of its corresponding eigenvector. Let $\mathbf{v}$ be normallized so that $\mathbf{v}^\top D\mathbf{v} = 1$.

By symmetry

$$M^{-1}N = -(D+L)^{-1}U = -(D+L)^{-1}L^\top.$$

Then,

$$\begin{aligned} M^{-1}N\mathbf{v} = -(D+L)^{-1}L^\top\mathbf{v} &= \lambda\mathbf{v} \\ -L^\top\mathbf{v} &= \lambda(D+L)\mathbf{v} = \lambda(D\mathbf{v}+L\mathbf{v}) \\ -\mathbf{v}^\top L^\top\mathbf{v} &= \lambda(\underbrace{\mathbf{v}^\top D\mathbf{v}}_{=1}+\mathbf{v}^\top L\mathbf{v}) \\ -\mathbf{v}^\top L^\top\mathbf{v} &= \lambda(1+\mathbf{v}^\top L\mathbf{v}). \end{aligned}$$

Since $\mathbf{v}^\top L^\top\mathbf{v}$ is a scalar,

$$\mathbf{v}^\top L^\top\mathbf{v} = \left(\mathbf{v}^\top L^\top\mathbf{v}\right)^\top = \mathbf{v}^\top L\mathbf{v}.$$

Hence,

$$\begin{aligned} -\mathbf{v}^\top L\mathbf{v} &= \lambda\left(1+\mathbf{v}^\top L\mathbf{v}\right) \\ \lambda = -\frac{\mathbf{v}^\top L\mathbf{v}}{1+\mathbf{v}^\top L\mathbf{v}} &\implies \lambda^2 = \frac{(\mathbf{v}^\top L\mathbf{v})^2}{(1+\mathbf{v}^\top L\mathbf{v})^2}. \end{aligned}$$

Since A is SPD, we know

$$\begin{aligned} 0 < \mathbf{v}^\top A\mathbf{v} &= \mathbf{v}^\top\left(D+L+L^\top\right)\mathbf{v} \\ &= \underbrace{\mathbf{v}^\top D\mathbf{v}}_{=1}+\mathbf{v}^\top L\mathbf{v}+\underbrace{\mathbf{v}^\top L^\top\mathbf{v}}_{=\mathbf{v}^\top L\mathbf{v}} \\ &= 1+2\mathbf{v}^\top L\mathbf{v}. \end{aligned}$$

Also,

$$\begin{aligned} \left(1+\mathbf{v}^\top L\mathbf{v}\right)^2 &= \underbrace{1+2\mathbf{v}^\top L\mathbf{v}}_{>0}+\left(\mathbf{v}^\top L\mathbf{v}\right)^2 \\ &> \left(\mathbf{v}^\top L\mathbf{v}\right)^2. \end{aligned}$$

Hence,

$$\lambda^2 = \frac{(\mathbf{v}^\top L\mathbf{v})^2}{(1+\mathbf{v}^\top L\mathbf{v})^2} < 1 \implies |\lambda| < 1.$$

Therefore, $\rho(M^{-1}N) < 1$, leading to convergence of Gauss-Seidel. Q.E.D. ■

Theorem 1.2.3 SOR Convergence
If A is SPD, then SOR converges for any $\mathbf{x}_0$ if and only if $0 < \omega < 2$.

Remark 5. (SOR Convergence).

- The choice of ω can affect convergence speed.
- If $\omega \to \infty$, $M = L$ is an ill-conditioned matrix.

1.2.3 More Stationary Iterative Methods

- SSOR (Symmetric Successive OverRelaxation)
- Regular Splittings
- Block Approaches.

Example 1.2.4 Block Jacobi

Suppose

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mm} \end{bmatrix}, \quad \text{where } A_{ij} \in \mathbb{R}^{p\times p}$$

is a block matrix. Assume the diagonal blocks A_{ii} are non-singular. Then, the block Jacobi will use

$$M = \begin{bmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{mm} \end{bmatrix}.$$

1.3 Iterative Refinement

1.3.1 Motivation

- Iteratively improve qualities of a computed quantity.
- For linear systems, there is a close relationship to the simple iteration.

1.3.2 Solve for $A\mathbf{x} = \mathbf{b}$, where $A \in \mathbb{R}^{n\times n}$ is non-singular

Suppose $\mathbf{x}_0$ is a computed solution of $A\mathbf{x} = \mathbf{b}$. Let $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ be the residual.

- Now, suppose we solve the linear system $A\mathbf{d} = \mathbf{r}_0$ and update

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{d}$$

- If we compute $\mathbf{d}$ exactly, then

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{x}_0 + \mathbf{d} \\ &= \mathbf{x}_0 + A^{-1}\mathbf{r}_0 \\ &= \mathbf{x}_0 + A^{-1}(\mathbf{b} - A\mathbf{x}_0) = A^{-1}\mathbf{b},\end{aligned}$$

  which gives the exact solution of $A\mathbf{x} = \mathbf{b}$ in just one iteration.

- Of course, we cannot solve $A\mathbf{d} = \mathbf{r}_0$ exactly. But, we might hope that $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{d}$ is a better approximation than $\mathbf{x}_0$.

- We can do this iteratively:

Algorithm 2: Naive Iterative Refinement

```
for k = 0, 1, 2, ... do
    r_k = b − A x_k;
    solve A d_k = r_k;
    update x_{k+1} = x_k + d_k;
```

  This looks like simple iteration with $M = A$.

- Question: Our set-up was that solving $A\mathbf{x} = \mathbf{b}$ is hard. But now we are asked to solve $A\mathbf{d}_k = \mathbf{r}_k$?

  Answer: We solve $A\mathbf{d}_k = \mathbf{r}_k$ with a lower precision.

- In practice, iterative refinement for $A\mathbf{x} = \mathbf{b}$ is implemented in multiple precisions. Specifically, consider three precisions:

$$\mu_r \leq \mu \leq \mu_s,$$

  where μ_r is the smallest unit roundoff (highest precision; usually the distance between 1 and the next floating point number), μ is the working precision, and μ_s is the largest unit roundoff (lowest precision).

Example 1.3.1 Three Precisions

- Double: $\mu_r = 1.11 \times 10^{-16}$
- Single: $\mu = 5.96 \times 10^{-8}$
- Half: $\mu_s = 4.88 \times 10^{-4}$

Algorithm 3: Iterative Refinement for $A\mathbf{x} = \mathbf{b}$ Using Three Precisions

Input: initial approximate solution $\mathbf{x}_0$, stored in precision μ

begin

  factorize A (e.g., `[L, U] = lu(A)`);

  for $k = 0, 1, 2, \ldots$ **do**

    compute: $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$ in precision μ_r;

    solve: $A\mathbf{d}_k = \mathbf{r}_k$ in precision μ_s [*can use GEPP, faster than QR;* `d=U\(L\r)`];

    update: $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$ in precision μ;

1.3.3 Relate Iterative Refinement with Simple Iteration

Simplified Assumption: Only errors in the algorithm come from solving $A\mathbf{d}_k = \mathbf{r}_k$.

We will recall from MATH 515/MATH 315 the backward error analysis (where we want to show the computed solution is the exact solution of a near-by problem).

So, in the iterative refinement context, the computed $\mathbf{d}_k$ is the exact solution of a near-by problem. That is,

$$\underbrace{(A + \Delta A_k)}_{M_k}\mathbf{d}_k = \mathbf{r}_k$$

$$\mathbf{d}_k = M_k^{-1}\mathbf{r}_k.$$

Repeating convergence discussion from simple iteration, if $\mathbf{e}_{k+1} = \mathbf{x} - \mathbf{x}_{k+1}$, then

$$\mathbf{e}_{k+1} = \left(I - M_k^{-1}A\right)\mathbf{e}_k$$

Hence, $\mathbf{e}_{k+1} \to 0$ if $\left\|I - M_k^{-1}A\right\| < 1 \quad \forall\, k$.

- We cannot take this convergence analysis any further since we don't know ΔA_k and hence don't know M_k and M_k^{-1}.
- Also, what do we mean by "convergence" is questionable: the exact solution might no be representable in precision μ at all!
- Instead, it makes more sense to try to find bounds on *limiting accuracy* and *limiting residual*:

$$\text{Limiting accuracy} = \frac{\|\widehat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \text{some bound},$$

$$\text{Limiting residual} = \|\mathbf{b} - A\widehat{\mathbf{x}}\| \leq \text{some bound},$$

  where $\widehat{\mathbf{x}}$ is an iterate $\mathbf{x}_k$.
- To identify those bounds, we need rounding error analysis, which is very technical.

1.3.4 Iterative Refinement for Least Squares Problems

Consider the least square problem

$$\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2^2, \tag{LS}$$

where $A \in \mathbb{R}^{m\times n}$ with $m \geq n$, and $\mathrm{rank}(A) = n$ (A is a full rank matrix).

- Naturally, we could apply iterative refinement to the Normal Equations:

$$A^\top A\mathbf{x} = A^\top \mathbf{b}. \tag{Normal Equations}$$

  But, recall that, in the case of $\|\cdot\|_2$, $\kappa_2(A^\top A) = (\kappa_2(A))^2$. Iterative refinement does not work well on ill-conditioned systems.

- An alternative approach is to apply iterative refinement to the following augmented system:

$$\begin{bmatrix} I & A \\ A^\top & 0 \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} \tag{1}$$

  The solution to (1) gives the solution of the least squares problem and its corresponding residual:

$$\mathbf{r} + A\mathbf{x} = \mathbf{b} \implies \mathbf{r} = \mathbf{b} - A\mathbf{x}.$$

$$A^\top \mathbf{r} = \mathbf{0} \implies A^\top(\mathbf{b} - A\mathbf{x}) = 0 \implies A^\top A\mathbf{x} = A^\top \mathbf{b}.$$

1.3.5 Other Linear Algebra Problems with Iterative Refinement

- Find an eigenvalue/eigenvector pair:

$$\text{Solve:} \qquad A\mathbf{x} = \lambda\mathbf{x}.$$

- Find a singular value/vector triplet:

$$\text{Solve:} \qquad A\mathbf{v} = \sigma\mathbf{u} \quad \text{and} \quad A^\top\mathbf{u} = \sigma\mathbf{v}.$$

1.4 Floating Point Numbers

Definition 1.4.1 (Normalized Floating Point Numbers). Usually, we write it as

$$x = \pm\beta^{e+1}\left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_t}{\beta^t}\right) = \pm\beta^{e+1} \times (0.d_1 d_2 \ldots d_t),$$

where

- $0 \leq d_i \leq \beta - 1$ with $d_1 \neq 0$ is called *mantissa.*
- $e_{\min} \leq e \leq e_{\max}$ is called *exponent,*where $e_{\min}$ is a large negative integer, and $e_{\max}$ is a large positive integer.

  [*In IEEE standard,* $e_{\min} = 1 - e_{\max}$.]
- t is called *precision*

Example 1.4.2 $\beta = 10$ **and** $t = 4$

- π is presented as

$$\pi_{\text{fl}} = 10^1 \times 0.3142 = 10^1 \times \left(\frac{3}{10} + \frac{1}{10^2} + \frac{4}{10^3} + \frac{2}{10^4}\right).$$

- Largest representable number is

$$\begin{aligned} x_{\max} &= 10^{e_{\max}+1}\left(\frac{9}{10} + \frac{9}{10^2} + \frac{9}{10^3} + \frac{9}{10^4}\right) \\ &= 10^{e_{\max}+1} \times 0.9999 \end{aligned}$$

  Note that if $m = 0.9999$, then

$$m + 10^{-4} = 0.9999 + 0.0001 = 1 \implies m = 1 - 10^{-4}.$$

  So,

$$x_{\max} = 10^{e_{\max}+1}(1 - 10^{-4}) = 10^{e_{\max}+1}(1 - 10^{-t})$$

- Smallest representable number is

$$\begin{aligned} x_{\min} &= 10^{e_{\min}+1}\left(\frac{1}{10} + \frac{0}{10^2} + \frac{0}{10^3} + \frac{0}{10^4}\right) \\ &= 10^{e_{\min}}. \end{aligned}$$

Example 1.4.3 $\beta = 2$ **and** $t = 4$

- Largest representable number:

$$\begin{aligned} x_{\max} &= 2^{e_{\max}+1}\left(\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}+\frac{1}{2^4}\right) \\ &= 2^{e_{\max}+1}(1-2^{-4}) = 2^{e_{\max}+1}(1-2^{-t}). \end{aligned}$$

- Smallest representable number:

$$\begin{aligned} x_{\min} &= 2^{e_{\min}+1}\left(\frac{1}{2}+\frac{0}{2^2}+\frac{0}{2^3}+\frac{0}{2^4}\right) \\ &= 2^{e_{\min}}. \end{aligned}$$

Example 1.4.4 For any base β **and precision** t

- $$1 = \beta^1(0.100\cdots0) = \beta^1\left(\frac{1}{\beta}+\frac{0}{\beta^2}+\frac{0}{\beta^3}+\cdots+\frac{0}{\beta^t}\right)$$

- The next representable FP number is

$$\begin{aligned} x &= \beta^1(0.100\cdots1) \\ &= \beta^1\left(\frac{1}{\beta}+\frac{0}{\beta^2}+\frac{0}{\beta^3}+\cdots+\frac{1}{\beta^t}\right) \\ &= 1+\beta^{1-t}. \end{aligned}$$

Definition 1.4.5 (Machine Epsilon and Unit Roundoff).

- *Machine epsilon*: $\varepsilon = \beta^{1-t}$ = distance from 1 to the next largest FP.

  So, any number in between 1 and $1+\varepsilon$ is NOT representable. They will be *rounded* to either 1 or $1+\varepsilon$.

- *Unit roundoff*: $\mu = \frac{1}{2}\beta^{1-t} = \frac{1}{2}\varepsilon$ = The largest rounding error when rounding a number in $(1, 1+\varepsilon)$ to the next FP number.

Remark.

- All computers use binary, i.e., $\beta = 2$. That is, FL numbers are represented with $\beta = 2$,

$d_i = 0$ or 1, and $d_1 = 1$. [*This means we know for sure $d_1 = 1$, and thus we don't store it!*]

- Computers store FP numbers with a sequence of bits (0 or 1), allocating some bits for the mantissa, and some for the exponent.
- $(+/-)$ usually takes 1 bit in the mantissa. [*Since $d_1 = 1$ and we are not storing it, we don't lose any bit by storing the sign.*]

Example 1.4.6 Bit Allocations for Commonly Used FP Precision Types

	Precision Type	# Bits for Mantissa	# Bits for Exponent
Google	half, `bfloat16`	8	8
IEEE Standards	half, `fp16`	11	5
	single, `fp32`	24	8
	double, `fp64`	53	11
	quad, `fp128`	113	15

Remark.

- Fewer bits for mantissa $\Longrightarrow$ more roundoff error.
- Fewer bits for exponent $\Longrightarrow$ less dynamic range (more risk for overflow or underflow).
- Fewer bits $\Longrightarrow$ lower storage and faster.

Now, the question is how do we determine $e_{\min}$ and $e_{\max}$.

- There is no sign bit in the exponent.
- Consider `fp32` (or `bfloat16`) for example. We have 8 bits for the exponent. These can represent integers $0 \sim 255$.
- [*Wait... We need negative exponents to represent small numbers. How to achieve this without the sign bit?*]
- To get negative integers, IEEE standards uses half the integers to denote positive exponents and the other half to denote negative exponents by subtracting a *bias*.
- In `fp32`, bias is 127. So,

$$\text{biased exponent} = \text{actual exponent} + 127.$$

	Actual Exponent	Biased Exponent	Binary Representation
Special case	-127	0	00000000
	-126	1	00000001
	-125	2	00000010
	-124	3	00000011
	$\vdots$	$\vdots$	$\vdots$
	0	127	01111111
	$\vdots$	$\vdots$	$\vdots$
	127	254	11111110
Special case	128	255	11111111

Remark.

- Because we assume $d_1 = 1$ and this is not stored, the mantissa alone cannot represent 0.
- [*This is ridiculous! We need* 0 *for sure in our computation.*]
- To represent 0, the IEEE standards define it as

$$\text{mantissa} = \text{all } 0 \text{ bits} \quad \text{and} \quad \text{exponent} = \text{all } 0 \text{ bits}.$$

- Representing ∞, IEEE standards define it as

$$\text{mantissa} = \text{all } 1 \text{ bits} \quad \text{and} \quad \text{exponent} = \text{all } 1 \text{ bits}.$$

- Therefore, for `fp32`, $e_{\max} = 127$ and $e_{\min} = -126$.

Precision	Mantissa	Exponent	μ	$e_{\min}$	$x_{\min}$	$e_{\max}$	$x_{\max}$
`bfloat16`	8	8	3.91×10^{-3}	-126	1.18×10^{-38}	127	3.39×10^{38}
`fp16`	11	5	4.88×10^{-4}	-14	6.1×10^{-5}	15	6.55×10^{4}
`fp32`	24	8	5.96×10^{-8}	-126	1.18×10^{-38}	127	3.40×10^{38}
`fp64`	53	11	1.11×10^{-16}	-1022	2.22×10^{-308}	1023	1.80×10^{308}
`fp128`	113	15	9.63×10^{-35}	-16382	3.36×10^{-4932}	16383	1.19×10^{4932}

Example 1.4.7 Verify

$x = \dfrac{1}{10} = (0.1)_{10}$

Its normalized binary representation:

$$x = 2^{-3} \times (0.110011001100\cdots).$$

2 Krylov Subspace Methods for Eigenvalues

2.1 Introduction to Krylov Subspaces

Definition 2.1.1 (k-th Krylov Subspace). Given $A \in \mathbb{R}^{n\times n}$, $\mathbf{v} \in \mathbb{R}^n$, $\mathbf{v} \neq 0$, the *k-th Krylov subspace* is

$$\mathcal{K}_k(A,\mathbf{v}) = \operatorname{span}\left\{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \ldots, A^{k-1}\mathbf{v}\right\},$$

where span includes all linear combinations.

Definition 2.1.2 (Basis of a Subspace). $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k\}$ is a *basis* if

- $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k$ span the subspace, and
- $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k$ are L.I.

So, a basis is the *minimal spanning set.*

- Therefore, the vectors $\left\{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \ldots, A^{k-1}\mathbf{v}\right\}$ will be a basis for $\mathcal{K}_k(A,\mathbf{v})$ if $\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \ldots, A^{k-1}\mathbf{v}$ are L.I. (they already form a spanning set).

Example 2.1.3

$A = I$, $\mathbf{v} \neq 0$.

- $\mathcal{K}_1(I,\mathbf{v}) = \{\mathbf{v}\} \implies$ basis for $\mathcal{K}_1(I,\mathbf{v})$.
- $\mathcal{K}_2(I,\mathbf{v}) = \{\mathbf{v}, I\mathbf{v}\} = \{\mathbf{v},\mathbf{v}\} \implies$ not a basis.

Example 2.1.4 Largest k

$A \in \mathbb{R}^{n\times n}, \mathbf{v} \neq 0 \in \mathbb{R}^n$. What is the largest k for which $\left\{\mathbf{v}, A\mathbf{v}, \ldots, A^{k-1}\mathbf{v}\right\}$ is L.I.?

Solution 1.

Since $A^j\mathbf{v} \in \mathbb{R}^n$, the most we can have is n L.I. vectors. So, the maximum k is n. □

- Thus, when generating vectors $\left\{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \ldots, A^{k-1}\mathbf{v}\right\}$, a linear dependence must eventually occur.
- Application of Krylov subspaces: Iterative methods for solving eigenvalue and singular value problems, and solving linear systems draw approximations from Krylov subspace.
- Generally, we expand the basis until a dependence occurs.
- But, $\left\{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \ldots, A^{k-1}\mathbf{v}\right\}$ might be an ill-conditioned basis. We will transform these into an orthonormal basis using Gram-Schmidt-like algorithms.

2.2 Arnoldi Method

Given $A = \mathbb{R}^{n\times n}$. Suppose we want to find an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ and an upper Hessenberg matrix H $s.t.$

$$Q^\top AQ = H.$$

That is, we also have [Q *is orthogonal:* $Q^{-1} = Q^\top$]

$$AQ = QH.$$

Let $Q = \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_n \end{bmatrix}$ and consider

$$A\begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_n \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_n & \mathbf{q}_{n+1} \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1n} \\ h_{21} & h_{22} & \cdots & h_{2n} \\ & h_{32} & \cdots & h_{3n} \\ & & \ddots & \vdots \\ & & & h_{nn} \\ & & & h_{n+1,n} \end{bmatrix} \tag{2}$$

- Let $\mathbf{q}_1$ be any vector with $\|\mathbf{q}_1\|_2 = 1$.
- Matching the first column in (2):

$$A\mathbf{q}_1 = h_{11}\mathbf{q}_1 + h_{21}\mathbf{q}_2. \tag{3}$$

[*We know* $\mathbf{q}_1$ *(picked by us). So, we need to find* $\mathbf{q}_2$, h_{11}, *and* h_{21}.]

We want $\mathbf{q}_2$ $s.t.$ $\mathbf{q}_1^\top \mathbf{q}_2 = 0$ and $\|\mathbf{q}_2\|_2 = 1$ since Q is orthonormal. Multiply (3) by $\mathbf{q}_1^\top$, we have

$$\mathbf{q}_1^\top A\mathbf{q}_1 = h_{11}\underbrace{\mathbf{q}_1^\top \mathbf{q}_1}_{=1} + h_{21}\underbrace{\mathbf{q}_1^\top \mathbf{q}_2}_{=0}$$

$$\boxed{h_{11} = \mathbf{q}_1^\top A\mathbf{q}_1}.$$

Now, define

$$h_{21}\mathbf{q}_2 = A\mathbf{q}_1 - h_{11}\mathbf{q}_1 =: \mathbf{w}_1.$$

If $\mathbf{w}_1 \neq 0$, then

$$\|\mathbf{w}_2\|_2 = \|h_{21}\mathbf{q}_2\|_2 = h_{21}\underbrace{\|\mathbf{q}_2\|_2}_{=1} = h_{21}.$$

So,

$$\boxed{h_{21} = \|\mathbf{w}_1\|_2}.$$

Then,

$$\boxed{\mathbf{q}_2 = \frac{\mathbf{w}_1}{h_{21}}}.$$

- Matching the second column in (2):

$$A\mathbf{q}_2 = h_{12}\mathbf{q}_1 + h_{22}\mathbf{q}_2 + h_{32}\mathbf{q}_3 \tag{4}$$

Unknowns: $\mathbf{q}_3, h_{12}, h_{22}, h_{32}$.

Find $\mathbf{q}_3$ $s.t.$ $\mathbf{q}_1^\top \mathbf{q}_3 = 0,\ \mathbf{q}_2^\top \mathbf{q}_3 = 0$, and $\|\mathbf{q}_3\|_2 = 1$.

Multiply (4) by $\mathbf{q}_1^\top$:

$$\mathbf{q}_1^\top A\mathbf{q}_2 = h_{12}\underbrace{\mathbf{q}_1^\top \mathbf{q}_1}_{=1} + h_{22}\underbrace{\mathbf{q}_1^\top \mathbf{q}_2}_{=0} + h_{32}\underbrace{\mathbf{q}_1^\top \mathbf{q}_3}_{=0}$$

$$\boxed{h_{12} = \mathbf{q}_1^\top A\mathbf{q}_2}.$$

Multiply (4) by $\mathbf{q}_2^\top$:

$$\mathbf{q}_2^\top A\mathbf{q}_2 = h_{12}\underbrace{\mathbf{q}_2^\top \mathbf{q}_1}_{=0} + h_{22}\underbrace{\mathbf{q}_2^\top \mathbf{q}_2}_{=1} + h_{32}\underbrace{\mathbf{q}_2^\top \mathbf{q}_3}_{=0}$$

$$\boxed{h_{22} = \mathbf{q}_2^\top A\mathbf{q}_2}.$$

Now, define

$$h_{32}\mathbf{q}_3 = A\mathbf{q}_2 - h_{12}\mathbf{q}_1 - h_{22}\mathbf{q}_2 =: \mathbf{w}_2.$$

If $\mathbf{w}_2 \neq 0$, then $\|\mathbf{w}_2\|_2 = \|h_{32}\mathbf{q}_3\|_2 = h_{32}\underbrace{\|\mathbf{q}_3\|_2}_{=1}$. So,

$$\boxed{h_{32} = \|\mathbf{w}_2\|_2}. \implies \boxed{\mathbf{q}_3 = \frac{\mathbf{w}_2}{h_{32}}}.$$

In general, we have the *Arnoldi Method* (Algorithm 4).

Algorithm 4: Arnoldi Method

Input: A, $\mathbf{q}_1$ with $\|\mathbf{q}_1\|_2 = 1$.

begin
  for $j = 1, 2, \ldots, m$ **do**
    `// This is Gram-Schmidt-like procedure`
    Compute $h_{ij} = \mathbf{q}_i^\top A\mathbf{q}_j$ for $i = 1, \ldots, j$;
    Compute $\mathbf{w}_j = A\mathbf{q}_j - \sum_{i=1}^{j} h_{ij}\mathbf{q}_i$;
    Compute $h_{j+1,j} = \|\mathbf{w}_j\|_2$;
    if $h_{j+1,j} = 0$ **then**
      `// We get linear dependence`
      $m = j$;
      break;
    Compute $\mathbf{q}_{j+1} = \mathbf{w}_j / h_{j+1,j}$;

Remark.

- If $\{\mathbf{v}, A\mathbf{v}, \ldots, A^m\mathbf{v}\}$ are L.I., then the above algorithm is just classical Gram-Schmidt to produce an orthonormal set $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{m+1}\}$, with

$$\operatorname{span}\{\mathbf{v}, A\mathbf{v}, \ldots, A^m\mathbf{v}\} = \operatorname{span}\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{m+1}\}.$$

- We know for better stability, we should use modified Gram-Schmidt (Algorithm 5).

Algorithm 5: Arnoldi (Modified Gram-Schmidt Approach)

Input: $A \in \mathbb{R}^{n\times n}$, $\mathbf{q}_1 \in \mathbb{R}^n$ with $\|q_1\|_2 = 1$.

begin
  for $j = 1, \ldots, n$ **do**
    $\mathbf{w}_j = A\mathbf{q}_j$ `// Most time-consuming part`
    for $i = 1, \ldots, j$ **do**
      $h_{ij} = \mathbf{q}_i^\top \mathbf{w}_j$;
      $\mathbf{w}_j = \mathbf{w}_j - h_{ij}\mathbf{q}_i$;
    $\omega = \|\mathbf{w}_j\|_2$;
    if $\omega = 0$ **then**
      $m = j$;
      break;
    $h_{j+1,j} = \omega$;
    $\mathbf{q}_{j+1} = \mathbf{w}_j / h_{j+1,j}$;

Output: $AQ = QH$, where $Q \in \mathbb{R}^{n\times m}$, $Q^\top Q = I \in \mathbb{R}^{m\times m}$, $n \geq m$, and H is upper Hessenberg.

2.2.1 Matrix Relations in Arnoldi

Definition 2.2.1 (Ritz Values and Vectors). Eigenvalues and eigenvectors of H_k are called the *Ritz values and vectors of* A.

- If $(\lambda, \mathbf{v})$ is an eigenpair of H_k, then $(\lambda, Q_k\mathbf{v})$ is an approximate eigenpair of A.
- From Important Relations to Eigenpair:

  From the derivation using matched columns,

$$AQ_k = Q_{k+1}\widetilde{H}_k,$$

  where

$$Q_k = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \ldots & \mathbf{q}_k\end{bmatrix},$$

and

$$\widetilde{H}_k = \left[\begin{array}{cccc} h_{11} & h_{12} & \cdots & h_{1k} \\ h_{21} & h_{22} & \cdots & h_{2k} \\ & \ddots & \ddots & \vdots \\ & & \ddots & h_{kk} \\ \hline & & & h_{j+1,k} \end{array}\right] = \left[\begin{array}{c} H_k \\ \hline h_{k+1,k}\mathbf{e}_k^\top \end{array}\right], \quad \text{where } \mathbf{e}_k^\top = \begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix}.$$

That is,

$$\begin{aligned} AQ_k &= Q_{k+1}\left[\begin{array}{c} H_k \\ \hline h_{k+1,k}\mathbf{e}_k^\top \end{array}\right] \\ &= \begin{bmatrix} Q_k & | & \mathbf{q}_{k+1} \end{bmatrix}\left[\begin{array}{c} H_k \\ \hline h_{k+1,k}\mathbf{e}_k^\top \end{array}\right] \\ \boxed{AQ_k = Q_k H_k + h_{k+1,k}\mathbf{q}_{k+1}\mathbf{e}_k^\top}. & \qquad \text{(Important Relation I)} \end{aligned}$$

Note, $h_{k+1,k}\mathbf{q}_{k+1}\mathbf{e}_k^\top$ above is a rank-1 matrix.

Also note

$$Q_k^\top AQ_k = \underbrace{Q_k^\top Q_k}_{=I} H_k + h_{k+1,k} \underbrace{Q_k^\top \mathbf{q}_{k+1}}_{=0,\text{ orthogonality}} \mathbf{e}_k^\top$$

$$\boxed{Q_k^\top AQ_k = H_k}. \qquad \text{(Important Relation II)}$$

Then, from the two relations, we get

$$\begin{aligned} AQ_k\mathbf{v} &= \left(Q_k H_k + h_{k+1,k}\mathbf{q}_{k+1}\mathbf{e}_k^\top\right)\mathbf{v} && [\textit{by (Important Relation I)}] \\ &= Q_k H_k\mathbf{v} + h_{k+1,k}\mathbf{v}_k\mathbf{q}_{k+1} && [\mathbf{e}_k^\top\mathbf{v} = \mathbf{v}_k] \\ &= \lambda Q_k\mathbf{v} + h_{k+1,k}\mathbf{v}_k\mathbf{q}_{k+1} && [\mathbf{v}\textit{ is eigenvector of } H_k] \\ (A-\lambda I)Q_k\mathbf{v} &= h_{k+1,k}\mathbf{v}_k\mathbf{q}_{k+1} && [\textit{move terms}] \\ \|(A-\lambda I)Q_k\mathbf{v}\|_2 &= \|h_{k+1,k}\mathbf{v}_k\mathbf{q}_{k+1}\|_2 && [\textit{take 2-norm}] \\ &= |h_{k+1,k}|\cdot\|\mathbf{v}_k\|_2 && [\|\mathbf{q}_{k+1}\|_2 = 1] \end{aligned}$$

If $h_{k+1,k} = 0$, them,

$$\begin{aligned} &\|(A-\lambda I)Q_k\mathbf{v}\|_2 = 0 \\ &\quad (A-\lambda I)Q_k\mathbf{v} = 0 \implies (\lambda, Q_k\mathbf{v}) \text{ is an eigenpair of } A. \end{aligned}$$

- So, if the Arnoldi algorithm stops, we know we get an eigenpair of A.
- This is the basic approach used by `ARPACK` (e.g., `eig` in `MATLAB`) for estimating a few eigenvalues/vectors for large sparse matrices.

Remark 1. (Clarifying Remarks).

- In Arnoldi, we use $\mathbf{q}_1$ to be any vector with $\|\mathbf{q}_1\|_2 = 1$.
- These are constructed so that

$$\text{span}\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k\} = \text{span}\left\{\mathbf{q}_1, A\mathbf{q}_1, \ldots, A^{k-1}\mathbf{q}_1\right\}.$$

- If we use $\mathbf{q}_1 = \mathbf{v}/\|\mathbf{v}\|_2$, then

$$\begin{aligned}\text{span}\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k\} &= \text{span}\left\{\mathbf{q}_1, A\mathbf{q}_1, \ldots, A^{k-1}\mathbf{q}_1\right\} \\ &= \text{span}\left\{\frac{\mathbf{v}}{\|\mathbf{v}\|_2}, A\frac{\mathbf{v}}{\|\mathbf{v}\|_2}, \ldots, A^{k-1}\frac{\mathbf{v}}{\|\mathbf{v}\|_2}\right\} \\ &= \text{span}\left\{\mathbf{v}, A\mathbf{v}, \ldots, A^{k-1}\mathbf{v}\right\} \\ &= \mathcal{K}_k(A, \mathbf{v}).\end{aligned}$$

2.3 Lanczos Method

- If we apply the Arnoldi method to a symmetric matrix, we get the Lanczos method.
- If A is symmetric, then $A^\top = A$. Also,

$$H_k^\top = \left(Q_k^\top A Q_k\right)^\top = Q_k^\top A^\top Q_k = Q_k^\top A Q_k = H_k.$$

So, H_k is also symmetric,

But H_k is upper Hessenberg, so it has the form *tridiagonal*.

$$H_k = \begin{bmatrix} h_{11} & h_{21} & & & \\ h_{21} & h_{22} & h_{32} & & \\ & h_{32} & h_{33} & & \\ & & \ddots & \ddots & h_{k,k-1} \\ & & & h_{k,k-1} & h_{k,k} \end{bmatrix} \longrightarrow \text{symmetric tridiagonal}$$

- Notation: In this case, we change notation: Arnoldi$\longleftrightarrow$Lanczos: $H_k \longleftrightarrow T_k$, and

$$\widetilde{T}_k = \left[\begin{array}{cccc} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \beta_2 & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \\ \hline & & & \beta_k \end{array}\right] = \left[\begin{array}{c} T_k \\ \hline \beta_k \mathbf{e}_k^\top \end{array}\right]$$

- We have similar matrix relations:
  1. $AQ_k = Q_k T_k + \beta_k \mathbf{q}_{k+1} \mathbf{e}_k^\top$, and
  2. $Q_k^\top A Q_k = T_k$.

Algorithm 6: Lanczos (Symmetric A)

Input: $\mathbf{q}_1$, with $\|\mathbf{q}_1\|_2 = 1$; symmetric A

```
begin
    Set β_0 = 0 and q_0 = 0;
    for k = 1, ..., n do
        w = A q_k;
        α_k = q_k^⊤ w;
        w = w − β_{k−1} q_{k−1} − α_k q_k // three-term recurrence
        β_k = ‖w‖_2;
        if β_k = 0 then
            m = k;
            break;
        q_{k+1} = w/β_k;
```

Remark.

- To generate symmetric matrices:
  1. $\frac{1}{2}(A + A^\top)$,
  2. $A^\top A$, and
  3. $\begin{bmatrix} 0 & A^\top \\ A & 0 \end{bmatrix}$
- Algorithm 6 computes $T_m \in \mathbb{R}^{m \times m}$, tridiagonal, and $Q_m \in \mathbb{R}^{n \times m}$, with orthonormal columns $s.t.\ AQ_m = Q_m T_m$.
- The vectors $\mathbf{q}_i$ are *supposed* to be orthogonal.
- But, as with classical Gram-Schmidt, round-off errors cause loss of orthogonality.
- Remedy: *reorthogonalize* (Algorithm 7).

Algorithm 7: Lanczos Algorithm with Full Reorthogonalization

Input: symmetric $A \in \mathbb{R}^{n\times n}$, $\mathbf{q}_1 \in \mathbb{R}^n$ with $\|\mathbf{q}_1\|_2 = 1$.

begin
  Set $\beta_0 = 0$;
  for $j = 1, \dots, n$ **do**
    $\mathbf{w} = A\mathbf{q}_j$;
    $\alpha_j = \mathbf{q}_j^\top \mathbf{w}$;
    $\mathbf{w} = \mathbf{w} - \beta_{j-1}\mathbf{q}_{j-1} - \alpha_j \mathbf{q}_j$;
    `// modified GS style`
    for $k = 1, \dots, j-1$ **do**
      $\mathbf{w} = \mathbf{w} - (\mathbf{q}_k^\top \mathbf{w})\mathbf{q}_k$;
    $\beta_j = \|\mathbf{w}\|_2$;
    if $\beta_j = 0$ **then**
      Set $m = j$;
      Break;
    $\mathbf{q}_{j+1} = \mathbf{w}/\beta_j$;

Remark.

- Reorthogonalization requires additional work. The work grows as we continue going.

We can use *Partial Reorthogonalization*: replace the loop with

```
for k = 1, ..., s do
    w = w − (q_k^⊤ w) q_k;
```

where we need to choose a small s such as

$$s = \min\{j-1, 10\}.$$

- Later, when we use Arnoldi or Lanczos to solve $A\mathbf{x} = \mathbf{b}$, reorthogoanlization is often not needed.

2.4 Golub-Kahan Bidiagonalization

GKB is very similar to Arnoldi and Lancaoz, but it can be used on general $A \in \mathbb{R}^{m\times n}$.

Suppose we want to find U, V with orthonormal columns and B an upper bidiagonal matrix $s.t.$

$$U^\top AV = B \quad \text{or} \quad V^\top A^\top U = B^\top.$$

[*This is the start of an SVD.*] That is, $AV = UB$ or $A^\top U = VB^\top$.

- Now, let's match columns of $AV = UB$.

$$A\begin{bmatrix}\mathbf{v}_1 & \cdots & \mathbf{v}_j & \cdots\end{bmatrix} = \begin{bmatrix}\mathbf{u}_1 & \cdots \mathbf{u}_{j-1} & \mathbf{u}_j & \cdots\end{bmatrix}\begin{bmatrix}\alpha_1 & \beta_1 & & & & \\ & \alpha_2 & \beta_2 & & & \\ & & \ddots & \ddots & & \\ & & & \ddots & \beta_{j-1} & \\ & & & & \alpha_j & \ddots \\ & & & & & \ddots\end{bmatrix}$$

$$A\mathbf{v}_j = \beta_{j-1}\mathbf{u}_{j-1} + \alpha_j\mathbf{u}_j$$
$$\alpha_j\mathbf{u}_j = A\mathbf{v}_j - \beta_{j-1}\mathbf{u}_{j-1} \qquad [\textit{Assume } \beta_0 = 0.]$$

Note that because we want $\|\mathbf{u}_j\|_2 = 1$, we have

$$\boxed{\alpha_j = \|A\mathbf{v}_j - \beta_{j-1}\mathbf{u}_{j-1}\|_2}$$
$$\boxed{\mathbf{u}_j = (A\mathbf{v}_j - \beta_{j-1}\mathbf{u}_{j-1})/\alpha_j}$$

- Similarly, match columns of $A^\top U = VB\top$.

$$A^\top\begin{bmatrix}\mathbf{u}_1 & \cdots & \mathbf{u}_j & \cdots\end{bmatrix} = \begin{bmatrix}\mathbf{v}_1 & \cdots & \mathbf{v}_j & \mathbf{v}_{j+1} & \cdots\end{bmatrix}\begin{bmatrix}\alpha_1 & & & & \\ \beta_1 & \alpha_2 & & & \\ & \beta_2 & \ddots & & \\ & & \ddots & \alpha_j & \\ & & & \beta_j & \ddots \\ & & & & \ddots\end{bmatrix}$$

$$A^\top\mathbf{u}_j = \alpha_j\mathbf{v}_j + \beta_j\mathbf{v}_{j+1}$$
$$\beta_j\mathbf{v}_{j+1} = A^\top\mathbf{u}_j - \alpha_j\mathbf{v}_j.$$

Because we want $\|\mathbf{v}_{j+1}\|_2 = 1$, we have

$$\boxed{\beta_j = \left\|A^\top\mathbf{u}_j - \alpha_j\mathbf{v}_j\right\|_2}$$
$$\boxed{\mathbf{v}_{j+1} = \left(A^\top\mathbf{i}_j - \alpha_j\mathbf{v}_j\right)/\beta_j}.$$

Algorithm 8: Golub-Kahan Bidiagonalization (GKB)

Input: $A \in \mathbb{R}^{m\times n}$, $\mathbf{v}_1 \in \mathbb{R}^n$ with $\|\mathbf{v}_1\|_2 = 1$

begin
  Set $\beta_0 = 0$;
  for $j = 1, \ldots, n$ **do**
    $\mathbf{u}_j = A\mathbf{v}_j - \beta_{j-1}\mathbf{u}_{j-1}$;
    $\alpha_j = \|\mathbf{u}_j\|_2$;
    if $\alpha_j = 0$ **then**
      `// Linear dependency in Krylov subspace generation`
      Set $m = j$;
      Break;
    $\mathbf{u}_j = \mathbf{u}_j/\alpha_j$;
    $\mathbf{v}_{j+1} = A^\top \mathbf{u}_j - \alpha_j \mathbf{v}_j$;
    $\beta_j = \|\mathbf{v}_{j+1}\|_2$;
    if $\beta_j = 0$ **then**
      `// Linear dependency`
      Set $m = j$;
      Break;
    $\mathbf{v}_{j+1} = \mathbf{v}_{j+1}/\beta_j$;

But... What is the Krylov subspace we are generating?

- $\mathcal{K}_k(A, \mathbf{v}) = \{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \ldots, A^{k-1}\mathbf{v}\}$. However, if $A \in \mathbb{R}^{m\times n}$ with $m \neq n$, A^2 doesn't make sense due to dimension mismatch.
- Similar to Arnoldi and Lanczos, GKB compute bases for Krylov subspaces:
  1. $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ is basis for
  $$\mathcal{K}_k\left(A^\top A, \mathbf{v}_1\right) = \text{span}\left\{\mathbf{v}_1, (A^\top A)\mathbf{v}_1, (A^\top A)^2\mathbf{v}_1, \ldots, (A^\top A)^{k-1}\mathbf{v}_1\right\}$$
  2. $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k$ is basis for
  $$\mathcal{K}_k\left(AA^\top, \mathbf{u}_1\right) = \text{span}\left\{\mathbf{u}_1, (AA^\top)\mathbf{u}_1, (AA^\top)^2\mathbf{u}_1, \ldots, (AA^\top)^{k-1}\mathbf{u}_1\right\}$$

Remark. As with Arnoldi and Lanczos, the vectors $\mathbf{u}_j$ and $\mathbf{v}_j$ can lose orthogonality. So, we need to do reorthogonalization (Algorithm 9).

Algorithm 9: Golub-Kahan Bidiagonalization with Full Reorthogonalization

Input: $A \in \mathbb{R}^{m\times n}$, $\mathbf{v}_1 \in \mathbb{R}^n$ with $\|\mathbf{v}_1\|_2 = 1$

begin
  Set $\beta_0 = 0$;
  for $j = 1, \ldots, n$ **do**
    $\mathbf{u}_j = A\mathbf{v}_j - \beta_{j-1}\mathbf{u}_{j-1}$;
    for $k = 1, \ldots, j-1$ **do**
      $\mathbf{u}_j = \mathbf{u}_j - (\mathbf{u}_k^\top \mathbf{u}_j)\mathbf{u}_k$;
    $\alpha_j = \|\mathbf{u}_j\|_2$;
    if $\alpha_j = 0$ **then**
      Set $m = j$;
      Break;
    $\mathbf{u}_j = \mathbf{u}_j/\alpha_j$;
    $\mathbf{v}_{j+1} = A^\top \mathbf{u}_j - \alpha_j \mathbf{v}_j$;
    for $k = 1, \ldots, j$ **do**
      $\mathbf{v}_{j+1} = \mathbf{v}_{j+1} - (\mathbf{v}_k^\top \mathbf{v}_{j+1})\mathbf{v}_k$;
    $\beta_j = \|\mathbf{v}_{j+1}\|_2$;
    if $\beta_j = 0$ **then**
      Set $m = j$;
      Break;
    $\mathbf{v}_{j+1} = \mathbf{v}_{j+1}/\beta_j$;

Remark. To verify that our Algorithm produces a basis for $\mathcal{K}_k(A, \mathbf{v})$ in general, build

$$K = \begin{bmatrix} \mathbf{v} & A\mathbf{v} & A^2\mathbf{v} & \cdots & A^{k-1}\mathbf{v} \end{bmatrix}.$$

Then,

$$\begin{bmatrix} Q & R \end{bmatrix} = \mathtt{qr}(K).$$

Suppose $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$ is the output from our algorithm. Then build

$$V = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_k \end{bmatrix}.$$

Check if $K^\top V = I$ (up to signs).

2.5 Hessenberg Method

In this section, we will do something similar to Arnoldi for $A \in \mathbb{R}^{n\times n}$, but we do not enforce orthogonality of Krylov basis vectors.

Specifically, we build

$$\begin{aligned} AL_k &= L_{k+1}\widetilde{H}_k \\ &= \begin{bmatrix} L_k & \boldsymbol{\ell}_{k+1} \end{bmatrix} \begin{bmatrix} H_k \\ h_{k+1}\mathbf{e}_k^\top \end{bmatrix}, \end{aligned}$$

where H_k is upper Hessenberg, and L_k is a nonsingular, unit lower triangular matrix:

$$L_k = \begin{bmatrix} 1 & & & & & \\ \ell_{21} & 1 & & & & \\ \ell_{31} & \ell_{32} & 1 & & & \\ \ell_{41} & \ell_{42} & \ell_{43} & \ddots & & \\ \vdots & \vdots & \vdots & & \ddots & \\ \ell_{k1} & \ell_{k2} & \ell_{k3} & \cdots & \cdots & 1 \\ \ell_{k+1,1} & \ell_{k+1,2} & \ell_{k+1,3} & \cdots & \cdots & \ell_{k+1,k} \\ \vdots & \vdots & \vdots & & & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & \cdots & \ell_{nk} \end{bmatrix}$$

To build L_k and H_k, let's match columns:

$$A\begin{bmatrix} \boldsymbol{\ell}_1 & \boldsymbol{\ell}_2 & \cdots & \boldsymbol{\ell}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{\ell}_1 & \boldsymbol{\ell}_2 & \cdots & \boldsymbol{\ell}_k & \boldsymbol{\ell}_{k+1} \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1k} \\ h_{21} & h_{22} & \cdots & \vdots \\ & h_{32} & \cdots & \vdots \\ & & \ddots & \vdots \\ & & & h_{kk} \\ & & & h_{k+1,k} \end{bmatrix}$$

- Let $\mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$ be any vector, with $v_1 \neq 0$. Then,

$$\boldsymbol{\ell}_1 = \mathbf{v}/v_1.$$

- Match first columns:

$$A\boldsymbol{\ell}_1 = h_{11}\boldsymbol{\ell}_1 + h_{21}\boldsymbol{\ell}_2.$$

Here, we know $\boldsymbol{\ell}_1$ and aim to find h_{11}, h_{21}, and $\boldsymbol{\ell}_2$. Denote $\mathbf{w} \coloneqq A\boldsymbol{\ell}_1$. Then,

$$\mathbf{w} = h_{11}\boldsymbol{\ell}_1 + h_{21}\boldsymbol{\ell}_2$$

That is,

$$\begin{bmatrix} w_1 \\ \vdots \\ \vdots \\ w_n \end{bmatrix} = h_{11} \begin{bmatrix} 1 \\ \ell_{21} \\ \vdots \\ \ell_{n1} \end{bmatrix} + h_{21} \begin{bmatrix} 0 \\ 1 \\ \vdots \\ \ell_{n2} \end{bmatrix}$$
$$\implies \boxed{h_{11} = w_1}.$$

Override $\mathbf{w} \coloneqq \mathbf{w} - h_{11}\boldsymbol{\ell}_1$. Then,

$$\mathbf{w} = h_{21}\mathbf{e}_2$$
$$\begin{bmatrix} w_1 & \\ w_2 & \\ \vdots & \vdots \\ w_n & \end{bmatrix} = h_{21} \begin{bmatrix} 0 \\ 1 \\ \ell_{32} \\ \vdots \\ \ell_{n2} \end{bmatrix}$$
$$\implies \boxed{h_{21} = w_2}$$
$$\boxed{\boldsymbol{\ell}_2 = \mathbf{w}/h_{21}}.$$

- Match second columns:

$$A\boldsymbol{\ell}_2 = h_{12}\boldsymbol{\ell}_1 + h_{22}\boldsymbol{\ell}_2 + h_{32}\boldsymbol{\ell}_3.$$

Denote $\mathbf{w} \coloneqq A\boldsymbol{\ell}_2$. Then,

$$\mathbf{w} = h_{12}\boldsymbol{\ell}_1 + h_{22}\boldsymbol{\ell}_2 + h_{32}\boldsymbol{\ell}_3$$
$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = h_{12} \begin{bmatrix} 1 \\ \ell_{21} \\ l_{31} \\ \vdots \\ l_{n1} \end{bmatrix} + h_{22} \begin{bmatrix} 0 \\ 1 \\ \ell_{32} \\ \vdots \\ l_{n2} \end{bmatrix} + h_{32} \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ \ell_{n3} \end{bmatrix}$$
$$\implies \boxed{h_{12} = w_1}.$$

Override $\mathbf{w} \coloneqq \mathbf{w} - h_{12}\boldsymbol{\ell}_1$. Then,

$$\mathbf{w} = h_{22}\boldsymbol{\ell}_2 + h_{32}\boldsymbol{\ell}_3$$

That is,

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = h_{22} \begin{bmatrix} 0 \\ 1 \\ \ell_{32} \\ \vdots \\ \ell_{n2} \end{bmatrix} + h_{32} \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ \ell_{n3} \end{bmatrix}$$

$$\implies \boxed{h_{22} = w_2}$$

Override $\mathbf{w} \coloneqq \mathbf{w} - h_{22}\boldsymbol{\ell}_2$. Then,

$$\mathbf{w} = h_{32}\boldsymbol{\ell}_3$$

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = h_{32} \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ \ell_{n3} \end{bmatrix}$$

$$\implies \boxed{h_{32} = w_3}$$

$$\boxed{\boldsymbol{\ell}_3 = \mathbf{w}/h_{32}}$$

Algorithm 10: Hessenberg Algorithm

Input: $A \in \mathbb{R}^{n\times n}$, $\mathbf{v} \in \mathbb{R}^n$ with $v_1 \neq 0$

```
begin
    // column-oriented forward substitution style
    Set ℓ_1 = v/v_1;
    for j = 1, ... do
        w = Aℓ_j;
        for i = 1, 2, ..., j do
            h_ij = w_j;
            w = w − h_ij ℓ_i;
        h_{j+1,j} = w_{j+1};
        ℓ_{j+1} = w/h_{j+1,j};
```

Remark.

(+) No inner product involved in Algorithm 10. Inner products are bottleneck for parallel computing because they are hard to parallelize.

(-) What if $v_1 = 0$? We have to do pivoting so that $v_1 \neq 0$.

(-) What if $h_{j+1,j} = 0$?

1. Try to pivot $\mathbf{w}$ so that $w_{j+1} \neq 0$.
2. If we cannot find non-zero entries, then stop. We reach linear dependency.

2.5.1 Relationship between Arnoldi and Hessenberg

Given $A \in \mathbb{R}^{n\times n}$ and $\mathbf{v} \in \mathbb{R}^n$:

- Arnoldi Matrix Relations: Let $\mathbf{q}_1 = \mathbf{v}/\|\mathbf{v}\|_2$, and compute

$$\begin{aligned} AQ_k &= Q_{k+1}\widetilde{H}_k^a \\ &= \begin{bmatrix} Q_k & \mathbf{q}_{k+1} \end{bmatrix} \widetilde{H}_k^a. \end{aligned}$$

- Hessenberg Matrix Relations: Let $\boldsymbol{\ell}_1 = \mathbf{v}/v_1$, and compute

$$\begin{aligned} AL_k &= L_{k+1}\widetilde{H}_k^h \\ &= \begin{bmatrix} L_k & \boldsymbol{\ell}_{k+1} \end{bmatrix} \widetilde{H}_k^h. \end{aligned}$$

Define

$$K_k = \begin{bmatrix} \mathbf{v} & A\mathbf{v} & \cdots & A^{k-1}\mathbf{v} \end{bmatrix}.$$

Then,

$$\begin{aligned} K_{k+1} &= \begin{bmatrix} \mathbf{v} & A\mathbf{v} & \cdots & A^{k-1}\mathbf{v} & A^k\mathbf{v} \end{bmatrix} \\ &= \begin{bmatrix} K_k & A^k\mathbf{v} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{v} & AK_k \end{bmatrix}. \end{aligned}$$

- Arnold Matrix Relations: Suppose we have QR factorizations:

$$K_k = Q_k\widetilde{R}_k, \quad K_{k+1} = Q_{k+1}\widetilde{R}_{k+1}.$$

Consider

$$\begin{aligned} AK_k &= \begin{bmatrix} \mathbf{v} & AK_k \end{bmatrix} \begin{bmatrix} \mathbf{0}^\top \\ I_k \end{bmatrix} \\ &= K_{k+1} \begin{bmatrix} \mathbf{0}^\top \\ I_k \end{bmatrix} \\ AQ_k\widetilde{R}_k &= Q_{k+1}\widetilde{R}_{k+1} \begin{bmatrix} \mathbf{0}^\top \\ I_k \end{bmatrix} \end{aligned}$$

$$AQ_k = Q_{k+1}\underbrace{\widetilde{R}_{k+1}\begin{bmatrix}\mathbf{0}^\top \\ I_k\end{bmatrix}\widetilde{R}_k^{-1}}_{=\widetilde{H}_k^a}$$
$$AQ_k = Q_{k+1}\widetilde{H}_k^a.$$

- Hessenberg Matrix Relations: Suppose we have LU factorizations:

$$K_k = L_kU_k, \quad K_{k+1} = L_{k+1}U_{k+1}.$$

Consider

$$\begin{aligned}
AK_k &= \begin{bmatrix}\mathbf{v} & AK_k\end{bmatrix}\begin{bmatrix}\mathbf{0}^\top \\ I_k\end{bmatrix} \\
&= K_{k+1}\begin{bmatrix}\mathbf{0}^\top \\ I_k\end{bmatrix} \\
AL_kU_k &= L_{k+1}U_{k+1}\begin{bmatrix}\mathbf{0}^\top \\ I_k\end{bmatrix} \\
AL_k &= L_{k+1}\underbrace{U_{k+1}\begin{bmatrix}\mathbf{0}^\top \\ I_k\end{bmatrix}U_k^{-1}}_{=\widetilde{H}_k^h} \\
AL_k &= L_{k+1}\widetilde{H}_k^h.
\end{aligned}$$

So, we have $K_k = L_kU_k = Q_k\widetilde{R}_k$. Then, we can find

$$\begin{aligned}
K_k = L_kU_k &= Q_k\widetilde{R}_k \\
L_k &= Q_k \underbrace{\widetilde{R}_kU_k^{-1}}_{\text{upper } \triangle \text{ matrix}} \\
L_k &= Q_kR_k, \quad \text{where } R_k = \widetilde{R}_kU_k^{-1} \\
&\quad \text{is the thin QR factorization of } L_k \\
Q_k &= L_kR_k^{-1}.
\end{aligned}$$

Furthermore,

$$\begin{aligned}
AQ_k = AL_kR_k^{-1} &= L_{k+1}\widetilde{H}_k^hR_k^{-1} && [AL_k = L_{k+1}\widetilde{H}_k^h] \\
&= Q_{k+1}R_{k+1}\widetilde{H}_k^hR_k^{-1} && [L_{k+1} = Q_{k+1}R_{k+1}] \\
Q_{k+1}\widetilde{H}_k^a &= Q_{k+1}R_{k+1}\widetilde{H}_k^hR_k^{-1} && [AQ_k = Q_{k+1}\widetilde{H}_k^a] \\
\boxed{\widetilde{H}_k^a = R_{k+1}\widetilde{H}_k^hR_k^{-1}}
\end{aligned}$$

2.5.2 Hessenberg Version of GKB

In the case of rectangular $A \in \mathbb{R}^{m\times n}$, we can do something similar to GKB.

Compute basis $\boldsymbol{\ell}_j, \mathbf{j}_j$ $s.t.$

$$AL_k = D_{k+1}\widetilde{H}_k \quad \text{and} \quad A^\top D_{k+1} = L_{k+1}W_{k+1},$$

where $\widetilde{H}_k$ is an upper Hessenberg $(k+1)\times k$ matrix, W_{k+1} is an upper triangular $(k+1)\times(k+1)$ matrix, and D_{k+1}, L_{k+1} are unit lower triangular matrices.

Remark. Basically, we apply Hessenberg method to $AA^\top$ and $A^\top A$.

3 Iterative Methods for $A\mathrm{x} = \mathrm{b}$ and LS Problems

3.1 Krylove Subspace Iteration

Consider solving $A\mathbf{x} = \mathbf{b}$, where $A \in \mathbb{R}^{n\times n}$ is nonsingular. The basic approach is to

1. $\mathbf{x}_0$ is an initial guess (could be $\mathbf{x}_0 = \mathbf{0}$)
2. update $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{z}_k$,

where $\mathbf{z}_k$ is a vector in an appropriate Krylov subspace $\mathcal{K}_k(A, \mathbf{v})$ (usually of increasing dimension).

Example 3.1.1 Naive Update

Suppose $\mathbf{x}_0$ is given. Then, the updates will look like

$$\begin{aligned}
\mathbf{x}_1 &= \mathbf{x}_0 + \mathbf{z}_1, \quad \mathbf{z}_1 \in \mathcal{K}_1(A, \mathbf{v}) = \operatorname{span}\{\mathbf{v}\}.\\
\mathbf{x}_2 &= \mathbf{x}_0 + \mathbf{z}_2, \quad \mathbf{z}_2 \in \mathcal{K}_2(A, \mathbf{v}) = \operatorname{span}\{\mathbf{v}, A\mathbf{v}\}.\\
\mathbf{x}_3 &= \mathbf{x}_0 + \mathbf{z}_3, \quad \mathbf{z}_3 \in \mathcal{K}_3(A, \mathbf{v}) = \operatorname{span}\{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}\}.\\
&\vdots
\end{aligned}$$

3.1.1 What should we choose for v in Krylov subspace?

Typically, we use

$$\mathbf{v} = \mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 \qquad \text{(Residual)}$$

3.1.2 How to represent z_k?

If we have a basis for $\mathcal{K}_k(A, \mathbf{r}_0)$ (e.g., $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k\}$), then $\mathbf{z}_k = \mathcal{K}_k(A, \mathbf{r}_0) \implies$ we can write $\mathbf{z}_k$ as a linear combination of $\mathbf{q}_i$. That is, $\exists$ scalars $y_1, y_2, \ldots, y_k$ $s.t.$

$$\mathbf{z}_k = y_1\mathbf{q}_1 + y_2\mathbf{q}_2 + \cdots + y_k\mathbf{q}_k.$$

Equivalently, $\exists$ a vector $\mathbf{y}_k$ $s.t.$

$$\mathbf{z}_k = Q_k\mathbf{y}_k.$$

3.1.3 What optimality condition should we use to get a good z_k?

Minimum Residual Approach Find

$$\mathbf{z}_k = \arg\min_{\mathbf{z}\in\mathcal{K}_k(A,\mathbf{r}_0)} \|\mathbf{b} - A\mathbf{x}\|_2^2, \quad \mathbf{x} = \mathbf{x}_0 + \mathbf{z}.$$

Notice that

$$\begin{aligned}\|\mathbf{b} - A\mathbf{x}\|_2 &= \|\mathbf{b} - A(\mathbf{x}_0 + \mathbf{z})\|_2 \\ &= \|\underbrace{\mathbf{b} - A\mathbf{x}_0}_{=\mathbf{r}_0} - A\mathbf{z}\|_2 \\ &= \|\mathbf{r}_0 - A\mathbf{z}\|_2\end{aligned}$$

If we restrict $\mathbf{z} \in \mathcal{K}_k(A, \mathbf{r}_0)$, then

$$\|\mathbf{b} - A\mathbf{x}\|_2 = \|\mathbf{r}_0 - AQ_k\mathbf{y}\|_2, \quad \mathbf{z} = Q_k\mathbf{y}.$$

The Basic Idea of Minimum residual Approach

1. Find basis Q_k for $\mathcal{K}_k(A, \mathbf{r}_0)$
2. Compute $\mathbf{y}_k = \arg\min_{\mathbf{y}} \|\mathbf{r}_0 - AQ_k\mathbf{y}\|_2$
3. Update $\mathbf{x}_k = \mathbf{x}_0 + \underbrace{Q_k\mathbf{y}_k}_{\mathbf{z}_k}$

How we solve (2) depends on the basis Q_k (e.g., Arnoldi, Lanczos, etc.).

Minimum A-Norm Error Approach If A is SPD, we can try to find $\mathbf{z} = Q_k\mathbf{y}$ to minimize

$$\|\mathbf{e}_k\|_A^2 = \|\mathbf{x} - \mathbf{x}_k\|_A^2,$$

where $\mathbf{x}$ is the true solution.

Definition 3.1.2 (A-Norm).

$$\|\mathbf{e}_k\|_A^2 = \mathbf{e}_k^\top A\mathbf{e}_k.$$

Note: $\mathbf{e}_k$ is the error, not the unit vector.

We want to find $\mathbf{z} \in \mathcal{K}_k(A, \mathbf{r}_0)$ to minimize

$$\begin{aligned}\|\mathbf{e}_k\|_A^2 &= \|\mathbf{x} - \mathbf{x}_k\|_A^2 \\ &= \|\mathbf{x} - (\mathbf{x}_0 + \mathbf{z}_k)\|_A^2\end{aligned}$$

But if $\mathbf{z} \in \mathcal{K}_k(A, \mathbf{r}_0)$, $\mathbf{z} = Q_k\mathbf{y}$, where $\mathbf{y} \in \mathbb{R}^k$. Therefore, an equivalent problem is to find $\mathbf{y} \in \mathbb{R}^k$ to minimize

$$\|\mathbf{x} - (\mathbf{x}_0 + Q_k\mathbf{y})\|_A^2 = \|\mathbf{e}_0 - Q_k\mathbf{y}\|_A^2, \quad \text{where } \mathbf{e}_0 = \mathbf{x} - \mathbf{x}_0.$$

Note that

$$\underbrace{\mathbf{e}_0^\top A(Q_k\mathbf{y})}_{\text{scalar}} = \Big(\mathbf{e}_0^\top A(A_k\mathbf{y})\Big)^\top = (Q_k\mathbf{y})^\top A^\top \mathbf{e}_0 = (Q_k\mathbf{y})^\top A\mathbf{e}_0 \qquad [\textit{Since } A \textit{ is SPD}]$$

and

$$\mathbf{e}_0^\top A(Q_k\mathbf{y}) = \Big(\mathbf{e}_0^\top AQ_k\Big)\mathbf{y} = \Big(Q_k^\top A\mathbf{e}_0\Big)^\top \mathbf{y}.$$

We will use these two relationships later in the derivation.

Let $\mathbf{e}(\mathbf{y}) = \mathbf{e}_0 - Q_k\mathbf{y}$ be a function of $\mathbf{y}$. Consider

$$\begin{aligned}
f(\mathbf{y}) &= \|\mathbf{e}(\mathbf{y})\|_A^2 \\
&= \|\mathbf{e}_0 - Q_k\mathbf{y}\|_A^2 \\
&= (\mathbf{e}_0 - Q_k\mathbf{y})^\top A(\mathbf{e}_0 - Q_k\mathbf{y}) \\
&= \mathbf{e}_0^\top A\mathbf{e}_0 - (Q_k\mathbf{y})^\top A\mathbf{e}_0 - \mathbf{e}_0^\top A(Q_k\mathbf{y}) + (Q_k\mathbf{y})^\top A(Q_k\mathbf{y}) \\
&= \mathbf{e}_0^\top A\mathbf{e}_0 - 2\Big(Q_k^\top A\mathbf{e}_0\Big)^\top \mathbf{y} + \mathbf{y}^\top Q_k^\top AQ_k\mathbf{y} \qquad [\textit{by relationships above}] \\
\nabla f(\mathbf{y}) &= -2Q_k^\top A\mathbf{e}_0 + 2Q_k^\top AQ_k\mathbf{y}.
\end{aligned}$$

Set $\nabla f(\mathbf{y}) = 0$:

$$\begin{aligned}
-2Q_k^\top A\mathbf{e}_0 + 2Q_k^\top AQ_k\mathbf{y} &= \mathbf{0} \\
-Q_k^\top A\mathbf{e}_0 + Q_k^\top AQ_k\mathbf{y} &= \mathbf{0} \\
Q_k^\top A(\mathbf{e}_0 - Q_k\mathbf{y}) &= \mathbf{0} \\
\implies Q_k^\top A\mathbf{e}(\mathbf{y}) &= \mathbf{0}.
\end{aligned}$$

Then, we let $\mathbf{y}_k$ be the vector for which $Q_k^\top A\mathbf{e}(\mathbf{y}_k) = 0$.

Problem Note that

$$\begin{aligned}
\mathbf{e}(\mathbf{y}_k) &= \mathbf{e}_0 - Q_k\mathbf{y}_k \\
&= \mathbf{x} - \mathbf{x}_0 - Q_k\mathbf{y}_k
\end{aligned}$$

✘ But... we don't know $\mathbf{x}$.

Resolution Do we really need to know $\mathbf{x}$?

$$\begin{aligned}
A\mathbf{e}(\mathbf{y}_k) = A(\mathbf{x} - \mathbf{x}_0 - Q_k\mathbf{y}_k) &= A\mathbf{x} - A\mathbf{x}_0 - AQ_k\mathbf{y}_k \\
&= \mathbf{b} - A\mathbf{x}_0 - AQ_k\mathbf{y}_k && [A\mathbf{x} = \mathbf{b}] \\
&= \mathbf{r}_0 - AQ_k\mathbf{y}_k && [\mathbf{b} - A\mathbf{x}_0 = \mathbf{r}_0]
\end{aligned}$$

✔ So, we don't need to know $\mathbf{x}$!!

Therefore, we want to find $\mathbf{y}_k$ $s.t.$

$$Q_k^\top(\mathbf{r}_0 - AQ_k\mathbf{y}_k) = 0 \implies \underbrace{Q_k^\top AQ_k}_{\substack{A \text{ is SPD} \Rightarrow \text{Lanczos} \\ \text{tridiagonal}}} \quad \mathbf{y}_k = Q_k^\top \mathbf{r}_0$$

Let's take a closer look at the RHS. Suppose $Q_k = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_k\end{bmatrix}$. Then, by Lanczos, we know $\mathbf{q}_1 = \dfrac{\mathbf{r}_0}{\|\mathbf{r}_0\|}$. Therefore, $\mathbf{q}_1^\top \mathbf{r}_0$ is a scalar. However,

$$\mathbf{q}_2^\top \mathbf{r}_0 = 0 \quad \text{since } \mathbf{q}_2 \perp \mathbf{q}_1 = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|}.$$

Summary of Krylov Subspace Iterative Methods

To use iterative methods to solve $A\mathbf{x} = \mathbf{b}$, with $A \in \mathbb{R}^{n\times n}$, we construct Krylov subspaces.

1. Let $\mathbf{x}_0$ be an initial guess, and compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$.
2. Construct a basis for the Krylov subspace $\mathcal{K}_k(A, \mathbf{r}_0)$, $Q_k = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_k\end{bmatrix}$.
3. Compute $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{z}$, where $\mathbf{z} \in \mathcal{K}_k(A, \mathbf{r}_0)$ (i.e., $\mathbf{z} = Q_k\mathbf{y}$, where $\mathbf{y} \in \mathbb{R}^k$) and $\mathbf{z}$ satisfies an optimality condition:
   - Minimize the residual:
   $$\|\mathbf{b} - A(\mathbf{x}_0 + \mathbf{z})\|_2^2 = \|\mathbf{r}_0 - A\mathbf{z}\|_2^2 = \|\mathbf{r}_0 - AQ_k\mathbf{y}\|_2^2 \qquad \mathbf{z} \in \mathcal{K}_k(A, \mathbf{r}_0) \text{ and } \mathbf{y} \in \mathbb{R}^k.$$
   - Or, if A is SPD, compute $\mathbf{z} \in \mathcal{K}_k(A, \mathbf{r}_0)$ to minimize the A-norm of the error:
   $$\|\mathbf{x} - (\mathbf{x}_0 + \mathbf{z})\|_A^2 = \|\mathbf{e}_0 - \mathbf{z}\|_A^2 = \|\mathbf{e}_0 - Q_k\mathbf{y}\|_A^2 \qquad \mathbf{z} \in \mathcal{K}_k(A, \mathbf{r}_0) \text{ and } \mathbf{y} \in \mathbb{R}^k,$$
   where $\|\mathbf{e}_0 - Q_k\mathbf{y}\|_A^2 = (\mathbf{e}_0 - Q_k\mathbf{y})^\top A(\mathbf{e}_0 - Q_k\mathbf{y})$.
   This is equivalent to compute $\mathbf{y} \in \mathbb{R}^k$ $s.t.$
   $$Q_k^\top(\mathbf{r}_0 - AQ_k\mathbf{y}) = 0.$$

3.1.4 Different Iterative Methods

We get different iterative methods using different bases for the Krylov subspaces and different optimization criteria.

Suppose $A \in \mathbb{R}^{n\times n}$, and we use Arnoldi and minimal residual

From Arnoldi, we have $AQ_k = Q_{k+1}\widetilde{H}_k$. We need to consider

$$\|\mathbf{r}_0 - AQ_k\mathbf{y}\|_2^2 = \left\|\mathbf{r}_0 - Q_{k+1}\widetilde{H}_k\mathbf{y}\right\|_2^2$$

In Arnoldi, we use

$$\mathbf{q}_1 = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|_2} \implies \mathbf{r}_0 = \|\mathbf{r}_0\|_2 \mathbf{q}_1 = \beta \mathbf{q}_1, \quad \text{where } \beta = \|\mathbf{r}_0\|_2.$$

So,

$$\|\mathbf{r}_0 - AQ_k\mathbf{y}\|_2^2 = \left\|\beta\mathbf{q}_1 - Q_{k+1}\widetilde{H}_k\mathbf{y}\right\|_2^2$$

Note that $Q_{k+1}\mathbf{e}_1 = \mathbf{q}_1$, we have $\beta\mathbf{q}_1 = Q_{k+1}(\beta\mathbf{e}_1)$. So,

$$\begin{aligned}
\|\mathbf{r}_0 - AQ_k\mathbf{y}\|_2^2 &= \left\|Q_{k+1}(\beta\mathbf{e}_1) - Q_{k+1}\widetilde{H}_k\mathbf{y}\right\|_2^2 \\
&= \left\|Q_{k+1}\left(\beta\mathbf{e}_1 - \widetilde{H}_k\mathbf{y}\right)\right\|_2^2 \\
&= \left\|\beta\mathbf{e}_1 - \widetilde{H}_k\mathbf{y}\right\|_2^2 \qquad \textit{[This is a } (k+1)\times k \textit{ LS problem.]}
\end{aligned}$$

[The final equality holds because Q_{k+1} has orthonormal columns, so it is $2-$norm invariant under orthogonal transformation.]

This leads to the basic *Generalized Minimum Residual Method (GMRES)*. Algorithm 11 outlines the algorithm.

Algorithm 11: Basic Generalized Minimum Residual (GMRES)

Input: $A \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$, and $\mathbf{x}_0 \in \mathbb{R}^n$

begin
  Set $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|_2$, and $\mathbf{q}_1 = \mathbf{r}_0/\beta$;
  for $j = 1, \ldots, m$ **do**
    `// Arnoldi Method`
    $\mathbf{w}_j = A\mathbf{q}_j$;
    for $i = 1, \ldots, j$ **do**
      $h_{ij} = q_i^\top \mathbf{w}_j$;
      $\mathbf{w}_j = \mathbf{w}_j - h_{ij}\mathbf{q}_i$;
    $h_{j+1,j} = \|\mathbf{w}_j\|_w$;
    if $h_{j+1,j} = 0$ **then**
      Set $m = j$;
      Break;
    $\mathbf{q}_{j+1} = \mathbf{w}_j / h_{j+1,j}$;
  $\mathbf{y}_m = \arg\min_{\mathbf{y}} \left\|\beta\mathbf{e}_1 - \widetilde{H}_m\mathbf{y}\right\|_2$ `// Solving the LS problem`
  $\mathbf{x}_m = \mathbf{x}_0 + Q_m\mathbf{y}_m$;

Remark. However, Algorithm 11 is not practical... To make it practical, we need

- Solve the LS for $\mathbf{y}_m$ more efficiently, and

- Better test for convergence than $h_{j+1,j} = 0$.

We will be talking about the practical considerations in Section 3.1.5 and outline the practical GMRES algorithm in Algorithm 12.

Suppose $A \in \mathbb{R}^{n\times n}$, with $A = A^\top$, and we use Lanczos and minimal residual

Similar to Arnoldi, except instead of an upper Hessenberg, we got a tridiagonal matrix. From Lanczos iteration, we get $AQ_k = Q_{k+1}\widetilde{T}_k$. So,

$$\|\mathbf{r}_0 - AQ_k\mathbf{y}\| = \left\|\beta\mathbf{e}_1 - \widetilde{T}_k\mathbf{y}\right\|_2^2 \qquad [\textit{This is a } (k+1)\times k \textit{ LS problem}]$$

This gives the basic *Minimum Residual Method (MINRES)*.

Again, more needs to be done to make this a practical algorithm.

Suppose $A \in \mathbb{R}^{m\times n}$, and we use GKB and minimal residual

Using the initial vector $\mathbf{u}_1 = \mathbf{r}_0/\|\mathbf{r}_0\|_2$, then since $AV_k = U_{k+1}B_k$ from GKB iterations, we have

$$\begin{aligned}
\|\mathbf{r}_0 - AV_k\mathbf{y}\|_2^2 &= \|\mathbf{r}_0 - U_{k+1}B_k\mathbf{y}\|_2^2 && [V_k\mathbf{y} \in \mathcal{K}_k(A^\top A, \mathbf{v}_1)]\\
&= \|U_{k+1}(\beta\mathbf{e}_1 - B_k\mathbf{y})\|_2^2 \\
&= \underbrace{\|\beta\mathbf{e}_1 - B_k\mathbf{y}\|_2^2}_{(k+1)\times k \text{ LS Problem}} && [U_{k+1} \textit{ has orthonormal columns}]
\end{aligned}$$

This gives the basic *LSQR Algorithm*. Again, more needs to be done to make this a practical algorithm.

Suppose $A \in \mathbb{R}^{n\times n}$, and we use Hessenberg and minimal residual

If we use Hessenberg with $\boldsymbol{\ell}_1 = \mathbf{r}_0/\beta$, where $\beta = |\mathbf{r}_0(1)|$, the first entry of $\mathbf{r}_0$ (or, $\beta = \|\mathbf{r}_0\|_\infty$ and then permute, so the maximum entry is on the top). Then,

$$\begin{aligned}
\|\mathbf{r}_0 - AL_k\mathbf{y}\|_2^2 &= \left\|\mathbf{r}_0 - L_{k+1}\widetilde{H}_k\mathbf{y}\right\|_2^2 && [\textit{From Hessenberg: } AL_k = L_{k+1}\widetilde{H}_k.]\\
&= \left\|L_{k+1}\left(\beta\mathbf{e}_1 - \widetilde{H}_k\mathbf{y}\right)\right\|_2^2.
\end{aligned}$$

But here, L_{k+1} does not have orthonormal columns, so

$$\left\|L_{k+1}\left(\beta\mathbf{e}_1 - \widetilde{H}_k\mathbf{y}\right)\right\|_2^2 \neq \left\|\beta\mathbf{e}_1 - \widetilde{H}_k\mathbf{y}\right\|_2^2$$

in general. However, we can still proceed like GMRES: minimizing the "quasi-residual:"

$$\left\|\beta\mathbf{e}_1 - \widetilde{H}_k\mathbf{y}\right\|_2^2, \quad \text{and } \mathbf{x}_k = \mathbf{x}_0 + L_k\mathbf{y}.$$

This leads to the *Changing Minimum Residual Hessenberg Method (CMRH)*, which is part of the *Quasi-Minimum Residual Methods (QMR)* family.

Suppose $A \in \mathbb{R}^{n\times n}$, with A SPD and we use Lanczos and minimal A-norm error

Here, we need to find $\mathbf{y}$ satisfying

$$\begin{aligned} & Q_k^\top(\mathbf{r}_0 - AQ_k\mathbf{y}) = 0 \\ \implies \text{solve} \quad & \underbrace{Q_k^\top AQ_k}_{=T_k}\mathbf{y} = Q_k^\top \mathbf{r}_0 && [\textit{From Lanczos: } Q_k^\top AQ_k = T_k] \\ & T_k\mathbf{y} = \beta\mathbf{e}_1 && [Q_k^\top \mathbf{r}_0 = \beta\mathbf{e}_1] \end{aligned}$$

This gives the basic *Conjugate Gradient Method (CG)*. Again, more needs to be done to get a practical implementation.

Remark. We derived LSQR using GKB and the minimal residual criterion. This is equivalent to using Lanczos on the SPD system $A^\top A\mathbf{x} = A^\top \mathbf{b}$, with the minimal $A^\top A$-norm error criterion.

3.1.5 Practical Considerations of GMRES

Now, we focus on two practical details:

- Need to efficiently solve the LS problem to compute $\mathbf{y}_m$, and
- Need to compute an estimate of $\|\mathbf{r}_j\|_2$ at each iteration, to help determine stopping iteration.

Consider the LS problem at iteration $k-1$:

$$\min_{\mathbf{y}} \left\| \mathbf{g} - \widetilde{H}\mathbf{y} \right\|_2, \quad \text{where } \mathbf{g} = \beta\mathbf{e}_1$$

and

$$\widetilde{H} = \begin{bmatrix} h_{11} & \cdots & h_{1,k-1} \\ h_{21} & \cdots & \vdots \\ & \ddots & \vdots \\ & & h_{k-1,k-1} \\ & & h_{k,k-1} \end{bmatrix}.$$

Let $\widetilde{H} = \widetilde{Q}\widetilde{R}$ be the QR factorization of $\widetilde{H}$, where

$$\widetilde{Q}^\top\widetilde{Q} = I, \quad \text{and} \quad \widetilde{R} = \begin{bmatrix} r_{11} & \cdots & r_{1,k-1} \\ 0 & \cdots & \vdots \\ & \ddots & r_{k-1,k-1} \\ & & 0 \end{bmatrix}.$$

To solve the LS problem:

$$\begin{aligned}
\left\|\mathbf{g} - \widetilde{H}\mathbf{y}\right\|_2 &= \left\|Q^\top \mathbf{g} - \widetilde{Q}^\top \widetilde{H}\mathbf{q}\right\|_2 \\
&= \left\|\widehat{\mathbf{g}} - \widetilde{R}\mathbf{y}\right\|_2 && [\widehat{\mathbf{g}} = Q^\top \mathbf{g}] \\
&= \left\| \begin{bmatrix} \widehat{\mathbf{g}}(1:k-1) \\ \widehat{g}_k \end{bmatrix} - \begin{bmatrix} \widetilde{R}(1:k-1, 1:k-1) \\ \mathbf{0}^\top \end{bmatrix} \mathbf{y} \right\|_2 \\
&= \left\| \begin{bmatrix} \widehat{\mathbf{g}}(1:k-1) - \widetilde{R}(1:k-1, 1:k-1)\mathbf{y} \\ \widehat{g}_k \end{bmatrix} \right\|_2
\end{aligned}$$

This norm will be minimized if

$$\widehat{\mathbf{g}}(1:k-1) - \widetilde{R}(1:k-1, 1:k-1)\mathbf{y} = \mathbf{0}$$

This is equivalent to solve $\widetilde{R}(1:k-1, 1:k-1)\mathbf{y} = \widehat{\mathbf{g}}(1:k-1)$ by backward substitution (since $\widetilde{R}$ is an upper triangular matrix). In addition, the norm of the corresponding residual is $|\widehat{g}_k|$.

Note: If all we need is the norm of the residual, then we *do not* need to compute $\mathbf{y}$.

Now, at iteration k, out LS problem becomes

$$\min_{\mathbf{y}} \left\| \begin{bmatrix} \mathbf{g} \\ 0 \end{bmatrix} - \begin{bmatrix} \widetilde{H} & \mathbf{h} \\ \mathbf{0}^\top & h_{k+1,k} \end{bmatrix} \mathbf{y} \right\|_2^2.$$

Assume we know the QR factorization $\widetilde{H} = \widetilde{Q}\widetilde{R}$. Since $\widetilde{Q}$ is an orthogonal matrix, so is

$$\begin{bmatrix} \widetilde{Q} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix}.$$

Therefore,

$$\begin{aligned}
\left\| \begin{bmatrix} \mathbf{g} \\ 0 \end{bmatrix} - \begin{bmatrix} \widetilde{H} & \mathbf{h} \\ \mathbf{0}^\top & h_{k+1,k} \end{bmatrix} \mathbf{y} \right\|_2^2 &= \left\| \begin{bmatrix} \widetilde{Q}^\top & \mathbf{0}^\top \\ \mathbf{0} & 1 \end{bmatrix} \left(\begin{bmatrix} \mathbf{g} \\ 0 \end{bmatrix} - \begin{bmatrix} \widetilde{H} & \mathbf{h} \\ \mathbf{0}^\top & h_{k+1,k} \end{bmatrix} \mathbf{y} \right) \right\|_2^2 \\
&= \left\| \begin{bmatrix} \widehat{\mathbf{g}} \\ 0 \end{bmatrix} - \begin{bmatrix} \widetilde{Q}^\top \widetilde{H} & \widehat{\mathbf{h}} \\ 0 & h_{k+1,k} \end{bmatrix} \mathbf{y} \right\|_2^2 && [\widehat{\mathbf{h}} = \widetilde{Q}^\top \mathbf{h} \text{ and } \widehat{\mathbf{g}} = \widetilde{Q}^\top \mathbf{g}] \\
&= \left\| \begin{bmatrix} \widehat{\mathbf{g}} \\ 0 \end{bmatrix} - \begin{bmatrix} \widetilde{R} & \widehat{\mathbf{h}} \\ 0 & h_{k+1,k} \end{bmatrix} \mathbf{y} \right\|_2^2 && [\widetilde{H} = \widetilde{Q}\widetilde{R}]
\end{aligned}$$

Note, the matrix $\begin{bmatrix} \widetilde{R} & \widehat{\mathbf{h}} \\ 0 & h_{k+1,k} \end{bmatrix}$ takes a special form, and using 1 Givens rotation, we can easily turn it

into an upper triangular form:

$$\left[\begin{array}{ccc|c} * & \cdots & * & * \\ & \ddots & \vdots & \vdots \\ & & * & * \\ 0 & \cdots & 0 & * \\ \hline 0 & \cdots & 0 & * \end{array}\right]$$

That is, we find scalars c_k, s_k *s.t.* $c_k^2 + s_k^2 = 1$ and

$$\begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} \widehat{h}_k \\ h_{k+1,k} \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}.$$

Then, (mathematically) we get

$$\widetilde{Q}_k^\top = \left[\begin{array}{c|cc} I & 0 & \\ \hline 0 & c_k & s_k \\ & -s_k & c_k \end{array}\right].$$

Q_k is an orthogonal matrix, so at the k-th iteration, we can minimize

$$\underbrace{\left\| Q_k^\top \begin{bmatrix} \widehat{\mathbf{g}} \\ 0 \end{bmatrix} - Q_k^\top \begin{bmatrix} \widetilde{R} & \widehat{\mathbf{h}} \\ \mathbf{0}^\top & h_{k+1,k} \end{bmatrix} \mathbf{y} \right\|_2}_{\text{update and overwrite}} = \left\| \begin{bmatrix} \widehat{\mathbf{g}}' \\ \widehat{g}_{k+1} \end{bmatrix} - \begin{bmatrix} \widetilde{R}' \\ \mathbf{0}^\top \end{bmatrix} \mathbf{y} \right\|_2.$$

where $Q_k^\top \widehat{\mathbf{g}} = \widehat{\mathbf{g}}'$.

Now, we can again solve the LS problem to get $\mathbf{y}$, and the norm of the residual is just $|\widehat{g}_{k+1}|$.

Summary of Practical Implementation

- Let

$$\mathbf{g} = \begin{bmatrix} \beta \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \text{and} \quad \widetilde{H} = \begin{bmatrix} h_{11} & \cdots & h_{1,k-1} \\ h_{21} & \cdots & \vdots \\ & \ddots & h_{k-1,k-1} \\ & & h_{k,k-1} \end{bmatrix}.$$

  Also, $\widetilde{H} = \widetilde{Q}\widetilde{R}$ is the QR factorization.

- Suppose $\mathbf{g}$ and $\widetilde{H}$ have been overwritten with

$$\mathbf{g} \leftarrow \widetilde{Q}^\top \mathbf{g}, \quad \widetilde{H} \leftarrow \widetilde{Q}^\top \widetilde{H}.$$

  That is, $\widetilde{H}$ is now upper triangular.

- At the next iteration, we get a new column:

$$\begin{bmatrix} h_{1,k} \\ \vdots \\ h_{k,k} \\ h_{k+1,k} \end{bmatrix} = \begin{bmatrix} \mathbf{h} \\ h_{k+1,k} \end{bmatrix}.$$

- Overwrite

$$\mathbf{h} \leftarrow \widetilde{Q}^\top \mathbf{h}.$$

- Find c_k, s_k $s.t.$

$$\begin{bmatrix} c_k & k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} h_{kk} \\ h_{k+1,k} \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$$

- Overwrite

$$\begin{aligned} h_{kk} &\leftarrow c_k h_{kk} + s_k h_{k+1,k} \\ h_{k+1,k} &\leftarrow 0 \\ \begin{bmatrix} g_k \\ g_{k+1} \end{bmatrix} &\leftarrow \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} g_k \\ 0 \end{bmatrix} = \begin{bmatrix} c_k g_k \\ -s_k g_k \end{bmatrix}. \end{aligned}$$

- Compute the norm of residual:

$$\rho = |g_{k+1}|.$$

- If we want the solution, solve $H(1:k, 1:k)\mathbf{y} = \mathbf{g}(1:k)$ by back substitution, and set

$$\mathbf{x} = \mathbf{x}_0 + Q_k \mathbf{y}.$$

Remark.

- The orthogonal matrix $\widetilde{Q}^\top$ is a product of Givens rotations:

$$\widetilde{Q}^\top = \widetilde{Q}_{k-1}^\top \cdots \widetilde{Q}_2^\top \widetilde{Q}_1^\top,$$

where

$$\widetilde{Q}_1^\top = \begin{bmatrix} c_1 & s_1 & 0 \\ -s_1 & c_1 & \\ 0 & & I \end{bmatrix}, \quad \widetilde{Q}_2^\top \begin{bmatrix} 1 & & \\ & c_2 & s_2 \\ & -s_2 & c_2 \\ & & & I \end{bmatrix}, \dots$$

To apply $\widetilde{Q}^\top$, all we need is to store

$$\mathbf{c} = \begin{bmatrix} c_1 & c_2 & \cdots & c_{k-1} \end{bmatrix}^\top \quad \text{and} \quad \mathbf{s} = \begin{bmatrix} s_1 & s_2 & \cdots & s_{k-1} \end{bmatrix}^\top.$$

That is, to overwrite $\mathbf{h} \leftarrow \widetilde{Q}^\top \mathbf{h}$, we use

$$\begin{bmatrix} h_{i,k} \\ h_{i+1,k} \end{bmatrix} \leftarrow \begin{bmatrix} c_i & s_i \\ -s_i & c_i \end{bmatrix} \begin{bmatrix} h_{i,k} \\ h_{i+1,k} \end{bmatrix} \quad \text{for } i = 1, 2, \ldots, k-1.$$

- In GMRES, we only need $\mathbf{y}$ if we want to compute the solution

$$\mathbf{x}_k = \mathbf{x}_0 + Q_k \mathbf{y}_k.$$

But we don't need $\mathbf{x}_k$ to compute the residual norm

$$\|\mathbf{r}_k\|_2 = \|\mathbf{b} - A\mathbf{x}_k\|_2.$$

Instead, this comes for free: $\rho = |g_{k+1}|$.

Algorithm 12: Practical GMRES

begin
  Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|_2$, $\mathbf{q}_1 = \mathbf{r}_0/\beta$, and $Q_1 = \mathbf{q}_1$;
  Initialize $\mathbf{g} = \begin{bmatrix} \beta & 0 & \cdots & 0 \end{bmatrix}^\top$;
  for $k = 1, 2, \ldots$ **do**
    while $\rho >$ `tol` **do**
      Compute $\mathbf{q}_{k+1}$, $h_{i,k}$ for $i = 1, 2, \ldots, k+1$, using Arnoldi. `// most expensive operation: matrix-vector multiplication`
      Set $Q = \begin{bmatrix} Q & \mathbf{q}_{k+1} \end{bmatrix}$;
      for $i = 1, 2, \ldots, k-1$ **do**
        $\begin{bmatrix} h_{i,k} \\ h_{i+1,k} \end{bmatrix} = \begin{bmatrix} c_i & s_i \\ -s_i & c_i \end{bmatrix} \begin{bmatrix} h_{i,k} \\ h_{i+1,k} \end{bmatrix}$;
      Compute c_k, s_k *s.t.* $\begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} h_{k,k} \\ h_{k+1,k} \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$;
      Update $h_{k,k} = c_k h_{k,k} + s_k h_{k+1,k}$ and $h_{k+1,k} = 0$;
      Update $\begin{bmatrix} g_k \\ g_{k+1} \end{bmatrix} = \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} g_k \\ 0 \end{bmatrix}$;
      Set $\rho = |g_{k+1}|$;
  Solve upper triangular system $H(1:k, 1:k)\mathbf{y} = \mathbf{g}(1:k)$ using back substitution;
  Update $\mathbf{x} = \mathbf{x}_0 + Q\mathbf{y}$ `// Recall: this` Q `comes from Arnoldi`

Remark. Full practical GMRES can be expensive if many iterations are needed.

3.1.6 Cost in Practical GMRES

1. Computational cost:
   - $\mathcal{O}(nk)$ `flops` needed per iteration + matrix-vector multiplication (depending on the matrix).
   - $\mathcal{O}(k^2)$ for triangular solve.
   - $\mathcal{O}(nk)$ to compute $\mathbf{x}$.
2. Storage:
   - What ever we need to store A.
   - $\mathcal{O}(nk)$ for Q.

   That is, computational cost and storage increase at each iteration.

   $\Longrightarrow$ Improvement: GMRES(j) (Restarted GMRES)
   - Stops GMRES at j-th iteration and restarts with the latest iterate as an initial guess.
   - But it can delay convergence.

3.1.7 Lanczos and MINRES

Instead of $\widetilde{H}$, we have $\widetilde{T}$ tridiagonal.

We can exploit to reduce work per iteration and storage (as we don't need to save all $\mathbf{q}_i$'s).

3.2 Quadratic Functions and CG

3.2.1 Steepest Descent Through Optimization Methods

Theorem 3.2.1

If A is SPD, then the following problems are equivalent:

- Solve $A\mathbf{x} = \mathbf{b}$, and
- Minimize $\varphi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{x}^\top\mathbf{b}$.

Proof 1. ($\Rightarrow$): Suppose $\mathbf{x}$ solves $A\mathbf{x} = \mathbf{b}$, and consider

$$\varphi(\mathbf{y}) = \frac{1}{2}\mathbf{y}^\top A\mathbf{y} - \mathbf{y}^\top\mathbf{b}.$$

[*WTS:* $\varphi(\mathbf{x}) \le \varphi(\mathbf{y}) \quad \forall\, \mathbf{y} \in \mathbb{R}^n$] Trick: write $\mathbf{y}$ as $\mathbf{y} = \mathbf{x} + (\mathbf{y} - \mathbf{x})$. Then,

$$\begin{aligned}
\varphi(\mathbf{y}) &= \varphi\Big(\mathbf{x} + (\mathbf{y} - \mathbf{x})\Big) \\
&= \frac{1}{2}(\mathbf{x} + (\mathbf{y} - \mathbf{x}))^\top A(\mathbf{x} + (\mathbf{y} - \mathbf{x})) - (\mathbf{x} + (\mathbf{y} - \mathbf{x}))^\top\mathbf{b} \\
&= \frac{1}{2}\Big(\overbrace{\mathbf{x}^\top A\mathbf{x}} + \underbrace{\mathbf{x}^\top A(\mathbf{y} - \mathbf{x})} + (\mathbf{y} - \mathbf{x})^\top A\mathbf{x} + (\mathbf{y} - \mathbf{x})^\top A(\mathbf{y} - \mathbf{x})\Big) \overbrace{-\mathbf{x}^\top\mathbf{b}} - (\mathbf{y} - \mathbf{x})^\top\mathbf{b}
\end{aligned}$$

Note that $\mathbf{x}^\top A(\mathbf{y}-\mathbf{x})$ is a scalar, so $\mathbf{x}^\top A(\mathbf{y}-\mathbf{x}) = (\mathbf{y}-\mathbf{x})^\top A\mathbf{x}$. Then, we get

$$\begin{aligned}\varphi(\mathbf{y}) &= \underbrace{\frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{x}^\top\mathbf{b}}_{\varphi(\mathbf{x})} + \frac{1}{2}\cdot 2(\mathbf{y}-\mathbf{x})^\top A\mathbf{x} - (\mathbf{y}-\mathbf{x})^\top\mathbf{b} + \frac{1}{2}(\mathbf{y}-\mathbf{x})^\top A(\mathbf{y}-\mathbf{x}) \\ &= \varphi(\mathbf{x}) + (\mathbf{y}-\mathbf{x})^\top A\mathbf{x} - (\mathbf{y}-\mathbf{x})^\top\mathbf{b} + \frac{1}{2}(\mathbf{y}-\mathbf{x})^\top A(\mathbf{y}-\mathbf{x}) \\ &= \varphi(\mathbf{x}) + (\mathbf{y}-\mathbf{x})^\top \underbrace{(A\mathbf{x}-\mathbf{b})}_{=0} + \frac{1}{2}\underbrace{(\mathbf{y}-\mathbf{x})^\top A(\mathbf{y}-\mathbf{x})}_{\geq 0,\ A \text{ is SPD}}\end{aligned}$$

So, $\varphi(\mathbf{y}) \geq \varphi(\mathbf{x})$ $\square$

($\Leftarrow$): Suppose we wish to find the minimum of

$$\varphi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{x}^\top\mathbf{b}.$$

Use first order condition, we have

$$\boldsymbol{\nabla}\varphi(\mathbf{x}) = A\mathbf{x} - \mathbf{b} \overset{\text{set}}{=} 0 \implies A\mathbf{x} = \mathbf{b}$$

Q.E.D. ■

The previous Theorem 3.2.1 suggests that we can solve $A\mathbf{x} = \mathbf{b}$, when A is SPD, by finding a minimum of the function

$$\varphi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{x}^\top\mathbf{b}.$$

Hence, we motivate steepest descent:

- Start with an initial $\mathbf{x}_0$
- Go downhill by taking a step in the direction of

$$-\boldsymbol{\nabla}\varphi(\mathbf{x}_0) \quad \leftarrow \text{steepest descent direction}$$

- For a general step k:

$$\begin{aligned}&\boldsymbol{\nabla}\varphi(\mathbf{x}_k) = A\mathbf{x}_k - \mathbf{b} \\ \implies &-\boldsymbol{\nabla}\varphi(\mathbf{x}_k) = \mathbf{b} - A\mathbf{x}_k = \mathbf{r}_k \quad \leftarrow \text{residual}\end{aligned}$$

- If $\mathbf{r}_k \neq 0$, we are *not* at the minimum, and there must $\exists$ a scalar $s.t.$

$$\varphi(\mathbf{x}_k + \alpha\mathbf{r}_k) < \varphi(\mathbf{x}_k).$$

The question becomes "how to choose α?" We do so by *line search.*

Consider the function

$$\begin{aligned}\psi(\alpha) &= \varphi(\mathbf{x}_k + \alpha\mathbf{r}_k)\\ &= \frac{1}{2}(\mathbf{x}_k + \alpha\mathbf{r}_k)^\top A(\mathbf{x}_k + \alpha\mathbf{r}_k) - (\mathbf{x}_k + \alpha\mathbf{r}_k)^\top \mathbf{b}\\ &= \frac{1}{2}\mathbf{x}_k^\top A\mathbf{x}_k + \alpha\mathbf{r}_k^\top A\mathbf{x}_k + \frac{1}{2}\alpha^2\mathbf{r}_k^\top A\mathbf{r}_k - \mathbf{x}_k^\top\mathbf{b} - \alpha\mathbf{r}_k^\top\mathbf{b}.\end{aligned}$$

To find the optimal α, we use FoC, set $\psi'(\alpha) = 0$:

$$\begin{aligned}\psi'(\alpha) &= \mathbf{r}_k^\top A\mathbf{x}_k + \alpha\mathbf{r}_k^\top A\mathbf{r}_k - \mathbf{r}_k^\top\mathbf{b}\\ &= \alpha\mathbf{r}_k^\top A\mathbf{r}_k - \mathbf{r}_k^\top(\mathbf{b} - A\mathbf{x}_k)\\ &= \alpha\mathbf{r}_k^\top A\mathbf{r}_k - \mathbf{r}_k^\top\mathbf{r}_k \overset{\text{set}}{=} 0 \implies \boxed{\alpha = \frac{\mathbf{r}_k^\top\mathbf{r}_k}{\mathbf{r}_k^\top A\mathbf{r}_k}}.\end{aligned}$$

Algorithm 13: Steepest Descent

Input: $A \in \mathbb{R}^{n\times n}$ and $\mathbf{b}, \mathbf{x}_0 \in \mathbb{R}^n$

```
begin
    Compute r_0 = b − Ax_0;
    for k = 0, 1, 2, ... do
        w = Ar_k;
        α_k = (r_k^⊤ r_k)/(r_k^⊤ w);
        x_{k+1} = x_k + α_k r_k;
        r_{k+1} = r_k − α_k w;
```

Remark. Why don't we use $\mathbf{r}_{k+1} = \mathbf{b} - A\mathbf{x}_{k+1}$ to calculate the residual?

$$\begin{aligned}\mathbf{r}_{k+1} &= \mathbf{b} - A\mathbf{x}_{k+1}\\ &= \mathbf{b} - A(\mathbf{x}_k + \alpha_k\mathbf{r}_k)\\ &= \mathbf{b} - A^{-1}\mathbf{x}_k - \alpha_k A\mathbf{r}_k\\ &= \mathbf{r}_k - \alpha_k A\mathbf{r}_k\\ &= \mathbf{r}_k - \alpha_k\mathbf{w} && [\textit{Set } \mathbf{w} = A\mathbf{r}_k]\end{aligned}$$

Using $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_{k+1}$ requires an additional matrix vector multiplication, which we can avoid using the recursion $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{r}_{k+1}$.

Remark. The *Stochastic Gradient Descent* method is a form of steepest descent that is used in deep learning to minimize loss functions. It uses only one component, or a small batch of components, of the gradient at each step.

3.2.2 Alternative Derivation of Steepest Descent

Assume A is SPD, and consider

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{r}_k \quad \leftarrow \text{steepest descent update}$$

This time, we try to choose α_k to minimize the A-norm of the error. That is,

$$\mathbf{e}_{k+1} = \mathbf{x} - \mathbf{x}_{k+1}.$$

Recall: A-norm is given by $\|\mathbf{e}_{k+1}\|_A^2 = \mathbf{e}_{k+1}^\top A \mathbf{e}_{k+1}$. Notice that

$$\begin{aligned}
\mathbf{e}_{k+1} &= \mathbf{x} - \mathbf{x}_{k+1} \\
&= \mathbf{x} - (\mathbf{x}_k + \alpha_k \mathbf{r}_k) && \text{[From Steepest Descent]} \\
&= \underbrace{\mathbf{x} - \mathbf{x}_k} - \alpha_k \mathbf{r}_k \\
&= \mathbf{e}_k - \alpha_r \mathbf{r}_k.
\end{aligned}$$

Then, note that since A is SPD, we can write $A = R^\top R$, the Cholesky Factorization. Then,

$$\begin{aligned}
\|\mathbf{e}_{k+1}\|_A^2 &= \mathbf{e}_{k+1}^\top A \mathbf{e}_{k+1} \\
&= \mathbf{e}_{k+1}^\top R^\top R \mathbf{e}_{k+1} \\
&= \|R\mathbf{e}_{k+1}\|_2.
\end{aligned}$$

Thus, we want to minimize

$$\|\mathbf{e}_{k+1}\|_A^2 = \|\mathbf{e}_k - \alpha_k \mathbf{r}_k\|_A^2,$$

or, equivalently, minimize

$$\begin{aligned}
\|R\mathbf{e}_{k+1}\|_2^2 &= \|R(\mathbf{e}_k - \alpha_k \mathbf{r}_k)\|_2^2 \\
&= \|R\mathbf{e}_k - \alpha_k R\mathbf{r}_k\|_2^2
\end{aligned}$$

Let $\widehat{\mathbf{e}}_{k+1} = R\mathbf{e}_{k+1}$, $\widehat{\mathbf{e}}_k = R\mathbf{e}_k$, and $\widehat{\mathbf{r}}_k = R\mathbf{r}_k$. Consider

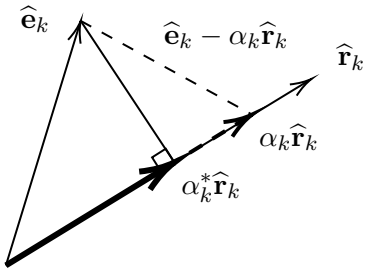
$$\|\widehat{\mathbf{e}}_{k+1}\|_2^2 = \|\widehat{\mathbf{e}}_k - \alpha_k \widehat{\mathbf{r}}_k\|_2^2 \tag{5}$$

Goal: Find α_k to minimize (5).

Visualization: vector projection

The minimum comes from orthogonal projection:

$$\widehat{\mathbf{r}}_k^\top (\widehat{\mathbf{e}}_k - \alpha_k \widehat{\mathbf{r}}_k) = 0 \implies \alpha_k = \frac{\widehat{\mathbf{r}}_k^\top \widehat{\mathbf{e}}_k}{\widehat{\mathbf{r}}_k^\top \widehat{\mathbf{r}}_k}.$$



That is,

$$\alpha_k = \frac{\mathbf{r}_k^\top \overbrace{R^\top R}^{A}\mathbf{e}_k}{\mathbf{r}_k^\top \underbrace{R^\top R}_{A}\mathbf{r}_k} = \frac{\mathbf{r}_k^\top A\mathbf{e}_k}{\mathbf{r}_k^\top A\mathbf{r}_k}.$$

Note: we don't know $\mathbf{e}_k$. But we also don't need it. What we really need is $A\mathbf{e}_k$. Observe that

$$\begin{aligned}
\mathbf{r}_k^\top A\mathbf{e}_k &= \mathbf{r}_k^\top A(\mathbf{x} - \mathbf{x}_k) && [\mathbf{e}_k = \mathbf{x} - \mathbf{x}_k] \\
&= \mathbf{r}_k^\top (A\mathbf{x} - A\mathbf{x}_k) \\
&= \mathbf{r}_k^\top (\mathbf{b} - A\mathbf{x}_k) && [A\mathbf{x} = \mathbf{b}] \\
&= \mathbf{r}_k^\top \mathbf{r}_k && [\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k]
\end{aligned}$$

So,

$$\alpha_k = \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{r}_k^\top A\mathbf{r}_k} \quad \leftarrow \text{This is exactly the same as we found previously.}$$

Remark. We see that if A is SPD, each iteration of steepest descent minimizes $\|\mathbf{e}_{k+1}\|_A$ over $\operatorname{span}\{\mathbf{r}_k\}$.

Recall: CG minimizes $\|\mathbf{e}_{k+1}\|_A$ over $\mathcal{K}_k(A, \mathbf{r}_0) = \{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\}$.

3.2.3 Conjugate Gradient Method

We derived CG using Lanczos method, which shows the connection of CG to Krylov subspaces. Extending the optimization approach will lead us to the same algorithm.

Assume A is SPD and consider

$$\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k\mathbf{p}_k && [\mathbf{p}_k \textit{ is a general descent direction.}] \\
\mathbf{e}_{k+1} &= \mathbf{x} - \mathbf{x}_{k+1}
\end{aligned}$$

- For steepest descent, we use the step direction: $\mathbf{p}_k = \mathbf{r}_k$.
- For conjugate gradient method, use $\mathbf{p}_k = \mathbf{r}_k - \beta_{k-1}\mathbf{r}_{k-1}$.

With the constraint $\mathbf{p}_{k-1}^\top A\mathbf{p}_k = 0$ (A-conjugate direction), we choose step length α_k so that

$$\|\mathbf{e}_{k+1}\|_A \quad \text{is minimized over } \operatorname{span}\{\mathbf{p}_k, \mathbf{p}_{k-1}\}.$$

Using projections and derivations previously, we have

$$\mathbf{p}_k^\top A\mathbf{e}_{k+1} = 0 \quad \text{and} \quad \mathbf{p}_{k-1}^\top A\mathbf{e}_{k+1} = 0.$$

To find β_{k-1}, we want

$$\begin{aligned} \mathbf{p}_{k-1}^\top A\mathbf{p}_k &= 0 \\ \mathbf{p}_{k-1}^\top A(\mathbf{r}_k - \beta_{k-1}\mathbf{p}_{k-1}) &= 0 \\ \implies \beta_{k-1} = \frac{\mathbf{p}_{k-1}^\top A\mathbf{r}_k}{\mathbf{p}_{k-1}^\top A\mathbf{p}_{k-1}} &= \cdots = -\frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{r}_{k-1}^\top \mathbf{r}_{k-1}}. \end{aligned}$$

To find α_k, denote $\mathbf{e}_{k+1} = \mathbf{e}_k - \mathbf{w}$, where $\mathbf{w} = \operatorname{span}\{\mathbf{p}_{k-1}, \mathbf{p}_k\}$. So,

$$\mathbf{e}_{k+1} = \mathbf{e}_k - \mathbf{w} = \mathbf{e}_k - c_1\mathbf{p}_k - c_2\mathbf{p}_{k-1}.$$

Then,

$$\begin{aligned} \mathbf{p}_k^\top A\mathbf{e}_{k+1} &= 0 && \text{[\textit{Optimality condition}]} \\ \mathbf{p}_k^\top A(\mathbf{e}_k - c_1\mathbf{p}_k - c_2\mathbf{p}_{k-1}) &= 0 \\ \mathbf{p}_k^\top A\mathbf{e}_k - c_1\mathbf{p}_k^\top A\mathbf{p}_k - c_2 \underbrace{\mathbf{p}_k^\top A\mathbf{p}_{k-1}}_{=0,\ A\text{-conjugate}} &= 0 \\ \implies c_1 = \frac{\mathbf{p}_k^\top A\mathbf{e}_k}{\mathbf{p}_k^\top A\mathbf{p}_k} &= \frac{\mathbf{p}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top A\mathbf{p}_k}. \end{aligned}$$

Meanwhile,

$$\begin{aligned} \mathbf{p}_{k-1}^\top A\mathbf{e}_{k+1} &= 0 \\ \mathbf{p}_{k-1}^\top A(\mathbf{e}_k - c_1\mathbf{p}_k - c_2\mathbf{p}_{k-1}) &= 0 \\ \underbrace{\mathbf{p}_{k-1}^\top A\mathbf{e}_k}_{\substack{=0,\ \text{optimality} \\ \text{condition}}} - c_1 \underbrace{\mathbf{p}_{k-1}^\top A\mathbf{p}_k}_{=0,\ A\text{-conjugate}} - c_2\mathbf{p}_{k-1}^\top A\mathbf{p}_{k-1} &= 0 \end{aligned}$$

Since $\mathbf{p}_{k-1}^\top A\mathbf{p}_{k-1} \neq 0$ in general, it must be $c_2 = 0$. So,

$$\alpha_k = c_1 = \frac{\mathbf{p}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top A\mathbf{p}_k} = \cdots = \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top A\mathbf{p}_k}.$$

Algorithm 14: Conjugate Gradient

Input: $A \in \mathbb{R}^{n\times n}$ and $\mathbf{b}, \mathbf{x}_0 \in \mathbb{R}^n$

begin
  Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$;
  Set $\mathbf{p}_0 = \mathbf{r}_0$;
  for $k = 0, 1, 2, \ldots$ **do**
    $\mathbf{w} = A\mathbf{p}_k$;
    $\alpha_k = \left(\mathbf{r}_k^\top \mathbf{r}_k\right)/(\mathbf{p}_k^\top \mathbf{w})$;
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{w}$;
    $\beta_k = \left(\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}\right)/\left(\mathbf{r}_k^\top \mathbf{r}_k\right)$;
    $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$;

3.3 Preconditioning

Important Krylov Subspace Iterative Methods to Solve $A\mathbf{x} = \mathbf{b}$ or $\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2$

- We use the conjugate gradient method (CG) when A is SPD.
  1. Or the preconditioned conjugate gradient method (PCG)
  2. Requires one matrix-vector multiplication with A per iteration.
- If A is symmetric, but not positive definite, then we can use the minimum residual method (MINRES)
  1. Requires one matrix-vector multiplication with A per iteration.
- If A is square, nonsingular, but not symmetric, then we can use the generalized minimum residual method (GMRES).
  1. Storage grows at each iteration.
  2. Requires one matrix-vector multiplication with A per iteration.
- For least squares problems, we can use the LSQR method.
  1. This is equivalent to using CG on the normal equations, $A^\top A\mathbf{x} = A^\top \mathbf{b}$.
  2. Requires one matrix-vector multiplication with A and one with $A^\top$ per iteration.
  3. A mathematically equivalent method is known as CGLS.

3.3.1 Introductory Remarks on Convergence

Conjugate Gradient In exact arithmetic, if $A \in \mathbb{R}^{n\times n}$ is SPD, then CG is guaranteed to converge in at most n iterations. [*But n is still too many...*]

- If A is SPD and can be written as $A = I + B$, where B is symmetric and $\mathrm{rank}(B) = r$ (i.e., B is a low rank matrix), then CG will converge in at most $r + 1$ iterations.

  Because B is symmetric, it can be written as $B = V\Lambda V^\top$, where $V^\top V = VV^\top = I$ and

$$\Lambda = \begin{bmatrix} \lambda_1 & & & & & \\ & \ddots & & & & \\ & & \lambda_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix}.$$

  Since $\mathrm{rank}(B) = r$, B has only r nonzero eigenvalues. So, $A = I + B = I + V\lambda V^\top = V(I + \Lambda)V^\top$. Then, $I + \Lambda$ decodes the eigenvalues of A.

$$I + \Lambda = \begin{bmatrix} 1+\lambda_1 & & & & & \\ & \ddots & & & & \\ & & 1+\lambda_r & & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix},$$

  where we know that $n - r$ eigenvalues of A are 1.

- This property implies that if the eigenvalues of A are clustered around a single value (different from 0), then CG will converge fast.
- Meanwhile, for CG, we can also show that

$$\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} \leq 2\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^k,$$

  where $\kappa(A)$ is the condition number of A.

  1. Smallest condition number is $\kappa(A) = 1$. Then,

$$\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} \leq 0 \implies \text{convergence in one step.}$$

  2. If A is well-conditioned (e.g., $\kappa(A) \approx 1$), we get fast convergence.
  3. If A is ill-conditioned, convergence *may* be slow.

- The previous two bullet points are related to each other. If all eigenvalues of A are clustered around a single point, then $\kappa(A) \approx 1$.

LSQR LSQR is equivalent to CG applied to $A^\top A\mathbf{x} = A^\top \mathbf{b}$, so convergence of LSQR depends on the eigenvalues and condition number of $A^\top A$.

- Clustered eigenvalues of $A^\top A \iff$ Clustered singular values of A.
- $\kappa(A^\top A) = \kappa^2(A)$. So, if A is ill-conditioned, $A^\top A$ is even worse.

GMRES More complicated convergence behavior

Definition 3.3.1 (Normal Matrix). If $A \in \mathbb{R}^{n\times n}$ and $A^\top A = AA^\top$, then we say A is a *normal matrix*.

- If A is normal, then A has a spectral decomposition (although eigenvalues and eigenvectors might be complex). That is,

$$A = V\Lambda V^*, \quad \text{where } V^*V = I.$$

  Recall that $V^* = \overline{V}^\top$ is the conjugate transpose (or Hermitian transpose) of V. Also, note that V and V^* are unitary matrix with $\kappa(V) = 1$.
  In this case, if eigenvalues are tightly clustered (away from 0), usually we have fast convergence.
- If A is not normal and $A = P\Lambda P^{-1}$ (diagonalizable) with $\kappa(P) \approx 1$, then tightly clustered eigenvalues (away from 0) usually means fast convergence.
- If A is not normal and $A = P\Lambda P^{-1}$ with large $\kappa(P)$, it's complicated...

3.3.2 Preconditioning

Basic Idea of Preconditioning

- Apply CG (or MINRES, or GMRES, or LSQR, or ...) to a "transformed" system $\widehat{A}\widehat{\mathbf{x}} = \widehat{\mathbf{b}}$, where $\widehat{A}$, $\widehat{\mathbf{x}}$, $\widehat{\mathbf{b}}$ are related to A, $\mathbf{x}$, $\mathbf{b}$.
- $\widehat{A}$ has a more favorable spectrum (e.g., more clustered eigenvalues) than A.
- Compute $\mathbf{x}$ from $\widehat{\mathbf{x}}$.

Left Preconditioning

$$\begin{aligned} A\mathbf{x} = \mathbf{b} &\iff M^{-1}A\mathbf{x} = M^{-1}\mathbf{b} \\ &\iff \widehat{A}\widehat{\mathbf{x}} = \widehat{\mathbf{b}}, \quad \text{where } \widehat{A} = M^{-1}A,\ \widehat{\mathbf{x}} = \mathbf{x},\ \widehat{\mathbf{b}} = M^{-1}\mathbf{b}. \end{aligned}$$

Note that this method can be sued for GMRES but not exactly for CG. This is because $\widehat{A}$ has to be SPD for CG, but $\widehat{A} = M^{-1}A$ is probably not symmetric.

Right Preconditioning

$$\begin{aligned} A\mathbf{x} = \mathbf{b} &\iff AM^{-1}M\mathbf{x} = \mathbf{b} \\ &\iff \widehat{A}\widehat{\mathbf{x}} = \widehat{\mathbf{b}}, \quad \text{where } \widehat{A} = AM^{-1},\ \widehat{\mathbf{x}} = M\mathbf{x},\ \widehat{\mathbf{b}} = \mathbf{b}. \end{aligned}$$

Again, AM^{-1} might not be symmetric.

Remark. When $\min \|\mathbf{b} - A\mathbf{x}\|_2^2$ using LSQR, we use right preconditioning more often, because

- M can be smaller in dimension: $\begin{bmatrix} \\ A \\ \\ \\ \end{bmatrix} \begin{bmatrix} \ M \ \end{bmatrix} \begin{bmatrix} \ M^{-1} \ \end{bmatrix}$
- $\left\|\mathbf{b} - AM^{-1}M\mathbf{x}\right\|_2^2$ doesn't change norm,
  but $\left\|M^{-1}(\mathbf{b} - A\mathbf{x})\right\|_2^2$ changes norm unless M is orthogonal.

Two-Sided Preconditioning Suppose M is SPD, then $M = R^\top R$ (Choleksy). Then,

$$A\mathbf{x} = \mathbf{b} \iff \underbrace{R^{-\top}AR^{-1}}_{\widehat{A}}\underbrace{R\mathbf{x}}_{\widehat{\mathbf{x}}} = \underbrace{R^{-\top}\mathbf{b}}_{\widehat{\mathbf{b}}}$$

In this case, $\widehat{A} = R^{-\top}AR^{-1}$. If A is SPD and R is nonsingular, then $\widehat{A}$ is also SPD.

If we apply CG to this $\widehat{A}\widehat{\mathbf{x}} = \widehat{\mathbf{b}}$, we get *Preconditioned CG* (or, *PCG*).

Remark 1. (Convergence of PCG). Convergence of PCG depends on eigenvalues of

$$\widehat{A} = R^{-\top}AR^{-1}.$$

But notice that

$$\begin{aligned} \widehat{A} &= R^{-\top}AR^{-1} \\ R^{-1}\widehat{A}R &= R^{-1}(R^{-\top}A\underbrace{R^{-1})R}_{=I} = (R^\top R)^{-1}A \\ R^{-1}\widehat{A}R &= M^{-1}A. \end{aligned}$$

So, $R^{-1}\widehat{A}R$ is a similarity transformation and eigenvalues of $\widehat{A}$ are the same as eigenvalues of $M^{-1}A$. So, we can investigate convergence of PCG by studying eigenvalues of $M^{-1}A$. [*One can also show $\widehat{A}$ is similar to AM^{-1}*]

Remark.

- Mathematically, PCG is CG for
$$\widehat{A}\widehat{\mathbf{x}} = \widehat{\mathbf{b}},$$
where $\widehat{A} = R^{-\top}AR^{-1}$, $\widehat{\mathbf{x}} = R\mathbf{x}$, $\widehat{\mathbf{b}} = R^{-\top}\mathbf{b}$, and $M = R^\top R$.
- But, we should *not* explicitly form $\widehat{A}$ or $\widehat{b}$. [*A is huge and sparse, but $\widehat{A}$ might not be sparse any more.*]
- Instead, we should do some algebra to show that the steps of CG on $\widehat{A}\widehat{\mathbf{x}} = \widehat{\mathbf{b}}$ can be done by *only* accessing A, $\mathbf{b}$, and M.

Algorithm 15: Preconditioned Conjugate Gradient (PCG)

Input: $A \in \mathbb{R}^{n\times n}$, $\mathbf{b}, \mathbf{x}_0 \in \mathbb{R}^n$, and SPD $M \in \mathbb{R}^{n\times n}$ [*There is not matrix R!*]

begin
- Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$;
- Solve $M\mathbf{z}_0 = \mathbf{r}_0$;
- Set $\mathbf{p}_0 = \mathbf{z}_0$;
- **for** $k = 0, 1, 2, \ldots$ **do**
  - $\mathbf{w} = A\mathbf{p}_k$;
  - $\alpha_k = (\mathbf{z}_k^\top \mathbf{r}_k)/(\mathbf{p}_k^\top \mathbf{w})$;
  - $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
  - $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{w}$;
  - Solve $M\mathbf{z}_{k+1} = \mathbf{r}_{k+1}$;
  - $\beta_k = (\mathbf{z}_{k+1}^\top \mathbf{r}_{k+1})/(\mathbf{z}_k^\top \mathbf{r}_k)$;
  - $\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k$;

The major work per iteration:

- One matrix-vector multiplication with A, and
- One linear system solve with M.

So, we need to select M so that the linear system solve is easy. For example, diagonal M (and this is called *Jacobi preconditioner*).

Remark. Preconditioners should be designed so that

- M (e.g., $R^\top R$) is not too expensive to construct.
- $M^{-1}A$ has a more favorable spectrum than A.
- Solving $M\mathbf{z} = \mathbf{r}$ should not be too expensive.

These conditions can be in conflict with each other.

Some Choices for Preconditioners

- 1. Jacobi:

$$M = \text{diag}(A) = \begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn} \end{bmatrix}.$$

[*Assumption was that A is SPD. So, diagonal entries are positive.*]

(+) Trivial to construct

(+) Trivial to solve $M\mathbf{z} = \mathbf{b}$

(+) If A has widely-varying diagonals, this may work OK.

2. Similarly, we can use Gauss-Seidel or SOR to construct M.

3. Block versions can also be sued. For example, Block Jacobi:

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \implies M = \begin{bmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{nn} \end{bmatrix}.$$

(+) Each A_{ii} are smaller problems to solve.

(+) Solving each system A_{ii} is independent from other smaller systems $\implies$ completely parallelizable!

- 1. Incomplete Cholesky (`ichol`, for SPD, CG):
Idea: Find triangular matrix R such that:
  - $A \approx R^\top R$
  - R is restricted to have a certain sparsity pattern.
  - Can also use drop tolerance (`droptol`) to get even more sparsity.

2. Incomplete LU factorization (`ilu`, for nonsymmetric A, GMRES)
Similar to incomplete Cholesky, but with

$$A \approx LU.$$

- Sparse Approximate Inverses
- Preconditioners based on insight into the problem.

Example 3.3.2

Consider the PDE:

$$-\boldsymbol{\nabla}(a(x,y)\boldsymbol{\nabla}u) = f(x,y) \quad 0 < x, y < 1 \tag{6}$$

with Dirichlet BCs: $u(0,y) = u(1,y) = u(x,0) = u(x,1) = 0$.

Goal Compute $u(x, y)$.

Discretize to solve: $A\mathbf{u} = \mathbf{f}$. One approach to preconditioning: set $a(x, y) = 1$. Then, (6) becomes

$$-u_{xx} - u_{yy} = f(x, y) \qquad \text{(Poisson's Equation)}$$

Discretize (Poisson's Equation) to get an appropriate M.

3.4 Convergence

3.4.1 GMRES Convergence

The basic idea is that at each iteration k, find $\mathbf{z} \in \mathcal{K}_k(A, \mathbf{r}_0)$ so that $\mathbf{x}_k = \mathbf{x}_0 + \mathbf{z}$

$$\min \|\mathbf{b} - A\mathbf{x}_k\|_2.$$

Suppose we take $\mathbf{x}_0 = \mathbf{0}$. So, $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 = \mathbf{b}$. Then, we want to find $\mathbf{x}_k \in \mathcal{K}_k(A, \mathbf{b})$ to

$$\min \|\mathbf{b} - A\mathbf{x}_k\|_2,$$

where $\mathcal{K}_k(A, \mathbf{b}) = \text{span}\left\{\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \ldots, A^{k-1}\mathbf{b}\right\}$.

In deriving GMRES, we use Arnoldi to find an orthonormal basis $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k\}$ of $\mathcal{K}_k$. Recall that $\mathbf{x}_k \in \mathcal{K}_k(A, \mathbf{b}) \implies \exists$ scalars $y_1, y_2, \ldots, y_k$ such that

$$\begin{aligned}\mathbf{x}_k &= y_1\mathbf{q}_1 + y_2\mathbf{q}_2 + \cdots + y_k\mathbf{q}_k \\ &= Q_k\mathbf{y}.\end{aligned}$$

So, in GMRES, at iteration k, we find $\mathbf{y} \in \mathbb{R}^k$ so that

$$\|\mathbf{b} - A\mathbf{x}_k\|_2 = \|\mathbf{b} - AQ_k\mathbf{y}\|_2 \quad \text{is minimized,} \tag{7}$$

where $Q_k = \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_k\end{bmatrix} \in \mathbb{R}^{n\times k}$. But $\mathbf{x}_k \in \mathcal{K}_k(A, \mathbf{b}) = \text{span}\left\{\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \ldots, A^{k-1}\mathbf{b}\right\}$ also means $\exists$ scalars such that

$$\begin{aligned}\mathbf{x}_k &= c_1\mathbf{b} + c_2A\mathbf{b} + \cdots + c_kA^{k-1}\mathbf{b} \\ &= \underbrace{\begin{bmatrix}\mathbf{b} & A\mathbf{b} & \cdots & A^{k-1}\mathbf{b}\end{bmatrix}}_{K_k}\mathbf{c} \\ &= K_k\mathbf{c}.\end{aligned}$$

Thus, a GMRES alternative to above is that at each iteration, find $\mathbf{c} \in \mathbb{R}^k$ so that

$$\|\mathbf{b} - A\mathbf{x}_k\|_2 = \|\mathbf{b} - AK_k\mathbf{c}\|_2 \quad \text{is minimized,} \tag{8}$$

where $K_k = \begin{bmatrix} \mathbf{b} & A\mathbf{b} & \cdots & A^{k-1}\mathbf{b} \end{bmatrix} \in \mathbb{R}^{n\times k}$

Connection between (7) and (8): If $K_k = Q_k R_k$ (thin, economy QR factorization, where $R_k \in \mathbb{R}^{k\times k}$ is nonsingular), then

$$\begin{aligned} \|\mathbf{b} - AK_k\mathbf{c}\|_2 &= \|\mathbf{b} - AQ_kR_k\mathbf{c}\|_2 \\ &= \|\mathbf{b} - AQ_k\mathbf{y}\|_2, \quad \text{where } \mathbf{y} = R_k\mathbf{c}. \end{aligned}$$

Now, let's consider the alternative derivation (8) of GMRES. Let

$$\begin{aligned} \mathbf{r}_k &= \mathbf{b} - AK_k\mathbf{c} \\ &= \mathbf{b} - A\Big(c_1\mathbf{b} + c_2A\mathbf{b} + \cdots + c_kA^{k-1}\mathbf{b}\Big) \\ &= \mathbf{b} - \Big(c_1A\mathbf{b} + c_2A^2\mathbf{b} + \cdots + c_kA^k\mathbf{b}\Big) \\ &= \underbrace{\Big(I - c_1A - c_2A^2 - \cdots - c_kA^k\Big)}_{p_k(A)}\mathbf{b} \\ &= p_k(A)\mathbf{b}, \end{aligned}$$

where $p_k(t)$ is a polynomial of degree k with $p_k(0) = 1$ (monic polynomial).

Thus, GMREs can now be considered as: at each iteration k, find $p_k \in P_k$ such that

$$\|p_k(A)\mathbf{b}\|_2 \quad \text{is minimized,}$$

where $P_k = \{\text{all polynomials } p \text{ of degree } \le k,\ p(0) = 1\}$. Note that

$$\|\mathbf{r}_k\|_2 = \|p_k(A)\mathbf{b}\|_2 \le \|p_k(A)\|_2\|\mathbf{b}\|_2.$$

The critical factor that determines how quickly $\|\mathbf{r}_k\|_2$ decreases is usually $\|p_k(A)\|_2$. Specifically,

$$\frac{\|\mathbf{r}_k\|_2}{\|\mathbf{b}\|_2} \le \min_{p_k\in P_k} \|p_k(A)\|_2.$$

Question: Given A and K, how small can $\|p_k(A)\|_2$ be?

If A is diagonalizable, then $A = V\Lambda V^{-1}$, where

$\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ are eigenvalues and $V = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{bmatrix}$ are eigenvectors.

Notice that

$$\begin{aligned} A^j &= A\cdot A\cdots A \\ &= V\Lambda\underbrace{V^{-1}V}_{=I}\Lambda V^{-1}\cdots V\Lambda V^{-1} \\ &= V\Lambda^jV^{-1}. \end{aligned}$$

So, if $p \in P_k$, then

$$\begin{aligned} p(A) &= I + c_1 A + c_2 A^2 + \cdots + c_k A^k \\ &= VV^{-1} + c_1 V\Lambda V^{-1} + c_2 V\Lambda^2 V^{-1} + \cdots + c_k V\Lambda^k V^{-1} \\ &= V\Big(I + c_1\Lambda + c_2\Lambda^2 + \cdots + c_k\Lambda^k\Big)V^{-1} \\ &= Vp(\Lambda)V^{-1}. \end{aligned}$$

Hence,

$$\begin{aligned} \|p(A)\|_2 &= \left\|Vp(\Lambda)V^{-1}\right\|_2 \\ &\leq \|V\|_2 \|p(\Lambda)\|_2 \left\|V^{-1}\right\|_2 \\ &= \underbrace{\|V\|_2 \left\|V^{-1}\right\|_2}_{=\kappa(V)} \|p(\Lambda)\|_2 \\ &= \kappa(V)\|p(\Lambda)\|_2. \end{aligned}$$

Now, look at

$$\begin{aligned} p(\Lambda) &= \underbrace{I + c_1\Lambda + c_2\Lambda^2 + \cdots + c_k\Lambda^k}_{\text{all diagonal matrices}} \\ &= \begin{bmatrix} 1 + c_1\lambda_1 + \cdots + c_k\lambda_1^k & & \\ & \ddots & \\ & & 1 + c_1\lambda_k + \cdots + c_k\lambda_k^k \end{bmatrix} \\ &= \begin{bmatrix} p(\lambda_1) & & & \\ & p(\lambda_2) & & \\ & & \ddots & \\ & & & p(\lambda_k) \end{bmatrix}, \end{aligned}$$

where $p(\lambda_j) = 1 + c_1\lambda_j + c_2\lambda_j^2 + \cdots + c_k\lambda_j^k$.

Theorem 3.4.1 Convergence of GMRES
If A is diagonalizable with $A = V\Lambda V^{-1}$, then

$$\frac{\|\mathbf{r}_k\|_2}{\|\mathbf{b}\|_2} \leq \min_{p_k \in P_k} \|p_k(A)\|_2 \leq \kappa(V) \min_{p_k \in P_k} \max_{\lambda_1,\ldots,\lambda_n} |p_k(\lambda_i)|,$$

where $V = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_n \end{bmatrix}$ are eigenvectors of A and $\lambda_1, \ldots, \lambda_n$ are eigenvalues of A.

Remark 1. (Special Case of the Theorem). Suppose A is diagonalizable, and A has m distinct eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ (non-zero) .
If $k \geq m$, then take

$$p_k(t) = \left(1 - \frac{t}{\lambda_1}\right)\left(1 - \frac{t}{\lambda_2}\right)\cdots\left(1 - \frac{t}{\lambda_m}\right) \in P_k$$

and $p_k(\lambda_j) = 0 \quad \forall\, j = 1, 2, \ldots, m$. Then, $p_k(A) = 0$.
That is, GMRES must converge in at most m iterations.

Remark.

- This argument holds only if A is diagonalizable.
- If A is not diagonalizable, then convergence analysis is very complicated.
- If A is diagonalizable, and $\kappa(V)$ is large, there may be a slow decrease in residual at each iteration.
- If A is symmetric (MINRES case), then A is *orthogonally* diagonalizable, $A = V\Lambda V^\top$, where $VV^\top = V^\top V = I$. In this case, $\kappa(V) = 1$. Convergence only depends on eigenvalues.

3.4.2 CG Convergence

Assume $A \in \mathbb{R}^{n\times n}$ is symmetric and positive definite. The approximation problem is: find $p_k \in P_k$ such that

$$\|\mathbf{e}_k\|_A = \|p_k(A)\mathbf{e}_0\|_A \quad \text{is minimized.}$$

Since $\mathbf{e}_k = p_k(A)\mathbf{e}_0$, where $p_k \in P_k$, the set of polynomials with degree $\leq k$ and $p(0) = 1$, we have

$$\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} = \min_{p_k \in P_k} \frac{\|p_k(A)\mathbf{e}_0\|_A}{\|\mathbf{e}_0\|_A}.$$

Since $A = V\Lambda V^\top$ (as A is SPD), then

$$\begin{aligned} p_k(A) = Vp_k(\Lambda)V^\top \implies p_k^\top(A) &= \left(Vp_k(\Lambda)V^\top\right)^\top \\ &= Vp_k^\top(\Lambda)V^\top = Vp_k(\Lambda)V^\top. \end{aligned}$$

So,

$$\begin{aligned} \|p_k(A)\mathbf{e}_0\|_A^2 = (p_k(A)\mathbf{e}_0)^\top A(p_k(A)\mathbf{e}_0) &= \mathbf{e}_0^\top p_k^\top(A)Ap_k(A)\mathbf{e}_0 \\ &= \mathbf{e}_0^\top \overbrace{Vp_k(\Lambda)V^\top}^{p_k^\top(A)}\overbrace{(V\Lambda V^\top)}^{A}\overbrace{(Vp_k(\Lambda)V^\top)}^{p_k(A)}\mathbf{e}_0 \\ &= \mathbf{e}_0^\top Vp_k(\Lambda)\Lambda p_k(\Lambda)V^\top\mathbf{e}_0. \end{aligned}$$

Note that $p_k(\Lambda)\Lambda p_k(\Lambda)$ are all diagonal matrices. So

$$p_k(\Lambda)\Lambda p_k(\Lambda) = D = \text{diag}(\lambda_1 p_k^2(\lambda_1), \lambda_2 p_k^2(\lambda_2), \ldots, \lambda_n p_k^2(\lambda_n)).$$

Let $\mathbf{d} = V^\top \mathbf{e}_0$, then

$$\begin{aligned}\|p_k(A)\mathbf{e}_0\|_A^2 &= \overbrace{\mathbf{e}_0^\top V}^{\mathbf{d}^\top} D \overbrace{V^\top \mathbf{e}_0}^{\mathbf{d}} \\ &= \mathbf{d}^\top D \mathbf{d} = \sum_{j=1}^n d_j^2 \lambda_j p_k^2(\lambda_j).\end{aligned}$$

Also, the denominator is

$$\begin{aligned}\|\mathbf{e}_0\|_A^2 = \mathbf{e}_0^\top A \mathbf{e}_0 &= \underbrace{\mathbf{e}_0^\top V}_{\mathbf{d}^\top} \Lambda \underbrace{V^\top \mathbf{e}_0}_{\mathbf{d}} \\ &= \mathbf{d}^\top \Lambda \mathbf{d} \\ &= \sum_{j=1}^n d_j^2 \lambda_j.\end{aligned}$$

So, we now have

$$\frac{\|p_k(A)\mathbf{e}_0\|_A^2}{\|\mathbf{e}_0\|_A^2} = \frac{\sum_{j=1}^n d_j^2 \lambda_j p_k^2(\lambda_j)}{\sum_{j=1}^n d_j^2 \lambda_j} \leq \frac{\sum_{j=1}^n d_j^2 \lambda_j M^2}{\sum_{j=1}^n d_j^2 \lambda_j},$$

where $M = \max_{\lambda_1,\ldots,\lambda_n} |p_k(\lambda_j)|$.

Theorem 3.4.2 Convergence of CG

If A is SPD with $A = V\Lambda V^{-1}$ (the spectral decomposition), then

$$\begin{aligned}\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} &\leq \min_{p_k \in P_k} \max_{\lambda_1,\ldots,\lambda_n} |p_k(\lambda_j)| \\ &\leq \min_{p \in P_k} \max_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} |p_k(\lambda)|.\end{aligned}$$

Distinct Eigenvalues

- We see again if $\lambda, \ldots, \lambda_m$ with $m \leq n$ are distinct eigenvalues. If $k \geq m$, then

$$p_k(t) = \left(1 - \frac{t}{\lambda_1}\right)\left(1 - \frac{t}{\lambda_2}\right) \cdots \left(1 - \frac{t}{\lambda_m}\right) \in P_k$$

and $p_k(0) = 1$. Also, $p_k(\lambda_j) = 0 \quad \forall\, j = 1, \ldots, m$. So, CG must converge by iteration m.

- Note: A special case of this is when

$$A = I + B,$$

where B is symmetric and $\operatorname{rank}(B) = m - 1$.

$$B = V \begin{bmatrix} \delta_1 & & & & & \\ & \ddots & & & & \\ & & \delta_{m-1} & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix} V^\top.$$

Then,

$$\begin{aligned} A &= I + B \\ &= VIV^\top + V \begin{bmatrix} \delta_1 & & & & & \\ & \ddots & & & & \\ & & \delta_{m-1} & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix} V^\top \\ &= V \left(I + \begin{bmatrix} \delta_1 & & & & & \\ & \ddots & & & & \\ & & \delta_{m-1} & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix} \right) V^\top \\ &= V \begin{bmatrix} \delta_1 + 1 & & & & & \\ & \ddots & & & & \\ & & \delta_{m-1} + 1 & & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix} V^\top. \end{aligned}$$

So, we have m distinct eigenvalues. So, CG on A converges by m iterations.

- Extension: If $A = I + B + E$, with B symmetric has $\operatorname{rank}(B) = m - 1$ and $\|E\|$ is "small," then CG for A will converge by approximately m iterations.

What can be said if we don't know the eigenvalue distribution? For any $p_k^* \in P_k$, we have

$$\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|} \leq \min_{p_k \in P_k} \max_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} |p_k(\lambda)| \leq \max_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} |p_k^*(\lambda)| \leq \boxed{?}$$

- We want to find a $p_k^*(\lambda)$, where $|p_k^*(\lambda)|$ has a tight upper bound.
- We get this from the first kind *Chebyshev polynomials*:

$$T_k(t) = \cos(k \arccos(t)) \quad \text{for } k = 0, 1, \ldots, \quad \text{where } t \in [-1, 1].$$

These may not look like polynomials, but notice:

$$\begin{aligned} T_0(t) &= \cos(0 \cdot \arccos(t)) = \cos(0) = 1 \\ T_1(t) &= \cos(1 \cdot \arccos(t)) = t \\ T_2(t) &= \cos(2 \arccos(t)) \qquad [\cos(2\theta) = 2\cos^2\theta - 1] \\ &= 2\cos^2(\arccos(t)) - 1 = 2t^2 - 1 \end{aligned}$$

It is not difficult to show that (with the right trigonometric identities):

$$T_k(t) + T_{k-2}(t) = 2tT_{k-1}(t), \quad k = 2, 3, \ldots$$

So,

$$T_k(t) = 2tT_{k-1}(t) - T_{k-2}(t), \quad k = 2, 3, \ldots$$

- Important Properties of T_k:

1.

$$\begin{aligned} T_k(-t) = \cos(k \arccos(-t)) &= \cos(k(\pi - \arccos(t))) \\ &= \cos(k\pi - k \arccos(t)) \\ &= \underbrace{\cos(k\pi)}_{\pm 1} \underbrace{\cos(k \arccos(t))}_{T_k(t)} + \underbrace{\sin(k\pi)}_{=0} \sin(\arccos(t)) \\ &= \pm T_k(t). \end{aligned}$$

So,

$$\boxed{|T_k(-t)| = |T_k(t)|}.$$

2. If $\alpha = e^{\mathrm{i}\theta} = \cos\theta + \mathrm{i}\sin\theta$, then $\alpha^{-1} = e^{-\mathrm{i}\theta} = \cos\theta - \mathrm{i}\sin\theta$. Hence,

$$t = \frac{\alpha + \alpha^{-1}}{2} = \frac{e^{\mathrm{i}\theta} + e^{-\mathrm{i}\theta}}{2} = \frac{\cos\theta + \mathrm{i}\sin\theta + \cos\theta - \mathrm{i}\sin\theta}{2} = \cos\theta.$$

So,

$$\begin{aligned} T_k\left(\frac{\alpha+\alpha^{-1}}{2}\right) = T_k(\cos\theta) &= \cos(k\arccos(\cos\theta)) \\ &= \cos(k\theta) \\ &= \frac{e^{\mathrm{i}k\theta}+e^{-\mathrm{i}k\theta}}{2} \\ &= \frac{\alpha^k+\alpha^{-k}}{2}. \end{aligned}$$

So,

$$\boxed{T_k\left(\frac{\alpha+\alpha^{-1}}{2}\right) = \frac{\alpha^k+\alpha^{-k}}{2}}.$$

3. For $t \in [-1,1]$,

$$|T_k(t)| = |\cos(k\arccos(t))| \leq 1.$$

This is a good tight bound on $T_k(t)$, but it only works on $[-1,1]$.

4. $T_k(t)$ is a polynomial of degree $\leq k$. But $T_k(0)$ is not always $= 1$.

- We would like $p_k^* \in P_k$ (i.e., $p_k^*(0) = 1$) that is bounded on $[\lambda_{\min}, \lambda_{\max}]$.

We can get p_k^* by shifting and scaling T_k:

1. Shifting: Let

$$t(\lambda) = \frac{2\lambda - (\lambda_{\max}+\lambda_{\min})}{\lambda_{\max}-\lambda_{\min}}.$$

This maps $\lambda \in [\lambda_{min}, \lambda_{\max}]$ to $t \in [-1,1]$ and

$$|T_k(t(\lambda))| \leq 1 \quad \text{for } \lambda \in [\lambda_{\min}, \lambda_{\max}].$$

2. Scaling: we want $p_k^*(0) = 1$. Let's use

$$p_k^*(\lambda) = \frac{T_k(t(\lambda))}{T_k(t(0))}.$$

[*One can verify that* $T_k(t(0)) \neq 0$.]

Now, we have

$$\begin{aligned} dfrac\|\mathbf{e}_k\|_A\|\mathbf{e}_0\|_A &\leq \min_{p_k\in P_k}\max_{\lambda\in[\lambda_{\min},\lambda_{\max}]}|p_k(\lambda)| \\ &\leq \max_{\lambda\in[\lambda_{\min},\lambda_{\max}]}|p_k^*(\lambda)| \\ &= \max_{\lambda\in[\lambda_{\min},\lambda_{\max}]}\left|\frac{T_k(t(\lambda))}{T_k(t(0))}\right| \\ &\leq \frac{1}{|T_k(t(0))|}. \end{aligned}$$

Now, let's look at $T_k(t(0))$:

$$\begin{aligned} t(0) &= \frac{-(\lambda_{\max} + \lambda_{\min})}{(\lambda_{\max} - \lambda_{\min})} \\ &= \frac{-(\lambda_{\max} + \lambda_{\min})}{(\lambda_{\max} - \lambda_{\min})} \cdot \frac{1/\lambda_{\min}}{1/\lambda_{\min}} \\ &= \frac{-(\lambda_{\max}/\lambda_{\min} + 1)}{(\lambda_{\max}/\lambda_{\min} - 1)} \\ &= -\frac{\kappa(A) + 1}{\kappa(A) - 1} \qquad [\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}}] \end{aligned}$$

So,

$$|T_k(t(0))| = \left|T_k\left(-\frac{\kappa(A)+1}{\kappa(A)-1}\right)\right| = \left|T_k\left(\frac{\kappa(A)+1}{\kappa(A)-1}\right)\right|.$$

Let $\alpha = \dfrac{\sqrt{\kappa(A)}+1}{\sqrt{\kappa(A)}-1}$. Then,

$$\begin{aligned} \alpha + \alpha^{-1} &= \frac{\sqrt{\kappa(A)}+1}{\sqrt{\kappa(A)}-1} + \frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1} \\ &= \frac{2(\kappa(A)+1)}{\kappa(A)-1}. \end{aligned}$$

So,

$$\frac{\alpha + \alpha^{-1}}{2} = \frac{\kappa(A)+1}{\kappa(A)-1} = t(0).$$

Hence,

$$\begin{aligned} |T_k(t(0))| &= \left|T_k\left(\frac{\alpha+\alpha^{-1}}{2}\right)\right| \\ &= \left|\frac{\alpha^k + \alpha^{-k}}{2}\right| \geq \frac{\alpha^k}{2}. \\ \implies \frac{1}{|T_k(t(0))|} &\leq \frac{2}{\alpha^k} = 2\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k. \end{aligned}$$

Theorem 3.4.3 Convergence of CG, II

If A is SPD, then

$$\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} \leq 2\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k.$$

3.4.3 CG Convergence, Revisited

We have shown that

$$\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} \leq \min_{p_k \in P_k} \max_{\lambda_1,\ldots,\lambda_n} |p_k(\lambda_i)|.$$

This tells us that if there are only m distinct eigenvalues, $\lambda_0, \lambda_1, \ldots, \lambda_m$, then CG should converge in at most m iterations. [*But it may still take a little bit more iterations, due to round-off errors.*]

We can see this by exhibiting a polynomial

$$\begin{aligned} p_k(\lambda) &= \left(1 - \frac{\lambda}{\lambda_1}\right)\left(1 - \frac{\lambda}{\lambda_2}\right)\cdots\left(1 - \frac{\lambda}{\lambda_m}\right) \\ &= \left(\frac{\lambda_1 - \lambda}{\lambda_1}\right)\left(\frac{\lambda_2 - \lambda}{\lambda_2}\right)\cdots\left(\frac{\lambda_m - \lambda}{\lambda_m}\right), \end{aligned}$$

which will be zero for all eigenvalues.

We also showed, using properly shifted and scaled Chebyshev polynomials,

$$\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} \leq \min_{p \in P_k} \max_{\lambda \in [\lambda_1, \lambda_n]} |p_k(\lambda)| \leq 2\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^k,$$

where $\lambda_1 = \lambda_{\min}$, $\lambda_n = \lambda_{\max}$, and $\kappa(A) = \dfrac{\lambda_n}{\lambda_n}$, condition number of A.

Now, suppose we have eigenvalues of A distributed like the following

$$\underbrace{0 < \lambda_1 \leq \cdots \leq \lambda_\ell}_{\text{outliers}} \leq \underbrace{\gamma_1 \leq \lambda_{\ell+1} \leq \cdots \leq \lambda_{n-m} \leq \gamma_2}_{\text{cluster}} \leq \underbrace{\lambda_{n-m+1} \leq \cdots \leq \lambda_n}_{\text{outliers}},$$

where $\lambda_1 = \lambda_{\min}$ and $\lambda_n = \lambda_{\max}$. Note that if $\gamma_1 \approx \gamma_2$ (e.g., $\gamma_1 = c - \varepsilon$ and $\gamma_2 = c + \varepsilon$), then we know that there are $n - (m + \ell)$ eigenvalues of A clustered around c.

Outlying Eigenvalues Think about these like the case of having $m + \ell$ distinct eigenvalues.

Let

$$q(\lambda) = \prod_{j=1}^{\ell}\left(\frac{\lambda - \lambda_j}{\lambda_j}\right)\prod_{j=n-m+1}^{n}\left(\frac{\lambda_j - \lambda}{\lambda_j}\right).$$

Note:

- $q(0) = \pm 1$ (we can easily adjust the sign by taking absolute values).
- $q(\lambda_j) = 0$ for all outlying eigenvalues.

  $\implies$ min max of $|q(\lambda)|$ for $\lambda_1, \lambda_2, \ldots, \lambda_n$ has to occur in $[\gamma_1, \gamma_2]$.

Interior (Clustered) Eigenvalues Use scaled and shifted Chebyshev polynomials.

Use $k - \ell - m$ degree Chebyshev polynomial (properly shifted and scaled):

$$\frac{T_{k-\ell-m}\left(\dfrac{\gamma_2 + \gamma_1 - 2\lambda}{\gamma_2 - \gamma_1}\right)}{T_{k-\ell-m}\left(\dfrac{\gamma_2 + \gamma_1}{\gamma_2 - \gamma_1}\right)}.$$

Now, let's look at the whole spectrum :

$$\widehat{p}_k(\lambda) = \frac{T_{k-\ell-m}\left(\dfrac{\gamma_2+\gamma_1-2\lambda}{\gamma_2-\gamma_1}\right)}{T_{k-\ell-m}\left(\dfrac{\gamma_2+\gamma_1}{\gamma_2-\gamma_1}\right)} \prod_{j=1}^{\ell}\left(\frac{\lambda-\lambda_j}{\lambda_j}\right) \prod_{j=n-m+1}^{n}\left(\frac{\lambda_j-\lambda}{\lambda_j}\right).$$

Observe:

$$\begin{aligned}\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} &\le \min_{p_k\in P_k}\max_{\lambda_j}|p_k(\lambda_j)| \\ &\le \underbrace{2\left(\frac{\gamma-1}{\gamma+1}\right)^{k-\ell-m}}_{\substack{\text{comes from the}\\ \text{Chebyshev polynomial}}} \underbrace{\max_{\lambda\in[\gamma_1,\gamma_2]}\left|\prod_{j=1}^{\ell}\left(\frac{\lambda-\lambda_j}{\lambda_j}\right)\prod_{j=n-m+1}^{n}\left(\frac{\lambda_j-\lambda}{\lambda_j}\right)\right|}_{\max \text{ occur within the clustered interval}},\end{aligned}$$

where $\gamma = \sqrt{\dfrac{\gamma_2}{\gamma_1}} \ge 1$.

Theorem 3.4.4 Convergence of CG, III

Let $A \in \mathbb{R}^{n\times n}$ be SPD and we use CG to solve $A\mathbf{x} = \mathbf{b}$. If the eigenvalues of A satisfy

$$0 < \lambda_1 \le \cdots \le \lambda_\ell \le \gamma_1 \le \lambda_{\ell+1} \le \cdots \le \lambda_{n-m} \le \gamma_2 \le \lambda_{n-m+1} \le \cdots \le \lambda_n,$$

then

$$\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} \le 2\left(\frac{\gamma-1}{\gamma+1}\right)^{k-\ell-m} \max_{\lambda\in[\gamma_1,\gamma_2]}\left|\prod_{j=1}^{\ell}\left(\frac{\lambda-\lambda_j}{\lambda_j}\right)\prod_{j=n-m+1}^{n}\left(\frac{\lambda_j-\lambda}{\lambda_j}\right)\right|,$$

where $\gamma = \sqrt{\dfrac{\gamma_2}{\gamma_1}} \ge 1$.

- Suppose $\gamma_1 = c-\varepsilon$ and $\gamma_2 = c+\varepsilon$, where $0 < \varepsilon \ll 1$.

  It is not too difficult ot show that

  $$\frac{\gamma-1}{\gamma+1} = \frac{c-\sqrt{c^2-\varepsilon^2}}{\varepsilon} < \varepsilon.$$

  In this case, we have

  $$\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} \le 2\underbrace{\varepsilon^{k-\ell-m}}_{\substack{\text{will be}\\ \text{small}}} \max_{\lambda\in[\gamma_1,\gamma_2]}\left|\prod_{j=1}^{\ell}\left(\frac{\lambda-\lambda_j}{\lambda_j}\right)\prod_{j=n-m+1}^{n}\left(\frac{\lambda_j-\lambda}{\lambda_j}\right)\right|.$$

- We might expect:

  1. At most 1 iteration for each outlying eigenvalue, and

2. Approximately 1 more iteration for each cluster. [*e.g., if we have 3 clusters, we expect to have approximately 3 more iterations.*]

- Goal for preconditioning:

  Find preconditioner M such that $M^{-1}A$ has clustered eigenvalues.

3.5 LSMR

Recall LSQR: Suppose we want to solve $\min_x \|\mathbf{b} - A\mathbf{x}\|_2$ through LSQR.

- Mathematically, it is equivalent to applying CG on the normal equations, $A^\top A\mathbf{x} = A^\top \mathbf{b}$.
- We use Golub-Kahan bidiagonalization.

That is, give $A, \mathbf{b}$, let $\mathbf{u}_1 = \mathbf{b}/\beta_1$ with $\beta_1 = \|\mathbf{b}\|_2$. GKB iterations compute

$$A\mathbf{v}_k = \alpha_k \mathbf{u}_k + \beta_k \mathbf{u}_{k+1}$$
$$A^\top \mathbf{u}_k = \beta_{k-1}\mathbf{v}_{k-1} + \alpha_k \mathbf{v}_k$$

and

$$AV_k = U_{k+1}B_k, \quad B_k = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \ddots & \ddots & \\ & & & \ddots & \beta_{k-1} \\ & & & & \alpha_k \end{bmatrix}.$$

- At each iteration, LSQR uses GKB to compute

$$\mathbf{y}_k = \arg\min_{\mathbf{y}} \|B_k\mathbf{y} - \beta_1 - \mathbf{e}_1\|_2,$$

  where $\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$, and then set $\mathbf{x}_k = V_k\mathbf{y}_k$.

- Also, recall that CG on normal equations: $A^\top A\mathbf{x} = A^\top \mathbf{b}$.

  CG minimizes $\|\mathbf{e}_k\|^2_{A^\top A} = \|\mathbf{x}^* - \mathbf{x}_k\|^2_{A^\top A}$, where $\mathbf{x}^*$ is the exact solution of the normal equation.

  [*Here, we abuse notation, and* $\mathbf{e}_k$ *= error* $= \mathbf{x}^* - \mathbf{x}_k$.]

Proposition 3.1 : If $\mathbf{x}^*$ is the exact least squares solution, then

$$\|\mathbf{x}^* - \mathbf{x}_k\|^2_{A^\top A} = \|\mathbf{r}_k - \mathbf{r}^*\|_2^2,$$

where $\mathbf{r}^* = \mathbf{b} - A\mathbf{x}^*$.

Proof 1. Note that

$$\begin{aligned}\|\mathbf{x}^* - \mathbf{x}_k\|_{A^\top A}^2 &= (\mathbf{x}^* - \mathbf{x}_k)^\top A^\top A(\mathbf{x}^* - \mathbf{x}_k) \\ &= [A(\mathbf{x}^* - \mathbf{x}_k)]^\top (A(\mathbf{x}^* - \mathbf{x}_k)) \\ &= \|A(\mathbf{x}^* - \mathbf{x}_k)\|_2^2 = \|A\mathbf{e}_k\|_2^2\end{aligned}$$

Also note that

$$\begin{aligned}A(\mathbf{x}^* - \mathbf{x}_k) &= A\mathbf{x}^* - A\mathbf{x}_k \\ &= A\mathbf{x}^* - \mathbf{b} + \mathbf{b} - A\mathbf{x}_k \\ &= -(\mathbf{b} - A\mathbf{x}^*) + (\mathbf{b} - A\mathbf{x}_k)\end{aligned}$$

Denote $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$, the residual from the k-th iteration, and $\mathbf{r}^* = \mathbf{b} - A\mathbf{x}^*$, the residual if we know the solution. Then,

$$A(\mathbf{x}^* - \mathbf{x}_k) = \mathbf{r}_k - \mathbf{r}^*.$$

Hence,

$$\|\mathbf{x}^* - \mathbf{x}_k\|_{A^\top A}^2 = \|\mathbf{r}_k - \mathbf{r}^*\|_2^2.$$

Q.E.D. ■

Proposition 3.2 : $(\mathbf{r}^*)^\top A\mathbf{x} = 0$ for any vector $\mathbf{z}$. In other words, $\mathbf{r}^* \perp \text{range}(A)$.

Proof 2. If $\mathbf{x}^*$ is the exact LS solution, then

$$A^\top(\mathbf{b} - A\mathbf{x}^*) = 0$$

by normal equation. So,

$$\begin{aligned}A^\top(\mathbf{r}^*) &= 0 \\ \mathbf{z}^\top A^\top \mathbf{r}^* &= 0 \quad \text{for any } \mathbf{z} \\ (\mathbf{r}^*)^\top A\mathbf{x} &= 0 \quad \text{for any } \mathbf{z} \qquad [\textit{take transpose.}]\end{aligned}$$

Q.E.D. ■

Proposition 3.3 : $\mathbf{r}_k^\top A\mathbf{v}_i$ for $i = 1, 2, \ldots, k$, where $\mathbf{v}_i$ are vectors computed from GKB.

Proof 3. Note that

$$\begin{aligned}\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k &= \mathbf{b} - AV_k\mathbf{y}_k \\ &= U_{k+1}(\beta_1\mathbf{e}_1 - B_k\mathbf{y}_k) && [U_{k+1}B_k = AV_k] \\ V_k^\top A^\top \mathbf{r}_k &= \underbrace{V_k^\top A^\top U_{k+1}}(\beta_1\mathbf{e}_1 - B_k\mathbf{y}_k) \\ &= B_k^\top(\beta_1\mathbf{e}_1 - B_k\mathbf{y}_k) && [B_k = U_{k+1}^\top AV_k] \\ &= \mathbf{0} && [\mathbf{y}_k = \arg\min_y \|\beta_1\mathbf{e}_1 - B_k\mathbf{y}_k\|_2 \textit{ is a LS problem}]\end{aligned}$$

Take transpose of everything, we have

$$\left(V_k^\top A^\top \mathbf{r}_k\right)^\top = \mathbf{r}_k^\top A V_k = \mathbf{0}.$$

So,

$$\mathbf{r}_k^\top A\mathbf{v}_i = 0 \quad \text{for } i = 1, 2, \ldots, k.$$

Q.E.D. ■

Proposition 3.4 : $(\mathbf{r}^*)^\top(\mathbf{r}_k - \mathbf{r}^*) = 0$.

Proof 4. We know $(\mathbf{r}^*)^\top A\mathbf{z} = 0$ for any $\mathbf{z}$. So,

$$\begin{aligned} (\mathbf{r}^*)^\top A\mathbf{e}_k &= 0 \\ (\mathbf{r}^*)^\top(\mathbf{r}_k - \mathbf{r}^*) &= 0 \qquad \text{[Proposition 3.1]} \end{aligned}$$

Q.E.D. ■

Proposition 3.5 : $\|\mathbf{r}_{k+1}\|_2^2 \leq \|\mathbf{r}_k\|_2^2$. That is, $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$ are monotonically non-increasing.

Proof 5. Since we are using CG on the normal equations, we know

$$\begin{aligned} \|\mathbf{e}_{k+1}\|_{A^\top A}^2 &\leq \|\mathbf{e}_k\|_{A^\top A}^2 \\ \|\mathbf{r}_{k+1} - \mathbf{r}^*\|_2^2 &\leq \|\mathbf{r}_k - \mathbf{r}^*\|_2^2 \qquad \text{[Proposition 3.1]} \end{aligned}$$

But notice that

$$\mathbf{r}_k = \mathbf{r}^* + (\mathbf{r}_k - \mathbf{r}^*).$$

Then,

$$\begin{aligned} \|\mathbf{r}_k\|_2^2 &= \|\mathbf{r}^* + (\mathbf{r}_k - \mathbf{r}^*)\|_2^2 \\ &= (\mathbf{r}^* + (\mathbf{r}_k - \mathbf{r}^*))^\top(\mathbf{r}^* + (\mathbf{r}_k - \mathbf{r}^*)) \\ &= (\mathbf{r}^*)^\top\mathbf{r}^* + 2\underbrace{(\mathbf{r}^*)^\top(\mathbf{r}_k - \mathbf{r}^*)}_{=0} + (\mathbf{r}_k - \mathbf{r}^*)^\top(\mathbf{r}_k - \mathbf{r}^*) \\ &= \|\mathbf{r}^*\|_2^2 + \|\mathbf{r}_k - \mathbf{r}^*\|_2^2 \qquad \text{[Proposition 3.4]} \end{aligned}$$

So, $\|\mathbf{r}_k - \mathbf{r}^*\|_2^2 = \|\mathbf{r}_k\|_2^2 - \|\mathbf{r}^*\|_2^2$. Similarly, $\|\mathbf{r}_{k+1} - \mathbf{r}^*\|_2^2 = \|\mathbf{r}_{k+1}\|_2^2 - \|\mathbf{r}^*\|_2^2$. We have previously shown

$$\begin{aligned} \|\mathbf{r}_{k+1} - \mathbf{r}^*\|_2^2 &\leq \|\mathbf{r}_k - \mathbf{r}^*\|_2^2 \\ \|\mathbf{r}_{k+1}\|_2^2 - \cancel{\|\mathbf{r}^*\|_2^2} &\leq \|\mathbf{r}_k\|_2^2 - \cancel{\|\mathbf{r}^*\|_2^2} \\ \|\mathbf{r}_{k+1}\|_2^2 &\leq \|\mathbf{r}_k\|_2^2. \end{aligned}$$

Q.E.D. ■

In summary

- LSQR residuals $\|\mathbf{r}_k\|_2 = \|\mathbf{b} - A\mathbf{x}_k\|_2$ are monotonically non-increasing.

- But it doesn't say that the normal equation residual

$$\left\|A^\top \mathbf{r}_k\right\|_2 = \left\|A^\top(\mathbf{b} - A\mathbf{x}_k)\right\|_2$$

  decrease monotonically.

LSMR

Like LSQR: we use GKB, but for the projected system, it solves

$$\mathbf{y}_k = \arg\min_{\mathbf{y}} \left\|B_k^\top(B_k\mathbf{y} - \beta_1\mathbf{e}_1)\right\|_2$$
$$\mathbf{x}_k = V_k\mathbf{y}_k.$$

This means that $\mathbf{x}_k$ minimizes

$$\left\|A^\top \mathbf{r}_k\right\|_2 = \left\|A^\top(\mathbf{b} - A\mathbf{x}_k)\right\|_2,$$

the normal equation residual. It corresponds to MINRES on the normal equations.

- Algorithm is very similar to LSQR>
- LSMR ensures $\left\|A^\top \mathbf{r}_k\right\|_2$ is monotonically non-increasing.
- $\|\mathbf{r}_k\|_2$ is "nearly" monotonically non-increasing.

4 Inverse Problems and Iterative Methods

4.1 Introduction to Inverse Problems

A linear inverse problem is usually written as

$$\mathbf{b} = A\mathbf{x}_{\text{exact}} + \boldsymbol{\eta} = \mathbf{b}_{\text{exact}} + \boldsymbol{\eta},$$

where $\boldsymbol{\eta}$ is a vector representing unknown errors/noise in the measured data, $\mathbf{b}$. [*We don't know $\boldsymbol{\eta}$, but we might know $\|\boldsymbol{\eta}\|$.*]

Ideally, we want to solve

$$A\mathbf{x}_{\text{exact}} = \mathbf{b}_{\text{exact}},$$

but we don't know $\mathbf{b}_{\text{exact}}$. So, since we only know $\mathbf{b}$, if we assume $\boldsymbol{\eta}$ is "small," we can try to solve

$$A\mathbf{x} = \mathbf{b} \iff \mathbf{x}_{\text{naïve}} = A^{-1}\mathbf{b}.$$

We call this method the *naïve inverse solution.*

$$\begin{aligned}
\mathbf{x}_{\text{naïve}} &= A^{-1}\mathbf{b} \\
&= A^{-1}(\mathbf{b}_{\text{exact}} + \boldsymbol{\eta}) \\
&= A^{-1}\mathbf{b}_{\text{exact}} + A^{-1}\boldsymbol{\eta} \\
&= \mathbf{x}_{\text{exact}} + A^{-1}\boldsymbol{\eta}.
\end{aligned}$$

Note that $A^{-1}\mathbf{b}_{\text{exact}} = \mathbf{x}_{\text{exact}}$ is the *exact solution* (i.e., what we want), but $A^{-1}\boldsymbol{\eta}$ is the *inverted noise.* This noise may be "large" and may dominate/hide anything in $\mathbf{x}_{\text{exact}}$. To see this, we will use SVD.

Singular Value Expansion of A

$$A = U\Sigma V^\top = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^\top \\ \mathbf{v}_2^\top \\ \vdots \\ \mathbf{v}_n^\top \end{bmatrix} = \sum_{i=1}^{n} \sigma \mathbf{u}_i \mathbf{v}_i^\top,$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$ and $\mathbf{u}_i\mathbf{v}_i^\top$ are rank-1 matrices.

Singular Value Expansion of A^{-1}

$$A^{-1} = V\Sigma^{-1}U^\top = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{bmatrix} \begin{bmatrix} 1/\sigma_1 & & & \\ & 1/\sigma_2 & & \\ & & \ddots & \\ & & & 1/\sigma_n \end{bmatrix} \begin{bmatrix} \mathbf{u}_1^\top \\ \mathbf{u}_2^\top \\ \vdots \\ \mathbf{u}_n^\top \end{bmatrix} = \sum_{i=1}^{n} \frac{1}{\sigma_i} \mathbf{v}_i \mathbf{u}_i^\top,$$

where $\mathbf{v}_i\mathbf{u}_i^\top$ are rank-1 matrices.

Now, let's reconsider the naïve inverse solution approximation:

$$\begin{aligned}
\mathbf{x}_{\text{naïve}} = A^{-1}\mathbf{b} &= \sum_{i=1}^{n} \frac{1}{\sigma_i}\mathbf{v}_i \underbrace{\mathbf{u}_i^\top \mathbf{b}}_{\text{scalar}} \\
&= \sum_{i=1}^{n} \frac{\mathbf{u}_i^\top \mathbf{b}}{\sigma_u}\mathbf{v}_i && [\textit{linear combination of } \mathbf{v}_i\textit{'s}] \\
&= \sum_{i=1}^{n} \frac{\mathbf{u}_i^\top (\mathbf{b}_{\text{exact}} + \boldsymbol{\eta})}{\sigma_i}\mathbf{v}_i \\
&= \sum_{i=1}^{n} \frac{u_i^\top \mathbf{b}_{\text{exact}}}{\sigma_i}\mathbf{v}_i + \sum_{i=1}^{n} \frac{\mathbf{u}_i^\top \boldsymbol{\eta}}{\sigma_i}\mathbf{v}_i \\
&= \mathbf{x}_{\text{exact}} + A^{-1}\boldsymbol{\eta}
\end{aligned}$$

What can we say about $A^{-1}\boldsymbol{\eta}$?

- If $\boldsymbol{\eta}$ is small, $\mathbf{u}_i^\top \boldsymbol{\eta}$ is small.
- $A^{-1}\boldsymbol{\eta}$ will be small if $\dfrac{\mathbf{u}_i^\top \boldsymbol{\eta}}{\sigma_i}$ are small.

Properties of Inverse Problems

- $\left|\mathbf{u}_i^\top \boldsymbol{\eta}\right|$ are small, and roughly the same order of magnitude for all i.
- $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ with $\sigma_n \approx 0$.
- Since $\sigma_j \to 0$ fast, σ_j is tiny for large j. The matrix A is very ill-conditioned ($\kappa(A)$ is huge).

$$\text{residual} = \frac{\|\mathbf{b} - A\mathbf{x}_k\|_2}{\|\mathbf{b}\|_2}$$

  In tis cases, residuals do not tell us how good the solution is,

- For large σ_i, $\sigma_i > \left|\mathbf{u}_i^\top \boldsymbol{\eta}\right|$
  For small σ_i, $\sigma_i < \left|\mathbf{u}_i^\top \boldsymbol{\eta}\right|$,
  and $\sigma_n \ll \left|\mathbf{u}_n^\top \boldsymbol{\eta}\right|$.
- The singular vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ oscillate more as i increases (that is, as σ_i decreases). Dividing by small σ_i amplifies oscillations in $\mathbf{v}_i$.

Question Can we compute an approximation where the inverted noise is not too large?

One idea Truncated SVD

Leave out components of $\mathbf{x}_{\text{naïve}}$ corresponding to small singular values. That is,

$$\mathbf{x}_k = \sum_{i=1}^{k} \frac{\mathbf{u}_i^\top \mathbf{b}}{\sigma_i}\mathbf{v}_i, \quad k < n \qquad [k \textit{ is not an iteration but where we truncate.}]$$

Note that

$$\mathbf{x}_k = \sum_{i=1}^{k} \frac{\mathbf{u}_i^\top \mathbf{b}_{\text{exact}}}{\sigma_i}\mathbf{v}_i + \sum_{i=1}^{k} \frac{\mathbf{u}_i^\top \boldsymbol{\eta}}{\sigma_i}\mathbf{v}_i.$$

If we select k well, we should have small $\boldsymbol{\eta}$ term,. while preserving information from $\mathbf{b}_{\text{exact}}$.

Theorem 4.1.1 Picard Condition Assumption
$\left|\mathbf{u}_i^\top \mathbf{b}_{\text{exact}}\right|$ decays on average faster than σ_i

With this assumption,

- $\displaystyle\sum_{i=1}^{k} \frac{\mathbf{u}_i^\top \mathbf{b}_{\text{exact}}}{\sigma_i}\mathbf{v}_i$ $\begin{cases} \text{small } k: \text{ get info about solution} & \leftarrow \text{Good!} \\ \text{large } k: \text{ terms} \to 0 \implies \text{we can discard these w/o losing much info} \end{cases}$
- $\displaystyle\sum_{i=1}^{k} \frac{\mathbf{u}_i^\top \boldsymbol{\eta}}{\sigma_i}\mathbf{v}_i$ $\begin{cases} \text{small } k: \text{ terms} \approx 0 & \leftarrow \text{Good!} \\ \text{large } k: \left|\mathbf{u}_i^\top \boldsymbol{\eta}\right| > \sigma_i, \text{ so can be large} & \leftarrow \text{BAD...} \end{cases}$

What is a good k? More later... But generally speaking, with Truncated SVD (TSVD), we will choose an optimal k_{opt} that balances reconstructing enough info, without letting the noise blow up:

$$\mathbf{x}_{\text{tsvd}} = \sum_{i=1}^{k_{\text{opt}}} \frac{\mathbf{u}_i^\top \boldsymbol{\eta}}{\sigma_i}\mathbf{v}_i, \quad 1 \le k_{\text{opt}} \le n.$$

4.2 Krylov Methods for Inverse Problems

Summary of Inverse Problems

- Large singular value components correspond to the solution.

  We want to reconstruct theses.
- Small singular value components magnify noise.

  We don't want to reconstruct these.
- The SVD is a great tool, but is too expensive for large-scale problems.
- We will use Krylov subspace iterative methods to solve inverse problems.

First consider $A \in \mathbb{R}^{n\times n}$, symmetric and positive definite. Then, all eigenvalues are positive, and

$$A = V\Lambda V^\top, \qquad \text{(Spectral Decomposition)}$$

where $VV^\top = V^\top V = I$, and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$.

If we write $V = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{bmatrix}$, then

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i, \quad A^2\mathbf{v}_i = \lambda_i A\mathbf{v}_i = \lambda_i^2 \mathbf{v}_i, \quad \ldots, \quad A^j\mathbf{v}_i = \lambda_i^j \mathbf{v}_i.$$

Assume $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.

Suppose we run k iterations of Lanczos on A with initial vector $\mathbf{q}$, with $\|\mathbf{q}\|_2 = 1$. This generates

$$\mathbf{q}_1 = \mathbf{q}, \quad \mathbf{q}_2, \quad \mathbf{q}_3, \quad \ldots, \quad \mathbf{q}_k,$$

an orthonormal basis for the Krylov subspace

$$\mathcal{K}_k(A, \mathbf{q}) = \operatorname{span}\left\{\mathbf{q}, A\mathbf{q}, A^2\mathbf{q}, \ldots, A^{k-1}\mathbf{q}\right\}$$

and

$$Q_k^\top A Q_k = T_k \in \mathbb{R}^{k\times k}.$$

What can we say about the vectors in $\mathcal{K}_k(A, \mathbf{q})$?

- Notice
$$A^j = \left(V\Lambda V^\top\right)^j = V\Lambda^j V^\top,$$
where $\Lambda^j = \operatorname{diag}(\lambda_1^j, \lambda_2^j, \ldots, \lambda_n^j)$.
- Consider vectors in $\mathcal{K}_k(A, \mathbf{q})$:
$$A^j\mathbf{q} = V\Lambda^j V^\top \mathbf{q} = V\Lambda^j \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \quad \mathbf{c} = V^\top \mathbf{q}$$
$$= V\begin{bmatrix} c_1\lambda_1^j \\ c_2\lambda_2^j \\ \vdots \\ c_n\lambda_n^j \end{bmatrix} = c_1\lambda_1^j V \underbrace{\begin{bmatrix} 1 \\ \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^j \\ \vdots \\ \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^j \end{bmatrix}}_{(\star)}$$

1. We assume $c_1 \neq 0$ (so that we can divide safely). If $c_1 = 0$, just choose another $\mathbf{q}$ to construct $\mathcal{K}_k(A, \mathbf{q})$ again.
2. We assume $\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n > 0$.
3. If $\lambda_1 \gg \lambda_2$, then $\dfrac{\lambda_i}{\lambda_1} \ll 1$, and it goes to 0 fast for large j.
4. Since $\dfrac{\lambda_i}{\lambda_1} < 1$ for $j = 2, \ldots, n$, we know that the vector $(\star) \to \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ as j increases.

So, let $c = c_1\lambda_1^j$ be a scalar, we have

$$A^j\mathbf{q} \to cV\begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix} = c\mathbf{v}_1,$$

where $\mathbf{v}_1$ is the eigenvector of A corresponding to its largest eigenvalue λ_1.

- Also notice that if we let $\widehat{\mathbf{q}} = A^j\mathbf{q}$, then

$$\begin{aligned}\widehat{\mathbf{q}}^\top A\widehat{\mathbf{q}} \to c^2\mathbf{v}_1^\top A\mathbf{v}_1 &= c^2\mathbf{v}_1^\top(\lambda_1\mathbf{v}_1) && [A\mathbf{v}_1 = \lambda_1\mathbf{v}_1]\\ &= c^2\lambda_1\mathbf{v}_1^\top\mathbf{v}_1 = c^2\lambda_1 && [\mathbf{v}_1^\top\mathbf{v}_1 = 1]\\ \implies \frac{\widehat{\mathbf{q}}^\top A\widehat{\mathbf{q}}}{c^2} &\to \lambda_1.\end{aligned}$$

- This implies that at early iterations of Lanczos,
  1. The largest eigenvalue of $T_k = Q_k^\top AQ_k$ will be a good approximation of the largest eigenvalue of A.
  2. Also, the Krylov subspace $\mathcal{K}_k(A,\mathbf{q})$ is closely aligned with $\mathbf{v}_1$ (eigenvector of A).
- This can be generalized:
  1. If $\lambda_1 > \lambda_2 > \cdots > \lambda_n$, then the early iterations of Lanczos produce $T_k = Q_k^\top AQ_k$ with largest eigenvalues approximating the largest eigenvalues of A.
  2. If the gaps between $\lambda_1, \lambda_2, \ldots, \lambda_n$ are large, convergence will be faster.

Bidiagonalization Methods for More General $A \in \mathbb{R}^{m\times n}$

- Golub-Kahan Bidiagonalization (GKB): produces upper bidiagonal matrix.

$$AV_k = U_kB_k \implies A^\top U_k - V_kB_k^\top + \text{rank-1 matirx}$$

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

- Paige-Saunders Bidiagonalization: produces lower bidiagonal matrix.

$$AV_k = U_k\widehat{B}_k + \text{rank-1 matrix} \implies A^\top U_k = V_k\widehat{B}_k$$

$$\widehat{B}_k = \begin{bmatrix}\alpha_1 & & & \\ \beta_1 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_{k-1} & \alpha_k\end{bmatrix}$$

- Lanczos on $A^\top A$ produces $T_k = B_k^\top B_k$.
  Lanczos on $AA^\top$ produces $T_k = \widehat{B}_k \widehat{B}_k^\top$.
- Eigenvalues of $A^\top A$ or $AA^\top \longleftrightarrow$ Singular values squared of A.

Lanczos Property Summary

If $A \in \mathbb{R}^{n\times n}$ is SPD, then at early iterations of Lanczos,

- The largest eigenvalues of
$$T_k = Q_k^\top A Q_k \in \mathbb{R}^{k\times k}$$
tend to be good approximations of the largest eigenvalues of A.
- The Krylov subspace tends to contain components of corresponding eigenvectors of A.

How fast the approximations become good depends on the spread of eigenvalues.

Bidiagonalization Property Summary

If $A \in \mathbb{R}^{m\times n}$, then at early iterations of Golub-Kahan Bidiagonalization,

- The largest singular values of
$$B_k = U_k^\top A V_k \in \mathbb{R}^{k\times k}$$
thend to be good approximations of the largest singular values of A.
- The Krylov subspace tends to contain components of corresponding singular vectors of A.

How fast the approximations become good depends on the spread of singular valules.

What This Means for Inverse Problems? If we use CG or LSQR, then

- Early iterations:
  Reconstruct information corresponding to largest singular value components.
  approximate solution gets better at each iteration.
  This is called *semi-convergence*.
- Later iterations:
  Reconstruct inverted noise.
  approximate solution starts to get worse.
- If we iterate too long:
  Then, $\mathbf{x}_k \to A^{-1}\mathbf{b} = \mathbf{x}_{\text{naïve}}$. This is a bad solution.
  So, *semi-convergence* does converge, but it converges to something bad eventually.

- Iterative methods behaves like:

  TSVD, but we don't need to compute the full SVD on A!

- Questions?

  1. Semi-convergence applies to $\frac{\|\mathbf{x}_k - \mathbf{x}_{\text{exact}}\|_2}{\|\mathbf{x}_{\text{exact}}\|_2}$. That is, we need to relative error to determine the optimal iteration number k. However, we don't know $\mathbf{x}_{\text{exact}}$.
  2. The residual does not show when semi-convergence occurs because the problems are ill-conditioned ($\kappa(A)$ is huge). So, how to determine when to stop?

4.3 Iterative Regularization

4.3.1 Motivation

A linear inverse problem is usually written as

$$\mathbf{b} = A\mathbf{x}_{\text{exact}} + \boldsymbol{\eta},$$

where $\boldsymbol{\eta}$ is a vector representing (unknown) erros/noise in the measured data $\mathbf{b}$.

- Ideally, we would like to solve

  $$A\mathbf{x}_{\text{exact}} = \mathbf{b}_{\text{exact}},$$

  but we don't know $\mathbf{b}_{\text{exact}}$.

- We could try to ignore $\boldsymbol{\eta}$ and compute the naïve inverse solution:

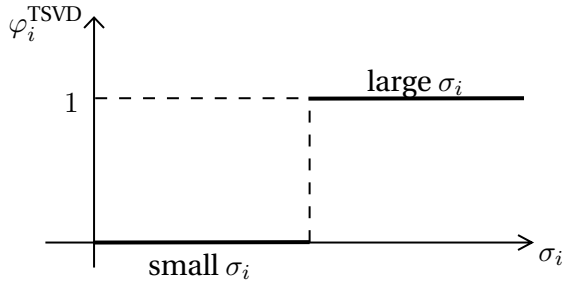
  $$\mathbf{x}_{\text{naïve}} = A^{-1}\mathbf{b},$$

  but ill-conditioning and even a small amount of noise produces a bad approximation to $\mathbf{x}_{\text{exact}}$.

- Truncated SVD (TSVD): Choose an optimal k_{opt} that balances reconstructing enough info, without letting the noise blow up:

  $$\begin{aligned}\mathbf{x}_{\text{tsvd}} &= \sum_{i=1}^{k_{\text{opt}}} \frac{\mathbf{u}_i^\top \mathbf{b}}{\sigma_i}\mathbf{v}_i, \quad 1 \le k_{\text{opt}} \le n \\ &= \sum_{i=1}^{n} \varphi_i \frac{\mathbf{u}_i^\top \mathbf{b}}{\sigma_i}\mathbf{v}_i, \quad \text{where } \varphi_i^{\text{TSVD}} = \begin{cases} 1 & \text{for "large" } \sigma_i \\ 0 & \text{for "small" } \sigma_i. \end{cases}\end{aligned}$$

  Notation 4.1. k_{opt} is called the *TSVD regularization parameter*.



- General SVD Filtering:

$$\mathbf{x}_{\text{filt}} = \sum_{i=1}^{n} \varphi_i \frac{\mathbf{u}_i^\top \mathbf{b}}{\sigma_i} \mathbf{v}_i, \quad \text{where } \varphi_i \approx \begin{cases} 1 & \text{for "large" } \sigma_i \\ 0 & \text{for "small" } \sigma_i \end{cases}$$

Example 4.3.2 Regularization Parameter and Tikhonov Filtering

Consider the following Tikhonov filtering:

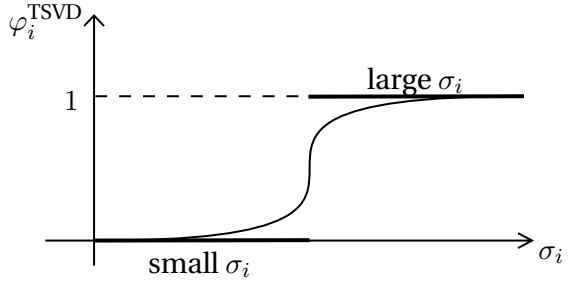
$$\varphi_i^{\text{Tik}} = \frac{\sigma_i^2}{\sigma_i^2 + \lambda^2}.$$

For example, $\lambda = 10^{-3} \implies \lambda^2 = 10^{-6}$. If $\sigma_i = 1$, then

$$\varphi_i^{\text{Tik}} = \frac{1}{1 + 10^{-6}} \approx 1,$$

and if $0 \approx \sigma_n \ll \lambda$, then

$$\varphi_0^{\text{Tik}} \approx 0.$$



Here, λ is called the *regularization parameter*. The region in-between is called the *transition area*.

We can motivate the Tikhonoc filtering by considering the *Tikhonov optimization problem*:

$$\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda^2 \|\mathbf{x}\|_2^2 = \min_{\mathbf{x}} \left\| \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} A \\ \lambda I \end{bmatrix} \mathbf{x} \right\|_2^2.$$

Or, the normal equations version:

$$\left(A^\top A + \lambda^2 I\right)\mathbf{x} = A^\top \mathbf{b}.$$

Replace A with $A = U\Sigma V^\top$ (SVD), and solve for $\mathbf{x}$, we get

$$\mathbf{x}^{\text{Tik}} = \sum_{i=1}^{n} \underbrace{\frac{\sigma_i^2}{\sigma_i^2 + \lambda^2}}_{\varphi_i^{\text{Tik}}} \frac{\mathbf{u}_i^\top \mathbf{b}}{\sigma_i}\mathbf{v}_i$$

Recall that CG/LSQR behave like TSVD:

- Early iterations: Reconstruct information corresponding to large singular value components.
- Later iterations: Start to reconstruct inverted noise.
- Iterate too long: Converge to the naïve inverse solution, which is dominated by the inverted noise.
- If we stop at the right spot, $\mathbf{x}_{\text{opt}}$, we get an approximate solution similar to TSVD.

We can write CG/LSQR solutions as filtering methods. Specifically, at each iteration, we can write

$$\mathbf{x}_k^{\text{LSQR}} = \sum_{i=1}^{n} \varphi_{i,k}^{\text{LSQR}} \frac{\mathbf{u}_i^\top \mathbf{b}}{\sigma_i}\mathbf{v}_i,$$

where

$$\varphi_{i,k}^{\text{LSQR}} = \sigma_i^2 R_k(\sigma_k^2), \quad R_k(t) \text{ is a (rather complicated) polynomial.}$$

There is no closed form for $R_k(t)$, but we can write a recursive formula.

- At early iterations, (e.g., $k = 1, 2, 3$),

$$\varphi_{i,k}^{\text{LSQR}} = \begin{cases} 1 & \text{for a few large } \sigma_i's \\ 0 & \text{for most of the rest.} \end{cases}$$

  So, it's filtering *a lot* in early iterations.

- As we iterate further,

$$\varphi_{i,k}^{\text{LSQR}} \approx \begin{cases} 1 & \text{for more of the large } \sigma_i's \\ 0 & \text{for fewer of the small } \sigma_i's. \end{cases}$$

- Eventually, if we iterate long enough,

$$\varphi_{i,k}^{\text{LSQR}} \approx 1 \quad \text{for all } \sigma_i's.$$

However, the question remains: how to choose k_{opt}?

4.3.2 Landweber Iteration

The Landweber method is the most used iterative methods for nonlinear inverse problems, and it makes a nice (and relatively easy) connection to SVD filtering for iterative methods. To motivate it, look at a simple iteration:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \underbrace{A^\top(\mathbf{b} - A\mathbf{x}_k)}_{\substack{\text{Normal equation;} \\ \text{Steepest descent} \\ \text{direction}}},$$

where:

- If we take α_k as the step size from line search, we get steepest descent method.
- If we use constant step size,

  $$\alpha_k = \alpha,$$

  we get *Landweber* or *Richardson Iteration*.

From the convergence results for simple iteration, we know it converges if

$$\rho\Big(I - \alpha_k A^\top A\Big) < 1,$$

where $\rho(\cdot)$ denotes the spectral radius.

Landweber Convergence Assume $\alpha_k = \alpha$ is constant.

Let $A = U\Sigma V^\top$. Then,

$$\begin{aligned} I - \alpha A^\top A &= V\Big(I - \alpha\Sigma^\top\Sigma\Big)V^\top \\ &= V\,\text{diag}\left(1 - \alpha\sigma_i^2\right)V^\top. \end{aligned}$$

So, for convergence, we need

$$\begin{aligned} \left|1 - \alpha\sigma_i^2\right| < 1 \implies -1 < 1 - \alpha\sigma_i^2 &< 1 \\ -2 < -\alpha\sigma_i^2 &< 0 \\ 0 < \alpha\sigma_i^2 &< 2 \\ 0 < \alpha &< \frac{2}{\sigma_i^2}. \end{aligned}$$

Therefore, we need

$$0 < \alpha < \frac{2}{\sigma_{\max}^2} = \frac{2}{\|A^\top A\|_2}.$$

But...How do we estimate $\sigma_{\max}^2 = \left\|A^\top A\right\|_2$?

- From Numerical Linear Algebra, $\left\|A^\top A\right\|_2 \le \|A\|_1\|A\|_\infty$.

- If A contains no negative values [*This may not be a bad assumption in inverse problems*], then

$$\|A\|_\infty = \text{max entry of } A\mathbf{1}$$
$$\|A\|_1 = \text{max entry of } A^\top \mathbf{1}.$$

- Therefore, we can take

$$\alpha = \frac{c}{\|A\|_1 \|A\|_\infty} \leq \frac{c}{\|A^\top A\|_2}, \quad 0 < c < 2.$$

Remark. If we run Landweber to "convergence,"

$$\mathbf{x}_k \to \text{naïve inverse solution (BAD!)}$$

So, we would wonder if early stopping working as LSQR for Landweber.

Landweber and SVD Filtering

Definition 4.3.3 (SVD Filtering Method). If $\mathbf{b} = A\mathbf{x}_{\text{exact}} + \boldsymbol{\eta}$, then an *SVD filtering method* is

$$\mathbf{x}_{\text{filt}} = \sum_{i=1}^{n} \varphi_i \frac{\mathbf{u}_i^\top \mathbf{b}}{\sigma_i} \mathbf{v}_i, \quad \text{where } \varphi_i \approx \begin{cases} 1 & \text{if } \sigma_i \text{ is large} \\ 0 & \text{if } \sigma_i \text{ is small.} \end{cases}$$

Now, consider

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha A^\top (\mathbf{b} - A\mathbf{x}_k) \\ &= \left(I - \alpha A^\top A\right)\mathbf{x}_k + \alpha A^\top \mathbf{b}. \end{aligned}$$

WLOG, assume $\mathbf{x}_0 = 0$, then

$$\begin{aligned} \mathbf{x}_1 &= \alpha A^\top \mathbf{b} \\ \mathbf{x}_2 &= \left(I - \alpha A^\top A\right)\mathbf{x}_1 + \alpha A^\top \mathbf{b} \\ &= \left(I - \alpha A^\top A\right)\left(\alpha A^\top \mathbf{b}\right) + \alpha A^\top \mathbf{b} \\ &= \alpha\left[I + \left(I - \alpha A^\top A\right)\right] A^\top \mathbf{b}. \\ \mathbf{x}_3 &= \left(I - \alpha A^\top A\right)\mathbf{x}_2 + \alpha A^\top \mathbf{b} \\ &= \left(I - \alpha A^\top A\right)\alpha\left[I + \left(I - \alpha A^\top A\right)\right] A^\top \mathbf{b} + \alpha A^\top \mathbf{b} \\ &= \alpha\left[I + \left(I - \alpha A^\top A\right) + (I - \alpha A^\top A)^2\right] A^\top \mathbf{b} \end{aligned}$$

In general,

$$\mathbf{x}_{k+1} = \alpha \sum_{j=0}^{k} \Big(I - \alpha A^\top A\Big)^j A^\top \mathbf{b}.$$

Now, use $A = U\Sigma V^\top$, we get

$$\begin{aligned}
\mathbf{x}_{k+1} &= \alpha \sum_{j=0}^{k} \Big(I - \alpha V\Sigma^\top \Sigma V^\top\Big)^j V\Sigma^\top U^\top \mathbf{b} \\
&= V \underbrace{\left(\alpha \sum_{j=0}^{k} \Big(I - \alpha \Sigma^\top \Sigma\Big)^j\right)}_{\text{diagonal}} \Sigma^\top U^\top \mathbf{b} \\
&= V D_k \Sigma^\top U^\top \mathbf{b}, \quad \text{where } D_k = \alpha \sum_{j=0}^{k} \Big(I - \alpha \Sigma^\top \Sigma\Big)^j \\
&= \sum_{i=1}^{n} d_i^{(k)} \sigma_i \Big(\mathbf{u}_i^\top \mathbf{b}\Big) \mathbf{v}_i,
\end{aligned}$$

where $d_i^{(k)} = \alpha \sum_{j=0}^{k} \left(1 - \alpha\sigma_i^2\right)^j$ is a geometric series.

Remark. For a geometric series, we know

$$\sum_{j=1}^{k} \alpha r^j = \frac{1 - r^{k+1}}{1 - r}$$

if $|r| < 1$.

In this case, we require $\left|1 - \alpha\sigma_i^2\right| < 1$ in order to have a convergent geometric series. We will choose α such that the inequality is satisfied. Therefore,

$$\begin{aligned}
d_i^{(k)} = \alpha \sum_{j=0}^{k} \left(1 - \alpha\sigma_i^2\right)^j &= \alpha \frac{1 - \left(1 - \alpha\sigma_i^2\right)^{k+1}}{1 - 1 + \alpha\sigma_i^2} \\
&= \cancel{\alpha} \frac{1 - \left(1 - \alpha\sigma_i^2\right)^{k+1}}{\cancel{\alpha}\sigma_i^2} \\
&= \frac{1 - (1 - \alpha\sigma_i^2)^{k+1}}{\sigma_i^2}.
\end{aligned}$$

Therefore, each iteration of Landweber can be written as

$$\mathbf{x}_{k+1} = \sum_{i=1}^{n} d_i^{(k)} \sigma_i \Big(\mathbf{u}_i^\top \mathbf{b}\Big) \mathbf{v}_i = \sum_{i=1}^{n} \frac{1 - \left(1 - \alpha\sigma_i^2\right)^{k+1}}{\sigma_i^{\cancel{2}}} \cancel{\sigma_i} \Big(\mathbf{u}_i^\top \mathbf{b}\Big) \mathbf{v}_i$$

Reorder terms, we get

$$\begin{aligned}\mathbf{x}_{k+1} &= \sum_{i=1}^{n} \underbrace{\left[1-\left(1-\alpha\sigma_i^2\right)^{k+1}\right]}_{\varphi_i^{(k+1)}} \frac{\left(\mathbf{u}_i^\top \mathbf{b}\right)}{\sigma_i}\mathbf{v}_i \\ &= \sum_{i=1}^{n} \varphi_i^{(k+1)} \frac{\mathbf{u}_i^\top \mathbf{b}}{\sigma_i}\mathbf{v}_i,\end{aligned}$$

where the *Landweber Filter Factors*, for iteration k, are

$$\varphi_i^{(k+1)} = 1-\left(1-\alpha\sigma_i^2\right)^{k+1}.$$

Notice: If $\alpha = 1$ and $\sigma_{\max} = \sigma_i = 1$, then

- $\varphi_1^{(k+1)} = 1-(1-1)^{k+1} = 1$
- $\varphi_n^{(k+1)} \approx 1-(1-0)^{k+1} \approx 0$
- If $k = 0$, then

$$\varphi_i^{(1)} = 1-\left(1-\sigma_i^2\right) = \sigma_i^2 \approx \begin{cases} 1 & \text{for large } \sigma_i's \\ 0 & \text{for small } \sigma_i's. \end{cases}$$

Thus, Landweber is an SVD filtering method.

Remark.

- If α_k is not constant, the method is steepest descent.
- For steepest descent (α_k not constant), we cannot get closed form expressions for $\varphi_i^{(k+1)}$. But as with LSQR, we can write

$$\varphi_i^{(k+1)} = \sigma_i \,\text{poly}\left(\sigma_i^2\right).$$

- Landweber and steepest descent converge much more slowly than Krylov subspace methods. This means semi-convergence is slower $\implies$ don't risk to a bad solution.
- We can precondition those methods, but care is needed.
  1. We want to reconstruct components of the solution corresponding to large σ_i's (i.e., cluster large σ_i's).
  2. But we don't want to reconstruct components of the solution corresponding to small σ_i's (i.e., don't cluster small σ_i's).

Landweber for Nonlinear Problems Consider a nonlinear inverse problem:

$$\mathbf{b} = F(\mathbf{x}_{\text{exact}}) + \boldsymbol{\eta},$$

where $F : \mathbb{R}^n \to \mathbb{R}^m$ is differentiable. As with the linear case, we can look at

$$\min_{\mathbf{x}} \|\mathbf{b} - F(\mathbf{x})\|_2^2.$$

Using Landweber, we get

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha F'(\mathbf{x}_k)^\top (\mathbf{b} - F(\mathbf{x}_k)),$$

where $F'(\mathbf{x})$ is the Jacobian of $F(\mathbf{x})$.

4.3.3 Hybrid Iterative Methods

Recall Tikhonov regularization for inverse problems computes

$$\mathbf{x}_\lambda = \arg\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda^2 \|\mathbf{x}\|_2^2,$$

or, equivalently,

$$\mathbf{x}_\lambda = \arg\min_{\mathbf{x}} \left\| \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} A \\ \lambda I \end{bmatrix} \mathbf{x} \right\|_2^2,$$

which could also be computed via the normal equations, by solving

$$\left(A^\top A + \lambda^2 I\right) \mathbf{x}_\lambda = A^\top \mathbf{b}.$$

We can use LSQR to solve the least squares problem.

- Since LSQR uses GKB to project on Krylov subspaces, we call this *regularize-then-project*.
- Challenge: What to use for λ?

An **alternative idea** is to start with trying to (iteratively) solve the LS problem

$$\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2^2.$$

That is, we use GKB to first project onto Krylov subspaces, and then enforce regularization.

Using Lower Bidiagonalization Approach $A \in \mathbb{R}^{m \times n}$, and the k steps of bidiagonalization computes

- $U_k = \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_k & \mathbf{u}_{k+1} \end{bmatrix} \in \mathbb{R}^{m \times (k+1)}$
- $V_k = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_k \end{bmatrix} \in \mathbb{R}^{n \times k}$
- $B_k = \begin{bmatrix} \alpha_1 & & \\ \beta_1 & \ddots & \\ & \ddots & \alpha_k \\ & & \beta_k \end{bmatrix} \in \mathbb{R}^{(k+1) \times k}$ such that

$$A^\top U_k = V_k B_k^\top + \alpha_{k+1} \mathbf{v}_{k+1} \mathbf{e}_{k+1}^\top \quad \text{and} \quad A V_k = U_k B_k.$$

We start the iteration with $\mathbf{u}_1 = \mathbf{b}/\|\mathbf{b}\|_2 = \mathbf{b}/\beta$, where $\beta = \|\mathbf{b}\|_2$.

Projected LS Problem At each iteration, we want to solve

$$\begin{aligned}\mathbf{x}_k = \arg\min_{\mathbf{x}\in R(V_k)} \|\mathbf{b} - A\mathbf{x}\|_2^2 &\iff \min_{\mathbf{y}\in\mathbb{R}^k} \left\|U_k^\top \mathbf{b} - B_k\mathbf{y}\right\|_2^2 \\ &\iff \arg\min_{\mathbf{y}\in\mathbb{R}^k} \|\beta\mathbf{e}_k - B_k\mathbf{y}\|_2^2,\end{aligned}$$

where $R(V_k) = \operatorname{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$ and $\mathbf{x}_k = V_k\mathbf{y}_k$.

Solving the Projected LS Problem

- If A is ill-conditioned, $\sigma_i \to 0$ fast. Then, B_k will have similar properties (provided k is large enough).
- Therefore, instead of solving

  $$\mathbf{y}_k = \arg\min_{\mathbf{y}\in\mathbb{R}^k} \|\beta\mathbf{e}_k - B_k\mathbf{y}\|_2^2,$$

  we could consider solving

  $$\begin{aligned}\mathbf{y}_k &= \arg\min_{\mathbf{y}\in\mathbb{R}^k} \|\beta\mathbf{e}_k - B_k\mathbf{y}\|_2^2 + \lambda_k^2\|\mathbf{y}\|_2^2 \\ &= \arg\min_{y\in\mathbb{R}^k} \left\| \begin{bmatrix}\beta\mathbf{e}_k \\ \mathbf{0}\end{bmatrix} - \begin{bmatrix}B_k \\ \lambda_k I\end{bmatrix}\mathbf{y}\right\|_2^2.\end{aligned}$$

  We call this *project-then-regularize.*
- We still have the question of how to choose λ_k (see the next subsection!)

Extension 4.1 *We can also do this for the LASSO regression, i.e.,*

$$\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2 + \lambda\|\mathbf{x}\|_1,$$

where the 1*-norm encourages sparsity. We will discuss in the future sections.*

4.3.4 Regularization Parameters

In this section, we finally get into different ways to estimate regularization parameters. Consider our standard linear inverse problem

$$\mathbf{b} = A\mathbf{x}_{\text{exact}} + \boldsymbol{\eta}$$

and suppose we want to compute a Tikhonov regularized solution

$$\mathbf{x}_\lambda = \arg\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda^2\|\mathbf{x}\|_2^2 = \arg\min_{\mathbf{x}} \left\| \begin{bmatrix}\mathbf{b} \\ \mathbf{0}\end{bmatrix} - \begin{bmatrix}A \\ \lambda I\end{bmatrix}\mathbf{x}\right\|_2^2$$

We can also write it in the normal equation form

$$\mathbf{x}_\lambda = \underbrace{\left(A^\top A + \lambda^2 I\right)^{-1} A^\top}_{=A_\lambda^\dagger} \mathbf{b}$$

Remark. There is no one fool proof method that is optimal for all problems. Methods should be consider guides.

Discrepancy Principle

- Requires knowing the norm of the noise: $\|\boldsymbol{\eta}\|_2$.
- Notice: $\mathbf{b} = A\mathbf{x}_{\text{exact}} + \boldsymbol{\eta}$.
  If $\mathbf{x}_\lambda$ is a good approximation of $\mathbf{x}_{\text{exact}}$, we might expect that $\|\mathbf{b} - A\mathbf{x}_\lambda\|_2 \approx \|\boldsymbol{\eta}\|_2$.
  Or,
  $$\|\mathbf{b} - A\mathbf{x}_\lambda\|_2^2 \approx \tau \|\boldsymbol{\eta}\|_2^2, \quad \text{where } \tau \gtrapprox 1.$$
  [*The notation $gtrapprox$ implies that τ is approximately equal to 1, but might be slightly bigger. It reads as "greater than about."*]
  $\implies D(\lambda) = \|\mathbf{b} - A\mathbf{x}_\lambda\|_2^2 - \tau\|\boldsymbol{\eta}\|_2^2 = 0$ $\quad \leftarrow$ A root finding problem.

Generalized Cross Validation (GCV)

- Do not need the norm of the noise.
- We skip details (tedious)
- Find λ to minimize
  $$G(\lambda) = \frac{n\|\mathbf{b} - A\mathbf{x}_\lambda\|_2^2}{\left[\text{tr}\left(I - AA_\lambda^\dagger\right)\right]^2},$$
  where $A_\lambda^\dagger = \left(A^\top A + \lambda^2 I\right)^{-1} A^\top$.

To implement these methods, we must first simplify the expressions. We will use SVD to simplify

$$\begin{aligned}
A &= U\Sigma V^\top \\
\mathbf{x}_\lambda &= \left(A^\top A + \lambda^2 I\right)^{-1} A^\top \mathbf{b} \\
&= V\left(\Sigma^\top \Sigma + \lambda^2 I\right)^{-1} \Sigma^\top U^\top \mathbf{b} \\
A\mathbf{x}_\lambda &= U\Sigma\left(\Sigma^\top \Sigma + \lambda^2 I\right)^{-1} \Sigma^\top U^\top \mathbf{b} \\
\mathbf{b} - A\mathbf{x}_\lambda &= UU^\top \mathbf{b} - U\Sigma\left(\Sigma^\top \Sigma + \lambda^2 I\right)^{-1} \Sigma^\top U^\top \mathbf{b} \\
&= U \underbrace{\left(I - \Sigma\left(\Sigma^\top \Sigma + \lambda^2 I\right)^{-1} \Sigma^\top\right)}_{\text{diagonal matrix}} U^\top \mathbf{b}
\end{aligned}$$

Note that

$$I - \Sigma\Big(\Sigma^\top\Sigma + \lambda^2 I\Big)^{-1}\Sigma^\top = \operatorname{diag}\left(1 - \frac{\sigma_i^2}{\sigma_i^2+\lambda^2}\right) = \operatorname{diag}\left(\frac{\lambda^2}{\sigma_i^2+\lambda^2}\right),$$

we have

$$\begin{aligned}
\|\mathbf{b} - A\mathbf{x}_\lambda\|_2^2 &= \left\| U \operatorname{diag}\left(\frac{\lambda^2}{\sigma_i^2+\lambda^2}\right) U^\top \mathbf{b} \right\|_2^2 \\
&= \left\| \operatorname{diag}\left(\frac{\lambda^2}{\sigma_i^2+\lambda^2}\right) \widehat{\mathbf{b}} \right\|_2^2 \qquad [\widehat{\mathbf{b}} = U^\top\mathbf{b} \;\&\; U \textit{ is orthogonal}] \\
&= \sum_{i=1}^n \left(\frac{\lambda^2 \widehat{b}_i}{\sigma_i^2+\lambda^2}\right)^2.
\end{aligned}$$

- DP is to find λ such that

$$\begin{aligned}
D(\lambda) = \|\mathbf{b} - A\mathbf{x}\|_2^2 - \tau\|\boldsymbol{\eta}\|_2^2 &= 0 \\
\sum_{i=1}^n \left(\frac{\lambda^2\widehat{b}_i}{\sigma_i^2+\lambda^2}\right)^2 - \tau\|\boldsymbol{\eta}\|_2^2 &= 0,
\end{aligned}$$

  where $\tau \gtrapprox 1$. This is a simple 1D root finding problem. We will use MATLAB's `fzero()` function.

- For GCV,

$$G(\lambda) = \frac{n\|\mathbf{b} - A\mathbf{x}_\lambda\|_2^2}{\left[\operatorname{tr}\left(I - AA_\lambda^\dagger\right)\right]^2} = \frac{n\displaystyle\sum_{i=1}^n \left(\frac{\lambda^2\widehat{b}_i}{\sigma_i^2+\lambda^2}\right)^2}{\left[\operatorname{tr}\left(I - AA_\lambda^\dagger\right)\right]^2}.$$

$$\begin{aligned}
A_\lambda^\dagger &= \Big(A^\top A + \lambda^2 I\Big)^{-1} A^\top = V\Big(\Sigma^\top\Sigma + \lambda^2 I\Big)^{-1}\Sigma^\top U^\top \\
AA_\lambda^\dagger &= U\Sigma\Big(\Sigma^\top\Sigma + \lambda^2 I\Big)^{-1}\Sigma^\top U^\top \\
I - AA_\lambda^\dagger &= I - U\Sigma\Big(\Sigma^\top\Sigma + \lambda^2 I\Big)^{-1}\Sigma^\top U^\top \\
&= U\underbrace{\left(I - \Sigma\Big(\Sigma^\top\Sigma + \lambda^2 I\Big)^{-1}\Sigma^\top\right)}_{\text{diagonal matrix}} U^\top \\
I - \Sigma\Big(\Sigma^\top\Sigma + \lambda^2 I\Big)^{-1}\Sigma^\top &= \operatorname{diag}\left(1 - \frac{\sigma_i^2}{\sigma_i^2+\lambda^2}\right) = \operatorname{diag}\left(\frac{\lambda^2}{\sigma_i^2+\lambda^2}\right) \\
\operatorname{tr}\left(I - AA_\lambda^\dagger\right) &= \operatorname{tr}\left(U \operatorname{diag}\left(\frac{\lambda^2}{\sigma_i^2+\lambda^2}\right) U^\top\right) \\
&= \operatorname{tr}\left(\operatorname{diag}\left(\frac{\lambda^2}{\sigma_i^2+\lambda^2}\right)\right) \\
&= \sum_{i=1}^n \frac{\lambda^2}{\sigma_i^2+\lambda^2}
\end{aligned}$$

Hence,

$$G(\lambda) = \frac{n\sum_{i=1}^{n}\left(\frac{\cancel{\lambda^2}\widehat{b}_i}{\sigma_i^2+\lambda^2}\right)^2}{\left(\sum_{i=1}^{n}\frac{\cancel{\lambda^2}}{\sigma_i^2+\lambda^2}\right)^2} = \frac{n\sum_{i=1}^{n}\left(\frac{\widehat{b}_i}{\sigma_i^2+\lambda^2}\right)^2}{\left(\sum_{i=1}^{n}\frac{1}{\sigma_i^2+\lambda^2}\right)^2}.$$

We can use, for example, MATLAB's `fminbnd` to solve it. The bound will be $[0, \sigma_1]$, where σ_1 is the largest singular value.

Remark. Objection!
Recall that A is often too large to compute its SVD. The formulas require us to have the full SVD computed, so it will not be feasible. But, we can use these formulas on the projected problems

$$\min_{\mathbf{y}\in\mathbb{R}^k} \|\beta\mathbf{e}_1 - B_k\mathbf{y}\|_2^2 + \lambda_k^2\|\mathbf{y}\|_2^2.$$

That is, at each iteration, use the SVD of B_k (a tiny matrix).

4.3.5 Other Regularization Approaches

In this section, we will see the LASSO regression:

$$\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1.$$

First notice: If $D_{\mathbf{x}} = \operatorname{diag}\left(\frac{1}{\sqrt{|x_i|}}\right)$, then

$$D_{\mathbf{x}}\mathbf{x} = \begin{bmatrix} x_1/\sqrt{|x_1|} \\ x_2/\sqrt{|x_2|} \\ \vdots \\ x_n/\sqrt{|x_n|} \end{bmatrix}.$$

So,

$$\begin{aligned}\|D_{\mathbf{x}}\mathbf{x}\|_2^2 &= \frac{x_1^2}{|x_1|} + \frac{x_2^2}{|x_2|} + \cdots + \frac{x_n^2}{|x_n|} \\ &= |x_1| + |x_2| + \cdots + |x_n| \\ &= \|\mathbf{x}\|_1.\end{aligned}$$

Can we just replaced $\|\mathbf{x}\|_1$ as $\|D_{\mathbf{x}}\mathbf{x}\|_2^2$? No.

Problems

- What if $x_i = 0$? Remedy: add a small entry to it. For example, $\frac{1}{\sqrt{|x_i|+\varepsilon}}$.

- $D_{\mathbf{x}}$ depends on $\mathbf{x}$, but we don't know $\mathbf{x}$. Remedy: Iterative update!

Algorithm 16: Iteratively Reweighted LS (Simple Idea)

begin
  $\mathbf{x}_0 =$initial guess;
  $D_k = \operatorname{diag}\left(1/\sqrt{\left|\mathbf{x}_0^{(i)}\right|}\right)$;
  for $k = 0, 1, 2, \ldots$ **do**
    $\mathbf{x}_{k+1} = \arg\min_{\mathbf{x}} \|\mathbf{b} - A\mathbf{x}\|_2^2 + \|D_k\mathbf{x}\|_2^2$;
    update $D_{k+1} = \operatorname{diag}\left(1/\sqrt{\left|\mathbf{x}_k^{(i)}\right|}\right)$

Remark. We are dividing by small numbers? Don't worry!
Let $\widehat{\mathbf{x}} = D\mathbf{x}$. So, $\mathbf{x} = D^{-1}\widehat{\mathbf{x}}$. Then,

$$\|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda\|D\mathbf{x}\|_2^2 = \left\|\mathbf{b} - AD^{-1}\widehat{\mathbf{x}}\right\|_2^2 + \lambda\|\mathbf{x}\|_2^2.$$

It is just a preconditioning method, and the preconditioner is

$$D^{-1} = \operatorname{diag}\left(\sqrt{|x_i|}\right).$$

We have no division at all.

Another Problem This requires solving a large LS problem at each iteration.

Workaround Consider

$$\min \|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda\|D_k\mathbf{x}\|_2^2 \tag{9}$$

If D_k is invertible, then we can rewrite (9) as

$$\min \left\|\mathbf{b} - AD_k^{-1}\widehat{\mathbf{x}}\right\|_2^2 + \lambda\|\mathbf{x}\|_2^2.$$

This looks like preconditioning. We can embed this into a Krylov solver, and use D_k as a preconditioner that changes at each iteration.

4.4 Flexible Methods

Non-Constant Preconditioners We know that preconditioning is often used in Krylov subspace methods. For example, to solve $A\mathbf{x} = \mathbf{b}$ with a right preconditioner M, we consider

$$AM^{-1}M\mathbf{x} = \mathbf{b} \iff \widehat{A}\widehat{\mathbf{x}} = \mathbf{b}.$$

Our discussion assumed that the preconditioner remained constant at each iteration. However, there are problems/applications where the preconditioner can change at each iteration. For example,

- M_j might depend on $\mathbf{x}_j$, or
- M_j is not (explicitly) a matrix, but instead there is a function to (inexactly or incompletely) solve linear systems with M_j.

GMRES Revisit To understand how to build non-constant preconditioners, we start from GMRES.

In GMRES, we use Arnoldi to build an orthonormal basis for the Krylov subspace:

$$\mathcal{K}_m(A, \mathbf{r}_0) = \text{span}\left\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{m-1}\mathbf{r}_0\right\}.$$

This gives

$$Q_{m+1} = \begin{bmatrix}\mathbf{q}_1 & \cdots & \mathbf{q}_m & \mathbf{q}_{m+1}\end{bmatrix}, \quad \mathbf{q}_1 = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|_2}, \quad \mathbf{r}_0 = \|\mathbf{r}_0\|_2\mathbf{q}_1$$
$$\text{upper Hessenberg} \quad \widetilde{H}_m \in \mathbb{R}^{(m+1)\times m} \; s.t. \; AQ_m = Q_{m+1}\widetilde{H}_m.$$

Algorithm 11 outlines the Basic GMRES Algorithm. If we use a preconditioner, Lines 4 and 14 will change.

Right Preconditioned GMRES We want to solve $A\mathbf{x} = \mathbf{b}$. Suppose $M \in \mathbb{R}^{n\times n}$, non-singular, is the preconditioner. Consider

$$AM^{-1}M\mathbf{x} = \mathbf{b},$$

or

$$\widehat{A}\widehat{\mathbf{x}} = \mathbf{b},$$

where $\widehat{A} = AM^{-1}$ and $\widehat{\mathbf{x}} = M\mathbf{x}$. Then, $\mathbf{x} = M^{-1}\widehat{\mathbf{x}}$.

Apply Arnoldi to the preconditioned system:

- Given $\mathbf{x}_0$, we get $\widehat{\mathbf{x}}_0 = M\mathbf{x}_0$.
- Notice that
$$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 = \mathbf{b} - AM^{-1}M\mathbf{x}_0 = \mathbf{b} - \widehat{A}\widehat{\mathbf{x}}_0 = \widehat{\mathbf{r}}_0.$$
Therefore,
$$\mathbf{q}_1 = \frac{\widehat{\mathbf{r}}_0}{\|\widehat{\mathbf{r}}_0\|_2} = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|_2}.$$
- Arnoldi builds an orthonormal basis for
$$\text{span}\left\{\widehat{\mathbf{r}}_0, \widehat{A}\widehat{\mathbf{r}}_0, \ldots, \widehat{A}^{m-1}\widehat{\mathbf{r}}_0\right\} = \text{span}\left\{\mathbf{r}_0, AM^{-1}\mathbf{r}_0, \ldots, \left(AM^{-1}\right)^{m-1}\mathbf{r}_0\right\}.$$
Again, this gives $Q_{m+1} = \begin{bmatrix}\mathbf{q}_1 & \cdots & \mathbf{q}_m & \mathbf{q}_{m+1}\end{bmatrix}$ and an upper Hessenberg matrix $\widetilde{H}_m$ such

that

$$\widehat{A}Q_m = Q_{m+1}\widetilde{H}_m \quad \text{or} \quad A\underbrace{M^{-1}Q_m}_{=Z_m} = Q_{m+1}\widetilde{H}_m \iff AZ_m = Q_{m+1}\widetilde{H}_m,$$

where

$$Z_m = \begin{bmatrix}\mathbf{z}_1 & \mathbf{z}_2 & \cdots & \mathbf{z}_m\end{bmatrix} = \begin{bmatrix}M^{-1}\mathbf{q}_1 & M^{-1}\mathbf{q}_2 & \cdots & M^{-1}\mathbf{q}_m\end{bmatrix}.$$

So, we will change $\mathbf{w}_j = A\mathbf{q}_j$ (Line 4 in Algorithm 11) into

$$\boxed{\mathbf{w}_j = AM^{-1}\mathbf{q}_j}.$$

Now, let's consider the optimality condition (minimizing residual). For $\widehat{A}\widehat{\mathbf{x}} = \mathbf{b}$, we compute

$$\widehat{\mathbf{x}}_m = \widehat{\mathbf{x}}_0 + Q_m\mathbf{y}, \quad \mathbf{y} \in \mathcal{K}_m\left(\widehat{A}, \widehat{\mathbf{r}}_0\right).$$

The residual is

$$\begin{aligned}
\widehat{\mathbf{r}}_m = \mathbf{b} - \widehat{A}\widehat{\mathbf{x}}_m &= \mathbf{b} - \widehat{A}(\widehat{\mathbf{x}}_0 + Q_m\mathbf{y}) \\
&= \underbrace{\mathbf{b} - \widehat{A}\widehat{\mathbf{x}}_0} - \widehat{A}Q_m\mathbf{y} \\
&= \widehat{\mathbf{r}}_0 - \widehat{A}Q_m\mathbf{y} \\
&= \mathbf{r}_0 - Q_{m+1}\widetilde{H}_m\mathbf{y} && [\widehat{A}Q_m = Q_{m+1}\widetilde{H}_m] \\
&= \beta\mathbf{q}_1 - Q_{m+1}\widetilde{H}_m\mathbf{y} && [\mathbf{r}_0 = \|\mathbf{r}_0\|_2\mathbf{q}_1 = \beta\mathbf{q}_1, \textit{ where } \beta = \|\mathbf{r}_0\|_2]
\end{aligned}$$

For GMRES, we compute

$$\begin{aligned}
\mathbf{y}_m &= \arg\min_{\mathbf{y}} \left\|\beta\mathbf{q}_1 - Q_{m+1}\widetilde{H}_m\mathbf{y}\right\|_2 \\
&= \arg\min_{\mathbf{y}} \left\|Q_{m+1}\left(\beta\mathbf{e}_1 - \widetilde{H}_m\mathbf{y}\right)\right\|_2 \\
&= \arg\min_{\mathbf{y}} \left\|\beta\mathbf{e}_1 - \widetilde{H}_m\mathbf{y}\right\|_2.
\end{aligned}$$

Once we get $\mathbf{y}_m$, we update $\widehat{\mathbf{x}}_0$:

$$\begin{aligned}
\widehat{\mathbf{x}}_m &= \widehat{\mathbf{x}}_0 + Q_m\mathbf{y}_m \\
M\mathbf{x}_m &= M\mathbf{x}_0 + Q_m\mathbf{y} && [\widehat{\mathbf{x}} = M\mathbf{x}] \\
\mathbf{x}_m &= \mathbf{x}_0 + M^{-1}Q_m\mathbf{y}_m && [M \textit{ is nonsingular}]
\end{aligned}$$

So, we change $\mathbf{x}_m = \mathbf{x}_m + Q_m\mathbf{y}_m$ (Line 14 in Algorithm 11) to

$$\boxed{\mathbf{x}_m = \mathbf{x}_0 + M^{-1}Q_m\mathbf{y}_m}.$$

We could also write

$$\mathbf{x}_m = \mathbf{x}_0 + Z_m\mathbf{y}_m, \quad \text{where } Z_m = M^{-1}Q_m.$$

Algorithm 17: Right Preconditioned GMRES

Input: $A \in \mathbb{R}^{n\times n}$, $M \in \mathbb{R}^{n\times n}$, $\mathbf{b}$, and $\mathbf{x}_0 \in \mathbb{R}^n$

begin
- Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|_2$, and $\mathbf{q}_1 = \mathbf{r}_0/\beta$;
- **for** $j = 1 : m$ **do**
  - $\mathbf{w}_j = A(M^{-1}\mathbf{q}_j)$;
  - **for** $i = 1 : j$ **do**
    - $h_{ij} = \mathbf{q}_i^\top \mathbf{w}_j$;
    - $\mathbf{w}_j = \mathbf{w}_j - h_{ij}\mathbf{q}_i$;
  - $h_{j+1,j} = \|\mathbf{w}_j\|_2$;
  - **if** $h_{j+1,j} = 0$ **then**
    - $m = j$;
    - break;
  - $\mathbf{q}_{j+1} = \mathbf{w}_j/h_{j+1,j}$;
- $\mathbf{y}_m = \arg\min_{\mathbf{y}} \left\| \beta]\mathbf{e}_1 - \widetilde{H}_m\mathbf{y} \right\|_2$;
- $\mathbf{x}_m = \mathbf{x}_0 + M^{-1}(Q_m\mathbf{y}_m)$;

Remark.

- We should compute $\mathbf{w}_j = A(M^{-1}\mathbf{q}_j)$ in two steps:

$$\mathbf{z}_j = M^{-1}\mathbf{q}_j \iff \text{solve } M\mathbf{z}_j = \mathbf{q}_j$$
$$\mathbf{w}_j = A\mathbf{z}_j$$

  It avoids matrix-matrix multiplications. Also, we do not need to store $\mathbf{z}_j$'s,

- We should compute $M^{-1}(Q_m\mathbf{y}_m)$ in two steps:

$$\widehat{\mathbf{z}} = Q_m\mathbf{y}_m$$
$$\mathbf{z} = M^{-1}\widehat{\mathbf{z}} \iff \text{solve } M\mathbf{z} = \widehat{\mathbf{z}}.$$

Non-constant Preconditioners Now, suppose M_j is a preconditioner that changes at each iteration. In this case, the Arnoldi relation still holds

$$AZ_m = Q_{m+1}\widetilde{H}_m,$$

where $Z_m = \begin{bmatrix} M_1^{-1}\mathbf{q}_1 & M_2^{-1}\mathbf{q}_2 & \cdots & M_m^{-1}\mathbf{q}_m \end{bmatrix}$.

We cannot write $Z_m = M_j^{-1}Q_m$ in this case, but that's okay. We just need to save all of the $\mathbf{z}_j$ vectors. We update the solution as

$$\boxed{\mathbf{x}_m = \mathbf{x}_0 + Z_m\mathbf{y}_m}$$

Algorithm 18: Flexible GMRES (FGMRES)

Input: A, solver for M_j, $\mathbf{b}$, and $\mathbf{x}_0$

begin
  Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\beta = \|\mathbf{r}_0\|_2$, and $\mathbf{q}_1 = \mathbf{r}_0/\beta$;
  for $j = 1 : m$ **do**
    $\mathbf{z}_j = M_j^{-1}\mathbf{q}_j$ and $\mathbf{w}_j = A\mathbf{z}_j$;
    for $i = 1 : j$ **do**
      $h_{ij} = \mathbf{q}_i^\top \mathbf{w}_j$;
      $\mathbf{w}_j = \mathbf{w}_j - h_{ij}\mathbf{q}_i$;
    $h_{j+1,j} = \|\mathbf{w}_j\|_2$;
    if $h_{j+1,j} = 0$ **then**
      $m = j$;
      break;
    $\mathbf{q}_{j+1} = \mathbf{w}_j/h_{j+1,j}$;
  $\mathbf{y}_m = \arg\min_{\mathbf{y}} \left\|\beta\mathbf{e}_1 - \widetilde{H}_m\mathbf{y}\right\|_2$;
  $\mathbf{x}_m = \mathbf{x}_0 + Z_m\mathbf{y}_m$;

Remark. It is easy to see that if M_j is constant, FGMRES is GMRES.

Preconditioned CG and Flexible CG also follow a similar scheme.

Algorithm 19: Preconditioned CG

Input: $A, \mathbf{b}, \mathbf{x}_0$, and SPD M

begin
  Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$;
  Solve $M\mathbf{z}_0 = \mathbf{r}_0$;
  $\mathbf{p}_0 = \mathbf{z}_0$;
  for $k = 0, 1, 2, \ldots$ **do**
    $\mathbf{w} = A\mathbf{p}_k$;
    $\alpha_k = (\mathbf{z}_k^\top \mathbf{r}_k)/(\mathbf{p}_k^\top \mathbf{w})$;
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$;
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{w}$;
    Solve $M\mathbf{z}_{k+1} = \mathbf{r}_{k+1}$;
    $\beta_k = (\mathbf{z}_{k+1}^\top \mathbf{r}_{k+1})/(\mathbf{z}_k^\top \mathbf{r}_k)$ `// Fletcher-Reeves Formula`
    $\beta_{k+1} = \mathbf{z}_{k+1} + \beta_k\mathbf{p}_k$;

Algorithm 20: Flexible Preconditioned CG

Input: $A, \mathbf{b}, \mathbf{x}_0$, and solver for SPD M_k

begin
  Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$;
  Solve $M_0\mathbf{z}_0 = \mathbf{r}_0$;
  $\mathbf{p}_0 = \mathbf{z}_0$;
  for $k = 0, 1, 2, \ldots$ **do**
    $\mathbf{w} = A\mathbf{p}_k$;
    $\alpha_k = \left(\mathbf{z}_k^\top \mathbf{r}_k\right) / \left(\mathbf{p}_k^\top \mathbf{w}\right)$;
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{w}$;
    Solve $M_k\mathbf{z}_{k+1} = \mathbf{r}_{k+1}$;
    $\beta_k = \left(\mathbf{z}_{k+1}^\top (\mathbf{r}_{k+1} - \mathbf{r}_k)\right) / \left(\mathbf{z}_k^\top \mathbf{r}_k\right)$ // Polak-Ribiére Formula (This is basically a reorthogonalization step)
    $\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k$;

5 Fast Fourier Transforms (FFT)

5.1 Integral Equations

$$b(s) = \int a(s,t)x(t)\,\mathrm{d}t$$

Forward Problem Given $a(s,t)$ and $x(t)$, compute $b(s)$.

Backward Problem Given $a(s,t)$ and $b(s)$, find $x(t)$.

> **Remark.** The interval of integration could be finite (e.g., $[0,1]$) or infinte (i.e., $(-\infty,\infty)$).

Discretize to Get a Linear Algebra Problem

- Use quadrature rule for integration: with notes $t_1, t_2, \ldots, t_n$ and weights $w_1, w_2, \ldots, w_n$.
- Sample $b(s)$ at $s_1, s_2, \ldots, s_n$.

This leads to

$$b(s_i) \approx \sum_{j=1}^{n} a(s_i,t_j)x(t_j)w_j.$$

Define a matrix

$$A = \Big[a(s_i,b_j)w_j\Big]_{i,j=1}^{n}$$

and vectors

$$\mathbf{x} = \Big[x(t_j)\Big]_{j=1}^{n} \quad \text{and} \quad \mathbf{b} = \Big[b(s_i)\Big]_{i=1}^{n}.$$

Then, we get a linear system

$$\mathbf{b} \approx A\mathbf{x},$$

or, the inverse problem

$$\mathbf{b} = A\mathbf{x}_{\text{exact}} + \boldsymbol{\eta}.$$

Convolution and Deconvolution This is an integral equation where $a(s,t)$ can be written as $a(s-t)$:

$$b(s) = \int a(s-t)x(t)\,\mathrm{d}t.$$

- Convolution: Given $a(s-t)$ and $x(t)$, compute $b(s)$.
- Deconvolution: Given $a(s-t)$ and $b(s)$, compute $x(t)$.

Example 5.1.1 Gaussian Kernel

$$a(s,t) = a(s-t) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(s-t)^2/(2\sigma^2)}$$

In discretization: Assume $w_j = w$ is a constant and equally spaced points. Then, each entry of matrix A is

$$a_{ij} = a(s_i - t_j)w \equiv a_{i-j}.$$

In this case:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \longrightarrow A = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdots \\ a_1 & a_0 & a_{-1} & \cdots \\ a_2 & a_1 & a_0 & \ddots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

This is called a *Toeplitz matrix*.

This implicitly assumes that $x(t)$ is zero outside the interval of convolution $\implies$ Zero Boundary conditions:

$$x(t_{-1}) = x(t_0) = 0, \underbrace{x(t_1) = x_1,\ x(t_2) = x_2, \ldots, x(t_{n-1}) = x_{n-1},\ x(t_n) = x_n}_{\text{In discretization}}, x(t_{n+1}) = x(t_{n+2}) = 0$$

Other BCs:

- Periodic BC: the function repeats itself.

$$x(t_{-1}) = x_{n-1},\ x(t_0) = x_n, \underbrace{x(t_1) = x_1,\ x(t_2) = x_2, \ldots, x(t_n) = x_n}_{\text{In discretization}}, x(t_{n+1}) = x_1,\ x(t_{n+2}) = x_2$$

- Neumann/Reflective: Mirror relfection of what's inside the interval:

$$x(t_{-1}) = x_2,\ x(t_0) = x_1, \underbrace{x(t_1) = x_1,\ x(t_2) = x_2, \ldots, x(t_n) = x_n}_{\text{In discretization}}, x(t_{n+1}) = x_n,\ x(t_{n+2}) = x_{n-1}$$

Different BCs lead to different structured matrices:

- Zero BC $\implies$ A is Toeplitz
- Periodic BC $\implies$ A is circulant
- Reflective BC $\implies$ A is Toeplitz + Honkel.

2D Convolution/Deconvolution Problems Images

We get block structured matrices. For example, with zero BCs, we have

$$A = \begin{bmatrix} A_0 & A_{-1} & A_{-2} & \cdots \\ A_1 & A_0 & A_{-1} & \cdots \\ A_2 & A_1 & A_0 & \ddots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \quad \text{where } A_k = \begin{bmatrix} a_0^{(k)} & a_{-1}^{(k)} & a_{-2}^{(k)} & \cdots \\ a_1^{(k)} & a_0^{(k)} & a_{-1}^{(k)} & \cdots \\ a_2^{(k)} & a_1^{(k)} & a_0^{(k)} & \ddots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Remark. Computations with these matrices use Fast Fourier Transforms.

5.2 Fast Fourier Transforms

5.2.1 Discrete Fourier Transform (DFT) Matrix

Let $\omega_n = e^{2\pi \mathrm{i}/n}$, where $\mathrm{i} = \sqrt{-1}$. Then, the unitary DFT matrix of dimension n is

$$F_n = \frac{1}{\sqrt{n}} \left[\omega_n^{-jk}\right]_{j,k=0}^{n-1}$$
$$= \frac{1}{\sqrt{n}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \cdots & \omega_n^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \cdots & \omega_n^{-(n-1)(n-1)} \end{bmatrix}$$

Observations

- F_n is symmetric, $F_n^\top = F_n$. But it is not Hermitian symmetric. That is, $F_n^H = \overline{F}_n^\top \neq F_n$.
- F_n is unitary. That is,
$$F_n^H F_n = I \iff F_n^{-1} = F_n^H.$$
- Recall:

$$\begin{aligned} e^{\mathrm{i}\theta} &= \cos\theta + \mathrm{i}\sin\theta \\ \implies e^{-\mathrm{i}\theta} &= \cos(-\theta) + \mathrm{i}\sin(-\theta) \\ &= \cos\theta - \mathrm{i}\sin\theta \\ &= \overline{e^{\mathrm{i}\theta}} \\ \implies \overline{\omega_n^{-jk}} &= \omega_n^{jk} \end{aligned}$$

So,

$$F_n^H = \frac{1}{\sqrt{n}} \left[\omega_n^{jk}\right]_{j,k=0}^{n-1}$$

Remark. Given an $n \times n$ matrix A, computing

$$\mathbf{y} = A\mathbf{x}$$

requires $\mathcal{O}(n^2)$ `flops`. For F_n and F_n^H, we can reduce this to $\mathcal{O}(n \log n)$ if $n = 2^p$ (We can also do $2^{p_2} \cdot 3^{p_3} \cdot 5^{p_5} \cdots$) For example, $n = 256 = 2^8$. Then, $n^2 = 65,536$ but $n \log n = 2,048$.

Problem The constants behind the $\mathcal{O}(\cdot)$ notation. But it's modest in this case.

5.2.2 Idea of FFTs: Divide and Conquer

Suppose $n = 2m$, and we want to compute

$$\mathbf{y} = F_n^H \mathbf{x},$$

where

$$F_n^H = \left[\frac{1}{\sqrt{n}}\omega_n^{jk}\right]_{j,k=0}^{n-1}, \quad \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Let E_n be a permutation matrix such that

$$E_n^\top \mathbf{x} = \begin{bmatrix} \mathbf{x}_e \\ \mathbf{x}_o \end{bmatrix}, \quad \text{where } \mathbf{x}_e = \begin{bmatrix} x_0 \\ x_2 \\ \vdots \\ x_{2(m-1)} \end{bmatrix} (\text{even entries}), \quad \mathbf{x}_o = \begin{bmatrix} x_1 \\ x_3 \\ \vdots \\ x_{2m-1} \end{bmatrix} (\text{odd entries}).$$

Since E_n is a permutation matrix, $E_n E_n^\top = I$. Therefore,

$$\begin{aligned} \mathbf{y} = F_n^H \mathbf{x} &= F_n^H E_n E_n^\top \mathbf{x} \\ &= \left(F_n^H E_n\right)\left(E_n^\top \mathbf{x}\right). \end{aligned}$$

Multiplying a permutation on the right permutes the columns, so

$$F_n^H E_n = \begin{bmatrix} \left[\omega_n^{j(2k)}\right]_{j,k=0}^{m-1} & \left[\omega_n^{j(2k+1)}\right]_{j,k=0}^{m-1} \\ \left[\omega_n^{(j+m)(2k)}\right]_{j,k=0}^{m-1} & \left[\omega_n^{(j+m)(2k+1)}\right]_{j,k=0}^{m-1} \end{bmatrix}.$$

Since $n = 2m$, it follows that

- $\omega_n^{j(2k)} = e^{\frac{2\pi \mathrm{i}}{n} j(2k)} = e^{\frac{2\pi \mathrm{i}}{2m} j(2k)} = e^{\frac{2\pi \mathrm{i}}{m} jk} = \omega_m^{jk}$
- $\omega_n^{(j+m)(2k)} = \omega_n^{j(2k)} \omega_n^{m(2k)} = \omega_m^{jk} \cdot e^{\frac{2\pi \mathrm{i}}{n} m(2k)} = \omega_n^{jk} \cdot e^{2\pi \mathrm{i} k}$. Note that

  $$e^{2\pi \mathrm{i} k} = \cos(2\pi k) + \mathrm{i}\sin(2\pi k) = 1,$$

  we have $\omega_n^{(j+m)(2k)} = \omega_m^{jk}$.
- $\omega_n^{j(2k+1)} = \omega_n^{j(2k)} \omega_n^j = \omega_m^{jk} \omega_n^j$
- $\omega_n^{(j+1)(2k+1)} = \cdots = -\omega_m^{jk} \omega_n^j$.

Hence,

$$F_n^H E_n = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{\sqrt{m}} \left[\omega_m^{jk}\right]_{j,k=0}^{m-1} & \frac{1}{\sqrt{m}} \left[\omega_m^{jk}\omega_n^{j}\right]_{j,k=0}^{m-1} \\ \frac{1}{\sqrt{m}} \left[\omega_m^{jk}\right]_{j,k=0}^{m-1} & \frac{1}{\sqrt{m}} \left[-\omega_m^{jk}\omega_n^{j}\right]_{j,k=0}^{m-1} \end{bmatrix} \qquad \left[n = 2m \implies \frac{1}{\sqrt{n}} = \frac{1}{\sqrt{2m}} = \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{m}}\right]$$
$$= \frac{1}{\sqrt{2}} \begin{bmatrix} F_m^H & D_m F_m^H \\ F_m^H & -D_m F_m^H \end{bmatrix},$$

where $D_m = \operatorname{diag}\left(\omega_n^j\right)_{j=0}^{m-1}$. Thus,

$$\begin{aligned} \mathbf{y} = F_n^H \mathbf{x} &= F_n^H E_n E_n^\top \mathbf{x} \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} F_m^H & D_m F_m^H \\ F_m^H & -D_m F_m^H \end{bmatrix} \begin{bmatrix} \mathbf{x}_e \\ \mathbf{x}_o \end{bmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} F_m^H \mathbf{x}_e + D_m F_m^H \mathbf{x}_o \\ F_m^H \mathbf{x}_e - D_m F_m^H \mathbf{x}_o \end{bmatrix}. \end{aligned}$$

Therefore, we reduced the n-dimensional DFT to two m-dimensional DFTs.

- If $m = 2\ell$, do it again.
- If $n = 2^p$, we can continue in this way $p = \log_2(n)$ times.

Definition 5.2.1 (Forward and Inverse FFTs).

- Forward FFT:
$$\mathbf{y} = F_m \mathbf{x}$$
- Inverse FFT
$$\mathbf{x} = F_n^H \mathbf{y} = F_n^H (F_n \mathbf{x}) = \mathbf{x}.$$

Remark. Most FFT software uses an alternative scaling

$$\begin{aligned} F_n^H F_n &= \underbrace{\left[\frac{1}{\sqrt{n}}\omega_n^{jk}\right]}_{\text{inverse FFT}} \underbrace{\left[\frac{1}{\sqrt{n}}\omega_n^{-jk}\right]}_{\text{forward FFT}} \\ &= \underbrace{\left[\frac{1}{n}\omega_n^{jk}\right]}_{\text{inverse FFT}} \underbrace{\left[\omega_n^{-jk}\right]}_{\text{forward FFT}} \end{aligned}$$

For example, $\mathbf{y} = F\mathbf{x}$ (unitary version) can be computed in MATLAB as $\mathbf{y} = \frac{1}{\sqrt{n}}\texttt{fft}(\mathbf{x})$ and $\mathbf{z} = F^H \mathbf{w}$ in MATLAB is $\mathbf{z} = \sqrt{n}\texttt{ifft}(\mathbf{w})$.

5.2.3 Higher Dimensions

- 2D FFT:

$$F_{2D} = F_{1D} \otimes F_{1D}, \quad \text{where } F_{1D} = F_n \text{ and } \otimes \text{ is the Kronecker product}$$

Recall that $(A \otimes B)\mathbf{x} = BXA^\top$. So,

$$\begin{aligned} \mathbf{y} = F_{2D}\mathbf{x} &= (F_{1D} \otimes F_{1D})\mathbf{x} \\ Y &= F_{1D}XF_{1D}, \qquad [F_{1D}^\top = F_{1D}, \textit{symmetric}] \end{aligned}$$

where $\mathbf{x} = \mathrm{vec}(X)$ and $\mathbf{y} = \mathrm{vec}(Y)$. In MATLAB, we will use `reshape()` to achieve this.

1. 2D FFT software works on arrays. That is,

$$Y = \texttt{fft2}(X).$$

2. Cost of 2D FFT:

$$\mathcal{O}(n^2 \log n^2) = \mathcal{O}(n^2 \log n) \quad [\log n^2 = 2 \log n]$$

- 3D FFT:

$$F_{3D} = F_{1D} \otimes F_{1D} \otimes F_{1D}.$$

5.3 Toeplitz and Circulant Matrices

Definition 5.3.1 (Toeplitz Matrix). Toeplitz matrices have constant diagonals:

$$T = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & t_{-3} \\ t_1 & t_0 & t_{-1} & t_{-2} \\ t_2 & t_1 & t_0 & t_{-1} \\ t_3 & t_2 & t_1 & t_0 \end{bmatrix}.$$

T is structurally sparse. It is dense, but storage is sparse. We only need to store $2n-1$ entries.

Definition 5.3.2 (Circulant Matrix). Circulant matrices are Toeplitz matrices where each column (row) is a circular shift of its previous column (row):

$$C = \begin{bmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{bmatrix}$$

Circulant matrix is also structurally sparse. We only need to store n entries.

5.3.1 Eigenvalues and Eigenvectors

Theorem 5.3.3
Every $n \times n$ circulant matrix has the same set of eigenvectors. Specifically, any circulant matrix can be written as

$$C = F^H \Lambda F,$$

where F is the unitary FFT matrix.

$$F_{1\mathrm{D}} = F_n \quad \text{and} \quad \Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n).$$

Notice that $C = F^H \Lambda F$. So, multiply by F, we get $FC = \Lambda F$, where

$$F = \frac{1}{\sqrt{n}} \begin{bmatrix} 1 & 1 & 1 & \cdots \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots \\ 1 & \omega_n^{-2} & \ddots & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

Then,

$$FC\mathbf{e}_1 = \Lambda F \mathbf{e}_1.$$

Hence,

$$F\mathbf{c}_1 = \Lambda\left(\frac{1}{\sqrt{n}}\mathbf{1}\right) = \frac{1}{\sqrt{n}}\Lambda\mathbf{1} \qquad \mathbf{1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}, \quad \text{and} \quad \mathbf{c}_1 \text{ is the first column of } C$$

$$= \frac{1}{\sqrt{n}} \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

$$= \frac{1}{\sqrt{n}} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix}.$$

So, $\sqrt{n}F\mathbf{c}_1 = \boldsymbol{\lambda}$, where $\boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix}$. Recall that in MATLAB, if F represents the unitary FFT, then

$$\mathbf{y} = F\mathbf{x} \iff \mathbf{y} = \frac{1}{\sqrt{n}}\texttt{fft}(\mathbf{x}).$$

Therefore,

$$\boxed{\texttt{fft}(\mathbf{c}_1) = \boldsymbol{\lambda}}.$$

Using FFT, $\boldsymbol{\lambda} = \texttt{fft}(\mathbf{c}_1)$ costs $\mathcal{O}(n \log n)$. Compared to standard eigenvalue solvers, for den matrices, which costs $\mathcal{O}(n^3)$, this is incredibly fast!

5.3.2 Matrix-Vector Multiplication and Solving Linear Systems

Circulant Matrix-Vector Multiplication

$$\begin{aligned}\mathbf{z} = C\mathbf{r} &= F^H \Lambda F \mathbf{r} \\ &= \cancel{\sqrt{n}}\texttt{ifft}\left(\texttt{fft}(\mathbf{c}_1) .* \frac{1}{\cancel{\sqrt{n}}}\texttt{fft}(\mathbf{r})\right) \\ &= \texttt{ifft}\big(\texttt{fft}(\mathbf{c}_1) .* \texttt{fft}(\mathbf{r})\big)\end{aligned}$$

- The cost is $\mathcal{O}(n \log n)$.
- If we are multiplying C times many vectors, we should only compute $\texttt{fft}(\mathbf{c}_1)$ once. i.e., store the result.

Solving Circulant Systems

$$\begin{aligned} C\mathbf{z} = \mathbf{r} &\iff \mathbf{z} = C^{-1}\mathbf{r} = \left(F^H \Lambda F\right)^{-1}\mathbf{r} = F^H \Lambda^{-1} F \mathbf{r} \\ &\iff \mathbf{z} = \texttt{ifft}\big(\texttt{fft}(\mathbf{r}) ./ \texttt{fft}(\mathbf{c}_1)\big)\end{aligned}$$

- The cost is $\mathcal{O}(n \log n)$.
- A typical linear solver for dense matrices cost $\mathcal{O}(n^3)$.
- If we need to solve many systems with the same matrix C, we should precompute $\texttt{fft}(\mathbf{c}_1)$ just once. [*Or, $1 ./ \texttt{fft}(\mathbf{c}_1)$ because division are more expensive than multiplication. Usually, to compute division on a computer, it is a root finding problem: We want to find $x = \frac{1}{a}$, i.e., solve $ax = 1$. We will apply Newton's method to $ax - 1 = 0$ to find the root.*]

Toeplitz Matrix-Vector Multiplication To solve $\mathbf{z} = T\mathbf{r}$, we use ideas from circulant matrices by embedding.

$$T = \begin{bmatrix} t_0 & t_{-1} & t_{-1} \\ t_1 & t_0 & t_{-1} \\ t_2 & t_1 & t_0 \end{bmatrix} \implies C = \left[\begin{array}{ccc|ccc} t_0 & t_{-1} & t_{-2} & 0 & t_2 & t_1 \\ t_1 & t_0 & t_{-1} & t_{-2} & 0 & t_2 \\ t_2 & t_1 & t_0 & t_{-1} & t_{-2} & 0 \\ \hline 0 & t_2 & t_1 & t_0 & t_{-1} & t_{-2} \\ t_{-2} & 0 & t_2 & t_1 & t_0 & t_{-1} \\ t_{-1} & t_{-2} & 0 & t_2 & t_1 & t_0 \end{array}\right]$$

In general, if T is an $n \times n$ Toeplitz matrix, we will embed it into a $(2n) \times (2n)$ circulant matrix:

$$C = \begin{bmatrix} T & T_1 \\ T_2 & T \end{bmatrix}.$$

Then,

$$C\begin{bmatrix}\mathbf{r}\\ \mathbf{0}\end{bmatrix} = \begin{bmatrix}T & T_1\\ T_2 & T\end{bmatrix}\begin{bmatrix}\mathbf{r}\\ \mathbf{0}\end{bmatrix} = \begin{bmatrix}T\mathbf{r}\\ *\end{bmatrix}.$$

Therefore, to compute $\mathbf{z} = T\mathbf{r}$:

1. Use the first row and column of T to build the first column of C, $\mathbf{c}$.
2. Set $\widetilde{\mathbf{r}} = \begin{bmatrix}\mathbf{r}\\ \mathbf{0}\end{bmatrix}$.
3. Compute $\widetilde{\mathbf{z}} = \texttt{ifft}\big(\texttt{fft}(\widetilde{\mathbf{r}})\texttt{.*fft}(\mathbf{c})\big)$
4. $\mathbf{z} = \widetilde{\mathbf{z}}(1:n)$.

The cost is $\mathcal{O}((2n)\log(2n)) = \mathcal{O}(n\log(2n)) = \mathcal{O}(n(\log 2 + \log n)) = \mathcal{O}(n\log n)$.

5.3.3 Preconditioning Toeplitz Matrices

Suppose we have a banded Toeplitz matrix

$$T = \begin{bmatrix}t_0 & t_{-1} & t_{-2} & 0 & 0\\ t_1 & t_0 & t_{-1} & t_{-2} & 0\\ t_2 & t_1 & t_0 & t_{-1} & t_{-2}\\ 0 & t_2 & t_1 & t_0 & t_{-1}\\ 0 & 0 & t_2 & t_1 & t_0\end{bmatrix}$$

To precondition iterative solver to $T\mathbf{x} = b$, we will find M as a circulant approximation of T.

That is, $M = \arg\min_C \|T - C\|$.

- For $\|\cdot\|_1$ and $\|\cdot\|_\infty$, the best M is obtained by filling in the corners:

$$M = \begin{bmatrix}t_0 & t_{-1} & t_{-2} & t_2 & t_1\\ t_1 & t_0 & t_{-1} & t_{-2} & t_2\\ t_2 & t_1 & t_0 & t_{-1} & t_{-2}\\ t_{-2} & t_2 & t_1 & t_0 & t_{-1}\\ t_{-1} & t_{-2} & t_2 & t_1 & t_0\end{bmatrix}$$

  That is, $M = T + E$, where E is a sparse, low-rank matrix with non-zero entries in the corners.

- For $\|\cdot\|_F$, the first column of M is

$$m_j = \frac{(n-j)t_j + jt_{j-n}}{n}, \quad j = 0, 1, \ldots, n-1.$$

- For $\|\cdot\|_2$, it is complicated.

Question How good are these approximations/preconditioners?

If the entries of T decay quickly away from the diagonal, then they are very good.

A Applications of Iterative Methods

A.1 Radioactive Imaging and Iterative Methods

- Consider $A\mathbf{x} = \mathbf{b}$, $A \in \mathbb{R}^{m\times n}$ with $m > n$. Let $\mathbf{b}$ be random, and

$$\mathbf{b}_i = \Big[A\mathbf{x}\Big]_i$$

- Assume $\mathbf{b}_i$ comes from counting (e.g., counting photons that hit a detector), then it makes sense to use a *Poisson model*. That is,

$$\mathbf{b} \sim \text{Poisson}(A\mathbf{x}).$$

- To find the best $\mathbf{x}$, write a likelihood function, and find its likelihood (maximum likelihood/minimum negative log likelihood).

- One can then derive the *Expectation Maximization Algorithm (EM)* (Algorithm 21).

Algorithm 21: Expectation Maximization (EM)

```
1 begin
2     for k = 0, 1, 2, ... do
3         x_{k+1} = x_k ⊗ (A^⊤(b ⊘ A x_k));
          /* ⊗ and ⊘ represents element-wise × and /.        */
```

- Lets have $1, 2, 3, \ldots, m$ observations and group them into subsets:

$$\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_p.$$

Then, we have the *Ordered Subset EM Algorithm* (OSEM, Algorithm 22).

Algorithm 22: Ordered Subset EM (OSEM)

```
1 begin
2     for k = 0, 1, 2, ... do
3         for ℓ = 1, 2, ..., p do
4             A_s = A(i_ℓ, :);
5             b_s = b(i_ℓ);
6             x_{k+1} = x_k ⊗ (A_s^⊤(b_s ⊘ A_s x_k));
```

Remark.

1. Subsets $\mathbf{i}_\ell$ are row vectors of integers. For example, $\mathbf{i}_1 = \Big[1, 2, 3, 4\Big], \mathbf{i}_2 = \Big[5, 6, 7, 8\Big]$.
2. OSEM works well when the columns of A is well-balanced. if A has non-well-balanced columns, the ordering matters and Algorithm 22 might fail.