Linear Algebra

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 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 **L** and and upper triangular matrix **U**, which satisfy:

$$\mathbf{L} \cdot \mathbf{U} = \mathbf{A} \tag{5}$$

How do you do this? Look at a 3×3 case:

$$\mathbf{A} = \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix} \tag{6}$$

We can multiply this by a matrix

$$\mathbf{L_0} = \frac{1}{A_{00}} \begin{pmatrix} 1 & 0 & 0 \\ -A_{10} & A_{00} & 0 \\ -A_{20} & 0 & A_{00} \end{pmatrix} \tag{7}$$

and find:

$$\mathbf{L_0} \cdot \mathbf{A} = \begin{pmatrix} 1 & B_{01} & B_{02} \\ 0 & B_{11} & B_{12} \\ 0 & B_{21} & B_{22} \end{pmatrix} \tag{8}$$

Then you can then multiply by:

$$\mathbf{L_1} = \frac{1}{B_{11}} \begin{pmatrix} B_{11} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -B_{21} & B_{11} \end{pmatrix} \tag{9}$$

which leads to:

$$\mathbf{L_1} \cdot \mathbf{L_0} \cdot \mathbf{A} = \begin{pmatrix} 1 & C_{01} & C_{02} \\ 0 & 1 & C_{12} \\ 0 & 0 & C_{22} \end{pmatrix} \tag{10}$$

and then you just multiply by:

$$\mathbf{L_2} = \frac{1}{C_{22}} \begin{pmatrix} C_{22} & 0 & 0 \\ 0 & C_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{11}$$

and then:

$$\mathbf{L_2} \cdot \mathbf{L_1} \cdot \mathbf{L_0} \cdot \mathbf{A} = \begin{pmatrix} 1 & D_{01} & D_{02} \\ 0 & 1 & D_{12} \\ 0 & 0 & 1 \end{pmatrix} \tag{12}$$

We can then write:

$$\mathbf{A} = (\mathbf{L_0}^{-1} \cdot \mathbf{L_1}^{-1} \cdot \mathbf{L_2}^{-1}) \cdot (\mathbf{L_0} \cdot \mathbf{L_1} \cdot \mathbf{L_2} \cdot \mathbf{A}) = \mathbf{L} \cdot \mathbf{U} \tag{13}$$

It is straightforward to extend this to any size matrix, so you can create $\mathbf{L_i}$ and just multiply out to get $\mathbf{U}$. Each $\mathbf{L_i}^{-1}$ can be found through backsubstitution (see below), but actually working through this by hand and multiplying out you will find:

$$\mathbf{L} = \begin{pmatrix} A_{00} & 0 & 0 \\ A_{10} & B_{11} & 0 \\ A_{20} & B_{21} & C_{22} \end{pmatrix} \tag{14}$$

So there is a good way to do the decomposition.

This then means we can solve:

$$\mathbf{L} \cdot \vec{y} = \mathbf{b} \tag{15}$$

and then:

$$\mathbf{U} \cdot \vec{x} = \mathbf{y} \tag{16}$$

and the resulting $\vec{x}$ satisfies:

$$\mathbf{A} \cdot \vec{x} = \mathbf{L} \cdot \mathbf{U} \cdot \vec{x} = \mathbf{L} \cdot \vec{y} = \vec{b} \tag{17}$$

Hooray, but how does that help us. Well, with a triangular matrix, the equations 15 and 16 can easily be solved by backsubstitution. That is, you can start at:

$$y_0 = \frac{b_0}{L_{00}} \tag{18}$$

and then find:

$$y_i = \frac{b_i - \sum_{j=0}^{i-1} L_{ij} y_j}{L_{11}} \tag{19}$$

How many operations does the solution take once you have L and U?

If you work through the operations above, the LU decomposition itself takes $\mathcal{O}(N^3)$; it requires a series of N matrix multiplications basically. 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 $\mathbf{A}$. The typical choice is to let the diagonal of $\mathbf{L}$ be just 1s, as done above.

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

If I have an LU decomposition of A, how can I calculate the determinant of 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 makes 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:

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^T \tag{20}$$

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:

$$\mathbf{V}^T \cdot \mathbf{V} = 1 \tag{21}$$

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

$$\mathbf{U}^T \cdot \mathbf{U} = 1 \tag{22}$$

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

$$\sum_{i=0}^{M-1} U_{ik} U_{in} = \delta_{kn} \tag{23}$$

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:

$$\mathbf{A} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \tag{24}$$

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 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.

This decomposition gives yields a useful quantity for quantifying the singularity of a matrix, called the *condition number*. The condition number of a matrix is the ratio between the maximum and minimum value of w_j. For a singular matrix this is infinity. However, whenever this number is close to the inverse of the machine precision (e.g. 10^{-15} in double precision) then the matrix is close enough to singular that numerics on it will become unstable.

4. Inversion of square matrices in SVD

Let us return to the question of $N \times N$ matrices that we perform SVD on.

What is the inverse of A in terms of the decomposition?

Clearly, then, if $\mathbf{A}$ is not singular or close to singular, once the SVD is done it is trivial to invert the matrix $\mathbf{A}$, and obtain the solution to the problem.

Now, what if the matrix $\mathbf{A}$ is singular? If so, then there are two possibilities. Either $\vec{b}$ is in the range of $\mathbf{A}$, or it is not.

If it is in the range, then there are an infinite set of solutions; if you find one solution, you can always take that solution $\vec{x}$ plus any solution in the null space $\vec{x}'$, and it is a new solution. You can find the solution with the smallest norm in the following way. Take all the zero w_j, and replace $1/w_j$ with 0 instead of infinity. What??? Yes.

You can prove this as follows. Say $\mathbf{W}^{-1}$ is the inverse of $\mathbf{W}$ with the adjustment made for zero w_j. Then:

$$\begin{aligned} |\vec{x} + \vec{x}'| &= \left| \mathbf{V} \cdot \mathbf{W}^{-1} \cdot \mathbf{U}^T \cdot \vec{b} + \vec{x}' \right| \\ &= \left| \mathbf{V} \cdot \left(\mathbf{W}^{-1} \cdot \mathbf{U}^T \cdot \vec{b} + \mathbf{V}^T \cdot \vec{x}' \right) \right| \\ &= \left| \mathbf{W}^{-1} \cdot \mathbf{U}^T \cdot \vec{b} + \mathbf{V}^T \cdot \vec{x}' \right| \end{aligned} \tag{25}$$

where the last equality follows from the orthonormality of $\mathbf{V}$. The left term has components only

where $w_j \neq 0$. The right term is in the null space, so has only components in directions where $w_j = 0$. We get to choose $\vec{x}$ so we should choose it to be zero.

If $\vec{b}$ is not in the range, then the same procedure, it turns out, yields an $\vec{x}$ then minimizes:

$$r = \left| \mathbf{A} \cdot \vec{x} - \vec{b} \right| \tag{26}$$

Therefore, the inverse with the conversion of $w_j = 0 \to 1/w_j = 0$ yields what we usually want in all cases. It is known as the *Moore-Penrose inverse* or *pseudoinverse* of a matrix.

The SVD solution of linear systems is in almost all cases the correct approach. Note that when w_j values become less than the machine precision, they often should also be treated as zeros to avoid numerical instability.

5. Sparse matrices

There are many applications where you have in principle very large matrices, but they are in fact very sparse. A common case is a tridiagonal or band-diagonal system. However, it is also possible that there are simply sparse off-diagonal terms.

For these systems, you can save a lot of time and memory by not operating on the full matrix, but only the non-sparse values.

SciPy has a whole set of routines for dealing with such matrices, including inverses, SVD, etc.