PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

MAY 2020: IN SEMESTER ASSESSMENT (ISA) B.TECH. IV SEMESTER

_UE18MA251- LINEAR ALGEBRA

MINI PROJECT REPORT

ON

Randomised Matrix Multiplication and Associated Algorithms in Numerical Linear Algebra

Submitted by

1.	Name:	Khushei Meghana Meda	SRN:	PES1201800416
2.	Name:	Mugdha Pattnaik	SRN:	PES1201800058
3.	Name:	Mahesh D	SRN:	PES1201800210

Branch & Section : CSE, F

PROJECT EVALUATION

(For Official Use Only)

Sl.No.	Parameter	Max Marks	Marks Awarded
1	Background & Framing of the problem	4	
2	Approach and Solution	4	
3	References	4	
4	Clarity of the concepts & Creativity	4	
5	Choice of examples and understanding of the topic	4	
6	Presentation of the work	5	
	Total	25	

Name of the Course Instructor : Prof H. Ramesh Bhat

Signature of the Course Instructor :

Table of Contents

INTRODUCTION

A lot of mathematical computations in the fields of Data Science, Big Data, Machine Learning, etc. are done using matrices. The fields of Big Data and Data Science often deal with computations involving extremely large matrices, be it a basic operation like the multiplication of two matrices or more complex ones like low-rank approximation. Our survey is based on randomized algorithms which help to optimize or speed up the computations involving large matrices. The idea is to choose a part of the original input matrix and carry out the operations over that rather than the entire input matrix and still get an almost accurate answer.

On one hand, the naive approach to matrix multiplication of two matrices A_{mxn} and B_{nxp} gives us the result AB=[Cij].

$$(AB)_{ik} = \sum_{j=1}^{n} A_{ij}B_{jk} = C_{mp}$$

This result in terms of function needs to iterate at three levels, this utilizes 3 for loops which makes the procedure of order O(mnp). On the other hand, using algorithms such as RandNLA that use random sampling methods, which make more efficient use of computational resources, such as computation time, random access memory (RAM), approximate results can be achieved in a reduced duration with considerably high accuracy.

We will be explaining in detail how the randomized method reduces time complexity for multiplication of two matrices, and then highlighting some other use cases of randomized algorithms: low-rank approximation, and tensor-train approximation.

LITERATURE REVIEW

For this project, several research papers on randomized algorithms and general linear algebra were reviewed.

From the first[1] paper we read, we clearly understood the principle of randomized algorithms- to reduce a large matrix into a smaller one by selectively choosing some columns from the matrix by a method called “sampling”. We also understood the mathematical applications of where these algorithms were used. To explain briefly but clearly, the following inferences were made based on reviewing this paper.

1. To sample some columns(s columns, so *m x s)* from the original *m x n* matrix so that computations like matrix multiplication, regression and low-rank approximation are simplified.
2. Random sampling is not good because some columns are more significant than some others.
3. Using sampling probabilities proportional to square of length of columns leads to provable error bounds.
4. In this method, we do trials for sampling from a matrix A of order m x n. In each trial, we pick a random X belonging to {1,2,...,n} with Pr(X=j)=pj. We define pj =$|A(:, j)|^2$/ (Frobenius norm(A))^2 , for j = 1, 2, . . . , n.
5. Proving $CC^T \sim AA^T$ and using Jensen's inequality, prove error is minimal.
6. Applying the above results in various applications like matrix multiplication, low-rank approximation, Tensor Train approximation, and verifying the **time complexity of computation is greatly reduced.**

However, if this paper is to be critiqued from our current knowledge and exposure, it would be that this was a rather detailed case study of all the mathematical applications of randomized algorithms and had a lot of mathematical jargon, much beyond our current exposure.

This led us to approach the problem from a different perspective which was to take a step back and look at the high level applications of randomized algorithms. That’s when our second[2] reference was insightful. It showed us how linear algebra and randomized algorithms are secretly the basis of a vast amount of Machine Learning, Data Science and Big Data which are domains that computer science students come across almost daily.

Our next reference[3] is very vast and has almost everything pertaining to this topic, and is a good but extensive read, for all those who are interested. It is well divided into sections and starts off simple, to dive deeper into the explanation of concepts page by page. We got a step-by-step documented derivation and proof of matrix multiplication using randomized algorithms from these lecture notes, which we present in the next section. The only comment about this document would be that it is too vast to completely read and comprehend in a couple of days.

Watching Gilbert Strang's video[5] on this topic was extremely important for our understanding of this topic because he sternly emphasized that the randomized algorithms that we are talking about so much, do not help in reducing stand-alone/single matrices, but rather, they help in obtaining near optimal answers for certain matrix computations like matrix multiplication.

Another video reference[6] helped us understand to what extent the error is minimal when new matrices are derived from the original input matrices after the randomized algorithm is applied. The video also made a mention about the statistical aspect of confidence intervals that come into picture when such approximations are made. Our comment about this reference source would be that it was very intriguing to see the statistical validation of this whole process, but it could have been explored in more detail.

Finally, after repeated readings of the material we found including textbooks[7][8], we tried to summarize and validate the random algorithm with respect to matrix multiplication as per our own understanding. The idea is presented below, and a more mathematical form of it is presented in the section that follows this one.

The idea behind randomized numerical linear algebra is the intuition that certain concepts in randomization, for instance sampling, can be effective and **can scale to massive data.** But in order to do so, we have to agree on the fact that **we can’t always get exact solutions**, but can only get a closely **approximate answer**. And this is not a problem because it is possible to determine quality control parameters, like by how much percent our approximate answer differs from what we would have got with the actual input matrices.

We’d also know the confidence interval and how often this method would succeed in giving a near optimal answer. As our applications are in areas like predictive algorithms of machine learning, we don’t worry so much about near optimal answers.

REPORT

After thorough review of the above mentioned literature the following compilation best helps us recreate a randomised matrix multiplication algorithm backed by suitable mathematical proofs. Conventional matrix multiplication does not help us completely visualize how randomisation could be effectively used to minimise operation times. The new idea is that we can view the product AB as a sum of rank one matrices,i.e matrix multiplication as returning a matrix that equals the sum of outer products of columns of A and the corresponding rows of B.

$$AB = \sum_{i=1}^{n} A^{(i)} B_{(i)},$$

where each $A^{(i)}B_{(i)} \in R_{m\times p}$ is an $m \times p$ rank-one matrix, computed as the outer product of two vectors in $\mathbb{R}$n.

Moving on we need to determine the probabilities associated with each of these rank one matrices .These probabilities heavily determine how large our variance for the approximated product is.

Now coming to numerical linear algebra, it works on the principle of representing data in the form of a matrix and then performing operations on it. The idea of picking some columns from the original matrix is so that the picked entities can be representatives of the rest.

Understanding matrix multiplication of two matrices A, of order m x n, and B, or order n x p: The naive algorithm takes order **O(mnp)** because we have to take every row of A and every column of B and that takes order n, and we've to consider n times p such combinations because every row goes with every column. In the usual method, $(AB)_{ij} = \sum_{k} A_{ik}B_{kj}$.

Algorithm: Vanilla three-loop matrix multiplication algorithm.

Input: An $m \times n$ matrix A and an $n \times p$ matrix B
Output: The product AB

```
1: for i= 1 to m do
2:     for j= 1 to p do
3:         $(AB)_{ij} = 0$
4:         for k= 1 to n do
5:             $(AB)_{ik}$ += $A_{ij}B_{jk}$
6:         end for
7:     end for
8: end for
9: return AB
```

The **new idea** is that we can view the product AB as a **sum of rank one matrices**. Consider the matrix A to be a collection of its columns and matrix be to be collection of its rows, so the dot product AB can also be written as the sum of rank 1 matrices, i.e, the sum of outer product of the first column A (A_{*1})[$] and the first row of B(B_{1*}), outer product of the second column A (A_{*2}) and the second row of B(B_{2*}), and so on. If we have a probability for each of the terms in this sum expression, we can neglect the less probable ones because they would be of less importance. So if p_i is the probability of the ith term, then $\sum_i^n p_i = 1$; n is the number of terms.

To get the probability values, we sample c times. And in each time of sampling, we pick one of the rank 1 matrices. So for each of the c samples, we pick the ith rank 1 matrix with probability p_i, with replacement. And then we normalize the matrix with the lower p_i. After sampling c times, we have a sum of c terms, obviously. Now to get an estimate of what the sum of the original outer product matrices would have yielded, we just divide the sum of c terms by c to get $\frac{\sum_{t=1}^{c} A_{*j_t} * B_{j_t*}}{c}$; t denotes the trial number.
This is the estimator of product AB.

$ A_{*i} is the MATLAB notation for ith column of A and A_{i*} is the notation for ith row of A

So the algorithm basically consists of the following two steps:

1. Pick c columns of A with replacement to form the matrix C and the corresponding rows of B to create R.
2. Calculate CR and return.

Algorithm: RandomisedMatrixMultiplication algorithm.

Input: An $m \times n$ matrix A and an $n \times p$ matrix B, a positive integer c and probabilities $\{p_i\}^n_{i=1}$ such that $p_i \geq 0$ and $\sum_{i=1}^{n} p_i = 1$

Output: Matrices $C_{m\times c}$ and $R_{c\times p}$ such that $CR \approx AB$

```
1: for t= 1 to c do
2:     Pick $i_t \in \{1, . . . , n\}$ with probability Pr $[i_t=k] = p_k$, in i.i.d.
       trials, with replacement
3:     Set $C^{(t)} = A^{(it)} / \sqrt{(cp_{it})}$ and $R_{(t)} = B_{(it)} / \sqrt{(cp_{it})}$
4: end for
5: return $C$ and $R$
```

Basically, what we want to show for this algorithm is that

$$\begin{aligned} AB &= \sum_{i=1}^{n} A^{(i)} B_{(i))} \\ &\approx \frac{1}{c} \sum_{t=1}^{c} \frac{1}{p_{i_t}} A^{(i_t)} B_{(i_t)} \end{aligned}$$

For much of what follows, it will be convenient to express this and related subsequent algorithms in a standardized matrix notation that we will call the *sampling matrix formalism*. To do so, let S $\in \mathbb{R}^{n\times c}$ be a matrix such that

$$S_{ij} = \begin{cases} 1 & \text{if the } i\text{-th column of } A \text{ is chosen in the } j\text{-th independent trial} \\ 0 & \text{otherwise} \end{cases}$$

and let $D \in \mathbb{R}^{c\times c}$ be a diagonal matrix such that

$$D_{tt} = 1 / \sqrt{(cp_{it})} .$$

With this notation, we can write the output of the `RandomisedMatrixMultiplication algorithm` and what it is doing as $C = ASD$, $R = (SD)^T B$, and $CR = ASD(SD)^T B = ASSB \approx AB$.

So $AB \approx CR$. The columns and rows are picked with non-uniform probabilities and scaled appropriately.

Now we've established that CR is a reduced equivalent form of AB, of sorts. So let's get into the details of how to find the probabilities for each term that we spoke about initially. The first method would be to choose all the p_i values uniformly as 1/n, which means that for every t=1 to n, if we choose ith column of A to put in C, we have to choose ith row of B to put in R. But this is not a good way of going about it because it would result in high variance. So we prefer the second method of weighing by row and column norm,

i.e, $p_i = \frac{|A_{*i}||B_{i*}|}{\sum_j |A_{*j}||B_{j*}|}$.

This implies that probabilities are proportional to the l2 norm of matrix A times the l2 norm of matrix B. The denominator is for normalization. For t = 1,...,c if j_t column of A and row of B is chosen, the respective normalized values $\frac{A_{*j_t}}{\sqrt{c p_{j_t}}}$ and $\frac{B_{*j_t}}{\sqrt{c p_{j_t}}}$ are put in C and R, respectively. There is a square root over c because we are distributing the normalization, to put it in a simplified manner.

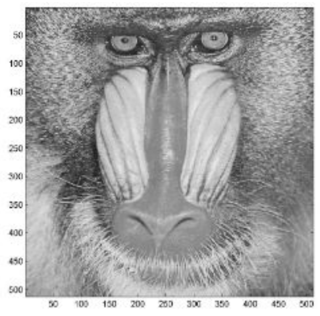
We can write this in terms of a sampling matrix S of size n x c. This sampling matrix will have, for every column, a single non-zero entry j_t at the position (j_t, t) whose value will be

$\frac{1}{\sqrt{c p_{j_t}}}$, i.e, $S_{j_t,t} = \frac{1}{\sqrt{c p_{j_t}}}$.

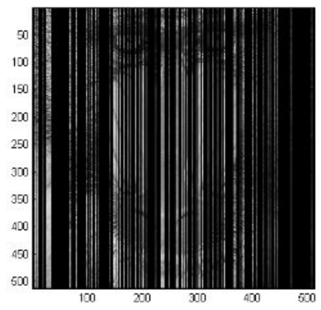
It is now easy to visualize that $CR = (AS)(S^t B)$. S is a common projection matrix for A and B.

RESULTS AND DISCUSSIONS

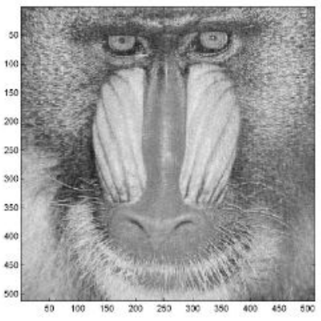
From reference [4]



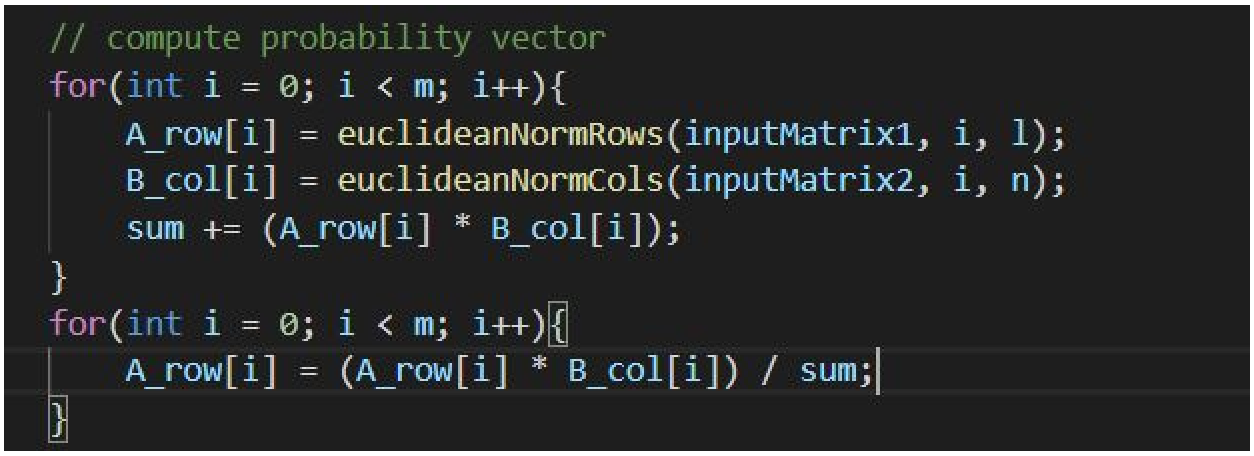
Original matrix A



Sampling



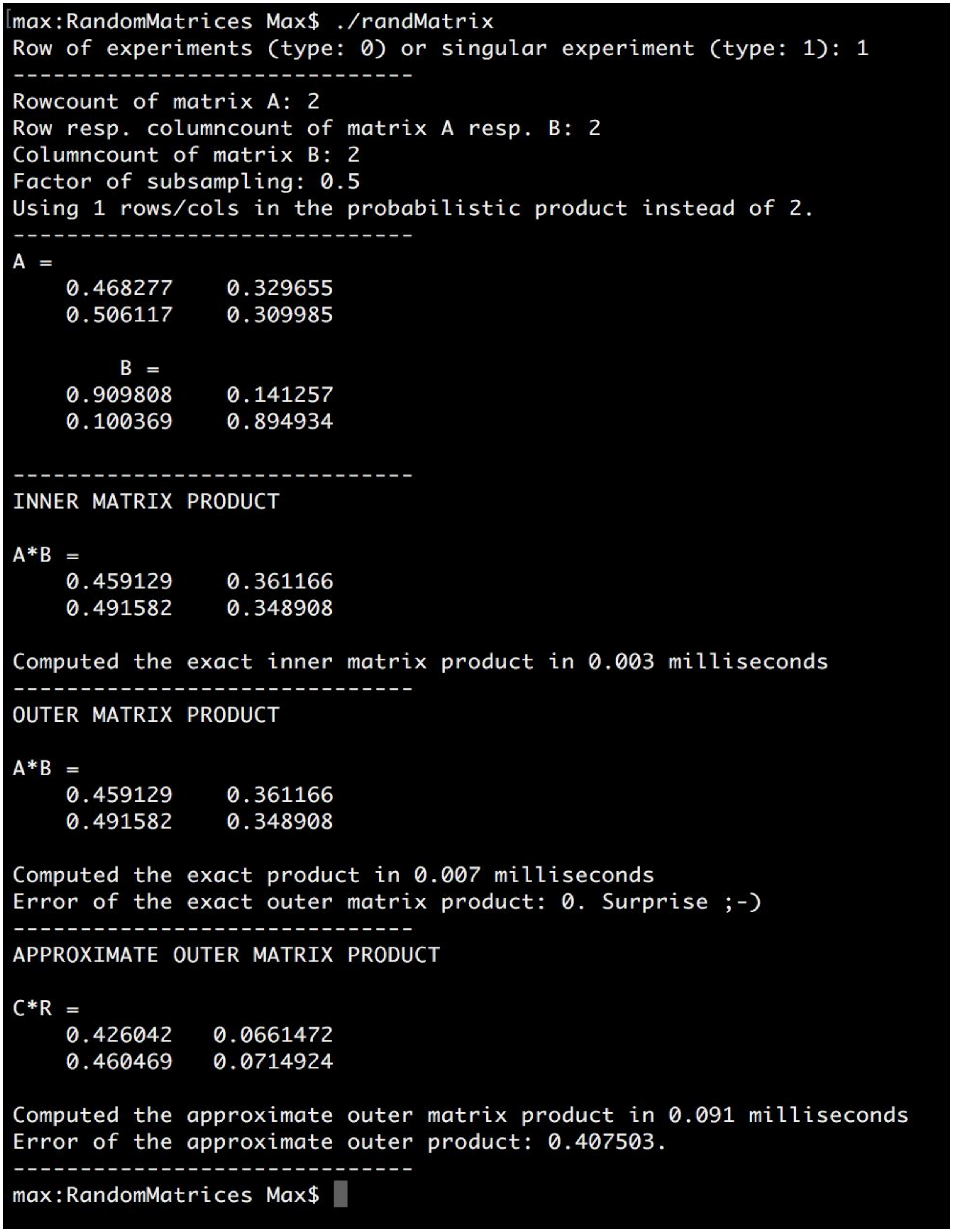
Final matrix CR



```
// compute probability vector
for(int i = 0; i < m; i++){
    A_row[i] = euclideanNormRows(inputMatrix1, i, l);
    B_col[i] = euclideanNormCols(inputMatrix2, i, n);
    sum += (A_row[i] * B_col[i]);
}
for(int i = 0; i < m; i++){
    A_row[i] = (A_row[i] * B_col[i]) / sum;
}
```

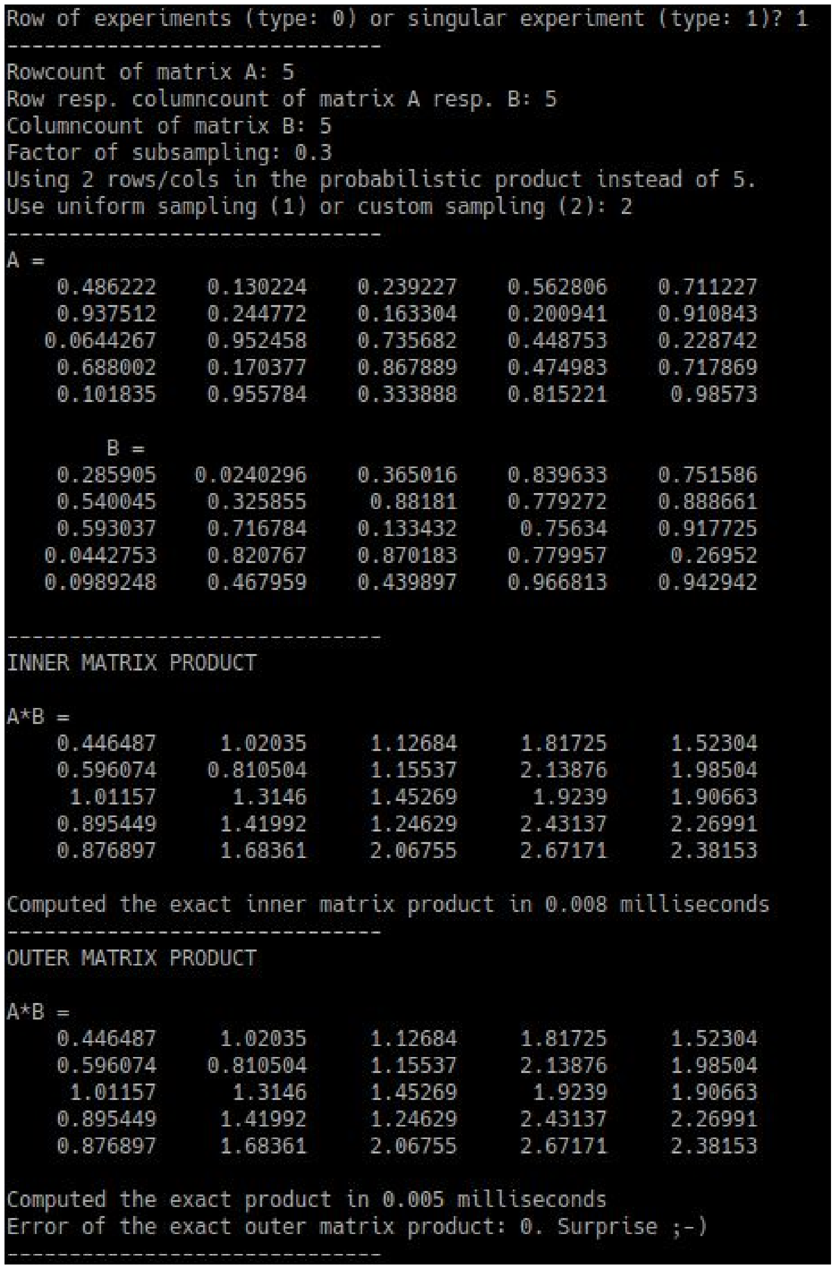
The code used to generate the probability vector.

Following is the output we generated after implementing the sampling randomised matrix multiplication algorithm using C++.

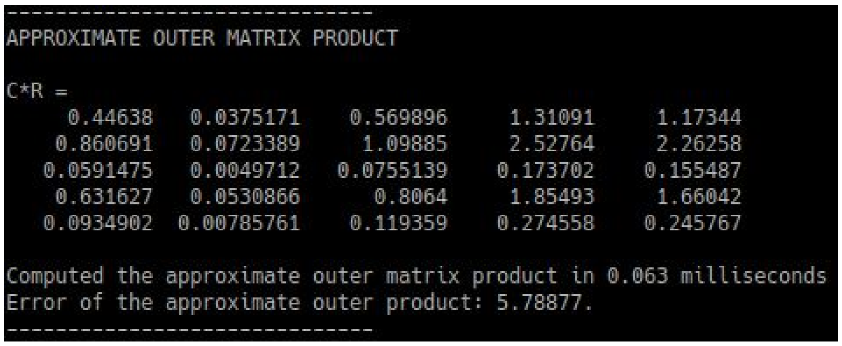


```
max:RandomMatrices Max$ ./randMatrix
Row of experiments (type: 0) or singular experiment (type: 1): 1
------------------------------
Rowcount of matrix A: 2
Row resp. columncount of matrix A resp. B: 2
Columncount of matrix B: 2
Factor of subsampling: 0.5
Using 1 rows/cols in the probabilistic product instead of 2.
------------------------------
A =
    0.468277    0.329655
    0.506117    0.309985

        B =
    0.909808    0.141257
    0.100369    0.894934

------------------------------
INNER MATRIX PRODUCT

A*B =
    0.459129    0.361166
    0.491582    0.348908

Computed the exact inner matrix product in 0.003 milliseconds
------------------------------
OUTER MATRIX PRODUCT

A*B =
    0.459129    0.361166
    0.491582    0.348908

Computed the exact product in 0.007 milliseconds
Error of the exact outer matrix product: 0. Surprise ;-)
------------------------------
APPROXIMATE OUTER MATRIX PRODUCT

C*R =
    0.426042   0.0661472
    0.460469   0.0714924

Computed the approximate outer matrix product in 0.091 milliseconds
Error of the approximate outer product: 0.407503.
------------------------------
max:RandomMatrices Max$ 
```

A sample output for n=2



```
Row of experiments (type: 0) or singular experiment (type: 1)? 1
------------------------------
Rowcount of matrix A: 5
Row resp. columncount of matrix A resp. B: 5
Columncount of matrix B: 5
Factor of subsampling: 0.3
Using 2 rows/cols in the probabilistic product instead of 5.
Use uniform sampling (1) or custom sampling (2): 2
------------------------------
A =
    0.486222    0.130224    0.239227    0.562806    0.711227
    0.937512    0.244772    0.163304    0.200941    0.910843
   0.0644267    0.952458    0.735682    0.448753    0.228742
    0.688002    0.170377    0.867889    0.474983    0.717869
    0.101835    0.955784    0.333888    0.815221     0.98573

        B =
    0.285905   0.0240296    0.365016    0.839633    0.751586
    0.540045    0.325855     0.88181    0.779272    0.888661
    0.593037    0.716784    0.133432     0.75634    0.917725
   0.0442753    0.820767    0.870183    0.779957     0.26952
   0.0989248    0.467959    0.439897    0.966813    0.942942

------------------------------
INNER MATRIX PRODUCT

A*B =
    0.446487     1.02035     1.12684     1.81725     1.52304
    0.596074    0.810504     1.15537     2.13876     1.98504
     1.01157      1.3146     1.45269      1.9239     1.90663
    0.895449     1.41992     1.24629     2.43137     2.26991
    0.876897     1.68361     2.06755     2.67171     2.38153

Computed the exact inner matrix product in 0.008 milliseconds
------------------------------
OUTER MATRIX PRODUCT

A*B =
    0.446487     1.02035     1.12684     1.81725     1.52304
    0.596074    0.810504     1.15537     2.13876     1.98504
     1.01157      1.3146     1.45269      1.9239     1.90663
    0.895449     1.41992     1.24629     2.43137     2.26991
    0.876897     1.68361     2.06755     2.67171     2.38153

Computed the exact product in 0.005 milliseconds
Error of the exact outer matrix product: 0. Surprise ;-)
------------------------------
```



```
------------------------------
APPROXIMATE OUTER MATRIX PRODUCT

C*R =
     0.44638   0.0375171    0.569896     1.31091     1.17344
    0.860691   0.0723389     1.09885     2.52764     2.26258
   0.0591475   0.0049712   0.0755139    0.173702    0.155487
    0.631627   0.0530866      0.8064     1.85493     1.66042
   0.0934902  0.00785761    0.119359    0.274558    0.245767

Computed the approximate outer matrix product in 0.063 milliseconds
Error of the approximate outer product: 5.78877.
------------------------------
```

Sample with n=5

Here are some inferences from the report:

1. $(CR)_{ij} = \sum_{t=1}^{c} \frac{A_{ij_t} B_{j_t j}}{p_{j_t}}$, because the (i,$j_t$) entry of CR is really a sum of c terms and as we recall from previously, for the tth term we had chosen the jth column of A and the jth row of B.
2. Finding expectation,

   $$E[(CR)_{ij}] = \frac{1}{c} * c * E[\frac{A_{ij_t} B_{j_t j}}{p_{j_t}}] = \sum_{k=1}^{n} (\frac{A_{ik} B_{kj}}{p_k}) * p_k = AB$$

   So $E[(CR)]_{ij} = (AB)_{ij}$.
3. Variance,

   $$var((CR)_{ij}) = \frac{1}{c} \sum_k \frac{A_{ik}^2 B_{kj}^2}{p_k} - \frac{1}{c}(AB)_{ij}^2 .$$

   Note that if c increases, variance decreases, as expected.

To find how close CR is to AB, we use Frobenius norm. We want to bound $|AB-CR|_F$.
Taking R as S^TB

$$E[|AB - CR|_F^2] = E[|AB - (AS)(S^t B)|^2] = \sum var(CR)_{ij}$$

Using the discussed values of p_i allows us to bound

$$E[|AB - CR|_F^2] \le \frac{1}{c} |A|_F^2 |B|_F^2 ;$$

we have a bound on the expected error. We can use Markov's inequality to say that since the expectation is small, the expectation of the random variable is not going to be much larger than its expectation with constant probability. We can improve the bound using Chernoff style inequalities. From Jensen's inequality,

$$E[|AB - CR|_F] = \sqrt{E(|AB - CR|_F^2)}$$

and this value can be plugged in to

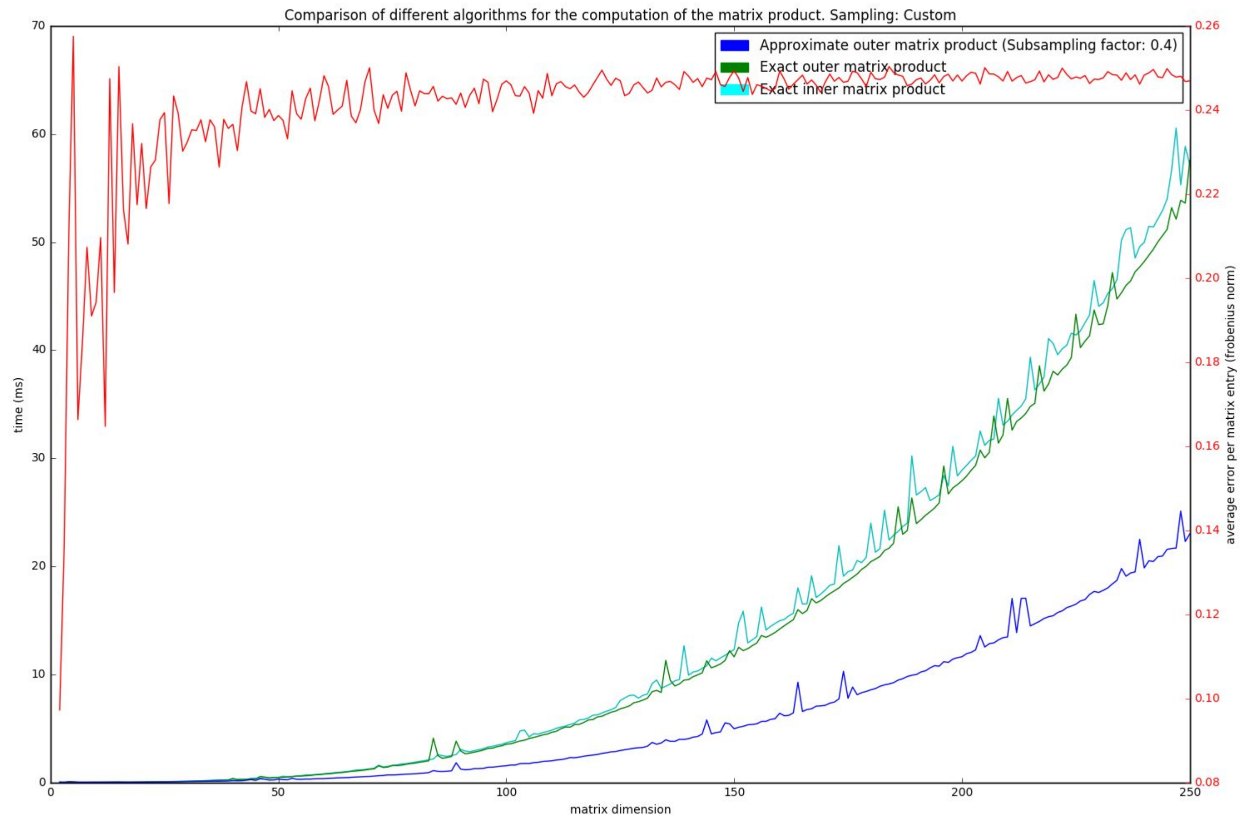
$$E[|AB - CR|_F^2] \leq \frac{1}{c}|A|_F^2|B|_F^2 .$$

Running time:

Using sampling matrix: **O(mn + np) + O(mcp)**. Doing the sampling takes **O(mn + np)** because we have to do a linear scan for calculating the probabilities, then we have to choose c samples which takes about nc time and then the multiplication of smaller matrices takes mcp time.
This is much less than the order O(mnp) for naive matrix multiplication.

On a final note, here we used sampling to obtain matrix S, but it could be obtained by sketching as well.

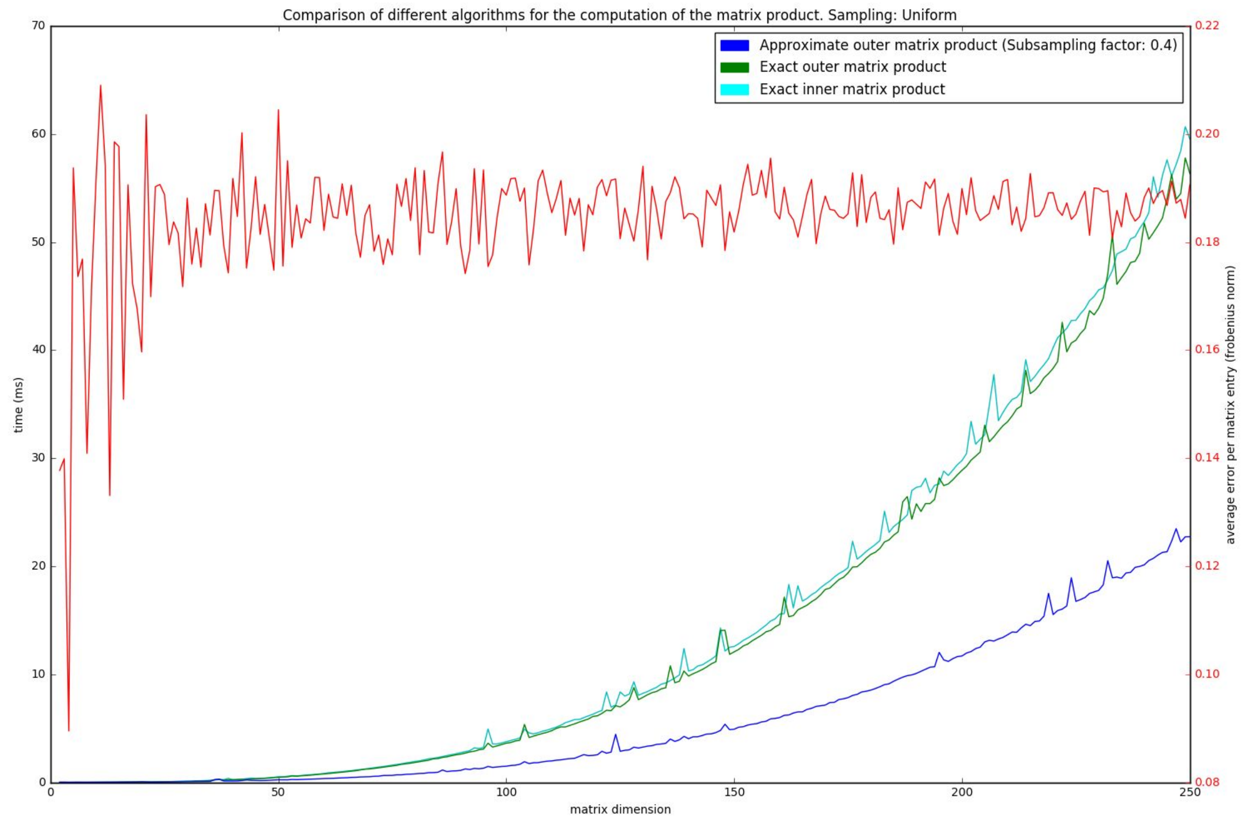
SUMMARY AND CONCLUSIONS



The above graph shows the time taken to generate the product of two matrices by 3 matrix multiplication algorithms.

The two naive algorithms ,i.e the inner and outer matrix multiplication algorithms represented by the blue and green curves show a steep increase in time taken as the size of the input matrices increase. We can safely conclude that the time taken by the randomised matrix multiplication algorithm is considerably lesser as compared to the naive algorithms. A clearer distinction can be made between the two classes of algorithms as the size of our input matrix increases.

The sampling method used here is a *custom sampling* method, where each rank one matrix is assigned a weight based on its value and these weights directly determine the probability vector. This sampling method yields slightly better results as opposed to the uniform sampling method where the probability of every rank one matrix is the same hence uniform. This method might have a higher variance and can perform worse than custom sampling in worst case scenarios.



Above graph is what we get when the sampling method is *uniform*.

OTHER MATHEMATICAL APPLICATIONS OF RANDOMISED MATRIX MULTIPLICATION

There are many applications where an approximate solution can replace the exact answer. The point of importance is the tradeoff between error value and runtime. Following are a few such examples, with higher payoff:

1. Low-Rank Approximation
2. Tensor-Train Decomposition
3. Approximating PCA(principal component analysis) using random projections
4. Large Markov Chains
5. Various machine learning and neural network algorithms employ the use of matrix multiplication, and most of the time the matrices are quite large. Thus this algorithm would prove helpful, especially in terms of time, which is key in ML algorithms.

BIBLIOGRAPHY

Papers:

[1] R. Kannan and S. Vempala(2017) - Randomized algorithms in numerical linear algebra, Acta Numerica(2017), pp. 95–135; doi:10.1017/S0962492917000058
Link: https://www.cc.gatech.edu/~vempala/papers/acta_survey.pdf

[2] N. Halko, P. G. Martinsson, and J. A. Tropp(2011) - Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions, SIAM Review, 2011, Vol. 53, No. 2 : pp. 217-288; doi:10.1137/090771806
Link: https://epubs.siam.org/doi/pdf/10.1137/090771806

[3] Michael W. Mahoney (2016)- Lecture Notes on Randomized Linear Algebra
Link: https://arxiv.org/pdf/1608.04481.pdf

[4] P. Drineas and M. Mahoney - Randomized Numerical Linear Algebra (RandNLA): Past, Present, and Future
Link: https://simons.berkeley.edu/sites/default/files/docs/518/drineasslides.pdf

Video lectures:

[5] G. Strang - 13. Randomized Matrix Multiplication
Link: https://www.youtube.com/watch?v=z0ykhV15wLw

[6] A. Dasgupta - Lecture 21: "Randomized Numerical Linear Algebra:a)Matrix multiplication + QB decomposition"
Link: https://www.youtube.com/watch?v=281sDWz-Qm0

Textbooks:

[7] Gilbert Strang - Linear Algebra and Learning from Data

[8] Gilbert Strang - Linear Algebra and its Applications