

Επιφάνεια Πρωτεϊνών

ΕΡΓΑΣΙΑ 4: ΥΠΟΛΟΓΙΣΤΙΚΗΣ ΚΑΙ ΕΠΕΞΕΡΓΑΣΙΑΣ
ΓΕΩΜΕΤΡΙΑΣ

ΚΟΤΣΙΝΗΣ ΔΗΜΗΤΡΙΟΣ – 1059482

Περιεχόμενα

Εισαγωγή	3
Σημείωση	4
Ερωτήματα	5
Ερώτημα 1 (Task 1).....	5
Ερώτημα 2 (Task 2).....	7
Ερώτημα 3 (Task 3).....	10
Ερώτημα 4 (Task 4).....	12
Υπο-ερώτημα i	12
Υπο-ερώτημα ii	13
Ερώτημα 5 (Task 5).....	16
Ερώτημα 6 (Task 6).....	19
Ερώτημα 7 (Task 7).....	21
Task 4	21
Task 5	22
Task 6	24
Παράρτημα(Ανάλυση Κώδικα)	26
Header File.....	26
//--- Task 1 ---//	27
//--- Task 2 ---//	27
//--- Task 3 ---//	28
//--- Task 4i ---//	28
//--- Task 4ii ---//	28
//--- Task 5 ---//	28
//--- Task 6 ---//	29
//--- Task 7 ---//	29
Source File	30
//--- Task 0 ---//	31
//--- Task 1 ---//	33
//--- Task 2 ---//	36
//--- Task 3 ---//	40
//--- Task 4i ---//	43
//--- Task 4ii ---//	45
//--- Task 5 ---//	46
//--- Task 6 ---//	49
//--- Task 7 ---//	52
//--- Draw ---//	58
Βιβλιογραφία.....	61

//--- Task 2 ---//	61
//--- Task 3 ---//	61

Εισαγωγή

Η εργασία «επιφάνειες πρωτεϊνών(Protein Surfaces)» του μαθήματος υπολογιστικής και επεξεργασίας γεωμετρίας, αναφέρεται στην επεξεργασία τρισδιάστατων πρωτεϊνών και στην ταξινόμησή τους σε ομάδες. Με λίγα λόγια, μας δίνονται ένα σύνολο πρωτεϊνών εκπαίδευσης (dataset πλήθους 369) και δοκιμών (test_set πλήθους 41) και μέσω των γεωμετρικών και φυσικοχημικών χαρακτηριστικών τους, τα ταξινομούμε στις σωστές ομάδες.

Η εργασία αυτή χωρίζεται σε τρία μέρη. Το πρώτο μέρος της εργασίας περιλαμβάνει την απεικόνιση των φυσικοχημικών και γεωμετρικών χαρακτηριστικών των πρωτεϊνών. Στο δεύτερο μέρος υπολογίζουμε την μήτρα ανομοιότητας (dissimilarity matrix) των πρωτεϊνών εκπαίδευσης, σύμφωνα με τα γεωμετρικά χαρακτηριστικά, και μέσω αυτής τις κατηγοριοποιούμε σε πέντε ομάδες(0-4). Μετά για κάθε πρωτεΐνη από το σύνολο δοκιμής, ανακτούμε την πιο όμοια πρωτεΐνη από το σύνολο εκπαίδευσης και αξιολογούμε το ποσοστό ακρίβειας της ανάκτησης (retrieval). Τέλος, στο τρίτο μέρος ακολουθούμε την ίδια διαδικασία με το δεύτερο, αλλά χρησιμοποιούμε συνδυασμό φυσικοχημικών και γεωμετρικών χαρακτηριστικών.

Σημείωση

Στα ερωτήματα που ακολουθούν περιγράφονται τα βήματα για την επίλυση της άσκησης, καθώς και τα αποτελέσματα αυτών. Ο κώδικας περιγράφεται αναλυτικά στο παράρτημα, δηλαδή τι συναρτήσεις, δομές, κλάσεις και βιβλιοθήκες υλοποιήθηκαν και χρησιμοποιήθηκαν για την εργασία και πώς λειτουργεί ο κώδικας. Επίσης τα γραφήματα έγιναν με το πρόγραμμα MATLAB και οι περισσότερες εικόνες δημιουργήθηκαν στο πρόγραμμα PowerPoint. Ακόμα ορισμένοι τύποι πάρθηκαν από εργασίες που αναφέρονται στην βιβλιογραφία. Τέλος στο .zip αρχείο θα είναι δύο φάκελοι. Ο ένας έχει όνομα src και θα περιέχει τα αρχεία των κωδίκων(.h .cpp) και ο άλλος resources ο οποίος περιέχει τους φακέλους που χρησιμοποιούνται στην άσκηση(όχι με αυτούς που δόθηκαν μαζί με την άσκηση).

Ερωτήματα

Ερώτημα 1 (Task 1)

Ο στόχος αυτού του ερωτήματος είναι η απεικόνιση των τριών φυσικοχημικών ιδιοτήτων μίας πρωτεΐνης. Κάθε πρωτεΐνη από το σύνολο εκπαίδευσης και δοκιμών, περιέχει ένα κείμενο με τις τρεις φυσικοχημικές ιδιότητες σε αριθμητική τιμή. Το κείμενο αυτό περιλαμβάνει τρεις στήλες, εκ των οποίων, η καθεμία αντιστοιχεί σε μία ιδιότητα, ενώ η κάθε γραμμή στήλης, αναφέρεται στο διάνυσμα του αντικειμένου. Επομένως, η πρώτη ενέργεια είναι να ανοίξουμε τα αρχεία αυτά και να τοποθετήσουμε τις ιδιότητες σε τρεις πίνακες ξεχωριστά.

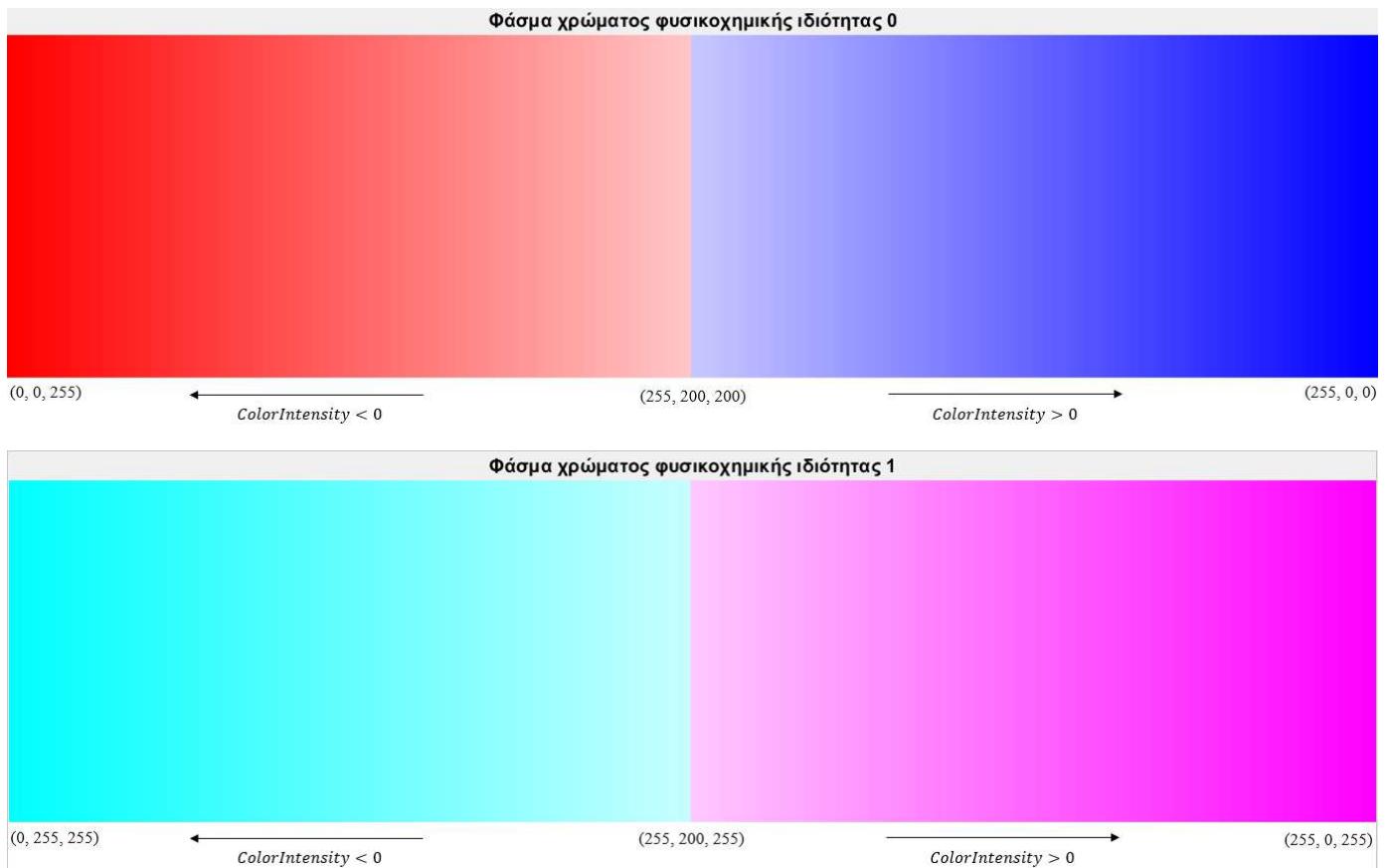
Όταν ολοκληρωθεί αυτή η διαδικασία, ορίζουμε σε κάθε τρίγωνο του αντικείμενου μία τιμή, την μέση(median value), των ιδιοτήτων που έχουν τα τρία σημεία του τριγώνου. Με αυτή την τιμή, μπορούμε να χρωματίσουμε τα τρίγωνα. Για τον χρωματισμό, χρησιμοποιούμε το σύστημα RGB. Στην συνέχεια, υπολογίζουμε την μέγιστη και την ελάχιστη τιμή κάθε ιδιότητας και μέσω του παρακάτω τύπου υπολογίζουμε την ένταση του χρώματος που θα έχει το κάθε τρίγωνο:

$$\text{ColorIntensity} = \frac{\text{MedianValue} * 200}{\text{MaxValue}}, \text{MedianValue} \geq 0, \quad (1.1)$$

$$\text{ColorIntensity} = \frac{\text{MedianValue} * 200}{\text{MinValue}}, \text{MedianValue} < 0, \quad (1.2)$$

Βάζουμε στον τύπο την τιμή 200 διότι με την τιμή 255 φαίνεται πολύ έντονα το άσπρο χρώμα με αποτέλεσμα να καλύπτει τα χρώματα των ιδιοτήτων. Τις τιμές αυτές τις αποθηκεύουμε σε ξεχωριστά διανύσματα.

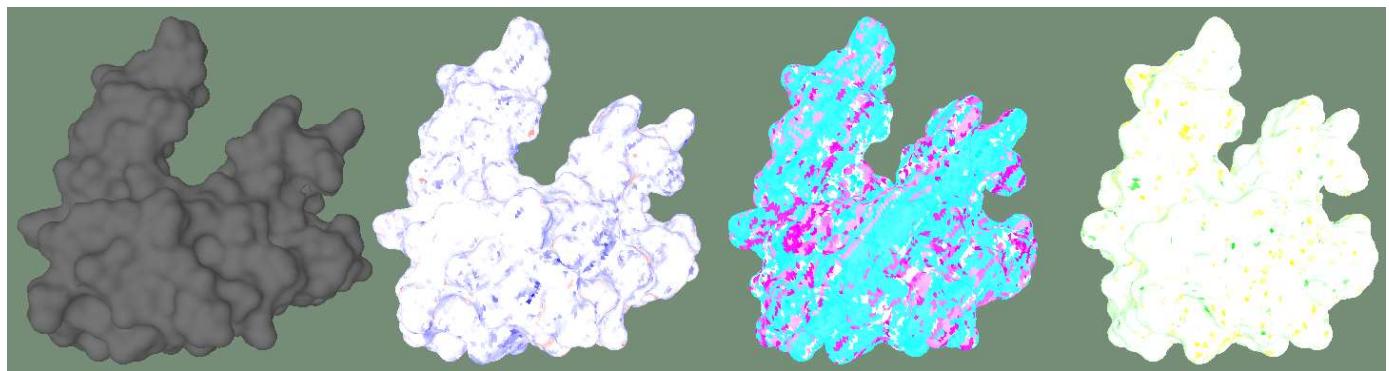
Έτσι τα φάσματα χρωμάτων που θα χρησιμοποιηθούν για τον χρωματισμό κάθε ιδιότητας είναι τα εξής:



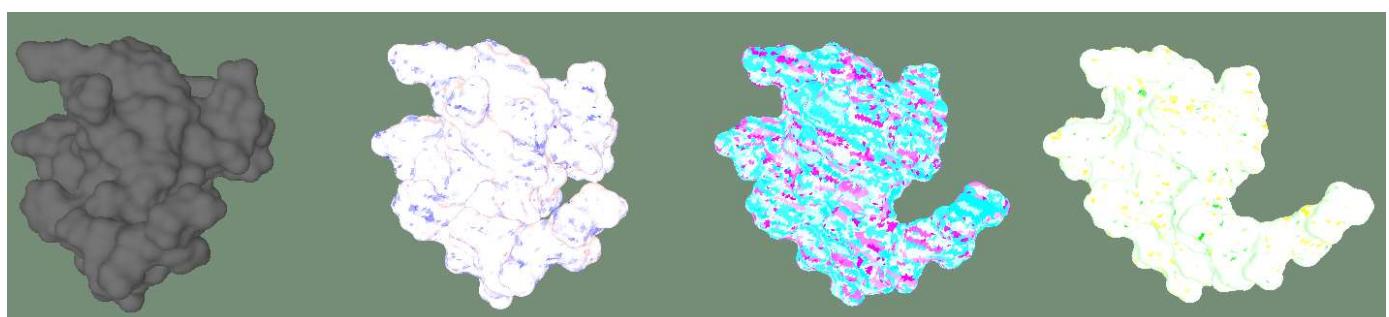


Τα αποτελέσματα που παίρνουμε είναι τα εξής:

“dataset/0.obj”



“test_set/20.obj”



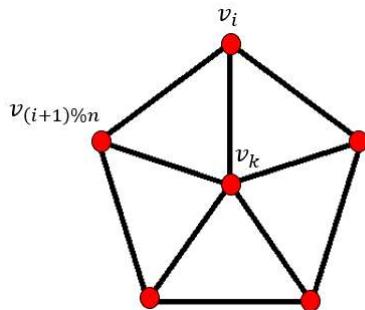
Ερώτημα 2 (Task 2)

Το ερώτημα αυτό ζητάει να ορίσουμε συνάρτηση η οποία βρίσκει την γκαουσιανή και την μέση καμπυλότητα των διανυσμάτων v_k μίας πρωτεΐνης της επιλογής μας. Για τον υπολογισμό αυτόν θα χρησιμοποιήσουμε την μέθοδο του Gauss-Bonnet Scheme[2.1].

$$K = \frac{2\pi - \sum_{i=0}^{n-1} a_i}{\frac{1}{3}A}, \text{gauss curvature, (2.1)}$$

$$H = 0.75 \frac{\sum_{i=0}^{n-1} \|e_i\| b_i}{A}, \text{mean curvature, (2.2)}$$

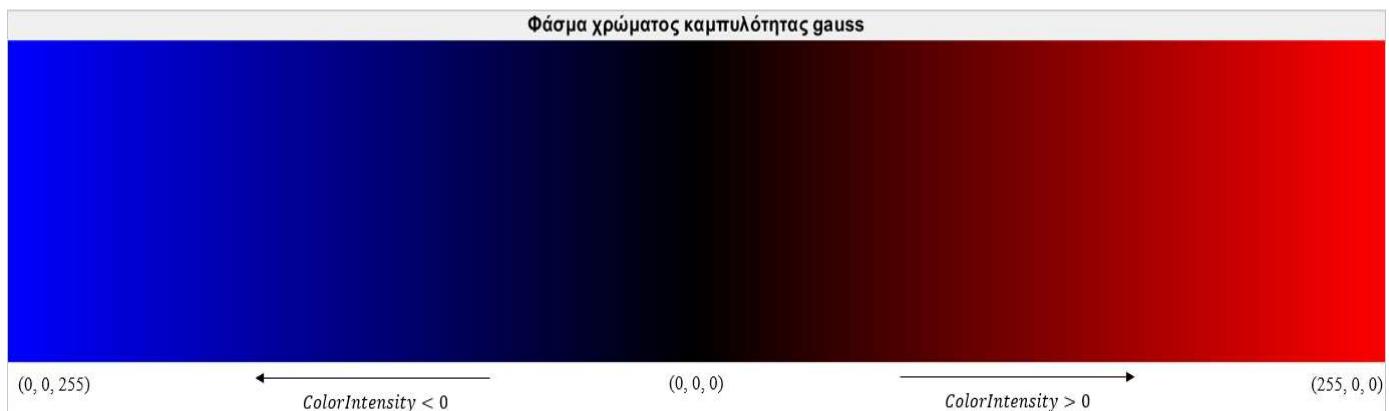
Οπου το a_i είναι οι γωνίες των τριγώνων που έχουν κοινό σημείο το διάνυσμα v_k , A το άθροισμα των εμβαδών των τριγώνων, $e_i = \overline{v_k v_i}$ η ακμή του v_k με τα γειτονικά του διανύσματα και $b_i = \tan \frac{N_i^v}{N_{(i+1)\%n}^v}$ μετρά την κανονική απόκλιση. Το $N_i^v = \frac{(v_i - v)X((v_{(i+1)\%n} - v))}{\|(v_i - v)X((v_{(i+1)\%n} - v))\|}$.

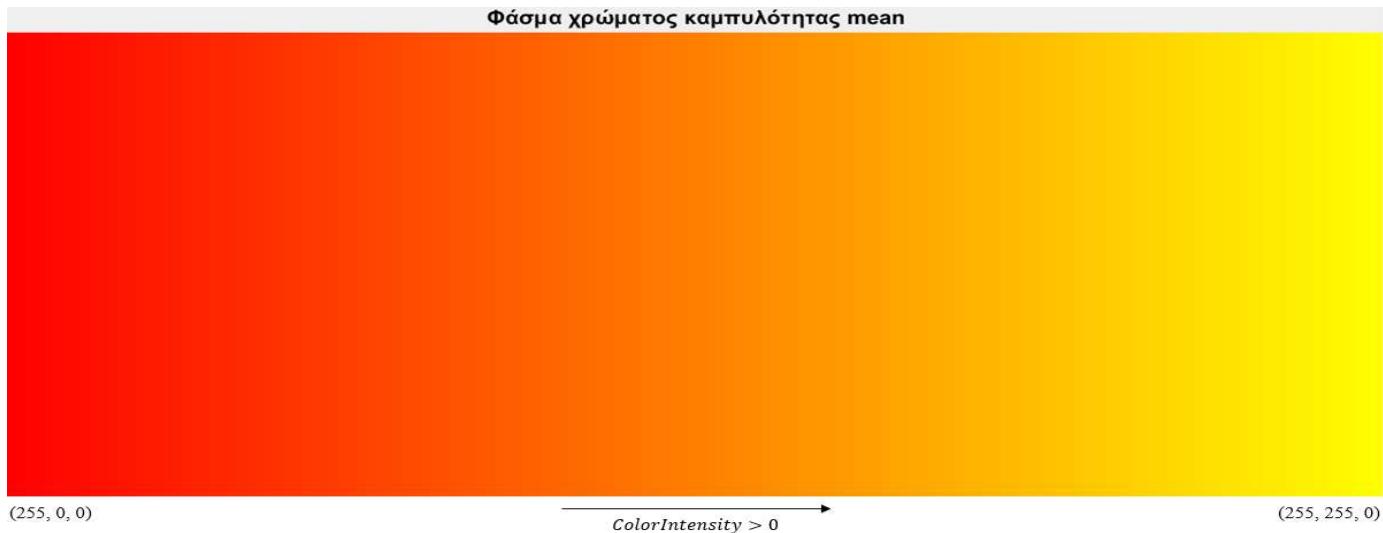


Αρχικά, προκειμένου να υπολογίσουμε τις καμπυλότητες εύκολα δημιουργούμε μία δομή *vertex_neighbours*, η οποία αναφέρεται στο διάνυσμα στο οποίο θέλουμε να υπολογίσουμε τις καμπυλότητες. Αυτό περιέχει τις θέσεις των γειτονικών διανυσμάτων του και τις θέσεις των τριγώνων που έχουν κοινό σημείο αυτό και τα N_i^v . Πολύ χρήσιμη δομή, την οποία θα χρησιμοποιήσουμε και στο ερώτημα 3 για να κατασκευάσουμε τον δυαδικό Γράφο.

Όταν βρούμε την δομή αυτή, για κάθε διάνυσμα υπολογίζουμε τις καμπυλότητές τους και χρωματίζουμε τα τρίγωνα με τον ίδιο τρόπο όπως στο ερώτημα 1.

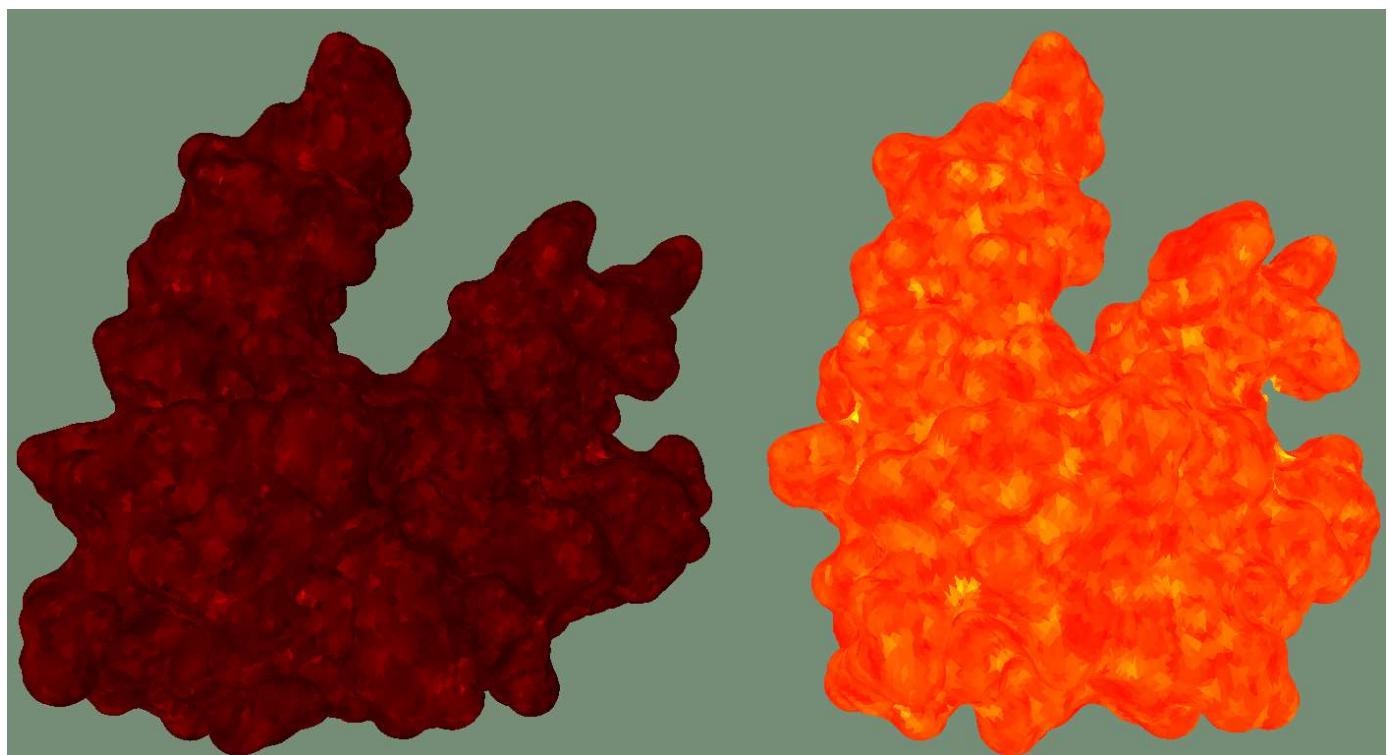
Έτσι, τα φάσματα χρωμάτων που θα χρησιμοποιηθούν για τον χρωματισμό κάθε ιδιότητας είναι τα εξής:



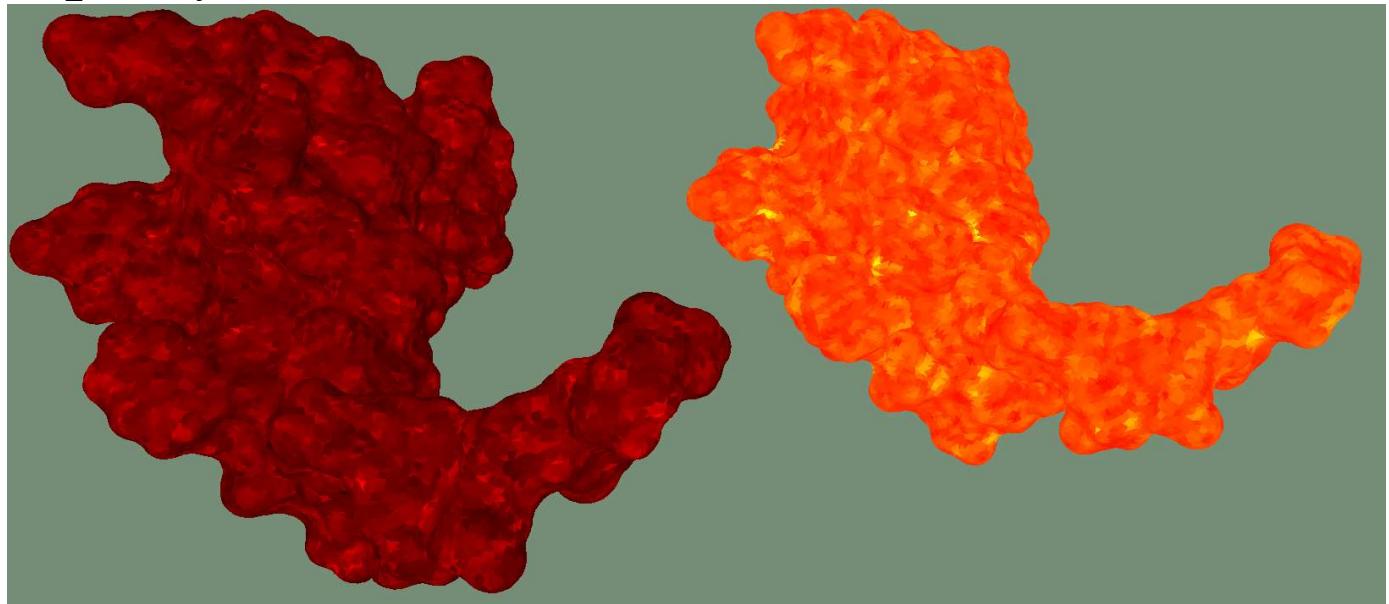


Τα αποτελέσματα που παίρνουμε είναι τα εξής:

“dataset/0.obj”

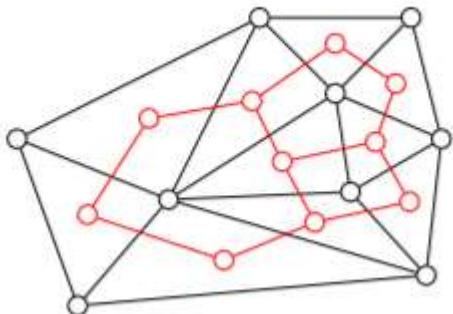


“test_set/20.obj”

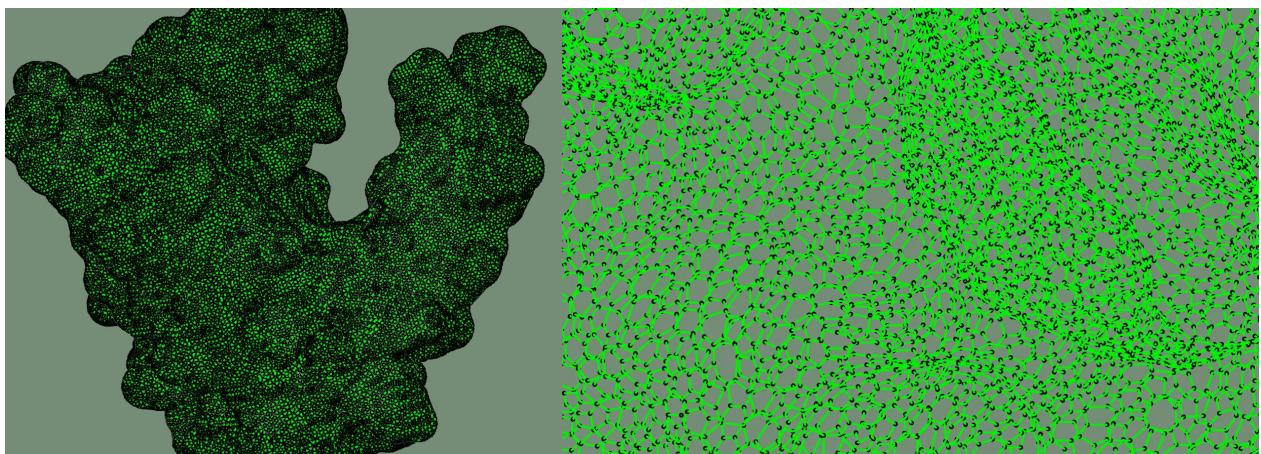


Ερώτημα 3 (Task 3)

Σε αυτό το ερώτημα, ζητείται να υπολογίσουμε τον χάρτη προεξοχής (protrusion map). Αρχικά, θα πρέπει να βρούμε τον δυαδικό Γράφο (dual graph) της πρωτεΐνης εισόδου. Δηλαδή υπολογίζουμε το κέντρο μάζας των τριγώνων του αντικειμένου και ενώνουμε τα κέντρα αυτά με τα αντίστοιχα κέντρα των γειτονικών τριγώνων. Έτσι δημιουργείται ένας Γράφος στον οποίο το κάθε κέντρο συνδέεται με τρία άλλα κέντρα, επειδή το τρίγωνο έχει τρεις πλευρές και η επιφάνεια του αντικειμένου είναι κλειστή, άρα τρία γειτονικά τρίγωνα.



“dataset/0.obj”



Η αρχική σκέψη για να βρούμε με ποια τρίγωνα συνορεύει το κάθε κέντρο, είναι να ελέγξουμε το κάθε τρίγωνο με όλα τα άλλα τρίγωνα του αντικειμένου το οποίο είναι χρονοβόρο σαν μέθοδος. Για να μειωθεί ο χρόνος, χρησιμοποιύμε την δομή *vertex_neighbours* του ερωτήματος 2 η οποία περιέχει για κάθε σημείο της πρωτεΐνης τα τρίγωνα που ανήκει. Έτσι ο έλεγχος για την γειτνίαση γίνεται σε αυτά τα τρίγωνα κάθε φορά και μειώνεται η χρονική πολυπλοκότητα.

Ο κάθε κόμβος του δυαδικού γράφου είναι μία δομή *dual_vertex*, η οποία περιέχει τις εξής μεταβλητές:

- *dual_v*: Ο κόμβος του δυαδικού γράφου, το κέντρο μάζας τριγώνου.
- *index_vec*: Θέση στοιχείου στον Γράφο.
- *count_neigh*: Αριθμός γειτόνων
- *neigh_dual_vec[3]*: Θέσεις των γειτόνων του στον Γράφο.
- *distances[3]*: Αποστάσεις από τους γειτόνους του.(Ευκλείδεια απόσταση)
- *area*: Εμβαδόν τριγώνου.
- *geodesic*: Γεωδαισιακή απόσταση.(Δεν είναι πίνακας απλά μία μεταβλητή η οποία χρησιμεύει στον αλγόριθμο Dijkstra)

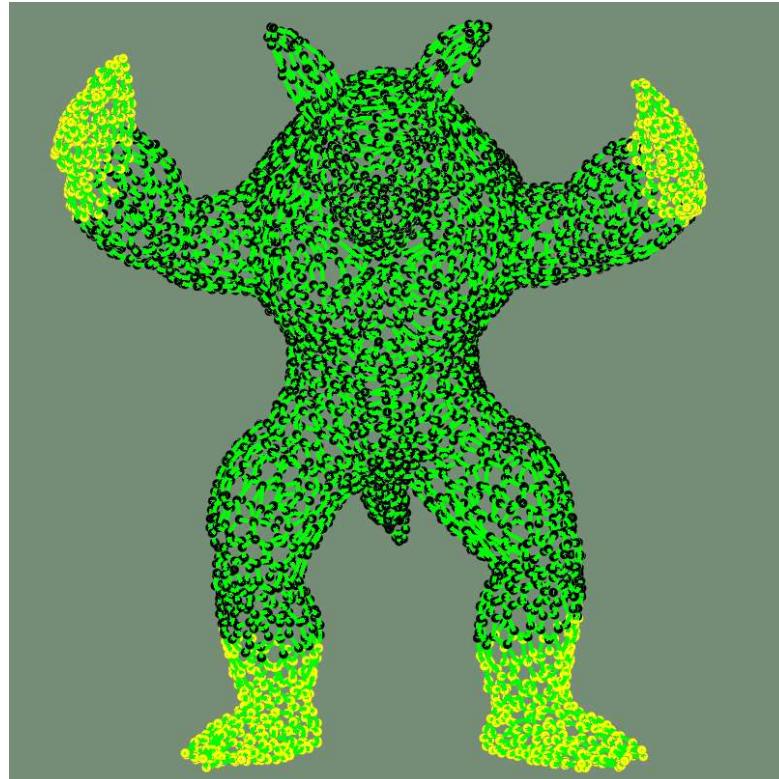
- p : Βαθμός προεξοχής(protrusion degree).
- *read*: Δείχνει αν ένας κόμβος έχει εξεταστεί από τον αλγόριθμο του Dijkstra.

Για να υπολογίσουμε τον βαθμό προεξοχής θα χρησιμοποιήσουμε τον αλγόριθμο του Dijkstra για τον υπολογισμό της γεωδισιακής απόστασης και από τον τύπο[3.1]:

$$p(u) = \sum_{i=1}^N g(u, v_i) * area(v_i) \quad (3.1)$$

Οπου u το δυαδικό διάνυσμα που εξετάζουμε, v_i κόμβος του δυαδικού γράφου, η συνάρτηση $g(.)$ υπολογίζει την γεωσιαδική απόσταση μέσω του αλγορίθμου Dijkstra και $area(.)$ το εμβαδόν του τριγώνου που αντιστοιχεί στον δυαδικό κόμβου v_i .

Παρακάτω απεικονίζεται ένα αντικείμενο στο οποίο για συγκεκριμένο κατώφλι, χρωματίζονται ορισμένοι κόμβοι.



Παρατηρούμε όμως δεν χρωματίζονται όλα τα άκρα. Οπότε χρησιμοποιούμε την μέθοδο N-rings για να βρούμε τα σημεία που προεξέχουν πιο πολύ από το αντικείμενο. Με την μέθοδο N-rings παίρνουμε N κόμβους και ελέγχουμε πιο έχει τον μεγαλύτερο βαθμό προεξοχής και βρίσκεται πάνω από ένα συγκεκριμένο κατώφλι.

ΣΗΜΕΙΩΣΗ: Δεν κατάφερα να βρω τον χάρτη προεξοχής, πρώτον διότι οι πρωτεΐνες είχαν μεγάλο πλήθος διανυσμάτων και αργούσε να βγάλει αποτελέσματα και δεύτερον δεν βρήκα τις κατάλληλες τιμές στο N-ring για να βρει τα σημεία προεξοχής.

Ερώτημα 4 (Task 4)

Ο πίνακας ανομοιομορφίας (dissimilarity matrix) είναι ένας τετραγωνικός πίνακας ο οποίος δείχνει τον βαθμό ανομοιομορφίας των αντικειμένων μεταξύ τους, σύμφωνα με γεωμετρικά χαρακτηριστικά ή όσον αναφορά τις πρωτεΐνες και με τις φυσικοχημικές ιδιότητές τους. Αν για δύο αντικείμενα, ο βαθμός ανομοιομορφίας τους είναι κοντά στο μηδέν, τότε έχουν πολύ μικρές διαφορές και αν πλησιάζει το ένα έχουν αρκετά μεγάλες. Ένα αντικείμενο με τον εαυτό του έχουν βαθμό μηδέν(διαγώνιος του πίνακα μηδενική). Ο πίνακας έχει πεδίο τιμών $[0, 1]$, διότι κάνουμε κανονικοποίηση με την μεγαλύτερη τιμή του.

Για να υπολογίσουμε τον βαθμό ανομοιομορφίας θα βρούμε το ιστόγραμμα του κάθε αντικειμένου για ένα συγκεκριμένο χαρακτηριστικό και θα υπολογίσουμε το τετραγωνικό σφάλμα κάθε περιοχής από τα δύο ιστογράμματα και θα το αθροίσουμε:

$$error = \sum_{i=0}^{HistSize-1} (histObjA[i] - histObjB[i])^2 \quad (4.1)$$

Το πεδίο ορισμού του ιστογράμματος θα είναι το διάστημα $[0, 1]$. Έτσι θα πρέπει να κανονικοποιήσουμε τα χαρακτηριστικά στο διάστημα αυτό. Ο τύπος είναι:

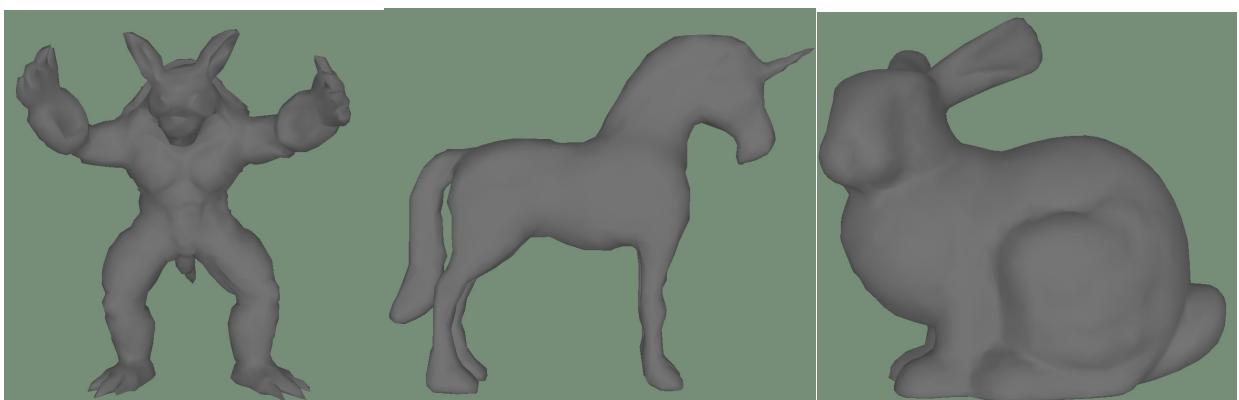
$$NormalizeValue = \frac{DataValue - MinValTotal}{MaxValueTotal - MinValueTotal} \quad (4.2)$$

Όπου *DataValue* η τιμή του χαρακτηριστικού, *MinValTotal* και *MaxValueTotal* η μικρότερη και η μέγιστη τιμή του χαρακτηριστικού για όλα τα αντικείμενα που εξετάζονται. Η μικρότερη και η μέγιστη τιμή βρίσκονται με προ επεξεργασία (Παράρτημα).

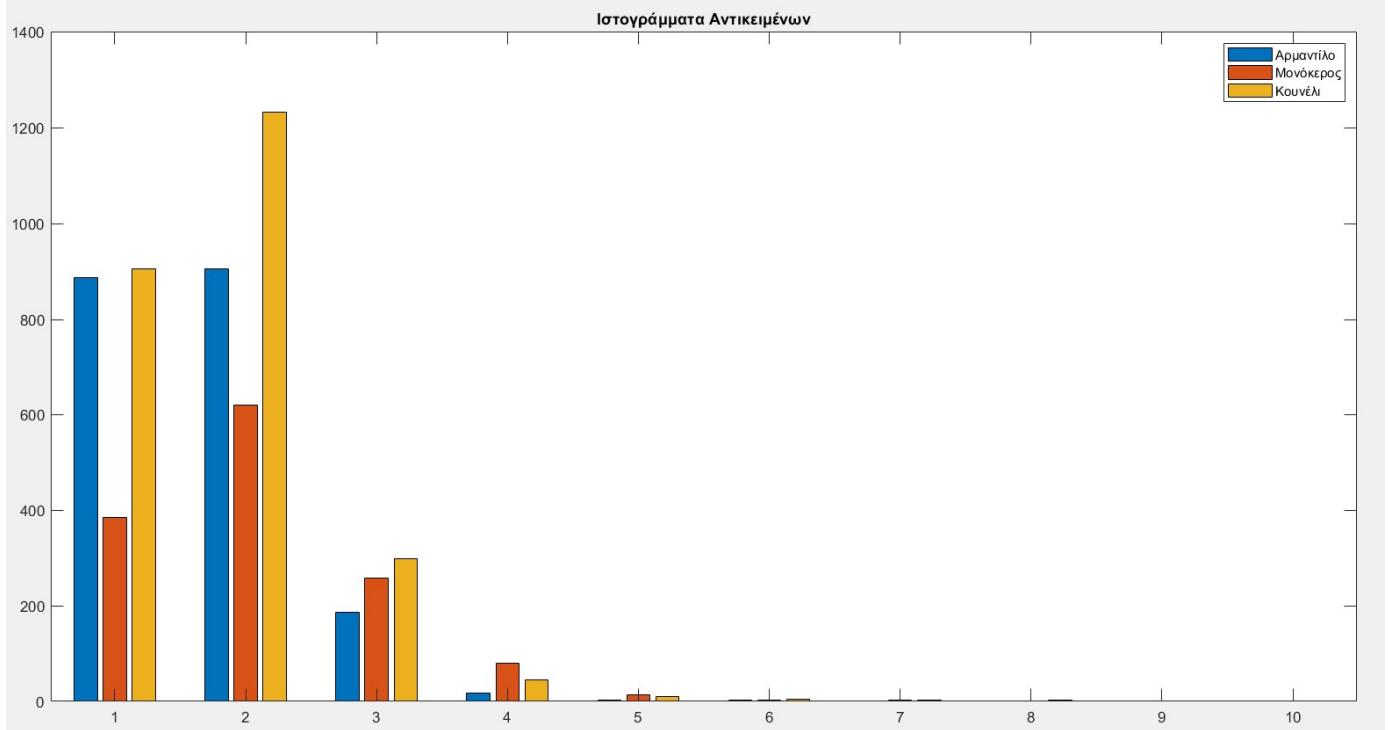
Στην συνέχεια το ιστόγραμμα το χωρίζουμε σε N υπό-διαστήματα, και αν το *NormalizeValue* είναι μέσα σε ένα από αυτά τότε το ανξάνουμε κατά ένα.

Υπό-ερώτημα i

Αρχικά θα κατασκευάσουμε τον πίνακα ανομοιομορφίας τριών αντικειμένων και θα τα συγκρίνουμε με την καμπύλοτη Gauss. Τα αντικείμενα αυτά είναι «armadillo_low_low.obj», «unicorn_low_low.obj» και «bunny_low.obj».



Τα ιστογράμματα τους με HIST_SIZE = 10 είναι τα εξής:



Και ο πίνακας ανομοιομορφίας είναι:

	Αρμαντίλο	Μονόκερος	Κουνέλι
Αρμαντίλο	0	1	0,2383
Μονόκερος	1	0	0,3787
Κουνέλι	0,2383	0,3787	0

Παρατηρούμε ότι ο αρμαντίλο έχει μεγάλη διαφορά με τον μονόκερο και με το κουνέλι έχει την πιο κοντινή ομοιότητα.

Επίσης να σημειώσουμε ότι στο ιστόγραμμα δεν κάνουμε κανονικοποίηση με τον αριθμό των διανυσμάτων που έχει το αντικείμενο διότι αν δύο αντικείμενα σε ένα διάστημα του ιστογράμματος, η διαφορά των τιμών τους είναι μεγάλη, με την κανονικοποίηση αυτή, υπάρχει πιθανότητα να έχουν την ίδια κατανομή στο διάστημα αυτό και ο βαθμός ανομοιομορφίας να προσεγγίζει το μηδέν.

Υπό-ερώτημα ii

Στο υπό-ερώτημα αυτό θα υπολογίσουμε τον πίνακα ανομοιομορφίας των 369 πρωτεϊνών από το σύνολο εκπαίδευσης dataset. Θα χρησιμοποιήσουμε και τις δύο καμπυλότητες, την μέση και την γκαουσιανή, για να βρούμε τα ιστογράμματά τους και να υπολογίσουμε την ανομοιομορφία με το παρακάτω τύπο:

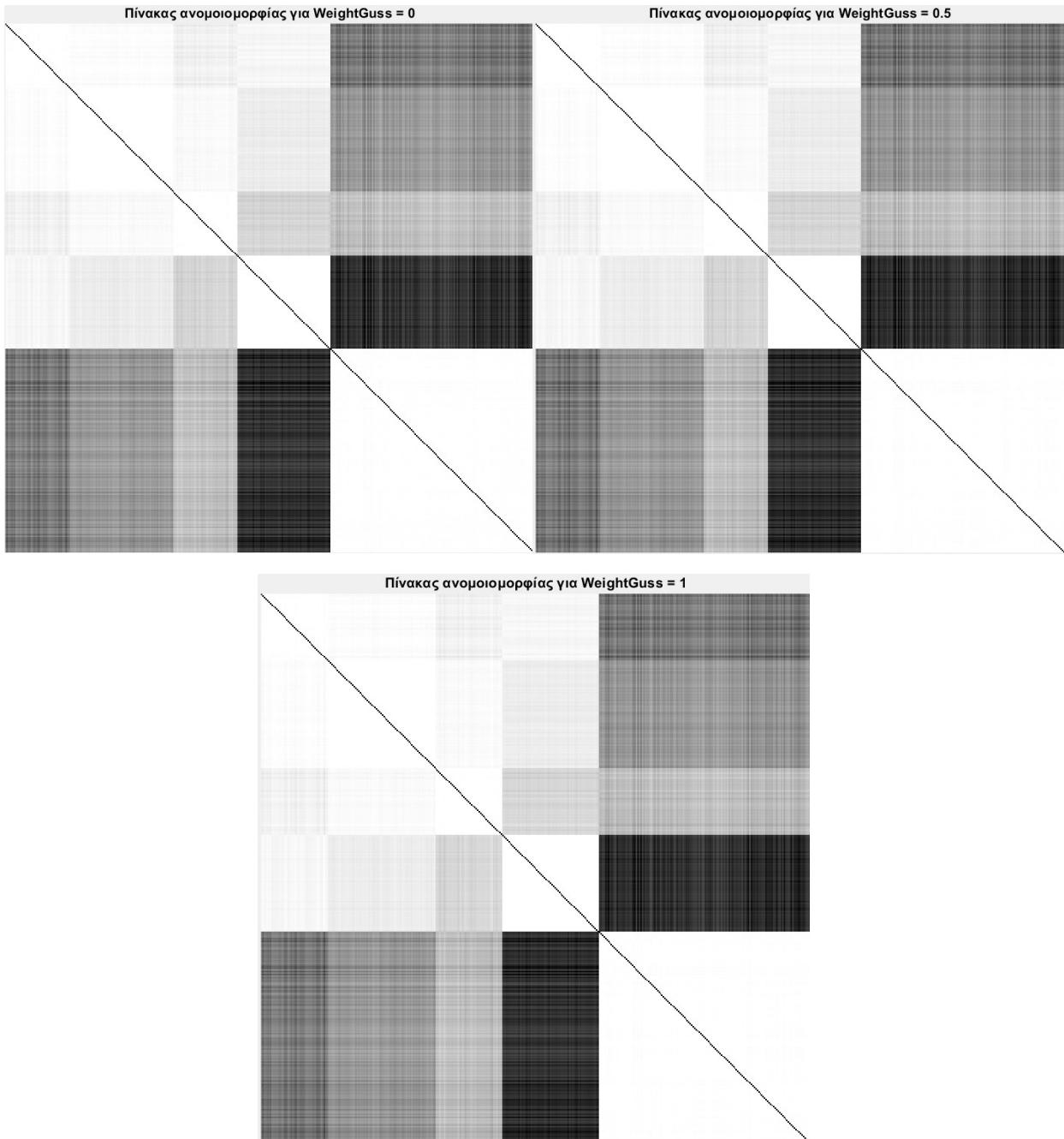
$$ErrorTotal(i,j) = WeightGauss * ErrorGauss(i,j) + (1 - WeightGauss) * ErrorMean(i,j) \quad (4.3)$$

Οπου το $ErrorTotal(i,j)$ το συνολικό ποσοστό ανομοιομορφίας, $ErrorGauss(i,j)$ και $ErrorMean(i,j)$ τα ποσοστά ανομοιομορφίας σύμφωνα με την γκαουσιανή και μέση καμπυλότητα αντίστοιχα και το

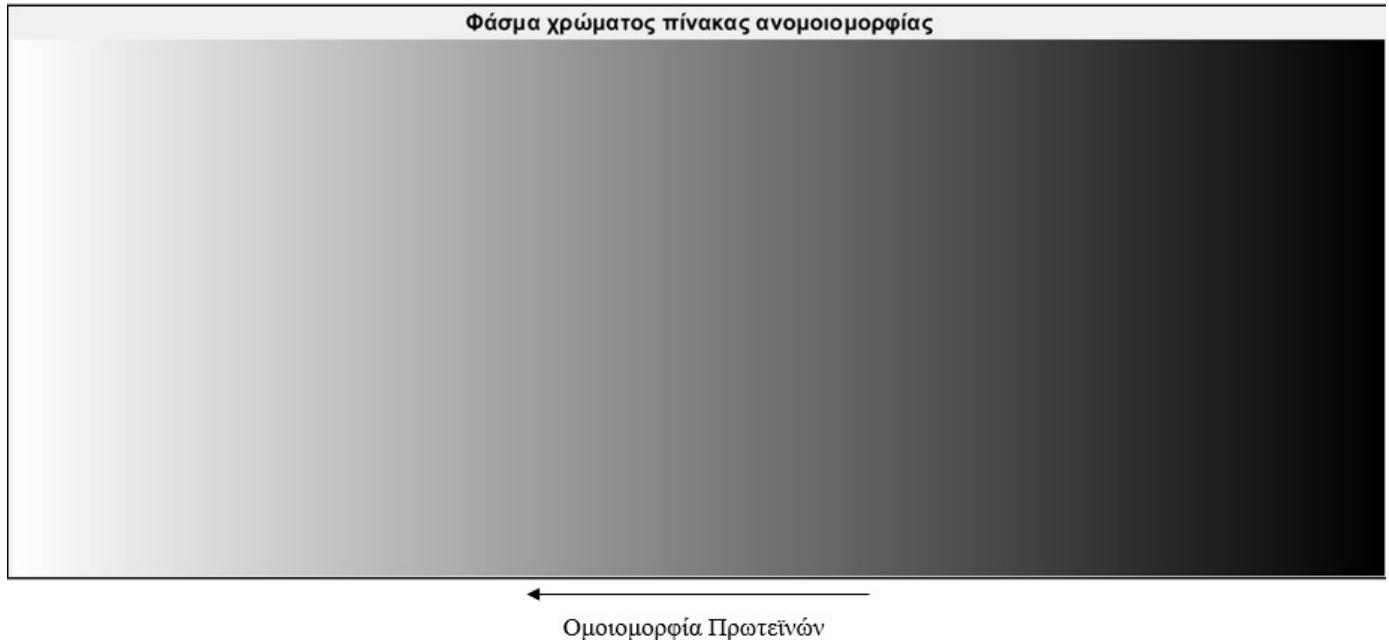
WeightGauss ένα βάρος για το πόσο θέλουμε να δώσουμε βάση στην γκαουσιανή καμπυλότητα και πόσο στην μέση, με διάστημα τιμών [0, 1]. Επίσης το πεδίο τιμών του ιστογράμματος θα χωριστεί σε 50 υπόδιαστήματα.

Έτσι μόλις υπολογιστούν οι καμπυλότητες και το ιστόγραμμα κάθε πρωτεΐνης, βρίσκουμε τον πίνακα ανομοιομορφίας *ErrorGauss* και *ErrorMean*, χωρίς κανονικοποίηση με την μεγαλύτερη τιμή και εφαρμόζουμε τον παραπάνω τύπο για να βγάλουμε τον συνολικό πίνακα ανομοιομορφίας *ErrorTotal*.

Παρακάτω φαίνεται ο πίνακας ανομοιομορφίας για *WeightGauss* = [0, 0.5, 1].



Φάσμα χρώματος πίνακας ανομοιομορφίας

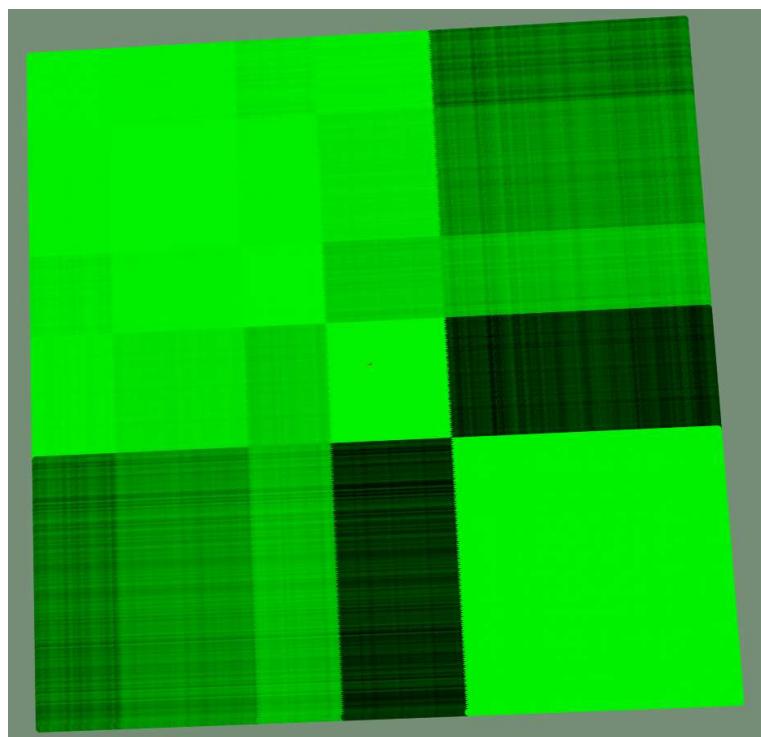


Όσο πιο λευκό είναι το εικονοστοιχείο τόσο πιο όμοια είναι η πρωτεΐνη με την αντίστοιχη της και όσο πιο μαύρο τόσο πιο ανομοιόμορφες είναι. Με μαύρο έχουν χρωματιστεί επίσης τα εικονοστοιχεία στα οποία η πρωτεΐνη συγκρίνεται με τον εαυτό της (διαγώνιος).

Παρότι που φαίνονται παρόμοιοι οι πίνακες, αν υπολογίσουμε το τετραγωνικό σφάλμα τους θα παρατηρήσουμε ότι έχουν διαφορές.

Τετραγωνικό σφάλμα	$WeightGauss = 0$	$WeightGauss = 0.5$	$WeightGauss = 1$
$WeightGauss = 0$	0	10.8163	13.6560
$WeightGauss = 0.5$	10.8163	0	2.3990
$WeightGauss = 1$	13.6560	2.3990	0

Μπορούμε να τυπώσουμε τον πίνακα και στο vvr.

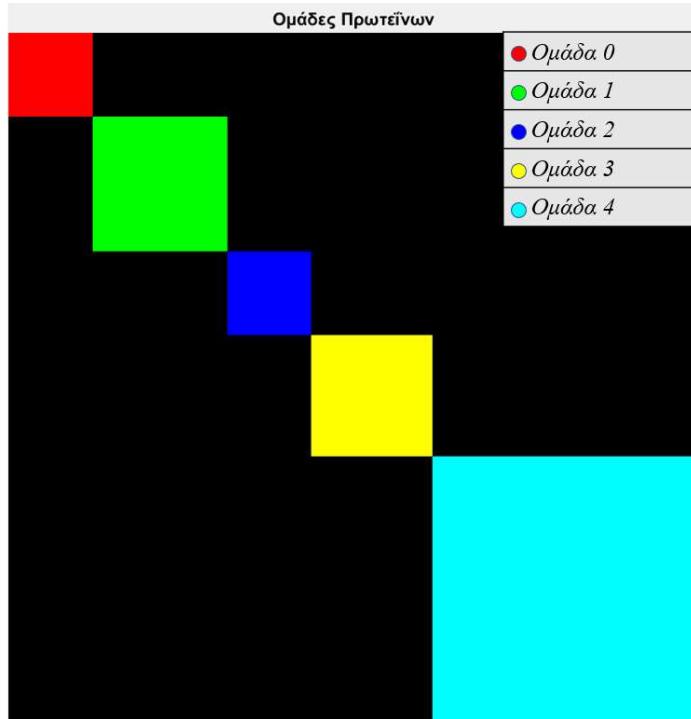


Ερώτημα 5 (Task 5)

Ο στόχος του ερωτήματος αυτού είναι να χωρίσουμε τις πρωτεΐνες σε πέντε ομάδες (0-4). Το πλήθος των πρωτεϊνών κάθε ομάδας στο σύνολο εκπαίδευσης είναι το εξής:

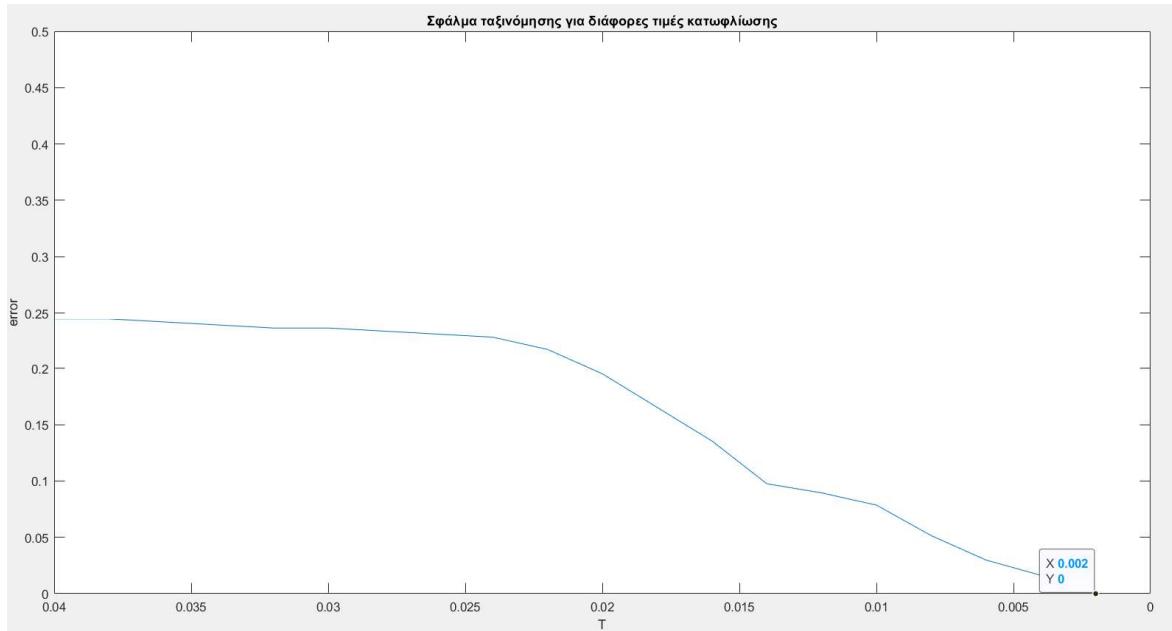
Oμάδα 0	Oμάδα 1	Oμάδα 2	Oμάδα 3	Oμάδα 4
45	72	45	65	142

Και αν το δούμε σε έναν πίνακα θα είναι:



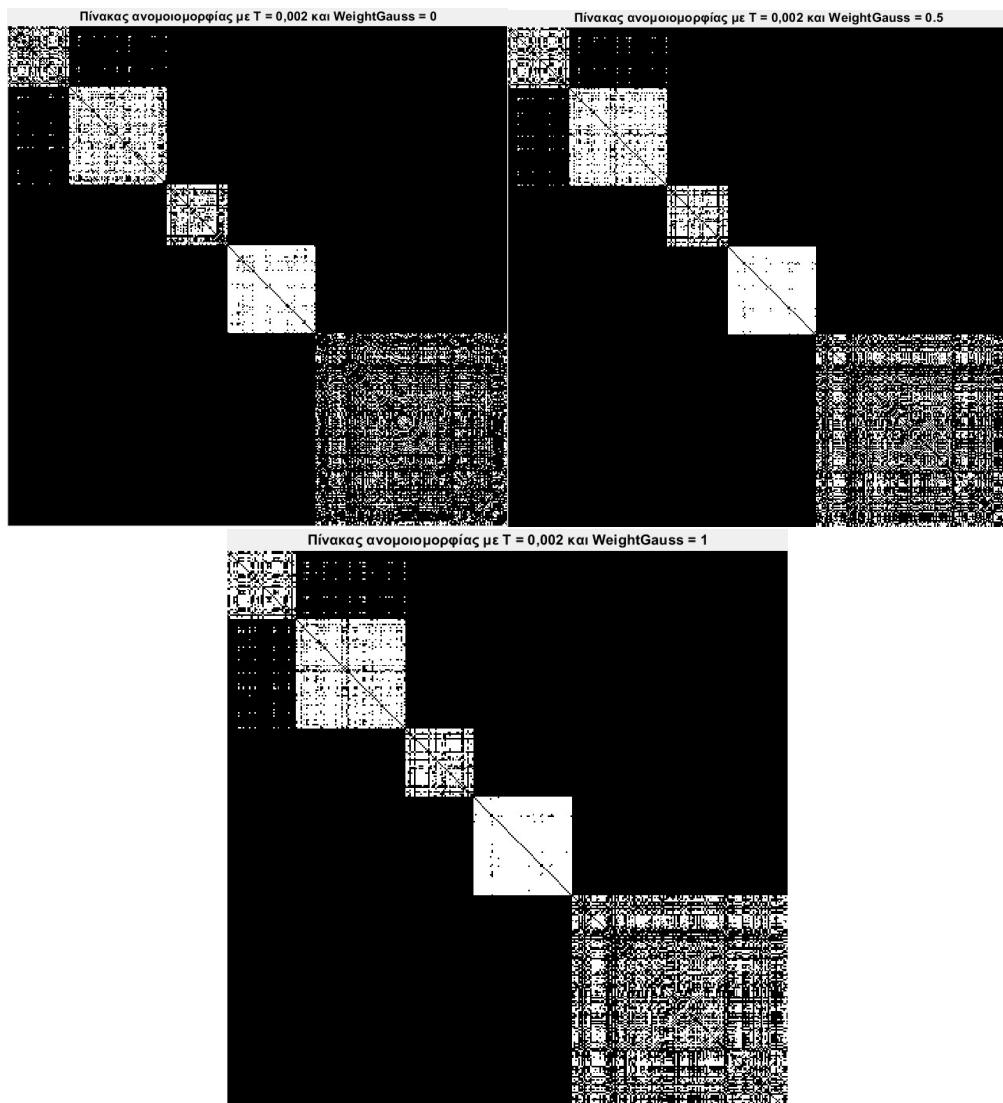
Η στοίχιση της κάθε πρωτεΐνης σε μία ομάδα θα γίνει με την πλειοψηφία των όμοιων πρωτεϊνών της αντίστοιχης ομάδας. Με λίγα λόγια θα κατασκευάσουμε έναν πίνακα ομοιομορφίας και θα ορίσουμε ένα κατώφλι. Αν το ποσοστό που αναφέρεται στον πίνακα ανομοιομορφίας είναι μικρότερο από αυτό θέτουμε 1, αλλιώς 0 στον νέο πίνακα. Το 1 δηλώνει ότι η πρωτεΐνη είναι όμοια με την αντίστοιχή της. Στην συνέχεια στον πίνακα αυτόν μετράμε σε κάθε πρωτεΐνη το πλήθος των αριθμών 1 που έχει η κάθε ομάδα και οποια ομάδα έχει τα περισσότερα, η πρωτεΐνη ανήκει σε αυτήν. Το αποθηκεύουμε σε ένα διάνυσμα όσο το μήκος τους, το οποίο θα χρειαστούμε για να εκτιμήσουμε το ποσοστό ανάκτησης.

Παίρνοντας τιμές κατωφλίου $T = [0.04 : -0.002 : 0.002]$, η απεικόνιση του σφάλματος ταξινόμησης θα είναι η εξής γραφική παράσταση (Για οποιαδήποτε τιμή του *WeightGauss* η μορφή είναι παρόμοια):



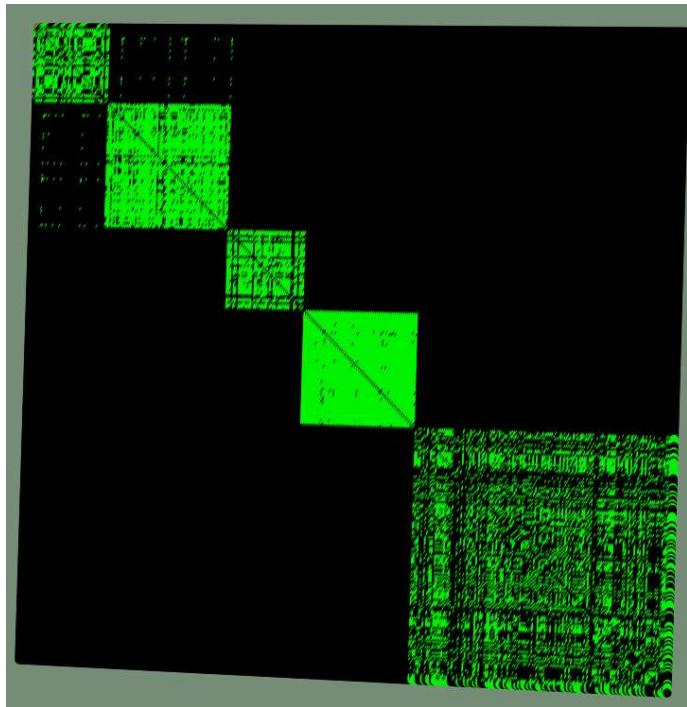
Παρατηρούμε ότι το σφάλμα ταξινόμησης για τιμή κατωφλίου $T = 0,002$ είναι μηδενικό. Οπότε οι πρωτεΐνες ταξινομούνται όπως μας έχουν δοθεί από τα αρχεία της άσκησης, όπως φαίνεται και στο παράθυρο εντολών.

Οι πίνακες ομοιότητας με τους αριθμούς 0 και 1 για κατώφλι 0,002 και $WeightGauss = [0, 0.5, 1]$ είναι οι εξής:



Παρόμοιοι με την παραπάνω εικόνα(Ομάδες Πρωτεϊνών).

Επίσης απεικονίζουμε τον πίνακα ομοιότητας και στο vvr.



Ερώτημα 6 (Task 6)

Σε αυτό το ερώτημα θα εκτιμήσουμε το ποσοστό ανάκτησης. Δηλαδή θα πάρουμε κάθε πρωτεΐνη από το σύνολο δοκιμής και θα ανακτήσουμε από το σύνολο εκπαίδευσης την πρωτεΐνη με τα πιο όμοια χαρακτηριστικά και θα ελέγχουμε αν ανήκουν στην ίδια ομάδα ή όχι.

Αρχικά για να λύσουμε το πρόβλημα θα πρέπει να υπολογίσουμε τα ιστογράμματα των πρωτεϊνών του συνόλου δοκιμής, για τις δύο καμπυλότητες γκαουστιανή και μέση. Μόλις το εκτελέσουμε αυτό, βρίσκουμε το ποσοστό ανομοιότητας της πρωτεΐνης δοκιμής με όλες τις πρωτεΐνες από την βάση εκπαίδευσης, όπως το υλοποιήσαμε στα παραπάνω ερωτήματα. Αυτές τις τιμές, για κάθε πρωτεΐνη δοκιμής τις αποθηκεύουμε σε έναν κανονικοποιημένο διάνυσμα μήκους 369 και βρίσκουμε την θέση της μικρότερης τιμής στον πίνακα. Η θέση αυτή είναι η ανακτώμενη πρωτεΐνη από το σύνολο εκπαίδευσης και αποθηκεύουμε σε ένα άλλο διάνυσμα μήκους 41 αυτήν την θέση.

Στην συνέχεια υπολογίζουμε το ποσοστό ανάκτησης. Σύμφωνα με το διάνυσμα με τις ανακτώμενες πρωτεΐνες, υπολογίζουμε το διάνυσμα που έχει για τιμές τον αριθμό της ομάδας που ανήκει η πρωτεΐνη σε σχέση με την ταξινόμηση που έγινε στο ερώτημα 5.

Τα ποσοστά ανάκτησης και η ταξινόμησης του συνόλου δοκιμών για κατώφλι $T = 0,002$ και $WeightGauss = [0, 0.5, 1]$ βρίσκονται παρακάτω:

```
Weight Gauss: 0
T: 0.002-> 2 0 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
retrieval:0.731707
```

```
Weight Gauss: 0.5
T: 0.002-> 0 0 0 0 0 2 2 2 1 2 2 1 2 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
retrieval:0.853659
```

```

Weight Gauss: 1
T: 0.002-> 0 0 0 0 0 1 1 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
retrieval:1

```

<i>WeightGauss</i>	<i>Retrieval</i>
0	0.7317
0,5	0,8537
1	1,0000

Παρατηρούμε πώς για τα διάφορα βάρη, παρότι οι πρωτεΐνες του συνόλου εκπαίδευσης ταξινομούνται στις σωστές ομάδες, για μικρές τιμές του *WeightGauss* το ποσοστό ανάκτησης είναι μικρό σε σχέση με αυτό που είναι ίσο με ένα. Άρα η καμπυλότητα gauss είναι το βέλτιστο χαρακτηριστικό για να ανακτήσουμε την σωστή ομάδα πρωτεΐνών στα συγκεκριμένα δεδομένα.

Ερώτημα 7 (Task 7)

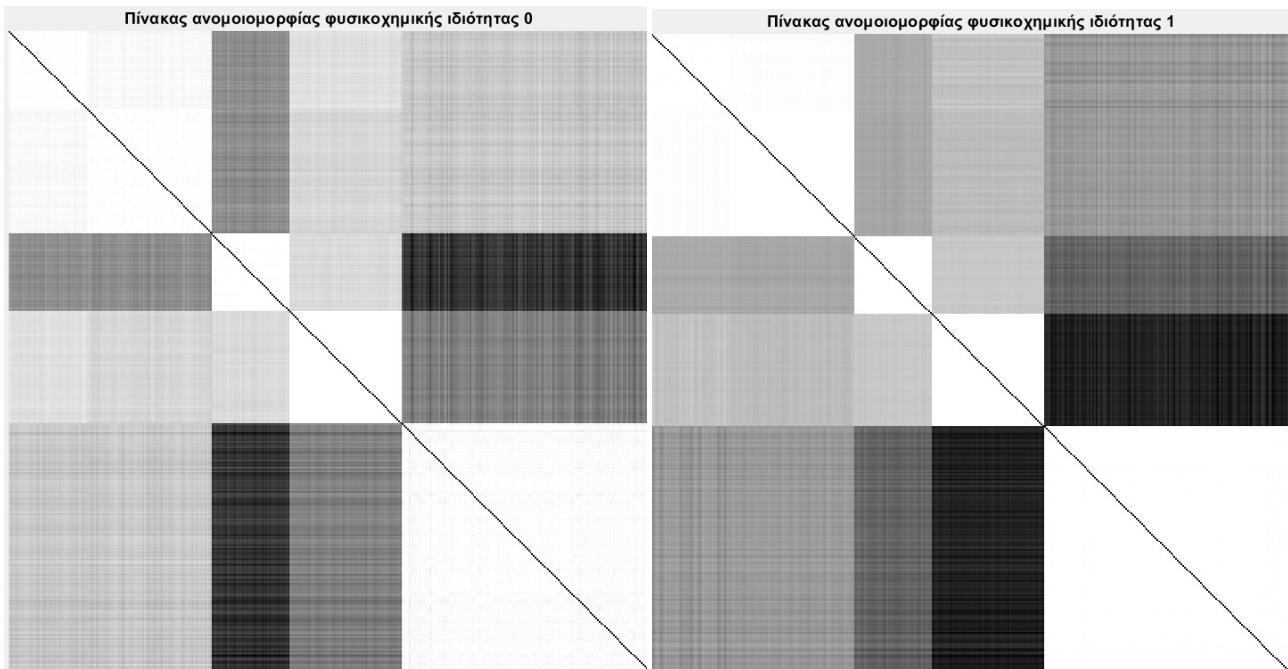
Στο ερώτημα αυτό θα ξανά υλοποιήσουμε τα ερωτήματα από το 4^ο μέχρι το 6^ο, όμως θα χρησιμοποιήσουμε μόνο τα φυσικοχημικά χαρακτηριστικά των πρωτεΐνων. Συγκεκριμένα θα βρούμε αρχικά, τον πίνακα ανομοιότητας για όλες τις ιδιότητες και συνδυασμό αυτών ο οποίος θα ακολουθεί τον παρακάτω τύπο:

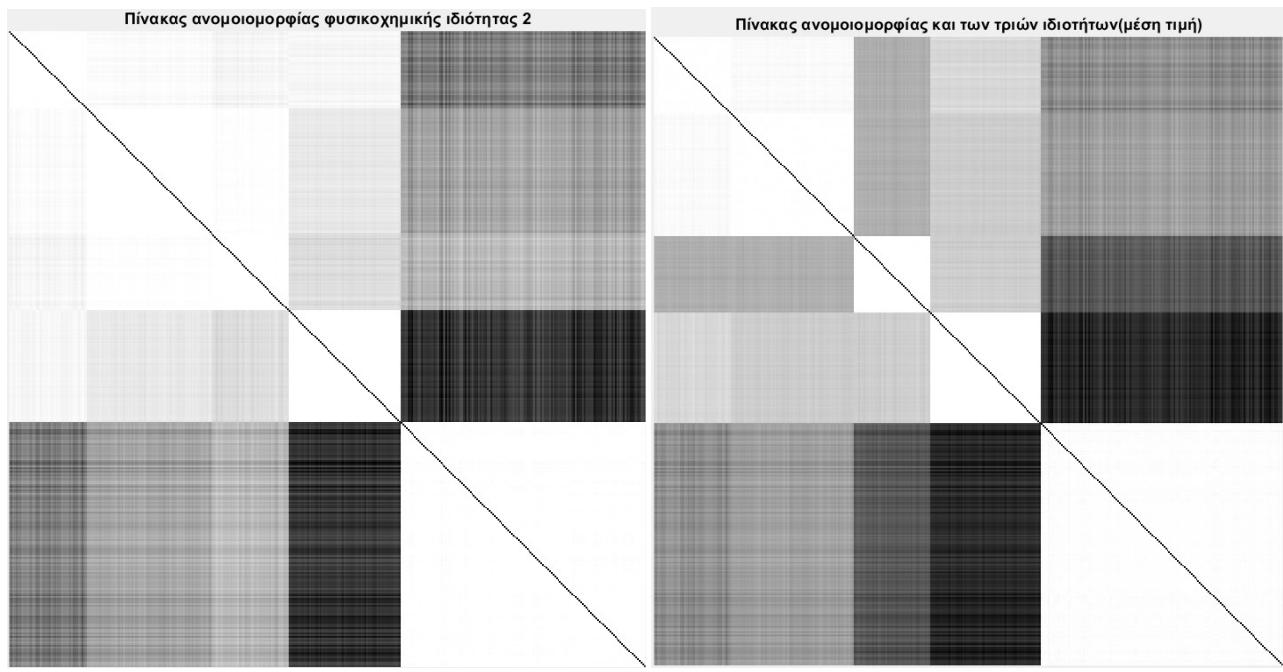
$$ErrorTotal = (ErrorProp0 + ErrorProp1 + ErrorProp2)/3$$

Στην συνέχεια θα υπολογίσουμε τον πίνακα ομοιότητας για ένα συγκεκριμένο κατώφλι T, θα ταξινομήσουμε τις πρωτεΐνες σε ομάδες και τέλος θα βρούμε το ποσοστό ανάκτησης.

Task 4

Υπολογίζουμε τους πίνακες ανομοιομορφίας για την ιδιότητα 0, 1, 2 και της μέσης τιμής τους, με τον ίδιο τρόπο όπως έχει εξηγηθεί στο ερώτημα 4 αλλά δεν χρησιμοποιούμε κάποιο βάρος. Οι πίνακες για τις τέσσερις περιπτώσεις είναι οι εξής(χρησιμοποιείται το ίδιο φάσμα χρώματος):

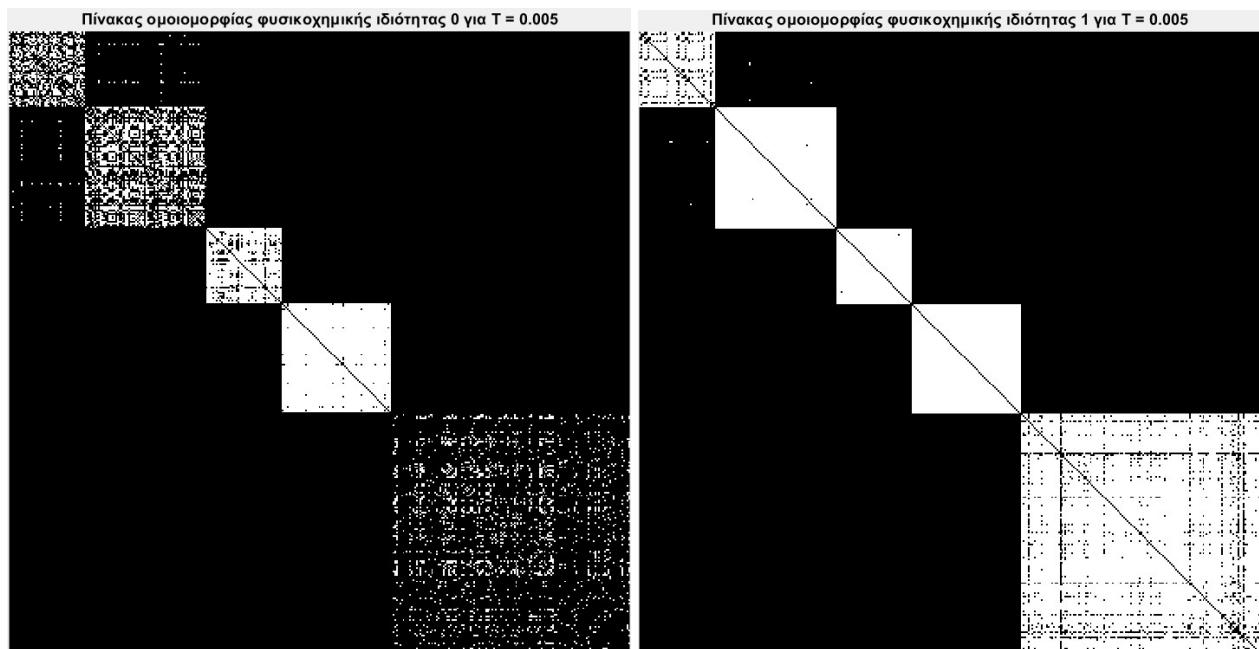




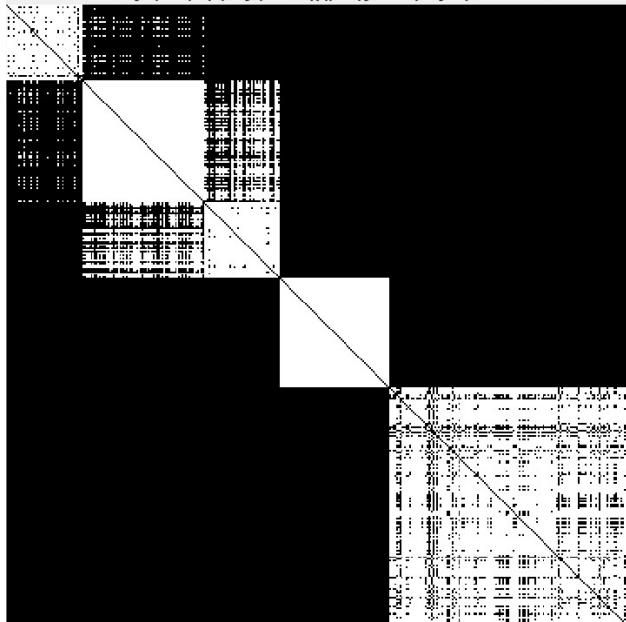
Παρατηρούμε σημαντικές διαφορές στους παραπάνω πίνακες σε σχέση με αυτούς στην καμπυλότητα.

Task 5

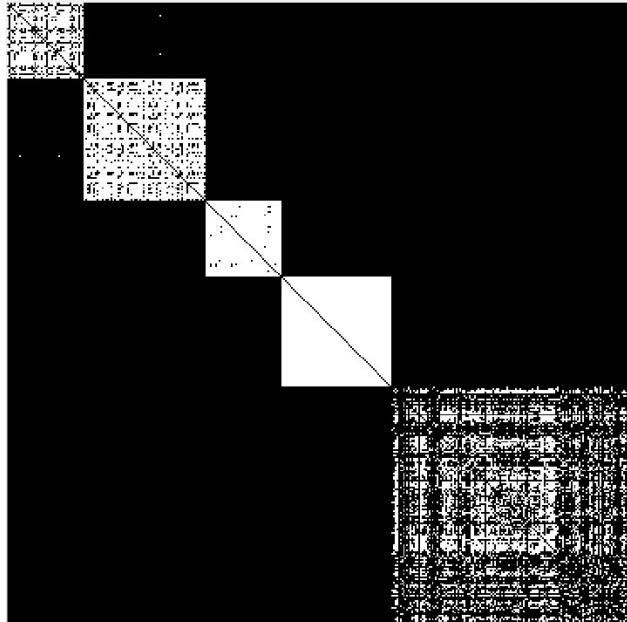
Από δοκιμές εκτιμάται ότι το κατώφλι $T = 0,005$ δίνει καλύτερα αποτελέσματα. Οι πίνακες ομοιότητας που προκύπτουν είναι οι εξής:



Πίνακας ομοιομορφίας φυσικοχημικής ιδιότητας 2 για $T = 0.005$



Πίνακας ομοιομορφίας και των τριών ιδιοτήτων για $T = 0.005$



Έτσι οι πρωτεΐνες ταξινομούνται όπως φαίνεται παρακάτω:

Prop 0->error classification: 0.00271004

classification:

Prop 1 -> error classification: 0

Proposed classification:

Prop 2->error classification: 0.0596206

Top 2 error classes

Τα σφάλματα που προκύπτουν από την ταξινόμηση είναι τα εξής;

<i>Φυσικοχημικές Ιδιότητες</i>	<i>Error Classification</i>
0	0.0027
1	0.0000
2	0.0596
<i>total</i>	0.0000

Παρατηρούμε ότι ταξινόμηση με μηδενικό σφάλμα γίνεται στην ιδιότητα 1 και στην μέση τιμή τους για το συγκεκριμένο κατώφλι.

Task 6

Τώρα θα αντιστοιχίσουμε κάθε πρωτεΐνη από το σύνολο δοκιμών με την πιο όμοια του πρωτεΐνη από το σύνολο εκπαίδευσης, θα το τοποθετήσουμε στην αντίστοιχη ομάδα και θα εκτιμήσουμε το ποσοστό ανάκτησης όπως έγινε στο ερώτημα 6. Οι ταξινομήσεις των πρωτεΐνών του συνόλου δοκιμής σε ομάδες με τα αντίστοιχα σφάλματα είναι:

Τα ποσοστά ανάκτησης για τις δύο περιπτώσεις με κατώφλι $T = 0,005$ είναι;

<i>Φυσικοχημικές Ιδιότητες</i>	<i>Retrieval</i>
<i>0</i>	<i>1.0000</i>
<i>1</i>	<i>1.0000</i>
<i>2</i>	<i>0.9756</i>
<i>total</i>	<i>1.0000</i>

Η ανάκτηση είναι μηδενική στις ιδιότητες 0, 1 και μέσης τιμής. Μόνο στην ιδιότητα 2 έχουμε μικρή απόκλιση.

Παράρτημα(Ανάλυση Κώδικα)

Header File

Στο αρχείο SceneMesh3D.h περιέχονται οι βιβλιοθήκες, οι δομές και οι κλάσεις που χρησιμοποιήθηκαν για την επίλυση της άσκησης. Οι κυριότερες από τις βιβλιοθήκες ήταν οι συναρτήσεις της Engine για πράξεις πινάκων και η VVRScene για απεικόνιση των 3d αντικειμένων. Στην παρακάτω εικόνα φαίνονται οι υπόλοιπες βιβλιοθήκες.

```
1  #include <VVRScene/canvas.h>
2  #include <VVRScene/mesh.h>
3  #include <VVRScene/settings.h>
4  #include <VVRScene/utils.h>
5  #include <MathGeolib.h>
6  #include <iostream>
7  #include <fstream>
8  #include <cstring>
9  #include <string>
10 #include <set>
11 #include <iomanip>
12 #include <algorithm>
13 #include <iterator>
14 #include "symmetriceigensolver3x3.h"
15 #include "canvas.h"
16 #include "Eigen/Dense"
17 #include "Eigen/Sparse"
18 #include "Eigen/SparseQR"
19 #include "Eigen/Eigenvalues"
20 #include "Eigen/SparseCholesky"
21 #include "math.h"
22
```

Επίσης οι δομές και οι κλάσεις είναι οι εξής:

```
30  // Task 2
31  [--- Γείτονες του κόμβου ---]
32  struct vertex_neighbours { ... };
33  // Task 3
34  [--- Δυαδικό διάνυσμα ---]
35  struct dual_vertice { ... };
36  [--- Κόμβος σωρού ---]
37  struct heap_node { ... };
38  [--- Κλάση του σωρού ---]
39  class Heap { ... };
40  [--- Κύρια κλάση ---]
41  class Mesh3DSceen { ... };
42
```

Οι δομές *vertex_neighbours* και *dual_vertice* αναφέρθηκαν αναλυτικά στα ερωτήματα. Η δομή *heap_node* είναι ο κόμβος του σωρού που χρησιμοποιείται στις συναρτήσεις της κλάσης *Heap*, ώστε να λειτουργήσει σωστά ο αλγόριθμος Dijkstra. Περιέχει την θέση του *dual_vertice* στον Γράφο και την γεωδισιακή απόσταση του.

```
32  struct vertex_neighbours {
33      std::vector<int> vertex;
34      std::vector<int> triangles;
35      std::vector<math::vec> N_surface;
36  };
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51  struct heap_node {
52      int index_vec;
53      double geodesic;
54  };
55
```

```

39 struct dual_vertex {
40     math::vec dual_v;
41     int index_vec;
42     int count_neigh;
43     int neigh_dual_vec[3] = { -1, -1, -1 };
44     double distances[3] = { -1, -1, -1 };
45     double area;
46     double geodesic;
47     double p = 0; // protrusion degree
48     bool read;
49 };
56 class Heap {
57 public:
58     /* Αρχικοποίηση του σωρού*/
59     void HEAP_init(int i);
60     /* Αφίρεση ρίζας*/
61     heap_node HEAP_pop();
62     /* Ανακατάταξη κόμβων όταν αλλάζει τιμή ένας από αυτούς*/
63     void change_data(int i, double x);
64     /* Εκτύπωση σωρού*/
65     void HEAP_print();
66     /* Αλλαγή τιμή των κόμβων*/
67     void swap_heap_node(heap_node& dv1, heap_node& dv2);
68     /* Μήκος σωρού*/
69     int plithos;
70     /* Διάνυσμα σωρού*/
71     std::vector<heap_node> heap_D; // heap
72 };

```

Η κλάση Mesh3DScene περιέχει τις συναρτήσεις και τις μεταβλητές που ορίστηκαν για κάθε ερώτημα. Ορισμένες από τις συναρτήσεις αυτές θα αναλυθούν στο SceneMesh3D.cpp. Οι μεταβλητές που ορίσαμε για κάθε ερώτημα είναι οι εξής:

//--- Task 1 ---//

```

216     //---Task 1---//
217     // Prop 0
218     std::vector<double> m_properties_col_0; // Διάνυσμα φυσικοχημικής ιδιότητας 0
219     std::vector<double> median_value_col_0; // Διάνυσμα με τις τιμές του κάθε τριγώνου
220     std::vector<vvr::Colour> colour_face_0; // Διάνυσμα με τα χρώματα του κάθε τριγώνου
221     double max_value_col_0; // Μέγιστη τιμή της φυσικοχημικής ιδιότητας 0
222     double min_value_col_0; // Ελάχιστη τιμή της φυσικοχημικής ιδιότητας 0
223     // Prop 1
224     std::vector<double> m_properties_col_1; // Διάνυσμα φυσικοχημικής ιδιότητας 1
225     std::vector<double> median_value_col_1; // Διάνυσμα με τις τιμές του κάθε τριγώνου
226     std::vector<vvr::Colour> colour_face_1; // Διάνυσμα με τα χρώματα του κάθε τριγώνου
227     double max_value_col_1; // Μέγιστη τιμή της φυσικοχημικής ιδιότητας 1
228     double min_value_col_1; // Ελάχιστη τιμή της φυσικοχημικής ιδιότητας 1
229     // Prop 2
230     std::vector<double> m_properties_col_2; // Διάνυσμα φυσικοχημικής ιδιότητας 2
231     std::vector<double> median_value_col_2; // Διάνυσμα με τις τιμές του κάθε τριγώνου
232     std::vector<vvr::Colour> colour_face_2; // Διάνυσμα με τα χρώματα του κάθε τριγώνου
233     double max_value_col_2; // Μέγιστη τιμή της φυσικοχημικής ιδιότητας 2
234     double min_value_col_2; // Ελάχιστη τιμή της φυσικοχημικής ιδιότητας 2
235     // Αντικείμενα με τις ιδιότητες της πρωτεΐνης
236     vvr::Mesh m_model_col_0, m_model_col_1, m_model_col_2;
237

```

//--- Task 2 ---//

```

238     //---Task 2---//
239     std::vector<vertex_neighbours> m_NeighboursList; // Διάνυσμα που περιέχει τους γειτόνους του κάθε διανύσματος
240     // Gauss
241     std::vector<double> K_gauss_curve; // Διάνυσμα καμπυλότητας gauss
242     double max_value_K; // Μέγιστη τιμή καμπυλότητα gauss
243     double min_value_K; // Ελάχιστη τιμή καμπυλότητα gauss
244     std::vector<double> median_value_K; // Διάνυσμα με τις τιμές του κάθε τριγώνου
245     std::vector<vvr::Colour> colour_K; // Διάνυσμα με τα χρώματα του κάθε τριγώνου
246     // Mean
247     std::vector<double> H_mean_curve; // Διάνυσμα καμπυλότητας mean
248     double max_value_H; // Μέγιστη τιμή καμπυλότητα mean
249     double min_value_H; // Ελάχιστη τιμή καμπυλότητα mean
250     std::vector<double> median_value_H; // Διάνυσμα με τις τιμές του κάθε τριγώνου
251     std::vector<vvr::Colour> colour_H; // Διάνυσμα με τα χρώματα του κάθε τριγώνου
252     // Αντικείμενα για αναπαράσταση
253     vvr::Mesh m_model_gauss, m_model_mean;
254

```

//--- Task 3 ---//

```
255 //---Task 3---//
256 std::vector< dual_vertice> Graph; // Διάνυσμα με τα δυαδικά διανύσματα
257 Heap heap_Dij; // Σωρός για τον αλγόριθμο Dijkstra
258 double max_p; // Μέγιστη τιμή του βαθμού προεξοχής
259 std::vector<int> is_projection; // Διάνυσμα αν είναι σημείο προεξοχής = 1
260
```

//--- Task 4i ---//

```
261 //---Task 4---//
262 //-- Ερώτημα i --//
263 // - Αρμαντίδο - //
264 vvr::Mesh m_model_1;
265 std::vector<vertex_neighbours> m_NeighList_1;
266 std::vector<double> K_1;
267 // - Μονόκαιρος - //
268 vvr::Mesh m_model_2;
269 std::vector<vertex_neighbours> m_NeighList_2;
270 std::vector<double> K_2;
271 // - Κουνέλι - //
272 vvr::Mesh m_model_3;
273 std::vector<vertex_neighbours> m_NeighList_3;
274 std::vector<double> K_3;
275 // Μέγιστη και ελάχιστη καμπυλότητα
276 double max_Gauss;
277 double min_Gauss;
278 double max_Mean;
279 double min_Mean;
280 // Ιστογράμματα
281 Eigen::VectorXf hist_obj_1;
282 Eigen::VectorXf hist_obj_2;
283 Eigen::VectorXf hist_obj_3;
284 Eigen::MatrixXf histogramm_Obj; // Πίνακας με τα τρία ιστογράμματα
285 // Πίνακας ανομοιομορφίας για τα τρία αντικείμενα
286 Eigen::MatrixXf dissimilarity_matrix;
```

//--- Task 4ii ---//

```
287 //-- Ερώτημα ii --//
288 vvr::Mesh m_model_protein; // Μοντέλο πρωτεΐνης
289 std::vector<vertex_neighbours> m_NeighList_protein;
290 std::vector<double> K_protein;
291 std::vector<double> H_protein;
292 // Ιστογράμματα
293 Eigen::VectorXf hist_obj_protein; // Βοηθητικό διάνυσμα
294 Eigen::MatrixXf histogramms_Proteins_Gauss; // Τα ιστογράμματα πρωτεΐνων εκπαίδευσης καμπυλότητας gauss
295 Eigen::MatrixXf histogramms_Proteins_Mean; // Τα ιστογράμματα πρωτεΐνων εκπαίδευσης καμπυλότητας mean
296 // Πίνακας ανομοιομορφίας σύμφωνα με τις καμπυλότητες
297 Eigen::MatrixXf dissimilarity_matrix_proteins;
298
```

//--- Task 5 ---//

```
299 //---Task 5---//
300 // Πίνακας ομοιομορφίας σύμφωνα με τις καμπυλότητες
301 Eigen::MatrixXf dissimilarity_matrix_proteins_bool;
302
```

```
///--- Task 6 ---//
```

```
303 //---Task 6---//
304 Eigen::VectorXf Vec_similar_protein; // Διάνυσμα με τις ομάδες που βρίσκονται οι πρωτεΐνες δοκιμής
305 Eigen::VectorXf similar_test_data; // Βοηθητικό διάνυσμα
306 // Βοηθητικά διάνυσματα για την σύγκριση του συνόλου δοκιμής με το σύνολο εκπαίδευσης
307 Eigen::VectorXf hist_obj_protein_dataset_gauss;
308 Eigen::VectorXf hist_obj_protein_dataset_mean;
309 Eigen::VectorXf hist_obj_protein_testset_gauss;
310 Eigen::VectorXf hist_obj_protein_testset_mean;
311
```

```
///--- Task 7 ---//
```

```
312 //---Task 7---//
313 // Διάνυσματα με τις φυσικοχημικές ιδιότητες της πρωτεΐνης
314 std::vector<double> protein_properties_col_0;
315 std::vector<double> protein_properties_col_1;
316 std::vector<double> protein_properties_col_2;
317 // Μέγιστρες και ελάχιστες τιμές
318 double max_col_0;
319 double min_col_0;
320 double max_col_1;
321 double min_col_1;
322 double max_col_2;
323 double min_col_2;
324 // Ιστογράμματα
325 Eigen::VectorXf hist_properties_col_0;
326 Eigen::VectorXf hist_properties_col_1;
327 Eigen::VectorXf hist_properties_col_2;
328 // Πίνακες ιστογραμμάτων
329 Eigen::MatrixXf arr_hist_properties_col_0;
330 Eigen::MatrixXf arr_hist_properties_col_1;
331 Eigen::MatrixXf arr_hist_properties_col_2;
332 // task 4 //
333 // Πίνακες ανομοιομορφίας
334 Eigen::MatrixXf diss_properties_col_0;
335 Eigen::MatrixXf diss_properties_col_1;
336 Eigen::MatrixXf diss_properties_col_2;
337 Eigen::MatrixXf diss_properties_total;
338 // task 5
339 // Πίνακας ομοιομορφίας
340 Eigen::MatrixXf diss_matrix_prop_bool;
341 // Ταξινόμηση των πρωτεΐνων σε ομάδες ανάλογα τις φυσικοχημικές ιδιότητες
342 Eigen::VectorXf c_prop_0;
343 Eigen::VectorXf c_prop_1;
344 Eigen::VectorXf c_prop_2;
345 Eigen::VectorXf c_prop_total;
346 // task 6
347 // Διάνυσμα με τις ομάδες που βρίσκονται οι πρωτεΐνες δοκιμής
348 Eigen::VectorXf Vec_similar_protein_col_0;
349 Eigen::VectorXf Vec_similar_protein_col_1;
350 Eigen::VectorXf Vec_similar_protein_col_2;
351 Eigen::VectorXf Vec_similar_protein_col_total;
352 Eigen::VectorXf similar_test_data_prop; // Βοηθητικό διάνυσμα
```

Source File

Ο κώδικας λειτουργεί κυρίως στα ερωτήματα 4, 5 και 6 με βοήθεια αρχείων για την απεικόνιση γραφικών παραστάσεων και τα ερωτήματα να τρέχουν ξεχωριστά και γρήγορα. Τα ερωτήματα εκτελούνται στην συνάρτηση *Tasks()*.

Οι σταθερές που έχουμε στην αρχή του κώδικα είναι οι εξής:

```
3     // 0: Προεπεξεργασία
4     // 1: Ερώτημα 1
5     // 2: Ερώτημα 2
6     // 3: Ερώτημα 3 ή 31 (Διαφορά στο draw())
7     // 41: Ερώτημα 4i
8     // 42: Ερώτημα 4ii
9     // 5: Ερώτημα 5
10    // 6: Ερώτημα 6
11    // 7: Ερώτημα 7
12    #define TASK 0
13    #define ON_FILE 0      // Αποθήκευση δεδομένων σε αρχεία
14    #define HIST_SIZE 50   // Αριθμός υποδιαστημάτων
15    #define WEIGHT_GAUSS 1 // Βάρος της καμπυλότητας gauss
16    #define T 0.002        // Κατώφλι όμοιων πρωτεΐνων
17
```

Το TASK δηλώνει το ερώτημα που επιθυμούμε να τρέξουμε, το ON_FILE είναι να μεταφέρουμε ορισμένους υπολογισμούς που παίρνουν χρόνο σε αρχεία .txt αν ισούται με το 1, το HIST_SIZE είναι ο αριθμός των υπό-διαστημάτων, το WEIGHT_GAUSS είναι το βάρος που έχουμε αναφερθεί (Ερώτημα 4ii) και το T είναι το κατώφλι για την δημιουργία του πίνακα ομοιομορφίας.

Επίσης ορίζουμε και την κλάση Mesh3DScene:

```
54     Mesh3DScene::Mesh3DScene()
55     {
56         // Load settings.
57         vvr::Shape::DEF_LINE_WIDTH = 4;
58         vvr::Shape::DEF_POINT_SIZE = 10;
59         m_perspective_proj = true;
60         m_bg_col = Colour("768E77");
61         m_obj_col = Colour("454545");
62         m_model_original = vvr::Mesh(objFileSurf);
63         reset();
64     }
65
```

//--- Task 0 ---//

Θέτοντας τα εξής στις σταθερές μεταβλητές:

```
12 #define TASK 0
13 #define ON_FILE 1      // Αποθήκευση δεδομένων σε αρχεία
14 #define HIST_SIZE 50   // Αριθμός υποδιαστημάτων
15 #define WEIGHT_GAUSS 1 // Βάρος της καμπυλότητας gauss
16 #define T 0.002         // Κατώφλι όμοιων πρωτεΐνων
17
```

Γίνεται μία προ-επεξεργασία για τα ερωτήματα 4, 5, 6 υπολογίζοντας τις καμπυλότητες gauss και mean για τις πρωτεΐνες του συνόλου εκπαίδευσης και δοκιμής. Επίσης βρίσκουμε τις μέγιστες και ελάχιστες τιμές των καμπυλοτήτων και των φυσικοχημικών ιδιοτήτων που θα χρειαστούν για την κανονικοποίηση των δεδομένων στο ιστόγραμμα. Έχουν οριστεί οι κατευθύνσεις που βρίσκονται τα δεδομένα αυτά:

```
// Κατεύθυνση αρχείων
[---Task 0---]
const string objDirDataset = getbasePath() + "resources/dataset/surfaces/";
const string objDirTestset = getbasePath() + "resources/test_set/surfaces/";
const string objDirPropDataset = getbasePath() + "resources/dataset/properties/";
const string objDirPropTestset= getbasePath() + "resources/test_set/properties/";
```

Η αποθήκευση των δεδομένων γίνεται στις παρακάτω κατευθύνσεις που βρίσκονται μέσα στις συναρτήσεις:

Καμπυλότητα Gauss σύνολο εκπαίδευσης:

```
string op_K = basePath() + "resources/curvature/dataset/Gauss/" + to_string(i) + ".txt";
```

Καμπυλότητα Mean σύνολο εκπαίδευσης:

```
string op_H = basePath() + "resources/curvature/dataset/Mean/" + to_string(i) + ".txt";
```

Αποθήκευση Μέγιστης και Ελάχιστης τιμής σύνολο εκπαίδευσης:

```
string op_Km = basePath() + "resources/curvature/dataset/Gauss/min_max.txt";
string op_Hm = basePath() + "resources/curvature/dataset/Mean/min_max.txt";
```

Καμπυλότητα Gauss σύνολο δοκιμής:

```
string op_K = basePath() + "resources/curvature/test_set/Gauss/" + to_string(i) + ".txt"
```

Καμπυλότητα Mean σύνολο δοκιμής:

```
string op_H = basePath() + "resources/curvature/test_set/Mean/" + to_string(i) + ".txt";
```

Αποθήκευση Μέγιστης και Ελάχιστης τιμής σύνολο δοκιμής:

```
string op_Km = basePath() + "resources/curvature/test_set/Gauss/min_max.txt";
string op_Hm = basePath() + "resources/curvature/test_set/Mean/min_max.txt";
```

Αποθήκευση Μέγιστης και Ελάχιστης τιμής σύνολο εκπαίδευσης(ιδιοτήτων):

```
string op0 = basePath() + "resources/properties/dataset/prop_0_min_max.txt";
string op1 = basePath() + "resources/properties/dataset/prop_1_min_max.txt";
string op2 = basePath() + "resources/properties/dataset/prop_2_min_max.txt";
```

Αποθήκευση Μέγιστης και Ελάχιστης τιμής σύνολο δοκιμής(ιδιοτήτων):

```

string op0 = getBasePath() + "resources/properties/test_set/prop_0_min_max_test.txt";
string op1 = getBasePath() + "resources/properties/test_set/prop_1_min_max_test.txt";
string op2 = getBasePath() + "resources/properties/test_set/prop_2_min_max_test.txt";

```

Οι συναρτήσεις που χρησιμοποιούνται είναι οι εξής:

```

479  /* Υπολογίζει και αποθηκεύει τις καμπυλότητες σε αρχεία .txt του συνόλου εκπαίδευσης*/
480  #void Mesh3DScene::load_dataset_curves(){...}
558  /* Υπολογίζει και αποθηκεύει τις καμπυλότητες σε αρχεία .txt του συνόλου δοκιμής*/
559  #void Mesh3DScene::load_test_set_curves(){...}
637  /* Ξεχωρίζει και αποθηκεύει τις φυσικοχημικές ιδιότητες σε αρχεία .txt του συνόλου εκπαίδευσης*/
638  #void Mesh3DScene::load_physiological_properties_dataset(){...}
709  /* Ξεχωρίζει και αποθηκεύει τις φυσικοχημικές ιδιότητες σε αρχεία .txt του συνόλου δοκιμής*/
710  #void Mesh3DScene::load_physiological_properties_test_set(){...}

```

Ακόμα κάτω από αυτές υπάρχουν γενικές συναρτήσεις, όπως εκτύπωση διανυσμάτων και πινάκων, εύρεση της θέσης και της ελάχιστης τιμής σε ένα διάνυσμα, κανονικοποίηση διανύσματος και συνάρτηση που ορίζει ένα διάνυσμα με τις σωστές ταξινομήσεις των ομάδων των πρωτεΐνων.

```

781  #vector<double> Mesh3DScene::find_max_min_value(vector<double> VecList){...}
795  /* Εκπύπωση διανύσματος VectorXf*/
796  #void Mesh3DScene::print_VectorXf(Eigen::VectorXf& v){...}
803  /* Εκπύπωση διανύσματος MatrixXf*/
804  #void Mesh3DScene::print_MatrixXf(Eigen::MatrixXf& arr){...}
814  /* Υπολογισμός του σφάλματος δύο διανυσμάτων*/
815  #float Mesh3DScene::calc_error(Eigen::VectorXf& v1, Eigen::VectorXf& v2){...}
823  /* Εύρεση της θέσης της μέγιστης τιμής*/
824  #int Mesh3DScene::max_index(Eigen::VectorXf v){...}
835  /* Εύρεση της θέσης της μικρότερης τιμής*/
836  #int Mesh3DScene::min_index(Eigen::VectorXf v){...}
847  /* Ορίζουμε τα διανύσματα ομάδων με τις σωστές ταξινομήσεις πρωτεΐνων*/
848  #Eigen::VectorXf Mesh3DScene::build_c_correct(int teams[5], int size_c){...}
860  /* Κανονικοποίηση διανύσματος VectorXf*/
861  #void Mesh3DScene::normalize_VectorXf(Eigen::VectorXf& v){...}
867

```

//--- Task 1 ---//

```
12  #define ...  
13  #define ON_FILE 0          // Αποθήκευση δεδομένων σε αρχεία  
14  #define HIST_SIZE 50       // Αριθμός υποδιαστημάτων  
15  #define WEIGHT_GAUSS 1     // Βάρος της καμπυλότητας gauss  
16  #define T 0.002            // Κατώφλι όμοιων πρωτεΐνων  
17
```

Στο ερώτημα 1 είναι η απεικόνιση των φυσικοχημικών ιδιοτήτων των πρωτεΐνων. Οι κατευθύνσεις των αρχείων είναι:

```
29  //---Task 1---//  
30  const string obj = "0"; // Αριθμός Πρωτεΐνης  
31  const string Set = "dataset"; // Σύνολο  
32  // Κατεύθυνση της πρωτεΐνης  
33  const string objDirSurfaces = getbasePath() + "resources/" + Set + "/surfaces/";  
34  const string objFileSurf = objDirSurfaces + obj + ".obj";  
35  // Κατεύθυνση των ιδιοτήτων  
36  const string objDirProperties = getbasePath() + "resources/" + Set + "/properties/";  
37  const string objFileProp = objDirProperties + obj + ".txt";
```

Στο *obj* μπαίνει ο αριθμός της πρωτεΐνης που επιθυμούμε να απεικονίσουμε και στο *Set* το σύνολο στο οποίο ανήκει η πρωτεΐνη(*dataset* ή *test_set*). Αν ορίσουμε την πρωτεΐνη τότε γίνεται η διαδικασία που αναφέρεται στο ερώτημα 1. Επίσης αν θέλουμε να ελέγξουμε ένα αντικείμενο, θέτουμε το όνομα του αντικειμένου στο *obj* και στο *Set* το αρχείο που ανήκει (*obj*).

Οι συναρτήσεις που χρησιμοποιούμε είναι:

```
871 //---Task 1---//  
872 /* Αποθήκευση σε διανύσματα τις φυσικοχημικές ιδιότητες των πρωτεΐνων*/  
873 void Mesh3DScene::open_physicochemical_properties(string file, std::vector<double>& m_properties_col_0, std::vector<double>& m_properties_col_1, std::vector<double>& m_properties_col_2) { ... }  
896 /* Εύρεση μέσης τιμής και τοποθέτηση στο αντίστοιχο τρίγωνο*/  
897 void Mesh3DScene::find_median_value_triangle(std::vector<double>& properties_col, std::vector<double>& median_value_vector) { ... }  
924 /* Συναρτήσεις τοποθέτησης χρώματος για τις τρεις ιδιότητες*/  
925 void Mesh3DScene::find_colour_face_0(std::vector<double>& median_value_vector, std::vector<vr::Colour>& colour_face) { ... }  
939 void Mesh3DScene::find_colour_face_1(std::vector<double>& median_value_vector, std::vector<vr::Colour>& colour_face) { ... }  
953 void Mesh3DScene::find_colour_face_2(std::vector<double>& median_value_vector, std::vector<vr::Colour>& colour_face) { ... }  
967
```

Η συνάρτηση *open_physicochemical_properties(string file, std::vector<double>& m_properties_col_0, std::vector<double>& m_properties_col_1, std::vector<double>& m_properties_col_2)*, ανοίγει τα αρχεία με τις φυσικοχημικές ιδιότητες των πρωτεΐνων και τις αποθηκεύει ξεχωριστά σε διανύσματα.

```
872 /* Αποθήκευση σε διανύσματα τις φυσικοχημικές ιδιότητες των πρωτεΐνων*/  
873 void Mesh3DScene::open_physicochemical_properties(string file, std::vector<double>& m_properties_col_0, std::vector<double>& m_properties_col_1, std::vector<double>& m_properties_col_2) {  
874     ifstream myReadfile;  
875     myReadfile.open(file);  
876     string output;  
877     if (myReadfile.is_open()) {  
878         int count = 0;  
879         while (!myReadfile.eof()) {  
880             count++;  
881             myReadfile >> output;  
882             if (count == 1) {  
883                 m_properties_col_0.push_back(stod(output));  
884             }  
885             else if (count == 2){  
886                 m_properties_col_1.push_back(stod(output));  
887             }  
888             else if (count == 3) {  
889                 count = 0;  
890                 m_properties_col_2.push_back(stod(output));  
891             }  
892         }  
893     }  
894     myReadfile.close();  
895 }
```

Η συνάρτηση *find_median_value_triangle(std::vector<double>& properties_col, std::vector<double>& median_value_vector)*, βρίσκει την μέση τιμή (median value) του κάθε τριγώνου.

```

896     /* Εύρεση μέσης τιμής και τοποθέτησε στο αντίστοιχο τρίγωνο*/
897     void Mesh3DScene::find_median_value_triangle(std::vector<double>& properties_col, std::vector<double>& median_value_vector) {
898         vector<vec> vertices = m_model.getVertices();
899         vector<vvr::Triangle> triangles = m_model.getTriangles();
900         for (int i = 0; i < triangles.size(); i++) {
901             int v1 = triangles[i].vi1;
902             int v2 = triangles[i].vi2;
903             int v3 = triangles[i].vi3;
904             double median1 = properties_col[v1];
905             double median2 = properties_col[v2];
906             double median3 = properties_col[v3];
907             if ((median1 >= median2 && median1 <= median3) || (median1 <= median2 && median1 >= median3)) {
908                 median_value_vector.push_back(median1);
909             }
910             else if ((median2 >= median1 && median2 <= median3) || (median2 <= median1 && median2 >= median3)) {
911                 median_value_vector.push_back(median2);
912             }
913             else if ((median3 >= median2 && median3 <= median1) || (median3 <= median2 && median3 >= median1)) {
914                 median_value_vector.push_back(median3);
915             }
916         }
917     }
918 }
919 }
```

Η συνάρτηση *find_colour_face_0*(*std::vector<double>& median_value_vector, std::vector<vvr::Colour>& colour_face*), υπολογίζει τα χρώματα για την απεικόνιση των ιδιοτήτων των πρωτεϊνών.

```

920     /* Συναρτήσεις τοποθέτησης χρώματος για τις τρεις ιδιότητες*/
921     void Mesh3DScene::find_colour_face_0(std::vector<double>& median_value_vector, std::vector<vvr::Colour>& colour_face) {
922         for (int i = 0; i < median_value_vector.size(); i++) {
923             if (median_value_vector[i] >= 0) {
924                 int colour_intensity = floor(median_value_vector[i] * 200 / max_value_col_0);
925                 Colour c(255, 200 - colour_intensity, 200 - colour_intensity);
926                 colour_face.push_back(c);
927             }
928             if (median_value_vector[i] < 0) {
929                 int colour_intensity = floor(median_value_vector[i] * 200 / min_value_col_0);
930                 Colour c(200 - colour_intensity, 200 - colour_intensity, 255);
931                 colour_face.push_back(c);
932             }
933         }
934     }
```

Το ίδιο κάνουν και οι παρακάτω άλλα ορίζουν άλλες αποχρώσεις.

```

935     void Mesh3DScene::find_colour_face_1(std::vector<double>& median_value_vector, std::vector<vvr::Colour>& colour_face) {
936         for (int i = 0; i < median_value_vector.size(); i++) {
937             if (median_value_vector[i] >= 0) {
938                 int colour_intensity = floor(median_value_vector[i] * 200 / max_value_col_1);
939                 Colour c(255, 200 - colour_intensity, 255);
940                 colour_face.push_back(c);
941             }
942             if (median_value_vector[i] < 0) {
943                 int colour_intensity = floor(median_value_vector[i] * 200 / min_value_col_1);
944                 Colour c(200 - colour_intensity, 255, 255);
945                 colour_face.push_back(c);
946             }
947         }
948     }
949     void Mesh3DScene::find_colour_face_2(std::vector<double>& median_value_vector, std::vector<vvr::Colour>& colour_face) {
950         for (int i = 0; i < median_value_vector.size(); i++) {
951             if (median_value_vector[i] >= 0) {
952                 int colour_intensity = floor(median_value_vector[i] * 200 / max_value_col_2);
953                 Colour c(200 - colour_intensity, 255, 200 - colour_intensity);
954                 colour_face.push_back(c);
955             }
956             if (median_value_vector[i] < 0) {
957                 int colour_intensity = floor(median_value_vector[i] * 200 / min_value_col_2);
958                 Colour c(255, 255, 200 - colour_intensity);
959                 colour_face.push_back(c);
960             }
961         }
962     }
```

Στις τρεις τελευταίες συναρτήσεις χρησιμοποιούνται οι τύποι (1.1) και (1.2).

Ο κώδικας είναι:

```

108 //---Task 1---/
109 else if (TASK == 1) {
110     // Τοποθέτηση ιδιοτήτων σε διανύσματα
111     open_physicochemical_properties(objFileProp, m_properties_col_0, m_properties_col_1, m_properties_col_2);
112     //// Εύρεση των τιμών και του χρώματος που θα πάρει το κάθε τρίγωνο
113     //propertie col 0
114     vector<double> vd0 = find_max_min_value(m_properties_col_0);
115     min_value_col_0 = vd0[0];
116     max_value_col_0 = vd0[1];
117     find_median_value_triangle(m_properties_col_0, median_value_col_0);
118     find_colour_face_0(median_value_col_0, colour_face_0);
119     // propertie col 1
120     vector<double> vd1 = find_max_min_value(m_properties_col_1);
121     min_value_col_1 = vd1[0];
122     max_value_col_1 = vd1[1];
123     find_median_value_triangle(m_properties_col_1, median_value_col_1);
124     find_colour_face_1(median_value_col_1, colour_face_1);
125     // propertie col 2
126     vector<double> vd2 = find_max_min_value(m_properties_col_2);
127     min_value_col_2 = vd2[0];
128     max_value_col_2 = vd2[1];
129     find_median_value_triangle(m_properties_col_2, median_value_col_2);
130     find_colour_face_2(median_value_col_2, colour_face_2);

```

Επίσης για να αναπαραστήσουμε τις τρεις ιδιότητες δημιουργούμε τρία αντικείμενα, συν ένα για την ίδια την πρωτεΐνη, και τις τοποθετούμε στην σειρά όπως δείχνετε παρακάτω.

```

131     //// Αναπαράσταση, αλλάγη θέσης των πρωτεινών
132     for (int i = 0; i < m_model.getVertices().size(); i++) {
133         vec v(-30, 0, 0);
134         m_model.getVertices()[i] = m_model.getVertices()[i] + v;
135     }
136     m_model.update();
137     // Prop 0
138     m_model_col_0 = m_model_original;
139     m_model_col_0.setBigSize(getSceneWidth() / 5);
140     m_model_col_0.update();
141     for (int i = 0; i < m_model_col_0.getVertices().size(); i++) {
142         vec v(-10, 0, 0);
143         m_model_col_0.getVertices()[i] = m_model_col_0.getVertices()[i] + v;
144     }
145     m_model_col_0.update();
146     // Prop 1
147     m_model_col_1 = m_model_original;
148     m_model_col_1.setBigSize(getSceneWidth() / 5);
149     m_model_col_1.update();
150     for (int i = 0; i < m_model_col_1.getVertices().size(); i++) {
151         vec v(10, 0, 0);
152         m_model_col_1.getVertices()[i] = m_model_col_1.getVertices()[i] + v;
153     }
154     m_model_col_1.update();
155     // Prop 2
156     m_model_col_2 = m_model_original;
157     m_model_col_2.setBigSize(getSceneWidth() / 5);
158     m_model_col_2.update();
159     for (int i = 0; i < m_model_col_2.getVertices().size(); i++) {
160         vec v(30, 0, 0);
161         m_model_col_2.getVertices()[i] = m_model_col_2.getVertices()[i] + v;
162     }
163     m_model_col_2.update();
164 }

```

//--- Task 2 ---//

```
12 #define TASK 2
13 #define ON_FILE 0          // Αποθήκευση δεδομένων σε αρχεία
14 #define HIST_SIZE 50       // Αριθμός υποδιαστημάτων
15 #define WEIGHT_GAUSS 1     // Βάρος της καμπυλότητας gauss
16 #define T 0.002            // Κατώφλι όμοιων πρωτεΐνων
17
```

Στο ερώτημα 2 υλοποιούμε μία συνάρτηση την *calc_gauss_mean_curve()*, η οποία υπολογίζει τις καμπυλότητες gauss και mean. Χρησιμοποιούμε την ίδια διαδρομή φακέλων όπως στο ερώτημα ένα για να βρούμε τις καμπυλότητες ενός αντικειμένου, χωρίς να χρειαστούμε τις ιδιότητες του.

```
965     /* Συνάρτηση υπολογισμού καμπυλοτήτων*/
966     void Mesh3DScene::calc_gauss_mean_curve() {
967         find_vertex_neighbours(m_model, m_NeighboursList);
968         find_Normal_vertices_surface(m_model, m_NeighboursList);
969         // Calculate gauss curve
970         calc_gauss_curve(m_model, m_NeighboursList, K_gauss_curve);
971         vector<double> vd_K = find_max_min_value(K_gauss_curve);
972         min_value_K = vd_K[0];
973         max_value_K = vd_K[1];
974         find_median_value_triangle(K_gauss_curve, median_value_K);
975         find_colour_face_gauss(median_value_K, colour_K);
976         // Calculate mean curve
977         calc_mean_curve(m_model, m_NeighboursList, H_mean_curve);
978         vector<double> vd_H = find_max_min_value(H_mean_curve);
979         min_value_H = vd_H[0];
980         max_value_H = vd_H[1];
981         find_median_value_triangle(H_mean_curve, median_value_H);
982         find_colour_face_mean(median_value_H, colour_H);
983     }
```

```
29     //---Task 1---/
30     const string obj = "0"; // Αριθμός Πρωτεΐνης
31     const string Set = "dataset"; // Σύνολο
32     // Κατεύθυνση της πρωτεΐνης
33     const string objDirSurfaces = getbasePath() + "resources/" + Set + "/surfaces/";
34     const string objFileSurf = objDirSurfaces + obj + ".obj";
35     // Κατεύθυνση των ιδιοτήτων
36     const string objDirProperties = getbasePath() + "resources/" + Set + "/properties/";
37     const string objFileProp = objDirProperties + obj + ".txt";
```

Οι συναρτήσεις που υλοποιούμε είναι οι εξής:

```
968     //---Task 2---/
969     /* Συνάρτηση υπολογισμού καμπυλοτήτων*/
970     void Mesh3DScene::calc_gauss_mean_curve(){ ... }
971     /* Κατασκευή ενός διανύματος της δομής vertex_neighbours*/
972     void Mesh3DScene::find_vertex_neighbours(vvr::Mesh m_model, std::vector<vertex_neighbours>& m_NeighboursList){ ... }
973     /* Υπολογισμός καμπυλότητας gauss*/
974     void Mesh3DScene::calc_gauss_curve(vvr::Mesh m_model, std::vector<vertex_neighbours>& m_NeighboursList, std::vector<double>& K_gauss_curve){ ... }
975     /* Υπολογισμός του normal διανυμάτων με μία επιφανεία*/
976     void Mesh3DScene::calc_normal_surface(math::vec u, math::vec ui, math::vec ui_n){ ... }
977     /* Εύρεση των normal διανυμάτων*/
978     void Mesh3DScene::find_Normal_vertices_surface(vvr::Mesh m_model, std::vector<vertex_neighbours>& m_NeighboursList){ ... }
979     /* Υπολογισμός καμπυλότητας mean*/
980     void Mesh3DScene::calc_mean_curve(vvr::Mesh m_model, std::vector<vertex_neighbours>& m_NeighboursList, std::vector<double>& H_mean_curve){ ... }
981     /* Συναρτήσεις τοποθέτησης χρώματος για τις καμπυλότητες*/
982     void Mesh3DScene::find_colour_face_gauss(std::vector<double>& median_value_vector, std::vector<vvr::Colour>& colour_face){ ... }
983     void Mesh3DScene::find_colour_face_mean(std::vector<double>& median_value_vector, std::vector<vvr::Colour>& colour_face){ ... }
```

Η συνάρτηση *find_vertex_neighbours(vvr::Mesh m_model, std::vector<vertex_neighbours>& m_NeighboursList)*, βρίσκει τους γειτονικούς κόμβους και τα τρίγωνα ενός διανύσματος, που τα αποθηκεύει

στην δομή `vertex_neighbours` και τα τοποθετεί σε ένα διάνυσμα. Η συνάρτηση εκτελεί έναν αλγόριθμο ο οποίος τοποθετεί τους γειτόνους του κόμβου στην σειρά, ώστε να υπολογίσουμε κάποιες παραμέτρους στους τύπους (2.1) και (2.2).

```

984     /* Κατασκευή ενός διανύσματος της δομής vertex_neighbours */
985     void Mesh3DScene::find_vertex_neighbours(vvr::Mesh m_model, std::vector<vertex_neighbours>& m_NeighboursList) {
986         for (int i = 0; i < m_model.getVertices().size(); i++) {
987             vertex_neighbours vnl;
988             m_NeighboursList.push_back(vnl);
989         }
990         vector<cvvr::Triangle>& triangles = m_model.getTriangles();
991         for (int i = 0; i < triangles.size(); i++) {
992             int v1 = triangles[i].v1;
993             int v2 = triangles[i].v2;
994             int v3 = triangles[i].v3;
995             // Πρόσθεση τριγώνου στο m_NeighboursList[v_]
996             m_NeighboursList[v1].triangles.push_back(i);
997             m_NeighboursList[v2].triangles.push_back(i);
998             m_NeighboursList[v3].triangles.push_back(i);
999             // Πρόσθεση γειτονικών σημείων στο m_NeighboursList[v_]
1000             // v1
1001             m_NeighboursList[v1].vertex.push_back(v2);
1002             m_NeighboursList[v1].vertex.push_back(v3);
1003             // v2
1004             m_NeighboursList[v2].vertex.push_back(v1);
1005             m_NeighboursList[v2].vertex.push_back(v3);
1006             // v3
1007             m_NeighboursList[v3].vertex.push_back(v1);
1008             m_NeighboursList[v3].vertex.push_back(v2);
1009         }
1010         // Τοποθέτηση στην σειρά τους κόμβους
1011         for (int i = 0; i < m_NeighboursList.size(); i++) {
1012             // Άλλαγή θέσεων
1013             for (int j = 1; j < m_NeighboursList[i].vertex.size(); j += 2) {
1014                 for (int k = j + 1; k < m_NeighboursList[i].vertex.size(); k++) {
1015                     if (m_NeighboursList[i].vertex[j] == m_NeighboursList[i].vertex[k]) {
1016                         if (k == j + 1) {
1017                             break;
1018                         }
1019                         else if (k == j + 2) {
1020                             vector<int>::iterator it = m_NeighboursList[i].vertex.begin();
1021                             int save_val = m_NeighboursList[i].vertex[k];
1022                             m_NeighboursList[i].vertex.erase(it + k);
1023                             m_NeighboursList[i].vertex.insert(it + j + 1, save_val);
1024                         }
1025                         else if (k % 2 == 0) {
1026                             vector<int>::iterator it = m_NeighboursList[i].vertex.begin();
1027                             int save_val = m_NeighboursList[i].vertex[k];
1028                             m_NeighboursList[i].vertex.erase(it + k);
1029                             m_NeighboursList[i].vertex.insert(it + j + 1, save_val);
1030                             save_val = m_NeighboursList[i].vertex[k + 1];
1031                             m_NeighboursList[i].vertex.erase(it + k + 1);
1032                             m_NeighboursList[i].vertex.insert(it + j + 2, save_val);
1033                             break;
1034                         }
1035                         else if (k % 2 == 1) {
1036                             vector<int>::iterator it = m_NeighboursList[i].vertex.begin();
1037                             int save_val = m_NeighboursList[i].vertex[k - 1];
1038                             m_NeighboursList[i].vertex.erase(it + k - 1);
1039                             m_NeighboursList[i].vertex.insert(it + j + 1, save_val);
1040                             save_val = m_NeighboursList[i].vertex[k];
1041                             m_NeighboursList[i].vertex.erase(it + k);
1042                             m_NeighboursList[i].vertex.insert(it + j + 1, save_val);
1043                             break;
1044                         }
1045                     }
1046                 }
1047             }
1048             for (int l = m_NeighboursList[i].vertex.size() - 1; l > 0; l -= 2) {
1049                 vector<int>::iterator it = m_NeighboursList[i].vertex.begin();
1050                 m_NeighboursList[i].vertex.erase(it + l);
1051             }
1052         }
1053     }

```

Η συνάρτηση `calc_gauss_curve(vvr::Mesh m_model, std::vector<vertex_neighbours>& m_NeighboursList, std::vector<double>& K_gauss_curve)`, υπολογίζει την καμπυλότητα gauss.

```

1054     /* Υπολογισμός καμπυλότητας gauss*/
1055     void Mesh3DScene::calc_gauss_curve(vvr::Mesh m_model, std::vector<vertex_neighbours>& m_NeighboursList, std::vector<double>& K_gauss_curve) {
1056         vector<vec>& vertices = m_model.getVertices();
1057         vector<vvr::Triangle>& triangles = m_model.getTriangles();
1058         for (int i = 0; i < vertices.size(); i++) {
1059             vertex_neighbours vn = m_NeighboursList[i];
1060             double angle = 0;
1061             double area_triangle = 0;
1062             int N = vn.vertex.size();
1063             for (int j = 0; j < N; j++) {
1064                 angle += vertices[vn.vertex[j]].AngleBetween(vertices[vn.vertex[(j + 1) % N]]);
1065                 math::Triangle tr(triangles[vn.vertex[j]].v1(), triangles[vn.vertex[j]].v2(), triangles[vn.vertex[j]].v3());
1066                 area_triangle += tr.Area();
1067             }
1068             double K = 3 * (2 * pi - angle) / area_triangle;
1069             K_gauss_curve.push_back(K);
1070         }
1071     }

```

Η συνάρτηση *calc_normal_surface*(*math::vec u*, *math::vec ui*, *math::vec ui_n*), υπολογίζει τα διανύσματα *N*.

```

1072     /* Υπολογισμός του normal διανυσμάτων σε μία επιφάνεια*/
1073     vec Mesh3DScene::calc_normal_surface(math::vec u, math::vec ui, math::vec ui_n) {
1074         vec num = (ui - u).Cross(ui_n - u);
1075         vec N = num.Normalized();
1076         return N;
1077     }
1078     /* Εύρεση των normal διανυσμάτων*/

```

Η συνάρτηση *find_Normal_vertices_surface*(*vvr::Mesh m_model*, *std::vector<vertex_neighbours>& m_NeighboursList*), βρίσκει τα *N* διανύσματα που ανήκουν σε κάθε δομή *vertex_neighbours*.

```

1078     /* Εύρεση των normal διανυσμάτων*/
1079     void Mesh3DScene::find_Normal_vertices_surface(vvr::Mesh m_model, std::vector<vertex_neighbours>& m_NeighboursList) {
1080         vector<vec>& vertices = m_model.getVertices();
1081         for (int i = 0; i < vertices.size(); i++) {
1082             vertex_neighbours vn = m_NeighboursList[i];
1083             int L = vn.vertex.size();
1084             for (int j = 0; j < L; j++) {
1085                 vec N = calc_normal_surface(vertices[i], vertices[vn.vertex[j]], vertices[vn.vertex[(j + 1) % L]]);
1086                 m_NeighboursList[i].N_surface.push_back(N);
1087             }
1088         }
1089     }

```

Η συνάρτηση *calc_mean_curve*(*vvr::Mesh m_model*, *std::vector<vertex_neighbours>& m_NeighboursList*, *std::vector<double>& H_mean_curve*), υπολογίζει την μέση καμπυλότητα.

```

1090     /* Υπολογισμός καμπυλότητας mean*/
1091     void Mesh3DScene::calc_mean_curve(vvr::Mesh m_model, std::vector<vertex_neighbours>& m_NeighboursList, std::vector<double>& H_mean_curve) {
1092         vector<vec>& vertices = m_model.getVertices();
1093         vector<vvr::Triangle>& triangles = m_model.getTriangles();
1094         for (int i = 0; i < vertices.size(); i++) {
1095             vertex_neighbours vn = m_NeighboursList[i];
1096             double num = 0;
1097             double area_triangle = 0;
1098             int N = vn.vertex.size();
1099             for (int j = 0; j < N; j++) {
1100                 double e = (vertices[vn.vertex[j]] - vertices[i]).Length();
1101                 vec N_s = calc_normal_surface(vertices[i], vertices[vn.vertex[(j + 1) % N]], vertices[vn.vertex[(j + 2) % N]]);
1102                 double b = vn.N_surface[j].AngleBetween(N_s);
1103                 num += e * b;
1104                 math::Triangle tr(triangles[vn.vertex[j]].v1(), triangles[vn.vertex[j]].v2(), triangles[vn.vertex[j]].v3());
1105                 area_triangle += tr.Area();
1106             }
1107             double H = 0.75 * (num) / area_triangle;
1108             H_mean_curve.push_back(H);
1109         }
1110     }

```

Και οι συναρτήσεις *find_colour_face_gauss*(*std::vector<double>& median_value_vector*, *std::vector<vvr::Colour>& colour_face*) και *find_colour_face_mean*(*std::vector<double>& median_value_vector*, *std::vector<vvr::Colour>& colour_face*), βρίσκουν το χρώμα που θα έχει το κάθε τρίγωνο. Έχει προηγηθεί η συνάρτηση εύρεση της μέσης τιμής όπως στο προηγούμενο ερώτημα.

Ο κώδικας είναι:

```
165 //---Task 2---/
166 else if (TASK == 2) {
167     // Υπολογισμός gauss και mean καμπυλότητας
168     calc_gauss_mean_curve();
169     //// Αναπαράσταση, αλλάγη θέσης των πρωτεινών
170     // Gauss
171     m_model_gauss = m_model_original;
172     m_model_gauss.setBigSize(getSceneWidth() / 2);
173     m_model_gauss.update();
174     for (int i = 0; i < m_model_gauss.getVertices().size(); i++) {
175         vec v(-25, 0, 0);
176         m_model_gauss.getVertices()[i] = m_model_gauss.getVertices()[i] + v;
177     }
178     m_model_gauss.update();
179     // Mean
180     m_model_mean = m_model_original;
181     m_model_mean.setBigSize(getSceneWidth() / 2);
182     m_model_mean.update();
183     for (int i = 0; i < m_model_mean.getVertices().size(); i++) {
184         vec v(25, 0, 0);
185         m_model_mean.getVertices()[i] = m_model_mean.getVertices()[i] + v;
186     }
187     m_model_mean.update();
188 }
```

///--- Task 3 ---//

```
12 #define TASK 3
13 #define ON_FILE 0      // Αποθήκευση δεδομένων σε αρχεία
14 #define HIST_SIZE 50   // Αριθμός υποδιαστημάτων
15 #define WEIGHT_GAUSS 1 // Βάρος της καμπυλότητας gauss
16 #define T 0.002        // Κατώφλι όμοιων πρωτεΐνων
17
18 // Διατουςίσασια
```

Στο ερώτημα 3 βρίσκουμε τον χάρτη προεξοχής.

Το 3 και το 31 στο TASK βρίσκεται διαφορά στην συνάρτηση draw() που θα αναφέρουμε παρακάτω.

Για την απεικόνιση αντικειμένου, όχι πρωτεΐνη, πρέπει να γίνει αλλαγή μεταβλητής στον ορισμό της κλάσης Meh3DScene, στο σημείο που τοποθετούμε το όνομα του αρχείου που ανήκει το αντικείμενο.

```
54  Mesh3DScene::Mesh3DScene()
55  {
56      //! Load settings.
57      vvr::Shape::DEF_LINE_WIDTH = 4;
58      vvr::Shape::DEF_POINT_SIZE = 10;
59      m_perspective_proj = true;
60      m_bg_col = Colour("768E77");
61      m_obj_col = Colour("454545");
62      m_model_original = vvr::Mesh(objFile);
63      reset();
64 }
```

Για να χρησιμοποιήσουμε τον αλγόριθμο Dijkstra θα πρέπει να υλοποιήσουμε μία κλάση σωρού(Heap) η οποία έχει ως ιδιότητες, αρχικοποίηση, αφαίρεση από την ρίζα, αλλαγή δεδομένων ενός κόμβου και τοποθέτησή του στην κατάλληλη θέση, και αλλαγή δεδομένων κόμβων.

```
1295     ///--- Heap functions ---///
1296     /* Αρχικοποίηση του σωρού*/
1297     +void Heap::HEAP_init(int i){ ... }
1301     /* Αφαίρεση ρίζας*/
1302     +heap_node Heap::HEAP_pop(){ ... }
1344     /* Ανακατάταξη κόμβων όταν αλλάζει τιμή ένας από αυτούς*/
1345     +void Heap::change_data(int i, double x){ ... }
1401     /* Εκπτύπωση σωρού*/
1402     +void Heap::HEAP_print(){ ... }
1408     /* Αλλαγή τιμή των κόμβων*/
1409     +void Heap::swap_heap_node(heap_node& dv1, heap_node& dv2){ ... }
1413 }
```

Ετσι μπορούμε να τρέξουμε τον αλγόριθμο Dijkstra και να υπολογίσουμε την γεωδισιακή απόσταση.

```

1230     /* Αλγόριθμος Dijkstra*/
1231     void Mesh3DScene::dijkstra(int i) {
1232         while (heap_Dij.plithos > 0) {
1233             heap_node current = heap_Dij.HEAP_pop();
1234             dual_vertex& dv = Graph[current.index_vec];
1235             dv.geodesic = current.geodesic;
1236             dv.read = true;
1237             Graph[i].p += dv.geodesic * dv.area;
1238             for (int neigh = 0; neigh < dv.count_neigh + 1; neigh++) {
1239                 if (Graph[dv.neigh_dual_vec[neigh]].read == false) {
1240                     int index_neigh = dv.neigh_dual_vec[neigh];
1241                     double distance = dv.geodesic + dv.distances[neigh];
1242                     heap_Dij.change_data(index_neigh, distance);
1243                 }
1244             }
1245         }
1246     }

```

Οι συναρτήσεις που χρησιμοποιούμε είναι:

```

1141     /* Εύρεση των διανυσμάτων του δυαδικού γράφου*/
1142     void Mesh3DScene::find_dual_vertices() { ... }
1157     /* Εύρεση των γειτονικών τρίγωνων*/
1158     void Mesh3DScene::find_neigh_triangles() { ... }
1189     /* Έλεγχος αν είναι γειτονικά*/
1190     bool Mesh3DScene::is_neighbours_triangles(vvr::Triangle& tri1, vvr::Triangle& tri2) { ... }
1209     /* Υπολογισμός του βαθμού προεξοχής*/
1210     void Mesh3DScene::find_protrusion_degree() { ... }
1230     /* Αλγόριθμος Dijkstra*/
1231     void Mesh3DScene::dijkstra(int i) { ... }
1247     /*Αλγόριθμος N-rings*/
1248     void Mesh3DScene::N_ring_algorithm() { ... }
1294

```

Η συνάρτηση *find_dual_vertices()*, βρίσκει τα δυαδικά διανύσματα.

```

1137     /* Εύρεση των διανυσμάτων του δυαδικού γράφου*/
1138     void Mesh3DScene::find_dual_vertices() {
1139         cout << "find_dual_vertices()" << endl;
1140         int size_triangles = m_model.getTriangles().size();
1141         for (int i = 0; i < size_triangles; i++) {
1142             vvr::Triangle tri = m_model.getTriangles()[i];
1143             math::Triangle tri_math(tri.v1(), tri.v2(), tri.v3());
1144             vec center_mass = tri.getCenter();
1145             dual_vertex dv;
1146             dv.dual_v = center_mass;
1147             dv.index_vec = i;
1148             dv.count_neigh = -1;
1149             dv.area = tri_math.Area();
1150             Graph.push_back(dv);
1151         }
1152     }

```

Η συνάρτηση *find_neigh_triangles()*, βρίσκει τα γειτονικά τρίγωνα για την δημιουργία του γράφου. Χρησιμοποιούμε την *find_vertex_neighbours(vvr::Mesh m_model, std::vector<vertex_neighbours>& m_NeighboursList)*, ώστε να μειώσουμε την χρονική πολυπλοκότητα για την εύρεση αυτών.

```

1153  /* Εύρεση των γειτονικών τριγώνων*/
1154  void Mesh3DScene::find_neigh_triangles() {
1155      cout << "find_neigh_triangles()" << endl;
1156      vector<vr::Triangle*> triangles = m_model.getTriangles();
1157      int size_dual_list = Graph.size();
1158      int size_Neigh_lsit = m_NeighboursList.size();
1159      for (int i = 0; i < size_Neigh_lsit; i++) {
1160          for (int j = 0; j < m_NeighboursList[i].triangles.size(); j++) {
1161              int ind_j = m_NeighboursList[i].triangles[j];
1162              if (Graph[ind_j].count_neigh >= 2) {
1163                  //cout << i << endl;
1164                  continue;
1165              }
1166              for (int k = j + 1; k < m_NeighboursList[i].triangles.size(); k++) {
1167                  int ind_k = m_NeighboursList[i].triangles[k];
1168                  if (is_neighbours_triangles(triangles[ind_j], triangles[ind_k]) &&
1169                      find(begin(Graph[ind_k].neigh_dual_vec), end(Graph[ind_k].neigh_dual_vec), ind_j) == end(Graph[ind_k].neigh_dual_vec)) {
1170                      // +1 count
1171                      Graph[ind_j].count_neigh++;
1172                      Graph[ind_k].count_neigh++;
1173                      // Γειτονικά διανύσματα
1174                      Graph[ind_j].neigh_dual_vec[Graph[ind_j].count_neigh] = ind_k;
1175                      Graph[ind_k].neigh_dual_vec[Graph[ind_k].count_neigh] = ind_j;
1176                      // Απόσταση
1177                      double dist = Graph[ind_j].dual_v.Distance(Graph[ind_k].dual_v);
1178                      Graph[ind_j].distances[Graph[ind_j].count_neigh] = dist;
1179                      Graph[ind_k].distances[Graph[ind_k].count_neigh] = dist;
1180                  }
1181              }
1182          }
1183      }
1184  }

```

Η συνάρτηση *find_protrusion_degree()*, βρίσκει τον βαθμό προεξοχής κάθε δυαδικού κόμβου και χρησιμοποιεί τον αλγόριθμό Dijkstra.

```

1205  /* Υπολογισμός του βαθμού προεξοχής*/
1206  void Mesh3DScene::find_protrusion_degree() {
1207      max_p = 0;
1208      int size_heap = Graph.size();
1209      for (int i = 0; i < size_heap; i++) {
1210          for (int j = 0; j < size_heap; j++) {
1211              Graph[j].read = false;
1212              heap_node h;
1213              h.geodesic = std::numeric_limits<double>::infinity();
1214              h.index_vec = Graph[j].index_vec;
1215              heap_Dij.heap_D.push_back(h);
1216          }
1217          heap_Dij.HEAP_init(i);
1218          dijkstra(i);
1219          if (max_p < Graph[i].p) max_p = Graph[i].p;
1220      }
1221      cout << max_p << endl;
1222      for (int i = 0; i < size_heap; i++) {
1223          Graph[i].p /= max_p;
1224      }
1225  }

```

Η συνάρτηση *N_ring_algorithm()*, είναι η μέθοδος N_ring για την εύρεση των προεξοχών.

```

1244 void Mesh3DScene::N_ring_algorithm() {
1245     for (int i = 0; i < m_model.getTriangles().size(); i++) {
1246         is_projection.push_back(0);
1247         Graph[i].read = false;
1248     }
1249     int count_vec = 0;
1250     int N_ring = floor(m_model.getTriangles().size() * 0.4);
1251     int i = 0;
1252     vector<Dual_Vertice> queue;
1253     queue.push_back(Graph[count_vec]);
1254     double max_projection_degree = 0;
1255     queue.front().read = true;
1256     while (count_vec < m_model.getTriangles().size()) {
1257         for (int j = 0; j < queue[i].count_neigh + 1; j++) {
1258             Dual_Vertice& dv = Graph[queue[i].neigh_dual_vec[j]];
1259             if (!dv.read) {
1260                 queue.push_back(dv);
1261                 dv.read = true;
1262             }
1263         }
1264         i++;
1265         if (i >= N_ring) {
1266             int index_max_projection = -1;
1267             for (int j = 0; j < queue.size(); j++) {
1268                 if (queue[j].p > max_projection_degree) {
1269                     max_projection_degree = queue[j].p;
1270                     index_max_projection = j;
1271                 }
1272             }
1273             if(max_projection_degree > 0.9){
1274                 is_projection[queue[index_max_projection].index_vec] = 1;
1275             }
1276             count_vec++;
1277             for (int k = 0; k < m_model.getTriangles().size(); k++) Graph[k].read = false;
1278             max_projection_degree = 0;
1279             i = 0;
1280             queue.clear();
1281             if (count_vec == m_model.getTriangles().size()) break;
1282             queue.push_back(Graph[count_vec]);
1283             queue.front().read = true;
1284         }
1285     }
}

```

Το N_ring είναι ο αριθμός των γειτονικών κόμβων που θα γίνει ο έλεγχος για την εύρεση του μεγαλύτερου βαθμού προεξοχής σε αυτά και αν είναι μεγαλύτερο από κάποιο κατώφλι.

Ο κώδικας είναι:

```

190     else if (TASK == 3 || TASK == 31) {
191         find_dual_vertices();
192         find_vertex_neighbours(m_model, m_NeighboursList);
193         find_neigh_triangles();
194         find_protrusion_degree();
195         N_ring_algorithm();
196         cout << "draw Graph" << endl;
197     }
}

```

//--- Task 4i ---//

```

12 #define TASK 41
13 #define ON_FILE 0      // Αποθήκευση δεδομένων σε αρχεία
14 #define HIST_SIZE 10    // Αριθμός υποδιαστημάτων
15 #define WEIGHT_GAUSS 1  // Βάρος της καμπυλότητας gauss
16 #define T 0.002          // Κατώφλι όμοιων πρωτεΐνων
17

```

Στο υπο-ερώτημα 4i βρίσκουμε τον πίνακα ανομοιομορφίας τριών αντικειμένων.

Μπορούμε να χρησιμοποιήσουμε και άλλον αριθμό για το HIST_SIZE, απλά θα αλλάξει ο πίνακας ανομοιομορφίας.

Οι διευθύνσεις των αντικειμένων, καθώς και η αποθήκευση των ιστογραμμάτων είναι:

```

41  //---Task 4I---//
42  const string objDir41 = getBasePath() + "resources/obj/";
43  // Αντικείμενο 1
44  const string objFile1 = objDir41 + "armadillo_low_low.obj";
45  // Αντικείμενο 2
46  const string objFile2 = objDir41 + "unicorn_low_low.obj";
47  // Αντικείμενο 3
48  const string objFile3 = objDir41 + "bunny_low.obj";
49  // Αποθήκευση Ιστογραμμάτων
50  const string hist_obj1 = getBasePath() + "resources/histogram/Task4i/Object1.txt";
51  const string hist_obj2 = getBasePath() + "resources/histogram/Task4i/Object2.txt";
52  const string hist_obj3 = getBasePath() + "resources/histogram/Task4i/Object3.txt";
53

```

Το ιστόγραμμα το υπολογίζουμε με την συνάρτηση *calc_histogramm()*:

```

1444 void Mesh3DScene::calc_histogramm(double min_value, double max_value, std::vector<double>& data, Eigen::VectorXf& hist, std::string name) {
1445     for (int i = 0; i < data.size(); i++) {
1446         double norm = (data[i] - min_value) / (max_value - min_value);
1447         int index_hist = 0;
1448         if (norm >= 1) {
1449             index_hist = HIST_SIZE - 1;
1450         }
1451         else if (norm <= 0) {
1452             index_hist = 0;
1453         }
1454         else {
1455             index_hist = floor(HIST_SIZE * norm);
1456         }
1457         hist[index_hist]++;
1458     }
1459     // Καταγραφή σε txt
1460     ofstream myfile(name);
1461     if (myfile.is_open()) {
1462         for (int i = 0; i < hist.size(); i++) {
1463             int lambda = hist[i];
1464             myfile << lambda << "\n";
1465         }
1466         myfile.close();
1467     }
1468     else cout << "Unable to open file";
1469 }

```

Χρησιμοποιείται ο τύπος (4.2). Αφού υπολογίζουμε τα ιστογράμματα και τα προσθέσουμε σε έναν πίνακα, υπολογίζουμε τον πίνακα ανομοιομορφίας.

```

1415     /* Προσθήκη του ιστογράμματος του αντικειμένου σε έναν πίνακα MatrixXf*/
1416     void Mesh3DScene::add_histObj_MatrixXf() {
1417         int rows_matrix = histogramm_Obj.rows();
1418         int cols_matrix = histogramm_Obj.cols();
1419         for (int i = 0; i < cols_matrix; i++) {
1420             histogramm_Obj(0, i) = hist_obj_1(i) / hist_obj_1.sum();
1421             histogramm_Obj(1, i) = hist_obj_2(i) / hist_obj_2.sum();
1422             histogramm_Obj(2, i) = hist_obj_3(i) / hist_obj_3.sum();
1423         }
1424     }

```

```

1426     void Mesh3DScene::calc_dissimilarity_matrix(Eigen::MatrixXf& arr, Eigen::MatrixXf& diss_Matrix) {
1427         int rows_matrix = arr.rows();
1428         int cols_matrix = arr.cols();
1429         for (int i = 0; i < rows_matrix; i++) {
1430             for (int j = 0; j < rows_matrix; j++) {
1431                 VectorXf v1 = arr.row(i);
1432                 VectorXf v2 = arr.row(j);
1433                 diss_Matrix(i, j) = calc_error(v1, v2);
1434             }
1435         }
1436         // Κανονικοποίηση πίνακα
1437         float max_val = diss_Matrix.maxCoeff();
1438         for (int i = 0; i < rows_matrix; i++) {
1439             for (int j = 0; j < rows_matrix; j++) {
1440                 diss_Matrix(i, j) /= max_val;
1441             }
1442         }
1443     }

```

//--- Task 4ii ---//

ΣΗΜΕΙΩΣΗ: Άμα θέλουμε να μελετήσουμε ένα βάρος *WEIGHT_GAUSS* τρέχουμε πρώτα το ερώτημα 4ii, μετά το 5 και μετά το 6.

```

12     #define TASK 42
13     #define ON_FILE 0      // Αποθήκευση δεδομένων σε αρχεία
14     #define HIST_SIZE 50    // Αριθμός υποδιαστημάτων
15     #define WEIGHT_GAUSS 1  // Βάρος της καμπυλότητας gauss
16     #define T 0.002        // Κατώφλι όμοιων πρωτεΐνων
17

```

Στο υπό-ερώτημα 4ii

To *WEIGHT_GAUSS* παίρνει όποια τιμή θέλουμε στο διάστημα που αναφέρεται στο ερώτημα 4. Για να τρέξουμε το ερώτημα αυτό θα πρέπει να έχουμε τρέξει το Task 0, για να έχουμε τις καμπυλότητες των πρωτεϊνών. Με την συνάρτηση

```

1471     /* Άνοιγμα του αρχείου που περιέχει την καμπυλότητα του αντικειμένου και προσθήκη σε ένα διάνυσμα*/
1472     void Mesh3DScene::open_curves(std::string file, std::vector<double>& cuerve) {
1473         ifstream myReadFile;
1474         myReadFile.open(file);
1475         string output;
1476         if (myReadFile.is_open()) {
1477             while (!myReadFile.eof()) {
1478                 myReadFile >> output;
1479                 cuerve.push_back(stod(output));
1480             }
1481         }
1482         myReadFile.close();
1483     }

```

Ανοίγουμε τους φακέλους με τις καμπυλότητες. Οι διευθύνσεις που χρησιμοποιούνται είναι:

Άνοιγμα αρχείου με την καμπυλότητα Gauss μίας πρωτεΐνης

string op = basePath() + "resources/curvature/dataset/Gauss/" + to_string(i) + ".txt";

Αποθήκευση ιστογράμματος καμπυλότητας Gauss μίας πρωτεΐνης

string hist = basePath() + "resources/histogram/curvature_dataset/Gauss/" + to_string(i) + ".txt";

Άνοιγμα αρχείου με την καμπυλότητα Mean μίας πρωτεΐνης

op = basePath() + "resources/curvature/dataset/Mean/" + to_string(i) + ".txt";

Αποθήκευση ιστογράμματος καμπυλότητας Mean μίας πρωτεΐνης

```
hist = getBasePath() + "resources/histogram/curvature_dataset/Mean/" + to_string(i) + ".txt";
```

Για τον υπολογισμό του πίνακα ομοιομορφίας εκτελούμε την παρακάτω συνάρτηση:

```
1493 void Mesh3DScene::calc_dissimilarity_matrix_combination(Eigen::MatrixXf& gauss, Eigen::MatrixXf& mean, Eigen::MatrixXf& diss_Matrix, std::string name, float pr_gauss) {  
1494     MatrixXf diss_gauss = MatrixXf::Zero(369, 369);  
1495     MatrixXf diss_mean = MatrixXf::Zero(369, 369);  
1496     int rows_matrix = diss_gauss.rows();  
1497     int cols_matrix = diss_gauss.cols();  
1498     // dissimilarity gauss  
1499     for (int i = 0; i < rows_matrix; i++) {  
1500         for (int j = 0; j < rows_matrix; j++) {  
1501             VectorXf v1 = gauss.row(i);  
1502             VectorXf v2 = gauss.row(j);  
1503             diss_gauss(i, j) = calc_error(v1, v2);  
1504         }  
1505     }  
1506     // dissimilarity mean  
1507     for (int i = 0; i < rows_matrix; i++) {  
1508         for (int j = 0; j < rows_matrix; j++) {  
1509             VectorXf v1 = mean.row(i);  
1510             VectorXf v2 = mean.row(j);  
1511             diss_mean(i, j) = calc_error(v1, v2);  
1512         }  
1513     }  
1514     // dissimilarity matrix  
1515     diss_Matrix = pr_gauss * diss_gauss + (1 - pr_gauss) * diss_mean;  
1516     float max_val = diss_Matrix.maxCoeff();  
1517     for (int i = 0; i < rows_matrix; i++) {  
1518         for (int j = 0; j < rows_matrix; j++) {  
1519             diss_Matrix(i, j) /= max_val;  
1520         }  
1521     }  
1522     ofstream myfile(name);  
1523     if (!myfile.is_open()) {  
1524         for (int i = 0; i < 369; i++) {  
1525             for (int j = 0; j < 369; j++) {  
1526                 float lambda = diss_Matrix(i, j);  
1527                 myfile << lambda << " ";  
1528             }  
1529             myfile << "\n";  
1530         }  
1531     }  
1532     myfile.close();  
1533 }  
1534 }
```

Αποθηκεύουμε τον πίνακα αυτόν στην διεύθυνση:

```
289 cout << "Calculate dissimilarity Matrix" << endl;  
290 string op = "gauss_mean_" + to_string(static_cast<int>(WEIGHT_GAUSS*10)) + ".txt";  
291 string name = getBasePath() + "resources/curvature_dissimilarity_matrix/arrays/" + op;  
292 cout << "Calculate dissimilarity Matrix precent_gauss: " << WEIGHT_GAUSS << endl;  
293 calc_dissimilarity_matrix_combination(histogramms_Proteins_Gauss, histogramms_Proteins_Mean, dissimilarity_matrix_proteins, name, WEIGHT_GAUSS);  
294 cout << "Print dissimilarity matrix" << endl;  
295 }
```

Το αρχείο ονομάζεται ανάλογα με την τιμή που έχει το *WEIGHT_GAUSS*.

//--- Task 5 ---//

```
12 #define TASK 5  
13 #define ON_FILE 0           // Αποθήκευση δεδομένων σε αρχεία  
14 #define HIST_SIZE 50        // Αριθμός υποδιαστημάτων  
15 #define WEIGHT_GAUSS 1       // Βάρος της καμπυλότητας gauss  
16 #define T 0.002              // Κατώφλι ομοιων πρωτεΐνων  
17
```

Στο ερώτημα 5 ταξινομούμε σε ομάδες τις πρωτεΐνες.

Στο ερώτημα αυτό βάζουμε κατώφλι 0,002 με το οποίο έχουμε μηδενικό σφάλμα, δηλαδή ταξινομούνται στις σωστές ομάδες οι πρωτεΐνες.

Με την συνάρτηση *open_dissimilarity_matrix(std::string file, Eigen::MatrixXf& diss_Matrix)*, ανοίγουμε το αρχείο που έχουμε αποθηκεύσει από το προηγούμενο ερώτημα τον πίνακα ανομοιότητας με το κατάλληλο βάρος.

```

1537     /* Άνοιγμα αρχείου που περιέχει τον πίνακα ανομοιομορφίας*/
1538     void Mesh3DScene::open_dissimilarity_matrix(std::string file, Eigen::MatrixXf& diss_Matrix) {
1539         ifstream myReadFile;
1540         myReadFile.open(file);
1541         string output;
1542         int i = 0;
1543         int j = 0;
1544         if (myReadFile.is_open()) {
1545             while (!myReadFile.eof() && i < 369) {
1546                 myReadFile >> output;
1547                 diss_Matrix(i, j) = stod(output);
1548                 if (j < 368) {
1549                     j++;
1550                 }
1551                 else {
1552                     j = 0;
1553                     i++;
1554                 }
1555             }
1556         }
1557         myReadFile.close();
1558     }

```

Η συνάρτηση *similar_protein(float Thres, std::string name)*, υπολογίζουμε τον πίνακα ομοιομορφίας όπως έχει περιγράφει στο αντίστοιχο ερώτημα.

```

1559     /* Υπολογισμός του πίνακα ομοιομορφίας των πρωτεΐνων με συγκεκριμένο κατώφλι*/
1560     void Mesh3DScene::similar_protein(float Thres, std::string name) {
1561         int rows_matrix = dissimilarity_matrix_proteins_bool.rows();
1562         int cols_matrix = dissimilarity_matrix_proteins_bool.cols();
1563         for (int i = 0; i < rows_matrix; i++) {
1564             for (int j = 0; j < rows_matrix; j++) {
1565                 if (dissimilarity_matrix_proteins(i, j) <= Thres && i != j) {
1566                     dissimilarity_matrix_proteins_bool(i, j) = 1;
1567                 }
1568             }
1569         }
1570         ofstream myfile(name);
1571         if (myfile.is_open()) {
1572             for (int i = 0; i < 369; i++) {
1573                 for (int j = 0; j < 369; j++) {
1574                     float lambda = dissimilarity_matrix_proteins_bool(i, j);
1575                     myfile << lambda << " ";
1576                 }
1577                 myfile << "\n";
1578             }
1579             myfile.close();
1580         }
1581         else cout << "Unable to open file";
1582     }

```

Η συνάρτηση *classification_protein(int teams[5], std::string name)*, ταξινομεί τις πρωτεΐνες στην ομάδα με την μεγαλύτερη πλειοψηφία.

```

1583     /* Ταξινόμηση πρωτεΐνων*/
1584     float Mesh3DScene::classification_protein(int teams[5], std::string name) {
1585         int size_c = 369;
1586         VectorXf c_correct = build_c_correct(teams, size_c);
1587         VectorXf c = VectorXf::Zero(369);
1588         int rows_matrix = dissimilarity_matrix_proteins_bool.rows();
1589         int cols_matrix = dissimilarity_matrix_proteins_bool.cols();
1590         for (int i = 0; i < rows_matrix; i++) {
1591             VectorXf Rep = VectorXf::Zero(5);
1592             for (int j = 0; j < rows_matrix; j++) {
1593                 if (dissimilarity_matrix_proteins_bool(i, j) == 1) {
1594                     for (int k = 0; k < 5; k++) {
1595                         if (j <= teams[k]) {
1596                             Rep[k]++;
1597                         break;
1598                     }
1599                 }
1600             }
1601         }
1602         int class_prot = static_cast<int>(max_index(Rep));
1603         c(i) = class_prot;
1604     }
1605     // Αποθήκευση των ταξινομήσεων σ
1606     ofstream myfile(name);
1607     if (!myfile.is_open()) {
1608         for (int k = 0; k < c.rows(); k++) {
1609             float lambda = c(k);
1610             myfile << lambda << "\n";
1611         }
1612         myfile.close();
1613     }
1614     else cout << "Unable to open file";
1615     // Υπολογισμός σφαλμάτος
1616     VectorXf Rc = VectorXf::Zero(5);
1617     for (int i = 0; i < 369; i++) {
1618         if (c(i) == c_correct(i)) {
1619             Rc(static_cast<int>(c(i)))++;
1620         }
1621     }
1622     float error = 1 - Rc.sum() / 369;
1623     cout << "error classification: " << error << endl;
1624     cout << "classification: " << error << endl;
1625     print_VectorXf(c);
1626     return error;
1627 }

```

Στον πίνακα c ταξινομούνται οι πρωτεΐνες σε ομάδες, σύμφωνα με τον πίνακα ομοιομορφίας και για τον υπολογισμό του σφαλμάτος, συγκρίνεται με το πίνακα $c_correct$ που είναι οι σωστές ταξινομήσεις. Ο πίνακας $teams$ δείχνει σε ποια θέση και μετά ξεκινάει η κάθε ομάδα στο αρχείο label.txt.

Ο κώδικας στο *Tasks()* είναι:

```

297     else if (TASK == 5) {
298         double errors = 0;
299         dissimilarity_matrix_proteins = MatrixXf::Zero(369, 369);
300         string op = "gauss_mean_" + to_string(static_cast<int>(WEIGHT_GAUSS * 10)) + ".txt";
301         cout << op << endl;
302         string name_diss = getbasePath() + "resources/curvature_dissimilarity_matrix/arrays/" + op;
303         open_dissimilarity_matrix(name_diss, dissimilarity_matrix_proteins);
304         dissimilarity_matrix_proteins_bool = MatrixXf::Zero(369, 369);
305         int teams[5] = { 44, 116, 161, 226, 368 };
306         cout << "Bounder T = " << T << "->";
307         string name_boolean = getbasePath() + "resources/curvature_dissimilarity_matrix/boolean/diss_matrix_bool.txt";
308         similar_protein(T, name_boolean);
309         string op2 = "pr_" + to_string(static_cast<int>(WEIGHT_GAUSS * 10)) + "_class.txt";
310         name_diss = getbasePath() + "resources/curvature_dissimilarity_matrix/classification/" + op2;
311         errors = classification_protein(teams, name_diss);
312     }

```

Οι διευθύνσεις που χρησιμοποιούνται για την ανάγνωση και αποθήκευση των δεδομένων είναι:

Άνοιγμα αρχείου με τον πίνακα ομοιομορφίας:

```
string op = "gauss_mean_" + to_string(static_cast<int>(WEIGHT_GAUSS * 10)) + ".txt";
```

```
string name_diss = getBasePath() + "resources/curvature_dissimilarity_matrix/arrays/" + op;
```

Αποθήκευση πίνακα ανομοιομορφίας:

```
string name_boolean = getBasePath() +
"resources/curvature_dissimilarity_matrix/boolean/diss_matrix_bool.txt";
```

Αποθήκευση ταξινόμησης των ομάδων πρωτεϊνών:

```
string op2 = "pr_" + to_string(static_cast<int>(WEIGHT_GAUSS * 10)) + "_class.txt";
string name_c = getBasePath() + "resources/curvature_dissimilarity_matrix/classification/" + op2;
```

//--- Task 6 ---//

```
12  #define TASK 6
13  #define ON_FILE 1      // Αποθήκευση δεδομένων σε αρχεία
14  #define HIST_SIZE 50    // Αριθμός υπόδιαστημάτων
15  #define WEIGHT_GAUSS 1  // Βάρος της καμπυλότητας gauss
16  #define T 0.002         // Κατώφλι όμοιων πρωτεΐνων
17
```

Στο ερώτημα αυτό υπολογίζουμε το ποσοστό ανάκτησης.

Τοποθετούμε το ίδιο HIST_SIZE με τα προηγούμενα ερωτήματα και αν δεν έχουμε υπολογίσει και αποθηκεύσει τα ιστογράμματα των καμπυλοτήτων των πρωτεϊνών από το σύνολο δοκιμής, θέτουμε στο ON_FILE 1, αλλιώς 0.

Η συνάρτηση *open_hist(std::string name, Eigen::VectorXf& hist)*, αποθηκεύει σε ένα διάνυσμα το ιστόγραμμα μίας πρωτεΐνης.

```
1630 /* Άνοιγμα αρχείου που περιέχει το ιστόγραμμα του αντικειμένου*/
1631 void Mesh3DScene::open_hist(std::string name, Eigen::VectorXf& hist) {
1632     ifstream myReadFile;
1633     myReadFile.open(name);
1634     string output;
1635     if (myReadFile.is_open()) {
1636         int i = 0;
1637         while (!myReadFile.eof() && i < HIST_SIZE) {
1638             myReadFile >> output;
1639             hist[i] = stod(output);
1640             i++;
1641         }
1642     }
1643     myReadFile.close();
1644 }
```

Η συνάρτηση *open_class_weight_gauss(Eigen::VectorXf& c, std::string name)*, αποθηκεύει σε ένα διάνυσμα την ταξινόμηση των πρωτεϊνών.

```

1645  /* Άνοιγμα αρχείου που περιέχει την ταξινόμηση των πρωτεΐνων σε ομάδες*/
1646  void Mesh3DScene::open_class_weight_gauss(Eigen::VectorXf & c, std::string name) {
1647      ifstream myReadFile;
1648      myReadFile.open(name);
1649      string output;
1650      int i = 0;
1651      if (myReadFile.is_open()) {
1652          while (!myReadFile.eof() && i < 369) {
1653              myReadFile >> output;
1654              c[i] = stod(output);
1655              i++;
1656          }
1657      }
1658      else {
1659          cout << "Unable read the file";
1660      }
1661      myReadFile.close();
1662 }

```

Η συνάρτηση *find_similar_vector()*, ανοίγει τους φακέλους με τις καμπυλότητες για το σύνολο δοκιμής και για κάθε στοιχείο από το σύνολο αυτό συγκρίνεται με τις πρωτεΐνες από το σύνολο εκπαίδευσης για την εύρεση της πιο όμοιας πρωτεΐνης και δημιουργεί ένα διάνυσμα που η κάθε θέση του αντιστοιχεί με ποια πρωτεΐνη είναι όμοια.

```

1663  /* Εύρεση της πιο όμοιας πρωτεΐνης και προσθήκη σε διάνυσμα*/
1664  void Mesh3DScene::find_similar_vector() {
1665      for (int i = 0; i < 41; i++) {
1666          similar_test_data = VectorXf::Zero(369);
1667          hist_obj_protein_testset_gauss = VectorXf::Zero(HIST_SIZE);
1668          hist_obj_protein_testset_mean = VectorXf::Zero(HIST_SIZE);
1669          string name_test = to_string(i) + ".txt";
1670          string hist_test_gauss = getbasePath() + "resources/histogram/curvature_test_set/Gauss/" + to_string(i) + ".txt";
1671          open_hist(hist_test_gauss, hist_obj_protein_testset_gauss);
1672          string hist_test_mean = getbasePath() + "resources/histogram/curvature_test_set/Mean/" + to_string(i) + ".txt";
1673          open_hist(hist_test_gauss, hist_obj_protein_testset_mean);
1674          for (int j = 0; j < 369; j++) {
1675              hist_obj_protein_dataset_gauss = VectorXf::Zero(HIST_SIZE);
1676              hist_obj_protein_dataset_mean = VectorXf::Zero(HIST_SIZE);
1677              string name_data = to_string(j) + ".txt";
1678              string hist_data_gauss = getbasePath() + "resources/histogram/curvature_dataset/Gauss/" + to_string(j) + ".txt";
1679              open_hist(hist_data_gauss, hist_obj_protein_dataset_gauss);
1680              string hist_data_mean = getbasePath() + "resources/histogram/curvature_dataset/Mean/" + to_string(j) + ".txt";
1681              open_hist(hist_data_mean, hist_obj_protein_dataset_mean);
1682              // Υπολογισμός Ομοιομορφίας
1683              float sim_gauss = calc_error(hist_obj_protein_testset_gauss, hist_obj_protein_dataset_gauss);
1684              float sim_mean = calc_error(hist_obj_protein_testset_mean, hist_obj_protein_dataset_mean);
1685              float t = (WEIGHT_GAUSS * sim_gauss + (1 - WEIGHT_GAUSS) * sim_mean);
1686              similar_test_data(j) = t;
1687          }
1688          normalize_VectorXf(similar_test_data);
1689          int index_min = min_index(similar_test_data);
1690          Vec_similar_protein(i) = index_min;
1691      }
1692 }

```

Η συνάρτηση `retriaval_degree(int teams[5])`, βρίσκει, σύμφωνα με το διάνυσμα, που έχει υπολογιστεί πριν, τις ομάδες που ανήκει η κάθε πρωτεΐνη και υπολογίζει το ποσοστό ανάκτησης.

```

1693     /* Υπολογισμός ποσοστού ανάκτησης*/
1694     void Mesh3DScene::retriaval_degree(int teams[5]) {
1695         int size_c = 41;
1696         VectorXf c_correct = build_c_correct(teams, size_c);
1697         cout << "Weight Gauss: " << WEIGHT_GAUSS << endl;
1698         VectorXf c = VectorXf::Zero(369);
1699         VectorXf c_test = VectorXf::Zero(41);
1700         string name = "pr_" + to_string(static_cast<int>(WEIGHT_GAUSS * 10)) + "_class.txt";
1701         string file_class_pr = getbasePath() + "resources/curvature_dissimilarity_matrix/classification/" + name;
1702         open_class_weight_gauss(c, file_class_pr);
1703         for (int j = 0; j < Vec_similar_protein.rows(); j++) {
1704             c_test[j] = c[Vec_similar_protein[j]];
1705         }
1706         cout << "T: " << T << "->";
1707         // Εκπρήκωση των ομάδων του συνόλου δοκιμής
1708         print_VectorXf(c_test);
1709         VectorXf Rc = VectorXf::Zero(5);
1710         for (int j = 0; j < Vec_similar_protein.rows(); j++) {
1711             if (c_test[j] == c_correct(j)) {
1712                 Rc[c_test[j]] += 1;
1713             }
1714         }
1715         // retrieval
1716         float retrieval = Rc.sum() / 41;
1717         cout << "retrieval:" << retrieval << endl;
1718     }
1719 
```

Ο κώδικας στο `Tasks()` είναι:

```

314         else if (TASK == 6) {
315             // Υπολογισμός ιστογραμμάτων καμπυλοτήτων test_set
316             if (ON_FILE == 1) { ... }
317             Vec_similar_protein = VectorXf::Zero(41);
318             find_similar_vector();
319             int teams[5] = { 4, 12, 17, 22, 40 };
320             retrival_degree(teams);
321         }
322     } 
```

Οι διευθύνσεις που χρησιμοποιούνται για την ανάγνωση και αποθήκευση των δεδομένων είναι:

Άνοιγμα αρχείου με την καμπυλότητα gauss από το σύνολο δοκιμής:

```
string op = basePath() + "resources/curvature/test_set/Gauss/" + to_string(i) + ".txt";
```

Άνοιγμα αρχείου με την καμπυλότητα mean από το σύνολο δοκιμής:

```
op = basePath() + "resources/curvature/test_set/Mean/" + to_string(i) + ".txt";
```

Αποθήκευση ιστογράμματος καμπυλότητας gauss και mean:

```
string hist = basePath() + "resources/histogram/curvature_test_set/Gauss/" + to_string(i) + ".txt";
```

```
hist = basePath() + "resources/histogram/curvature_test_set/Mean/" + to_string(i) + ".txt";
```

Φόρτωση διανύσματος με τις ταξινομήσεις των πρωτεϊνών:

```
string name = "pr_" + to_string(static_cast<int>(WEIGHT_GAUSS * 10)) + "_class.txt";
string file_class_pr = basePath() + "resources/curvature_dissimilarity_matrix/classification/" +
name;
```

//--- Task 7 ---//

```
12  #define TASK 7
13  #define ON_FILE 1      // Αποθήκευση δεδομένων σε αρχεία
14  #define HIST_SIZE 50    // Αριθμός υποδιαστημάτων
15  #define WEIGHT_GAUSS 1 // Βάρος της καμπυλότητας gauss
16  #define T 0.005         // Κατώφλι όμοιων πρωτεΐνων
17
```

Τοποθετούμε στο ON_FILE 1 για να υπολογιστούν τα ιστογράμματα των φυσικοχημικών ιδιοτήτων των πρωτεΐνων στο σύνολο εκπαίδευσης και δοκιμής.

```
354 if (ON_FILE == 1) {
355     // dataset
356     for (int i = 0; i < 369; i++) {
357         hist_properties_col_0 = VectorXf::Zero(HIST_SIZE);
358         hist_properties_col_1 = VectorXf::Zero(HIST_SIZE);
359         hist_properties_col_2 = VectorXf::Zero(HIST_SIZE);
360         string name = to_string(i) + ".txt";
361         cout << name << endl;
362         string op = getbasePath() + "resources/dataset/properties/" + to_string(i) + ".txt";
363         open_physicochemical_properties(op, protein_properties_col_0, protein_properties_col_1, protein_properties_col_2);
364         string hist0_ = getbasePath() + "resources/histogram/properties_dataset/0/" + to_string(i) + ".txt";
365         string hist1_ = getbasePath() + "resources/histogram/properties_dataset/1/" + to_string(i) + ".txt";
366         string hist2_ = getbasePath() + "resources/histogram/properties_dataset/2/" + to_string(i) + ".txt";
367         calc_histogramm(min_col_0, max_col_0, protein_properties_col_0, hist_properties_col_0, hist0_);
368         calc_histogramm(min_col_1, max_col_1, protein_properties_col_1, hist_properties_col_1, hist1_);
369         calc_histogramm(min_col_2, max_col_2, protein_properties_col_2, hist_properties_col_2, hist2_);
370         protein_properties_col_0.clear();
371         protein_properties_col_1.clear();
372         protein_properties_col_2.clear();
373     }
374     // test_set
375     for (int i = 0; i < 41; i++) {
376         hist_properties_col_0 = VectorXf::Zero(HIST_SIZE);
377         hist_properties_col_1 = VectorXf::Zero(HIST_SIZE);
378         hist_properties_col_2 = VectorXf::Zero(HIST_SIZE);
379         string name = to_string(i) + ".txt";
380         cout << name << endl;
381         string op = getbasePath() + "resources/test_set/properties/" + to_string(i) + ".txt";
382         open_physicochemical_properties(op, protein_properties_col_0, protein_properties_col_1, protein_properties_col_2);
383         string hist0_ = getbasePath() + "resources/histogram/properties_test_set/0/" + to_string(i) + ".txt";
384         string hist1_ = getbasePath() + "resources/histogram/properties_test_set/1/" + to_string(i) + ".txt";
385         string hist2_ = getbasePath() + "resources/histogram/properties_test_set/2/" + to_string(i) + ".txt";
386         calc_histogramm(min_col_0, max_col_0, protein_properties_col_0, hist_properties_col_0, hist0_);
387         calc_histogramm(min_col_1, max_col_1, protein_properties_col_1, hist_properties_col_1, hist1_);
388         calc_histogramm(min_col_2, max_col_2, protein_properties_col_2, hist_properties_col_2, hist2_);
389         protein_properties_col_0.clear();
390         protein_properties_col_1.clear();
391         protein_properties_col_2.clear();
392     }
393 }
```

Τρέχουμε τώρα όλα τα ερωτήματα αλλά για τις φυσικοχημικές ιδιότητες. Οι συναρτήσεις είναι παρόμοιες:

Η συνάρτηση *calc_dissimilarity_matrix_prop(Eigen::MatrixXf& arr_histogramms, Eigen::MatrixXf& diss_matrix, std::string name)*, υπολογίζει και αποθηκεύει τον πίνακα ανομοιομορφίας για τα ιστογράμματα που έχουμε υπολογίσει. Την χρησιμοποιούμε για κάθε ιδιότητα ξεχωριστά.

```

1721  /* Υπολογισμός του πίνακα ανομοιομορφίας για τις φυσικοχημικές ιδιότητες*/
1722  void Mesh3DScene::calc_dissimilarity_matrix_prop(Eigen::MatrixXf& arr_histogramms, Eigen::MatrixXf& diss_matrix, std::string name) {
1723      for (int i = 0; i < 369; i++) {
1724          for (int j = 0; j < 369; j++) {
1725              VectorXf v1 = arr_histogramms.row(i);
1726              VectorXf v2 = arr_histogramms.row(j);
1727              diss_matrix(i, j) = calc_error(v1, v2);
1728          }
1729      }
1730      float max_val = diss_matrix.maxCoeff();
1731      for (int i = 0; i < 369; i++) {
1732          for (int j = 0; j < 369; j++) {
1733              diss_matrix(i, j) /= max_val;
1734          }
1735      }
1736      ofstream myfile(name);
1737      if (myfile.is_open()) {
1738          for (int i = 0; i < 369; i++) {
1739              for (int j = 0; j < 369; j++) {
1740                  float lambda = diss_matrix(i, j);
1741                  myfile << lambda << " ";
1742              }
1743              myfile << "\n";
1744          }
1745          myfile.close();
1746      }
1747      else cout << "Unable to open file";
1748  }

```

Η συνάρτηση *calc_dissimilarity_matrix_prop_total(std::string name)*, υπολογίζει τον πίνακα ανομοιομορφίας της μέσης τιμής των ιδιοτήτων.

```

/* Υπολογισμός του πίνακα ανομοιομορφίας για δλες τις φυσικοχημικές ιδιότητες*/
void Mesh3DScene::calc_dissimilarity_matrix_prop_total(std::string name) {
    diss_properties_total = (diss_properties_col_0 + diss_properties_col_1 + diss_properties_col_2) / 3;
    float max_val = diss_properties_total.maxCoeff();
    for (int i = 0; i < 369; i++) {
        for (int j = 0; j < 369; j++) {
            diss_properties_total(i, j) /= max_val;
        }
    }
    ofstream myfile(name);
    if (myfile.is_open()) {
        for (int i = 0; i < 369; i++) {
            for (int j = 0; j < 369; j++) {
                float lambda = diss_properties_total(i, j);
                myfile << lambda << " ";
            }
            myfile << "\n";
        }
        myfile.close();
    }
    else cout << "Unable to open file";
}

```

Η συνάρτηση *similar_prop(float Thres, Eigen::MatrixXf& diss_matrix, std::string name)*, υπολογίζει τον πίνακα ομοιομορφίας για τις φυσικοχημικές ιδιότητες.

```

1771  /* Υπολογισμός του πίνακα ομοιομορφίας των πρωτεΐνων με συγκεκριμένο κατώφλι για τις φυσικοχημικές ιδιότητες*/
1772  void Mesh3DScene::similar_prop(float Thres, Eigen::MatrixXf& diss_matrix, std::string name) {
1773      int rows_matrix = diss_matrix_prop_bool.rows();
1774      int cols_matrix = diss_matrix_prop_bool.cols();
1775      for (int i = 0; i < rows_matrix; i++) {
1776          for (int j = 0; j < rows_matrix; j++) {
1777              if (diss_matrix(i, j) <= Thres && i != j) {
1778                  diss_matrix_prop_bool(i, j) = 1;
1779              }
1780          }
1781      }
1782      ofstream myfile(name);
1783      if (myfile.is_open()) {
1784          for (int i = 0; i < 369; i++) {
1785              for (int j = 0; j < 369; j++) {
1786                  float lambda = diss_matrix_prop_bool(i, j);
1787                  myfile << lambda << " ";
1788              }
1789              myfile << "\n";
1790          }
1791          myfile.close();
1792      }
1793      else cout << "Unable to open file";
1794  }

```

Η συνάρτηση *classification_protein_prop(Eigen::MatrixXf& diss_matrix, Eigen::VectorXf& c)*, ταξινομεί τις πρωτεΐνες σε ομάδες και υπολογίζει το σφάλμα ταξινόμησης.

```

1795  /* Ταξινόμηση πρωτεΐνων*/
1796  void Mesh3DScene::classification_protein_prop(Eigen::MatrixXf& diss_matrix, Eigen::VectorXf& c) {
1797      int teams[5] = { 44, 116, 161, 226, 368 };
1798      int size_c = 369;
1799      VectorXf c_correct = build_c_correct(teams, size_c);
1800      int rows_matrix = diss_matrix_prop_bool.rows();
1801      int cols_matrix = diss_matrix_prop_bool.cols();
1802      for (int i = 0; i < rows_matrix; i++) {
1803          VectorXf Rep = VectorXf::Zero(5);
1804          for (int j = 0; j < rows_matrix; j++) {
1805              if (diss_matrix_prop_bool(i, j) == 1) {
1806                  for (int k = 0; k < 5; k++) {
1807                      if (j <= teams[k]) {
1808                          Rep[k]++;
1809                          break;
1810                      }
1811                  }
1812              }
1813          }
1814          int class_prot = static_cast<int>(max_index(Rep));
1815          c(i) = class_prot;
1816      }
1817      VectorXf Rc = VectorXf::Zero(5);
1818      for (int i = 0; i < 369; i++) {
1819          if (c(i) == c_correct(i)) {
1820              Rc(static_cast<int>(c(i)))++;
1821          }
1822      }
1823      float error = 1 - Rc.sum() / 369;
1824      cout << "error classification: " << error << endl;
1825      cout << "classification: " << endl;
1826      print_VectorXf(c);
1827  }

```

Η συνάρτηση *find_similar_vector_prop(Eigen::VectorXf& Vec_similar, std::string proprie)*, ανοίγει τους φακέλους με τις φυσικοχημικές ιδιότητες για το σύνολο δοκιμής και για κάθε πρωτεΐνη από το σύνολο αυτό συγκρίνει τις πρωτεΐνες από το σύνολο εκπαίδευσης για την εύρεση της πιο όμοιας πρωτεΐνης και δημιουργεί ένα διάνυσμα που η κάθε θέση του αντιστοιχεί με ποια πρωτεΐνη είναι όμοια.

```

1828     /* Εύρεση της πιο όμοιας πρωτεΐνης και προσθήκη σε διάνυσμα σύμφωνα με τις φυσικοχημικές ιδιότητες*/
1829     void Mesh3DScene::find_similar_vector_prop(Eigen::VectorXf& Vec_similar, std::string propertie) {
1830         for (int i = 0; i < 41; i++) {
1831             similar_test_data_prop = VectorXf::Zero(369);
1832             VectorXf hist_obj_protein_testset_prop = VectorXf::Zero(HIST_SIZE);
1833             string name_test = to_string(i) + ".txt";
1834             string hist_test_prop = getbasePath() + "resources/histogram/properties_test_set/" + propertie + to_string(i) + ".txt";
1835             open_hist(hist_test_prop, hist_obj_protein_testset_prop);
1836             for (int j = 0; j < 369; j++) {
1837                 VectorXf hist_obj_protein_dataset_prop = VectorXf::Zero(HIST_SIZE);
1838                 string name_data = to_string(j) + ".txt";
1839                 string hist_data_prop = getbasePath() + "resources/histogram/properties_dataset/" + propertie + to_string(j) + ".txt";
1840                 open_hist(hist_data_prop, hist_obj_protein_dataset_prop);
1841                 // Υπολογισμός Ομοιομορφίας
1842                 float sim_prop = calc_error(hist_obj_protein_testset_prop, hist_obj_protein_dataset_prop);
1843                 similar_test_data_prop(j) = sim_prop;
1844             }
1845             normalize_VectorXf(similar_test_data_prop);
1846             int index_min = min_index(similar_test_data_prop);
1847             Vec_similar(i) = index_min;
1848         }
1849     }

```

Και η συνάρτηση *find_similar_vector_prop_total()*, είναι το ίδιο αλλά για την μέση τιμή των φυσικοχημικών ιδιοτήτων.

```

1850     /* Εύρεση της πιο όμοιας πρωτεΐνης και προσθήκη σε διάνυσμα για όλες τις φυσικοχημικές ιδιότητες*/
1851     void Mesh3DScene::find_similar_vector_prop_total() {
1852         for (int i = 0; i < 41; i++) {
1853             similar_test_data_prop = VectorXf::Zero(369);
1854             VectorXf hist_obj_protein_testset_prop_0 = VectorXf::Zero(HIST_SIZE);
1855             VectorXf hist_obj_protein_testset_prop_1 = VectorXf::Zero(HIST_SIZE);
1856             VectorXf hist_obj_protein_testset_prop_2 = VectorXf::Zero(HIST_SIZE);
1857             string hist_test_prop_0 = getbasePath() + "resources/histogram/properties_test_set/0/" + to_string(i) + ".txt";
1858             string hist_test_prop_1 = getbasePath() + "resources/histogram/properties_test_set/1/" + to_string(i) + ".txt";
1859             string hist_test_prop_2 = getbasePath() + "resources/histogram/properties_test_set/2/" + to_string(i) + ".txt";
1860             open_hist(hist_test_prop_0, hist_obj_protein_testset_prop_0);
1861             open_hist(hist_test_prop_1, hist_obj_protein_testset_prop_1);
1862             open_hist(hist_test_prop_2, hist_obj_protein_testset_prop_2);
1863             for (int j = 0; j < 369; j++) {
1864                 VectorXf hist_obj_protein_dataset_prop_0 = VectorXf::Zero(HIST_SIZE);
1865                 VectorXf hist_obj_protein_dataset_prop_1 = VectorXf::Zero(HIST_SIZE);
1866                 VectorXf hist_obj_protein_dataset_prop_2 = VectorXf::Zero(HIST_SIZE);
1867                 string hist_data_prop_0 = getbasePath() + "resources/histogram/properties_dataset/0/" + to_string(j) + ".txt";
1868                 string hist_data_prop_1 = getbasePath() + "resources/histogram/properties_dataset/1/" + to_string(j) + ".txt";
1869                 string hist_data_prop_2 = getbasePath() + "resources/histogram/properties_dataset/2/" + to_string(j) + ".txt";
1870                 open_hist(hist_data_prop_0, hist_obj_protein_dataset_prop_0);
1871                 open_hist(hist_data_prop_1, hist_obj_protein_dataset_prop_1);
1872                 open_hist(hist_data_prop_2, hist_obj_protein_dataset_prop_2);
1873                 // Υπολογισμός Ομοιομορφίας
1874                 float sim_prop_0 = calc_error(hist_obj_protein_testset_prop_0, hist_obj_protein_dataset_prop_0);
1875                 float sim_prop_1 = calc_error(hist_obj_protein_testset_prop_1, hist_obj_protein_dataset_prop_1);
1876                 float sim_prop_2 = calc_error(hist_obj_protein_testset_prop_2, hist_obj_protein_dataset_prop_2);
1877                 similar_test_data_prop(j) = (sim_prop_0 + sim_prop_1 + sim_prop_2)/3;
1878             }
1879             normalize_VectorXf(similar_test_data_prop);
1880             int index_min = min_index(similar_test_data_prop);
1881             Vec_similar_protein_col_total(i) = index_min;
1882         }
1883     }

```

Η συνάρτηση *retriaval_degree_prop(Eigen::VectorXf& Vec_similar, Eigen::VectorXf& c)*, υπολογίζει το ποσοστό ανάκτησης.

```

1884     /* Υπολογισμός ποσοστού ανάκτησης*/
1885     void Mesh3DScene::retriaval_degree_prop(Eigen::VectorXf& Vec_similar, Eigen::VectorXf& c) {
1886         int size_c = 41;
1887         int teams_test[5] = { 4, 12, 17, 22, 40 };
1888         VectorXf c_correct_test = build_c_correct(teams_test, size_c);
1889         VectorXf c_test = VectorXf::Zero(size_c);
1890         for (int j = 0; j < Vec_similar.rows(); j++) {
1891             c_test[j] = c[Vec_similar[j]];
1892         }
1893         cout << "T: " << T << "->";
1894         VectorXf Rc = VectorXf::Zero(5);
1895         for (int j = 0; j < Vec_similar.rows(); j++) {
1896             if (c_test[j] == c_correct_test(j)) {
1897                 Rc[c_test[j]] += 1;
1898             }
1899         }
1900         print_VectorXf(c_test);
1901         // Retrieval
1902         float retrieval = Rc.sum() / 41;
1903         cout << "retrieval:" << retrieval << endl;
1904     }
1905 }
```

Ο κώδικας είναι:

```

346     //---Task 7---//  

347     else if (TASK == 7) {  

348         max_col_0 = 35.35;  

349         min_col_0 = -30.1416;  

350         max_col_1 = 4.5;  

351         min_col_1 = -4.5;  

352         max_col_2 = 1;  

353         min_col_2 = -1;  

354         if (ON_FILE == 1){ ... }  

394         // --Task 4-- //  

395         arr_hist_properties_col_0 = MatrixXf::Zero(369, HIST_SIZE);  

396         arr_hist_properties_col_1 = MatrixXf::Zero(369, HIST_SIZE);  

397         arr_hist_properties_col_2 = MatrixXf::Zero(369, HIST_SIZE);  

398         for (int i = 0; i < 369; i++) {  

399             hist_properties_col_0 = VectorXf::Zero(HIST_SIZE);  

400             hist_properties_col_1 = VectorXf::Zero(HIST_SIZE);  

401             hist_properties_col_2 = VectorXf::Zero(HIST_SIZE);  

402             string name = to_string(i) + ".txt";  

403             cout << name << endl;  

404             string hist0 = getBasePath() + "resources/histogram/properties_dataset/0/" + to_string(i) + ".txt";  

405             string hist1 = getBasePath() + "resources/histogram/properties_dataset/1/" + to_string(i) + ".txt";  

406             string hist2 = getBasePath() + "resources/histogram/properties_dataset/2/" + to_string(i) + ".txt";  

407             open_hist(hist0, hist_properties_col_0);  

408             open_hist(hist1, hist_properties_col_1);  

409             open_hist(hist2, hist_properties_col_2);  

410             arr_hist_properties_col_0.row(i) = hist_properties_col_0;  

411             arr_hist_properties_col_1.row(i) = hist_properties_col_1;  

412             arr_hist_properties_col_2.row(i) = hist_properties_col_2;  

413         }
```

```

414 // --Task 5-- //
415 string name;
416 // prop 0
417 string diss0 = getBasePath() + "resources/properties_dissimilarity_matrix/arrays/0.txt";
418 diss_properties_col_0 = MatrixXf::Zero(369, 369);
419 calc_dissimilarity_matrix_prop(arr_hist_properties_col_0, diss_properties_col_0, diss0);
420 diss_matrix_prop_bool = MatrixXf::Zero(369, 369);
421 name = getBasePath() + "resources/properties_dissimilarity_matrix/boolean/0.txt";
422 similar_prop(T, diss_properties_col_0, name);
423 cout << "Prop 0->";
424 c_prop_0 = VectorXf::Zero(369);
425 classification_protein_prop(diss_properties_col_0, c_prop_0);
426 // prop 1
427 string diss1 = getBasePath() + "resources/properties_dissimilarity_matrix/arrays/1.txt";
428 diss_properties_col_1 = MatrixXf::Zero(369, 369);
429 calc_dissimilarity_matrix_prop(arr_hist_properties_col_1, diss_properties_col_1, diss1);
430 diss_matrix_prop_bool = MatrixXf::Zero(369, 369);
431 name = getBasePath() + "resources/properties_dissimilarity_matrix/boolean/1.txt";
432 similar_prop(T, diss_properties_col_1, name);
433 cout << "Prop 1->";
434 c_prop_1 = VectorXf::Zero(369);
435 classification_protein_prop(diss_properties_col_1, c_prop_1);
436 // prop 2
437 string diss2 = getBasePath() + "resources/properties_dissimilarity_matrix/arrays/2.txt";
438 diss_properties_col_2 = MatrixXf::Zero(369, 369);
439 calc_dissimilarity_matrix_prop(arr_hist_properties_col_2, diss_properties_col_2, diss2);
440 diss_matrix_prop_bool = MatrixXf::Zero(369, 369);
441 name = getBasePath() + "resources/properties_dissimilarity_matrix/boolean/2.txt";
442 similar_prop(T, diss_properties_col_2, name);
443 cout << "Prop 2->";
444 c_prop_2 = VectorXf::Zero(369);
445 classification_protein_prop(diss_properties_col_2, c_prop_2);
446 // total
447 string disst = getBasePath() + "resources/properties_dissimilarity_matrix/arrays/total.txt";
448 diss_properties_total = MatrixXf::Zero(369, 369);
449 calc_dissimilarity_matrix_prop_total(disst);
450 diss_matrix_prop_bool = MatrixXf::Zero(369, 369);
451 name = getBasePath() + "resources/properties_dissimilarity_matrix/boolean/total.txt";
452 similar_prop(T, diss_properties_total, name);
453 cout << "Prop all->";
454 c_prop_total = VectorXf::Zero(369);
455 classification_protein_prop(diss_properties_total, c_prop_total);

```

```

456 // --Task 6-- //
457 // prop 0
458 Vec_similar_protein_col_0 = VectorXf::Zero(41);
459 find_similar_vector_prop(Vec_similar_protein_col_0, "0/");
460 cout << "Vec prop 0" << endl;
461 retrieval_degree_prop(Vec_similar_protein_col_0, c_prop_0);
462 // prop 1
463 Vec_similar_protein_col_1 = VectorXf::Zero(41);
464 find_similar_vector_prop(Vec_similar_protein_col_1, "1/");
465 cout << "Vec prop 1" << endl;
466 retrieval_degree_prop(Vec_similar_protein_col_1, c_prop_1);
467 // prop 2
468 Vec_similar_protein_col_2 = VectorXf::Zero(41);
469 find_similar_vector_prop(Vec_similar_protein_col_2, "2/");
470 cout << "Vec prop 2" << endl;
471 retrieval_degree_prop(Vec_similar_protein_col_2, c_prop_2);
472 // total
473 Vec_similar_protein_col_total = VectorXf::Zero(41);
474 find_similar_vector_prop_total();
475 cout << "Vec prop total" << endl;
476 retrieval_degree_prop(Vec_similar_protein_col_total, c_prop_total);
477 }

```

Οι διευθύνσεις που χρησιμοποιούνται για την ανάγνωση και αποθήκευση των δεδομένων είναι:

Άνοιγμα των ιστογραμμάτων των φυσικοχημικών ιδιοτήτων από το σύνολο εκπαίδευσης:

```

string hist0 = getbasePath() + "resources/histogram/properties_dataset/0/" + to_string(i) + ".txt";
string hist1 = getbasePath() + "resources/histogram/properties_dataset/1/" + to_string(i) + ".txt";
string hist2 = getbasePath() + "resources/histogram/properties_dataset/2/" + to_string(i) + ".txt";

```

Αποθήκευση των πίνακα ανομοιομορφίας και ομοιομορφίας:

```

// prop 0
string diss0 = getbasePath() + "resources/properties_dissimilarity_matrix/arrays/0.txt";
name = getbasePath() + "resources/properties_dissimilarity_matrix/boolean/0.txt";

// prop 1
string diss1 = getbasePath() + "resources/properties_dissimilarity_matrix/arrays/1.txt";
name = getbasePath() + "resources/properties_dissimilarity_matrix/boolean/1.txt";

// prop 2
string diss2 = getbasePath() + "resources/properties_dissimilarity_matrix/arrays/2.txt";
name = getbasePath() + "resources/properties_dissimilarity_matrix/boolean/2.txt";

// total
string disst = getbasePath() + "resources/properties_dissimilarity_matrix/arrays/total.txt";
name = getbasePath() + "resources/properties_dissimilarity_matrix/boolean/total.txt";

```

Άνοιγμα των ιστογραμμάτων των φυσικοχημικών ιδιοτήτων από το σύνολο εκπαίδευσης και σύνολο δοκιμής:

```

string hist_test_prop = getbasePath() + "resources/histogram/properties_test_set/" + propertie +
to_string(i) + ".txt";

string hist_data_prop = getbasePath() + "resources/histogram/properties_dataset/" + propertie +
to_string(j) + ".txt";

```

όπου το *propertie* παίρνει τιμές [“0/”, “1/”, “2/”] ανάλογα την ιδιότητα που μελετάμε.

//--- Draw ---//

Απλή απεικόνιση του αντικειμένου που έχουμε ορίσει:

```

1910     if (TASK == 0) {
1911         if (m_style_flag & FLAG_SHOW_SOLID) m_model.draw(m_obj_col, SOLID);
1912     }

```

Απεικόνιση φυσικοχημικών ιδιοτήτων πρωτεΐνων:

```

1913     //---Task 1---//
1914     if (TASK == 1) {
1915         if (m_style_flag & FLAG_SHOW_SOLID) m_model.draw(m_obj_col, SOLID);
1916         // physicochemical_properties column 0
1917         if (m_style_flag & FLAG_SHOW_SOLID) m_model_col_0.draw(m_obj_col, SOLID);
1918         vector<vr::Triangle*> triangles0 = m_model_col_0.getTriangles();
1919         for (int i = 0; i < triangles0.size(); i++) {
1920             vr::Triangle t = triangles0[i];
1921             Triangle3D t3d(
1922                 t.v1().x, t.v1().y, t.v1().z,
1923                 t.v2().x, t.v2().y, t.v2().z,
1924                 t.v3().x, t.v3().y, t.v3().z,
1925                 colour_face_0[i]);
1926             t3d.draw();
1927         }
1928         // physicochemical_properties column 1
1929         if (m_style_flag & FLAG_SHOW_SOLID) m_model_col_1.draw(m_obj_col, SOLID);
1930         vector<vr::Triangle*> triangles1 = m_model_col_1.getTriangles();
1931         for (int i = 0; i < triangles1.size(); i++) {
1932             vr::Triangle t = triangles1[i];
1933             Triangle3D t3d(
1934                 t.v1().x, t.v1().y, t.v1().z,
1935                 t.v2().x, t.v2().y, t.v2().z,
1936                 t.v3().x, t.v3().y, t.v3().z,
1937                 colour_face_1[i]);
1938             t3d.draw();
1939         }
1940         // physicochemical_properties column 2
1941         if (m_style_flag & FLAG_SHOW_SOLID) m_model_col_2.draw(m_obj_col, SOLID);
1942         vector<vr::Triangle*> triangles2 = m_model_col_2.getTriangles();
1943         for (int i = 0; i < triangles2.size(); i++) {
1944             vr::Triangle t = triangles2[i];
1945             Triangle3D t3d(
1946                 t.v1().x, t.v1().y, t.v1().z,
1947                 t.v2().x, t.v2().y, t.v2().z,
1948                 t.v3().x, t.v3().y, t.v3().z,
1949                 colour_face_2[i]);
1950             t3d.draw();
1951         }
1952     }

```

Απεικόνιση καμπυλοτήτων πρωτεϊνών:

```
1953 //---Task 2---//
1954 if (TASK == 2) {
1955     // Gauss
1956     if (m_style_flag & FLAG_SHOW_SOLID) m_model_gauss.draw(m_obj_col, SOLID);
1957     vector<vvr::Triangle>& triangles0 = m_model_gauss.getTriangles();
1958     for (int i = 0; i < triangles0.size(); i++) {
1959         vvr::Triangle& t = triangles0[i];
1960         Triangle3D t3d(
1961             t.v1().x, t.v1().y, t.v1().z,
1962             t.v2().x, t.v2().y, t.v2().z,
1963             t.v3().x, t.v3().y, t.v3().z,
1964             colour_K[i]);
1965         t3d.draw();
1966     }
1967     // Mean
1968     if (m_style_flag & FLAG_SHOW_SOLID) m_model_mean.draw(m_obj_col, SOLID);
1969     vector<vvr::Triangle>& triangles1 = m_model_mean.getTriangles();
1970     for (int i = 0; i < triangles1.size(); i++) {
1971         vvr::Triangle& t = triangles1[i];
1972         Triangle3D t3d(
1973             t.v1().x, t.v1().y, t.v1().z,
1974             t.v2().x, t.v2().y, t.v2().z,
1975             t.v3().x, t.v3().y, t.v3().z,
1976             colour_H[i]);
1977         t3d.draw();
1978     }
1979 }
```

Απεικόνιση σημείων με την μεγαλύτερο ποσοστό προεξοχής:

```
1980 //---Task 3---//
1981 if (TASK == 3) {
1982     for (int i = 0; i < Graph.size(); i++) {
1983         dual_vertex vd = Graph[i];
1984         int val = floor(Graph[i].p * 255);
1985         if (val >= 190) {
1986             vvr::Colour c(val, 255, 0);
1987             Point3D p3d(vd.dual_v.x, vd.dual_v.y, vd.dual_v.z, c);
1988             p3d.draw();
1989         }
1990         else {
1991             vvr::Colour c(0, 0, 0);
1992             Point3D p3d(vd.dual_v.x, vd.dual_v.y, vd.dual_v.z, c);
1993             p3d.draw();
1994         }
1995         for (int j = 0; j < vd.count_neigh + 1; j++) {
1996             LineSeg3D line3d(vd.dual_v.x, vd.dual_v.y, vd.dual_v.z,
1997                             Graph[vd.neigh_dual_vec[j]].dual_v.x, Graph[vd.neigh_dual_vec[j]].dual_v.y,
1998                             Graph[vd.neigh_dual_vec[j]].dual_v.z,
1999                             vvr::Colour::green);
2000             line3d.draw();
2001         }
2002     }
2003 }
```

Απεικόνιση αντικειμένου μετά την εφαρμογή της μεθόδου N-rings.

```
2004 if (TASK == 31) {
2005     for (int i = 0; i < Graph.size(); i++) {
2006         dual_vertex vd = Graph[i];
2007         if (is_projection[i] == 1) {
2008             vvr::Colour c(255, 255, 255);
2009             Point3D p3d(vd.dual_v.x, vd.dual_v.y, vd.dual_v.z, c);
2010             p3d.draw();
2011         }
2012         else {
2013             vvr::Colour c(0, 0, 0);
2014             Point3D p3d(vd.dual_v.x, vd.dual_v.y, vd.dual_v.z, c);
2015             p3d.draw();
2016         }
2017         for (int j = 0; j < vd.count_neigh + 1; j++) {
2018             LineSeg3D line3d(vd.dual_v.x, vd.dual_v.y, vd.dual_v.z,
2019                             Graph[vd.neigh_dual_vec[j]].dual_v.x, Graph[vd.neigh_dual_vec[j]].dual_v.y,
2020                             Graph[vd.neigh_dual_vec[j]].dual_v.z,
2021                             vvr::Colour::green);
2022             line3d.draw();
2023         }
2024     }
2025 }
```

Απεικόνιση των πίνακα ανομοιομορφίας:

```

2025 //---Task 4ii---/
2026 if (TASK == 42) {
2027     for (int i = -185; i < 184; i++) {
2028         for (int j = -185; j < 184; j++) {
2029             if ((i == -185 && j == 185) || (i == 0 && j == 0)) {
2030                 vec v(i, j, 0);
2031                 int save = static_cast<int>(dissimilarity_matrix_proteins(i + 185, j + 185) * 255);
2032                 Colour c(255, 0, 0);
2033                 Point3D p(v.x, v.y, v.z, c);
2034                 p.draw();
2035                 continue;
2036             }
2037             vec v(i, j, 0);
2038             int save = static_cast<int>(dissimilarity_matrix_proteins(i + 185, j + 185) * 255);
2039             Colour c(0, 255 - save, 0);
2040             Point3D p(v.x, v.y, v.z, c);
2041             p.draw();
2042         }
2043     }
2044 }
```

Ο πίνακας αυτός βγαίνει ανάποδα. Αυτό το κάνουμε διότι φαίνονται πιο έντονα το χρώμα σε κάποια σημεία και δεν ξεχωρίζονται καλά τα αποτελέσματα. Για αυτό τον λόγο πατάμε με το ποντίκι την εικόνα και την στρέφουμε προς τα πάνω για να δούμε τον πίνακα ανομοιομορφίας.

Απεικόνιση του πίνακα ομοιομορφίας:

```

2046 if (TASK == 5) {
2047     for (int i = -185; i < 184; i++) {
2048         for (int j = -185; j < 184; j++) {
2049             if ((i == -185 && j == -185) || (i == 0 && j == 0)) {
2050                 vec v(i, j, 0);
2051                 int save = static_cast<int>(dissimilarity_matrix_proteins_bool(i + 185, j + 185) * 255);
2052                 Colour c(255, 0, 0);
2053                 Point3D p(v.x, v.y, v.z, c);
2054                 p.draw();
2055                 continue;
2056             }
2057             vec v(i, j, 0);
2058             int save = static_cast<int>(dissimilarity_matrix_proteins_bool(i + 185, j + 185) * 255);
2059             Colour c(0, save, 0);
2060             Point3D p(v.x, v.y, v.z, c);
2061             p.draw();
2062         }
2063     }
2064 }
```

Το ίδιο συμβαίνει και εδώ.

Βιβλιογραφία

//--- Task 2 ---//

[2.1]: Tatiana Surazhsky, Evgeny Magid, Octavian Soldea, Gershon Elber and Ehud Rivlin, *A Comparison of Gaussian and Mean Curvatures Estimation Methods on Triangular Meshes*, October 2003 Proceedings - IEEE International Conference on Robotics and Automation 1:1021 - 1026 vol.1

//--- Task 3 ---//

[3.1]: Konstantinos Moustakas, Dimitrios Tzovaras, and Michael Gerassimos Strintzis, *SQ-Map: Efficient Layered Collision Detection and Haptic Rendering*, February 2007 IEEE Transactions on Visualization and Computer Graphics 13(1):80-93