Problem 1.
The purpose of this question is to familiarize yourself with computing time- complexity. You are not required to turn in your answer to this question. A naive way to think about the purpose of O(·) notation is to see how the run time of an algorithm depends on the problem parameters and not worry too much about the exact constants. For example, if you take the inner product between two vectors in d-dimensions, there are d multiplications and (d - 1) additions required. If a single addition or multiplication operation costs 5 units of time, then the overall time complexity of computing inner product is (d×5) + ((d-1) ×5) units. Instead of calculating this explicitly, people write it as O(d) as the overall complexity is linear in d which is what we care about (that is, we don't care if it is 10 d or 1000000 d. But we care if it is O(log d) or O(d) or O(d2)). The complexity of matrix operations (exact multiplication, exact inversion and exact sin- gular value decomposition) are listed in this Wikipedia link. For this question, we will assume we are using the standard algorithms for the above tasks. So the complexity(with appropri- ately defined matrices) of matrix multiplication is O(n3) or O(nmp) and that of inversion is O(n3). Based on this, calculate the complexity of computing the OLS (denoted as β in the notes) and Sketched-OLS (denoted as βs in the notes) based on their closed-form expressions.

Solution
For a normal OLS solution, we have the expression for β as:

$$\beta = (X^T X)^{-1} X^T y \tag{1}$$

Breaking every component individually, we have,

- $(X^T X) = \mathcal{O}(n * p^2)$ where $X \in \mathbb{R}^{n\times p}$
- $(X^T X)^{-1} = \mathcal{O}(p^3)$
- $X^T y = \mathcal{O}(n * p)$
- $(X^T X)^{-1} X^T y = \mathcal{O}(p * p)$

Therefore the computation should take $\mathcal{O}(np^2 + p^3 + np + p^2) \sim \mathcal{O}(np^2 + p^3)$

For the sketched OLS solution, we have the expression for β_s as:

$$\beta_s = ((\phi X)^T (\phi X))^{-1} (\phi X)^T (\phi y) \text{ where } \phi = S^T H D \tag{2}$$

Based on similar breakdown as before, the time complexity comes down to $\mathcal{O}(rp^2) \sim \mathcal{O}(p^4)$

Problem 2.
Let A be an n × n square matrix. Show that the following statements are equivalent:

(a) The columns of A are orthonormal.

(b) $AA^T = A^T A =$ I, where I is the identity matrix.

(c) The rows of A are orthonormal vectors.

Solution
Letting $A^T A = I$, by definition of an orthogonal matrix, we know that the matrix $A^T A$ is a matrix comprised of dot products of columns. To be clear, in general, if B is an $m \times n$ and A is an $n \times p$ matrix, then the jk^{th} element in the BA matrix is equal to the dot product $b_j a_k$ where b_j is the j^{th} row vector of matrix B and a_k is the k^{th} column vector of matrix A.
Because A is actually an $n \times n$ matrix, and if we let $B = A^T$, then it should be clear that $A^T A = BA$ where the jk^{th} element in BA is equal to the dot product $b_j a_k$, where b_j is just the transpose of a_k. Therefor, $A^T A = BA$ is comprised of dot products of column vectors of matrix A.
More visually, this can be represented by the following:

$$A^TA = I_n \Rightarrow \begin{pmatrix} | & | & ... & | \\ a_1 & a_2 & ... & a_n \\ | & | & ... & | \end{pmatrix}^T \begin{pmatrix} | & | & ... & | \\ a_1 & a_2 & ... & a_n \\ | & | & ... & | \end{pmatrix} = \begin{pmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ 0 & 0 & ... & 1 \end{pmatrix},$$

where $a_i \in \mathbb{R}^{n\times 1}$ $\forall i = 1, ..., n$. We can then see that the resulting matrix is comprised of elements of dot products of columns of matrix A:

$$\Rightarrow \begin{pmatrix} a_1^Ta_1 & a_1^Ta_2 & ... & a_1^Ta_n \\ a_2^Ta_1 & a_2^Ta_2 & ... & a_2^Ta_n \\ : & ... & ... & a_2^Ta_n \\ a_n^Ta_1 & ... & ... & a_n^Ta_n \end{pmatrix} = \begin{pmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & 1 \end{pmatrix}$$

Here, $a_i^Ta_i = 1$ $\forall i = 1, ..., n$. The length of the column vectors of matrix A = 1.
$a_i^Ta_j = 0$ $\forall i = 1, ..., n$; $j = 1, ..., n$ $i \neq j$. In other words, **the column vectors of matrix A are pairwise orthogonal**.
We can also show that for a square matrix A where $A^TA = I$ that

$$A^TA = I \Rightarrow (A^TA)^{-1} \Rightarrow A^{-1}A^{T-1} = I$$

$$AA^{-1}A^{T-1} = A \Rightarrow AA^{-1}A^{T-1}A^T = AA^T$$

$$\Rightarrow AA^T = I$$

As before, from here it is easy to show that AA^T implies that matrix A is comprised of row vectors that are orthogonal:

$$AA^T = I_n \Rightarrow \begin{pmatrix} | & | & ... & | \\ a_1 & a_2 & ... & a_n \\ | & | & ... & | \end{pmatrix} \begin{pmatrix} | & | & ... & | \\ a_1 & a_2 & ... & a_n \\ | & | & ... & | \end{pmatrix}^T = \begin{pmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ 0 & 0 & ... & 1 \end{pmatrix},$$

$$\Rightarrow \begin{pmatrix} a_1a_1^T & a_1a_2^T & ... & a_1a_n^T \\ a_2a_1^T & a_2a_2^T & ... & a_2a_n^T \\ : & ... & ... & a_2a_n^T \\ a_na_1^T & ... & ... & a_na_n^T \end{pmatrix} = \begin{pmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & 1 \end{pmatrix}$$

Here, $a_ia_i^T = 1$ $\forall i = 1, ..., n$. The length of the row vectors of matrix A = 1.
$a_ia_j^T = 0$ $\forall i = 1, ..., n$; $j = 1, ..., n$ $i \neq j$. The off-diagonal elements of AA^T are dot products of *rows* of matrix A and are by definition 0 since $AA^T = I$. Therefor the row vectors of matrix A are orthogonal as well.
To summarize, for $A^TA = I$, it can be shown that the matrix A is comprised of orthonormal column vectors. It can also be shown that $A^TA = I$ implies that $AA^T = I$, in which case, the matrix A can be shown to be comprised of orthonormal row vectors.

Problem 3.
Show how do you go from Equation (2) to Equation (3) in the variance calculation in Section (2.1) of the randomized matrix multiplication notes.

Solution
Fixing i, j, and for $l = 1, ..., r$, we have:

$$X_l = \frac{1}{rp_k}A_{i,k}B_{k,j} \Rightarrow \mathbb{E}(X_l)_{i,j} = \sum_{k=1}^{n} \frac{1}{rp_k}A_{i,k}B_{k,j} = \frac{1}{r}(AB)_{i,j}$$

$$\Rightarrow \mathbb{E}(X_l^2)_{i,j} = \sum_{k=1}^{n} \frac{1}{r^2 p_k} A_{i,k}^2 B_{k,j}^2$$

Since $M := \sum_{l=1}^{r} X_l$, we have $\mathbb{E}(M_{i,j}) = \sum_{l=1}^{r} \mathbb{E}(X_l)_{i,j} = (AB)_{i,j}$.
Since M_{ij} is the sum of r independent random variables, we also have that $Var(M_{i,j}) = \sum_{l=1}^{r} Var(X_l)_{i,j}$, where

$$Var(X_l) = \mathbb{E}(X_l^2) - \mathbb{E}(X_l)^2 = \sum_{k=1}^{n} \frac{1}{r^2 p_k} A_{i,k}^2 B_{k,j}^2 - \frac{1}{r^2}(AB)_{i,j}^2$$

With this information, it is easy to show that

$$\mathbb{E}(||M - AB||_F^2) = \mathbb{E}(\sum_{i,j}(M_{i,j} - A_{i.}B_{.j})^2) = \sum_{i,j} Var(M_{i,j} = \sum_{i,j}\sum_{l=1}^{r} Var(X_l)$$

Noting we have $Var(X_l)$ above (plugging in):

$$\sum_{i,j}\sum_{k=1} r\frac{r^2}{p_k} A_{i,k}^2 B_{k,j}^2 - \sum_{i,j} r\frac{1}{r^2}(AB)_{i,j}^2 \Rightarrow \frac{1}{r}\sum_{i,j}\sum_{k=1}\frac{1}{p_k} A_{i,k}^2 B_{k,j}^2 - \frac{1}{r}\sum_{i,j}(AB)_{i,j}^2$$

$$= \frac{1}{r}\sum_{k=1}\frac{1}{p_k}||A_{:,k}||_F^2 ||B_{k,:}||_F^2 - \frac{1}{r}||AB||_F^2$$

Problem 4. Randomized matrix multiplication:

(a) Implement the algorithm presented in class for randomized matrix multiplication (Algorithm 2 from the class notes on Randomized Linear Algebra)

(b) Apply the algorithm to the provided matrices selecting a number of columns r = 20,50,100,200. The matrices for this problem can be found in the attached files = "STA243 homework 1 matrix A.csv" and "STA243 homework 1 matrix B.csv".

(c) Calculate the relative approximation error $||M - AB||_F/(||A||_F||B||_F)$ for each of the estimates found in (b). Provide your results in a table.

(d) Visualize the estimates from (b) in Python (you can use Matplotlib).

Solution

(a) For random matrix multiplication, following were the parameters fed to the function:

- Matrix A & B = The matrices to be multiplied with each other such that $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times p}$ resulting in a matrix $M \in \mathbb{R}^{m\times p}$
- Sampling type = Non uniform or Uniform
- $C \in \mathbb{R}$ = Constant used to determine appropriate number of rows/columns to be used in the multiplication.
- $r \in \mathbb{R}$ = Number of rows/columns to be sampled.

Based on the given parameters, the function decides how to determine the probability for the sampling of the columns/rows of the matrices to be multiplied. In case of uniform sampling, probability $p_k = \frac{1}{n}$, while in case of non-uniform sampling, the p_k is decided using the following formula

$$p_k = \frac{||A||_{k,:}||B||_{:,k}}{\sum_{i=1}^{n}||A||_{i,:}||B||_{:,i}} \tag{3}$$

After determining the appropriate p_k, we sample 'r' indices based on these probabilities and add all the 'r' columnar matrices X_l such that $\sum_{l=1}^{r} X_l = M \sim AB_{m\times p}$, such that AB is the true multiplication of the matrices.

(b) On finalizing the function, we create a list of all the given r's and loop through them using a for loop and store the multiplication obtained from the function defined in (a).

(c) For calculating the error, we create a function which takes the following parameters:

- Actual Multiplication $AB \in \mathbb{R}^{m\times p}$ = The accurate value obtained by multiplying large matrices.
- Approximate Multiplication $M \in \mathbb{R}^{m\times p}$ = The approximate value obtained by the function created in (a).
- Matrix A & B = The actual matrices such that $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times p}$.

The error function returns the value determined by the following equation

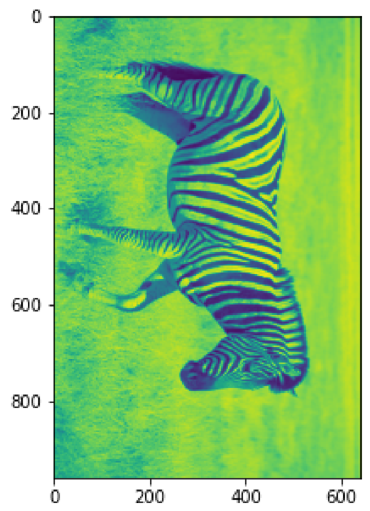
$$e_T = \frac{||M - AB||_F}{||A||_F \times ||B||_F} \tag{4}$$

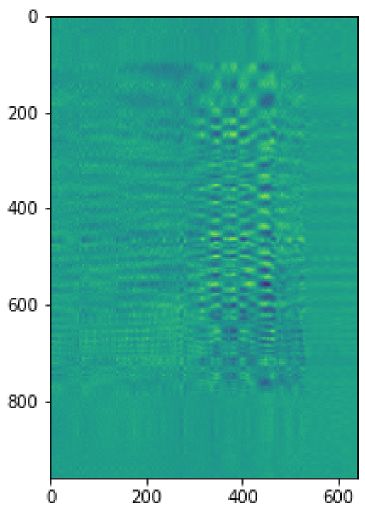
On applying the above function on all the r's mentioned in part (b), we get the following table as a result showing the errors with different r's

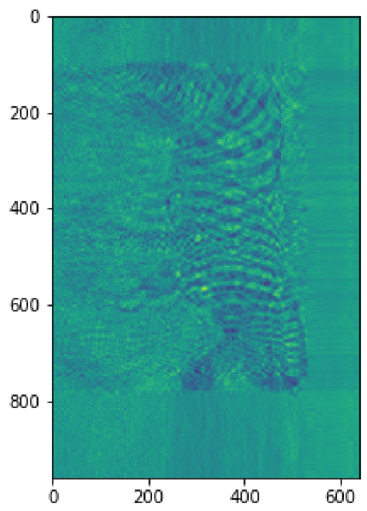
R (No of columns selected)	Error in the approximation
20.000	0.207
50.000	0.116
100.000	0.097
200.000	0.075

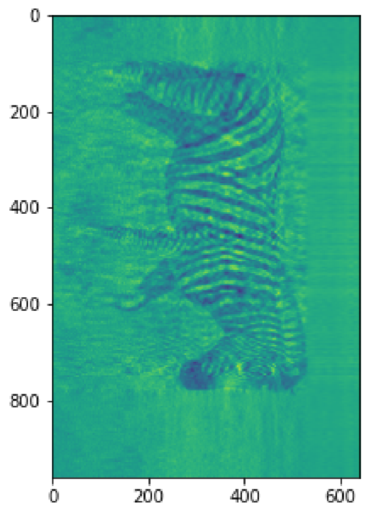
Table 1: R vs Error obtained

(d) The above estimates for every r can be seen on the next page


(a) *Actual*


(b) $r = 20$


(c) $r = 50$


(d) $r = 100$

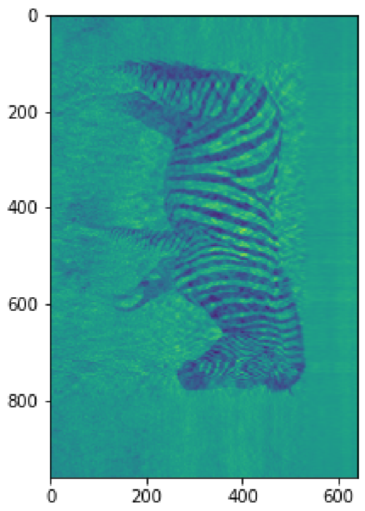

(e) $r = 200$

Figure 1: Plots for all images

Problem 5. Power method: Let X be a 10 ×10 matrix such that,

$$X = \lambda(vv^T) + E$$

where E is a random matrix with each entry being an i.i.d. standard Gaussian variable. Note that X is a rank-1 matrix perturbed with a random noise matrix (E). Your goal in this problem is to estimate the eigenvector v. The file power sim.py consists of a test routine that fixes the true eigenvector, initial vector used for power method and the noise matrix. The end goal is produce a plot of λ versus how well the estimated eigenvector (using the power method) is correlated (measured via inner-product) with the true eigenvector. For this you have to implement your own power method (function power iteration). Complete the code and run the test routine to produce the plot.

Solution
For the power method, following parameters are passed to the function:

- Matrix $A \in \mathbb{R}^{d \times d}$ = The matrix for which the eigenvector is to be found.
- Vector $v_0 \in \mathbb{R}^d$ = Initial unit vector decided randomly
- Error $\epsilon \sim 0$ = The error in the approximation of the vector in previous and current iterations
- Maxiter = Defaults to a 100, is the maximum number of iterations that the code will run for.

Using the above parameters we try to approximate the eigen vector by first multiplying the initial vector v_0 by the matrix A into a new vector v. Then, we normalized the vector v to a unit distance and we check if the l_2 norm of vector v_0 and v is less than the defined error. If it is less, we return the obtained v, otherwise we update $v_0 = v$.
Using the driver function given to us in *power_sim.py* file, we get the plot shown below.

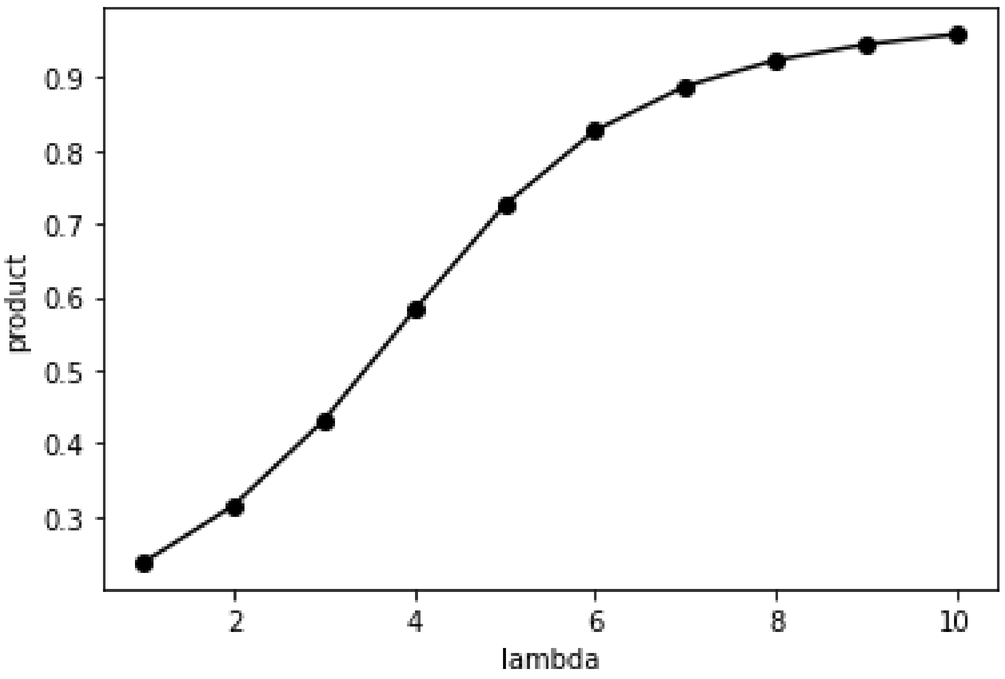


Figure 2: Performance of how well the algorithm approximates the eigen vectors based on λ

With increase in lambda, we can see that the product of the intial vector and the vector obtained from our defined function settles down to approximately 0.9.

Problem 6. Sketching for Least-squares:

(a) Implement the Sketched-OLS algorithm presented in class (Algorithm 1 from the class notes on Randomized Algorithms for Least Squares).

(b) Generate a 1048576 ×20 design matrix X and a 1048576 ×1 response y with elements drawn iid from a Uniform(0,1) distribution.

(c) Compare the calculation time for the full least squares problem and the sketched OLS. For the matrix $\phi = S^T HD$, first calculate $X* = \phi X$ and $y* = \phi y$. Once finished, use the system.time() function in R to time the calculation of $(X *^T X*)^{-1} X *^T y*$ and compare to the calculation time of $(X^T X)^{-1} X^T y$. Repeat these steps for ϵ = .1,.05,.01,.001 and present your results in a table.

Solution

(a) To perform the sketched OLS, we mainly used 2 functions for the implementation. The functions were namely *sketchify* and *sketched_ols*. For the *sketched_ols* function, following parameters were passed:

- $X \in \mathbb{R}^{n\times d}$: = The design matrix consisting of all the covariate/predictor data.
- $y \in \mathbb{R}^n$ = Response variable for which we try to find the estimates.
- $\epsilon \in \mathbb{R}$ = Error that helps in deciding the size of the sub-sample matrix S.

We decide the size of the sub-sample matrix S using the ϵ passed. Let the number of columns be r. It is given by

$$r = \frac{d \log n}{\epsilon} \tag{5}$$

Using this, we create a sub sample matrix $S \in \mathbb{R}^{n\times r}$ where in every column is a canonical column which is multiplied by the constant $\sqrt{\frac{n}{r}}$. For increasing the speed, it was stored as a sparse matrix using *scipy.sparse.csc_matrix*. We then create an array D, which is a diagonal matrix consisting of -1 or 1. The *sketchify* function takes the input of the matrix DX and applies the Fast Walsh Hadamard transform (FWHT) to the given matrix. In the final step, using the Sub-sample matrix created we get the sketched estimates by applying:

$$b_s = (S^T HDX)^! S^T HDy \text{ where } A^! = (A^T)^{-1}A^T \tag{6}$$

to the computed matrices to get our estimates.

(b) The values are generated using the seed *920211348* using numpy module. The function used to generate the uniform values is *np.random.uniform* which generates data from a distribution that is $\sim Uniform(0, 1)$ in the given shape.

(c) The time of execution using the naive method of execution comes out to be 0.1947s.

Following is the summary table of implementing for the given epsilons

ϵ	Computation Times	% Improvement
0.100	0.003	5959.999%
0.050	0.004	4743.544%
0.010	0.012	1520.503%
0.001	0.076	156.690%

Table 2: Table summarises the computation times and the improvement in comparison to the naive implementation

Pledge:

Please sign below (print full name) after checking (✓) the following. If you can not honestly check each of these responses, please email me at kbala@ucdavis.edu to explain your situation.

- We pledge that we are honest students with academic integrity and we have not cheated on this homework.
- These answers are our own work.
- We did not give any other students assistance on this homework.
- We understand that to submit work that is not our own and pretend that it is our is a violation of the UC Davis code of conduct and will be reported to Student Judicial Affairs.

- We understand that suspected misconduct on this homework will be reported to the Office of Student Support and Judicial Affairs and, if established, will result in disciplinary sanctions up through Dismissal from the University and a grade penalty up to a grade of "F" for the course.

Team Member 1: Collin Kennedy Team Member 2: Jay Bendre