Chapter 2

Solving Linear Systems

2.1 Triangular Matrices and Their Applications

Upper triangular matrices and lower triangular matrices play a critical role in various matrix decomposition methods such as LU decomposition and QR decomposition. They facilitate easier computations for operations such as matrix inversion and determinant calculation due to their triangular structure.

2.1.1 Upper Triangular Matrices

An upper triangular matrix **U** has all non-zero entries above and on its diagonal, and zeros below it:

$$\mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix} \tag{2.1.1}$$

Key properties of upper triangular matrices include:

- The determinant of **U** is the product of its diagonal entries:

$$\det(\mathbf{U}) = \prod_{i=1}^{n} u_{ii} \tag{2.1.2}$$

- An upper triangular matrix is invertible if and only if all its diagonal entries are non-zero. The inverse of an invertible upper triangular matrix is also upper triangular.

Proof: Determinant of an Upper Triangular Matrix

Let $\mathbf{U} = [u_{ij}]$ be an $n \times n$ upper triangular matrix, so that $u_{ij} = 0$ whenever $i > j$. The determinant is given by the Leibniz formula:

$$\det(\mathbf{U}) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} u_{i,\sigma(i)},$$

where S_n is the set of all permutations of $\{1, 2, \ldots, n\}$.

Suppose σ is a permutation with $\sigma \neq \mathrm{id}$ (i.e. not every $\sigma(i) = i$). Assume for contradiction that $\sigma(i) \geq i$ for all i. Then

$$\sum_{i=1}^{n} \sigma(i) \geq \sum_{i=1}^{n} i.$$

However, since σ is merely a reordering of $\{1, 2, \ldots, n\}$, the sum on the left is exactly $\sum_{i=1}^{n} i$. Equality forces $\sigma(i) = i$ for every i, which contradicts the assumption that σ is nontrivial. Thus, there must exist at least one index i with $\sigma(i) < i$.

Because $\mathbf{U}$ is upper triangular, any term $u_{i,\sigma(i)}$ with $i > \sigma(i)$ is zero. Hence, every permutation $\sigma \neq \mathrm{id}$ gives a zero contribution in the sum. Only the identity permutation survives, yielding

$$\det(\mathbf{U}) = \prod_{i=1}^{n} u_{ii}.$$

2.1.2 Solving Systems with Upper Triangular Matrices

Systems of linear equations involving an upper triangular matrix $\mathbf{U}$ can be efficiently solved using the backward substitution method. This process leverages the structure to solve each variable sequentially, starting from the last.

Backward Substitution Method

Consider an upper triangular matrix $\mathbf{U}$ and a vector $\mathbf{b}$ in the system $\mathbf{Ux} = \mathbf{b}$. The matrix $\mathbf{U}$ is defined as:

$$\mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} \tag{2.1.3}$$

and $\mathbf{b}$ is a column vector of size n. The goal is to find the vector $\mathbf{x}$.

The solution is obtained by solving the last equation for the last unknown, then substituting this back into the previous equation to solve for the second last unknown, and repeating this process up to the first equation. The steps are as follows:

1. Initialize x_n:

$$x_n = \frac{b_n}{u_{nn}} \tag{2.1.4}$$

2. For each i from $n-1$ to 1, compute:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} u_{ij}x_j}{u_{ii}} \tag{2.1.5}$$

2.1.3 Example: Solving an Upper Triangular System

Consider the following system where $\mathbf{U}$ is an upper triangular matrix:

$$\mathbf{U} = \begin{bmatrix} 2 & 3 & 0 & 0 \\ 0 & 1 & 4 & 0 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 5 \\ 2 \\ 8 \\ 10 \end{bmatrix} \tag{2.1.6}$$

Using backward substitution:

- Solve for x_4:

$$2x_4 = 10 \implies x_4 = 5 \tag{2.1.7}$$

- Solve for x_3 using x_4:

$$x_3 + 5x_4 = 8 \implies x_3 = 8 - 5 \times 5 \implies x_3 = -17 \tag{2.1.8}$$

- Solve for x_2 using x_3:

$$x_2 + 4x_3 = 2 \implies x_2 = 2 - 4 \times (-17) \implies x_2 = 70 \tag{2.1.9}$$

- Solve for x_1 using x_2 and x_3:

$$2x_1 + 3x_2 = 5 \implies 2x_1 = 5 - 3 \times 70 \implies x_1 = -102.5 \tag{2.1.10}$$

The solution vector $\mathbf{x}$ is:

$$\mathbf{x} = \begin{bmatrix} -102.5 \\ 70 \\ -17 \\ 5 \end{bmatrix} \tag{2.1.11}$$

Python Implementation

Below is a Python code for backward substitution, suitable for solving systems involving upper triangular matrices.

```python
import numpy as np

def backward_substitution(U, b):
    n = len(b)
    x = np.zeros(n)
    # Start solving from the last row moving upwards
    for i in range(n-1, -1, -1):
        sum_Ux = np.dot(U[i, i+1:], x[i+1:])
        x[i] = (b[i] - sum_Ux) / U[i, i]
    return x

# Example usage:
U = np.array([
    [2, 3, 0, 0],
    [0, 1, 4, 0],
    [0, 0, 1, 5],
    [0, 0, 0, 2]
])
b = np.array([5, 2, 8, 10])

# Compute the solution
x = backward_substitution(U, b)
print("Solution x:", x)
```

This Python script implements the backward substitution method, calculating the solution vector $\mathbf{x}$ for systems with upper triangular matrices. By beginning at the last row and progressing upwards, it leverages the upper triangular structure to simplify and accelerate the computational process.

2.1.4 Applications

Upper triangular matrices are extensively used in numerical linear algebra for solving systems of linear equations, particularly when these systems are transformed via methods like Gaussian elimination or QR decomposition, as we will see below. Their properties allow for efficient computational algorithms, reducing the complexity and enhancing the stability of solutions in computational physics.

2.2 QR Decomposition

When dealing with systems of linear equations or studying the properties of vector spaces in computational physics, we often encounter matrices that are difficult to work with directly. One powerful approach is to decompose these matrices into products of simpler matrices, each with special properties that make computations easier. The QR decomposition is one such technique, where we write a matrix **A** as the product of an orthogonal matrix **Q** and an upper triangular matrix **R**:

$$\mathbf{A} = \mathbf{QR} \tag{2.2.1}$$

The power of this decomposition lies in combining two particularly useful matrix properties: orthogonality and triangularity. Orthogonal matrices preserve lengths and angles, making them numerically stable to work with, while triangular matrices allow for efficient forward or backward substitution in solving linear systems, as discussed before.

Why would we want to perform such a decomposition? Consider the problem of solving a system of linear equations $\mathbf{Ax} = \mathbf{b}$. Using QR decomposition, we can rewrite this as:

$$\mathbf{QRx} = \mathbf{b} \tag{2.2.2}$$

Since **Q** is orthogonal, we can multiply both sides by $\mathbf{Q}^T$ (recall that $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$):

$$\mathbf{Rx} = \mathbf{Q}^T\mathbf{b} \tag{2.2.3}$$

Now we have transformed our original system into one involving an upper triangular matrix **R**, which is much easier to solve through back substitution. This is just one example of how QR decomposition simplifies complex linear algebra problems.

2.2.1 Computing the QR Through The Gram-Schmidt Process

In the following, we will make extensive use of Dirac's notation for linear algebra. Recall a n dimensional vector v with elements v_i is represented as a ket:

$$|v\rangle \ = \ \sum_i v_i |i\rangle, \tag{2.2.4}$$

where $|i\rangle$ are the orthonormal unit basis states, such that $\langle i|j\rangle = \delta_{ij}$. Also, given an $n \times n$ matrix A, with matrix elements A_{ij}, it is represented in Dirac notation as

$$A \ = \ \sum_{ij} A_{ij} |i\rangle\langle j|. \tag{2.2.5}$$

Also, each column of a matrix can be viewed as a vector in an n-dimensional space, such that :

$$A_{ij} = \langle i|a_j\rangle \tag{2.2.6}$$

where $|a_j\rangle$ is the j-th column vector of A. Explicitly:

$$|a_j\rangle = \sum_{i=1}^{n} A_{ij}|i\rangle \tag{2.2.7}$$

Our goal is to decompose A into the product of an orthogonal matrix Q and an upper triangular matrix R. We will achieve this by applying the Gram-Schmidt process to the column vectors $\{|a_j\rangle\}_{j=1}^{n}$.

Starting with the column vectors $|a_1\rangle, |a_2\rangle, ..., |a_n\rangle$, we construct an orthonormal set $|q_1\rangle, |q_2\rangle, ..., |q_n\rangle$ as follows:

1. First basis vector:

$$|q_1\rangle = \frac{|a_1\rangle}{\sqrt{\langle a_1|a_1\rangle}} \tag{2.2.8}$$

2. For $k = 2, \ldots, n$: Remove components parallel to previous orthonormal vectors:

$$|u_k\rangle = |a_k\rangle - \sum_{j=1}^{k-1} |q_j\rangle\langle q_j|a_k\rangle \tag{2.2.9}$$

Then normalize:

$$|q_k\rangle = \frac{|u_k\rangle}{\sqrt{\langle u_k|u_k\rangle}} \tag{2.2.10}$$

This process generates an orthonormal set:

$$\langle q_i|q_j\rangle = \delta_{ij}, \tag{2.2.11}$$

as it is proven below. Now, each original column vector can be expressed in this new orthonormal basis we have constructed:

$$|a_1\rangle = |q_1\rangle\langle q_1|a_1\rangle \tag{2.2.12}$$
$$|a_2\rangle = |q_1\rangle\langle q_1|a_2\rangle + |q_2\rangle\langle q_2|a_2\rangle \tag{2.2.13}$$
$$|a_3\rangle = |q_1\rangle\langle q_1|a_3\rangle + |q_2\rangle\langle q_2|a_3\rangle + |q_3\rangle\langle q_3|a_3\rangle \tag{2.2.14}$$

More generally, for any column k:

$$|a_k\rangle = \sum_{j=1}^{k} |q_j\rangle\langle q_j|a_k\rangle, \tag{2.2.15}$$

where by construction $|q_j\rangle$ for $j > k$ do not enter the expansion in the Gram-Schmidt basis.

Proof that the Gram–Schmidt Process Yields an Orthonormal Set

Base Case: For $k = 1$, define

$$|q_1\rangle = \frac{|a_1\rangle}{\sqrt{\langle a_1|a_1\rangle}},$$

so that

$$\langle q_1|q_1\rangle = \frac{\langle a_1|a_1\rangle}{\langle a_1|a_1\rangle} = 1.$$

Inductive Step: Assume that for all indices $1 \le i, j < k$ the vectors satisfy

$$\langle q_i|q_j\rangle = \delta_{ij}.$$

For $k \ge 2$, define

$$|u_k\rangle = |a_k\rangle - \sum_{j=1}^{k-1} |q_j\rangle\langle q_j|a_k\rangle,$$

and normalize to obtain

$$|q_k\rangle = \frac{|u_k\rangle}{\sqrt{\langle u_k|u_k\rangle}}.$$

For any $1 \le i < k$, consider the inner product

$$\langle q_i|q_k\rangle = \frac{1}{\sqrt{\langle u_k|u_k\rangle}}\langle q_i|\Big(|a_k\rangle - \sum_{j=1}^{k-1} |q_j\rangle\langle q_j|a_k\rangle\Big).$$

Distributing the inner product gives

$$\langle q_i|q_k\rangle = \frac{1}{\sqrt{\langle u_k|u_k\rangle}}\left(\langle q_i|a_k\rangle - \sum_{j=1}^{k-1} \langle q_i|q_j\rangle\langle q_j|a_k\rangle\right).$$

By the induction hypothesis, $\langle q_i|q_j\rangle = \delta_{ij}$. Thus, the sum reduces to

$$\sum_{j=1}^{k-1} \delta_{ij}\langle q_j|a_k\rangle = \langle q_i|a_k\rangle.$$

Therefore,

$$\langle q_i|q_k\rangle = \frac{1}{\sqrt{\langle u_k|u_k\rangle}}\Big(\langle q_i|a_k\rangle - \langle q_i|a_k\rangle\Big) = 0.$$

Furthermore, by construction,

$$\langle q_k|q_k\rangle = 1.$$

Thus, by induction the set $\{|q_1\rangle, |q_2\rangle, \dots, |q_n\rangle\}$ is orthonormal:

$$\langle q_i|q_j\rangle = \delta_{ij}.$$

2.2.2 Explicit Construction of Q and R

The matrices Q and R are constructed as follows:

1. Matrix Q: The columns of Q are just the components of $|q_j\rangle$ in the standard basis:

$$Q_{ij} = \langle i|q_j\rangle \tag{2.2.16}$$

2. Matrix R: The elements of R are the coefficients from the Gram-Schmidt process:

$$R_{ij} = \langle q_i|a_j\rangle \tag{2.2.17}$$

Note that $R_{ij} = 0$ for $i > j$ by construction.

To prove that $A = QR$, we examine the matrix elements of both sides:

1. Left side: Matrix element (i, j) of A is:

$$A_{ij} = \langle i|a_j\rangle \tag{2.2.18}$$

2. Right side: Matrix element (i, j) of QR is:

$$(QR)_{ij} = \sum_{k=1}^{n} Q_{ik}R_{kj} = \sum_{k=1}^{n}\langle i|q_k\rangle\langle q_k|a_j\rangle \tag{2.2.19}$$

3. Using the completeness relation of the orthonormal basis $\{|q_k\rangle\}$:

$$\langle i|a_j\rangle = \langle e_i| \left(\sum_{k=1}^{n} |q_k\rangle\langle q_k|\right) |a_j\rangle = \sum_{k=1}^{n}\langle i|q_k\rangle\langle q_k|a_j\rangle \tag{2.2.20}$$

Therefore, $A_{ij} = (QR)_{ij}$ for all i, j, proving that $A = QR$.

2.2.3 Properties of the Decomposition

We can also immediately verify two of the most important properties of this decomposition, namely:

1. Q is orthogonal:

$$(Q^\dagger Q)_{ij} = \sum_k Q^*_{ki}Q_{kj} = \sum_k\langle q_i|k\rangle\langle k|q_j\rangle = \langle q_i|q_j\rangle = \delta_{ij} \tag{2.2.21}$$

2. R is upper triangular by construction of the Gram-Schmidt process:

$$R_{ij} = \langle q_i|a_j\rangle = 0 \quad \text{for } i > j \tag{2.2.22}$$

2.2.4 Implementation and Numerical Considerations

Here's a basic implementation in Python that illustrates the process:

```python
def gram_schmidt_qr(A):
    """
    Compute the QR decomposition using Gram-Schmidt process.
    """
    m, n = A.shape
    Q = np.zeros((m, n))
    R = np.zeros((n, n))

    for j in range(n):
        v = A[:, j]
        for i in range(j):
            # Calculate projection coefficients
            R[i, j] = np.dot(Q[:, i].T, A[:, j])
            v = v - R[i, j] * Q[:, i]
        R[j, j] = np.linalg.norm(v)
        if R[j, j] != 0:
            Q[:, j] = v / R[j, j]

    return Q, R
```

This implementation explicitly shows how we build up the **Q** and **R** matrices column by column. The upper triangular structure of **R** emerges naturally from the process of orthogonalization.

While the Gram-Schmidt process is conceptually elegant, it can suffer from numerical instability in practice. In most commonly used linear algebra software, one determines the decomposition through an approach known as the Householder reflections. We will not discuss it here, but it is worth knowing that this is the standard way of computing the QR decomposition in a numerically stable way.

2.2.5 Computing Determinants Using QR Decomposition

Besides being useful for solving linear systems, the QR decomposition can be used for many other applications. For example, once **A** is decomposed into **Q** and **R**, the determinant of **A** can be computed efficiently. Since **Q** is orthogonal, $\det(\mathbf{Q}) = \pm 1$, and the determinant of **A** is given by:

$$\det(\mathbf{A}) = \det(\mathbf{Q})\det(\mathbf{R}) \tag{2.2.23}$$

Since **R** is upper triangular, its determinant is the product of its diagonal entries:

$$\det(\mathbf{R}) = \prod_{i=1}^{n} r_{ii} \tag{2.2.24}$$

Thus, the determinant of **A** is:

$$\det(\mathbf{A}) = \det(\mathbf{Q})\prod_{i=1}^{n} r_{ii} \tag{2.2.25}$$

This approach simplifies the computation of determinants for large matrices and enhances numerical stability in calculations.

2.2.5.1 Physical Example: Slater Determinants

Slater determinants are used in quantum mechanics to construct wavefunctions for systems of fermions, such as electrons, that obey the Pauli exclusion principle. This principle states that no two fermions can occupy the same quantum state simultaneously. In practical terms, this means the overall wavefunction of a system must be antisymmetric with respect to the exchange of any two electrons.

Consider a simple system of two electrons. The wavefunction for each electron can be described in terms of its spatial (ψ) and spin (χ) components. The total wavefunction Ψ for the system, considering only the spatial part for simplicity, must change sign when the positions of the two electrons are swapped. This antisymmetry is crucial for reflecting the fermionic nature of electrons and is elegantly handled by the Slater determinant.

For two electrons with wavefunctions $\psi_1(x_1)$ and $\psi_2(x_2)$, the Slater determinant is given by:

$$\Psi(x_1, x_2) = \frac{1}{\sqrt{2}} \begin{vmatrix} \psi_1(x_1) & \psi_2(x_1) \\ \psi_1(x_2) & \psi_2(x_2) \end{vmatrix} \tag{2.2.26}$$

This determinant ensures that Ψ is antisymmetric under the exchange of x_1 and x_2, i.e., swapping x_1 and x_2 changes the sign of Ψ.

The use of a determinant in forming the wavefunction for multiple electrons ensures antisymmetry automatically. If any two electrons were to occupy the same quantum state, their wavefunctions would be identical, leading to two identical rows in the determinant, which makes the determinant (and hence the wavefunction) zero. This zeros-out probability of finding two electrons in the same state, in accordance with the Pauli exclusion principle.

As discussed in the context of matrix operations like QR decomposition, computing determinants for large systems can be efficiently achieved using numerical methods. For Slater determinants, where the matrix size grows with the number of electrons, QR decomposition provides a numerically stable method to compute the determinant as discussed earlier:

$$\det(\mathbf{A}) = \prod_{i=1}^{n} r_{ii} \tag{2.2.27}$$

where $\mathbf{A}$ is the matrix formed from the wavefunctions of the electrons, and r_{ii} are the diagonal elements of $\mathbf{R}$ from the QR decomposition of $\mathbf{A}$.

For a two-electron system, QR decomposition not only confirms the antisymmetry but also allows for efficient numerical computation of properties derived from the wavefunction, such as energy and probability densities, especially important in computational chemistry and physics simulations involving larger molecules or more complex atomic structures.

Chapter 3

Solving The Eigenvalue Problem

Eigenvalue problems are pivotal in computational physics for understanding the behavior of various physical systems through their eigenmodes and eigenfrequencies. This chapter will delve into eigenvalue problems, particularly focusing on the computational methods to find eigenvalues and eigenvectors.

3.1 Jacobi Eigenvalue Algorithm

The eigenvalue problem for a matrix A consists of finding scalars ω and non-zero states $|v\rangle$ that satisfy:

$$A|v\rangle = \omega|v\rangle \tag{3.1.1}$$

For an $n \times n$ matrix, we can collect all n eigenvectors into a resolution of identity:

$$\sum_{i=1}^{n} |v_i\rangle\langle v_i| = I \tag{3.1.2}$$

The eigenvalue equation can then be written in a more complete form:

$$A\sum_{i=1}^{n} |v_i\rangle\langle v_i| = \sum_{i=1}^{n} \omega_i|v_i\rangle\langle v_i| \tag{3.1.3}$$

If the set of eigenvectors forms a complete basis, we can express the diagonalization as:

$$\langle v_i|A|v_j\rangle = \omega_i\delta_{ij} \tag{3.1.4}$$

For symmetric matrices, which are our focus in the following, we have additional important properties:

- All eigenvalues ω_i are real

- Eigenvectors corresponding to different eigenvalues are orthogonal: $\langle v_i | v_j \rangle = \delta_{ij}$
- The transformation matrix can be chosen to be unitary: $V^\dagger = V^{-1}$

3.1.1 The Jacobi Method

The Jacobi method, introduced by Carl Gustav Jacob Jacobi in 1846, provides an iterative approach to finding eigenvalues and eigenvectors of a symmetric matrix. The key insight is that a symmetric matrix is diagonal if and only if all its off-diagonal elements are zero. In the bra-ket notation, we seek a sequence of unitary transformations U_k such that:

$$\langle v_i | U_k^\dagger A U_k | v_j \rangle \to \omega_i \delta_{ij} \tag{3.1.5}$$

The method works by applying a sequence of similarity transformations:

$$A^{(k+1)} = G_k^\dagger A^{(k)} G_k \tag{3.1.6}$$

where each G_k is a rotation matrix chosen to eliminate one off-diagonal element.

3.1.2 Givens Rotations

The rotation matrices G_k used in the Jacobi method are known as *Givens rotations.* In an n-dimensional space with an orthonormal basis $\{|1\rangle, |2\rangle, \dots, |n\rangle\}$, the Givens rotation in the (p, q)–plane is denoted by $G(p, q, \theta)$ and acts as the identity on all coordinates except in the (p, q)–plane. In that subspace, the matrix is given by the 2×2 block

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

Thus, the full $n \times n$ matrix can be schematically written as

$$G(p,q,\theta) = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos\theta & \cdots & -\sin\theta & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \sin\theta & \cdots & \cos\theta & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix}.$$

When we multiply the matrix $G(p,q,\theta)$ by a basis vector, we are effectively selecting a column of the matrix. Hence, the images of the basis states in the (p,q)–subspace are determined by the columns of the 2×2 block:

$$\begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}.$$

Specifically,

- The $\mathbf{p}^{\mathbf{th}}$ column is
$$\begin{pmatrix}\cos\theta\\ \sin\theta\end{pmatrix},$$
so
$$G(p,q,\theta)\,|p\rangle = \cos\theta\,|p\rangle + \sin\theta\,|q\rangle.$$
- The $\mathbf{q}^{\mathbf{th}}$ column is
$$\begin{pmatrix}-\sin\theta\\ \cos\theta\end{pmatrix},$$
so
$$G(p,q,\theta)\,|q\rangle = -\sin\theta\,|p\rangle + \cos\theta\,|q\rangle.$$
- For any $r\neq p,q$, the matrix acts as the identity:
$$G(p,q,\theta)\,|r\rangle = |r\rangle.$$

Rotated Matrix Elements

For an operator A with matrix elements $\langle i|A|j\rangle$, the rotated matrix elements in the new basis (obtained by applying $G(p,q,\theta)$) are given by

$$\langle i'|A|j'\rangle,$$

where the rotated states in the (p,q)–subspace are

$$\begin{aligned}|p'\rangle &= G(p,q,\theta)|p\rangle = \cos\theta\,|p\rangle + \sin\theta\,|q\rangle,\\ |q'\rangle &= G(p,q,\theta)|q\rangle = -\sin\theta\,|p\rangle + \cos\theta\,|q\rangle.\end{aligned}$$

In particular, consider the off–diagonal element in this subspace:

$$\langle p'|A|q'\rangle = \Big(\cos\theta\,\langle p| + \sin\theta\,\langle q|\Big)A\Big(-\sin\theta\,|p\rangle + \cos\theta\,|q\rangle\Big).$$

Expanding, we obtain

$$\langle p'|A|q'\rangle = -\sin\theta\,\cos\theta\,\langle p|A|p\rangle + \cos^2\theta\,\langle p|A|q\rangle - \sin^2\theta\,\langle q|A|p\rangle + \sin\theta\,\cos\theta\,\langle q|A|q\rangle.$$

If A is symmetric (or Hermitian) so that $\langle p|A|q\rangle = \langle q|A|p\rangle$, this simplifies to

$$\langle p'|A|q'\rangle = \cos(2\theta)\,\langle p|A|q\rangle - \frac{1}{2}\sin(2\theta)\Big(\langle p|A|p\rangle - \langle q|A|q\rangle\Big),$$

using the identities

$$\cos^2\theta - \sin^2\theta = \cos(2\theta) \quad \text{and} \quad 2\sin\theta\,\cos\theta = \sin(2\theta).$$

To zero out the off–diagonal element (the key step in the Jacobi method), one sets $\langle p'|A|q'\rangle = 0$. This yields the equation

$$\cos(2\theta)\,\langle p|A|q\rangle - \frac{1}{2}\sin(2\theta)\Big(\langle p|A|p\rangle - \langle q|A|q\rangle\Big) = 0,$$

which can be rearranged to determine the rotation angle:

$$\tan(2\theta) = \frac{2\,\langle p|A|q\rangle}{\langle p|A|p\rangle - \langle q|A|q\rangle}. \tag{3.1.7}$$

For numerical stability—especially when $\tan(2\theta)$ is large—it is common to define

$$\tau = \tan(2\theta), \quad t = \operatorname{sign}(\tau)\,\frac{1}{|\tau| + \sqrt{1+\tau^2}},$$

and then compute

$$\cos\theta = \frac{1}{\sqrt{1+t^2}}, \quad \sin\theta = t\,\cos\theta.$$

3.1.3 Eigenvector Construction

After N rotations, we have:

$$A^{(N)} = G_N^\dagger G_{N-1}^\dagger ... G_1^\dagger A G_1 ... G_{N-1} G_N \tag{3.1.8}$$

When the algorithm converges, $A^{(N)}$ is diagonal, so:

$$D = G_N^\dagger G_{N-1}^\dagger ... G_1^\dagger A G_1 ... G_{N-1} G_N \tag{3.1.9}$$

where D is a diagonal matrix containing the eigenvalues. Let's define:

$$V = G_1 G_2 ... G_N \tag{3.1.10}$$

Then we can write:

$$D = V^{\dagger} A V \tag{3.1.11}$$

By multiplying on the left with V and remembering that $V^{\dagger}V = I$ since it's a unitary transformation, the equation above means:

$$AV = VD \tag{3.1.12}$$

Looking at this equation column by column, for the i-th column v_i of V:

$$A|v_i\rangle = \omega_i |v_i\rangle \tag{3.1.13}$$

This shows that the columns of $V$are just the eigenvectors of A, thus by accumulating products of the rotation matrices we obtain both eigenvalues and eigenvectors of the original matrix.

3.1.4 Algorithm Structure

The complete Jacobi algorithm proceeds as follows:

1. Initialize $\mathbf{V} = \mathbf{I}$ (to accumulate eigenvectors)
2. While off-diagonal elements are larger than tolerance:
   (a) Find indices (p, q) of largest off-diagonal element
   (b) Compute rotation angle θ using Eq. 3.1.7
   (c) Construct Givens rotation matrix $\mathbf{G}(p, q, \theta)$
   (d) Update matrix: $\mathbf{A} \leftarrow \mathbf{G}^T \mathbf{A} \mathbf{G}$
   (e) Accumulate eigenvectors: $\mathbf{V} \leftarrow \mathbf{V}\mathbf{G}$
3. Extract eigenvalues from diagonal of $\mathbf{A}$

Here is the implementation in Python:

```python
def jacobi_eigenvalues(A, tol=1e-10, max_iter=50):
    """
    Computes the eigenvalues and eigenvectors of a symmetric matrix A using the
   Jacobi method.
    """
    n = A.shape[0]
    V = np.eye(n)  # Initialize eigenvector matrix as identity
    D = A.copy()

    for _ in range(max_iter):
        # Find largest off-diagonal element
        p, q = np.unravel_index(np.abs(np.triu(D, 1)).argmax(), D.shape)
        if np.abs(D[p, q]) < tol:
            break

        # Compute the Jacobi rotation using numerically stable formulas
        tau = (D[q, q] - D[p, p]) / (2 * D[p, q])
        t = np.sign(tau) / (abs(tau) + np.sqrt(1 + tau**2))
        c = 1 / np.sqrt(1 + t**2)
        s = t * c

        R = np.eye(n)
        R[p, p], R[q, q] = c, c
        R[p, q], R[q, p] = s, -s

        # Apply rotation
        D = R.T @ D @ R
        V = V @ R

    return np.diag(D), V  # Eigenvalues and eigenvectors
```

3.1.5 Convergence Analysis

The convergence of the Jacobi method can be analyzed using the sum of squares of off-diagonal elements:

$$S(\mathbf{A}) = \sum_{i \neq j} |a_{ij}|^2 \tag{3.1.14}$$

Each rotation reduces this sum by:

$$S(\mathbf{A}) - S(\mathbf{A}') = 2|a_{pq}|^2 \tag{3.1.15}$$

For the classical Jacobi method where we always choose the largest off-diagonal element, we can prove:

$$S(\mathbf{A}^{(k+1)}) \leq \left(1 - \frac{2}{n(n-1)}\right) S(\mathbf{A}^{(k)}) \tag{3.1.16}$$

This shows linear convergence with rate at least $1 - \frac{2}{n(n-1)}$.

3.1.6 References

1. Jacobi, C.G.J. (1846). "Über ein leichtes Verfahren, die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen"

2. Golub, G.H.; Van Loan, C.F. (2013). "Matrix Computations"

3. Press, W.H. et al. (2007). "Numerical Recipes: The Art of Scientific Computing"

Chapter 4

Linear Algebra in Tensor Network Notation

Tensor network notation, also known as Penrose graphical notation, is a concise way to represent vectors, matrices, and higher-rank tensors, along with their contractions. Instead of writing sums explicitly, repeated indices are shown as lines connecting tensor nodes. This approach can simplify many linear algebra manipulations.

In index notation, for example, a matrix is written as M_{ij}, a vector as v_i, and a rank-3 tensor as T_{ijk}. Below we show how to represent these objects diagrammatically and demonstrate common operations like matrix-vector multiplication, matrix-matrix multiplication, and traces. We also discuss why the trace is cyclic and how the identity matrix is drawn in this notation.

4.1 Basic Rules

1. A tensor is represented by a small shape (e.g. a circle) labeled with a letter such as M or T, but *not* with the full index structure.

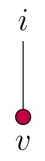
2. Each index of the tensor is depicted as a line (leg) emerging from that shape.

3. Connecting two lines denotes a sum (contraction) over the shared index.

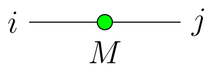
4. Any line not connected to another shape is a free index, meaning it remains in the final result.

4.2 Examples of Low-Rank Tensors

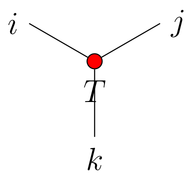
Vector v_i



Matrix M_{ij}



Rank-3 Tensor T_{ijk}



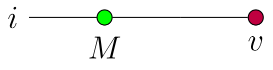
4.3 Matrix-Vector and Matrix-Matrix Products

Matrix-Vector Product: $(Mv)_i = \sum_j M_{ij}\, v_j$

Algebraically:

$$(Mv)_i = \sum_j M_{ij}\, v_j.$$

Diagrammatically:

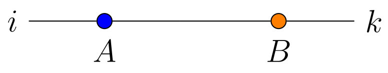


Matrix-Matrix Product: $(AB)_{ik} = \sum_j A_{ij}\, B_{jk}$

Algebraically:

$$(AB)_{ik} = \sum_j A_{ij}\, B_{jk}.$$

Diagrammatically:



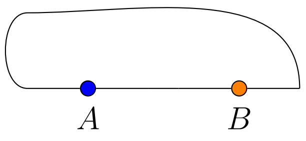
4.4 Trace and Its Cyclic Property

Trace: $\mathrm{Tr}(AB) = \sum_{i,j} A_{ij}\, B_{ji}$

Algebraically:

$$\mathrm{Tr}(AB) = \sum_{i,j} A_{ij}\, B_{ji}.$$

Diagrammatically, the lines form a loop:



Cyclic Property

A key property of the trace is $\mathrm{Tr}(AB) = \mathrm{Tr}(BA)$. In index notation,

$$\mathrm{Tr}(AB) = \sum_{i,j} A_{ij}\, B_{ji} = \sum_{j,i} B_{ji}\, A_{ij} = \mathrm{Tr}(BA).$$

Diagrammatically, because the lines form a closed loop, the order in which the tensors appear around the loop does not matter.

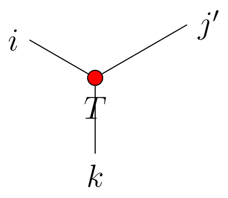
4.5 Identity Matrices and the Identity Trick

Because contracting a tensor over one of its indices with the identity matrix has no effect, it is customary to notate identity matrices as plain lines with no blob or shape. Diagrammatically, this looks like:

$$\delta_{ij} \;=\; i \;\text{———}\; j \;.$$

This notation is useful, because the diagram for contracting with an identity matrix simply extends an index line, having no effect on the tensor. For example,

$$\sum_j T_{ijk}\,\delta_{jj'} \;=$$



$$= T_{ij'k}.$$

Since $\delta_{jj'}$ merely replaces j with j', the tensor T_{ijk} is unchanged apart from the index relabeling.