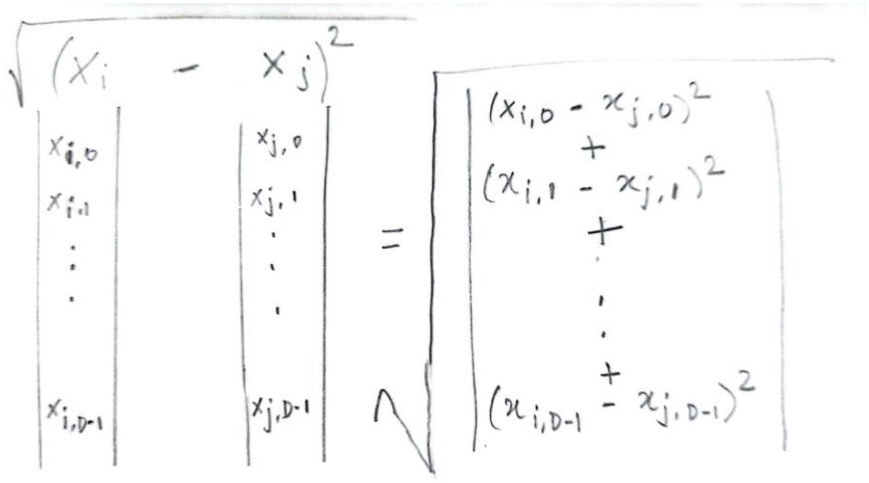
Zeehan Rahman

Solutions:

1. Distance Matrix:
   a. With Loop: I have used a two-level nested loop, iterating through all pairs of vectors, and calculating their Euclidean distance. In each iteration, element wise subtraction is performed between the two vectors of all possible pairs, difference is squared and the result is summed up. Lastly, taking out the square root of the sum, completes the Euclidean distance formula:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

The diagram below depicts the functionality of the algorithm

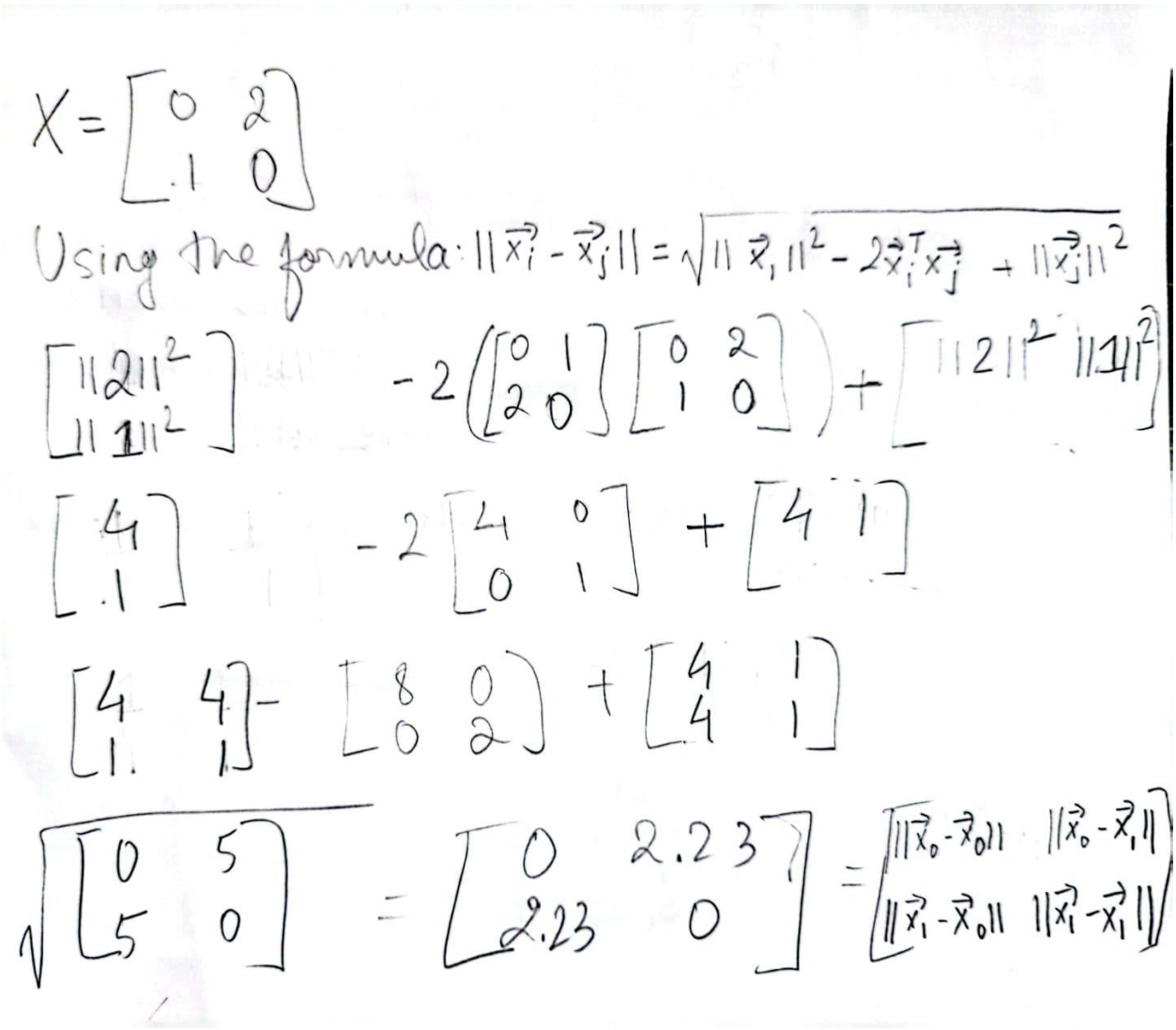


The Euclidean distance of each pair of vectors is stored in a matrix of dimensions N X N

   b. Without Loop: Vectorization replaces the inefficient looping in python. Using the formula given in the question:

$$\|\vec{x_i} - \vec{x_j}\| = \sqrt{\|\vec{x_i}\|^2 - 2\vec{x_i}^T\vec{x_j} + \|\vec{x_j}\|^2}$$

For the first term: $\|\vec{x_i}\|^2$, the algorithm performs the square operation on the whole matrix X (as $\mathbf{x} \cdot \mathbf{x} = \|\mathbf{x}\|^2$), and sum of the element of the row vectors in matrix X
For the second term: $-2\vec{x_i}^T\vec{x_j}$, basic matrix operation (dot, transpose) is performed
For the third term: $\|\vec{x_j}\|^2$, the same steps are involved as for the first term
Lastly, the three terms are added and the absolute value is taken before the square root is taken for the matrix.
The resulting matrix is the distance matrix.
The diagram below uses an example to depict the working of the algorithm:



Both algorithms have been tested in matrices of different dimensions:

```
matrix dimensions:  100 10
matrix dimensions:  200 20
matrix dimensions:  300 30
matrix dimensions:  400 40
matrix dimensions:  500 50
matrix dimensions:  600 60
matrix dimensions:  700 70
matrix dimensions:  800 80
matrix dimensions:  900 90
matrix dimensions:  1000 100
matrix dimensions:  1100 110
matrix dimensions:  1200 120
matrix dimensions:  1300 130
matrix dimensions:  1400 140
```

2. **Correlation Matrix:**

a. **With Loop:** The algorithm calculates the sample mean of the matrix X. Consequently, matrix X is subtracted by the sample mean using broadcasting (lets call this new matrix X_m). In a two-level nested loop, element wise multiplication is performed in all possible pairs of column vectors of X_m and the product is added. The result of each pair of row vectors are stored in a matrix S, where S is the sample covariance of the original matrix X.

   Afterwards, vector st is formed by extracting the diagonal of the matrix S. After performing square root operation in vector st, st holds all the standard deviation of the matrix X.

   To achieve the desired matrix:

$$R = \begin{pmatrix} \frac{s_{0,0}}{\sigma_0\sigma_0} & \frac{s_{0,1}}{\sigma_0\sigma_1} & \cdots & \frac{s_{0,D-1}}{\sigma_0\sigma_{D-1}} \\ \frac{s_{1,0}}{\sigma_1\sigma_0} & \frac{s_{1,1}}{\sigma_1\sigma_1} & \cdots & \frac{s_{1,D-1}}{\sigma_1\sigma_{D-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{s_{D-1,0}}{\sigma_{D-1}\sigma_0} & \frac{s_{D-1,1}}{\sigma_{D-1}\sigma_1} & \cdots & \frac{s_{D-1,D-1}}{\sigma_{D-1}\sigma_{D-1}} \end{pmatrix}$$

   Matrix S is divided by the vector st, using broadcasting. The result is stored in a matrix S2.
   Lastly, transpose of S2 is divided by the vector st and the result is the correlation matrix. Correlation matrix is symmetric so transpose of correlation matrix is equal to the correlation matrix

   Using the vector

b. **Without Loop:**
   The second approach is similar to the first one(with loops), with the only difference being, that everything is vectorized in the second approach. Using numpy's basic matrix operation, loops have been avoided thus significantly speeding up the calculations.

   Instead of using nested loops to perform element wise multiplication and adding the products, between all the possible pairs of column vectors of matrix X, dot product operation has been used. The dot product of transpose of X and X does the exact same task but using less lines and time compared to the two-level loops.

   Apart from this difference, other parts of this algorithm are similar to the one with loop.
   The diagram below depicts the last part(dividing matrix S with the denominators):

$$S = \begin{bmatrix} s_{0,0} & s_{0,1} & \cdots & s_{0,D-1} \\ s_{1,0} & s_{1,1} & \cdots & s_{1,D-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{D-1,0} & s_{D-1,1} & & s_{D-1,D-1} \end{bmatrix} \quad st = [\sigma_0, \sigma_1, \ldots, \sigma_{D-1}]$$

$$S2 = S/st = \begin{bmatrix} \frac{s_{0,0}}{\sigma_0} & \frac{s_{0,1}}{\sigma_0} & \cdots & \frac{s_{0,D-1}}{\sigma_0} \\ \frac{s_{1,0}}{\sigma_1} & \frac{s_{1,1}}{\sigma_1} & \cdots & \frac{s_{1,D-1}}{\sigma_1} \\ \vdots & & & \\ \frac{s_{D-1,0}}{\sigma_{D-1}} & \frac{s_{D-1,1}}{\sigma_{D-1}} & \cdots & \frac{s_{D-1,D-1}}{\sigma_{D-1}} \end{bmatrix}$$

$$Corr\_matrix = S2^T/st$$

$$\begin{bmatrix} \frac{s_{0,0}}{\sigma_0\sigma_0} & \frac{s_{1,0}}{\sigma_0\sigma_0} & \cdots & \frac{s_{D-1,0}}{\sigma_{D-1}\sigma_0} \\ \frac{s_{0,1}}{\sigma_0\sigma_1} & \frac{s_{1,1}}{\sigma_1\sigma_1} & \cdots & \frac{s_{D-1,1}}{\sigma_{D-1}\sigma_1} \\ \vdots & & & \\ \frac{s_{0,D-1}}{\sigma_0\sigma_{D-1}} & \frac{s_{1,D-1}}{\sigma_1\sigma_{D-1}} & & \frac{s_{D-1,D-1}}{\sigma_{D-1}\sigma_{D-1}} \end{bmatrix}$$

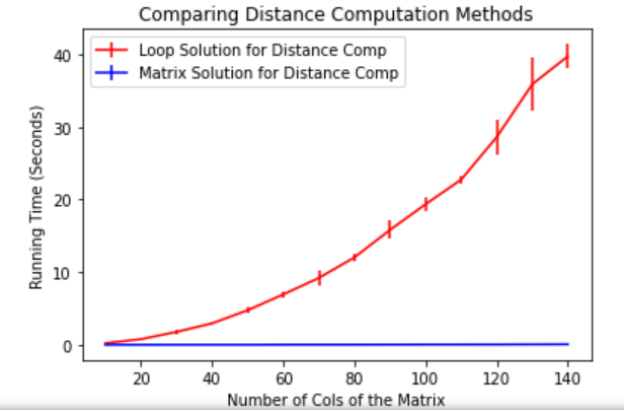
Correlation matrix is symmetrical.
Therefore:

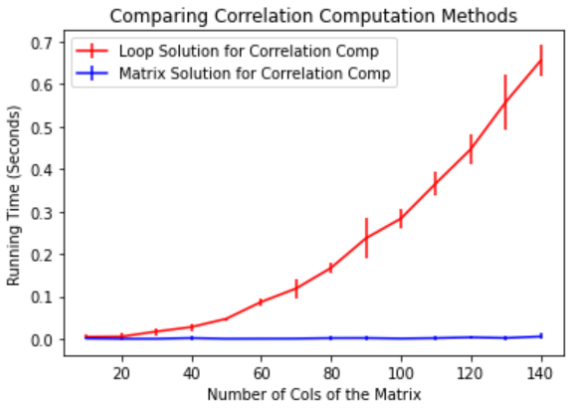
$$corr\_matrix = corr\_matrix^T$$

Both algorithms have been tested in matrices of different dimensions:

```
matrix dimensions:  100 10
matrix dimensions:  200 20
matrix dimensions:  300 30
matrix dimensions:  400 40
matrix dimensions:  500 50
matrix dimensions:  600 60
matrix dimensions:  700 70
matrix dimensions:  800 80
matrix dimensions:  900 90
matrix dimensions:  1000 100
matrix dimensions:  1100 110
matrix dimensions:  1200 120
matrix dimensions:  1300 130
matrix dimensions:  1400 140
```

Experiments:





As we can see from both the graphs the matrix solution is clearly the winner (the control variables being the values of the matrix, number of columns and rows that are held same for the two methods). Matrix solution finishes the computation in near 0 seconds, while the loop solution grows considerably as the number of column increases. (The rows also increase 10 times the number of columns).

For distance computation, the matrix solution takes near 0 seconds when the number of columns is 140 and the number of rows is 1400. For the same dimensions of the matrix with the same contents, the loop solution takes around 40 seconds.

For correlation computation, the matrix solution takes less than 0.1 seconds when the number of columns is 140 and the number of rows is 1400. For the same dimensions and the contents of the matrix, the loop solution takes around 0.7 seconds.

Similar result is also seen in datasets obtained from sklearn.

Distance Computation

Dataset	Loop Solution (Mean Time in sec)	Matrix Solution (Mean Time in sec)
Iris Dataset (150 X 4)	0.48	0.0
Breast Cancer (569 X 30)	6.22	0.0107

Digits Dataset (1797 X 64)	64.19	0.10

Correlation Computation

Dataset	Loop Solution (Mean Time in sec)	Matrix Solution (Mean Time in sec)
Iris Dataset (150 X 4)	0.002	0.0
Breast Cancer (569 X 30)	0.02	0.0004
Digits Dataset (1797 X 64)	0.1522	0.0

Reason for the Matrix solution being the clear winner:

Python is an interpreted language and thus has a poor performance when compared to a compiled language. Therefore, looping in python is also slow.

However, vectorization saves us from the slow looping and indexing in python. And it is replaced with the pre-compile C code. The matrix solution is executed at near C-speeds, providing explanations for the tables and charts above.

These insights are useful when training our model on large datasets. We should always go for the matrix solution instead of using looping and indexing in python.