1. Write a Spark program to find the least squares fit on the following 10 data points. The variable \( y \) is the response variable, and \( X_1, X_2 \) are the independent variables.

\[
\begin{array}{cccc}
X_1 & X_2 & y \\
[1,] & -0.5529181 & -0.5465480 & 0.009519836 \\
[2,] & -0.5428579 & -1.5623879 & 0.982464609 \\
[3,] & -1.3038629 & 0.5715549 & 0.499441144 \\
[4,] & 0.6564096 & 1.1806877 & 0.495705999 \\
[5,] & -1.2061171 & 1.3430651 & 0.153477135 \\
[6,] & 0.2938439 & -1.7966043 & 0.914381381 \\
[7,] & -0.2578953 & 0.2596407 & 0.815623895 \\
[8,] & 0.9659582 & 2.3697927 & 0.320880634 \\
[9,] & -0.4038109 & 0.9846071 & 0.488856619 \\
[10,] & 0.6029003 & -0.3202214 & 0.380347546
\end{array}
\]

More precisely, find \( w_1, w_2 \), such that \( \sum_{i=1}^{10} (w_1 X_{1i} + w_2 X_{2i} - y_i)^2 \) is minimized. Report \( w_1, w_2 \), and the Root Mean Square Error and submit code in Spark. You will need to write a gradient descent subroutine which was demonstrated in class on Lecture 16.

Lastly, consider how you would change your algorithm if Spark supported AllReduce; write pseudocode and analyze the resulting algorithm in terms of all-to-all, one-to-all, and all-to-one communication patterns.

2. Write a Spark program to compute the Singular Value Decomposition of the following 10 \times 3 matrix:

\[
\begin{array}{cccc}
\text{-0.5529181} & \text{-0.5465480} & \text{0.009519836} \\
\text{-0.5428579} & \text{-1.5623879} & \text{0.982464609} \\
\text{-1.3038629} & \text{0.5715549} & \text{0.499441144} \\
\text{0.6564096} & \text{1.1806877} & \text{0.495705999} \\
\text{-1.2061171} & \text{1.3430651} & \text{0.153477135} \\
\text{0.2938439} & \text{-1.7966043} & \text{0.914381381} \\
\text{-0.2578953} & \text{0.2596407} & \text{0.815623895} \\
\text{0.9659582} & \text{2.3697927} & \text{0.320880634} \\
\text{-0.4038109} & \text{0.9846071} & \text{0.488856619} \\
\text{0.6029003} & \text{-0.3202214} & \text{0.380347546}
\end{array}
\]

Assume the matrix is tall and skinny, so the rows should be split up and inserted into an RDD. Each row can fit in memory on a single machine. Report all singular vectors and values and submit your Spark program.

3. Given a matrix \( M \) in row format as an RDD["ARRAY"\[DOUBLE\]] and a local vector \( x \) given as an ARRAY[DOUBLE], give Spark code to compute the matrix vector multiply \( Mx \).

Solution:
In class we saw how to compute highly similar pairs of $m$-dimensional vectors $x, y$ via sampling in the mappers, where the similarity was defined by cosine similarity: 
$$
x^T y \
------------------------ 
\|x||y|$$
Show how to modify the sampling scheme to work with overlap similarity, defined as
$$\text{overlap}(x, y) = \frac{x^T y}{\min(|x|^2, |y|^2)}$$

(a) Prove shuffle size is still independent of $m$, the dimension of $x$ and $y$.
(b) Assuming combiners are used with $B$ mapper machines, analyze the shuffle size.

Solution:
(a) We modify the DIMSUM mapper as follows:

Algorithm 1 DIMSUMOverlapMapper($r_i$)

1: for all pairs $(a_{ij}, a_{ik})$ in $r_i$ do
2: With probability $\min(1, \gamma \min(\|c_i\|_2^2, \|c_j\|_2^2))$
3: emit $((j, k) \rightarrow a_{ij}a_{ik})$
4: end for

The shuffle size of this scheme is $O(nL\gamma/H^2)$ where $H$ is the smallest nonzero element of $A$ in magnitude. To show this we start with the expected contribution from each pair of columns.

$$
= \sum_{i=1}^n \sum_{j=i+1}^n \sum_{k=1}^n P(\text{DIMSUMOverlapEmit}(c_i, c_j))
= \sum_{i=1}^n \sum_{j=i+1}^n \|c_i\|_2^2 \|c_j\|_2^2 P(\text{DIMSUMOverlapEmit}(c_i, c_j))
\leq \sum_{i=1}^n \sum_{j=i+1}^n \gamma \frac{\|c_i\|_2^2 \|c_j\|_2^2}{\|c_i\|_2^2 \|c_j\|_2^2}
= \sum_{i=1}^n \sum_{j=i+1}^n \gamma \frac{\|c_i\|_2^2 \|c_j\|_2^2}{\|c_i\|_2^2 \|c_j\|_2^2}
\leq \gamma \sum_{i=1}^n \frac{1}{\|c_i\|_2^2 \|c_j\|_2^2} \sum_{j=1}^n \#(c_i, c_j)
\leq \gamma \sum_{i=1}^n \frac{\#(c_i)H^2}{\|c_i\|_2^2} \sum_{j=1}^n \#(c_i, c_j)
= \gamma \frac{L\#(c_i)H^2}{\|c_i\|_2^2} \sum_{j=1}^n \#(c_i, c_j)
= \gamma L\frac{n}{H^2}
$$

The fourth equality comes from assuming WLOG $\|c_i\|_2^2 \leq \|c_j\|_2^2$. 

x_bc = sc.broadcast(x)
output = M.map(lambda row: np.dot(row, x_bc.value)).collect()
(b) In the naive case with combiners, each of the $B$ machines will emit at most $n^2$ pairs — one for each element in $A^T A$. However, without combiners we know that DIMSUM will have a shuffle size of at most $nL\gamma / H^2$. Thus the shuffle size is at most $O(\min(Bn^2, nL\gamma / H^2))$. 