

These notes draw heavily on *Numerical Recipes*, a valuable resource for entry-level understanding of numerical methods relevant to physics.

1. Linear Algebra in Computation

A huge fraction of numerical computation involves linear algebra at some level or another. This includes simple application of matrix multiplication, but also applications of matrix inversion, decompositions, and other important operations. Vectors and higher-dimensional objects in linear algebra are held as arrays in NumPy (and other languages).

You are aware of the usual matrix notation:

$$\begin{aligned} x_i &\rightarrow \vec{x} \\ Q_{ij} &\rightarrow \mathbf{Q} \end{aligned} \tag{1}$$

so for example:

$$y_i = \sum_j Q_{ij} x_j \tag{2}$$

may also be written:

$$\vec{y} = \mathbf{Q} \cdot \vec{x} \tag{3}$$

If you have an array in Python or another language, you can perform these operations explicitly, but it is better to use the explicit matrix operations in NumPy and associated packages. It will be far faster. If you delve deep into high-order objects (e.g. T_{ijkl}) you can definitely run into cases where the standard routines won't do the operation you want, but most cases will work fine.

I show some examples in the workbook.

2. Linear systems of equations

Many physical and statistical problems boil down to solving a linear system of the form:

$$\mathbf{A} \cdot \vec{x} = \vec{b} \tag{4}$$

where \mathbf{A} and \vec{b} are known, and we want to know \vec{x} .

If \mathbf{A} is an $M \times N$ matrix (and therefore \vec{x} is an N -vector) then under what conditions is there a unique, exact solution \vec{x} to this equation?

How can we check if matrices are singular?

You have probably done solved linear systems by hand using a Gauss-Jordan type of technique. This is not what is done numerically. Instead, the simplest matrix inversion technique of practical use is *LU decomposition*.

LU decomposition yields a lower triangular matrix \mathbf{L} and an upper triangular matrix \mathbf{U} , which satisfy:

$$\mathbf{L} \cdot \mathbf{U} = \mathbf{A} \quad (5)$$

This then means we can solve:

$$\mathbf{L} \cdot \vec{y} = \mathbf{b} \quad (6)$$

and then:

$$\mathbf{U} \cdot \vec{x} = \mathbf{y} \quad (7)$$

and the resulting \vec{x} satisfies:

$$\mathbf{A} \cdot \vec{x} = \mathbf{L} \cdot \mathbf{U} \cdot \vec{x} = \mathbf{L} \cdot \vec{y} = \vec{b} \quad (8)$$

Hooray, but how does that help us. Well, with a triangular matrix, the equations 6 and 7 can easily be solved by backsubstitution. That is, you can start at:

$$y_0 = \frac{b_0}{L_{00}} \quad (9)$$

and then find:

$$y_i = \frac{b_i - \sum_{j=0}^{i-1} L_{ij} y_j}{L_{ii}} \quad (10)$$

How many operations does the solution take once you have \mathbf{L} and \mathbf{U} ?

It turns out that the LU decomposition can be done efficiently, with the same scaling with N . Now, in most implementations, in fact what is done is that the rows of \mathbf{A} are shuffled in the procedure so as to keep the numerics stable.

Also, the decomposition is not unique. There are $N^2 + N$ values to set in \mathbf{L} and \mathbf{U} but only N^2 values in \mathbf{A} . A typical choice is to let the diagonal of \mathbf{L} be just 1s.

This gives me a way of solving the equations. How do I determine the inverse of \mathbf{A} ?

If I have an LU decomposition of \mathbf{A} , how can I calculate the determinant of \mathbf{A} ?

I show in the notebook an example of using the `linalg` routine `solve`, which utilizes this technique to solve a set of linear equations.

3. Singular Value Decomposition

The above methods are great if you have an invertible $N \times N$ matrix. But what if the solution to $\mathbf{A} \cdot \vec{x} = \vec{b}$ is not unique, or doesn't exist? Or, what if \mathbf{A} is sufficiently close to singular that round-off error starts making the solution for the inverse numerically unstable. What do we do? The answer is *singular value decomposition* (SVD), which provides a stable way to deal with all of these cases.

SVD relies on the fact that any $M \times N$ matrix can be written as:

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^T \quad (11)$$

where \mathbf{U} is $M \times N$ with columns that are all orthonormal, \mathbf{W} is an $N \times N$ diagonal matrix, and \mathbf{V}^T is the transpose of an $N \times N$ orthonormal matrix.

The components of the diagonal matrix \mathbf{W} are the *singular values*. By convention in these decompositions, they are arranged in descending order. If $M < N$ (or if for any other reason the row rank is less than N), then there are fewer independent equations than unknowns, and there will be w_j values that take the value zero.

The condition of orthonormality on \mathbf{V} means:

$$\mathbf{V}^T \cdot \mathbf{V} = 1 \quad (12)$$

And on \mathbf{U} this means if $M \geq N$:

$$\mathbf{U}^T \cdot \mathbf{U} = 1 \quad (13)$$

but if $M < N$ then some of the w_j are zero, so:

$$\sum_{i=0}^{M-1} U_{ik} U_{in} = \delta_{kn} \quad (14)$$

for $k \leq M - 1$ and $n \leq M - 1$, but for higher k and n the sum yields zero.

Performing this decomposition can be done very stably, using routines within `linalg`. Exactly how it is done is outside our scope here.

What is the point of doing this? Consider $\mathbf{A} \cdot \vec{x} = \vec{b}$. \vec{x} is an N -dimensional vector, and \vec{b} is M -dimensional. \mathbf{A} maps between these two spaces, but it may not be able to map to the full M -dimensional space. It may have the property that it can only map to a subspace.

A trivial example is:

$$A = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \quad (15)$$

Any vector \vec{x} I multiple by ends up on the line $b_0 = -b_1$. \mathbf{A} maps from a two-dimensional space into a one-dimensional one.

Is this \mathbf{A} invertible?

The *range* of \mathbf{A} is the subspace \mathbf{A} can map into. The *rank* of \mathbf{A} is the dimension of the range. If the rank is less than the number of dimensions (N) of \vec{x} , then what happens to all the other dimensions? It indicates that there is a *null space* for which $\mathbf{A} \cdot \vec{x} = 0$. The dimension of the null space is N minus the rank.

What SVD does is basically an analysis of \mathbf{A} which tells you the range and null space, and the rank of the matrix. Specifically, the columns of \mathbf{U} corresponding to non-zero w_j form an orthonormal basis of the range of \mathbf{A} — clearly the $\mathbf{W} \cdot \mathbf{V}^T$ that multiplies into \mathbf{U} must yield something that lives in this subspace. The columns of \mathbf{V} corresponding to zero w_j span the null space. The rank is the number of non-zero w_j .

4. Sparse matrices

There are many applications where you have in principle very large matrices, but they are in fact very sparse.

band-diagonal