Principal Component Analysis

Dr. Kalidas Y., IIT Tirupati

119) key phrase… “Principal Component Analysis”

- Principal = Important
- Component = Eigen vector

- Typically used to reduce data dimensionality
- E.g. 1,000,000 dimensional data point will just become 2 dimensional!

- This dimensionality reduction is most applicable when data is already into almost clean clusters in high dimensions

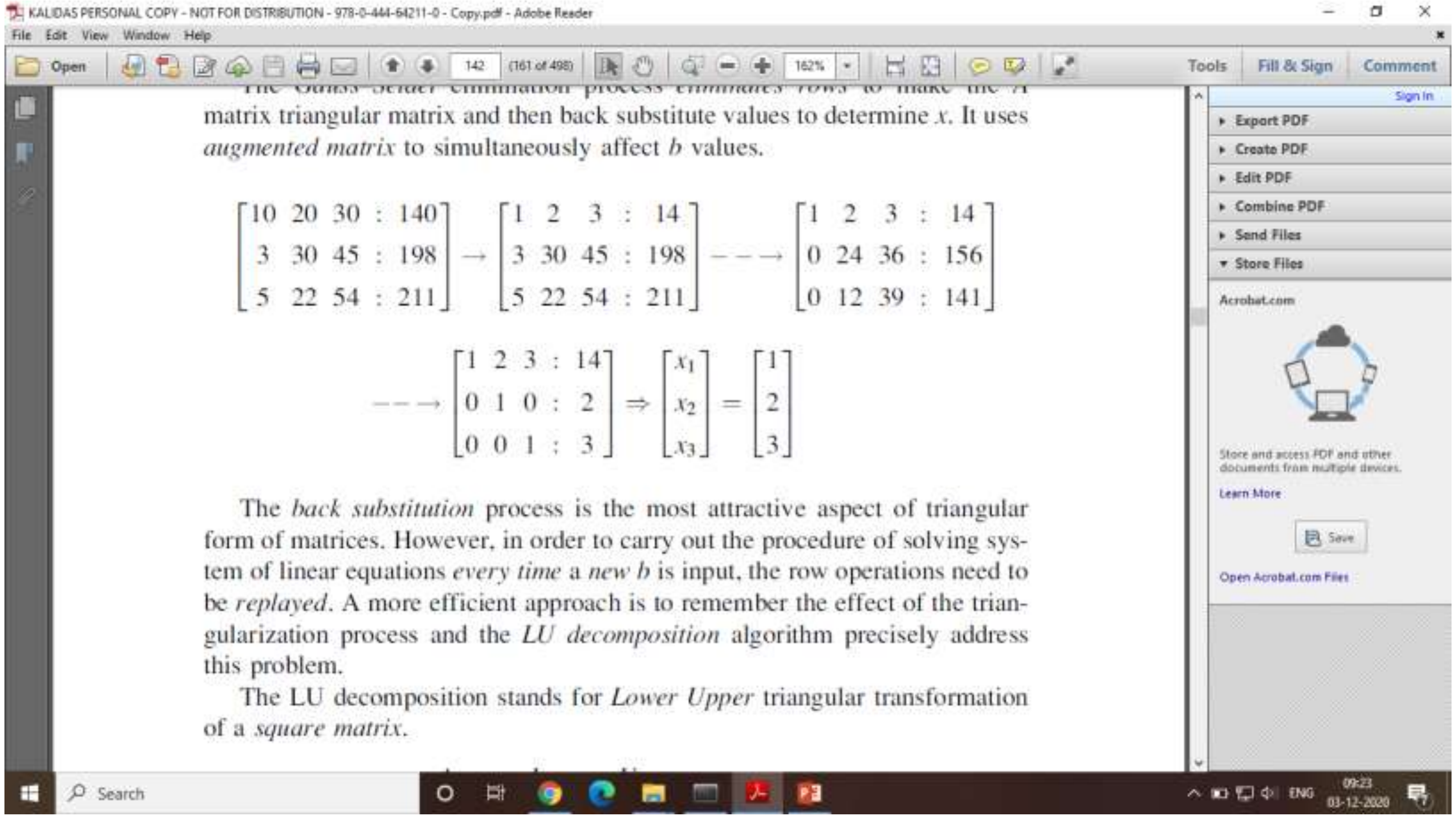
- The idea is, you can project data onto vectors along which data appears to be neatly separated out

- The dimension of maximum spread of maximum variance

- However, it is not so much useful when data is convoluted
  - E.g. it is not applicable to ~~*concentric circles type of data in 2D (components = 1)*~~
  - E.g. it is not applicable to ~~*moons type of data in 2D (components = 1)*~~
  - E.g. it is not applicable to ~~*concentric sine curves in 2D (components = 1)*~~

Solving a system of linear equations

120) key phrase… *“Gauss-Siedel Elimination Process”*



matrix triangular matrix and then back substitute values to determine x. It uses *augmented matrix* to simultaneously affect b values.

$$\left[\begin{array}{ccc:c} 10 & 20 & 30 & 140 \\ 3 & 30 & 45 & 198 \\ 5 & 22 & 54 & 211 \end{array}\right] \rightarrow \left[\begin{array}{ccc:c} 1 & 2 & 3 & 14 \\ 3 & 30 & 45 & 198 \\ 5 & 22 & 54 & 211 \end{array}\right] \dashrightarrow \left[\begin{array}{ccc:c} 1 & 2 & 3 & 14 \\ 0 & 24 & 36 & 156 \\ 0 & 12 & 39 & 141 \end{array}\right]$$

$$\dashrightarrow \left[\begin{array}{ccc:c} 1 & 2 & 3 & 14 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right] \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

The *back substitution* process is the most attractive aspect of triangular form of matrices. However, in order to carry out the procedure of solving system of linear equations *every time* a *new* b is input, the row operations need to be *replayed*. A more efficient approach is to remember the effect of the triangularization process and the *LU decomposition* algorithm precisely address this problem.

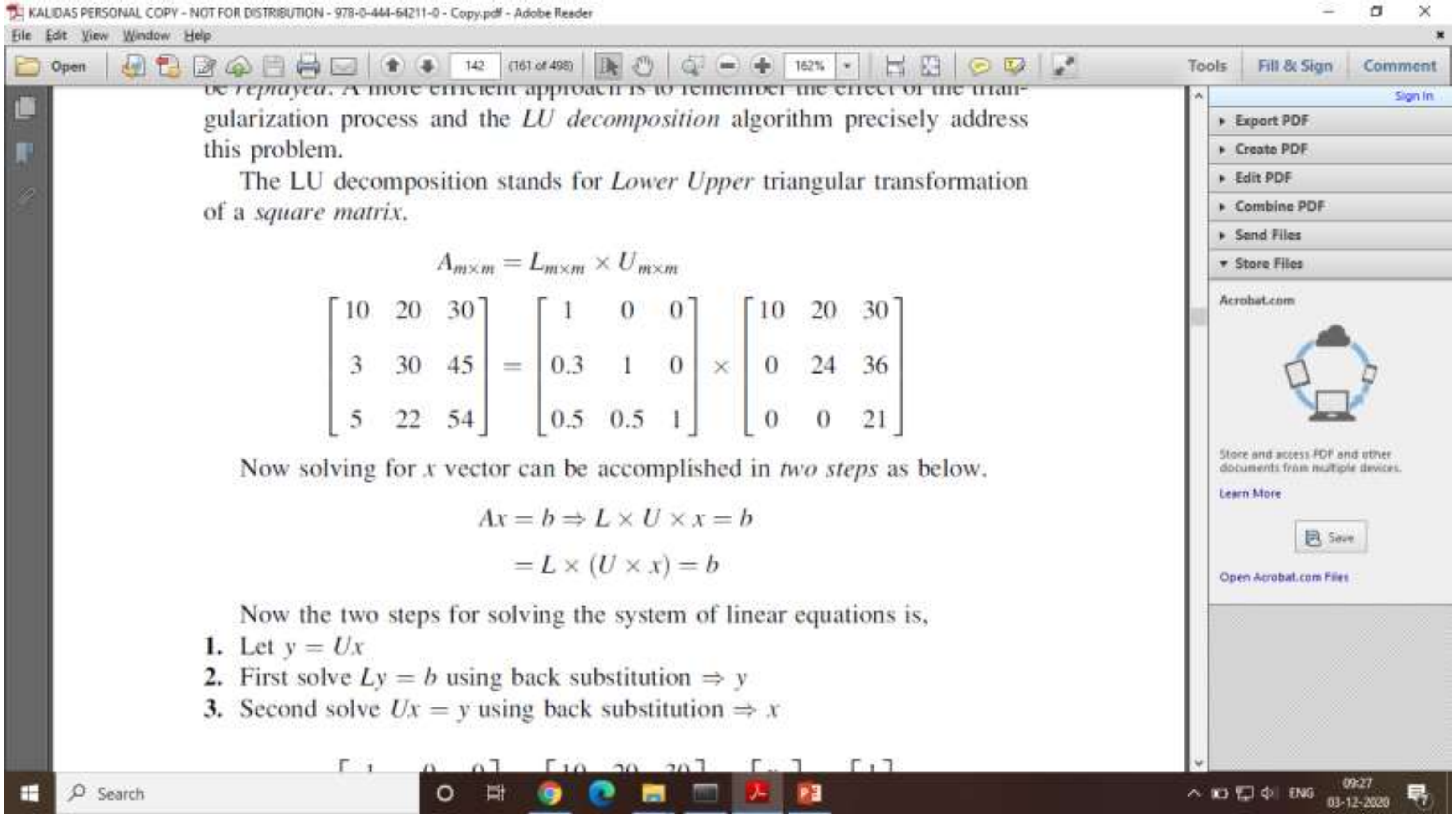
The LU decomposition stands for *Lower Upper* triangular transformation of a *square matrix*.

Ax = b

When b changes, the whole procedure has to repeat – This is not efficient idea!

Solving a system of linear equations

121) key phrase… *“LU Decomposition”*



be replayed. A more efficient approach is to remember the effect of the triangularization process and the *LU decomposition* algorithm precisely address this problem.

The LU decomposition stands for *Lower Upper* triangular transformation of a *square matrix*.

$$A_{m\times m} = L_{m\times m} \times U_{m\times m}$$

$$\begin{bmatrix} 10 & 20 & 30 \\ 3 & 30 & 45 \\ 5 & 22 & 54 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0.3 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{bmatrix} \times \begin{bmatrix} 10 & 20 & 30 \\ 0 & 24 & 36 \\ 0 & 0 & 21 \end{bmatrix}$$

Now solving for x vector can be accomplished in *two steps* as below.

$$Ax = b \Rightarrow L \times U \times x = b$$

$$= L \times (U \times x) = b$$

Now the two steps for solving the system of linear equations is,

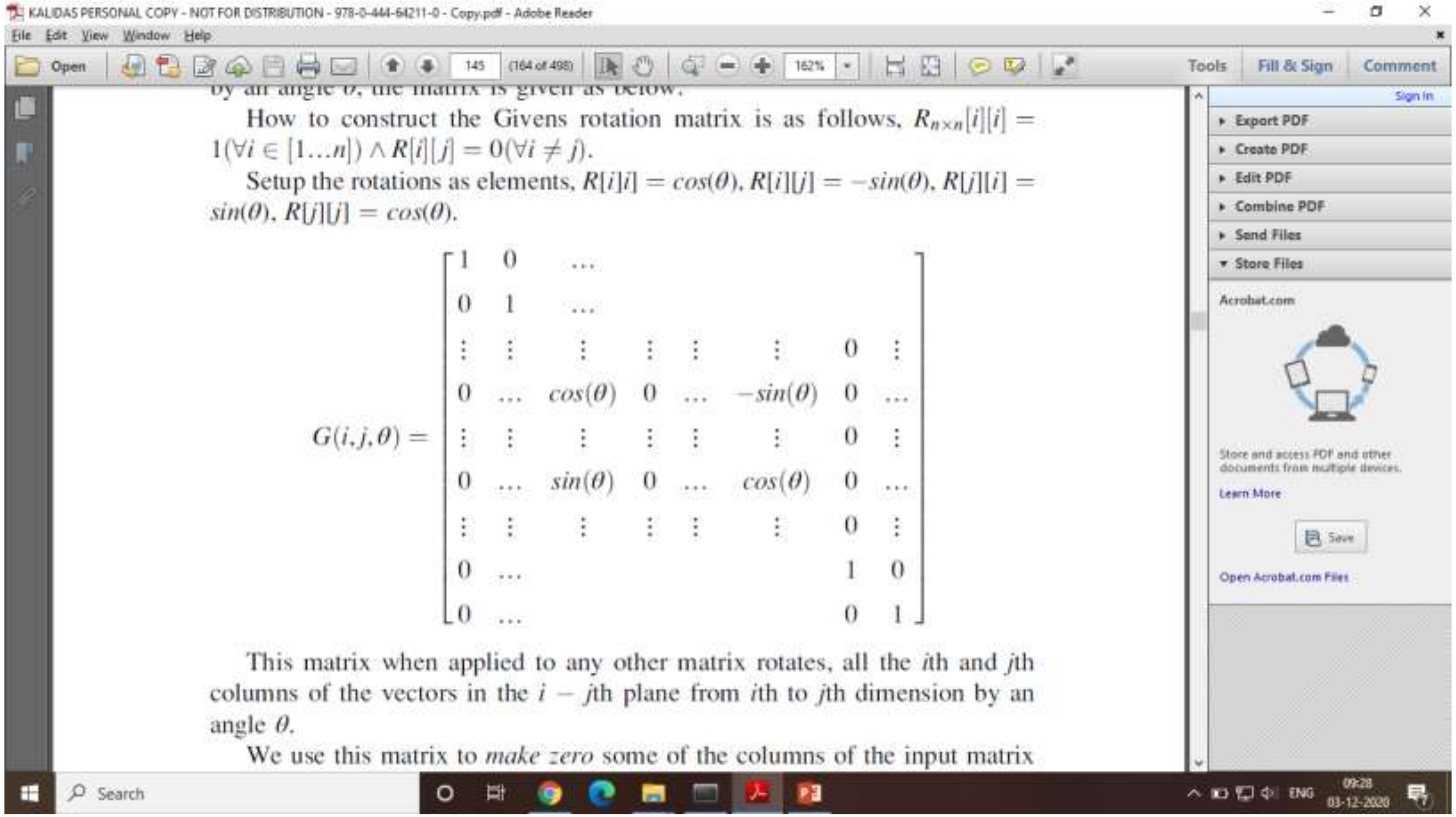
1. Let $y = Ux$
2. First solve $Ly = b$ using back substitution $\Rightarrow y$
3. Second solve $Ux = y$ using back substitution $\Rightarrow x$

Ax = b

When b changes, No Need to Repeat the Upper triangulation process!

Solving a system of linear equations

122) key phrase… *“Givens Rotation Matrix”*



by an angle θ, the matrix is given as below.

How to construct the Givens rotation matrix is as follows, $R_{n\times n}[i][i] = 1(\forall i \in [1\ldots n]) \wedge R[i][j] = 0(\forall i \neq j)$.

Setup the rotations as elements, $R[i][i] = cos(\theta)$, $R[i][j] = -sin(\theta)$, $R[j][i] = sin(\theta)$, $R[j][j] = cos(\theta)$.

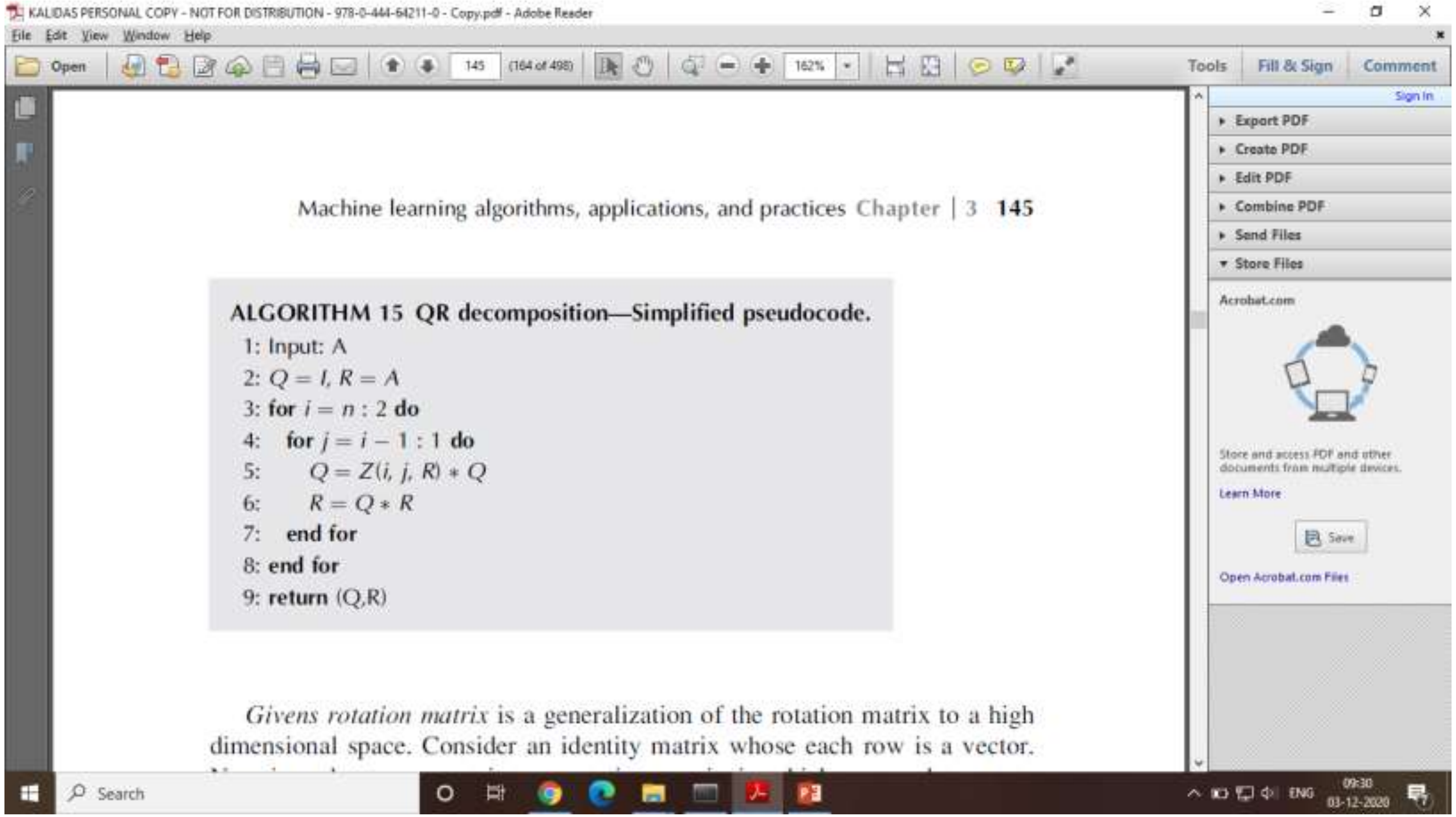
$$G(i,j,\theta) = \begin{bmatrix} 1 & 0 & \ldots & & & & & \\ 0 & 1 & \ldots & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 0 & \vdots \\ 0 & \ldots & cos(\theta) & 0 & \ldots & -sin(\theta) & 0 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 0 & \vdots \\ 0 & \ldots & sin(\theta) & 0 & \ldots & cos(\theta) & 0 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 0 & \vdots \\ 0 & \ldots & & & & & 1 & 0 \\ 0 & \ldots & & & & & 0 & 1 \end{bmatrix}$$

This matrix when applied to any other matrix rotates, all the ith and jth columns of the vectors in the $i-j$th plane from ith to jth dimension by an angle θ.

We use this matrix to *make zero* some of the columns of the input matrix

Solving a system of linear equations

123) key phrase… *"QR Decomposition"*



ALGORITHM 15 QR decomposition—Simplified pseudocode.

```
Input: A
Q = I, R = A
for i = n : 2 do
    for j = i − 1 : 1 do
        Q = Z(i, j, R) ∗ Q
        R = Q ∗ R
    end for
end for
return (Q,R)
```

Givens rotation matrix is a generalization of the rotation matrix to a high dimensional space. Consider an identity matrix whose each row is a vector.

Solving a system of linear equations

124) key phrase… *“Singular Value Decomposition - SVD”*

ALGORITHM 16 SVD Algorithm—Simplified pseudocode.

```
Input : A_{m×n} matrix.
Û_{m×m}, Z_{m×n} = QR(A) //QR factorization of A
⟹ A = Û × Z
Note that Z is an upper triangular matrix
Consider Z^T_{n×m}
V_{n×n}, D_{n×m} = QR(Z^T) //QR factorization of Z^T
⟹ Z^T = V × D
A = Û × (Z^T)^T = Û × (V × D)^T = U × D^T × V^T
Note that D^T is still a diagonal matrix
Now, D^T needs to be cast as Σ with diagonal element ordering
Let (∃P) : D^T = P × Σ
Let U = Û × P // to absorb the row permutations
Then, we have A = U × Σ × V^T as required by the SVD factorization
```

Some iterations to happen instead of 1 step

Solving a system of linear equations
125) key phrase… *"Principal Components"*

- Let $X_{N \times k}$ be the given data matrix (each data point is k dimensional)
- $X = U \, \Sigma \, V^T$
- Determine these matrices by a function call, svd() , $\mathrm{u}, \mathrm{s}, \mathrm{v}h = svd(X)$
- Now, $\mathrm{v}h$ is a matrix (*of k x k shape*) of eigen vectors
- These eigen vectors are used for projection of input to lower dimensional space
- The 0th eigen vectors corresponds to the one with highest eigen value, 1st element the next and so on in this vh-matrix
- Example of 2d transformation
  - Lets, say, we take vh[:,0] and vh[:,1]
  - Each xi is now projected into 2 dimensional space as,
  - $xi \vdash (xi \cdot vh[:,0], xi \cdot vh[:,1])$
- Example of 3d transformation
  - Lets, say, we take vh[:,0], vh[:,1], vh[:,2]
  - Each xi is now projected into 3 dimensional space as,
  - $xi \vdash (xi \cdot vh[:,0], xi \cdot vh[:,1], xi \cdot vh[:,2])$
- Likewise…

Caveats!

- Centroid of the data matters!

- If you change the centroid, the principal components would change
- Imagine, vectors to Delhi from Tirupati
- vs, vectors from within Delhi to Delhi… *the angle subtended at the origin matters*

- PCA
  - First determines centroid of the training set (call it PCA-centroid say)
  - And then goes on to detect eigen vectors

- So, when a new data point comes..
  - 1) It will be translated to PCA-centroid
  - 2) After that, it will be projected on to the eigen vectors

126) key phrase… “PCA Transformation”

- PCA(n_components=2)
  - Specify the number of components to use
- Train PCA on Training set – PCA.train(X_train)
  - Origin translation
    - PCA_center of X_train will be determined
    - All the points will be ‘centered’ about
    - X_train_shifted = X_train – PCA_center
  - Determine top 2 eigen vectors
- Data transformation
  - Usage
    - X_train_pca = PCA.transform(X_train)
    - X_test_pca = PCA.transform(X_test)
  - What happens when a new data point xi comes?
    - STEP 1: Centering: (*internally*) xi_t = xi – PCA_center
    - STEP 2: Dimensionality transformation: ((xi_t dot v1), (xi_t dot v2)) are the new dimensions

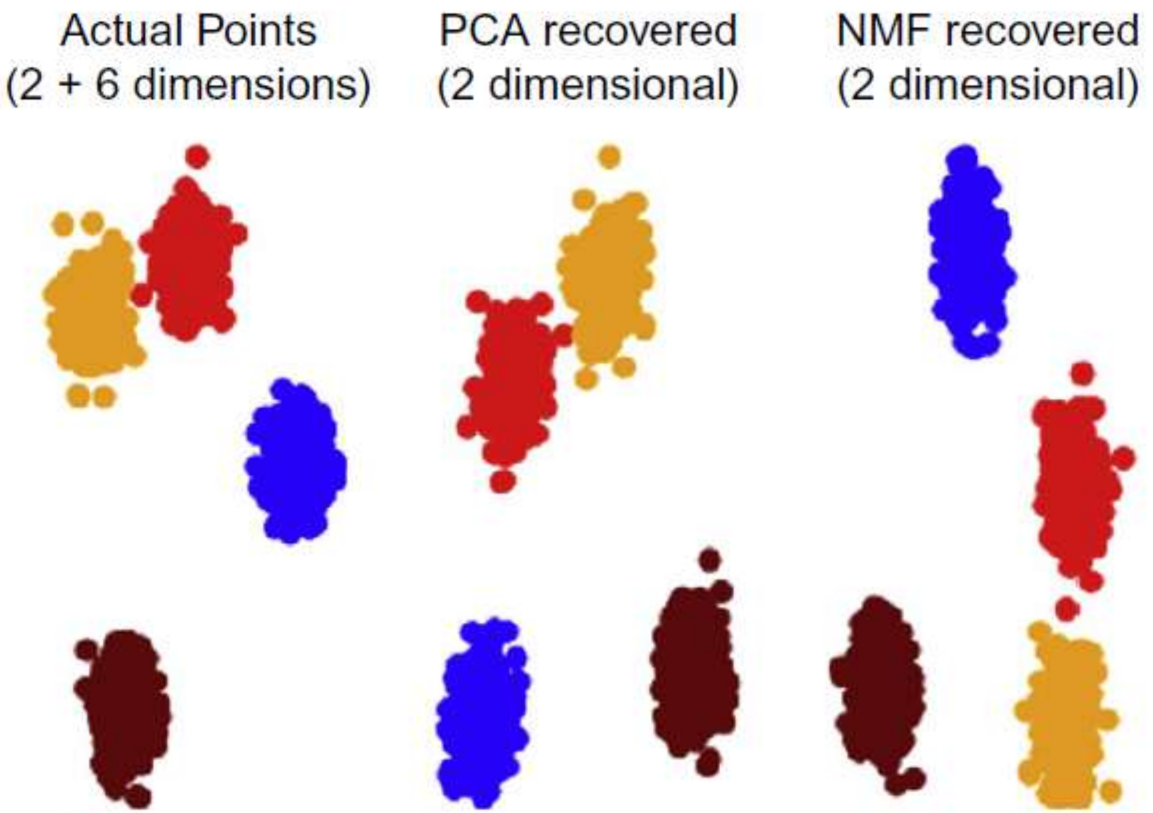


FIG. 23 Principle component analysis (PCA) and nonnegative matrix factorization (NMF)-based lower dimensional feature transformation of data points in 8 dimensions is depicted here. A set of 1000 data points are generated from four natural groups of points shown by four distinct colors (*left most figure*). The points are then transformed into 8 dimensional space by the transformation $(x_1,x_2)\vdash(R(),R(),x_1,x_2,R(),R(),R(),R())$ where $R()$ is a function that generates a Gaussian random number 1 + 0.5 * $N(0, 1)$. The input data is then transformed using PCA into a lower dimensional space by specifying number of components as 2, i.e., to consider only top

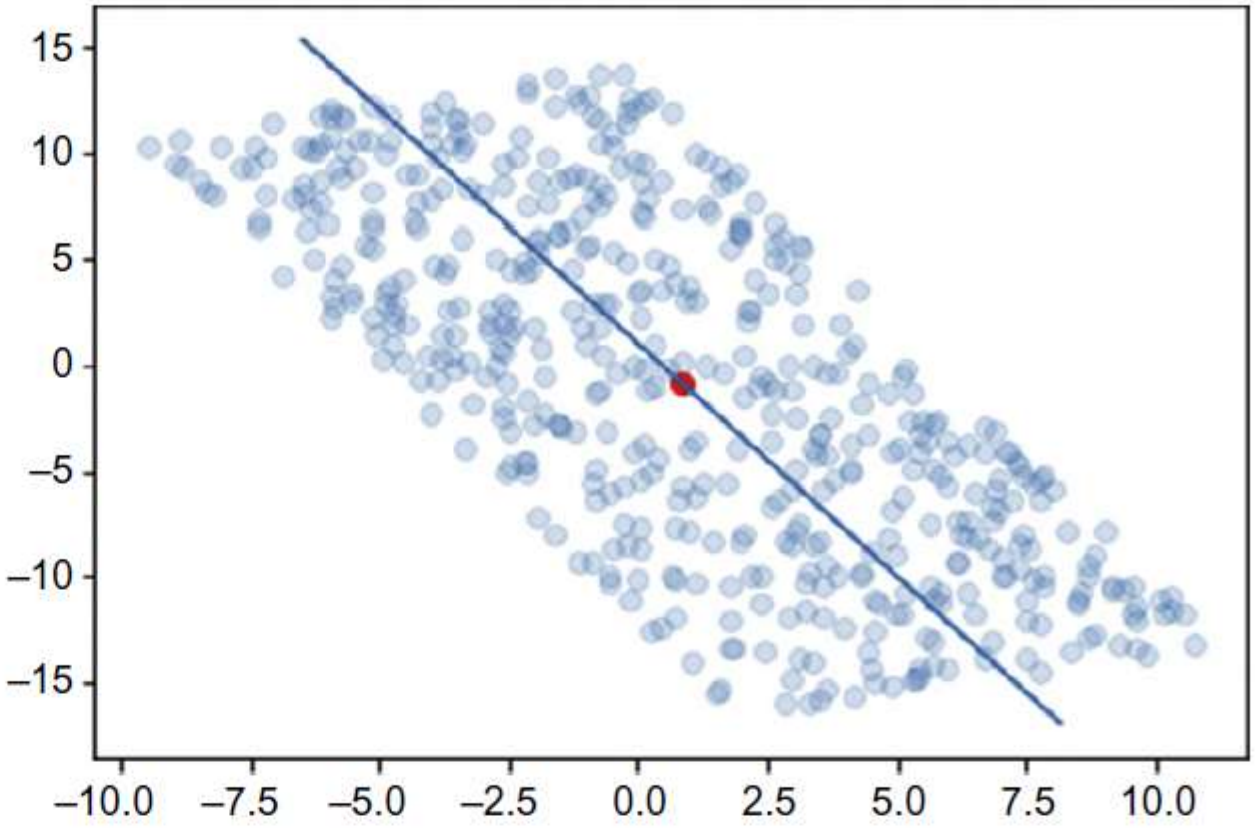


FIG. 24 Principle component analysis is used for determining major axis of a distribution of 1000 points along a parallelogram. The top eigenvector is plotted as a *line* in the 2 dimensional space. Center of the data is shown by a *red dot*.