08/14/2014 MSAN 502 SUMMER, 2015

REVIEW OF LINEAR ALGEBRA: PROJECT

CANDIDATE NAME: ABHISHEK SINGH
USF-ID: 20363537

TABLE OF CONTENTS

1. BINARY MATRIX

We discuss the concept of Binary matrix in this project. Fundamentally any Matrix with only binomial entries (1 or 0) becomes a Binary matrix.

- *This may or may not be a Square Matrix*
- *Identity matrix is one such example*
- *It may or may not be invertible*
- *It may or may not have pivots in every column*
- *Other matrices with only 1 as the non-zero elements fall in this category too*
- *One example of a 4 by 4 Binary matrix would be:*

1	1	0	1
0	0	0	1
1	0	0	0
0	1	1	0

2. PROJECT DEFINITION

We have to figure out ***that by doing a coin toss for every entry of an $n \times n$-matrix whether it is* 0 *or* 1*, can you expect this matrix to be invertible?***

Now to answer this question, we take 2 computational approaches namely:

- ***Brute Force Technique:*** *of using the entire sample space for determining the probability of Invertibility for a given order of a Matrix*
- ***Sampling technique:*** *when using a higher order matrix with bias & sampling space as parameters to determine the probability of Invertibility of this order. Here we restrict our study to* ***Binary Diagonal matrixes*** *only. An example of a Binary diagonal matrix would be:*

1	0	0	0
0	0	0	0
0	0	1	0
0	0	0	0

And we take a **Theoretical approach**, for any order matrix and try to estimate the chances of its Invertibility for a given order of matrixes.

3. COMPUTATIONAL TECHNIQUES

1. Brute force Technique (USING PYTHON)

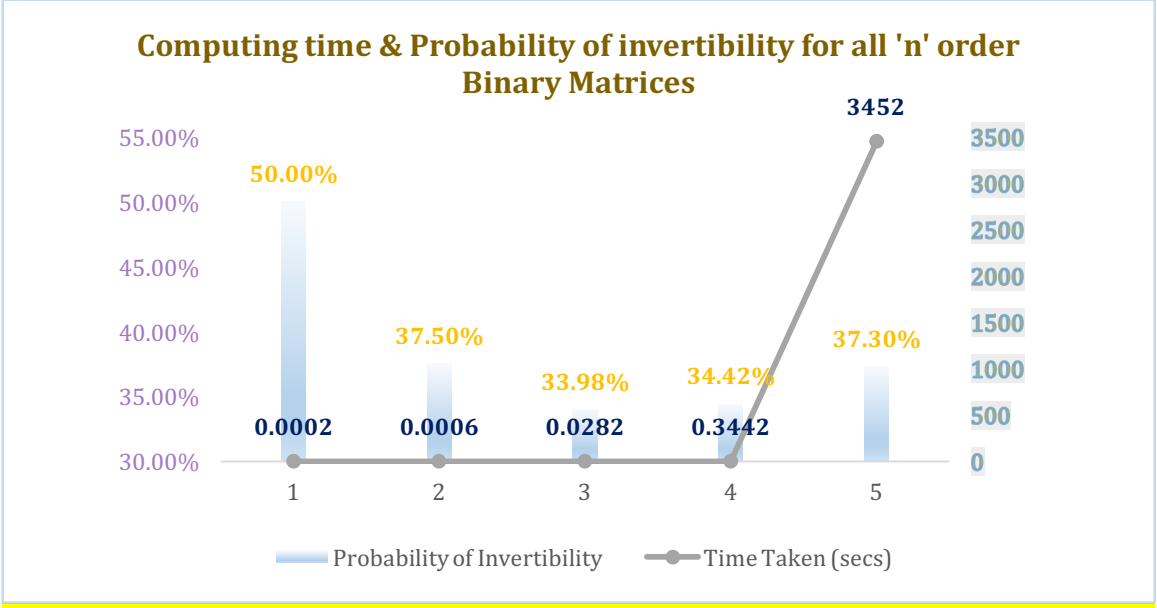
We have used a python code to determine Invertibility probability for low order matrixes (n <= 5). This is done by calculating the ration between the total number of binary matrixes possible (A) and calculating the number of binary matrixes invertible (B), such that **B ⊆ A** .
Probability of Invertibility of Matrixes **(P) = B/A**

```
#I have used the below url's to help do this
# "http://stackoverflow.com/questions/21407620/numpy-create-fill-with-random-binary-data"
# "http://stackoverflow.com/questions/5622976/how-do-you-calculate-program-run-time-in-python"

#Mathematical operations
import numpy as np
#for permutations
import itertools as iter
#To track speed of function execution
import timeit

#Generating a random matrix through a function
def matrix_generator(n):
    start = timeit.default_timer()
    length = n**2
    combinations = [list(seq) for seq in iter.product("01", repeat = length)]
    Invert = []
    Non_Invert = []
    for j in range(0, len(combinations)):
        a = [int(i) for i in combinations[j]]
        matrix = np.mat(a).reshape((n, n))
        if np.linalg.det(matrix)==0:
            Non_Invert.append(matrix)
        else:
            Invert.append(matrix)
#Printing the total number of binary matrixes possible
    total_matrixes = (len(Invert)+ len(Non_Invert))
    print "The total number of binary matrixes is ",total_matrixes
#Printing the total number of Invertible binary matrixes possible
    total_Invertible = len(Invert)
    print "The total number of Invertible binary matrixes is ",total_Invertible
#Printing the probability of getting an invertible matrix
    probability = float(total_Invertible)/total_matrixes
#Time for function execution
    stop = timeit.default_timer()
    print "The probability of Invertibility at rank ",n,"is ",float(probability)
#Calculating the time taken to perform this operation
    print("--- %s seconds taken---" % (stop - start))


#Enter order of the matrix in function parameter
matrix_generator(2)
```

Performance parameters

Below is a table of the computational performance of the Brute Force algorithm on the following orders of matrixes

Order of a Matrix	Probability of Invertibility	Time Taken (secs)	Total Binary Matrixes	Total Invertible Matrixes
1	50.00%	0.0002	2	1
2	37.50%	0.0006	16	6
3	33.98%	0.0282	512	174
4	34.42%	0.3442	65,536	22,560
5	37.30%	3452	33,554,432	12,514,320



for matrix orders greater than 5, the computational load is too high for current systems to handle. We either need high speed and high capacity servers to help for evaluations or a cloud based service could prove handy.

The smart & inexpensive way here would be to do random sampling repeatedly to arrive at values. This is what I have done next.

2. Selective Sampling (R CODE)

Scope

For higher order matrixes we have taken random sampling to generate an interval of their probabilities. This has been done because, the computer struggles to generate, store, compute and tabulate the probability of higher order matrixes (n>5).

Thus random sampling proves to be the best bet here. This helps us in the following ways:

1. To maintain randomness and give us an unbiased estimate
2. To calculate values, without damaging or overloading the computer RAM
3. It saves over 99% time taken otherwise for higher dimensionality matrixes
4. This can be generalized for a higher order Binary matrix (n) too

Methodology

1. We have taken 1000 samples of binary matrix of a fixed order
2. For each of the 1000 samples we can generate 1000 Binary matrixes, hence 1 sample contains 1000 binary matrix
3. For each sample I compute the number of matrixes which are invertible **(Ni)**
4. For the number of matrixes invertible, we get the probability of Invertibility of the sample (p) = **Ni/1000** as our sample is 1000 long and we **Ni** number of invertible matrixes
5. Since we have 1000 samples we get 1000 probabilities, with each sample giving a probability based on 1000 such matrixes
6. Once we have obtained an array of 1000 probabilities, we compute a Confidence Interval for the probability estimate of the population of 'n' order Random binary matrixes
7. We compute the mean, Standard deviation & standard Error (SE) of these 1000 probabilities
8. Having computed the above, we estimate Lower Confidence Interval (LI)= Mean – t*SE
   And Higher Confidence Interval (HI)= Mean + t *SE
   At 95% confidence Interval using a t- distribution

R Code (Random Sampling)

```
"
A function that takes the order of matrix (n) & bias (p)
as parameters, and gives the probability of
its invertibility
Sample binary numbers to get a n by n matrix
and generate a 1000 samples of 1000 such matrix per sample
"
prob_binary_matrix <- function(p = .5, n = 6) {
 start.time <- Sys.time()
 all.probs <- NULL
"for generating 1000 probabilities from 1000
sampling exerisizes"
 for (i in 1:1000) {
"For each sample we generate 1000 binary matrixes
 and estimate the probability of their invertibility"
  inv.length <- NULL
  for (j in 1:1000) {
#Sampling with bias 'p'
   binary <- sample(c(0, 1), n * n, replace = TRUE
              , prob = c(p, 1 - p))
   mat <- matrix(binary, n, n)
   inv.length[j] <- ifelse(det(mat) == 0, 0, 1)
  }
  a <- length(inv.length[inv.length == 1])
  probability <- a/1000
  all.probs[i] <- probability
 }
"
Generating confidence intervals for the probability
of the samples, we generate LI & HI and use a t-distribution
at 95% confidence for 999 degrees of freedom
"
 mean.probability <- mean(all.probs)
 sd.probability <- sd(all.probs)
 se.probability <- sd.probability/sqrt(1000)
 LI <- mean.probability - qt(.975,999) * se.probability
 HI <- mean.probability + qt(.975,999) * se.probability
 cat("probability of invertibility for ", n ," order matrix is ")
 cat(LI," and ",HI)
 end.time <- Sys.time()
 time.taken <- end.time - start.time
 cat("time taken is ", time.taken)
}
```

To run the function for 7 order matrix with probability of 0 as .4

```
prob_binary_matrix(7, .4)
```

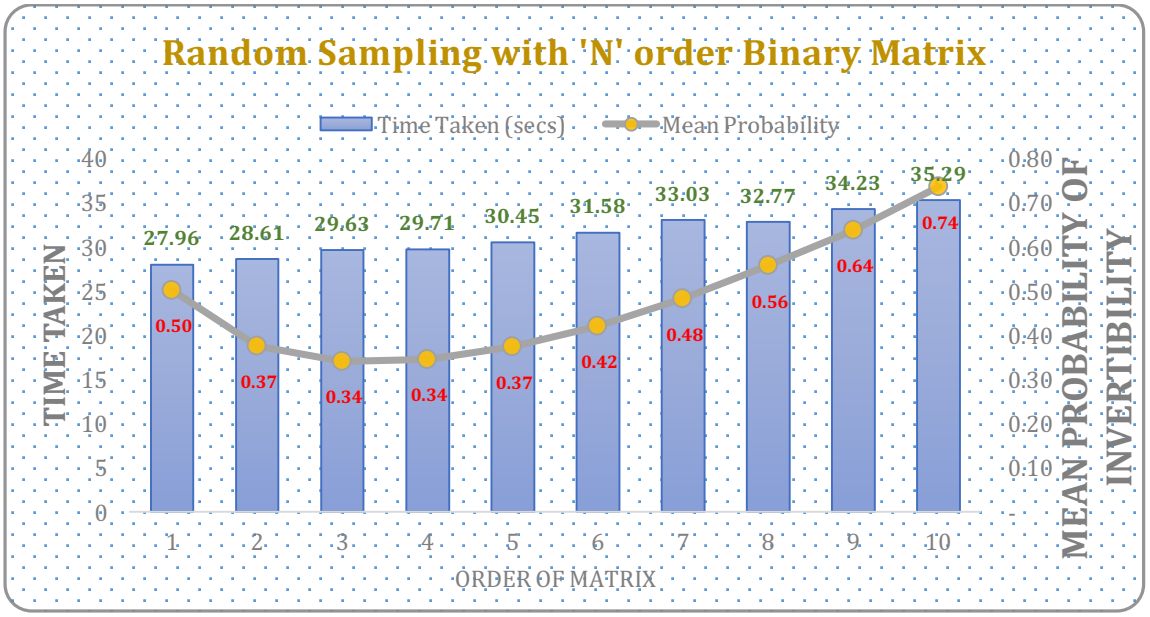
To run the function for **k** order matrix with probability of 0 as p

```
prob_binary_matrix(k, p)
```

Performance parameters

Below is a table of the computational performance of the Random sampling algorithm on the following orders of matrixes **(For bias = .5)**

Order of a Matrix	Lower Interval (A)	Higher Interval (B)	Mean Probability AVG(A,B)	Time Taken (secs)
1	0.50	0.50	0.50	27.96
2	0.37	0.38	0.37	28.61
3	0.34	0.34	0.34	29.63
4	0.34	0.35	0.34	29.71
5	0.37	0.37	0.37	30.45
6	0.42	0.42	0.42	31.58
7	0.48	0.48	0.48	33.03
8	0.56	0.56	0.56	32.77
9	0.63	0.64	0.64	34.23
10	0.74	0.73	0.74	35.29



4. THEORETICAL TECHNIQUE

1. Probability of Invertibility of Binary Matrix in Field of 2nd order (F2)

I have used the below urls to help explain this:

- "http://math.stackexchange.com/questions/54246/probability-that-a-random-binary-matrix-is-invertible"
- "https://docs.google.com/file/d/0BytCi6b1o1EOYWFkNF95NkdndUk/edit?pli=1"

for a **n** order square Binary matrix, the first row can be linear independent in $(2^n - 1)$ ways (where all elements being zero is linear Dependency, so we substract this case).
Linear Independency (1^{st} row) = (2^n -1)

By same logic, the 2^{nd} row can get linearly dependent in 2 ways. For 2^{nd} row the first element can take 2 values and if all other elements are zero then it becomes linearly dependent. So there are two ways of becoming linearly dependent. So:
Linear Independency (2^{nd} row) = (2^n - 2)

Same way for 3^{rd} row there are 2*2 (4) ways of becoming linearly dependent as first 2 elements can take all 4 possible values (permutations of 0 & 1)
Linear Independency (3^{rd} row) = (2^n - 2^2)
Linear Independency (k^{th} row) = (2^n - 2^{k-1})
Linear Independency (k^{th} row) = (2^n - 2^{n-1})

Now if a matrix has any row that is linearly dependent, its rank becomes less then the dimensionality. If the rank of the matrix is less then its total Dimensionality (n), its determinant becomes zero.

So probability, Matrix doesn't have a zero determinant is
probability = (Linear independency of 1^{st} row) * (Linear independency of 2^{nd} row)* …… * (Linear independency of n^{th} row)/Total matrixes possible **(2^{n*n})**
Number = $[(2^n - 1) * (2^n - 2) * (2^n - 2^2) * (2^n - 2^3)\ldots\ldots * (2^n - 2^{n-1})]$/ **($2^{n*n}$)**

$= (1-(1/2^n))* (1-(1/2^{n-1}))* (1-(1/2^{n-2})) * (1-(1/2^{n-3}))\ldots\ldots * (1-(1/2^{n-n+1}))$

$= (1-(1/2^n))* (1-(1/2^{n-1}))* (1-(1/2^{n-2})) * (1-(1/2^{n-3}))\ldots\ldots * (1-(1/2^1))$

Reversing the order of the products in the equation, above:
$= (1-(1/2^1)) * (1-(1/2^2)) * (1-(1/2^3)) * \ldots\ldots\ldots (1-(1/2^{n-1})) * (1-(1/2^n))$

$= {}^{n}\prod_{1} (1-(1/2^k))$

Probability of 1^{st} order (F_2) = $(1 - (1/2)) = (1-.5) = .5$
Probability of 2^{nd} order (F_2) = $(1 - (1/2)) * (1 - (1/2^2)) = (1 - .5) * (1 - .25) = .5 * .75 = .375$

For a given order of field **(q),** The **probability of Invertibility** would be $^{n}\prod_{1}(1 - (1/q^{k}))$

So at any point of time Probability of Invertibility (p) < (1 – 1/q)

So if q -> Infinity
P will tend to (1 – 1/infinity) => (1 – 0) => 1

2. Probability of Invertibility of a Diagonal matrix

For a Diagonal Binary matrix of n order, there would be total $\mathbf{2^n}$ **matrixes** possible. Here each element could take 2 possible values (0 & 1). So with n elements all occupying diagonal elements there are exactly 2^n possibilities.

Number of Invertible matrix, would be when each column is linearly independent. This is the case with each pivotal element having a non-zero value.

In case of Binary matrixes, these Non-zero values are 1.
So only a matrix with 1 at all diagonal positions, will form a Linearly independent matrix. This would be 1 possibility out of 2^n. Just as 1111 having a possibility $1/2^4$ in a 4 number string of binaries.

So probability of Invertibility for N-order Binary Diagonal matrixes = $\mathbf{1/2^n}$