CME302 class notes

Erich Trieschman

2021 Fall quarter

Contents

1 Linear algebra review

1.1 Vector products

The **inner product**, also known as the dot product, results in a scalar

- $x^T y = \sum x_i * y_i$
- $x^T y = \|x\|_2 \|y\|_2 \cos\theta$

- $x^T y = 0 \Leftrightarrow x \perp y$

The **outer product** results in a matrix. It is the outer sum of the two vectors, which can be of different lengths.

1.2 Norms

All norms, matrix or vector, satisfy

- Only zero vector has zero norm: $\|x\|_x = 0 \Leftrightarrow x = 0$
- $\|\alpha x\|_x = |\alpha| \|x\|_x$
- $\|x + y\|_x \leq \|x\|_x + \|y\|_x$ (Triangle inequality I), $\|x - y\|_x \geq \|x\|_x - \|y\|_x$ (Triangle inequality II)

1.2.1 Vector norms

Types of **vector norms**, $x \in \mathbb{R}^n$ (norm selection can give you solutions with different properties)

- $\|x\|_1 = \sum_{i=1}^n |x_i|$
- $\|x\|_2 = \sqrt{\sum_{i=1}^n (x_i)^2}$
- $\|x\|_\infty = \max_{i \in i,\ldots,n} |x_i|$
- $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{\frac{1}{p}}$

Cauchy-Schwarts Inequality: $|x^T y| \leq \|x\|_2 \|y\|_2$ (note equality when $x^T y = 0$)

Holder's Inequality: $|x^T y| \leq \|x\|_p \|y\|_q$, for p, q , s.t. $\frac{1}{p} + \frac{1}{q} = 1$

1.2.2 Matrix norms

Types of **matrix norms**, $A \in \mathbb{R}^{n \times m}$

- $\|A\|_\infty = \sup_{x \neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} = \max_{\|x\|_\infty = 1} \|Ax\|_\infty = \max_i \left\|a_i^T\right\|_1$
- $\|A\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p = 1} \|Ax\|_p$
- $\|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2} = \sqrt{tr(AA^T)} = \sqrt{tr(A^T A)} = \sqrt{\sum_{k=1}^{min(m,n)} \sigma_k^2}$

Submultiplicative inverse: $\|AB\|_p \leq \|A\|_p \|B\|_p$. Note: this is not always true for Frobenius norms.

Induced p-norm: $\|Ay\|_p \leq \|A\|_p \|y\|_p$

Orthogonally invariant: Orthogonal matrices do not change the norms of vectors or matrices:

- $\|Qx\|_x = \|x\|_x$
- $\|QA\|_x = \|A\|_x , x \in \{p, F\}$

Other norm properties:

- $\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n} \|x\|_\infty$
- $\|A\|_2 \leq \sqrt{m} \|A\|_\infty$
- $\|A\|_\infty \leq \sqrt{n} \|A\|_2$

1.3 Matrix properties

Matrices represent the following linear operations on a vector: Scaling, 1D reflection, 2D reflection (about a plane in N-dim space), Dimension reduction or increase ($A : x \in \mathbb{R}^m \rightarrow y = Ax \in \mathbb{R}^n$)

1.3.1 Determinant

The **determinant** represents how the volume of a hypercube is transformed by the matrix.

- For square matrix, $det(\alpha A) = \alpha^n det(A)$
- For square matrices, $det(AB) = det(A)det(B)$
- $det(A) = det(A^T)$
- $det(A^{-1}) = \frac{1}{det(A)}$
- For square matrix, A singular $\Leftrightarrow det(A) = 0 \Leftrightarrow$ columns of A are not linearly independent

1.3.2 Trace

The trace of a matrix $A \in \mathbb{R}^{mxn}, tr(A)$, is equal to the sum of the entries in its diagonal, $tr(A) = \sum_{i=1}^{n} a_{ii}$. And a few properties of the trace:

- $tr(A) = tr(A^T)$
- $tr(A + \alpha B) = tr(A) + \alpha tr(B)$
- Trace is invariant under cyclic permutations, that is $tr(ABCD) = tr(BCDA) = tr(CDAB) = tr(DABC)$
- For two vectors, $u, v \in \mathbb{R}, tr(uv^T) = v^T u$

1.3.3 Inverses and transposes

The inverse of the transpose is the transpose of the inverse:

- $A^T(A^{-1})^T = (A^{-1}A)^T = I^T = I$
- $(A^{-1})^T A^T = (AA^{-1})^T = I^T = I$

1.3.4 Sherman-Morrison-Woodbury formula

for $A \in \mathbb{R}^{n\times n}, U, V \in \mathbb{R}^{n\times k}$

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^T A^{-1}U)^{-1}V^T A^{-1}$$

The significance of this formula is that you can compute the inverse of the sum of two matrices using the inverse of a known matrix, A, and the inverse of a much smaller matrix (assuming $k < n$) in $(I + V^T A^{-1}U)$

Proof: begin with the inverse of the LHS multiplied by the RHS: $(A + UV^T)(A^{-1} - A^{-1}U(I + V^T A^{-1}U)^{-1}V^T A^{-1})$. Next perform matrix multiplication. The end result will be I, implying that the RHS is an inverse of $(A + UV^T)$

1.4 Matrix multiplication

$$\text{Show: } AB = a_1 b_1^T + a_2 b_2^T + ... + a_n b_n^T, A, B \in \mathbb{R}$$

$$\text{Let } A = \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_n \\ | & | & & | \end{bmatrix}, B = \begin{bmatrix} — & b_1^T & — \\ — & b_2^T & — \\ & \vdots & \\ — & b_n^T & — \end{bmatrix}$$

$$a_1 b_1^T = \begin{bmatrix} a_{11}b_{11} & \dots & a_{11}b_{1n} \\ \vdots & & \vdots \\ a_{n1}b_{11} & \dots & a_{n1}b_{1n} \end{bmatrix} \Rightarrow \sum_{i=1}^{n} a_i b_i^T = \begin{bmatrix} \sum a_{1i}b_{i1} & \dots & \sum a_{1i}b_{in} \\ \vdots & & \vdots \\ \sum a_{ni}b_{i1} & \dots & \sum a_{ni}b_{in} \end{bmatrix} \Rightarrow AB$$

1.5 Orthogonal matrices

An orthogonal matrix, Q is a matrix whose columns are orthonormal. That is, $q_i^T q_j = 1$ for $i = j$, and $q_i^T q_j = 0$ for $i \neq j$. Equivalently, $Q^T Q = I$. For square matrices, $Q^T Q = QQ^T = I$

1.6 Projections, reflections, and rotations

1.6.1 Projections

A projection, v, of vector x onto vector y can be written in the form

$$v = \frac{y^T x}{y^T y} y$$

Which can be interpreted as the portion of x in the direction of y ($y^T x$), times the direction of y, divided by the length of y twice ($y^T y = \|y\|_2^2$), since y appears in the dot product and in the vector. Observe, the denominator would be 1 if y were a unit vector

Projection matrices are square matrices, P, s.t., $P^2 = P$.

1.6.2 Reflection

- P is a reflection matrix $\Leftrightarrow P^2 = I$
- P can be written in the form $P = I - \beta v v^T$, with $\beta = \frac{2}{v^T v}$, and v the vector orthogonal to the line/plane of reflection
- It can be shown that $Px = x \Leftrightarrow v^T x = 0$. These x are called the "fixed points" of P

1.7 Symmetric Positive Definite (SPD) Matrices

For A**, SPD,** i) $A = A^T$, ii) $x^T A x > 0 \; \forall x \neq 0$, iii) $a_{ii} > 0$, iv) $\lambda(A) \geq 0$, v) for B nonsingular, $B^T A B$ is also SPD.

When proving properties of SPDs, use the **following tricks:** i) Multiply by e_i since $e_i \neq 0$, ii) Use matrix transpose property, $x^T A^T = (Ax)^T$ to rearrange formulas

1.7.1 $B^T A B$ is also SPD

If A SPD $\Rightarrow B^T A B$ SPD for B nonsingular:

$$x^T B^T A B x = (Bx)^T A (Bx) > 0, (\text{since } B \text{ nonsingular} \Rightarrow Bx \neq 0)$$

1.8 Eigenvalues

Observe by definition $Ax = \lambda x \longleftrightarrow Ax - \lambda x = 0 \longleftrightarrow (A - \lambda I)x = 0$.

To find lambda, we solve for the system of equations to satisfy $(A - \lambda I)x = 0$

The **algebraic multiplicity** of an eigenvalue, λ_i, is the number of times that λ_1 appears in $\lambda(A)$
The **geometric multiplicity** of an eigenvalue, λ_i, is the dimension of the space spanned by the eigenvectors of λ_i

Other **eigenvalue properties**

- $\lambda(A) = \lambda(A^T)$
- Courant-Fischer minmax theorem: $\lambda_1 = \max_{x \neq 0} \frac{x^T A x}{\|x\|_2^2}$

1.8.1 Determinants and trace

$$det(A) = \prod_{i=1}^{n} \lambda_i \qquad\qquad tr(A) = \sum_{i=1}^{n} \lambda_i$$

1.8.2 Triangular matrices

For T triangular, the eigenvalues appear on the diagonal: $t_{ii} = \lambda_i, \forall i \in \{1, \ldots, n\}$
Corollary: T nonsingular $\Leftrightarrow$ all $t_{ii} \neq 0$

1.8.3 Gershgorin disc theorem

Gershgorin disc, $\mathbb{D}_i$, defined

$$\mathbb{D}_i = \{z \in \mathbb{C} \mid |z - a_{ii}| \leq \sum_{j \neq i} |a_{ij}|\}$$

All eigenvalues of A, $\lambda(A) \in \mathbb{C}$ are located in one of its Gershgorin discs. **Proof:**

$$Ax = \lambda x \longleftrightarrow (A - \lambda I)x = 0 \longleftrightarrow \sum_{j \neq i} a_{ij}x_j + (a_{ii} - \lambda)x_i = 0, \ \forall i \in \{1, \ldots, n\}$$

Choose $i \ s.t. |x_i| = \max_i |x_i|$

$$|(a_{ii} - \lambda)| = |\sum_{j \neq i} \frac{a_{ij}x_j}{x_i}| \leq \sum_{j \neq i} |\frac{a_{ij}x_j}{x_i}| \text{ , by triangle inequality}$$

$$|(\lambda - a_{ii})| \leq \sum_{j \neq i} |a_{ij}|, \text{ since } |\frac{x_j}{x_i}| \leq 1$$

2 Matrix Decompositions

2.1 Schur Decomposition

For any $A \in \mathbb{C}^{n \times n}$, $A = QTQ^H$, where Q unitary $(Q^HQ = I), Q \in \mathbb{C}^{n \times n}$, T upper triangular

When $A \in \mathbb{R}^{n \times n}$, $A = QTQ^T$, where Q orthogonal $(Q^TQ = I), Q \in \mathbb{R}^{n \times n}$, T upper triangular

Note: If T is relaxed from strict upper triangular to block upper triangular (blocks of 2×2 or 1×1 on the diagonal), then Q can be selected to be in $\mathbb{R}^{n \times n}$.

2.2 Eigenvalue Decomposition

For A diagonalizable ($A \in \mathbb{R}^{n \times n}$ with n linearly independent eigenvectors), it can be decomposed as

$$A = X\Lambda X^{-1}, \text{ where } \Lambda \text{ a diagonal matrix of the eigenvalues of } A$$

For A real symmetric, A can be decomposed as $A = Q\Lambda Q^T, Q$ orthogonal

For A unitarily diagonalizable ($\Leftrightarrow$ normal: $A^HA = AA^H$), $A = Q\Lambda Q^H, Q$ unitary. When A complex Hermitian ($A = A^H$), $\Lambda \in \mathbb{R}$

2.3 Singular Value Decomposition

Definition: For any $A \in \mathbb{C}^{m \times n}$ there exist two unitary matrices, $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$, and a diagonal matrix $\Sigma \in \mathbb{R}^{m \times n}$ such that $A = U\Sigma V^H$. When $A \in \mathbb{R}^{m \times n}$, $A = U\Sigma V^T$ with $U, V, \Sigma \in \mathbb{R}$

The singular values, σ_i of Σ are always ≥ 0. And by convention, they're ordered in decreasing order, so $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$

Motivation: Consider the action of a matrix, A on a sphere. A maps the sphere to a hyperellipsoid, E

- The lengths of the semi-axes of E are denoted $\sigma_1, \ldots, \sigma_n$ called **singular values** of A
- The directions of the semi axes are denoted by unit vectors, $u_1, \ldots, u_n$ called **left singular vectors** of A
- For each u_i there is some unit vector v_i so that $Av_i = \sigma_i u_i$. The vectors $v_1, \ldots, v_n$ are called the **right singular vectors**

Derivation: Observe A^TA symmetric: $(A^TA)^T = A^TA$

A^TA symmetric $\Rightarrow \exists\, Q$ orthogonal and Λ diagonal matrix of λ_i s.t.,

$$A^TA = Q\Lambda Q^T$$
$$Q^TA^TAQ = Q^TQ\Lambda Q^TQ$$
$(AQ)^T(AQ) = \Lambda$, note AQ is orthogonal, but not scaled to 1. Instead, each row is scaled to the eigenvalue in that row: $\lambda_i = \|Aq_i\|_2^2$

When A is full rank,

$$A = AQQ^T$$
$$= (AQ)Q^T$$
$$= AQD^{-1}DQ^T, \text{ where } D = \begin{bmatrix} \sqrt{\lambda_1} & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & \sqrt{\lambda_n} \end{bmatrix} \text{ and } D^{-1} = \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}} & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & \frac{1}{\sqrt{\lambda_n}} \end{bmatrix}$$
$$A = U\Sigma V^T, \text{ where } U = AQD^{-1}, \Sigma = D, V^T = Q^T$$

When A is not full rank, this does not hold since $\lambda_i = 0$ for some i so we cannot construct U with D^{-1}

$$\text{Start with } AQ = \begin{bmatrix} | & & | & | & & | \\ r_1 & \dots & r_r & 0 & \dots & 0 \\ | & & | & | & & | \end{bmatrix}$$
$$A = AQD^{-1}DQ^T, \text{ where } D = \begin{bmatrix} \sqrt{\lambda_1} & \dots & 0 & \\ \vdots & & \vdots & \\ 0 & \dots & \sqrt{\lambda_r} & \\ \vdots & & \vdots & I \\ 0 & \dots & 0 & \end{bmatrix} \text{ (observe this matrix has inverse, } D^{-1})$$
$A = U\Sigma V^T$, where

$U =$ [left r columns of AQ] $\times$ [upper-left diagonal block of $D^{-1} \in \mathbb{R}^{r\times r}$],

$\Sigma =$ [upper-left diagonal block of $D \in \mathbb{R}^{r\times r}$]

$V^T =$ [left block of Q, or upper block of Q^T]

And a few properties and remarks of $A \in \mathbb{R}^{n\times m}$ SVD

- $\|A\|_2 = \sigma_1$; $\left\|A^{-1}\right\|_2 = \frac{1}{\sigma_n}$ when A nonsingular; $\|A\|_F = \sqrt{\sum_i^{min\{n,m\}} \sigma_i^2}$
- When A symmetric, $\sigma_i = |\lambda_i|$; When A orthogonal, $\sigma_1 = \cdots = \sigma_n = 1$
- The eigenvalues of A^TA and AA^T are the squares of the singular values of A, $\sigma_1^2, \dots, \sigma_n^2$
- By construction, V contains the eigenvectors of A^TA and U contains the eigenvectors of AA^T, so $A^TAv_i = \sigma_i^2 v_i$ and $AA^Tu_i = \sigma_i^2 u_i$
- **Condition number**, $\kappa(A) = \|A\|_2 \left\|A^{-1}\right\|_2 = \frac{\sigma_1}{\sigma_n}$

3 Error analysis

3.1 Floating point arithmetic

The cause of most roundoff errors steps from addition/subtraction resulting in lower floating point precision. General floating point number equation:

$$\pm(\sum_{i=1}^{t-1} d_i\beta^{-i})\beta^e$$

Where

- β is the base (in floating point computation, $\beta = 2$)
- $d_0 \geq 1$, and $d_i \leq \beta - 1$.

- e is called the **exponent**, this is the location of the decimal place.
- $t-1$ in the summand is called the **precision** and indicates the number of digits (in base β) that can be stored with the number.
- Lastly, the part of the equation in the parenthesis is referred to as the **significand** or **mantissa**

3.2 Unit roundoff

The **unit roundoff** for a floating-point number is

$$u = \frac{1}{2} \times \beta^{-(t-1)} \text{ (distance between the smallest digits stored in a floating-point number)}$$

For double precision floating point numbers (64 bits), $u \approx 10^{-16}$
The **floating point truncation operator**, $fl(a)$, takes as input a and returns the nearest floating point, $fl(a)$. Observe

$$fl(a+b) = a + b + \epsilon(a+b),\ |\epsilon| \leq u, \text{ the unit roundoff}$$

To **prove** this inequality, i) write $fl(x)$ and x using floating point equations, ii) show the difference between these numbers is bounded by the smallest bit represented by $fl(x)$, iii) The $\frac{1}{2}$ enters the equation as a bound on the selection of the last digit of $fl(x)$ to approximate x.

3.3 Forward/Backward error analysis

Forward error analysis looks to create bounds between the computed quantity $\tilde{f}(A,b)$ and true value $f(A,b)$. The forward error is $\left\| \tilde{f}(x) - f(x) \right\|_p$. i.e., What is the error in the solution computed with our algorithm? This is difficult to compute.

Backward error analysis tries to find the error in A that leads to observed answer $\tilde{x}$, $\tilde{E}$ such that $(A+\tilde{E})\tilde{x} = b$. i.e., what is the problem that our algorithm actually solved? An algorithm is regarded as *backward stable* if $\|E\|_p \in O(u)$

The relative sensitivity of a problem is often called the **conditioning** of the problem

- Sensitivity: $\frac{\left\| \tilde{f}(x) - f(x) \right\|_p}{\|\tilde{x} - x\|_p}$
- Relative sensitivity: $\frac{\left\| \tilde{f}(x) - f(x) \right\|_p \|x\|_p}{\|\tilde{x} - x\|_p \|f(x)\|_p}$

4 LU Factorization

The LU factorization makes it computationally easier to solve linear equations If we can decompose a matrix, A, into a product of a lower triangular matrix, L, and an upper triangular matrix, U, then to solve $Ax = b$, we can start by solving $Lz = b$, and then $Ux = z$. x, here, is the solution!

4.1 Basic algorithm

We construct matrices L and U by iteratively subtracting outer products of vectors that sequentially "zero-out" the rows and columns of A. We know $LU = l_1 u_1^T + \cdots + l_n u_n^T$, and when l_1, u_1^T are from lower/upper respectively, $LU - l_1 u_1^T$ yields a matrix with zeros in the first row and column. We use this principle for the basic algorithm

- Construct u_1^T equal to the first row of A, a_1^T
- Construct l_1 equal to each of the elements in the first column of A, a_1, divided by a_{11}, the "pivot"
- Calculate $A' \leftarrow A - l_1 u_1^T$. In practice (and somewhat confusingly), A' is now referred to as A
- Repeat the algorithm with the updated A, and the next row/column. Observe each l_i, u_i^T constructed are the rows/columns of the lower and upper triangular matrices of L, U respectively.

4.1.1 Gauss transforms

Guass transformation matrices are linear transformations that zero out all entries below a certain entry (this is another way to think about the LU factorization). The columns of a Gauss transformation look like the values of l_i, where nonzero entries are divided by a pivot entry.

To compute $A = LU$, consider $L^{-1}A = U$, with L^{-1} that "zeros-out" the columns of A to get U. Call L^{-1}, G. As with the iterative algorithm above, we can multiply A by iterative G_i's to get U:

$$L^{-1}A = G_n G_{n-1} \dots G_2 G_1 A = U$$
$$A = G_1^{-1} \dots G_n^{-1} U = LU$$

4.2 Pivoting

4.2.1 When pivoting is needed

Notice that this algorithm relies on the pivots, a_{kk}, being nonzero. It turns out this will occur if none of the $k \times k$ blocks of A, $A[1:k, 1:k]$, have a determinant of 0. **Proof by induction**:
Case k=1:

$A_1 = L_1U_1 \longleftrightarrow det(A_1) = det(L_1U_1) \longleftrightarrow det(A_1) = det(L_1)det(U_1)$, by property of determinants
$det(A_1) = det(U_1)$, since determinant of a triangular matrix is a product of the diagonals and the diagonal of L_1 are 1's
$det(A_1) = a_{11} = u_{11} \rightarrow$ so when determinant is not zero, we have a nonzero pivot

Case k=n: assumed to be true
Case k=n+1:

$A_1 = L_1U_1 \longleftrightarrow det(A_{k+1}) = det(L_{k+1}U_{k+1}) \longleftrightarrow det(A_{k+1}) = det(L_{k+1})det(U_{k+1})$
$det(A_{k+1}) = det(U_{k+1}) \longleftrightarrow det(A_{k+1}) = u_{11} * u_{22} * \dots * u_{kk}$
but we know $u_{ii} \neq 0$ for $i \leq k$ from induction step, so when determinant is not zero,
we have pivot, $a_{k+1,k+1}$ nonzero

What's more, if the entries of L are large (which occurs when entries in A are really small and land on the pivot locations), then because of roundoff errors in a computer, this algorithm can generate errors. The **key** is to not have small values in the diagonal! Consider $A \in \mathbb{R}^{2\times 2}$ below. The issue arises when we need to calculate $\epsilon^{-1} + (\pi - \epsilon^{-1})$. With finite precision and ϵ small, this value is very different from π:

$$A = \begin{bmatrix} \epsilon & 1 \\ 1 & \pi \end{bmatrix}, L = \begin{bmatrix} 1 & 0 \\ \epsilon^{-1} & 1 \end{bmatrix}, U = \begin{bmatrix} \epsilon & 1 \\ 0 & \pi - \epsilon^{-1} \end{bmatrix},$$

4.2.2 Pivoting algorithms

Pivoting algorithms pivot the iterative version of A to avoid the numerical issues identified above

- **Partial/Row pivoting** performs row swaps at each step in the LU factorization so that the largest entry in a column appears in the pivot location. And we solve $PA = LU$, with P being a matrix storing the successive row swaps of A
- **Full pivoting** performs row and column swaps at each step in the LU factorization so that at each step, the largest remaining entry appears in the next pivot location. Here we solve $PAQ^T = LU$, with P swapping rows of A, and Q^T swapping columns.

  Full pivoting is **rank-revealing** since once the rank of the matrix r iterations have been performed, the remaining block will contain only zeros and the algorithm can stop early (plus we learned something about the rank of A!
- **Rook pivoting** performs row and column swaps at each step in the LU algorithm, but instead of swapping the pivot for the largest remaining entry, it swaps the next pivot for the first entry encountered that is maximum in its row and column. This pivoting approach is also rank-revealing and computationally less expensive!

4.3 Cholesky factorization

The Cholesky factorization is an LU factorization for Symmetric Positive Definite (SPD) matrices, where SPD matrix, $A = GG^T$, with G lower triangular.
Intuition: An SPD matrix, A, can be written of the form

$$A = \begin{bmatrix} a & C^T \\ C & B \end{bmatrix} \text{ where a is 1x1, C is n-1x1, and b is n-1xn-1}$$

After the first step of the LU factorization, we have the following matrix product, $A = L_1 U_1$

$$\begin{bmatrix} a & C^T \\ C & B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ C/a & I \end{bmatrix} \begin{bmatrix} a & C^T \\ 0 & B - (1/a)CC^T \end{bmatrix}$$

Notice since A is symmetric, B is also symmetric, so $B - (1/a)CC^T$ must by symmetric by construction. We are also guaranteed to have the pivot, a in entry $(1,1)$ of A, to be strictly greater than zero since A is SPD: $a = e_1^T A e_1 > 0$. Next, we can further decompose the second matrix to

$$A = \begin{bmatrix} 1 & 0 \\ C/a & I \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & B - (1/a)CC^T \end{bmatrix} \begin{bmatrix} 1 & C^T/a \\ 0 & I \end{bmatrix}$$

Using the fact that A SPD $\Rightarrow B^T A B$ SPD for B nonsingular, observe that matrix $\begin{bmatrix} 1 & 0 \\ C/a & I \end{bmatrix}$ is nonsingular so therefore the matrix $\begin{bmatrix} a & 0 \\ 0 & B - (1/a)CC^T \end{bmatrix}$ must be SPD. Which also means the submatrix $B - (1/a)CC^T$ is SPD. We can use induction to prove that the Cholesky factorization exists.

Continuing with this factorization, we get an equation of the form $A = LDL^T$ for D, diagonal, and L, lower triangular. It's common to rewrite $A = LDL^T$ in the form $A = GG^T$, where $G = LD^{\frac{1}{2}}$

4.3.1 Cholesky factorization is unique

By contradiction, suppose $A = GG^T = MM^T$ for $G \neq M$. We know G, M nonsingular (consider $det(A)$) so

$$\begin{aligned} GG^T &= MM^T \\ I &= G^{-1}MM^TG^{-T} = (G^{-1}M)(G^{-1}M)^T, \text{since}(A^{-1})^T = (A^T)^{-1} \\ (G^{-1}M)^{-T} &= (G^{-1}M) \\ &\Rightarrow G^{-1}M \text{ diagonal since } G^{-1}M \text{ lower triangular and } (G^{-1}M)^{-T} \text{ upper triangular} \\ &\Rightarrow G^{-1}M = D \Rightarrow M = GD \\ I &= (G^{-1}GD)(G^{-1}GD)^T = DD^T = D^2 \Rightarrow \text{ so the entries of D are on the order of 1} \end{aligned}$$

4.4 Schur complement

A useful way to think about the LU factorization is with the **Schur complement** matrix structure. First observe A can be written in the following form

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

If we run the LU factorization algorithm for k steps, the resulting $A' = A$ is equal to

$$A = \begin{bmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix} \begin{bmatrix} I & A_{21}A_{11}^{-1} \\ 0 & I \end{bmatrix}$$

The bottom-right block of $A' = A, A'_{22} = A_{22}$ is equal to $A_{22} - A_{21}A_{11}^{-1}A_{12}$ from the original matrix. This is called the **Schur complement** of A

4.4.1 Schur complement derivation

At any step in the LU factorization, A can be written in the form

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

From this equality, we can create a system of equations and derive

$$\begin{aligned} U_{12} &= L_{11}^{-1}A_{12} \\ L_{11}^{-1} &= L_{21}A_{21}A_{11}^{-1} \\ A_{22} - L_{21}U_{12} &= L_{22}U_{22} \\ A_{22} - A_{21}A_{11}^{-1}A_{12} &= L_{22}U_{22} \end{aligned}$$

Notice that the Schur complement equals the product of $L_{22}U_{22}$. The next step in the derivation is to show that A'_{22} in the LU factorization is equal to $A_{22} - L_{21}U_{12}$ since at each step we're subtracting $l_i U_i^T$, which can be stored as the nonzero rows/columns of $L_{21}U_{12}$. So

$$\begin{aligned} A'_{22} &= A_{22} - L_{21}U_{12} \\ &= (L_{21}U_{12} + L_{22}U_{22}) - L_{21}U_{12} \\ &= L_{22}U_{22} \\ &= A_{22} - A_{21}A_{11}^{-1}A_{12} \end{aligned}$$

5 QR factorization

The QR factorization decomposes a matrix, $A \in \mathbb{R}^{m\times n}, m \geq n$ into an orthogonal (orthonormal) matrix, Q, and an upper triangular matrix, R. When $A \in \mathbb{C}^{m\times n}$, Q is unitary.

Recall for $Q \in \mathbb{R}$, orthogonal, $Q^TQ = I$; for $Q \in \mathbb{C}$, unitary, $Q^HQ = I$; $\|Qx\|_2 = \|x\|_2$

If A is skinny (i.e., $n << m$), QR can take two different forms. $Q \in \mathbb{R}^{m\times m}$ can be square and $R \in \mathbb{R}^{m\times n}$ can be skinny. Or $Q \in \mathbb{R}^{m\times n}$ can be skinny and $R \in \mathbb{R}^{n\times n}$ can be square.

5.1 The QR factorization is unique

Proof that the QR factorization is unique for full rank matrix, A:

$$A = QR \longleftrightarrow Q^TA = R \longleftrightarrow^T Q^TA = R^TR \longleftrightarrow (QR)^TA = R^TR \longleftrightarrow A^TA = R^TR$$

We now have a matrix, A^TA that can be written of the form R^TR, which is the structure of the Cholesky factorization. Suffice to show that A^TA is Symmetric and Positive Definite (SPD) to prove the uniqueness of R.

Since A is full rank, it follows that Q is also unique (since $AR^{-1} = Q$). A^TA SPD:

Symmetric: $(A^TA)^T = A^TA$

Positive definite: for $x \neq 0$,

$$x^TA^TAx = (Ax)^T(Ax) = (QRx)^T(QRx) = x^TR^TQ^TQRx = (Rx)^T(Rx)$$

Rx is of the form $Rx = \begin{bmatrix} r_{11}x_1 \\ r_{12}x_1 + r_{22}x_2 \\ \vdots \\ \sum_{i=1}^n r_{in}x_i \end{bmatrix}$, so $(Rx)^T(Rx) = \sum_{i=1}^n (\sum_{j\leq i} r_{ij}x_j)^2$

So, $(Rx)^T(Rx) > 0$ for $x \neq 0$

5.2 Householder reflection

The Householder reflection is a QR factorization algorithm. It relies on the principles of reflection matrices.

5.2.1 Householder reflection algorithm

- Construct Q^T for each column in A that projects it onto a corresponding column of an upper right triangular matrix, R.
- E.g., for first column a_1: Want Q_1^T such that $Q_1^Ta_1 = r_1$, where $r_1 = \pm\|a_1\|_2 e_1$ (since Q^T is orthogonal). This equates to finding Q_1^T that reflects a_1 onto e_1
- **The key** to the iterative part of the algorithm is to construct $Q_i^T, i > 1$ with an identity matrix in the upper-left $i-1 \times i-1$ quadrant, and a smaller Q_i^{*T} in the lower right $n-i \times n-i$ quadrant, filling the remaining sections of the matrix with 0's.

5.2.2 Constructing the Householder reflection permutation

The **Householder reflection** maps $a \to \|a\|_2 e_1$ with

$$P = I - \beta vv^T, \text{ where } v = a - \|a\|_2 e_1, \text{ and } \beta = 2/v^Tv$$

- Mechanics: multiplying Px is the same as taking the vector x and subtracting $\frac{2vv^T}{v^Tv}x$ from it, twice the projection of x onto v (this is reflection)

- Householder: In our case we want to reflect a onto $\|a\|_2 e_1$. $a + \|a\|_2 e_1$ is the line of reflection, and $a - \|a\|_2 e_1$, perpendicular to this, is the vector that defines the line of reflection
- In cases where the other entries in a are much smaller than a_1, it may be advantageous to project onto $-\|a\|_2 e_1$ instead of $-\|a\|_2 e_1$ (to avoid roundoff errors. In this case, we choose $v = a + \|a\|_2 e_1$.

Aside: The fixed points of a reflection, P, remain unchanged when multiplied by the reflection, $Px = x$. Geometrically, these are the points that are *orthogonal* to the vector v defining the reflection (i.e., $v^T x = 0$)

5.3 Givens transformation

While the Householder reflection is useful for operating on dense matrices, if we are presented with a sparse matrix, the sequential reflections of will transform a sparse matrix into a dense matrix and create unnecessary complexity. For example, consider upper Hessenberg matrix, H. the **Givens transformation** will allow us to zero our the subdiagonal with less complexity!

5.3.1 Givens transformation algorithm

A **Givens rotation** rotates $u = (u_1, u_2)^T$ to $\|u\|_2 e_1$. The matrix that does this, G^T, is defined by

$$G^T = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}, c = \frac{u_1}{\|u\|_2}, s = -\frac{u_2}{\|u\|_2}$$

A full matrix, P_i, can be constructed to only contain this targeted transformation. Sequentially, the P_i's can multiply A to arrive at R

5.4 Gram-Schmidt transformation

Householder and Givens transformations produce square $Q \in \mathbb{R}^{n \times n}$ matrices. If $A \in \mathbb{R}^{m \times n}$ is tall and thin, then we may want a method to create a tall and thin $Q \in \mathbb{R}^{m \times n}$. The Gram Schmidt transformation does this.

Similar to the LU factorization, the **Gram-Schmidt Transformation** starts with the property that $A = QR$ can be written as a sum of the outer products of the columns of Q and rows of R: $A = QR = q_1 r_1^T + \dots q_m r_m^T$:

$$\begin{aligned}
r_{11} &= \|a_1\|_2, \text{ since } \|a_1\|_2 = \|q_1 r_{11}\|_2 \text{ and } q_i \text{ orthogonal} \\
q_1 &= \frac{1}{r_{11}} a_1, \text{ since } a_1 = q_1 r_{11} \text{ by construction of } QR \\
r_{1j} &= q_1^T a_j, \text{ (repeat for all } j\text{) since} \\
&\quad a_j = q_1 r_{1j} + \dots + q_j r_{jj} \\
&\quad q_1^T a_j = q_1^T q_1 r_{1j} + \dots + q_1^T q_j r_{jj} \\
&\quad q_1^T a_j = r_{1j}, \text{ since } q_i \text{ orthonormal} \\
A' &= A - q_1 r_1^T
\end{aligned}$$

Repeat for A', the construction of r_{kk}, q_k, r_{kj}

$$a_k = \sum_{i=1}^{k} r_{ik} q_i = r_{kk} q_k + \sum_{i=1}^{k-1} r_{ik} q_i$$

1. $r_{ik} = q_i^T a_k$ for each $r_{ik}, i < k$, since Q orthonormal and q_{k-1} known
2. $z = r_{kk} q_k = q_k - \sum_{i=1}^{k-1} r_{ik} q_i$
3. $r_{kk} = \|z\|_2, \; q_k = \frac{z}{r_{kk}}$

5.5 QR factorization to solve least-squares problems

When A is tall and thin, it is unlikely that we get a solution to $Ax = b$. Instead, we choose to solve the least-squares problem, $argmin_x \|Ax - b\|_2$.

5.5.1 Method of normal equations

Assuming A full rank. Geometrically, the point, x which solves $argmin_x \|Ax-b\|_2$ is one where $b-Ax$ is orthogonal to the range of A. To solve for this:

$$\text{Want: } (b-Ax) \perp \{z|z=Ay\} \longleftrightarrow (b-Ax) \perp range(A) \longleftrightarrow (b-Ax) \perp a_i, \forall i \in A$$
$$a_1^T(b-Ax)=0, \forall i \in A \longleftrightarrow A^T(b-Ax)=0 \longleftrightarrow x=(A^TA)^{-1}A^Tb$$

We can use Cholesky fast/accurate solve since A^TA is SPD. Notice, condition number of $A^TA, \kappa(A^TA)=\kappa(A)^2$, so if A is poorly conditioned, this method can get inaccurate.

5.5.2 QR method for least squares

Assuming A full rank. The QR method for least squares attempts to address the issue of poor conditioning and may also lead to faster computation. We construct the QR method for least squares with one of the normal equation equalities:

$$A^T(Ax-b)=0 \longleftrightarrow R^TQ^T(Ax-b)=0$$
$$Q^T(Ax-b)=0, \text{ since we assume } A,R \text{ full rank (multiply both sides by } R^{-T})$$
$$Q^TQRx-Q^Tb=0 \longleftrightarrow Rx=Q^Tb \longleftrightarrow x=R^{-1}Q^Tb$$

5.5.3 SVD for rank-deficient A

When A not full rank, we can get infinite solutions (a line of points that satisfy $argmin_x \|Ax-b\|_2$). To choose x, we add constraint $\min_x \|x\|_2$ to our original objective function, $argmin_x \|Ax-b\|_2$.

We can use the "thin" version of the Singular Value Decomposition to solve this, with $A \in \mathbb{R}^{m\times n}$, $rank(A)=r$, construct $U \in \mathbb{R}^{m\times r}$, $\Sigma \in \mathbb{R}^{r\times r}$(, notice this Σ has an inverse), $V^T \in \mathbb{R}^{r\times n}$. And calculate x as

$$(Ax-b) \perp range(A) \longleftrightarrow (Ax-b) \perp range(U), \text{ since } R(A)=R(U) \text{ for } A=U\Sigma V^T$$
$$U^T(Ax-b)=0 \longleftrightarrow U^T(U\Sigma V^Tx-b)=0 \longleftrightarrow \Sigma V^Tx=U^Tb$$
$$x=V\Sigma^{-1}U^Tb \text{ (the "thin" SVD here provides a nonsingular } \Sigma \in \mathbb{R}^{r\times r}, \text{ so we can take the inverse}$$

Observe for $\min_x \|x\|_2$ that the $x \perp N(A)$ is the shortest vector between $N(A)$ and the vector/plane of solutions to $argmin_x \|Ax-b\|_2$. This value must be in $R(V)$ since $R(V)=N(A)^\perp$

6 Iterative methods to find eigenvalues

6.1 Power iteration

Given $\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n \in \lambda(A)$, the **Power iteration** is a process for finding λ_1. The basic idea is to repeatedly multiply matrix A times a vector (normalizing each time) and eventually the first eigenvalue and eigenvector will emerge. This process assumes A is diagonalizable, meaning it can be written of the form $A=X\Lambda X^{-1}$. Reminder: for A diagonalizable $\implies A^k=X\Lambda^kX^{-1}$

$$A^k=\sum_{i=1}^{n}\lambda_i^kx_iy_i^T \text{ where } Y=X^{-1}$$
$$A^k \approx \lambda_1^kx_1y_1^T \text{ since } \lambda_1>\lambda_2$$
$$A^kq \approx \lambda_1^kx_1y_1^Tq=\lambda_1^k(y_1^Tq)x_1, \text{ since } y_1^Tq \text{ is a scalar. Observe } A^kq \parallel x_1$$

This theory is implemented in practice with the following formula

1. q_0, vector chosen at random
2. $z_k=Aq_k=A^kq_0$, evaluating for convergence if $z_k \parallel q_k \to z_k^Tx_k=\|z\|_2\|x\|_2$
3. $q_{k+1}=\dfrac{z_k}{\|z_k\|_2}=\dfrac{A^kq_0}{\|A^kq_0\|_2} \approx (\dfrac{\lambda_2}{|\lambda_1|})^kx_1$

Since $A^kq_0=Aq_k \approx \lambda_1x_1$, where $\|x_1\|_2=1$ $(WLOG)$ and $q_k \parallel x_1$, we can solve for λ:

$$Aq_k \approx \lambda_1 x_1 \Longrightarrow Ax_1 \approx \lambda_1 x_1 \Rightarrow x_1^H A x_1 \approx \lambda_1$$

Convergence: $O((|\frac{\lambda_1}{\lambda_2}|)^K)$, since

$$A^k q_0 = \sum_i \alpha_i A^k x_i = \sum_1 \alpha_i \lambda_i^k x_i$$

$$= \alpha_1 \lambda_1^k (x_i + \frac{\alpha_2}{\alpha_1}(\frac{\lambda_2}{\lambda_1})^k + \cdots + \frac{\alpha_n}{\alpha_1}(\frac{\lambda_n}{\lambda_1})^k)$$

$$\Longrightarrow \left\|A^k q_0\right\|_2 = |\alpha_1 \lambda_1^k|(1 + O(\frac{\lambda_2}{\lambda_1})^k)$$

Convergence: $O(|\frac{\lambda_2}{\lambda_1}|^k)$

6.2 Inverse iteration

This process finds the eigenvector (and corresponding eigenvalue) of A that is closest to the value μ. The basic idea is to multiply matrix $(A - \mu I)^{-1}$ iteratively by a random vector, z, normalizing each time. Eventually, you will get the eigenvector for the eigenvalue closest to μ.

Observe $(A - \mu I)^{-1}$ has the same eigenvectors of A:

$$(A - \mu I)^{-1} x = \lambda x \longleftrightarrow x = (A - \mu I)x = \lambda Ax - \lambda \mu x \longleftrightarrow \lambda Ax = x + \lambda \mu x \longleftrightarrow Ax = \frac{(1 + \lambda \mu)}{\lambda} x$$

Performing the power iteration on $(A - \mu I)^{-1}$, the largest eigenvalue to emerge will be of the form $\frac{1}{\lambda_i - \mu}$, and we get

$$(A - \mu I)^{-1k} q_0 = (A - \mu I)^{-1} q_k \approx \lambda_i x_i, \text{ where } \|x_i\|_2 = 1 \ (WLOG) \text{ and } q_k \parallel x_i$$

Since x_i is also an eigenvalue of A, we can solve $x_i^H A x_i = \lambda_i$ for the λ_i closest in magnitude to μ. **Convergence:** $O(|\frac{\lambda_i - \mu}{\lambda_j - \mu}|)^k)$, where λ_j is the next closest eigenvalue to μ

6.3 Eigenvalues of similar matrices

Theorem: For S nonsingular and $A = S^{-1}BS$, then i) $\lambda(A) = \lambda(B)$ and ii) x eigenvector of $A \Leftrightarrow S^{-1}x$ eigenvector of B.

i) $\lambda(A) = \lambda(B)$:

$det(A - \lambda I) = det(S^{-1}) det(A - \lambda I) det(S) = det(S^{-1}(A - \lambda I)S) = det(B - \lambda I)$

ii) x eigenvector of $A \Leftrightarrow S^{-1}x$ eigenvector of B

$Ax = \lambda x \rightarrow S^{-1}Ax = \lambda S^{-1}x \rightarrow S^{-1}ASS^{-1}x = \lambda S^{-1}x \rightarrow B(S^{-1}x) = \lambda(S^{-1}x)$

6.4 Eigenvalues from invariant subspaces

Theorem: $X \in \mathbb{R}^{n \times m}$ is an invariant subspace of $A \in \mathbb{R}^{n \times n} \Leftrightarrow$ there is a $B \in \mathbb{R}^{n \times m}$ such that $AX = XB$. **Proof:**

$$\Rightarrow: X \text{ invariant} \longrightarrow Ax_i \in X \longrightarrow Ax_i = \sum_{j=1}^{m} x_j b_{ji} \longrightarrow AX = XB$$

$$\Leftarrow: AX = XB \longrightarrow Ax_i = \sum_{j=1}^{m} x_j b_{ji} \longrightarrow Ax_i \in X \longrightarrow X \text{ invariant}$$

Furthermore, when $AX = XB$, the m eigenvalues of B are also eigenvalues of A: $By = \lambda y \longrightarrow XBy = \lambda Xy \longrightarrow AXy = \lambda Xy$

6.5 Orthogonal iteration

This process finds r eigenvalues and eigenvectors of A in a single iterative process.

First, consider how to construct orthogonal columns to reveal subsequent eigenvalues. Assume we use power iteration to compute q_1. To get q_2:

$$A^k = \lambda_1 x_1 y_1^T + \lambda_2 x_2 y_2^T + \ldots$$

$$PA^k = \lambda_1 P x_1 y_1^T + \lambda_2 P x_2 y_2^T + \ldots, \text{ where } P = I - x_1 x_1^T$$

$$PA^k = 0 + \lambda_2 P x_2 y_2^T + \ldots, \text{ since } Px_1 = Ix_1 - x_1 x_1^T x_1 = x_1 - x_1 = 0$$

PA can now be used to apply the power iteration to to reveal λ_2 and $(I - x_1^T x_1)x_2$

The general process is:

- Start with λ_1, q_1 from power iteration
- Build P_2, orthogonal projector onto $\{q_1\}^\perp$, use power iteration to reveal (λ_2, q_2)
- Build P_r, orthogonal projector onto $\{q_1, \ldots, q_{r-1}\}^\perp$, use power iteration to reveal (λ_r, q_r)

Now consider the QR decomposition of X, observing its connection to the Schur Decomposition:

$$A = X\Lambda X^{-1} = QR\Lambda R^{-1}Q^H = QTQ^H, \text{ where upper triangular } T = R\Lambda R^{-1}$$

- The eigenvalues of A are on the diagonal of T
- By construction, each column of Q is projecting the corresponding column of X onto a vector orthogonal to the preceding ones
- The span of the columns of $Q, span\{q_1, \ldots, q_n\}$ will be equal to the span of the columns of $X, span\{x_1, \ldots, x_n\}$.

The process for the **orthogonal iteration** is:

1. $AQ_k \to Z$, where k is the iteration and $Q_0 = I$
2. $Z \to Q_{k+1}R_{k+1}$, the QR factorization of Z
3. Repeat $AQ_{k+1} \to Z$ and eventually $Q_k \to Q$

Note in each iteration we are calculating $Q_{k+1}^H AQ_k = R_{k+1}$

6.5.1 Reveal eigenvectors of A from T

Motivation: $A = X\Lambda X^{-1}$ can be hard to calculate.

$$A = X\Lambda X^{-1} = QR\Lambda R^{-1}Q^H = QTQ^H, \text{ where } T = R\Lambda R^{-1}$$
$$A = QY\Lambda Y^{-1}Q^H, \text{ where } T = Y\Lambda Y^{-1} \text{ is easier to compute}$$

Focusing on $T = Y\Lambda Y^{-1}$, choose some λ_i (we could get from power or QR iteration).

$$\begin{aligned} Tx &= \lambda_i x \\ (T - \lambda_i I)x &= 0 \\ (T - \lambda_i I)x &= \begin{bmatrix} T_{11} - \lambda_i I & T_{12} & T_{13} \\ 0 & 0 & T_{23} \\ 0 & 0 & T_{33} - \lambda_i I \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \text{ where one diagonal element is 0} \end{aligned}$$

And solve with back substitution:

$$\begin{aligned} &X_3 = 0 : (T_{33} - \lambda_i I)X_3 = 0 \\ &X_2 \text{ is a free parameter} \in \mathbb{R} : 0X_2 + T_{33}X_3 = 0 \implies 0X_2 = 0 \\ &X_1 = -(T_{11} - \lambda_i I)^{-1}T_{12}X_2 : (T_{11} - \lambda_i I)X_1 + T_{12}X_2 + T_{13}X_3 = 0 \end{aligned}$$

So, if T upper triangular with λ_i on diagonal of T, you can figure out all the columns of Y for $T = Y\Lambda Y^{-1}$. It follows the eigenvectors of A are Qy_i. Note, $(T_{11} - \lambda_i I)$ nonsingular as long as the algebraic multiplicity of λ_i is 1.

6.5.2 Rate of convergence in orthogonal (and QR) iteration

Property: the angle between two subspaces, U and V, is defined as $\left\|UU^T - VV^T\right\|_2$
In orthogonal interation, the span of those i columns of $Q_k, span\{q_1, \cdots, q_i\} \longrightarrow$ the span of those columns of $X, span\{x_1, \cdots, x_i\}$. Convergence is dictated by how quickly these spans converge. The rate of convergence is $O(|\frac{\lambda_{i+1}}{\lambda_i}|^k)$. **Note:** difficulties arrise when $|\frac{\lambda_{i+1}}{\lambda_i}|$ is close to 1.

6.6 QR iteration

The QR iteration builds directly on the framework of the orthogonal iteration. In orthogonal iteration, we compute T_{k+1} with the eigenvalues of A appearing on the diagonal of T_{k+1}

$$Q_{k+1}^H AQ_k = T_{k+1} \text{ with } AQ_k = Z = Q_{k+1}T_{k+1}$$

In the QR iteration, we ask if we can go from T_k to T_{k+1} directly. Observe

$$A = Q_k T_k Q_k^H \Longrightarrow T_k = Q_k^H A Q_k$$
$$AQ_k = Q_{k+1} R_{k+1} \Longrightarrow Q_{k+1}^H A = R_{k+1} Q_k^H$$

$$T_k = Q_k^H (Q_{k+1} R_{k+1}) \longrightarrow T_k = U_{k+1} R_{k+1} \text{ for } U_{k+1} = Q_k^H Q_{k+1}$$
$$T_{k+1} = (R_{k+1} Q_k^H) Q_{k+1} \longrightarrow T_{k+1} = R_{k+1} U_{k+1} \text{ for } U_{k+1} = Q_k^H Q_{k+1}$$

So we have an algorithm for $T_k \to T_{k+1}$, this process is the **QR iteration**:

1. $T_k \longrightarrow U_{k+1} R_{k+1}$, the QR factorization of T_k
2. $R_{k+1} U_{k+1} \longrightarrow T_{k+1}$
3. Repeat with T_{k+1}

Proof by induction: R_{k+1} is the same in both QR factorization of $A = Q_{k+1} R_{k+1}$ and $T_k = U_{k+1} R_{k+1}$

$$\begin{aligned}
&case\ 1: \\
&A = AQ_0 = Q_1 R_1, A = T_0 = U_1 R_1^*, \text{ and } T_1 = Q_k^H A Q_1 \\
&U_1 R_1^* = Q_0^T Q_1 R_1 = Q_1 R_1 \implies R_1^* = R_1 \text{ and } U_1 = Q_0^T Q_1 \\
&case\ k: \text{Assume } R_k^* = R_k, U_k = Q_{k-1}^T Q_k, \text{ and } T_k = Q_k^H A Q_k \\
&case\ k+1: \\
&AQ_k = Q_{k+1} R_{k+1} \\
&T_k = U_{k+1} R_{k+1}^* = Q_k^H A Q_k = Q_k^H Q_{k+1} R_{k+1} \Longrightarrow R_{k+1}^* = R_{k+1} \text{ and } U_{k+1} = Q_k^H Q_{k+1} \\
&T_{k+1} = R_{k+1} U_{k+1} = Q_{k+1}^H (Q_{k+1} R_{k+1}) U_{k+1} = Q_{k+1}^H A Q_k Q_k^H Q_{k+1} \Longrightarrow T_{k+1} = Q_{k+1}^H A Q_{k+1}
\end{aligned}$$

6.7 QR iteration on upper Hessenberg

Each QR iteration step of a dense matrix is $O(n^3)$. If we run for $O(k)$ iterations, then this algorithm is $O(kn^3)$. To reduce flops, we can first convert A to upper Hessenberg ($H = Q^H A Q$) with $O(n^3)$, and proceed with QR iteration on H using Givens rotations with complexity $O(n^2)$ (so overall complexity is reduced to $O(n^3 + kn^2)$):

$$\text{Choose } Q_1^T = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_1 \end{bmatrix} \text{ to perform a Householder rotation onto the first two entries of } a_1 \in A$$

$$\text{Observe } Q_1^T A Q_1 = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_1 \end{bmatrix} A \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P_1^T} \end{bmatrix} = \begin{bmatrix} x & x & \cdots \\ x & x & \cdots \\ 0 & x & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \text{ where } a_{11} \text{ is never changed, the rest of } a_1$$

$$\text{is only operated on by } \tilde{P}_1, \text{ and the rest of } a_1^T \text{ is only operated on by } \tilde{P_1^T}$$

$$\text{Continuing on, } Q_n^T \dots Q_2^T Q_1^T A Q_1 Q_2 \dots Q_n = H = Q^H A Q \text{ where } Q_k^T = \begin{bmatrix} I_k & 0 \\ 0 & \tilde{P}_k \end{bmatrix}$$

H remains upper Hessenberg in QR iteration: This follows since in the first step of QR iteration, H_k is transformed to R_k with givens rotations, $U_k^H H_k = R_k$. And in the second step of QR iteration, H_{k+1} is created as $R_k U_k = H_{k+1} = U_k^H H_k U_k$. Since U_k is a series of givens rotations, these rotations can be constructed/ordered so that H_{k+1} preserves upper Hessenberg.

6.8 QR iteration with shift

When λ_{i+1} is close to λ_i, **QR iteration with shift** accelerates convergence. First observe for $\lambda_i \in \lambda(A) \to (\lambda_i - \mu) \in \lambda(A - \mu I)$. In this algorithm, at each step we shift T_k by μI. For μ close to λ_{i+1} close to λ_i, the resulting converence, $|[(\lambda_{i+1} - \mu)/(\lambda_i - \mu)]|^k$ will be faster. Shift does not require that $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$.

In general, the **QR iteration with shift** works by shifting the last eigenvalue (smallest in absolute value), updating the shift in each iteration. The last eigenvalue makes sense here because it preserves the eigenvalue ordering:

1. $\mu_k = T_k[n, n]$
2. $(T_k - \mu_k I) \longrightarrow U_{k+1} R_{k+1}$, QR factorization of the shifted T_k
3. $R_k U_k + \mu_k I \longrightarrow T_{k+1}$, and repeat!

Observe, this shift preserves the original QR iteration:

$$\begin{aligned}(T_k - \mu_I) = U_{k+1}R_{k+1} &\implies U_{k+1}^H T_k - \mu_k U_{k+1}^H = R_{k+1} \\ T_{k+1} = R_{k+1}U_{k+1} + \mu_k I &\implies T_{k+1} = (U_{k+1}^H T_k - \mu_k U_{k+1}^H)U_{k+1} + \mu_k I \\ T_{k+1} &= U_{k+1}^H T_k U_{k+1} - \mu_k I + \mu_k I = U_{k+1}^H T_k U_{k+1}\end{aligned}$$

6.8.1 Implicit Q theorem

The **implicit Q theorem** tells us that if i) we get any upper Hessneberg, H_{k+1} from a transformation of $H_k \to H_{k+1}$ of the form $U^T H_k U$ ii) $We_1 = Qe_1$ for two such transformations, then the columns of W and Q are equal, up to a sign.

Proof: We show for $A = QHQ^T$, Q orthogonal and H upper Hessenberg, that Q, H are determined by A and Qe_1:

$$\begin{aligned}AQ &= QH, \text{ assume we know } q_1, \dots, q_k \text{ of } Q \\ A\begin{bmatrix}Q_k & X\end{bmatrix} &= \begin{bmatrix}Q_k & X\end{bmatrix}\begin{bmatrix}H_k & X \\ 0 & X\end{bmatrix}, X \text{ unknown and } H_k \in \mathbb{R}^{k\times k} \\ Aq_k &= \sum_{i=1}^{k} h_{i,k}q_i + k_{k+1,k}q_{k+1}, \text{ the kth column of } AQ, \text{ where } q_j^T A q_k = h_{j,k} \\ k_{k+1,k}q_{k+1} &= Aq_k - \sum_{i=1}^{k} h_{i,k}q_i, \text{ the RHS of which is known} \\ &\Rightarrow |h_{k+1,k}| = \left\| Aq_k - \sum_{i=1}^{k} h_{i,k}q_i \right\|_2 \text{ and } q_{k+1} = \frac{Aq_k - \sum_{i=1}^{k} h_{i,k}q_i}{h_{k+1,k}}\end{aligned}$$

Conclusion: if we know the first k columns of Q, the subsequent column and elements of an upper Hessenberg matrix are determined up to a sign.

6.8.2 Fracis shift

The **Francis shift** is a way of selecting shifts based on the bottom-right 2×2 block in a way that maintains a real-valued matrix. In effect, we double-shift using complex conjugates, $\mu, \overline{\mu}$:

$$\begin{aligned}H_{k-1} - \mu I &= U_k R_k \\ H_k &= R_k U_k + \mu I \\ H_k - \overline{\mu} I &= U_{k+1}R_{k+1} \\ H_{k+1} &= R_{k+1}U_{k+1} + \overline{\mu} I \\ H_{k+1} &= U_{k+1}^H H_k U_{k+1} = U_{k+1}^H U_k^H H_{k-1} U_k 1 U_{k+1} = (U_k U_{k+1})^H H_{k-1}(U_k U_{k+1})\end{aligned}$$

Proof Consider QR factorization to show (U_1U_2) is real

$$\begin{aligned}(U_kU_{k+1})(R_{k+1}R_k) &= U_k(H_k - \overline{\mu}I)R_k = U_k(R_kU_k + \mu I - \overline{\mu}I)R_k = U_kR_k(U_kR_k + (\mu - \overline{\mu})I) \\ &= (H_{k-1} - \mu I)(H_{k-1} - \mu I + (\mu - \overline{\mu})I) = (H_{k-1} - \overline{\mu}I)(H_{k-1} - \mu I) \\ &= H_{k-1}^2 - (\mu + \overline{\mu})H_{k-1} + |\mu|^2 I, \text{ where each component of the polynomial is } \in \mathbb{R}\end{aligned}$$

From uniqueness of QR factorization, (U_1U_2) must be real as well. So at any step of the Francis shift, we want $H_{k+1} = Q^T H_{k-1} Q$

- Define $M = H_{k-1}^2 - (\mu + \overline{\mu})H_{k-1} + |\mu|^2 I$, noticing Me_1 only has nonzero entries in the first three rows
- Want to build $V = U_1U_2$ to do shift, noticing we can get V from QR factorization of $M = VR = (U_1U_2)(R_2R_1)$
- Using bulge chasing starting with $P_1^T Me_1 = e_1$, noticing i) for $M = (U_1U_2)(R_2R_1)$ that U_1U_2 also has only has nonzero entries in the first three rows, ii) by implicit Q theorem $V^THV = (U_1U_2)^T H(U_1U_2)$ is upper Hessenberg

6.9 QR iteration with deflation

Deflation allows us to break up the current QR iteration process into two smaller/easier problems.

- If any sub-diagonal element of an upper Hessenberg matrix, H, is 0, it can be written as $H = \begin{bmatrix}H_{11} & H_{12} \\ 0 & H_{22}\end{bmatrix}$ with H_{11} and H_{22} upper Hessenberg and $\lambda(H) = \lambda(H_{11}) \cup \lambda(H_{22})$

- Therefore, if when updating $T_k = R_k U_k + \mu_k I$, any sub-diagonal element of $T_k = 0$, then T_k can be written in this form and the QR iteration can be performed on $(T_k)_{11}$ and $(T_k)_{22}$ separately (simpler problems)

Theorem: $\lambda(H) = \lambda(H_{11}) \cup \lambda(H_{22})$ for H block upper triangular. **Proof:**

$$\implies Hx = \lambda x \longrightarrow \begin{bmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} H_{11}x_1 + H_{12}x_2 \\ H_{22}x_2 \end{bmatrix} = \begin{bmatrix} \lambda x_1 \\ \lambda x_2 \end{bmatrix}$$

and either $x_2 = 0$ and $\lambda \in \lambda(H_{11})$ or not and $\lambda \in \lambda(H_{22})$

$$\impliedby H_{11}p_1 = \lambda p_1 \longrightarrow \begin{bmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{bmatrix} \begin{bmatrix} p_1 \\ 0 \end{bmatrix} = \begin{bmatrix} H_{11}p_1 \\ 0 \end{bmatrix} = \begin{bmatrix} \lambda p_1 \\ 0 \end{bmatrix}$$

$$\impliedby H_{22}p_2 = \lambda p_2 \longrightarrow \begin{bmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{bmatrix} \begin{bmatrix} x \\ p_2 \end{bmatrix} = \begin{bmatrix} H_{11}x + H_{12}p_2 \\ H_{22}p_2 \end{bmatrix} = \begin{bmatrix} \lambda x_1 \\ 0 \end{bmatrix}$$

where $H_{11}x + H_{12}p_2 = \lambda x$ for $x = -(H_{11} - \lambda I)^{-1}H_{12}p_2$, making $\lambda \in \lambda(H)$

Theorem: If H is singular unreduced upper Hessenberg, then in QR factorization, $H = QR$, the last row of R is zero.
Explanation: When constructing QR iteration, each column of R can be linearly independent from the previous ones (since we're adding a dimension) except for the last one (since H and R must be singluar):

$$h_1 = h_{11}e_1 + h_{21}e_2 \qquad h_2 = h_{12}e_1 + h_{22}e_2 + h_{32}e_3 \qquad h_{n-1} = \sum_{i=1}^{n} h_{n-1,i}e_i$$

6.10 QR iteration on symmetric matrices

Upper Hessenberg symmetric matrices are tri-diagonal matrices

- Unsymmetric case complexity: Transform to upper Hessenberg: $O(n^3)$; QR iteration step: $O(n^2)$; overall QR iteration: $O(pn^3)$, where p is the number of iterations per eval (assume quadratic convergence)
- Symmetric case complexity: Transform to upper Hessenberg: $O(n^3)$; QR iteration step: $O(n)$; overall QR iteration: $O(pn^2)$, where p is the number of iterations per eval (assume cubic convergence)

7 Finding eigenvalues of sparse matrices

Define **a sparse matrix** as a matrix with the number of nonzero entries on the order of $O(1)$ (i.e., does not scale with matrix size). The main operation for reducing complexity in this area is matrix-vector multiplication, $Ax = \sum_j a_{ij}x_j$, where you can skip all a_{ij} when $a_{ij} = 0$.

7.1 Arnoldi process

The **Arnoldi process** is used to reveal the first k eigenvalues of a sparse matrix using a process similar to *Gram-Schmidt*
With Arnoldi process, we start with equation $Q^H AQ = H \implies AQ = QH$ and use Q to make H where each subsequent column of AQ is made orthogonal to all preceeding columns. The process follows:

1. Begin with random $q_1 \in Q$, such that $\|q_1\|_2 = 1$

Iterate through each of the first k columns of Q with

2. $Aq_j = \sum_{k=1}^{j+1} h_{kj}q_k$, observing we can recover all h_{ij} for $i \leq j$ since $q_i^T Aq_j = h_{ij}$
3. $Aq_j = \sum_{k=1}^{j} h_{kj}q_k + h_{j+1,j}q_{j+1}$
4. $r = Aq_j - \sum_{k=1}^{j} h_{kj}q_k = h_{j+1,j}q_{j+1}$, where only r is unknown
5. $\|q_{j+1}\|_2 = 1 \implies h_{j+1,j} = \|r\|_2$ and $q_{j+1} = \dfrac{r}{h_{j+1,j}}$

The output of this process is k columns of Q and the upper $k \times k$ block of upper Hessenberg matrix, H, which can be used in the QR iteration to reveal k eigenvalues close to $\lambda(A)$:

$$AQ = QH \implies AQ_k = Q_k H_k + h_{k+1,k}q_{k+1}e_k^T, \text{ where } Q_k = Q[:, 1:k], H_k = [1:k, 1:k]$$
$$AQ_k = Q_k X_k \Lambda_k X_k^{-1} + h_{k+1,k}q_{k+1}e_k^T, \text{ where } H_k = X_k \Lambda_k X_k^{-1} \text{ through QR iteration}$$
$$A(Q_k X_k) = (Q_k X_k)\Lambda_k + h_{k+1,k}q_{k+1}x_k^T, \text{ where } x_k^T \text{ is the } k^{th} \text{ column of } X$$

And we get an equation where i) $AQ_k \approx Q_k H_k$, ii) Λ_k contains k eigenvalues close to $\lambda_i \in \lambda(A)$, iii) $(Q_k X_k)$ serve as eigenvectors for those eigenvalues, and iv) $h_{k+1,k} q_{k+1} x_k^T$ represents something like an error term.

7.2 Krylov spaces

A **Krylov subspace** is defined as a space of sparse Matrix-vector products: $K(A, q, k) = span\{q_1, Aq_1, A^2 q_1, \ldots, A^k q_1\}$

In general, Krylov spaces can be used approximate linear algebra problems of $A \in \mathbb{R}^{n \times n}$ in a $K(A, q_1, k) \in \mathbb{R}^{k \times k}$ space instead.

7.2.1 QR factorization of Krylov subspace contains Q_k from Arnoldi

Proof: We show for $K_k = Q_k R_k$, that R_k is upper triangular.

$$\text{Start with } Q^T K_k = R \text{ upper triangular for } K_k = \begin{bmatrix} | & | & & | \\ q_1 & Aq_1 & \ldots & A^k q_1 \\ | & | & & | \end{bmatrix}$$

$$\begin{aligned} Q^T k_j = Q^T A^{j-1} q_1 &= Q^T Q H^{j-1} Q^T q_1, \text{ since } A^k = Q^T H^k Q \\ &= H^{j-1} Q^T q_1 = H^{j-1} e_1, \text{ since } Q \text{ orthogonal} \\ \Rightarrow r_j \in R = h_1 &\in H^{j-1}, \text{ which has top } j \text{ rows nonzero} \end{aligned}$$

The last statement can be checked by iteratively checking the first column of H^i. This result indicates that $Q_k K_k$, produces an upper right triangular matrix since Q_k is the first k columns of Q. This also means Q_k forms a basis for $K(A, q_1, k)$.

7.2.2 Arnoldi process generates a minimal polynomial

Polynomial properties

- If A is diagonalizable, i.e., $A = X \Lambda X^{-1}$, then polynomial $f(A) = X f(\Lambda) X^{-1}$
- **Characteristic polynomial** of A is $p_A(z) = det(zI - A) = \prod (z - \lambda_i)$ and $p_A(\lambda_i) = 0$ for $\lambda_i \in \lambda(A)$
- $f(A) = 0 \Longrightarrow \lambda_i \in \lambda(A)$ are the roots of the polynomial (e.g., $p_A(A) = X p_A(\Lambda) X^{-1} = 0$

Our hope with the Arnoldi process is that for $p_k(H_k) = 0$, revealed in Arnoldi, $p_k(A)$ is minimally small among degree $k-1$ polynomials. Instead of showing $\|p_K(A)\|_2$ is minimized (which is hard), we show $\|p_K(A) q_1\|_2$ is minimized:

$$\begin{aligned} f(x) &= x^k + f_{k-1} x^{k-1} + \cdots + f_0, \text{ for } f \text{ that minimizes } \|f(A) q_1\|_2 \\ f(A) &= (A^k + f_{k-1} A^{k-1} + \cdots + f_0) q_1 = A^k q_1 + K_k f, \text{ where } f \text{ is a vector of coefficients} \\ &= A^k q_1 + Q_k y, \text{ for some } y, \text{ since } Q_k \text{ forms a basis for Krylov space} \\ \text{Minimal } \|f(A) q_1\|_2 &\Longrightarrow \text{minimal } \left\|A^k q_1 + Q_k y\right\|_2, \text{ so we need to choose } y \text{ to minimize polynomial} \\ \text{minimal } \left\|A^k q_1 + Q_k y\right\|_2 &\Longrightarrow Q_k^T f(A) q_1 = 0 \\ Q_k^T f(A) q_1 = Q_k^T Q f(A) Q^T q_1 &= \begin{bmatrix} I_k & 0 \end{bmatrix} f(H) e_1 = I_k f(H_k) e_1 \end{aligned}$$

This proof shows that $\|f(A) q_1\|_2$ is minimal $\Leftrightarrow I_k f(H_k) e_1 = 0$, the first column of $f(H_k)$ is zero. Now observe that $p_k(H_k)$ achieves this since $p_k(H_k) = 0$. Finally, assuming K_k is full rank, we know p_k must uniquely minimize this norm.

7.3 Lanczos process

The **Lanczos process** is a parallel process to the Arnoldi process, but for symmetric matrices. Reminder: A symmetric upper Hessenberg matrix, T is tri-diagonal. Note this tri-diagonal matrix is of the form

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \ldots \\ \beta_1 & \alpha_2 & \beta_2 & \ldots \\ 0 & \ddots & \ddots & \ddots \end{bmatrix}$$

The process follows

1. $\alpha_k = q_k^T A q_k \Longrightarrow \alpha_k q_k = A q_k$
2. $r_k = A q_k - \beta_{k-1} q_{k-1} - \alpha_k q_k \Longrightarrow r_k = \beta_{k-1} q_{k-1}$, r_k becomes the orthogonal part of $A q_k$
3. $\beta_k = \|r_k\|_2$
4. $q_{k+1} = \dfrac{r_k}{\beta_k}$

The orthogonalization in step 2 is reduced from $O(k)$ in Arnoldi to $O(1)$ in Lanczos because of the symmetry of A. Professor Darve notes the "magic" in this.

7.3.1 Process for revealing the max eigenvalue of A

$$\lambda(T_k) \approx \lambda(A)$$

$$\lambda_1 \in \lambda(T_k) = \max_{x \neq 0} \frac{y^T Q_k^T A Q y}{\|y\|_2^2}, \text{ by property that } \lambda_1 \in \lambda(A) = \max_{x \neq 0} \frac{x^T A x}{\|x\|_2^2}$$

$$\implies \text{want max } x \text{ of the form } Q_k y$$

$$\implies \text{want max } x \text{ in Krylov space, a subspace of } \mathbb{R}^k$$

$$\implies \lambda_1 \in \lambda(T_k) \leq \lambda_1 \in \lambda(A), \text{ since it is the max in a smaller space}$$

$$\lambda_1 \in \lambda(T_k) = \max_{x \neq 0} \frac{q_1^T p(A) A p(A) q_1}{q_1^T p(A)^2 q_1}, \text{ and see textbook for step from here to next step}$$

$$\implies \lambda_1 \in \lambda(T_k) \leq \lambda_1 - (\lambda_1 - \lambda_n)(\frac{\tan(\theta)}{T_{k-1}^{Cheb}(1+2p_1)}), \text{ where } p_1 = \frac{\lambda_1 - \lambda_2}{\lambda_2 - \lambda_n}$$

Observe that the RHS approaches λ_1 when λ_1 is well separated from the other eigenvalues.

8 Iterative splitting methods for solving linear systems

Use iterative methods to solve linear systems when i) A large and direct methods are computationally expensive, or ii) A is sparse and Ax easily computed. Ideally these methods can reduce cost below $O(n^3)$
General idea:

For $A = M - N$, where M nonsingular: $Ax = b \iff Mx - Nx = b \iff x = M^{-1}Nx + M^{-1}b$
And x becomes a fixed point of $f(x) = M^{-1}Nx + M^{-1}b$

Fixed points can be identified by iteratively computing $f(x), f(f(x)), \ldots$ on starting input, x.

Convergence depends on $M^{-1}N$: Define the error at step $k+1$ as $e_{k+1} = x_{k+1} - x$, then

$$\begin{aligned} e_{k+1} &= x_{k+1} - x = M^{-1}Nx_k + M^{-1}b - M^{-1}Nx - M^{-1}b \\ &= M^{-1}Nx_k - M^{-1}Nx = M^{-1}N(x_k - x) = M^{-1}Ne_k \\ e_{k+1} &= (M^{-1}N)^k e_0 \end{aligned}$$

And convergence only occurs when the spectral radius of $M^{-1}N$, $\rho(M^{-1}N) < 1$. Where $\rho(A) = \max_{\lambda_i \in \lambda(A)} |\lambda_i|$
Proof:

Let $Gv = \lambda v$, where $G = M^{-1}N$ and pick $x_0 = x + v$ as a first guess solve
Then $e_k = G^k e_0 = G^k(x_0 - x) = G^k v = \lambda^k v$

And $e_k \to 0$ if $|\lambda| < 1$.

The following sections review how to pick M and N

8.1 Jacobi

Definition: Let $A = D - L - U$, and choose $M = D$ and $N = L + U$. Then we have $Dx_{k+1} = (L+U)x_k + b$
Convergence: Converges for any x_0 when A is strictly diagonally dominant: $|a_i i| > \sum_{j \neq i} |a_i j|$
Proof:

$$g_{ii} = 0 \text{ for } G = M^{-1}N \implies \lambda \in D_i = \{z \mid |z| \leq \sum_{i \neq j} |g_{ij}|\}, \text{ by Gershgorin Disc Theorem}$$

$$\text{Suffice to show } \sum_{i \neq j} |g_{ij}| < 1 : \quad \sum_{i \neq j} |g_{ij}| < 1 \implies \sum_{i \neq j} \frac{|a_{ij}|}{|a_{ii}|} < 1 \implies \sum_{i \neq j} |a_{ij}| < |a_{ii}|$$

$$\implies A \text{ is strictly diagonoally dominant}$$

8.2 Gauss-Seidel

Definition: Let $A = D - L - U$, and choose $M = D - L$ and $N = U$. Then we have $(D - L)x_{k+1} = Ux_k + b$
Householder-John Theorem: if $A, B \in \mathbb{R}^{m\times n}$ and A and $[A - B - B^T]$ are SPD, then $\rho(H) < 1$ where $H = (A - B)^{-1}B$
Proof:

$$Hx = \lambda x \Longrightarrow (A-B)^{-1}Bx = \lambda x \Longrightarrow Bx = \lambda(A-B)x \Longrightarrow Bx = \frac{\lambda}{1+\lambda}Ax$$

$$x^HBx = \frac{\lambda}{1+\lambda}x^HAx \Longrightarrow x^HB^Tx = \frac{\tilde{\lambda}}{1+\tilde{\lambda}}x^HA^Tx, \text{ by taking the complex conjugate of both sides}$$

$$0 < x^HAx - x^HBx - x^HB^Tx = \frac{1-|\lambda|^2}{|1+\lambda|^2}x^HAx, \text{ since } A - B - B^T \text{ SPD}$$

$$0 < x^HAx \Longrightarrow |\lambda| < 1, \text{ since } A \text{ SPD}$$

Convergence: If A SPD, then Gauss-Seidel converges for any x_0
Proof: Choose A SPD with $A = D - L - U$ and $B = -U = -L^T$

$$A - B - B^T = (D - L - U) - (-U) - (-L) = D, \text{ observe } D \text{ SPD}$$

$$H = [(D - L - U) - (-U)]^{-1}(-U) = -(D - L)^{-1}U = -G, \text{ so } \rho(B) = \rho(-G) = \rho(H) < 1$$

8.3 Successive Over-Relaxation (SOR)

Definition: Using the Gauss-Seidel method, we attempt to increase acceleration using $0 < \omega < 2$. The iterative method is $x_{k+1} = x_k + \omega\left[D^{-1}(b + Lx_{k+1} + Ux_k) - x_k\right]$

$$\text{In Gauss-Seidel: } (D - L)x_{k+1} = b + Ux_k \Longrightarrow x_{k+1} = x_k + D^{-1}(b + Lx_{k+1} + Ux_k) - x_k$$

$$x_{k+1} = x_k + \Delta x_{k+1}, \text{ where } \Delta x_{k+1} = D^{-1}(b + Lx_{k+1} + Ux_k) - x_k$$

$$\text{In SOC: } x_{k+1} = x_k + \omega\Delta x_{k+1} = x_k + \omega(D^{-1}(b + Lx_{k+1} + Ux_k) - x_k)$$

SOR is itself a splitting method with $A = D - L - U$, $M = \frac{1}{\omega}D - L$ and $N = (\frac{1}{\omega} - 1)D + U$
Convergence: If A SPD, then SOR converges for any $\omega \in (0, 2)$
Proof: In the Householder-John Theroem, choose $A_{HJ} = \omega A$ and $B_{HJ} = (\omega - 1)D - \omega U$

$$A \text{ SPD and } A_{HJ} - B_{HJ} - B_{HJ}^T = (2 - \omega)D \text{ SPD if } 0 < \omega < 2$$

$$\text{Then } (A_{HJ} - B_{HJ})^{-1}B_{HJ} = (D - \omega L)^{-1}((\omega - 1)D - \omega U) = G_{SOR}$$

$$\text{Then } \rho(G_{SOR}) = \rho((A_{HJ} - B_{HJ})^{-1}B_{HJ}) = \rho(H) < 1$$

Also, for $\omega \notin (0, 2)$ there exists x_0 s.t. SOR will not converge
Proof:

$$det(G) = det(M^{-1}N) = \frac{det((1-\omega)D + \omega U)}{det(D - \omega L)} = \frac{\prod_i (1-\omega)d_{ii}}{\prod_i d_{ii}} = (1-\omega)^n, \text{ since } L, U \text{ triangular with 0 on diagonal}$$

$$det(G) = \prod_{\lambda_i \in \lambda(G)} \lambda_i = (1-\omega)^n \Longrightarrow |\lambda_{max}(G)| \geq |1-\omega| \Longrightarrow \text{Convergence only when } |1-\omega| < 1$$

8.4 Chebyshev Semi-iterative Method

The most efficient splitting method, but requires us knowing the interval containing $\lambda(A)$. It also uses knob, ω, but we can choose to update this at each step
Definition: With $A = M - N$, $G = M^{-1}N$, and $\omega_k \in \mathbb{R}$, the iterative method is

$$x_{k+1} = x_k + \omega_k((M^{-1}b + Gx_k) - x_k) = x_k + \omega_k M^{-1}(b - Ax_k)$$

$$\text{With } e_k = (I - \omega_{k-1}M^{-1}A)e_{k-1} = \left(\prod_{i=0}^{k-1}(I - \omega_i M^{-1}A)\right)e_0$$

We can minimize error e_k with $\left\|q_k(M^{-1}A)e_0\right\|_2$, where $q_k(x) = (1 - \omega_{k-1}x)\ldots(1 - \omega_0 x)$.

$$\|e_0\|_2 = \left\|q_k(M^{-1}A)e_0\right\|_2 \leq \max_\lambda q_k(\lambda)\,\|e_0\|_2$$

Recalling $q_k(M^{-1}A) = Xq_k(\Lambda)X^{-1}$ for $M^{-1}A = X\Lambda X^{-1}$ with each diagonal element in $q_k(\Lambda) = q_k(\lambda_i)$

$$\|e_k\|_2 \leq \max_\lambda q_k(\lambda)\,\|e_0\|_2 \leq \frac{\|e_0\|_2}{T_k\left(\frac{\beta+\alpha}{\beta-\alpha}\right)}, \text{ where } |q_k(x)| \leq \frac{1}{T_k\left(\frac{\beta+\alpha}{\beta-\alpha}\right)}$$

9 Iterative Krylov methods for solving linear systems

Motivation: We can look for solutions to $Ax = b$ in the increasing dimensions of the Krylov subspace.
At a given step in splitting methods we have

$$x_k = b + (I - A)x_{k-1} = x_{k-1} + b - Ax_{k-1}, \text{ for } M = I$$

Unrolling the iterations, we see x_k is a linear combination of $\{b, Ab, \ldots, A^{k-1}b\}$, a Krylov Subspace of A. We can write

- $x_k = Q_k y$ for Q_k the orthonormal basis of $\mathcal{K}_k(A, b, k)$
- $r_k = b - Ax_k = b - AQ_k y$
- Minimizing $r_k \iff r_k \perp \mathcal{K}(A, b, k)$.

9.1 Conjugate Gradient

Krylov method for SPD matrices. Error, $x - Q_k y$ minimized in A norm. Residual, r_k, minimized in A^{-1} norm. O(n) per iteration.

9.1.1 Naive approach to CG

$$\text{Minimize } \|r_k\|^2_{A^{-1}} = (b - AQ_k y)^T A^{-1}(b - AQ_k y) = b^T x - 2y^T Q_k^T b + y^T Q_k^T A Q_k y$$
$$\frac{d}{dy}\left(b^T x - 2y^T Q_k^T b + y^T Q_k^T A Q_k y\right) = 0 \Longrightarrow Q_k^T A Q_k y = Q_k^T b, \text{ minimizes y}$$

And the iterative process becomes

- Construct Q_k from Lanczos process (O(n))
- Compute y from $Q_k^T A Q_k y = Q_k^T b = \|b\|_2 e_1$ (O(k), since H_k assembled in Lanczos)
- Compute $x_k = Q_k y$ (O(kn), the expensive step we'll try to simplify)

9.1.2 Efficient approach to CG

Search directions: We increase efficiency by working with search directions, $\Delta x_k = x_{k+1} - x_k$, instead of x_{k+1} directly
Search directions, $\Delta x_k, \Delta x_l$ are A-conjugate: $(\Delta x_k)^T A \Delta x_l = 0$ for $k \neq l$
Proof:

$$r_k - r_{k+1} = (b - Ax_k) - (b - Ax_{k+1}) = Ax_{k+1} - Ax_k = A\Delta x_k$$
$$\text{Since } r_k, r_{k+1} \perp Q_k \Longrightarrow A\Delta x_k \perp Q_k$$
$$\Delta x_l = x_{l+1} - x_l \in \mathcal{K}_{l+1} \text{ and } \Delta x_k = x_{k+1} - x_k \in \mathcal{K}_{k+1} \Longrightarrow \text{For } l < k, A\Delta x_k \perp \Delta x_l \Rightarrow \Delta x_l^T A \Delta x_k = 0$$
$$\Delta x_l^T A \Delta x_k = (\Delta x_l^T A^T)\Delta x_k \Longrightarrow A\Delta x_l \perp \Delta x_k, \text{ since } A \text{ SPD}$$
$$\therefore (\Delta x_k)^T A \Delta x_l = 0 \text{ for } k \neq l$$

We can work with search directions directly using the following equalities

$$\Delta x_k = \mu_{k+1} p_{k+1} \qquad p_{k+1} = r_k + \sum_{l=1}^{k} \tau_{lk} p_l$$

Determined by

$$p'_{k+1} \in span\{r_0, \ldots, r_k\} = span\{p_1, \ldots, p_k, r_k\} \Longrightarrow p'_{k+1} = \alpha_k r_k + \sum_{l=1}^{k} \tau'_{lk} p'_l \text{ for some } \alpha_k, \tau'_{lk}$$

$$p_{k+1} = r_k + \sum_{l=1}^{k} \tau_{lk} p_l, \text{ setting } \alpha_k = \mu_{k+1}, \;\; p_{k+1} = \frac{1}{\mu_{k+1}} p'_{k+1}, \text{ and } \tau_{lk} = \frac{\mu_l}{\mu_k} \tau'_{lk}$$

Let $p'_k = \Delta x_{k-1}$. We rely on several properties to achieve the above

- **Property:** when $x_{k+1} = x_k \Rightarrow r_{k+1} = r_k \Rightarrow r_k = 0$ since $r_k \in \mathcal{K}_{k+1} \perp r_{k+1}$
- **Property:** $span\{p'_1, \ldots, p'_l\} = \mathcal{K}_l$

- $x_k, x_{k-1} \in \mathcal{K}_k \rightarrow p'_k \in \mathcal{K}_k$
- $x_k \neq x_{k-1} \rightarrow \Delta x_{k-1} \neq 0 \rightarrow p'_k \notin \mathcal{K}_{k-1}$
- $p'_k \in \mathcal{K}_k$ and $p'_k \notin \mathcal{K}_l$ for $l < k \Longrightarrow span\{p'_1, \ldots, p'_l\} = \mathcal{K}_l$

- **Property:** $span\{r_0, \ldots, r_{l-1}\} = \mathcal{K}_l$
  - $b, Ax_{k-1} \in \mathcal{K}_k \Longrightarrow r_{k-1} = b - Ax_{k-1} \in \mathcal{K}_k$
  - $r_{k+1} \neq r_k \Longrightarrow r_{k-1} \notin \mathcal{K}_{k-1}$
  - $r_{k-1} \in \mathcal{K}_k$ and $r_{k-1} \notin \mathcal{K}_l$ for $l < k \Longrightarrow span\{r_0, \ldots, r_{l-1}\} = \mathcal{K}_l$
- **Property:** $span\{r_0, \ldots, r_{l-1}\} = \mathcal{K}_l = span\{p'_1, \ldots, p'_l\}$
- **Property:** $\mu_{k+1} = \frac{r_k^T r_k}{p_{k+1}^T A p_{k+1}}$

$$
\begin{aligned}
&r_k - r_{k+1} = A\Delta x_{k+1} = \mu_{k+1} A p_{k+1} \\
&p_{k+1}^T r_k = \mu_{k+1} p_{k+1}^T A p_{k+1}, \text{ since } -p_{k+1}^T r_{k+1} = 0 \\
&\mu_{k+1} = \frac{p_{k+1}^T r_k}{p_{k+1}^T A p_{k+1}} = \frac{r_k^T r_k}{p_{k+1}^T A p_{k+1}}, \text{ since } p_{k+1} = r_k + \tau_k p_k \Longleftrightarrow r_k^T p_{k+1} = r_k^T r_k
\end{aligned}
$$

- **Property:** $\tau_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$

$$
\begin{aligned}
&p_{k+1} = r_k + \sum_{l=1}^{k} \tau_{lk} p_l \\
&p_k^T A p_{k+1} = p_k^T A r_k + \sum_{l=1}^{k} \tau_{lk} p_k^T A p_l \Longleftrightarrow 0 = p_k^T A r_k + \tau_{kk} p_k^T A p_k \\
&\tau_k = \frac{-p_k^T A r_k}{p_k^T A p_k} = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}, \text{ since } A p_k = \frac{1}{\mu_k}(r_{k+1} - r_k) \Longleftrightarrow r_k^T A p_k = \frac{r_k^T r_k}{\mu_k} \text{ with } \mu_k = \frac{r_{k-1}^T r_{k-1}}{p_k^T A p_k}
\end{aligned}
$$

9.1.3 Key orthogonality results

- $r_k \perp Q_k$, by construction of r_k to minimize $\|r_k\|_{A^{-1}}$
- $r_k \perp r_l$ for $l \neq k$, since $span\{r_0, \ldots, r_{l-1}\} = \mathcal{K}_l$
- $p_k \perp A p_l$ for $l \neq k$, since $r_k, r_{k-1} \perp \mathcal{K}_{k-1}$ and $A p_k = \frac{1}{\mu_k} A \Delta x_k = \frac{1}{\mu_k} A(r_k - r_{k-1})$ then $A p_k \perp \mathcal{K}_{k-1}$, but for $l < k$, $p_l = \frac{1}{\mu_l}(x_l - x_{l-1}) \in \mathcal{K}_{k-1}$ so $p_l \perp A p_k$. We can get same result for $l > k$ using A SPD properties.
- $r_k \perp p_l$ for $l \leq k$, TODO
- $r_k \perp A p_l$ for $l < k$ since $p_l^T A r_k = (A p_l)^T r_k$ with $A p_l = \frac{1}{\mu_l}(x_l - x_{l-1}) \in \mathcal{K}_{l+1}$ and $r_k \perp \mathcal{K}_{l+1}$ for $l + 1 < k + 1 \Leftrightarrow l < k$

9.1.4 Congugate Gradient algorithm

(i) Choose some x_0 (could be $x_0 = 0$)

(ii) $r_0 = b - Ax_0,\ p_0 = 0,\ k = 1$

(iii) While $r_{k-1} \neq 0$

$$
\tau_{k-1} = \frac{\|r_{k-1}\|_2^2}{\|r_{k-1}\|_2^2}, \quad p_k = r_{k-1} + \tau_{k-1} p_{k-1}, \quad \mu_k = \frac{\|r_{k-1}\|_2^2}{p_k^T A p_k}, \quad x_k = x_{k-1} + \mu_k p_k, \quad r_k = r_{k-1} - \mu_k A p_k
$$

$k \leftarrow k + 1$

(iv) Return x_{k-1}

9.2 GMRES

Krylov method for general matrices, Generalized Minimal Residual Method (GMRES). Error, $x - x_k = x - Q_k y$ minimized in $A^T A$ norm. Residual, r_k, minimized in 2-norm. O(kn) per iteration.

$$\|x - x_k\|_{A^T A}^2 = (x - x_k)^T A^T A (x - x_k) = (b - Ax_k)^T (b - Ax_k) = \|b - Ax_k\|_2^2 = \|r_k\|_2^2$$

We can use least squares methods to minimize y in $\|r_k\|_2^2$:

$$\begin{aligned}\|r_k\|_2^2 = \|b - Ax_k\|_2^2 &= \left\|Q_{k+1}Q_{k+1}^T b - Q_{k+1}Q_{k+1}^T A Q_k y\right\|_2 \\ &= \left\|Q_{k+1}(Q_{k+1}^T b - Q_{k+1}^T A Q_k y)\right\|_2 = \left\|Q_{k+1}^T b - Q_{k+1}^T A Q_k y\right\|_2 = \left\|\|b\|_2 e_1 - H_k y\right\|_2\end{aligned}$$

Lastly, use givens rotations to make H_k upper triangular, and then find solution y_k.

9.3 Preconditioning

The condition number and distribution of eigenvalues of A influence convergence in both CG and GMRES, with clustered eigenvalues leading to faster convergence. Matrices can be conditioned, with the tradeoff being i) cost of applying a matrix preconditioner vs ii) savings from reduced iterations

- Left preconditioning: $(M_1 A)x = M_1 b$
- Right preconditioning: $(AM_2)z = b$, and solve $x = M_2 z$
- Symmetric preconditioning: $(M_1 A M_2)z = M_1 b$, and solve $x = M_2 z$

9.3.1 CG predonditioning

Choose M SPD and define $C^2 = M$. CG preconditioning follows symmetric preconditioning, solving $CACy = Cb$ and $Cy = x$, but with tricks that only require computing MA.

- MA is similar to CAC (suffice to show $\lambda(MA) = \lambda(CAC)$)

$$\begin{aligned}MAx = \lambda x \Leftrightarrow CCACz = \lambda Cx \Leftrightarrow CACz = \lambda z, \text{ for } x = Cz \\ CACy = \lambda y \Leftrightarrow MACy = \lambda Cy \Leftrightarrow MAx = \lambda x, \text{ for x = Cy}\end{aligned}$$

- $MAx = Mb \Longleftrightarrow (CAC)C^{-1} = Cb$

$$MAx = Mb \Leftrightarrow CCAx = CCb \Leftrightarrow CACC^{-1}x = Cb$$

- $C^{-1}x \in \mathcal{K}(CAC, Cb, k) \Longleftrightarrow x \in \mathcal{K}(MA, Mb, k)$

$$\begin{aligned}C^{-1}x \in \mathcal{K}(CAC, Cb, k) &\Longleftrightarrow C^{-1}x = \alpha_0 Cb + \sum_{i=1}^{k} \alpha_i (CAC)^i Cb = \alpha_0 Cb + \sum_{i=1}^{k} \alpha_i CA(MA)^{i-1} CCb \\ &\Longleftrightarrow x = \alpha_0 Mb + \sum_{i=1}^{k} \alpha_i MA(MA)^{i-1} Mb = \sum_{i=1}^{k} \alpha_i (MA)^i Mb \Longleftrightarrow x \in \mathcal{K}(MA, Mb, k)\end{aligned}$$

10 Direct methods

Direct methods are an alternative to iterative methods for solving linear systems, and take advantage of sparse data formatting.

10.1 Matrix storage

$$A = \begin{bmatrix} 4.1 & 0 & 2.9 & 0 \\ 1.2 & -0.3 & 0 & -0.1 \\ 0 & 7.2 & 9.2 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

Coordinate format (COO): (i, j, a_{ij}), e.g. $COO(A) = (1,1,4.1), (1,3,2.9), \ldots, (4,4,1.0)$
Compressed Sparse Row (CSR): $\{nzval, colval, rowptr\}$, e.g.,

$$CSR(A) = \begin{cases} nzval & = [4.1, 2.9, \ldots, 1.0] \text{ (nonzero values)} \\ colval & = [1, 3, \ldots, 4] \text{ (column values)} \\ rowptr & = [1, 3, 6, 8, 9] \text{ (index of nzval that starts each row)} \end{cases}$$

Compressed Sparse Column (CSC): $\{nzval, rowval, colptr\}$, e.g.,

$$CSR(A) = \begin{cases} nzval & = [4.1, 1.2, \ldots, 1.0] \text{ (nonzero values)} \\ rowval & = [1, 2, \ldots, 4] \text{ (row values)} \\ colptr & = [1, 3, 5, 7, 9] \text{ (index of nzval that starts each column)} \end{cases}$$

Example matrix-vector product, Ax:

```
for i=1:A.m
    y[i] = 0.0
    for k=A.rowptr[i]:A.rowptr[i+1]-1
        y[i] += A.nzval[k]*x[A.colval[k]]
    end
end
```

Conceptualizing the graph of A, G_A: Edge $j \to i$ exists if $a_{ij} \neq 0$

10.2 Solving triangular sparse systems

Dense b: For $Lx = b$ with sparse lower triangular matrix, L, and dense vector, b: $x_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} x_j \right)$

```
for i=1:A.m
    x[i] = b[i]
    for k=A.rowptr[i]:A.rowptr[i+1]-2
        x[i] -= A.nzval[k]*x[A.colval[k]]
    end
    x[i] /= A.nzval[A.rowptr[i+1]-1]
end
```

Sparse b: We can improve on this code when b sparse by first determining the nonzero pattern of x (nontrivial). Once we have the nonzero pattern of x, we can run the above code on sparse x. Observing the equation above we see the following rules for the nonzero pattern in x

$$x_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} x_j \right) \Longrightarrow \begin{cases} b_i \neq 0 & \Rightarrow x_i \neq 0 \\ \exists j < i \text{ s.t. } l_{ij} \neq 0 \text{ and } x_j \neq 0 & \Rightarrow x_i \neq 0 \end{cases}$$

Using **graph theory**, the statements above are equivalent to saying

- X is the set of nodes reachable from B, the set of nodes for which $b_i \neq 0$, on G_L
- The reach of node j is all i for which there is a path $j \rightsquigarrow i$
- Use recursive backtracking ("depth-first search") to determine set X.

10.3 Cholesky factorization

Direct methods for computing sparse Cholesky factorizations, $A = LL^T$

10.3.1 Up-looking Cholesky factorization

Starting with known L', an upper $k \times k$ block of L, we can use the **up-looking Cholesky factorization** to determine the $(k+1)^{st}$ row/column of L

(i) For A SPD, initialize $L' = \sqrt{a_{11}}$

(ii) For $k = 2, \ldots, n$, write top $k \times k$ block as $\begin{bmatrix} L' & 0 \\ x^T & w \end{bmatrix} \begin{bmatrix} L'^T & x \\ 0 & w \end{bmatrix} = \begin{bmatrix} A' & b \\ b^T & a \end{bmatrix}$

Solve $L'x = b$ for x, compute $w = \sqrt{a - x^T x}$, and update/return $L' = \begin{bmatrix} L' & 0 \\ x^T & w \end{bmatrix}$

Notice $L'_{k-1} l_k^T = a_k^T$, so we can leverage the same nonzero pattern relationship as above:

$$l_{ij} = \frac{1}{l_{ii}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{jk} l_{ik} \right) \Longrightarrow \text{For } j < i \begin{cases} a_{ij} \neq 0 & \Rightarrow l_{ij} \neq 0 \\ \exists k < j \text{ s.t. } l_{jk} \neq 0 \text{ and } l_{ik} \neq 0 & \Rightarrow l_{ij} \neq 0 \end{cases}$$

10.3.2 Elimination trees

The above procedure takes A and produces G_L, wich reflects the nonzero pattern of L in $A = LL^T$. As a special property of A SPD, we can construct a more simple **Elimination Tree**, E_T, a graph of the first nonzero off-diagonal element in each column of L.

The elimination tree has the same reach as G_L:

- For any j, let i' be the smallest row index s.t., $L_{i'j} \neq 0$. We show removing $j \to i$ from G_L, with $i > i'$ does not change the reach of G_L
- **Proof:** Consider $k \in Reach(j)$ and how/if it changes after we remove edge $j \to i$:
  - If i wasn't in the path of $j \rightsquigarrow k$, then the reach is unchanged
  - If i was in path of $j \to i \rightsquigarrow k$, the reach is still unchanged because $l_{ij} \neq 0$ and $l_{i'j} \neq 0 \Longrightarrow l_{i'i} \neq 0$, so a path, $j \rightsquigarrow k$, can still be constructed: $j \to i' \to i \rightsquigarrow k$

10.3.3 Worked example of $A \longrightarrow E_T \longrightarrow G_L$:

$A \Longrightarrow E_T$

$$A = \begin{bmatrix} X & - & X & X & X \\ - & X & - & - & X \\ X & - & X & - & - \\ X & - & - & X & - \\ X & X & - & - & X \end{bmatrix} \longrightarrow E_T = \begin{cases} i=1: & a_{11} \neq 0 \Rightarrow ① \text{ (only looking at or below the diagonal } \forall i) \\ i=2: & ① \mid a_{22} \neq 0 \Rightarrow ② \\ i=3: & a_{31} \neq 0 \Rightarrow ① \to ③ \mid ② \\ i=4: & a_{41} \neq 0 \Rightarrow ① \to ③ \to ④ \mid ② \\ i=5: & a_{51} \neq 0 \Rightarrow ① \to ③ \to ④ \to ⑤ \leftarrow ② \Leftarrow a_{52} \neq 0 \text{ (final } E_T) \end{cases}$$

$E_T \longrightarrow G_L$: (Relying on $L'_{k-1} l_k^T = a_k^T$)

$$\begin{cases} i=1: l_{11}l_{21} = a_{12} \Longleftrightarrow [X][l_{21}] = [0] & a_{12} = 0 \Rightarrow l_{21} = 0 \\ i=2: \begin{bmatrix} X & 0 \\ 0 & X \end{bmatrix} \begin{bmatrix} l_{31} \\ l_{32} \end{bmatrix} = \begin{bmatrix} a_{13} \\ a_{23} \end{bmatrix} = \begin{bmatrix} X \\ 0 \end{bmatrix} & a_{13} \neq 0 \Rightarrow l_{31} \neq 0 \\ i=3: \begin{bmatrix} X & 0 & 0 \\ 0 & X & 0 \\ X & 0 & X \end{bmatrix} \begin{bmatrix} l_{41} \\ l_{42} \\ l_{43} \end{bmatrix} = \begin{bmatrix} a_{14} \\ a_{24} \\ a_{34} \end{bmatrix} = \begin{bmatrix} X \\ 0 \\ 0 \end{bmatrix} & a_{14} \neq 0 \Rightarrow l_{41} \neq 0 \Longrightarrow l_{31} \neq 0 \Rightarrow l_{43} \neq 0, \text{ since 3 in reach of } 1 \in G_L \\ i=4: \begin{bmatrix} X & 0 & 0 & 0 \\ 0 & X & 0 & 0 \\ X & 0 & X & 0 \\ X & 0 & X & X \end{bmatrix} \begin{bmatrix} l_{51} \\ l_{52} \\ l_{53} \\ l_{54} \end{bmatrix} = \begin{bmatrix} a_{15} \\ a_{25} \\ a_{35} \\ a_{45} \end{bmatrix} = \begin{bmatrix} X \\ X \\ 0 \\ 0 \end{bmatrix} & a_{15} \neq 0 \Rightarrow l_{51} \neq 0 \Longrightarrow l_{53}, l_{54} \neq 0 \text{ since } 3,4 \text{ in reach of } 1 \in G_L \\ & a_{25} \neq 0 \Rightarrow l_{52} \neq 0 \text{ (but nothing else in reach of } 2 \in G_L) \end{cases}$$

$$G_L = \begin{bmatrix} X & - & - & - & - \\ - & X & - & - & - \\ X & - & X & - & - \\ X & - & X & X & - \\ X & X & X & X & X \end{bmatrix}$$