The Determinant Chance of Being Zero

Mario Velasquez

December 13, 2018

Abstract

The topic I wished to investigate is the possibility of the determinant being zero with having a few factors to test. These factors where tested in a computer program that will insert random integers from specific controlled sets of elements. The results found showed that particular length of sets , size of the matrix and the possibility of zero being an element hold great impact on whether or not the determinant will be zero. The test also showed a negligible factor that could affect the determinant being zero for any given matrix .The tests were done a great multiple of instances that can concretely show patterns in the data.

1 Opening

The experiment seeks to observe the results of determinants when they hold random integers in their columns and rows. The experiment will look at 2x2 , 3x3, and 4x4 determinants. With a specific range of random numbers. With particular sets , increased range of numbers, and even the effect of a single particular element . It aims to see if any have a result on the probability of the determinant being zero.

The method that will be used will be finding the determinant through simulation of the co-factor expansion method. It will be simulated in the computer language c++ on an online compiler. It will hope that the results can be of insight or confirmation to anyone who already uses matrices and would like to know some of the factors that can increase the possibility of the determinant being zero.

2 Experiment Process and Functions

2.1 Experiment

The experiment itself is a computer program that will be ran three times. Once for the 2x2 matrix data, once for the 3x3 matrix data, and once for the 4x4 matrix data. The experiment will have something called trials. Trials will refer to the amount of times the determinant will be calculated for each NxN matrix. Each Trial will load up new random integers onto the matrix, these numbers will be based on a specific set of possible integers. After all the trials are done there will be a recorded average percentage that represents the likelihood of zero being the determinant of NxN matrix. Each one million trials and average of

the total trials will be called a run. There will be a total of one million trials for each run. We will do one hundred runs to have an additional average for the determinant being zero based of the total runs and the standard deviation will be based of the average found through the runs. There will be 100 runs for each set. There will be a total of 12 sets, each with a specific range of possible integers. The 2x2 , 3x3, and 4x4 matrices will all be subjugated to all 12 sets , which hold 100 runs, and each run holding one million trials.

2.2 Simulation of co-factor expansion through a Function

The algorithm for finding the determinant is a recursive function. The reason why it is a recursive function in the actual programming instead of a specific method for each NxN, is because co-factoring expansion is recursive in nature. This means that an algorithm that can simulate the recursive nature will be much simpler to reuse when we change the matrix to any NxN we wish. The same algorithm will be used to solve the 2x2, 3x3, and 4x4 matrix's determinant to remain consistent and efficiently use one algorithm for all.

2.3 Random Integer Function

The second most important function is the random integer function. This function will be in responsible to create a random integer within the range of our desired range between two integers. Integers will be used to accurately obtain a determinant due to our set of integers not having issues with floating point errors. The following elements will be a possible choice for our random integer function for each specific set. Each specific set of numbers will be a possible element for each 2x2, 3x3, and 4x4 Matrix. All sets will be tested for each Matrix of NxN.
$set(A) = \{0, 1\}$ $set(B) = \{r|0 \leq r \leq 10\}$ $set(C) = \{r|0 \leq r \leq 100\}$
$set(D) = \{1, 2\}$ $set(E) = \{r|1 \leq r \leq 11\}$ $set(F) = \{r|1 \leq r \leq 101\}$
$set(G) = \{-1, 0\}$ $set(H) = \{r| - 10 \leq r \leq 0\}$ $set(I) = \{r| - 100 \leq r \leq 0\}$
$set(J) = \{-2, -1\}$ $set(K) = \{r| - 11 \leq r \leq -1\}$ $set(L) = \{r| - 101 \leq r \leq -1\}$
These sets where carefully chosen to detect any patterns, such as the effect of having zero as a possible integer inside the matrix. Sets A,D,G,J all have the same quantity of elements. Sets B,E,H,K all have the same amount of elements. Sets C, F, I, L all have the same amount of elements.
Sets A,B,C, all contain zero and the rest of the elements are positive integers. Sets G,H,I, all contain zero and the remaining elements are negative integers. Elements D, E, F, do not contain zero and are all positive integers. Elements J,K,L do not contain zero and are positive integers.

2.4 Other functions that will be used

Other functions in the program , will be functions such as finding the percentage of a giving matrix in which its determinant will be zero. This average will be also be used in a standard deviation function, to know how much the average will deviate. A Function such as displaying the matrix will be use full when verifying the determinant of displayed matrix with random integers from a specific set. Not necessarily a function , but there is an important loop where each trial will insert random integers from a given set

into the matrix in the main.cpp.

3 How we verify

3.1 Running a short form

By running a short version of the program , with the information being displayed for each matrix. We can clarify that the average, standard deviation and most importantly the determinant are all being calculated correctly.
There will be displayed 3 runs with 2 trials for a 2x2 matrix with $set(A)$ for the checking of the average and standard deviation as well as to visually see that no random integer will get out of scope.
For the verification of the determinant we will use a custom set where $set(M) = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$ has positive integers, negative integers, and zero.

3.2 Verifying the determinant function

```
Computing 2 trials for a 3x3 Matrix...
With random numbers between -5 through 5 ...


0, 1, 0,
5, -4, -2,
-5, 3, 3,
The determinant is: -5

-3, -1, 2,
4, 0, -4,
-3, -1, -5,
The determinant is: -28
```

$$X = \begin{bmatrix} 0 & 1 & 0 \\ 5 & -4 & -2 \\ -5 & 3 & 3 \end{bmatrix} = \begin{vmatrix} 0 & 1 & 0 \\ 5 & -4 & -2 \\ -5 & 3 & 3 \end{vmatrix}$$

$$= +0\begin{vmatrix} -4 & -2 \\ 3 & 3 \end{vmatrix} - +1\begin{vmatrix} -2 & 5 \\ 3 & -5 \end{vmatrix} + 0\begin{vmatrix} 5 & -4 \\ -5 & 3 \end{vmatrix}$$

$$= 0((-4 * 4) - (-2 * 3)) - 1((5 * 3) - (-2 * -5)) + 0((5 * 3) - (-4 * -5))$$

$$= 0((-16) - (-6)) - 1((15) - (10)) + 0((15) - (20))$$

$$= -(5) = -5$$

Determinant of X is -5, The algorithm runs in a recursive manner and will run itself recursively for any NxN , just like how it solved the 2x2 matrices within

itself, it will solve the 3x3 cases and 2x2 of which it holds for a 4x4 matrix. Therefore the algorithm will be success full for any N x N.

3.3 Verifying the percentage

```
Computing 2 trials for a 2x2 Matrix...
With random numbers between 0 through 1 ...


Run Number: 0

The determinant is: 0

The determinant is: 0

How many times zero was the determinant presented: 2
Percentage of times zero was the determinant: 100%
Run Number: 1

The determinant is: 0

The determinant is: 1

How many times zero was the determinant presented: 1
Percentage of times zero was the determinant: 50%
Run Number: 2

The determinant is: 1

The determinant is: -1

How many times zero was the determinant presented: 0
Percentage of times zero was the determinant: 0%

Repeated the test 3 times to obtain the following percentages:
Percentage average of zero being the determinant ≈ 50% With a sta
ndard deviation of σ =  +/- 40.8248%
```

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

$$\overline{x} = \frac{(100\% + 50\% + 0\%)}{3}$$

$$\overline{x} = 50$$

$$\sigma = \sqrt{\frac{1}{3}((100-50)^2 + (50-50)^2 + (0-50)^2)}$$

$$\sigma = \sqrt{\frac{1}{3}(5000)}$$

$$\sigma = \sqrt{\frac{5000}{3}}$$

$$\sigma = 48.824829$$

The terminal shows the exact same average and standard deviation. It is evident these functions can scale for the algorithms in order to fit the required runs and trials for the experiment at hand.

4 Results

Results are in percentages on the left. Standard Deviation on the Right

Table 1: Results

	2x2	3x3	4x4	$\sigma 2x2$	$\sigma 3x3$	$\sigma 4x4$
S(A)	62.4961%	66.0152%	65.5534%	0.046683%	0.049443%	0.054673%
S(B)	4.9098%	1.5925%	0.34448%	0.021287%	0.010282%	0.006107%
S(C)	0.089003%	0.003411%	0.000081%	0.002998%	0.000585%	9.24E-05%
S(D)	37.4934%	48.4365%	51.0625%	0.047019%	0.05246%	0.052332%
S(E)	2.17824%	0.890236%	0.23671%	0.016077%	0.010176%	0.006104%
S(F)	0.05091%	0.002607%	0.00006%	0.002454%	0.000572%	6.63E-05%
S(G)	62.5124%	66.0178%	65.5813%	0.051661%	0.043405%	0.048501%
S(H)	4.91009%	1.58907%	0.373502%	0.022051%	0.0118%	0.005933%
S(I)	0.088693%	0.003416%	0.000075%	0.003197%	0.000529%	8.29E-05%
S(J)	37.5053%	48.436%	51.0719%	0.049663%	0.050586%	0.0509%
S(K)	2.21762%	0.889719%	0.237955%	0.012188%	0.009116%	0.004442%
S(L)	0.050635%	0.002644%	0.00006%	0.002442%	0.000496%	4.9E-05%

The results in a bar chart manner for comparison use

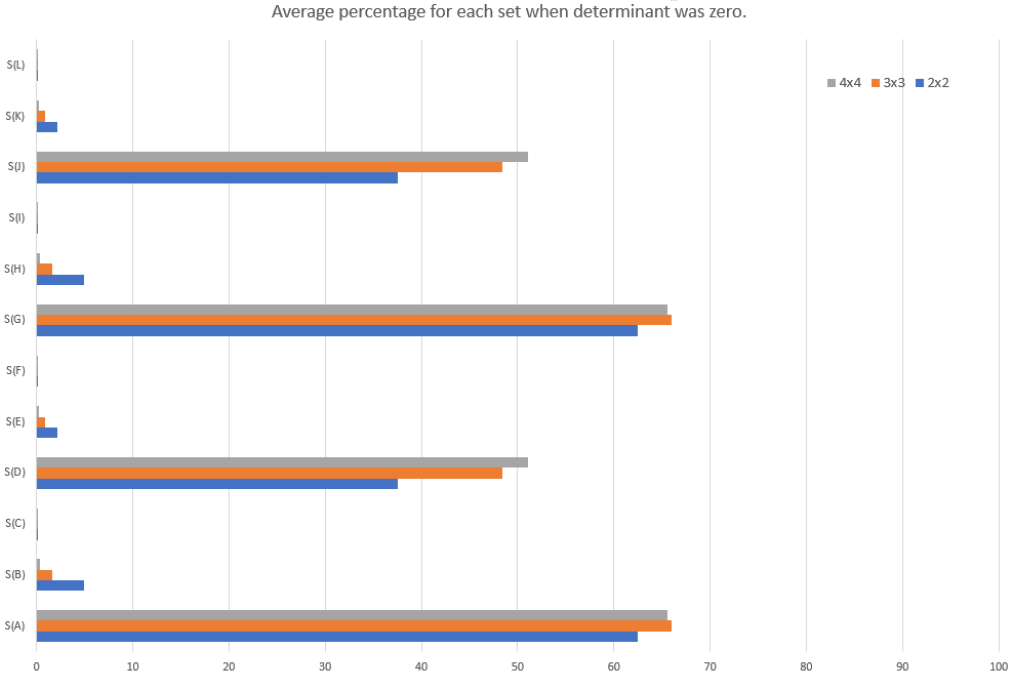

Table 2

5 Closing

Upon analyzing the results it is apparent that sets A, D , G , and J all have significant greater chances of having a determinant of zero. One thing all of these sets share is the range between numbers is 2. Each set has two elements , which could be why they show increased chances of having a determinant of zero ac cross all 2x2, 3x3 and 4x4 matrices. However, sets J and D have a lower percentage compared to A , G , despite all four having incredibly high percentages. It is to note that J and D do not have zero as a possible element. The same trend can be seen on sets B , E , H , K with similarity possible elements to be chosen at random, where sets K , and E sets have no possible element of zero and thus have reduced percentages compared to sets B and H whose sets contain the element zero. Sets B , E , H , K which has 11 possible elements show decreased chances of producing a determinant zero at random compared to sets A, D, G, and J. It is evident when looking at sets C , F , I and L that increased range of possible elements in a set at random heavily decrease the chance of the determinant being zero. When excluding sets A , D , G, and J and observing all other sets, it is evident that when increasing the N in an NxN matrix will most likely decrease the chance of obtaining a determinant of zero.

When comparing sets D and J where D holds all positive numbers and J holds all negative numbers. We can see that the impact of whether the sets are negative integers or positive integers are almost negligible in being a significant factor to result in a determinant being zero.

When a matrix has a zero as an element it has increased chances of having a determinant of zero. This makes sense due to how the determinant is found, when an element is zero the multiple methods of obtaining the determinant such as the co-factor expansion method , the likelihood of running into a zero could simplify down to more zeros , eventually to the very determinant. Similarly the higher the dimensions of a Matrix , the more opportunities for the Determinant to be calculated as zero even without the need of the element of zero itself. It is especially true when the set of possible elements is larger than 2.

This could mean for any science that uses matrices. The chances of the determinant being zero result in a Matrix having many more traits or lack off. The result is a matrix who is linearly independent and , may or not be, of use for particular purposes. Where its volume could be shranked into a single vector or a single point. The experiment showed the chances of this happening. Seeing the results show how different variables can lead to a determinant of zero. Which can be more impactfull, such as the situation where having a zero in their matrix , increases the chance of having a determinant of zero. Depending on the the particular use , one could be striving for a determinant of zero , or one could be striving for a determinant that is anything but zero or any of their own purposes and uses. By having this data one can now expect it and have probability depending on the different variables tested for. These factors can be applied and seen when working with many different matrices.

6 Source Code & Comments

The original intention of the code was to be able to dynamically change N in a NxN matrix. In c++ , when declaring multi-dimensional arrays, you must have a constant that can not be changed. The way around this was to declare a dynamic two dimensional array with heap allocation , where it would require memory management and It would have been even slower dynamically changing the array and handling memory. The other method was to use multi threading to create a NxN matrix , but proved out of my skill. At this point trying to implement the dynamic array , believing i could change the N value after most of my source code was completed. I opted for a different solution to change the NxN matrix. By only having one un-commented 2 dimensional array within lines 40-48 on main.cpp. You must also change both constants in line 9 of function.h and line 6 of main.cpp. Throughout the soarce code there are many iostream comands that where used to display information visually and helpful for the "Verifying" part.
Nevertheless this did not affect the results in the experiment.
Source code was written in https://repl.it/

main.cpp

```
#include<iostream>
#include"functions.h"
using namespace std;

int main(){
const int size = 4;//Must change in functions.h aswell

// Choose the random numbers
srand(static_cast<unsigned int>(time(0)));

  int runs = 100;

  float sd[runs-1];

    //integers for the random numbers in the matrix
    int hi = 0;
    int lo = -1;
    //for function r(hi,low)

    int trials = 1000000;

    /*
    do {
      cout << "Largest Random Number: ";
      cin >> hi;
      cout << endl;
      cout << "Smallest Random Number: ";
      cin >> lo;
      cout << endl;
    } while (hi < lo);
    */
    cout << endl << "\nComputing " << trials << " trials for a " << size << "x" << size << " Matrix..." << endl
    cout << "With random numbers between " << lo << " through " << hi << " ..." << endl;
    //where i plan to store in the determinants results
    const int capacity = size*size;

    cout << endl;

  //Manually have to uncomment

  /*for use when size = 2*/
  //int matrix[size][size] ={{1,1},{1,1}};

  /*for use when size = 3*/
```

```
  //int matrix[size][size] ={{1,1,1},{1,1,1},{1,1,1}};

  /*for use when size = 4*/
  int matrix[size][size] ={{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1}};

for(int j = 0; j < runs; j++){

  //cout << endl << "Run Number: "  << j << endl;


  int storingDet[trials];

  for(int i = 0; i < trials; i++){
    //filsl up the array
    for(int row= 0; row < size; row++){
    for(int clm = 0; clm < size; clm++){
      matrix[row][clm] = r(hi,lo);
      }
    }

    //displayMat(matrix);

    //cout << endl << "The determinant is: " << determinant(matrix, size) << endl;
    storingDet[i] = determinant(matrix, size);

  }

  //cout << endl <<"How many times zero was the determinant presented: " << detzero(storingDet, trials);
  //cout << endl << "Percentage of times zero was the determinant: " << zeroperc(storingDet, trials) << "%";

  sd[j] = zeroperc(storingDet, trials);




    //pagebrk();
  }
  pagebrk();
  cout << "Repeated the test " << runs << " times to obtain the following percentages: " << endl;
  cout <<  "Percentage average of zero being the determinant  " << (avg(sd, runs)) << "%";
  cout <<  " With a standard deviation of  =  +/- " << stanD(sd, runs) << "%" << endl;

return 0;
system("pause");
}
```

functions.h

```
#include<iostream>
#include<cstdlib>
#include<time.h>
#include <iomanip>
#include <cmath>//for the standard deviation
using namespace std;


const int size = 4;//must change in main.cpp


//for random integers
int r(int high, int low){
if (low < 0) {
low = low * -1;
```

```
high += low;
return ((rand() % ++high) - low);

}
else if (high == low)
return high;

else if (low > 0){
return low+(rand() % (++high-low));
}
else
return ((rand() % ++high));

}

//functions just incase
//average MOST IMP one
float avg(float *ar, int cap) {

float sum = 0;

for (int i = 0; i < cap; i++) {
sum = float(ar[i]) + sum;
}
return (sum / cap);
}



//middle
int median(int *ar) {
int med = 0;



return med;
}

//most common
int mode(int ar[],int cap){
int doCount=0, count, popular = 10;

for ( int i=0 ; i<cap ;i++){
  count=0;

  for (int j=0 ; j<cap ; j++){

    if (ar[i]==ar[j]) count++;
  }

  if (count>=doCount){
    doCount=count;

    if (popular > ar[i])
        popular=ar[i];
  }

}

  return popular;
}
```

```
//determinant

int sxs= size*size;

// Unfortunately, no one can be...told what the Matrix is. You have to see it for yourself. - Morpheus
void displayMat(int ex[size][size]){
    for(int row= 0; row < size; row++){
      cout << endl;
    for(int clm = 0; clm < size; clm++){
      cout << ex[row][clm] << ", ";
    }
  }
}


void factor(int det[size][size], int tmp[size][size], int tmpa, int tmpb, int lw){
  int i = 0, j = 0;
  for (int row = 0; row < lw; row++){
    for (int clm = 0; clm < lw; clm++){

      if (row != tmpa && clm != tmpb){
        tmp[i][j++] = det[row][clm];

        if (j == lw - 1){
          j = 0;
          i++;
        }
      }
    }
  }
}
int determinant(int det[size][size], int lw){
  int answr = 0;
  if (lw == 1) return det[0][0];
  int tmp[size][size];
  int symbol = 1;
  for (int k = 0; k < lw; k++){
    factor(det, tmp, 0, k, lw);
    answr +=  det[0][k]* symbol * determinant(tmp, lw - 1);
    symbol = -symbol;
  }
  return answr;
}

int detzero(int a[], int size){
  int count = 0;
  for(int l = 0; l < size; l++){
    if(a[l] == 0)
    count ++;
  }

  return count;

}

float zeroperc(int a[], int size){
  float percentage;
  int count = 0;
  for(int l = 0; l < size; l++){
    if(a[l] == 0)
    count ++;
  }
```

```
  percentage = (float(count)/float(size))*100;

  return percentage;

}
//standard dev
float stanD(float arr[], int runs){
  float sum = 0.0, average = 0.0, deviation = 0.0;
  for(int i = 0; i < runs; ++i){
  sum += arr[i];
  }
  average = avg(arr, runs);
  for(int i = 0; i < runs; ++i)
    deviation += pow(arr[i] - average, 2);
  return (sqrt(deviation / runs));

}

void pagebrk(){
  cout << endl << "__________________" << endl ;
}
```