Each implementation of MatrixGen calculated the proportions for how a company’s ownership should be divided and stored the results in a list. The matrix was then rotated before it was returned so that each list within the 2d array now represented how much the company controlled of others.

Initial implementation of the MatrixGen() function ensured that no company had a sum of ownership > 1, and the matrix columns would sum to 1. This was achieved by using random integers to generate a set of numerators (one for each company) which where then divided by a denominator which was equal to sum of numerators. The problem with this function is it tends to create ownership rates which cluster around 1/n (where n = number of companies). This means that as n increases it is very unlikely that the function would produce a matrix where any company would have direct ownership.

The second implementation MatrixGen2() used a different approach, ensuring that each column’s sum < 1 by tracking how much of a company’s share had already been claimed. This approach has the advantage that there is 50% probability that the first share apportioned will be greater than 0.5, and an even higher probability at least one company will have a controlling share.

The third implementation MatrixGen3() used the same foundation as MatrixGen(2) but included an additional step which added the unclaimed share of a company to another (randomly selected ) company’s controlling share. The function was adjusted to divide it’s shares between only n – 1 companies (because 0 would be inserted in after in the diagonal position).

I used np.zeros() to create a matrix of size 10^8 using the smallest possible size for each element in the matrix (8bit unsigned int)

matrix = zeros((100000000, 100000000), dtype= 'uint8')

Got the following error code:

python(3081,0x7fff745f3300) malloc: \*\*\* mach\_vm\_map(size=10000000000000000) failed (error code=3)

According to the error code, this required 10^16 bytes = 10^7GB = ten million gigabytes.

Testing np.zeros() on a matrix size 10^7 did work. Using the .nbytes method the matrix size was measured at 10^14 bytes = 10^5 GB = 100,000,000 gigabytes.

Consider using Scipi’s sparce matrix.

Look at numbers in row > 0.5

For each control, take column and look at numbers in row

Recursively:

Have set of controlpositions

Add to the set by searching row, if found add to set\

Consider using dtype to optimise the matrix size in memory

: a = np.arange(2708000000, dtype=np.int8)