Rank

The rank of a matrix is equivalently:

1. The number of linearly independent rows.
2. The number of linearly independent columns.
3. The number of non-zero singular values.
4. And there are many other alternatives: https://en.wikipedia.org/wiki/Rank_(linear_algebra)#Alternative_definitions

Thus the maximum rank of an $n \times m$ matrix is $\min(n, m)$. A matrix is called *full rank* if it hits this maximum.

The rank of a matrix is *very* important in determining how we can use it. Unfortunately in much of your math career, it will have been assumed that all the matrices you deal with a full rank, and hence "easy" to work with.

But here is one example: we want to solve the linear equations $A\mathbf{x} = \mathbf{b}$. It is common in this context to also use the augmented matrix $M = [A|\mathbf{b}]$. We can determine how many solutions there will be based on ranks as follows (assuming there are n variables, so A has n columns):

- If $rank(A) < rank(M)$ then the equations are inconsistent (there are no solutions).
- If $rank(A) = rank(M) = n$ there is a single, unique solution.
- If $rank(A) = rank(M) < n$ then there are infinitely many solutions.

Note that $rank(A)$ can't be $> rank(M)$ because M is just the matrix A with one added column so the rank of M is either the same as A or one larger, so the above rules encapsulate all of the possibilities. The result is sometimes called the Rouché-Capelli theorem.

So rank is very important. But this course is not about solving linear equations.

Also, one of the common themes, however, will be about approximation. Low-rank approximations of matrices are sometimes very appealing to work with both numerically and in terms of understanding results. We'll see may examples of this, but for the moment, calculate the ranks of the following two matrices:

$$A_1 = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \text{ and } A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3.0001 \\ 1 & 2.0001 & 3 \end{bmatrix}.$$

```python
import torch
A = torch.tensor([[1.0,2.0,3.0], [1.0,2.0,3.0], [1.0,2.0,3.0]])
torch.linalg.matrix_rank(A)
```

Now try using

```python
torch.linalg.matrix_rank(A, tol=0.01)
```

Tolerance does something here, but what?

Other things to think about

- Can you train a classifier to determine the rank of a matrix? See https://www.oneweirdkerneltrick.com/rank.pdf
- Real matrices come from many applications – what sort of ranks do you see in real matrices?
- If you know part of a matrix, and you know it has low rank, could we fill in the gaps? Filling gaps is called the matrix completion problem and is big business (see the Netflix problem) if you can do it well.

Links

- https://www.math.tamu.edu/~fnarc/psfiles/rank2005.pdf
- https://www.oneweirdkerneltrick.com/rank.pdf
- https://machinelearning.tf.fau.de/pdfs/mmmlisp19/6-matrix_completion.pdf