CME302 class notes

Erich Trieschman

2021 Fall quarter

1 Linear algebra review

1.1 Vector products

There are two vector products that are helpful for a lot of linear algebra. The **inner product**, also known as the dot product, results in a scalar. It is the inner sum of two vectors of the same length: $x^T y = \sum x_i * y_i$

- $x^T y = \|x\|_2 \|y\|_2 \cos\theta$
- $x^T y = 0 \Leftrightarrow x \perp y$

And 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

Measures of the length of vectors and matrices. All norms 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)
- $\|x - y\|_x \geq \|x\|_x - \|y\|_x$ (another Triangle inequality)

1.2.1 Vector norms

In machine learning applications, the selection of the norm can give you solutions with different properties. Many theorems and properties work out nicely when we choose the 2-norm. Types of **vector norms**, $x \in \mathbb{R}^n$

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

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 2-norm: $\|Ay\|_p \leq \|A\|_p \|y\|_p$

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

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

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

One way to think about the determinant of a matrix is that it represents how the volume of a hypercube is transformed by the matrix. And a few properties of the determinant

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

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

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 $(I + V^TA^{-1}U)$

Proof: begin with the inverse of the LHS multiplied by the RHS: $(A+UV^T)(A^{-1} - A^{-1}U(I+V^TA^{-1}U)^{-1}V^TA^{-1})$. 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

$$\text{Show: } AB = a_1b_1^T + a_2b_2^T + ... + a_nb_n^T, A, B \in \mathbb{R}$$

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

$$a_1b_1^T = \begin{bmatrix} a_{11}b_{11} & \dots & a_{11}b_{1n} \\ \vdots & & \vdots \\ a_{n1}b_{11} & \dots & a_{n1}b_{1n} \end{bmatrix} \Rightarrow \sum_{i=1}^{n} a_ib_i^T = \begin{bmatrix} \sum a_{1i}b_{i1} & \dots & \sum a_{1i}b_{in} \\ \vdots & & \vdots \\ \sum a_{ni}b_{i1} & \dots & \sum a_{ni}b_{in} \end{bmatrix} \Rightarrow AB$$

1.5 Orthogonal matrices

An orthogonal matrix, Q is a matrix whose columns are orthonormal. That is

- $q_i^Tq_j = 1$ for $i = j$
- $q_i^Tq_j = 0$ for $i \neq j$

Equivalently, $Q^TQ = I$. For square matrices, $Q^TQ = QQ^T = I$

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

$$v = \frac{y^Tx}{y^Ty}y$$

Which can be interpreted as the portion of x in the direction of y (y^Tx), times the direction of y, divided by the length of y twice ($y^Ty = \|y\|_2^2$), 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 $P = I - \beta vv^T$, with $\beta = \frac{2}{v^Tv}$, and v the vector orthogonal to the line/plane of reflection
- It can be shown that $Px = x \Leftrightarrow v^Tx = 0$. These x are called the "fixed points" of P

1.7 Symmetric Positive Definite (SPD) Matrices

For A, SPD

- $A = A^T$
- $x^T Ax > 0 \; \forall x \neq 0$
- $a_{ii} > 0$
- $\lambda(A) \geq 0$
- for B nonsingular, $B^T AB$ is also SPD

When proving properties of SPDs, use the following tricks

- Multiply by e_i since $e_i \neq 0$
- Use matrix transpose property, $x^T A^T = (Ax)^T$ to rearrange formulas

1.7.1 $B^T AB$ is also SPD

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

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

1.8 Eigenvalues

Observe by definition

$$\begin{aligned} Ax &= \lambda x \\ Ax - \lambda x &= 0 \\ (A - \lambda I)x &= 0 \end{aligned}$$

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 the value λ_i appears as an eigenvalue of A
e.g., for characteristic equation $p(x) = (x-2)^3(x-1)^2$, $\lambda = 2$ has algebraic multiplicity of 3, and $\lambda = 1$ has algebraic multiplicity of 2
The **geometric multiplicity** of an eigenvalue, λ_i, is the dimension of the space spanned by the eigenvectors of λ_i
Other eigenvalue properties

- $\lambda(A) = \lambda(A^T)$

1.8.1 Determinants and trace

$$det(A) = \prod_{i=1}^{n} \lambda_i, \text{ Intuition: the lower triangular 0s cancel everything, but the values in the diagonal}$$

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

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:

$$\begin{aligned}
Ax &= \lambda x \\
(A - \lambda I)x &= 0 \\
\sum_{j \neq i} a_{ij}x_j + (a_{ii} - \lambda)x_i &= 0, \ \forall i \in \{1, \ldots, n\} \\
\text{Choose } i \ s.t. |x_i| &= \max_i |x_i| \\
|(a_{ii} - \lambda)| &= |\sum_{j \neq i} \frac{a_{ij}x_j}{x_i}| \leq \sum_{j \neq i} |\frac{a_{ij}x_j}{x_i}| \text{ , by triangle inequality} \\
|(\lambda - a_{ii})| &\leq \sum_{j \neq i} |a_{ij}|, \text{ since } |\frac{x_j}{x_i}| \leq 1
\end{aligned}$$

2 Matrix Decompositions

2.1 Schur Decomposition

For any $A \in \mathbb{C}^{n \times n}$

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

When $A \in \mathbb{R}^{n \times n}$, the Real Schur Decomposition becomes

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

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

$$A = Q\Lambda Q^T, Q \text{ orthogonal}$$

For A unitarily diagonalizable ($\Leftrightarrow$ normal: $A^HA = AA^H$), A can be decomposed as below. When A complex Hermitian ($A = A^H$), $\Lambda \in \mathbb{R}$

$$A = Q\Lambda Q^H, Q \text{ unitary}$$

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

$$\begin{aligned}
A &= U\Sigma V^H \\
A &= U\Sigma V^T \text{ when } A \in \mathbb{R}^{m \times n}, \text{ with } U, V, \Sigma \in \mathbb{R}
\end{aligned}$$

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

$$
\begin{aligned}
A^TA \text{ symmetric} \Rightarrow \exists\, Q \text{ orthogonal and } \Lambda \text{ diagonal matrix of } \lambda_i \text{ s.t.,}& \\
A^TA &= Q\Lambda Q^T \\
Q^TA^TAQ &= Q^TQ\Lambda Q^TQ \\
(AQ)^T(AQ) &= \Lambda, \text{ note } AQ \text{ is orthogonal, but not scaled to 1. Instead, each row is} \\
&\quad \text{scaled to the eigenvalue in that row: } \lambda_i = \|Aq_i\|_2^2
\end{aligned}
$$

When A is full rank,

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

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

$$
\text{Start with } AQ = \begin{bmatrix} | & & | & | & & | \\ r_1 & \ldots & r_r & 0 & \ldots & 0 \\ | & & | & | & & | \end{bmatrix}
$$

$$
A = AQD^{-1}DQ^T, \text{ where } D = \begin{bmatrix} \sqrt{\lambda_1} & \ldots & 0 & & \\ \vdots & & \vdots & & \\ 0 & \ldots & \sqrt{\lambda_r} & & \\ \vdots & & \vdots & I & \\ 0 & \ldots & 0 & & \end{bmatrix} \text{ (observe this matrix has inverse, } D^{-1})
$$

$A = U\Sigma V^T$, 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, $\sigma_i = |\lambda_i|$
- The eigenvalues of A^TA and AA^T are the squares of the singular values of A, $\sigma_1^2, \ldots, \sigma_n^2$
- 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 $AA^Tu_i = \sigma_i^2 u_i$
- When A orthogonal, $\sigma_1 = \cdots = \sigma_n = 1$
- **Condition number**, $\kappa(A) = \|A\|_2, \left\|A^{-1}\right\|_2 = \frac{\sigma_1}{\sigma_n}$

3 Error analysis

3.1 Floating point arithmetic

Motivation: Values stored in a computer are floating type, which means the digits of the number are stored, and then separately the location of the decimal place is stored. This means that when adding or subtracting numbers with limited overlap in where the digits lie can lead to truncation and accuracy errors.

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,

- start by writing $fl(x)$ and x using floating point equations
- show that the difference between these numbers is bounded by the smallest bit that can be represented by $fl(x)$.
- The $\frac{1}{2}$ enters the equation as a bound on the selection of the last digit of $fl(x)$ to approximate x: it cannot be greater than $\frac{1}{2}$ of the digit away, or a closer digit could have been chosen.

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? But this is difficult to compute.

Backward error analysis is an easier error estimation to solve and tries to find the error in A that lets us observe the $\tilde{x}$ observed. More specifically, find $\tilde{E}$ such that $(A + \tilde{E})\tilde{x} = b$. i.e., what is the problem that our algorithm actually solved? The backward error can also be referred to as $\|E\|_p$, and 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 **motivation** for the LU factorization is that it is relatively straightforward computationally to solve linear equations when the matrix is of a triangular form. So, if we can decompose a matrix, A, into a product of a lower triangular matrix, L, and an upper triangular matrix, U, then we can solve an easier problem.

Once we have $A = LU$, to solve $Ax = b$, we can start by solving $Lz = b$, and then $Ux = z$. x, here, is the solution!

4.1 Basic algorithm

In the basic LU factorization algorithm, we construct matrices L and U by iteratively subtracting the outer products of vectors that sequentially "zero-out" the rows and columns of A. We know $LU = l_1u_1^t + \cdots + l_nu_n^t$, and when l_1, u_1^t are from lower/upper respectively, $LU - l_1u_1^t$ 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 $A' \leftarrow A - l_1u_1^T$. 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

Another way to think about the basic LU factorization algorithm is with Gauss transforms. **Guass transformation matrices** are linear transformations that zero out all entries below a certain entry. 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:

$$G_nG_{n-1}\dots G_2G_1A = U$$
$$A = G_1^{-1}\dots G_n^{-1}U$$

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:

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

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

$$A_1 = L_1U_1$$
$$det(A_{k+1}) = det(L_{k+1}U_{k+1})$$
$$det(A_{k+1}) = det(L_{k+1})det(U_{k+1})$$
$$det(A_{k+1}) = det(U_{k+1})$$
$$det(A_{k+1}) = u_{11} * u_{22} * \cdots * u_{kk}$$
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 in each iteration to avoid the numerical issues identified above. There are several pivoting algorithms

- **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, $A = L_1U_1$

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

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 $\Rightarrow B^TAB$ 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 the determinants of the equation above) so

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

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 $l_iU_i^T$, 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, $Q^H Q = I$
- $\|Qx\|_2 = \|x\|_2$

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:

$$\begin{aligned} A &= QR \\ Q^T A &= R \\ R^T Q^T A &= R^T R \\ (QR)^T A &= R^T R \\ A^T A &= R^T R \end{aligned}$$

We now have a matrix, $A^T A$ that can be written of the form $R^T R$, which is the structure of the Cholesky factorization. Suffice to show that $A^T A$ 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 $AR^{-1} = Q$). $A^T A$ SPD:

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

Positive definite: for $x \neq 0$,

$$x^T A^T A x = (Ax)^T (Ax) = (QRx)^T (QRx) = x^T R^T Q^T Q R x = (Rx)^T (Rx)$$

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 an algorithm to construct matrices Q and R for the QR factorization of A. The Householder reflection relies on the principles of reflection matrices, which (turns out) are easier to construct than rotation matrices.

5.2.1 Householder reflection algorithm

The high-level approach of the algorithm is to construct Q^T for each column in A that projects it onto a corresponding column of an upper right triangular matrix, R. It's easiest to see this in the case of column a_1:

- Want Q_1^T such that $Q_1^T a_1 = r_1$, where $r_1 = \pm \|a_1\|_2 e_1$ (since Q^T is orthogonal)
- Put differently, want 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

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

- Notice that multiplying Px is the same as taking the vector x and subtracting $\frac{2vv^T}{v^T v}x$ from it, where $\frac{2vv^T}{v^T v}x$ is twice the projection of x onto v. When you draw this out, you'll see this vector subtraction is the same thing as a reflection about the plane defined by v
- In our case where we want to reflect a onto $\|a\|_2 e_1$, $a + \|a\|_2 e_1$ is the line of reflection. Perpendicular to this line is $a - \|a\|_2 e_1$, the vector that defines the line of reflection

- In some cases where the other entries in a are much smaller than a_1, it may be advantageous to choose to project onto $-\|a\|_2 e_1$ instead of $-\|a\|_2 e_1$ to avoid roundoff errors from subtracting small numbers. In this case, we choose $v = a + \|a\|_2 e_1$.

Aside: The fixed points of a reflection, P, are the points that 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

The Givens transformation is a much more precise algorithm for creating an upper triangular matrix, R, through an orthogonal transformation, Q^T, of A, $Q^T A = R$. While the householder reflection is useful for operating on dense matrices, if we are presented with a sparse matrix, the sequential reflections of the House transformation will create more work for us.

For example, consider a matrix, A, that has a dense upper triangular portion, and a diagonal row of nonzero entries just below the diagonal. In this case, the **Givens transformation** will allow us to zero our these limited rows 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

The Householder and Givens transformations produce square $Q \in \mathbb{R}^{n \times n}$ matrices. However, if $A \in \mathbb{R}^{m \times n}$ is tall and thin, with $m >> n$, then we 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: $A = QR = q_1 r_1^T + \dots q_m r_m^T$.

The algorithm proceeds as follows

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

Repeat for A', the construction of r_{11}, q_1, r_{1j}

5.5 QR factorization to solve least-squares problems

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

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, x:

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

We can use Cholesky method for fast and accurate solve of this method since A^TA is SPD. This method can run into issues when A is poorly conditioned. 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:

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

5.5.3 SVD for rank-deficient A

It may be that A is not full rank. When this is the case, we can get infinite solutions (a line of points that satisfy $argmin_x \|Ax-b\|_2$). To choose x, we add the additional criteria $\min_x \|x\|_2$ to our original objective function of $argmin_x \|Ax-b\|_2$.

We can use the "thin" version of the Singular Value Decomposition to solve this! By thin we mean, for $A \in \mathbb{R}^{m\times n}$ with rank, 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

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

Observe $\min_x \|x\|_2$ the $x \perp N(A)$, the shortest vector between $N(A)$ and the vector/plane of solutions to $argmin_x \|Ax-b\|_2$. This value it turns out 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 $\implies A^k = X\Lambda^kX^{-1}$

$$\begin{aligned} A^k &= \sum_{i=1}^{n} \lambda_i^k x_i y_i^T \text{ where } Y = X^{-1} \\ A^k &\approx \lambda_1^k x_1 y_1^T \text{ since } \lambda_1 > \lambda_2 \\ A^kq &\approx \lambda_1^k x_1 y_1^T q = \lambda_1^k (y_1^Tq)x_1, \text{ since } y_1^Tq \text{ is a scalar. Observe } A^kq \sim\parallel x_1 \end{aligned}$$

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 $z_k \sim\parallel q_k \to z_k^Tx_k = \sqrt{\|z\|_2 \|x\|_2}$
3. $q_{k+1} = \dfrac{z_k}{\|z_k\|_2} = \dfrac{A^kq_0}{\|A^kq_0\|_2} \approx (\dfrac{\lambda_1}{|\lambda_1|})^kx_1$

Since $A^k q_0 = Aq_k \approx \lambda_1 x_1$, where $\|x_1\|_2 = 1$ $(WLOG)$ and $q_k \sim\| x_1$, we can solve for λ:

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

Convergence: $O((|\frac{\lambda_1}{\lambda_2}|)^K)$, bounds the error since λ_2 is the second largest term.

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:

$$\begin{aligned}(A - \mu I)^{-1} x &= \lambda x \\ x &= (A - \mu I)x = \lambda A x - \lambda \mu x \\ \lambda A x &= x + \lambda \mu x \\ Ax &= \frac{(1 + \lambda\mu)}{\lambda} x\end{aligned}$$

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 Orthogonal iteration

This process finds all eigenvalues and eigenvectors of A in a single iterative process. We arrive at the final result with three building blocks. First, we show how we can compute each q_i to capture λ_i. Next, we justify a matrix decomposition (based on Schur) that allows us to reveal Q. And lastly, we combine these blocks to show how this can be done at the same time for all of A.

First, consider how orthogonal columns could reveal subsequent eigenvalues. Assume we have used the power iteration to compute q_1. To get q_2

$$\begin{aligned}A^k &= \lambda_1 x_1 y_1^T + \lambda_2 x_2 y_2^T + \dots \\ PA^k &= \lambda_1 P x_1 y_1^T + \lambda_2 P x_2 y_2^T + \dots, \text{ where } P = I - x_1 x_1^T \\ PA^k &= 0 + \lambda_2 P x_2 y_2^T + \dots, \text{ since } Px_1 = Ix_1 - x_1 x_1^T x_1 = x_1 - x_1 = 0\end{aligned}$$

PA can now be used to apply the power iteration to to reveal x_2 and λ_2

The general process is:

- Start with λ_1, q_1 from power iteration
- Build P_2, projector orthogonal to q_1, use power iteration to reveal (λ_2, q_2)
- Build P_3, projector orthogonal to $\{q_1, q_2\}$, use power iteration to reveal (λ_3, q_3)
- Build P_i, projector orthogonal to $\{q_1, \dots, q_{i-1}\}$, use power iteration to reveal (λ_i, q_1)

Now, consider the QR decomposition of X:

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

Observe, the last line is the Schur Decomposition, so

- 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 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

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 $Q_k \to Q$ revealing $A = QR\Lambda R^{-1}Q^H$

Note in each iteration we are calculating $Q_{k+1}^H AQ_k = R_{k+1}$ (This R_k is commonly referred to as T_k)

6.3.1 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:

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

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

6.3.2 Rate of convergence in orthogonal (and QR) iteration

Convergence of a given eigenvalue is dictated the the eigenvalues in the adjacent rows. For row i, the rate of convergence is

$$O(|\frac{\lambda_{i+1}}{\lambda_i}|^k)$$

The span of those i columns of $Q_k, span\{q_1, \cdots, q_i\} \longrightarrow$ the span of those columns of $X, span\{x_1, \cdots, x_i\}$. **Note:** difficulties arrise when $|\frac{\lambda_{i+1}}{\lambda_i}|$ is close to 1.

6.4 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

$$\begin{aligned} A = Q_k T_k Q_k^H &\Longrightarrow T_k = Q_k^H AQ_k \\ A = Q_{k+1}R_{k+1} \text{ and } T_k = U_{k+1}R_{k+1} &\Longrightarrow U_{k+1}^H T_k = Q_{k+1}^H A \\ T_{k+1} = Q_{k+1}^H AQ_{k+1} = Q_{k+1}^H Q_k T_k Q_k^H Q_{k+1} &= U_{k+1}^H T_k U_{k+1} \\ \Rightarrow T_k &= U_{k+1}R_{k+1} \\ \Rightarrow T_{k+1} &= R_{k+1}U_{k+1} \end{aligned}$$

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

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

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

6.4.1 QR iteration on upper Hessenberg

Each QR iteration step of a dense matrix is $O(n^3)$. To reduce flops (floating point operations) in QR iteration, we can first convert A to upper Hessenberg ($H = Q^H AQ$), and proceed with QR iteration on H using Givens rotations with complexity $O(n^2)$:

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

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

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

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

6.4.2 QR iteration with shift

Recall, convergence is dictated by $|(\lambda_{i+1}/\lambda_i)|^k$. When λ_{i+1} is close to λ_i, **QR iteration with shift** helps accelerate 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.

In general, the **QR iteration with shift** process works by shifting the last eigenvalue (smallest in absolute value), updating the shift in each iteration. It works as follows:

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. $R_k U_k + \mu_k I \longrightarrow T_{k+1}$
4. Repeat with T_{k+1}

Observe, this shift preserves the original QR iteration

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

And lastly, another useful speed-up property is called **deflaction**, which is a process that 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)

Proof of $\lambda(H) = \lambda(H_{11}) \cup \lambda(H_{22})$

6.4.3 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 quadradic 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

7.1 Arnoldi process

7.2 Krylov spaces

7.3 Lanczos process