Nick Yallop – ny16281

Exercise 1:

**Matrix Inversion**

Report:

*Note: time measurements are the order of ten seconds that it took for the algorithm to find the inverse of a matrix, averaged over 100 random non-near-singular iterations and the effect of near-singularity was investigated by setting the last column to a normalised sum of the other columns, then adding some noise (for the “near” part). All sample matrices have uniform random cells between -5 and 5.*

**Inversion Algorithms**

Task 1 – Cramer’s Rule: For my own routine for Cramer’s rule for matrix inversion I defined a function DET to call itself in the case of an NxN matrix for N>2 and returns A[0,0]\*A[1,1]-A[0,1]\*A[1,0] for N=2. It also stores the overall determinant and the determinant of each 2x2 for the construction of the final inverse.

Dimension: 4 5 6

Cramer’s Time(s): ~0.001 ~0.1 ~10

[Average power of 10 for the solution time of a DxD matrix of uniformly random cells from -5 to 5 ]

The effect of dimension number on Cramer’s time is exponential, so Cramer’s rule is incredibly ineffective for large matrices, even though the simple solution and iterative method allows for minimal inaccuracies (measured by the sum over each [(inverse\*input)-(identity)] term). Fortunately, though, due to all the calculations being simple multiplication and addition of the cells, the matrix being near-singular has a negligible effect on the Cramer time, but near-singular matrices can have Cramer inaccuracies greater that (e-10).

Task 2 – LU vs SVD: LU decomposition is the break-down of the input matrix into a lower-triangular (L) and upper triangular (U) matrix such that LU=input. From there the inverse is far easier to calculate (input)-1 = U-1L-1. Each term in the input matrix is defined by a unique linear combination of U and L terms allowing for an iterative method to solve each L, U cell with high accuracy (that does compound over dimensions, though) and from there it’s a standard iterative method for the inverse of the triangular matrices

Dimension: 2 3 4 // 10

LU time: ~0.001 ~0.001 ~0.001 // ~0.001

[Average power of 10 for the solution time of a DxD matrix of uniformly random cells from -5 to 5]

[N.B. Some of the measured times were so small the ‘time’ package quoted it as 0.0]

The LU decomposition method led to far shorter computation times even at D=5 and, while there is not enough data to establish a relationship between LU time and dimension number, it clearly isn’t as rapidly exponential and maintains uncertainty under (e-13) so is a far better choice for larger matrices than Cramer’s rule. One downside of LU decomposition is that the resulting error rises exponentially as the matrix becomes singular, though this effect is negligible unless the matrix can be referred to as “close-singular”.

SVD decomposition involves finding the eigenvectors of the two spaces defined by AAT and ATA and their eigenvalues. The matrix of the eigenvectors of AAT is U and for ATA it’s V. Each vector in U should have the same eigenvalue as a vector in V. The eigenvalues of AAT and ATA are known as the “singular values” of the matrix and in the case where one of these values tends towards being negligible compared to the others the matrix tends towards being singular. To reconstruct the original input matrix, one simply needs to put the positive square roots of the singular values along the diagonal of an empty matrix and line the corresponding eigenvectors in U and V up to multiply by the right singular value root.

The SVD decomposition of a matrix can be useful for researching the behaviour of certain matrices. For example, due to U and V being matrices of unit vectors their determinants are 1, so the entire determinant must come only from the diagonal matrix of singular value roots and the determinant of a diagonal matrix is just the product of the diagonal terms. The relative size of these singular values also give one the best way to investigate how singular a matrix is (The determinant being near zero is an irrelevant condition because any matrix can be multiplied by a constant c such that DET(cA) = cNDET(A)) – one singular value being small/negligible compared to the others *is* near-singularity.

One more application of this is when the initial matrix is NxM for N=/=M. In this case an inverse is too loosely defined to be of any use for the LU or Cramer methods, and the determinant is non-logical. With the use of SVD, though, one can investigate the behaviour of non-square matrices. Since ATA and AAT will both be square matrices defining N basis vector on RN or M basis vectors on RM, and will have exclusively positive eigenvalues, one can use known iterative methods for eigen(vectors)/(values) in these two new spaces. From here one can argue a parallel to the determinant would be the product of the singular values that appear from both AAT and ATA and one can still construct the explicit inverses for each of these spaces and, using which singular values both spaces share, investigate how those spaces map between each other. i.e in the case of a near singular matrix due to 2 columns being nearly the same, the singular values that will appear will be those of the main columns, one for the vector of nearly-shared value, and one negligible one for the difference between the two.

For my SVD method the times stated below are useful as they show how quick the method is, though for each of these (due to the deliberate non-singular nature of the matrices) the inaccuracies are of the order of 10Dim.

Dimension: 2 3 4 // 10

LU time: ~0.001 ~0.001 ~0.001 // ~0.001

[Average power of 10 for the solution time of a DxD matrix of uniformly random cells from -5 to 5]

[N.B. Some of the measured times were so small the ‘time’ package quoted it as 0.0]

Thus, for the next section a succinct matrix inversion function had to be put together. For this I chose to calculate the singular values for the input matrix first. Then, if one of them is far smaller than the others SVD is our best approach, if not then LU decomposition, and in the case where that still results in too large an error – Cramer’s rule.

**PHYSICAL PROBLEM**

For the physical problem one needs to solve the equation ΘT + W = 0 (multidimensional Newton’s 1st Law), where T = [T1, T2, T3] (where Ti is the tension in the i’th drum), W = [0, 0, -9.81\*mass] and Θ is the matrix of unit vectors in the directions of the wires. N.B. there is a solution for T because there are 3 drums in 3D space – any more (dimensions)/(drums) and there would be (no)/(a space of) solutions. Thus, to solve for T one needs to find Θ-1W. For a given position then, one finds the vectors from the position to the drums, normalises them, puts them in a matrix, finds the inverse of that matrix, then multiplies it by (-W).

As an extended test of my final “inverter” function I created an iterative finder to find the maximum tension of any given wire in the range of positions defined by the stage. This resulted in many, many matrices needing to be inverted, some of which were incredibly near-singular. My method coped with each of these and provided the same result as when np.linal.inv() was used to test (this function appears nowhere in the final code, as was used for checking only). One problem that arose was that of positive tension solution, which would be impossible using supportive strings, so those solutions were discarded because, as earlier, there is only a single unique tension solution for each unique acrobat location.

**Limitations**

One limitation of the code I handed in is the lack of capability for the SVD decomposition to assess MxN (M=/=N) matrices, which meant I couldn’t investigate the effect of degenerate maps - but I could still find the singular values of a (fully)/(near) singular square matrix and investigate the effect of maps onto degenerate spaces. However, from there, there was still quite high compound errors for the inverse (where it existed for near-singular), so a distinct shortcoming of the method was the struggle of finding the inverse of very near-singular matrices to an acceptable accuracy.