[MPHY0030_MATLAB_COURSEWORK_18-19]

Student number: 14018713

2019-2-3

MODULE:MPHY0030

UCL

Introduction

Summary: This assignment uses data from finite element simulations of prostate gland motion as training data to build a PDM. Each prostate shape is represented as a set of *node points* that define the vertices of finite elements used.
All of the datasets used are stored in the data file data_shapes.mat, and this data file contains three variables include:

- ✧ nodes_training is a 200-by-1 cell. Each cell contains a 642-by-3 matrix that specify the node coordinates of a training shape.

- ✧ nodes_test is a 100-by-1 cell. Each cell is a 642-by-3 matrix containing node coordinates of a test shape.

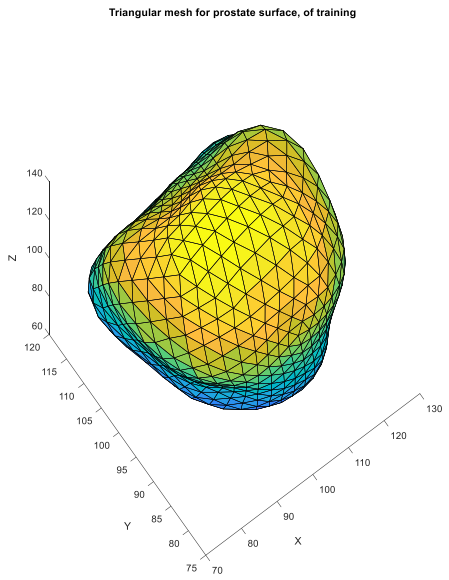
- ✧ tris is a 1280-by-3 matrix containing triangulated meshes for all of the above shapes (used only for plotting the surface).

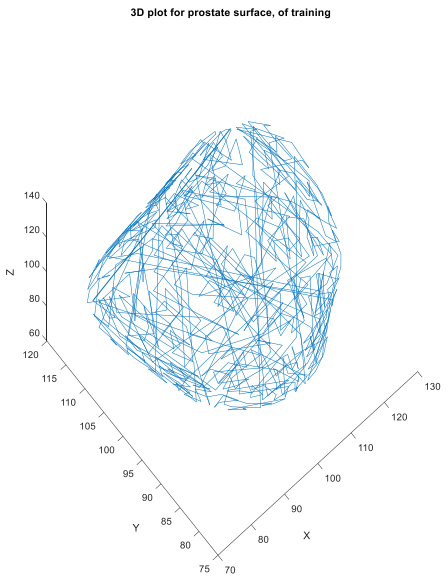
Methods

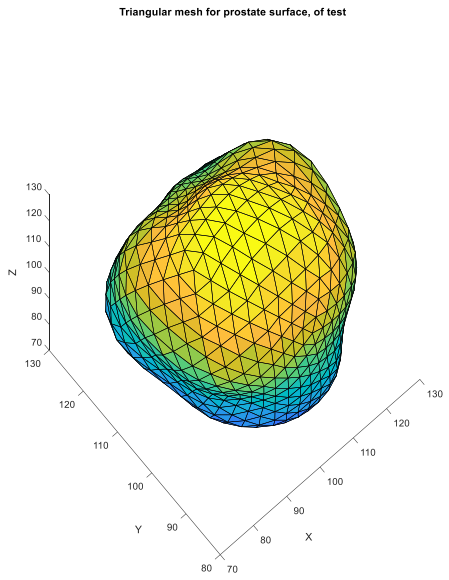
1. Notes: the results of visualisation may look slightly different due to different rotation angle of view, but this will not affect the shape shown.
2. Task 1
   Target: Visualisation of the surface of an individual prostate drawn from the training and test data.

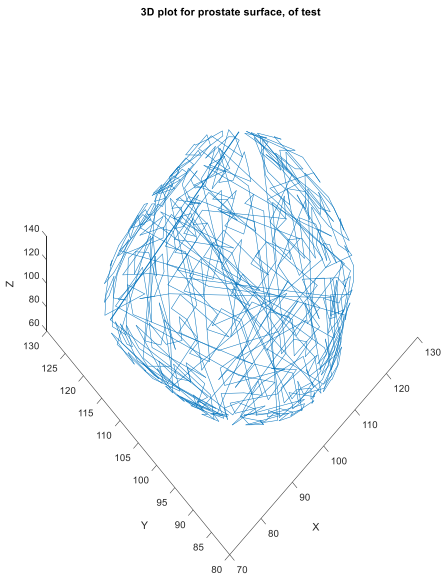
   Steps:
   (1) Initially, datasets of tris, nodes_training and nodes_test were acquired from the load of “data_shapes.mat” into the workspace.
   (2) Then, a PDMvis function was created to visualise the data.
   (3) The triangular mesh and 3D plot of both nodes_training and nodes_test data at (ntest)th cell or shape are visualised, where the (ntest) number have been selected at 1, 50 and 100.

   Results:
   (1) Visualisation of training and test data, at ntest= 1

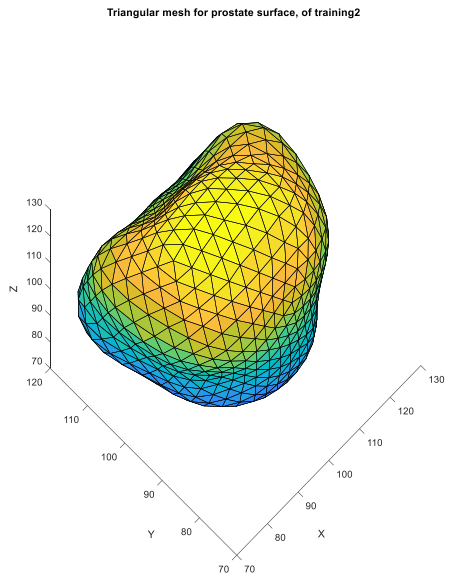


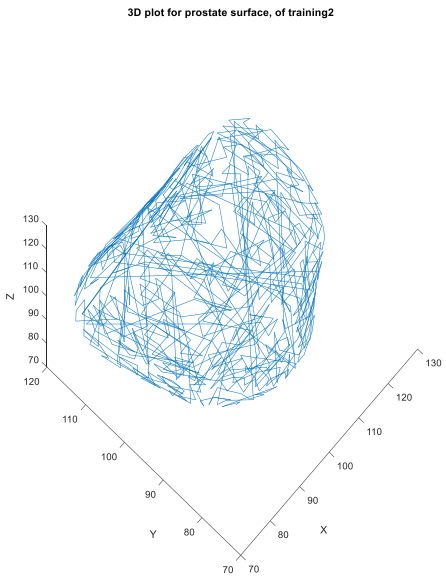


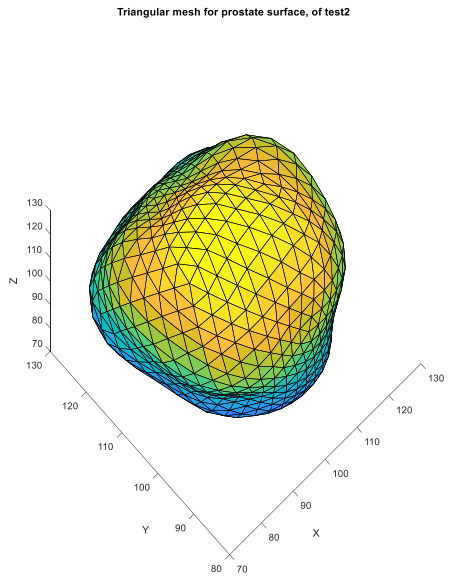


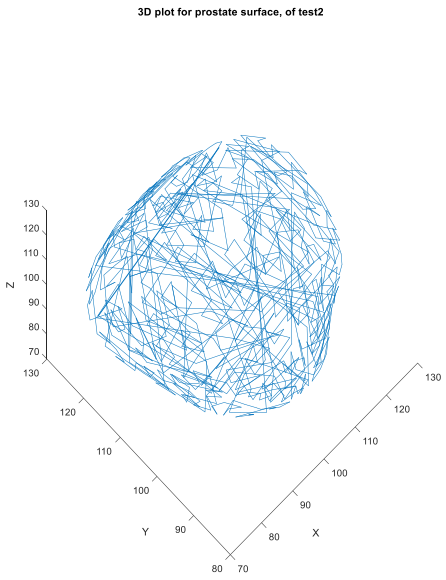


(2) Visualisation of training and test data, at ntest= 50

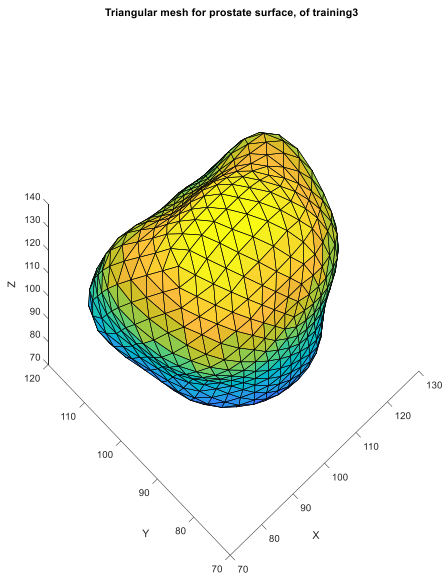


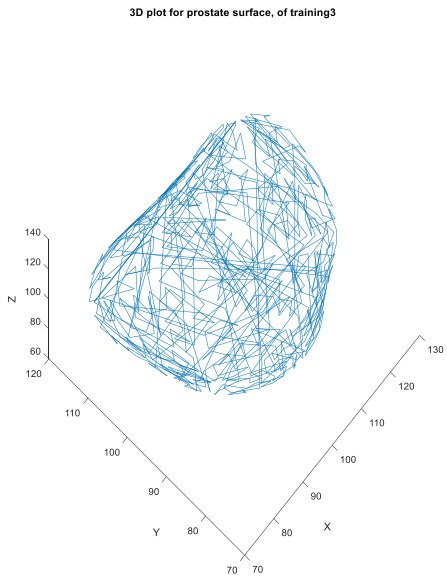


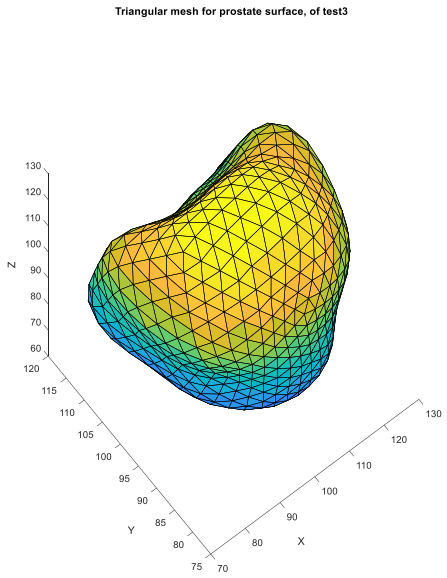


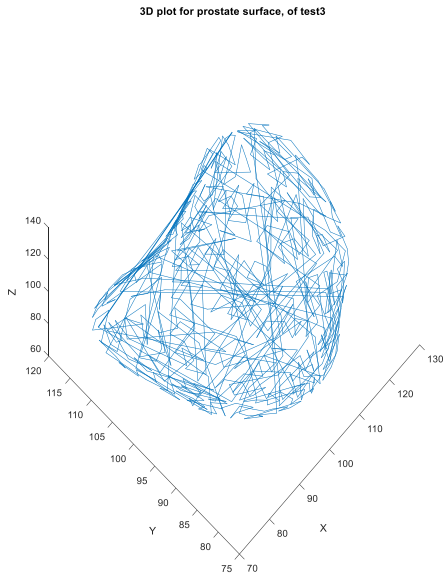


(3) Visualisation of training and test data, at ntest= 100









3. Task 2 and 3

   Target:

   Create PDMbuild_cov that performs Principal Component Analysis (PCA) given nodes_training by decomposing the covariance matrix C.

   Steps:

   (1) Initially, create a variable called “x_3d”, which stores N xn values, where N is the number of cells of nodes_training and xn is a 3D shape represented by 3M*1 vector. M = 642 in this case, and the elements of xn is shown below.

$$\vec{x}_n = [x_1, y_1, z_1, x_2, y_2, z_2, \dots x_M, y_M, z_M]^{\mathrm{T}} \quad (1 \le n \le N)$$

(2) Then, obtain the mean of x and assemble the 3M*N data matrix X, by using the following formula.

1. Assemble the $3M \times N$ (mean-adjusted) data matrix $\mathbf{X} = [\vec{x}_1 - \bar{x} \quad \vec{x}_2 - \bar{x} \ \ldots \ \vec{x}_N - \bar{x}]$, where $\bar{x} = \frac{1}{N}\sum_{n=1}^{N} \vec{x}_n$

(3) Compute the covariance matrix using the below formula.

$$\text{Compute the covariance matrix: } \mathbf{C} = \frac{1}{N-1}\mathbf{X}\mathbf{X}^{\mathrm{T}}$$

(4) Perform eigendecomposition of the covariance matrix, as shown below, and obtain U and sigma matrices. U is composed of u column vectors from order 1 to 3M and sigma is a matrix whose diagonal elements are the eigenvalues of C.

Perform an eigen-decomposition of the covariance matrix such that $\mathbf{C} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^{\mathrm{T}}$, where

$$\boldsymbol{\Sigma} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_{3M} \end{bmatrix}.$$

and $\mathbf{U}$ is a $3M \times 3M$ matrix. The column vectors $\vec{u}_n$ of the matrix $\mathbf{U} = [\vec{u}_1 \quad \vec{u}_2 \quad \cdots \quad \vec{u}_{3M}]$

Steps (1)-(4) show how the PDMbuild_cov function were created and eigen-decomposition of covariance matrix were performed.

(5) To create a PDMbuild_gram function, steps(1)-(2) are repeated, and the step(3) is replaced with computation of gram matrix using the following formula.

$$\mathbf{G} = \frac{1}{N-1}\mathbf{X}^{\mathrm{T}}\mathbf{X},$$

Then, eigen-decomposition of G matrix enables the acquirement of V matrix and sigma_G according to $\mathrm{G} = \mathrm{V}\Sigma\mathrm{V}^{\mathrm{T}}$. Then, U_G is obtained by using:

$$\mathbf{U} = \mathbf{X}\mathbf{V}\left((N-1)\boldsymbol{\Sigma}\right)^{-\frac{1}{2}}$$

4. Task 4 and 5

Task 4:

(1) Similarities of eigenvalues (sigma matrices):

a) The eigenvalue generated by the covariance matrix(sigma) and that generated by using the gram matrix (sigma_G) appear in decreasing order magnitude, this means that projection lies in the direction of largest variance which is the opposite direction to the decreasing trend.

b) Diagonal elements in both eigenvalues are non-zero values.

c) The centred parts of sigma and sigma_G coincide.

(2) Differences between eigenvalues

a) The size of sigma is different to that of sigma_G. It has been observed that the size of sigma_G tends to be smaller. This means that the PCA

performed using eigen-decomposition of the covariance matrix reduces the dimension of data while maintaining the structure of data. In contrast, PCA performed using eigen-decomposition of gram matrix reduces the dimensions and keep only the extracted "features" that can represent the original data.

b) The non-centred part of values of sigma_G is different to that in sigma. And sigma tends to have complex number values.

(3) Similarities of eigenvectors
   a) The magnitude of the values in the centred part of the eigenvectors decomposed using covariance matrix (U) is equal to that that of eigenvectors decomposed using gram matrix(U_G).
   b) The number of rows of U is equal to that of U_G, both are 1926.

(4) Differences between eigenvectors
   a) The size of U is larger than that of U_G. For U, the size is 1926*1926 and the size of U_G is 1926*200.
   b) Despite the magnitudes of centred part of values are the same , the sign may be different in U_G and U.
   c) Non-centred part of values in U is different to that in U_G.
   d) U has complex number values where U_G does not have any.

Task 5:

Results:

```
The time of building PDM basde on eigendecomposition of the covariance matrix: 5.846 seconds
The time of building PDM basde on eigendecomposition of the Gram matrix,  G: 0.099 seconds
```

Finding: the PDM based on the eigen-decomposition of the gram matrix(PDMbuild_gram) is faster to compute than that of the covariance matrix(PDMbuild_cov).

The size of the Gram matrix produced by PDMbuild_gram (200*200) is smaller as compared to that of the covariance matrix produced by PDMbuild_cov (1926*1926). Hence, the eigen-decomposition of the gram matrix is faster than that of the covariance matrix.

5. Task 6

Target:

Create PDMinstance that uses a PDM to generate a shape instance given a shape vector $\vec{\beta}$ and a particular value of L.

Steps :

In "CW_test" test script:

(1) Initially, perform the change of basis projection for all of the original shape vectors, and a shape vector B represents all shapes stored is obtained using the below formula. bn is a scalar weight (or score), $\vec{b}n$ = [$b1$, $b2$, ⋯ , bK]T is a new shape vector that represents the nth shape, and K = 3M or N, depending on whether the covariance matrix or the Gram matrix was

decomposed. And B is the shape vector contains $\vec{bn}$ from order of 1 to K. The L subset of B at (ntest)th shape is the input shape vector (beta) to the PDMinstance function.

$$\mathbf{B} = [\vec{b}_1\ \vec{b}_2\ \ldots \vec{b}_K] = \mathbf{U}^{\mathrm{T}}\mathbf{X}$$

(2) Store values of U, sigma and mean x vector x_m into a single structure called pdm.

In PDMinstance:

1. Initially, impose an error checking mechanism on the input values of L, and extract quantities like U, sigma and x_m from the input structure pdm.
2. Then, compute variance ratio, alpha, using the following formula.

$$\alpha = 100 \times (\sigma_L^2/\sigma_K^2)$$

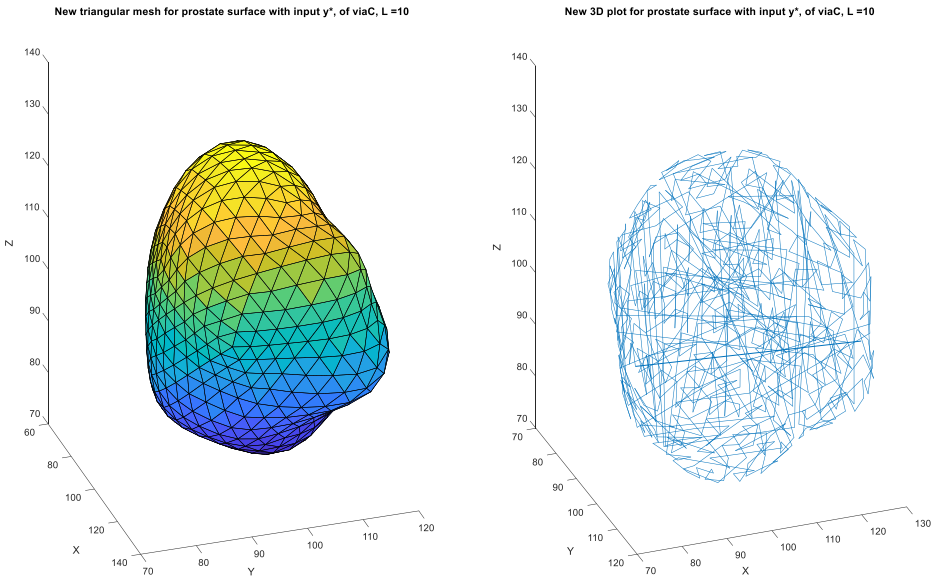
3. Obtain the L subset of U, UL. And approximate the new instance nth shape by using y = x_m + UL*beta

6. Task 7

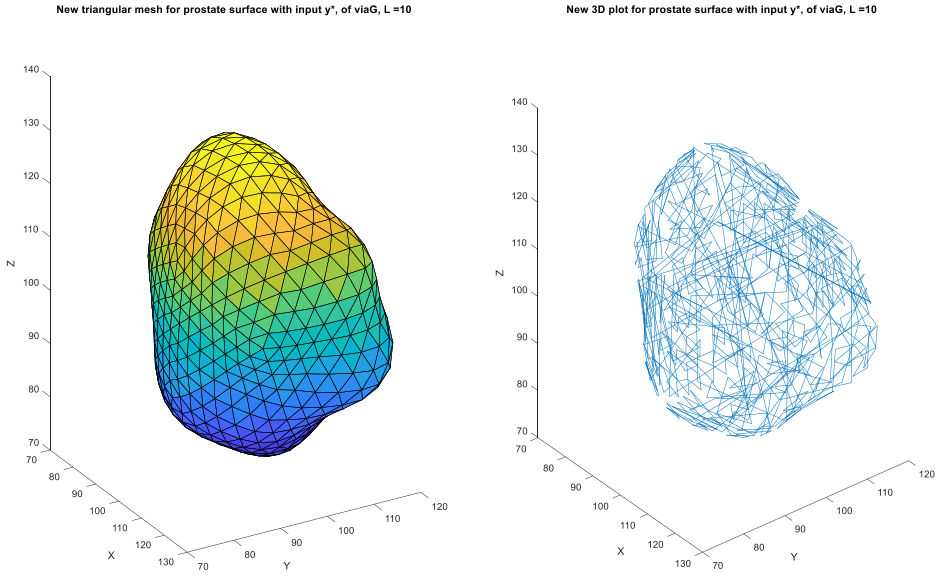
Target: Using PDMinstance and PDMvis to visualize the instantiated shapes, experiment calling the function with different values of L and bi to generate different shapes.

Results:

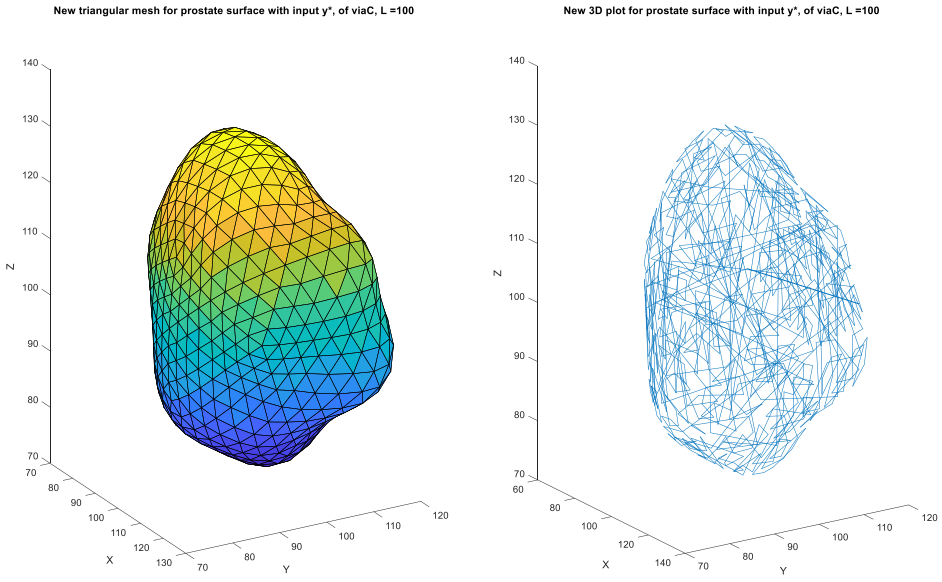
Case1.1: L =10, quanties like U, sigma are obtained from the eigen-decomposition of covariance matrix.



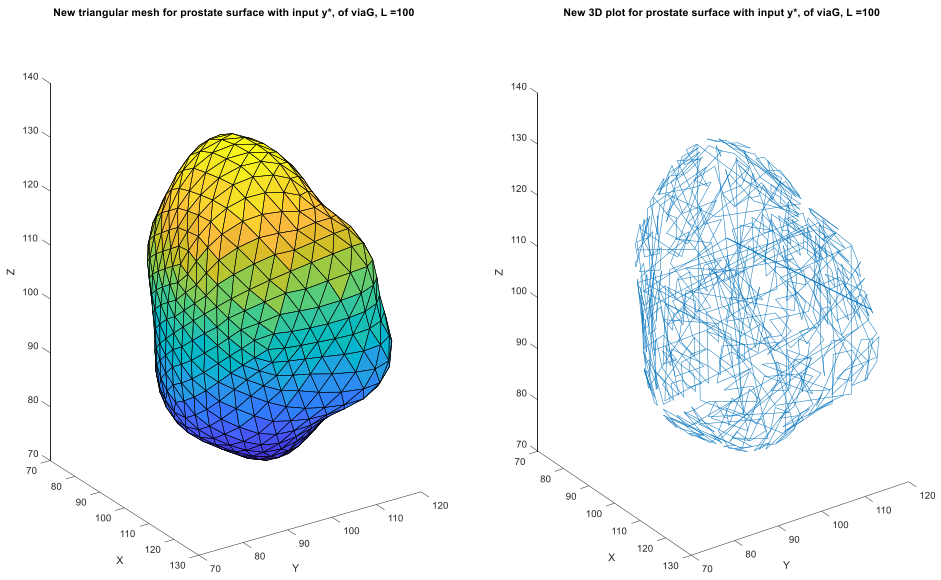
Case1.2: L = 10, quanties like U, sigma are obtained from the eigen-decomposition of the gram matrix.



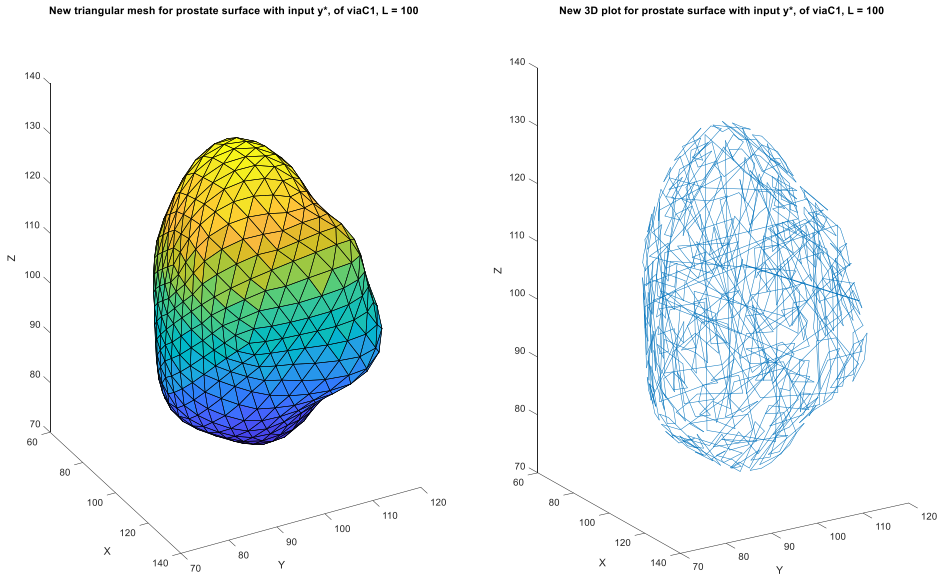
Case2.1: L =100, quantities like U, sigma are obtained from the eigen-decomposition of covariance matrix.



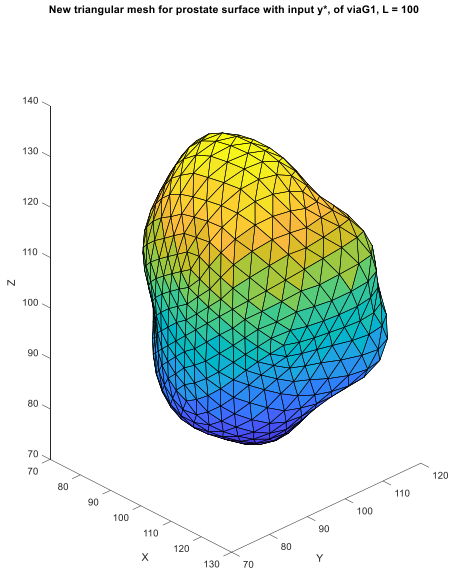
Case2.2: L =100, quantities like U, sigma are obtained from the eigen-decomposition of gram matrix.

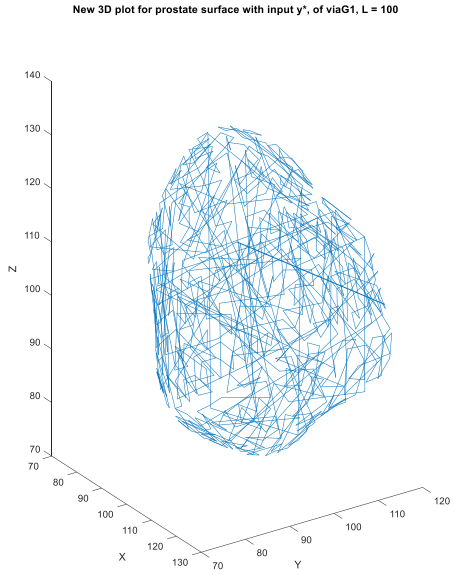


Case3.1: L =100, quantities like U, sigma are obtained from the eigen-decomposition of covariance matrix, with beta = 1.5* Beta.

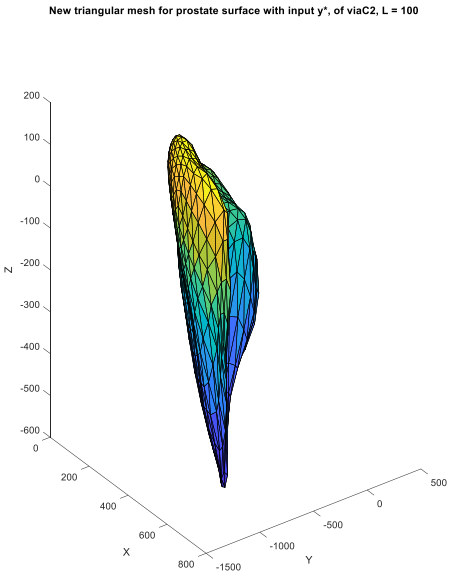


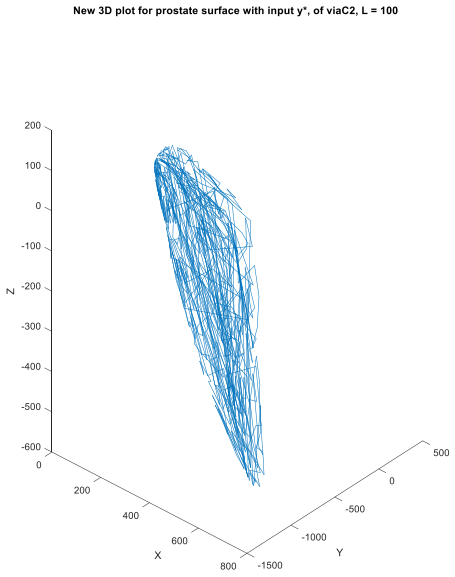
Case3.2: L =100, quantities like U, sigma are obtained from the eigen-decomposition of gram matrix, with beta = 1.5* Beta.



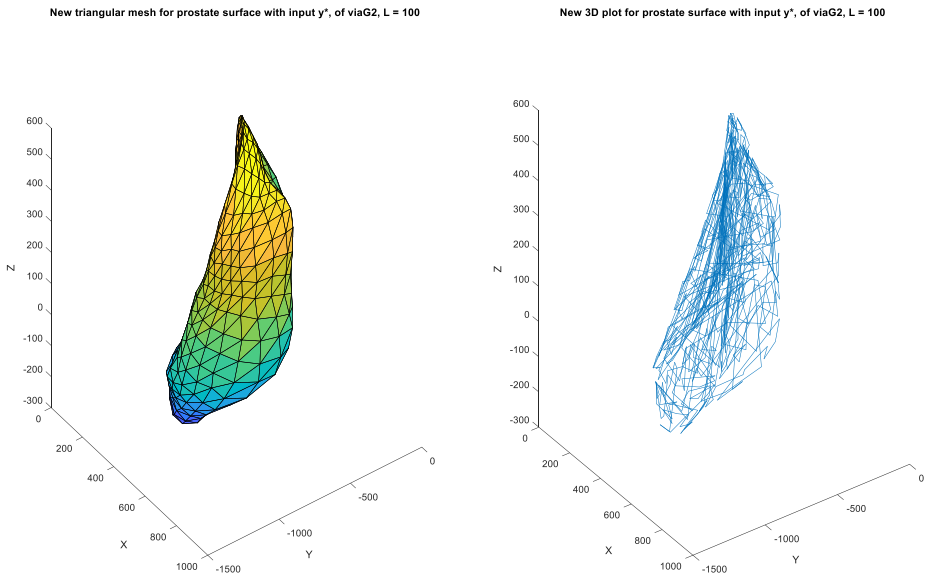


Case4.1: L =100, quantities like U, sigma are obtained from the eigen-decomposition of covariance matrix, with beta = 200* sqrt(λi).

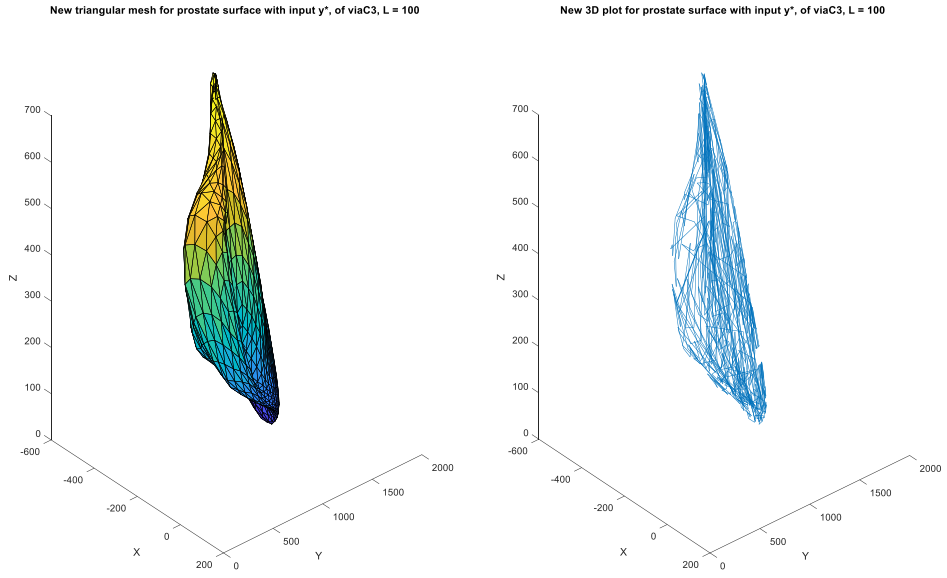




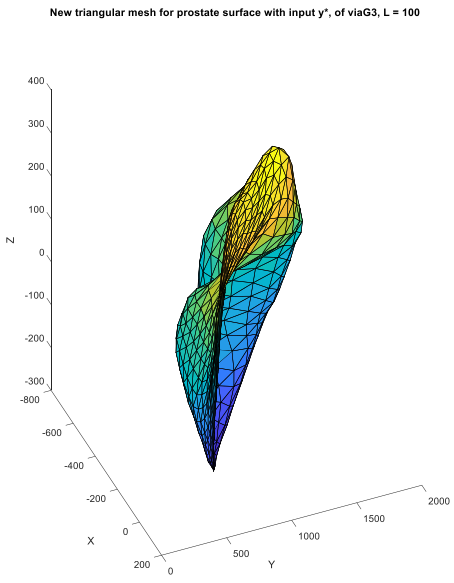
Case4.2: L =100, quantities like U, sigma are obtained from the eigen-decomposition of gram matrix, with beta = 200* sqrt(λi).

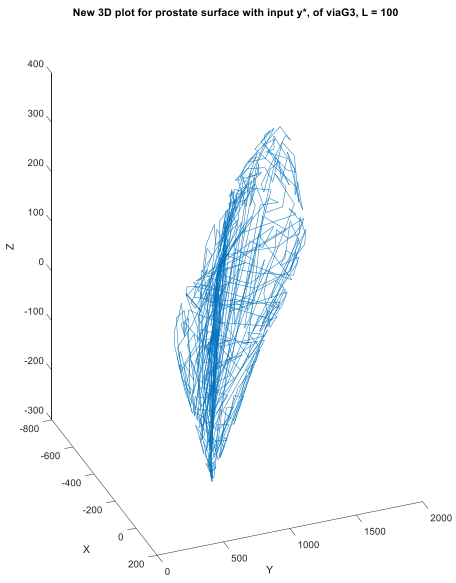


Case5.1: L =100, quantities like U, sigma are obtained from the eigen-decomposition of covariance matrix, with beta = -200* sqrt(λi).



Case5.2: L =100, quantities like U, sigma are obtained from the eigen-decomposition of covariance matrix, with beta = -200* sqrt(λi).





Findings:
When the values of bi exceed ±k*sqrt(λi) where k >> 3, the instantiated shape will become squeezed.

7. Task 8
   Target:
   Create PDMreconstruct that uses a PDM to reconstruct a vector $\vec{y}$, given a particular value of L

   Steps:
   In PDMreconstruct:
   (1) Initially, compute the shape vector, $\vec{\beta}y$, using $\vec{\beta}y$ = UL T($\vec{y} - \bar{x}$), where UL is the L subsets of U, T means the transpose, y is obtained from PDMinstance and $\bar{x}$ is the mean x vector stored in pdm.
   (2) Compute reconstructed shape coordinates,y_star, according to $\vec{y}* = \bar{x} + \mathbf{U}L * \vec{\beta}y$.
   (3) Computer the Root Mean Squared distance error by using the following formula, where M = 642 in this case:

$$\epsilon = \sqrt{((\vec{y}^* - \vec{y})^{\mathrm{T}}(\vec{y}^* - \vec{y})/M)}$$

8. Task9
   Target:
   Test PDMreconstruct and provide some examples to demonstrate that it works correctly.

Steps:

Testing strategy:

(1) Initially, extract the (ntest)th shape in the training sets. And the extracted shape of data, xn, is like y in $\beta^{\rightarrow} y$ = UL T($y^{\rightarrow} - \bar{x}$), representing a shape coordinate. Hence, we assign the values of y equals to that of xn.
(2) This y is then used as an input to the PDMreconstruct together with L_t and pdm produced in task 6, a reconstructed shape coordinates, y_star_rc, was then generated together with By and Root Mean Squared distance error (RMS_error) as outputs.
(3) If the size of y_star_rc is same as that of y, then we repeat the matrix of RMS_error to the same size as that of y and y_star_rc. Otherwise, return the error, as y_star_rc and y do not have the same size.
(4) If the dimensions of both shape coordinates match, then y values are used as a "ground truth data". According to the formula below, the reconstructed shape coordinates is supposed to be approximately equal to the generated instance shape coordinates.

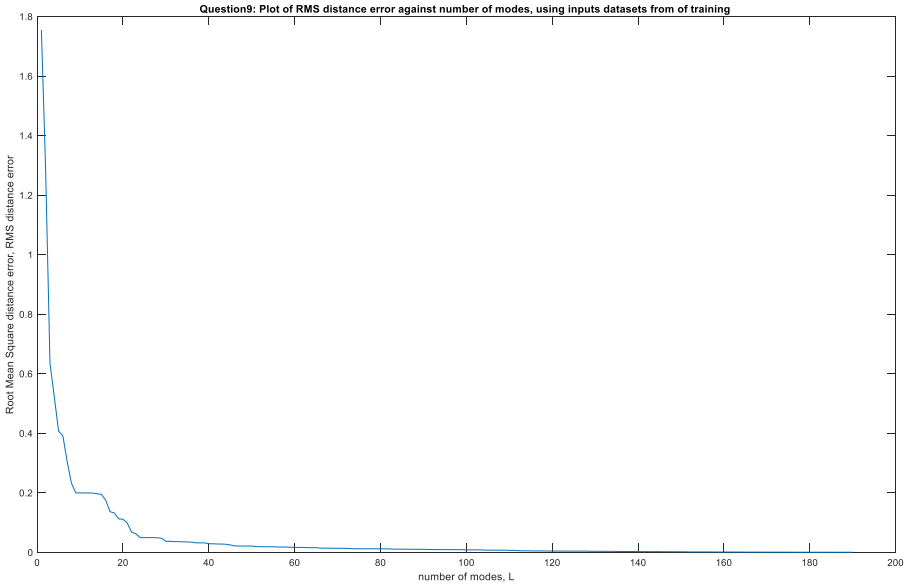
$$\vec{y}^{*} = \bar{x} + \mathbf{U}_L \vec{\beta}_y \approx \vec{y}$$

(5) Hypothesis: Hence, RMS error should have values of zero or very close to zero. And y must lie in the region of y_star_rc – RMS_error <= y <= y_star_rc + RMS_error. And this condition is defined as "bound conditions" in this report.
(6) A certain range of L_t is tested for different ntest numbers. If the bound condition is satisfied, then the corresponding L_t number is recorded in a array called L_true. This L-true array allows us to compare with the original range of L_t. If the ranges match, this means bound condition is satisfied for every single L_t value at the corresponding (ntest) th shape.
(7) A plot for RMS_error against L_t is also produced to see whether RMS error is close to zero.

Results:
Case: L_t = 1:190 and ntest = 1;
Interface:

```
PDMreconstruct works correctly! ntest = 1
```

Plot:

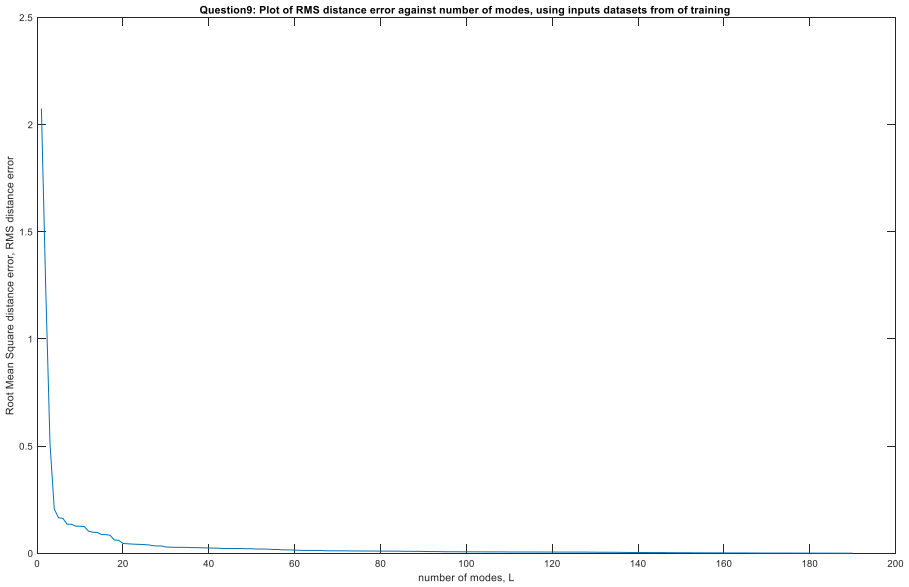


Case: L_t = 1:190 and ntest = 50;

Interface:

```
PDMreconstruct works correctly! ntest = 50
```

Plot:

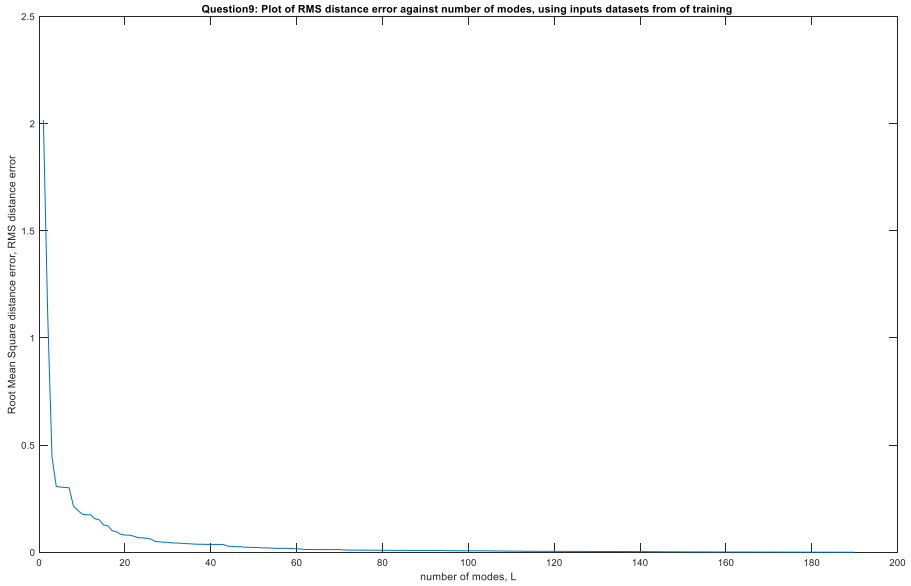


Case: L_t = 1:190 and ntest = 100;

Interface:

```
PDMreconstruct works correctly! ntest = 100
```

Plot:



Summary:

All of the RMS errors tend to have increasing values as L is reduced, however, most of errors are very close to zero or equal to zero. The large values of errors are known as numerical errors, which arise during computations due to round-off errors and truncation errors. And in all the tested cases for different n values, y lie in the range of bound conditions. Therefore, it can conclude that PDMreconstruct satisfies all the hypothesis proposed and PDMreconstruct is working correctly.

Task 10

Target:

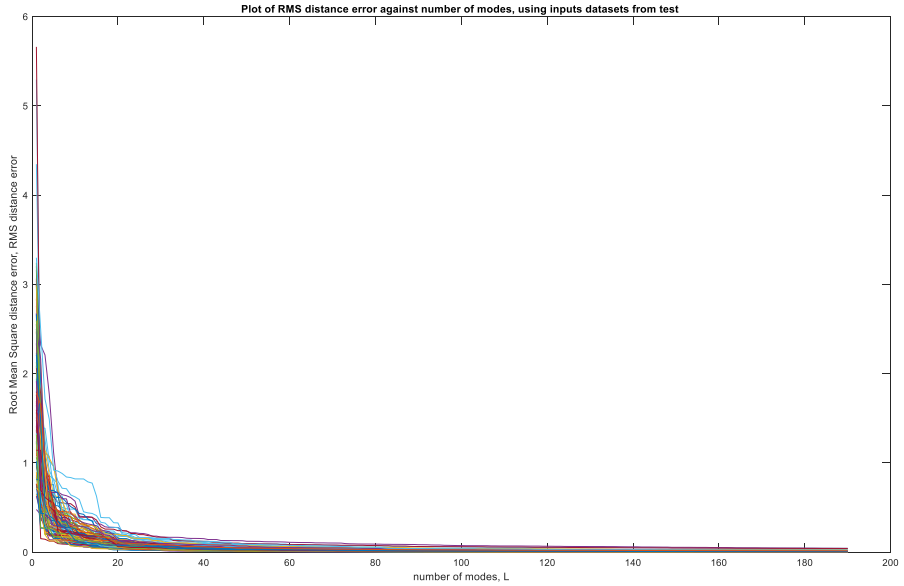
Using the test data provided in nodes_test, explore the relationship between the RMS distance error ϵ of shapes reconstructed using the PDM and L.

Steps:

(1) Initially, check the input of L within the allowed boundary.
(2) Extract the data from nodes_test and store those data into x_3d
(3) Compute the RMS error across all shapes for L = 1:190
(4) Plot RMS error against number of modes L

Results:

Plot: summarises this relationship of RMS vs L for the 100 test datasets, where ntest = 1



Summary:
Across all shapes, all of the RMS errors tend to have increasing values as L is reduced, however, most of errors are very close to zero or equal to zero. The large values of errors are known as numerical errors, which arise during computations due to round-off errors and truncation errors.

9. Task11
   Target:
   Modify PDMreconstruct so that it can handle missing points in vector $\vec{y}$, denoted by setting the missing point ordinates to take the value of NaN.
   Write a short script that "removes" p% of the nodes at random from each test dataset. Now, after choosing an appropriate value of L, call modified version of PDMreconstruct to reconstruct the test shapes using these sparse point datasets.

   Steps:
   In rd_remove test file:
   (1) Remove p% of values randomly from the y vectors.
   (2) RMS_error between the reconstructed data and the original test data is computed, by inputting the "removed" test data at ntest th shape.
   (3) RMS_error between the original test data and the mean training shape described by x_m is computed, by inputting the original test data at ntest th shape.

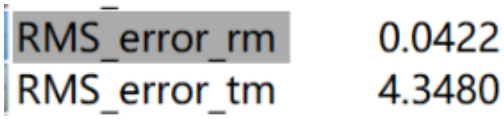
   In modified PDMreconstruct file:
   Additional steps compared to original file:
   (1) Check if there are any values in input y are NAN(not a number), if there is,

fill those values with the constant value 0.In the corresponding positions, replace the values in x_m with 0.

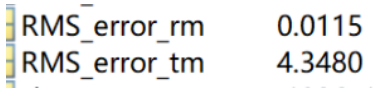
(2) If test_mean equals to 1, return RMS error between the original test data and the mean training shape described by x_m is computed. Otherwise, return RMS error as one between the reconstructed data and input dataset y.

Results:

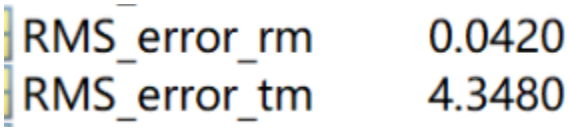
Case: p = 0.2, ntest = 1 and L =190



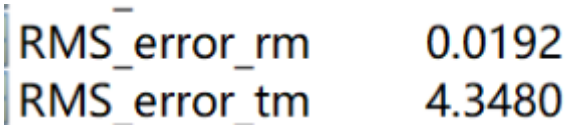
Case: p = 0, ntest = 1 and L=190



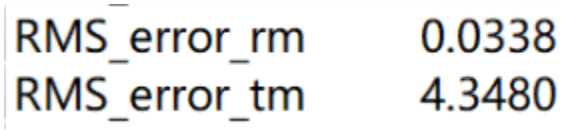
Case: p = 0.25, ntest = 1 and L = 190



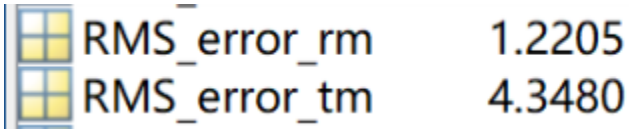
Case: p = 0,ntest = 1 and L =100



Case: p = 0.3, ntest = 1 and L = 100



Case: p = 12, ntest =1 and L = 100



Case: p = 30,ntest =1 and L =100

RMS_error_rm	1.8730
RMS_error_tm	4.3480

Case: p = 60,ntest =1 and L =100

RMS_error_rm	2.0917
RMS_error_tm	4.3480

Commments:

It has been observed that the p limit of RMS distance error between the reconstructed data and the original test data(RMS_error_rm) is smaller than 1% of the RMS distance between the original test data and the mean shape described by x(RMS_error_tm), is less than 0.3 . A higher value of L will result in a lower RMS_error_rm. Above the 0.3, it is much less likely to get RMS_error_rm smaller than 1% of RMS_error_tm. A higher value of p will lead to a generally larger value of RMS_error_tm. However, as the value gets higher, the increasing amount will become less.