ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΤΜΗΜΑ ΜΟΡΙΑΚΗΣ ΒΙΟΛΟΓΙΑΣ ΚΑΙ ΓΕΝΕΤΙΚΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

grcarma :

Ένα πρόγραμμα γραφικού περιβάλλοντος διεπαφής χρήστη για την ανάλυση τροχιακών μοριακής δυναμικής.

Κούκος Παναγιώτης

Επιβλέπων:

Δρ. Νικόλαος Μ. Γλυκός

Επίκουρος Καθηγητής Υπολογιστικής και Δομικής Βιολογίας

Τμήμα Μοριακής Βιολογίας και Γενετικής

Δημοκρίτειο Πανεπιστήμιο Θράκης

Αλεξανδρούπολη

Σεπτέμβριος 2013

Ευχαριστίες

Πρωτίστως, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Νικόλαο Γλυκό για την καθοδήγηση που μου προσέφερε καθ όλη την διάρκεια συγγραφής του προγράμματος το οποίο ήταν το αντικείμενο της διπλωματικής μου εργασίας. Απόντων των προτάσεων αλλά και των εξαντλητικών δοκιμών του η εικόνα του προγράμματος θα ήταν πολύ πτωχότερη.

Επιπλέον θα ήθελα να ευχαριστήσω την οικογένειά μου, για όλα όσα μου προσέφεραν και συνεχίζουν να μου προσφέρουν.

Πίνακας Περιεχομένων

Περίληψη

Το **grcarma** είναι ένα πρόγραμμα το οποίο ενσωματώνει πολλούς αυτοματισμούς σκοπός των οποίων είναι η απλούστευση της ανάλυσης τροχιακών μοριακής δυναμικής βιολογικών μακρομορίων. Το πρόγραμμα παρέχει ένα περιβάλλον γραφικής διεπαφής χρήστη (graphical user interface – GUI) για το πρόγραμμα **carma**, κάνοντας χρήση της γλώσσας προγραμματισμού Perl και του πακέτου (module) Perl/Tk για την δημιουργία του γραφικού περιβάλλοντος. Είναι διαθέσιμο για όλα τα διαδεδομένα λειτουργικά συστήματα (Linux, MACOSX, Windows). Έχει σχεδιαστεί με τέτοιον τρόπο ώστε να εξυπηρετεί τις ανάγκες έμπειρων αλλά και αρχάριων χρηστών, διατηρώντας ταυτόχρονα φιλική προς τον χρήστη σχεδίαση. Το σημαντικότερο στοιχείο του προγράμματος είναι οι προαναφερθέντες αυτοματισμοί. Σε αυτούς συγκαταλέγονται οι : εξαγωγή συμπλεγμάτων παρόμοιων δομών μετά από δίεδρη ή καρτεσιανή ανάλυση κύριων συνιστωσών (PCA), ανάλυση δευτεροταγούς δομής, υπολογισμό και προβολή πινάκων RMSD, υπολογισμό εντροπίας, υπολογισμό και ανάλυση πινάκων διακύμανσης-συνδιακύμανης (variance-covariance) κα. Το πρόγραμμα είναι δωρεάν και ελεύθερο λογισμικό ανοικτού κώδικα.

Abstract

Title

grcarma : A GUI program for the analysis of molecular dynamics trajectories.

grcarma is a program encoding for a fully automated set of tasks aiming to simplify the analysis of molecular dynamics trajectories of biological macromolecules. It is a cross-platform, Perl/Tk based front-end to the program **carma** and is designed to facilitate the needs of the novice as well as those of the expert user, while at the same time maintaining a user-friendly and intuitive design. Particular emphasis was given to the automation of several tedious tasks, such as extraction of clusters of structures based on dihedral and Cartesian principal component analysis, secondary structure analysis, calculation and display of RMSD matrices, calculation of entropy, calculation and analysis of variance-covariance matrices, calculation of the fraction of native contacts, etc. The program is free-open source software available immediately for download.

Κεφάλαιο 1 – Μοριακή Δυναμική

1.1 Εισαγωγή

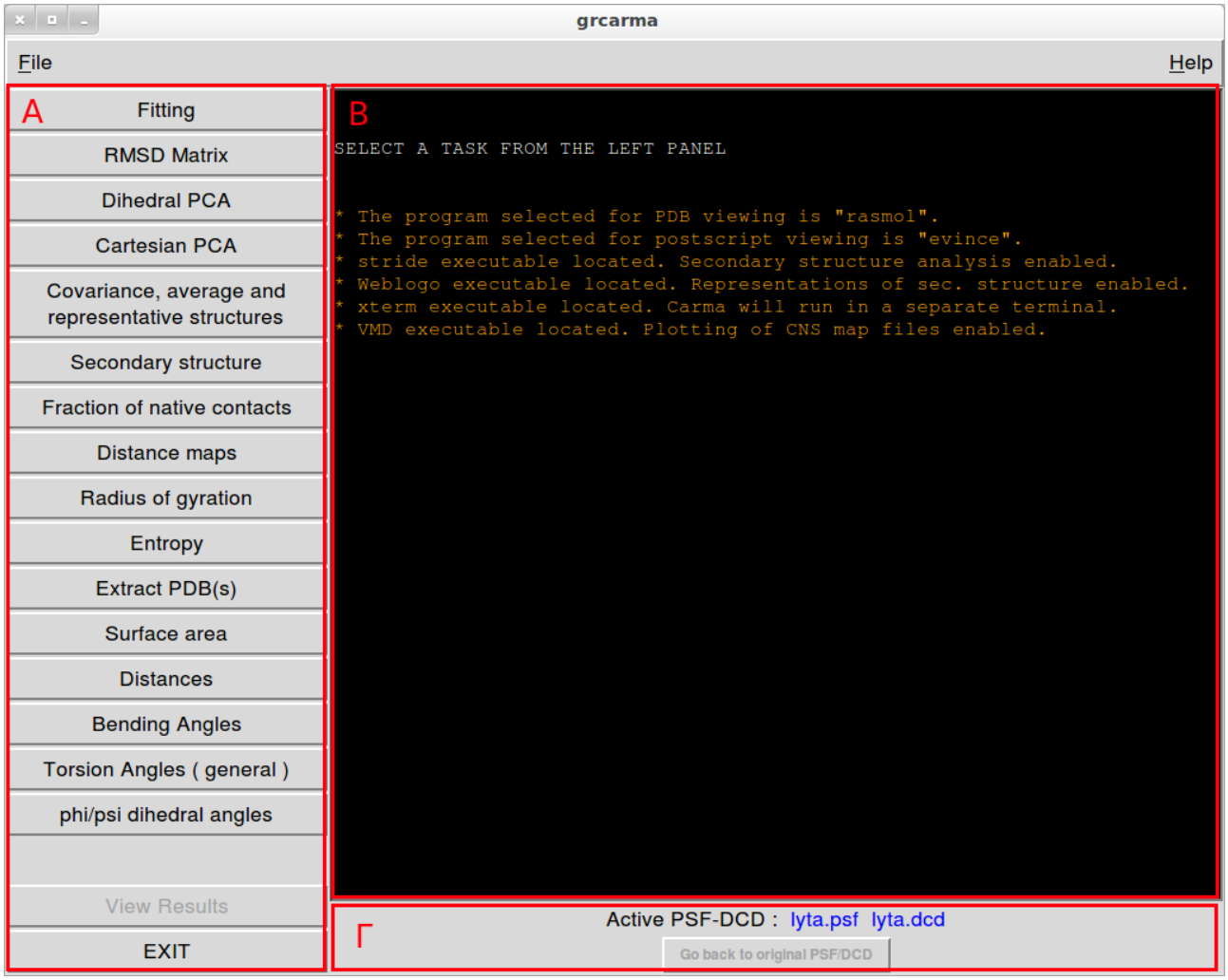
Οι προσομοιώσεις μοριακής δυναμικής (Molecular Dynamics – MD) βιολογικών μακρομορίων, όπως πρωτεΐνες και νουκλεϊκά οξέα, συνίστανται στην, *in silico,* επίλυση των νόμων της κίνησης του Νεύτωνα, για όλα τα άτομα του συστήματος, υιοθετώντας απλουστεύσεις όπως την χρήση του νόμου του Hooke για τις δεσμικές αλληλεπιδράσεις και το δυναμικό Leonard-Jones για τις μη δεσμικές αλληλεπιδράσεις[1-4]. Οι προσομοιώσεις μοριακής δυναμικής προσφέρουν πληροφορίες για τις κινητικές αλλά και θερμοδυναμικές παραμέτρους του συστήματος υπό μελέτη σε ατομιστική ακρίβεια[2]. Συνεπώς είναι κατάλληλες για την μελέτη διεργασιών η χρονική διάρκεια των οποίων εκτείνεται από μερικά nanoseconds εώς και μερικά milliseconds[1], επιτρέποντας έτσι την διαλεύκανση γεγονότων που εκτυλίσσονται σε αυτήν την χρονική κλίμακα όπως η πρωτεϊνική αναδίπλωση κα. Τα εμπειρικά δυναμικά πεδία (Empirical Force Fields), αποτελούν το σύνολο των μαθηματικών εξισώσεων που περιγράφουν την δυναμική ενέργεια των σωματίων της προσομοίωσης. Τα πιο διαδεδομένα πεδία σε χρήση είναι τα : AMBER[5], CHARMM[6] & GROMOS[7], ενώ μερικά από τα πιο διαδεδομένα προγράμματα για ανάλυση τροχιακών μοριακής δυναμικής είναι τα VMD[8] & NAMD[9]. Το **grcarma** λειτουργεί ως GUI για το πρόγραμμα ανάλυσης τροχιακών ΜΔ, **carma**[10].

1.2 Εφαρμογές των προσομοιώσεων Μοριακής Δυναμικής

Οι προσομοιώσεις ΜΔ χρησιμοποιούνται σε πολλούς διαφορετικούς κλάδους όπως η δομική βιοχημεία, η βιοφυσική, επιστήμη υλικών, η ενζυμολογία, η μοριακή βιολογία, η ενζυμολογία, η φαρμακευτική χημεία. Πρόσφατα έχει επεκταθεί σε κλάδους και εφαρμογές που παλιότερα θεωρούνταν υπερβολικά πολύπλοκοι για προσομοίωση[11-12], χάρη στην δραματική αύξηση της υπολογιστικής δύναμης, της βελτίωσης των πρωτοκόλλων επικοινωνίας ανάμεσα σε υπολογιστικές μονάδες οδηγώντας έτσι στην διάδοση της χρήσης των υπολογιστικών clusters, δηλαδή ομάδων υπολογιστών οι οποίες επικοινωνούν μεταξύ τους, προκειμένου να επισπεύσουν την επίλυση ενός προβλήματος, και της βελτιστοποίησης των εμπειρικών πεδίων. Βιολογικά προβλήματα τα οποία προσεγγίζονται, ανάμεσα σε άλλους τρόπους, και μέσω των προσομοιώσεων ΜΔ είναι ο καρκίνος[13-14], η ανακάλυψη νέων φαρμάκων (structure guided drug design) [15], η διαλεύκανση της δράσης ιών[16] και η πρωτεϊνική αναδίπλωση και πρόβλεψη πρωτεϊνικών δομών[17-18].

Κεφάλαιο 2 – Το πρόγραμμα

2.1 Περιγραφή των δυνατοτήτων του προγράμματος



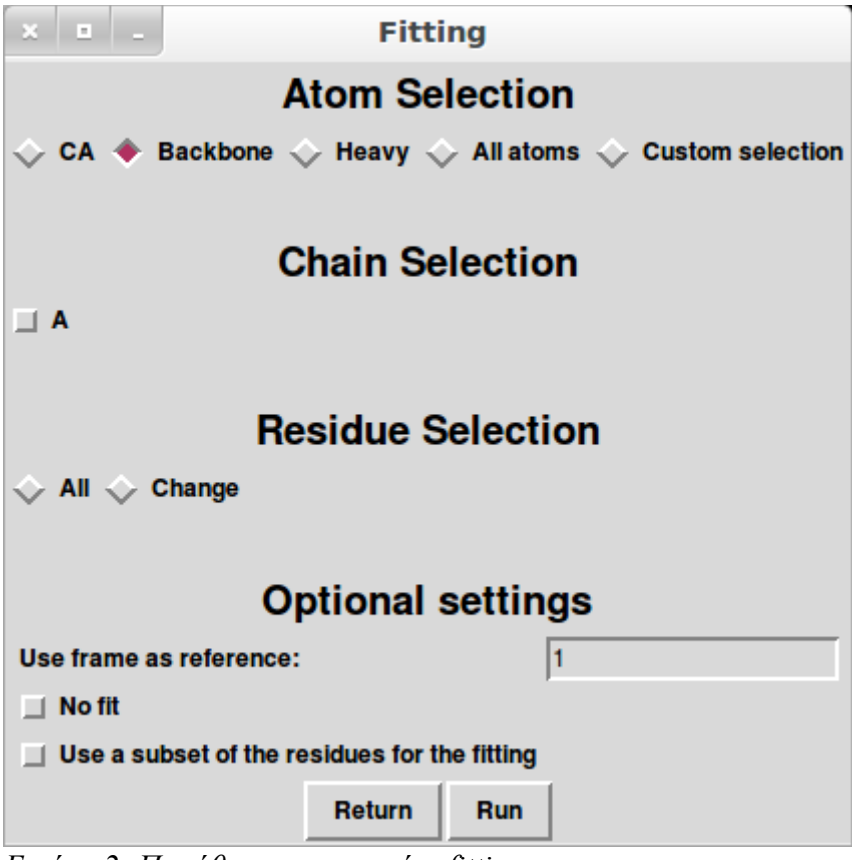
Εικόνα 1: Κύριο παράθυρο του προγράμματος.

Στην εικόνα 1 φαίνεται το κεντρικό παράθυρο του **grcarma,** το οποίο έχει χωριστεί σε τρεις περιοχές. Στην περιοχή Α στο αριστερό μέρος του παραθύρου εδράζονται τα κουμπιά τα οποία περιγράφουν τις λειτουργίες του προγράμματος, με κάθε κουμπί να αντιστοιχεί σε μία λειτουργία. Σχεδόν για όλες τις διεργασίες είναι δυνατή η επιλογή

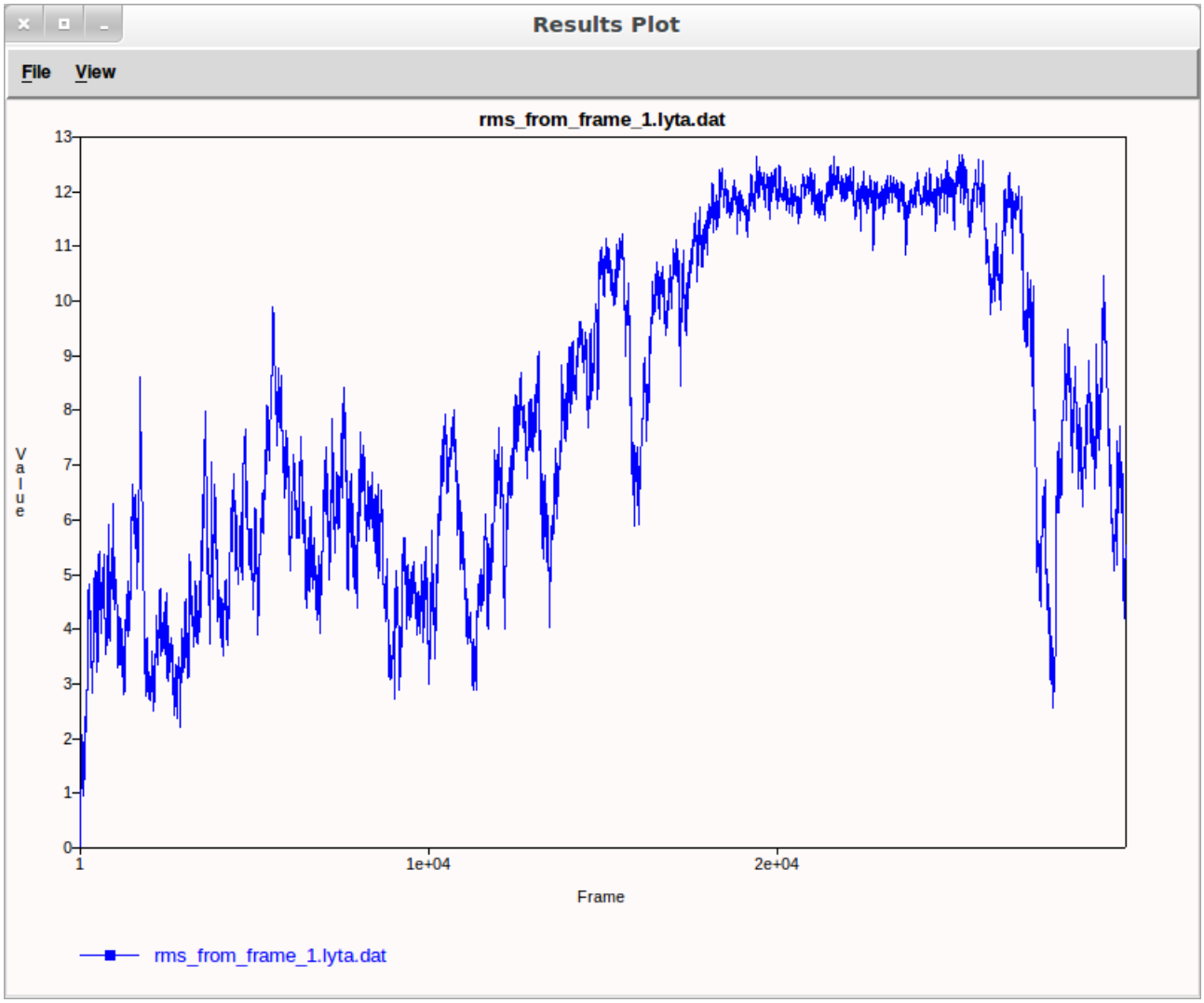
των πρωτεϊνικών αλυσίδων, ατόμων και καταλοίπων στα οποία θα πραγματοποιηθεί η εκάστοτε ανάλυση, για παράδειγμα στα άτομα C, Cα, N, O (backbone) των καταλοίπων 3-12 της υπομονάδας A μίας διμερούς πρωτεΐνης. Οι λειτουργίες αυτές θα περιγραφούν στο επόμενο κομμάτι. Το μέρος του παραθύρου το οποίο αντιστοιχεί στο γράμμα Β, χρησιμοποιείται για την εκτύπωση μηνυμάτων τα οποία παρέχουν πληροφορίες στον χρήστη, όπως για παράδειγμα, ότι κάποια εργασία ολοκληρώθηκε επιτυχώς. Τέλος η περιοχή που αντιστοιχεί στο γράμμα Γ, είναι η περιοχή στην οποία φαίνεται ποια είναι τα ενεργά αρχεία κάθε στιγμή, καθώς η αλλαγή αρχείων είναι δυνατή μέσω του προγράμματος. Το **grcarma,** όπως και το **carma**, είναι συμβατό με αρχεία PSF/DCD. Εκτός από όπου αναφέρεται κάτι διαφορετικό για τους παρακάτω υπολογισμούς χρησιμοποιήθηκε το πεπτίδιο LytA[18].

2.1.1 Fitting

Η λειτουργία αυτή επιτρέπει την αφαίρεση των περιστροφών και μετατοπίσεων (global rotations-translations removal) από ένα τροχιακό μέσω της υπέρθεσης (superposition) όλων των δομών που περιέχονται σε ένα τροχιακό στην πρώτη ή σε οποιαδήποτε άλλη δομή. Το παράθυρο της εργασίας φαίνεται στην εικόνα 2, ενώ στην εικόνα 3 παρουσιάζεται ένα γράφημα της απόστασης των δομών από την δομή η οποία χρησιμοποιήθηκε ως σημείο αναφοράς, εν προκειμένω η πρώτη. Η προβολή της εικόνας γίνεται μέσω του **grcarma,** χωρίς την μεσολάβηση εξωτερικού προγράμματος.



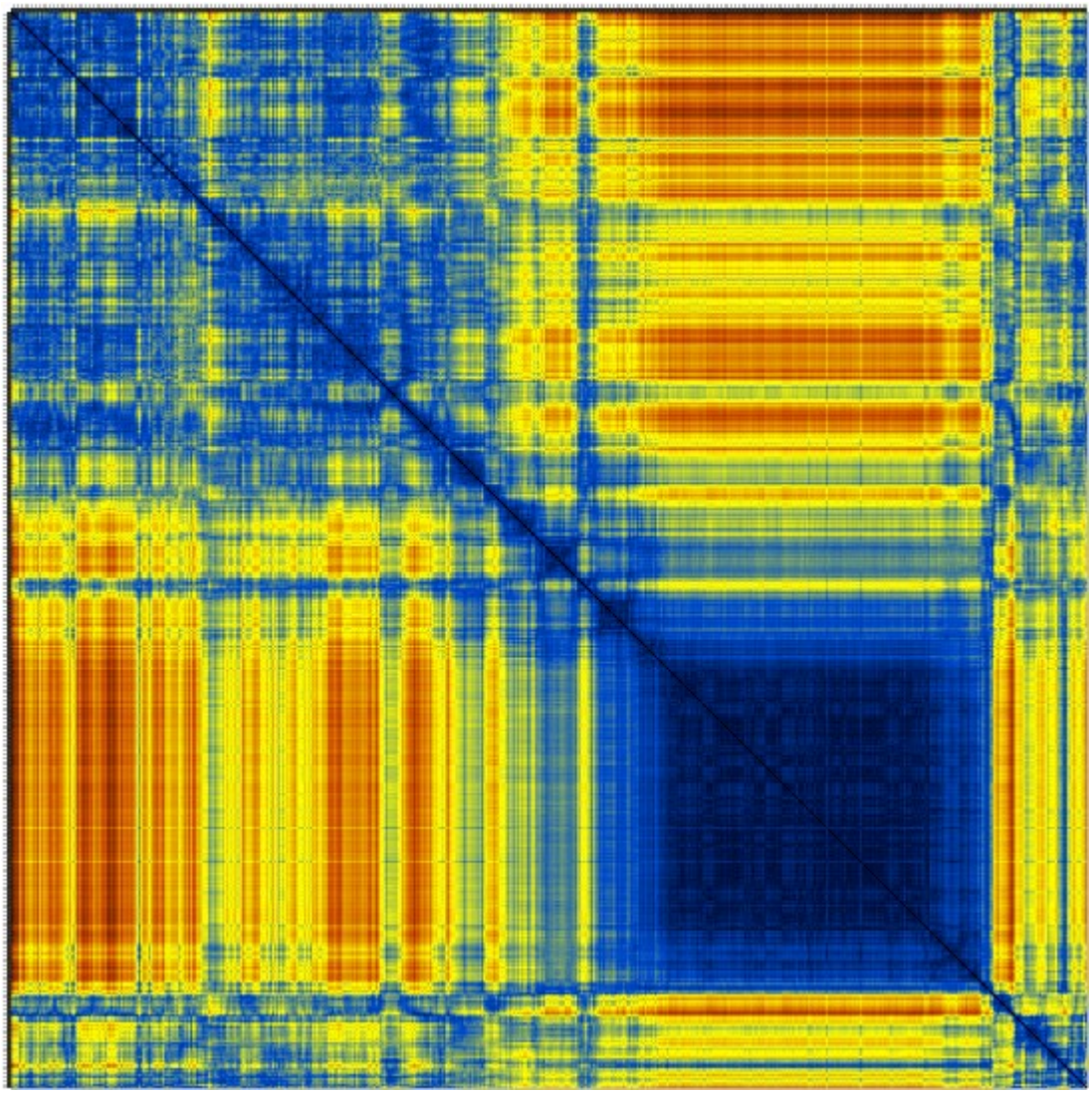
Εικόνα 2: Παράθυρο της εργασίας fitting.



Εικόνα 3: Γράφημα της απόστασης των δομών από την πρώτη

2.1.2 RMSD matrix

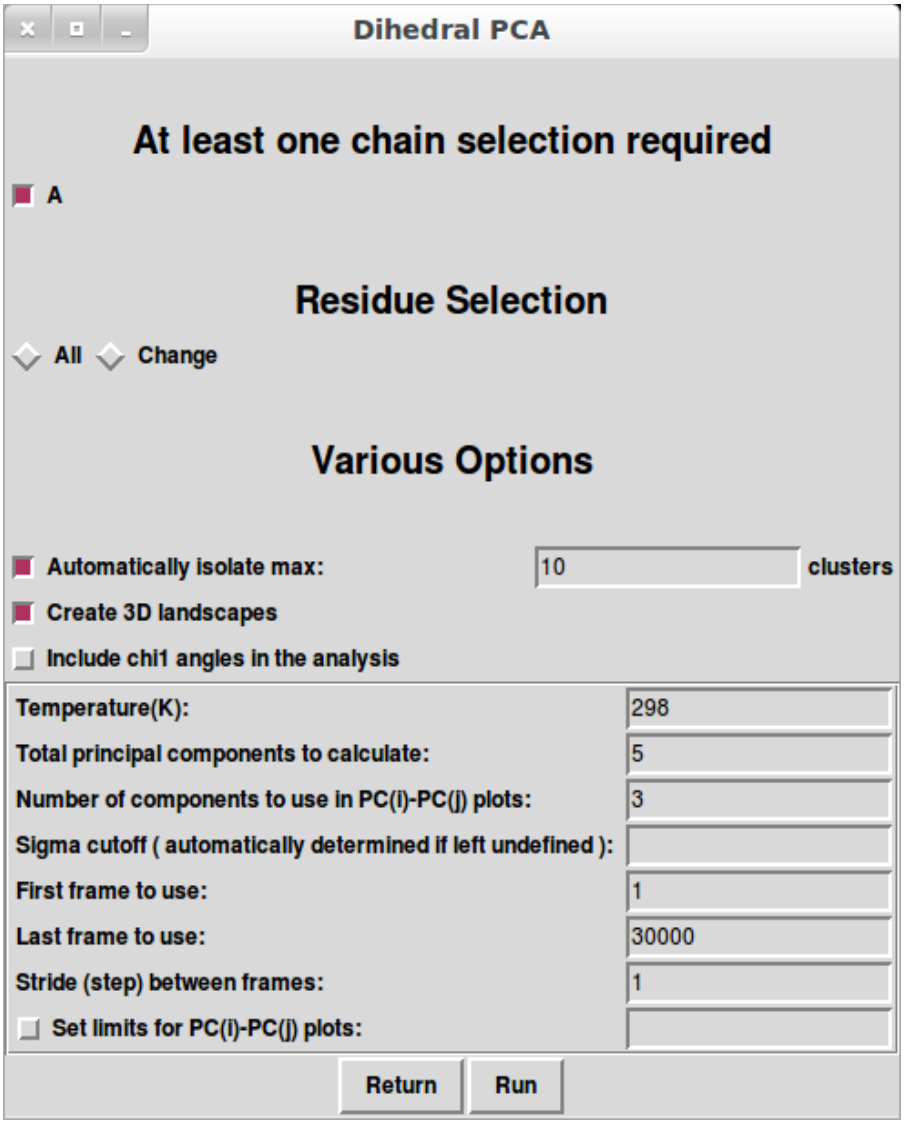
Ο πίνακας RMSD αποτελεί ένα μέτρο της σύγκρισης των δομών που υιοθετεί η πρωτεΐνη κατά την διάρκεια της προσομοίωσης, το οποίο μας επιτρέπει να αξιολογήσουμε την σταθερότητα της πρωτεΐνης. Μετά τον υπολογισμό αυτού του πίνακα το πρόγραμμα αυτόματα δημιουργεί μία γραφική αναπαράσταση του όπως αυτή που φαίνεται στην εικόνα 4. Τα μπλε σημεία αντιστοιχούν σε περιοχές με χαμηλό RMSD, ενώ τα κόκκινα σε περιοχές με υψηλό.



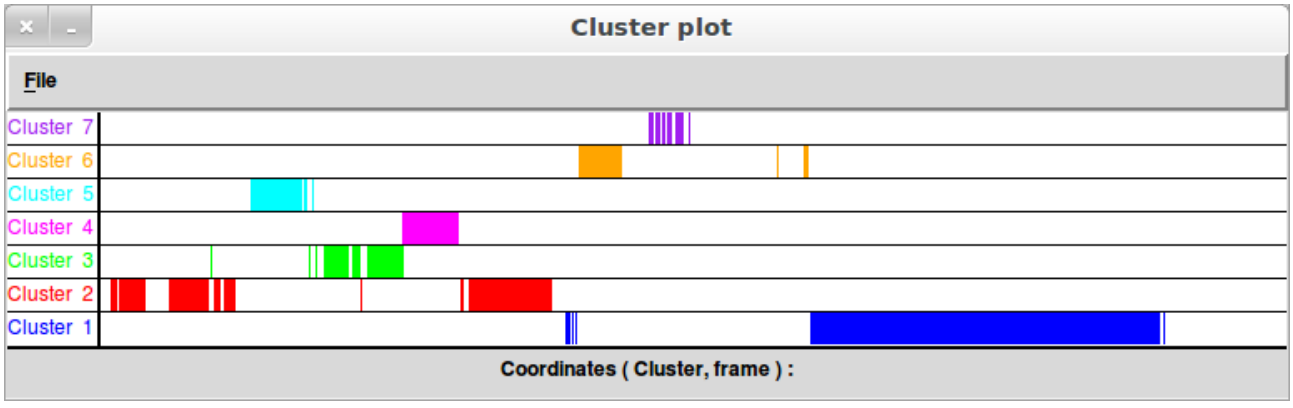
Εικόνα 4: Πίνακας RMSD Cα ατόμων

2.1.3 Dihedral & Cartesian Principal Component Analysis

Η ανάλυση PCA είναι αναπόσπαστο κομμάτι της ανάλυσης οποιουδήποτε τροχιακού ΜΔ, και για αυτόν τον λόγο έχει δοθεί ιδιαίτερη σημασία στην αυτοματοποίηση της διαδικασίας αυτής, μέσω της δυνατότητας της αυτόματης εξαγωγής clusters παρόμοιων δομών, πραγματοποίηση fitting στα backbone άτομα κάθε cluster, και τέλος εξαγωγή της μέσης, της αντιπροσωπευτικής και 500 δομών από κάθε cluster. Στην εικόνα 5 φαίνεται το παράθυρο dPCA με μερικές από τις παραμέτρους οι οποίες είναι ειδικές για τις αναλύσεις PCA, αλλά και αυτές που είναι κοινές για όλες τις αναλύσεις όπως το βήμα (stride), ή το εύρος των δομών που θα συμπεριληφθούν στην ανάλυση.

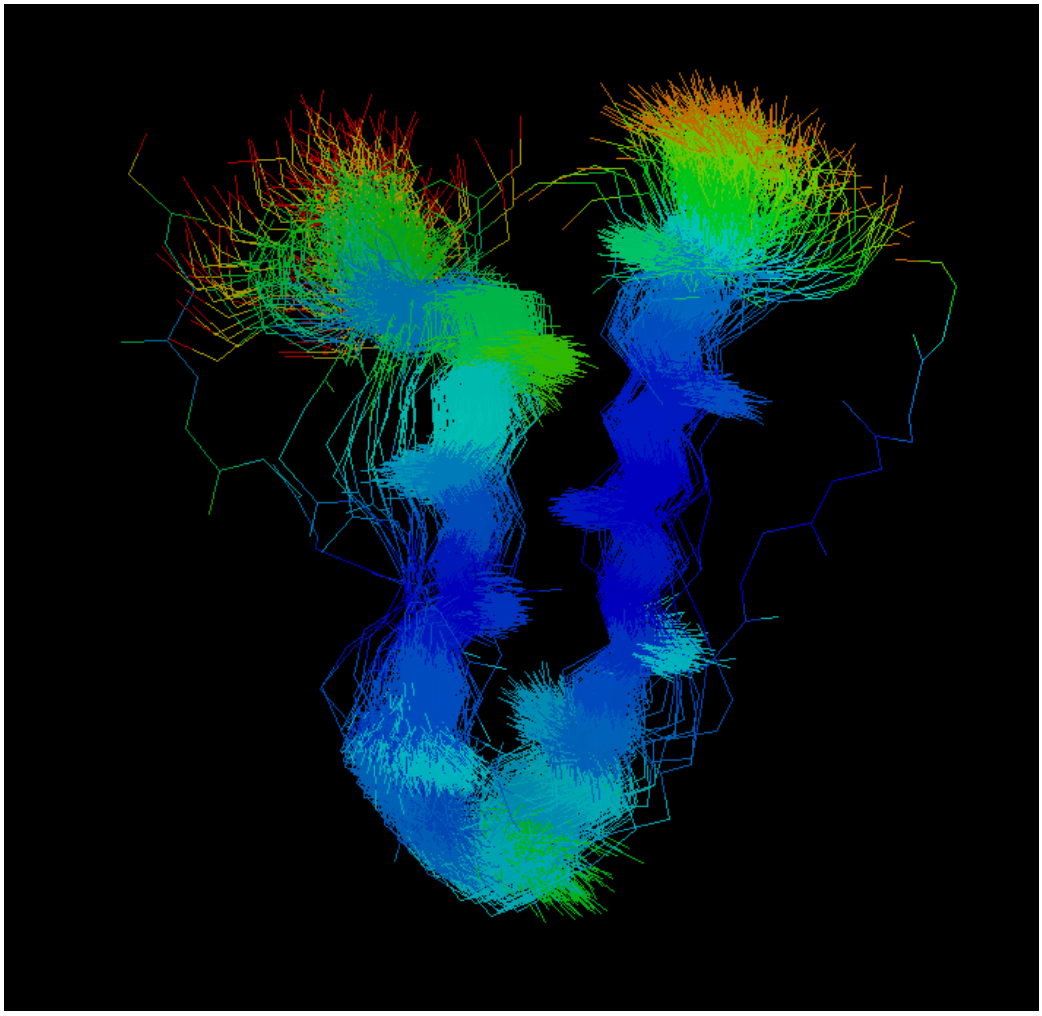


Εικόνα 5: Παράθυρο της διεργασίας dPCA



Εικόνα 6: Κατανομή των δομών σε clusters μετά από ανάλυση dPCA.

Στην εικόνα 6 φαίνεται η κατανομή των δομών για την διάρκεια της προσομοίωσης όπως προκύπτει από την ανάλυση dPCA, ενώ στην εικόνα 7 φαίνεται η υπέρθεση 500 δομών που ανήκουν στο cluster 1.



Εικόνα 7: Υπέρθεση 500 δομών του cluster 1 και χρωματισμός ανάλογα με την θερμοκρασία.

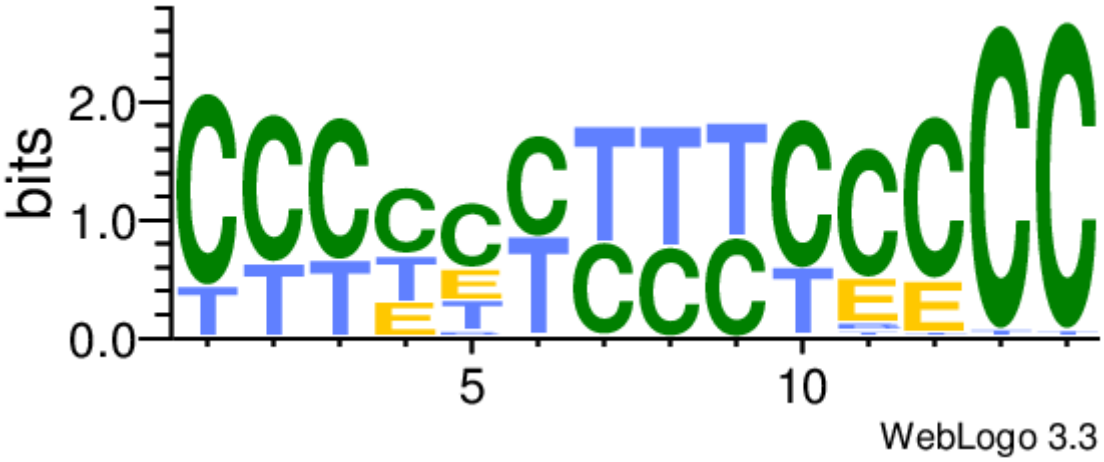
Το πρόγραμμα μοριακών γραφικών που χρησιμοποιήθηκε για την δημιουργία της εικόνας 7 είναι το RASMOL[19].

2.1.4 Covariance, average and representative structures

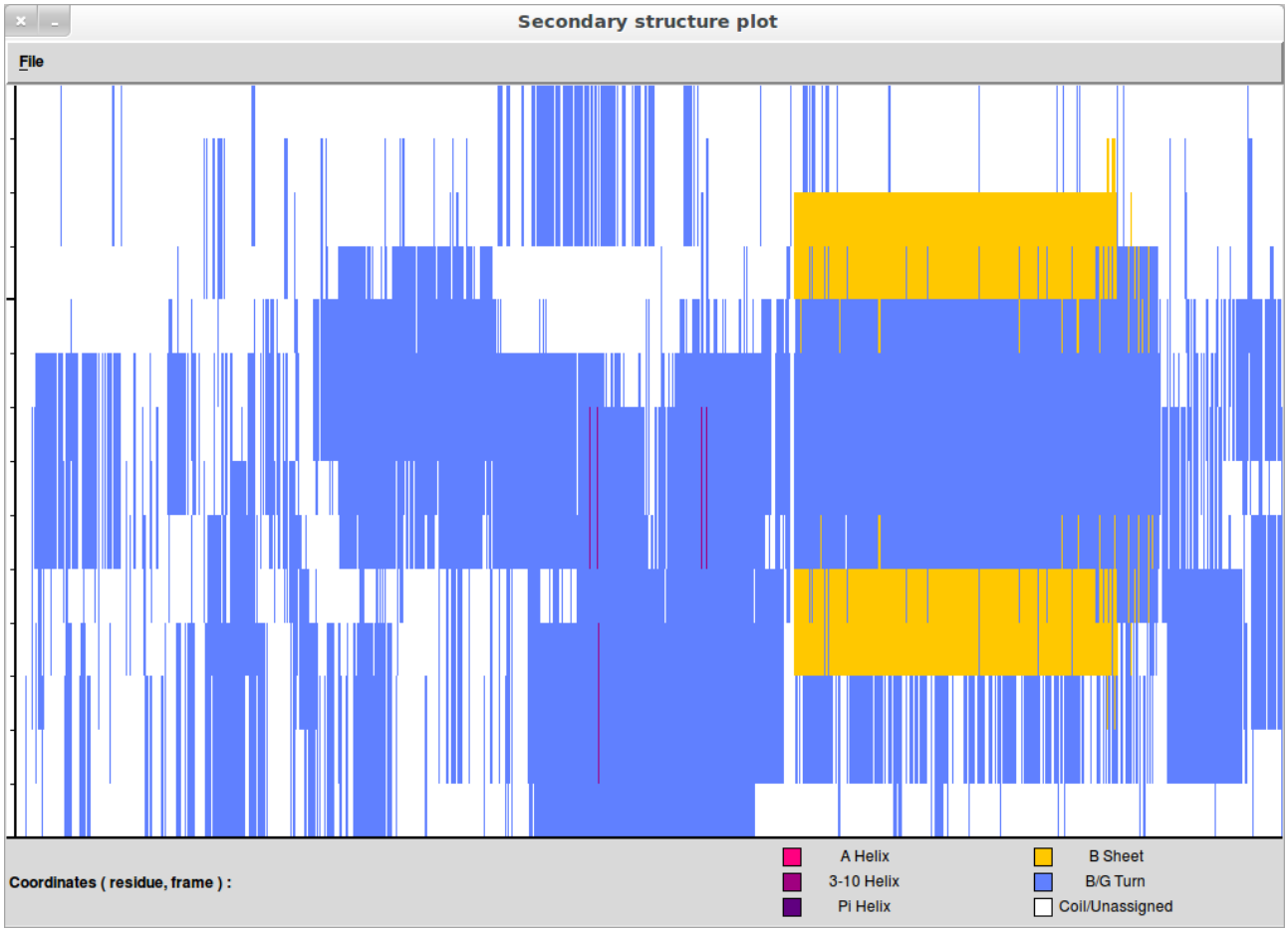
Υπολογισμός του πίνακα covariance και της μέσης, της αντιπροσωπευτικής και 500 δομών σε υπέρθεση ομοίως με το τελευταίο σκέλος των αναλύσεων PCA.

2.1.5 Secondary Structure

Στην ανάλυση αυτή πραγματοποιείται ανάθεση δευτεροταγούς δομής για όλες τις δομές του τροχιακού, μέσω του προγράμματος STRIDE[20], υπό την προϋπόθεση ότι το πρόγραμμα τρέχει σε Linux ή MACOSX, και ότι το STRIDE βρίσκεται στο PATH. Επιπλέον, αν το πρόγραμμα seqlogo/weblogo[21], βρεθεί στο PATH, θα προετοιμαστεί ένα γράφημα όπως αυτό της εικόνας 8 πέρα από το γράφημα της εικόνας 9 το οποίο ετοιμάζεται μετά από κάθε επιτυχή ολοκλήρωση της ανάλυσης καθώς δεν χρησιμοποιεί κάποιο εξωτερικό πρόγραμμα.



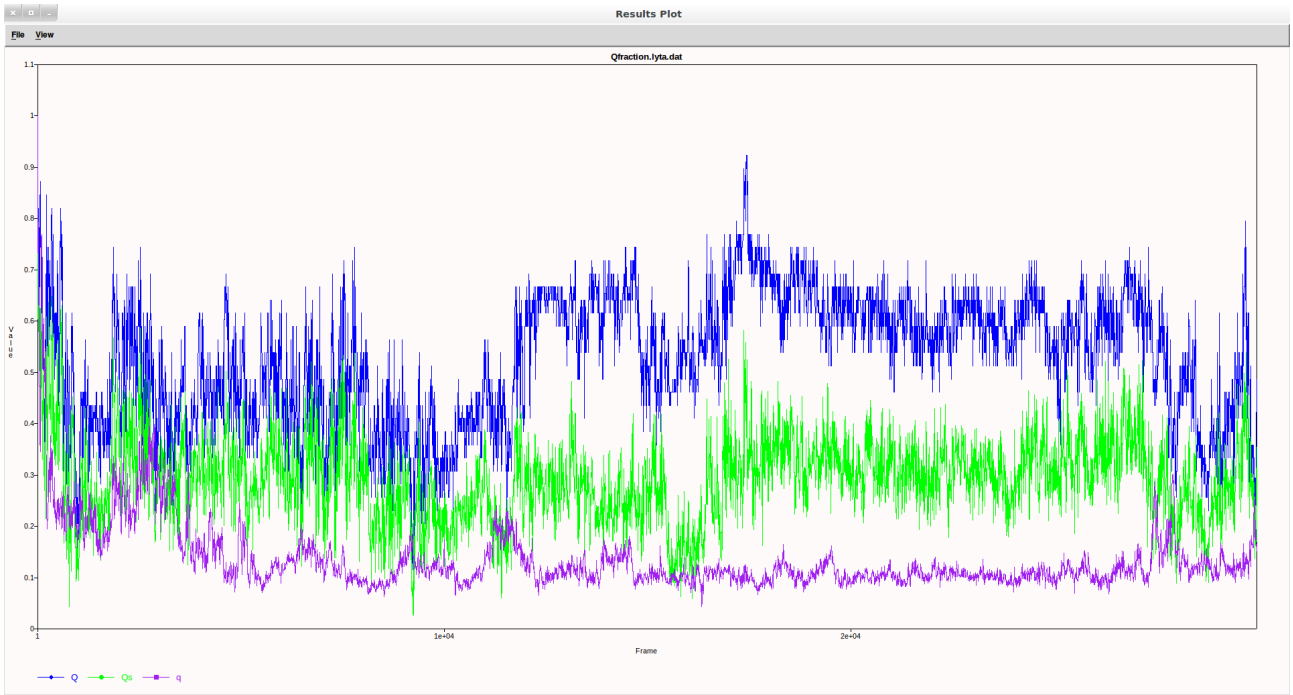
Εικόνα 8: Γράφημα δευτεροταγούς δομής από το weblogo. Το ύψος του κάθε γράμματος υποδηλώνει την συχνότητα με την οποία απαντάται η κατάσταση δευτεροταγούς δομής στην οποία αντιστοιχεί το κάθε γράμμα για την διάρκεια της προσομοίωσης. Οι συντεταγμένες στον οριζόντιο άξονα είναι ο αύξων αριθμός καταλοίπου.



Εικόνα 9: Γράφημα δευτεροταγούς δομής. Στον οριζόντιο άξονα είναι τα frames, ενώ στον κάθετο, τα κατάλοιπα. Στο κάτω δεξί μέρος εξηγείται ποια κατάσταση αντιστοιχεί σε κάθε χρώμα.

2.1.6 Fraction of native contacts

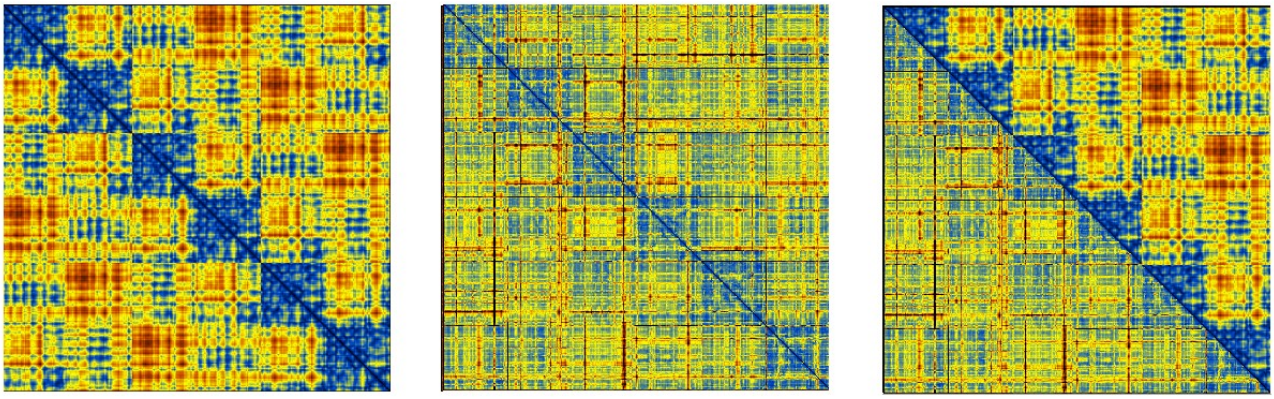
Στην ανάλυση αυτή υπολογίζονται τα native contacts όπως ορίζονται στις συμπληρωματικές πληροφορίες του άρθρου των Cho, Levy & Wolynes[21], και τα αποτελέσματα παρουσιάζονται με την μορφή του γραφήματος της εικόνας 10.



Εικόνα 10: Γράφημα των native contacts.

2.1.7 Distance Maps

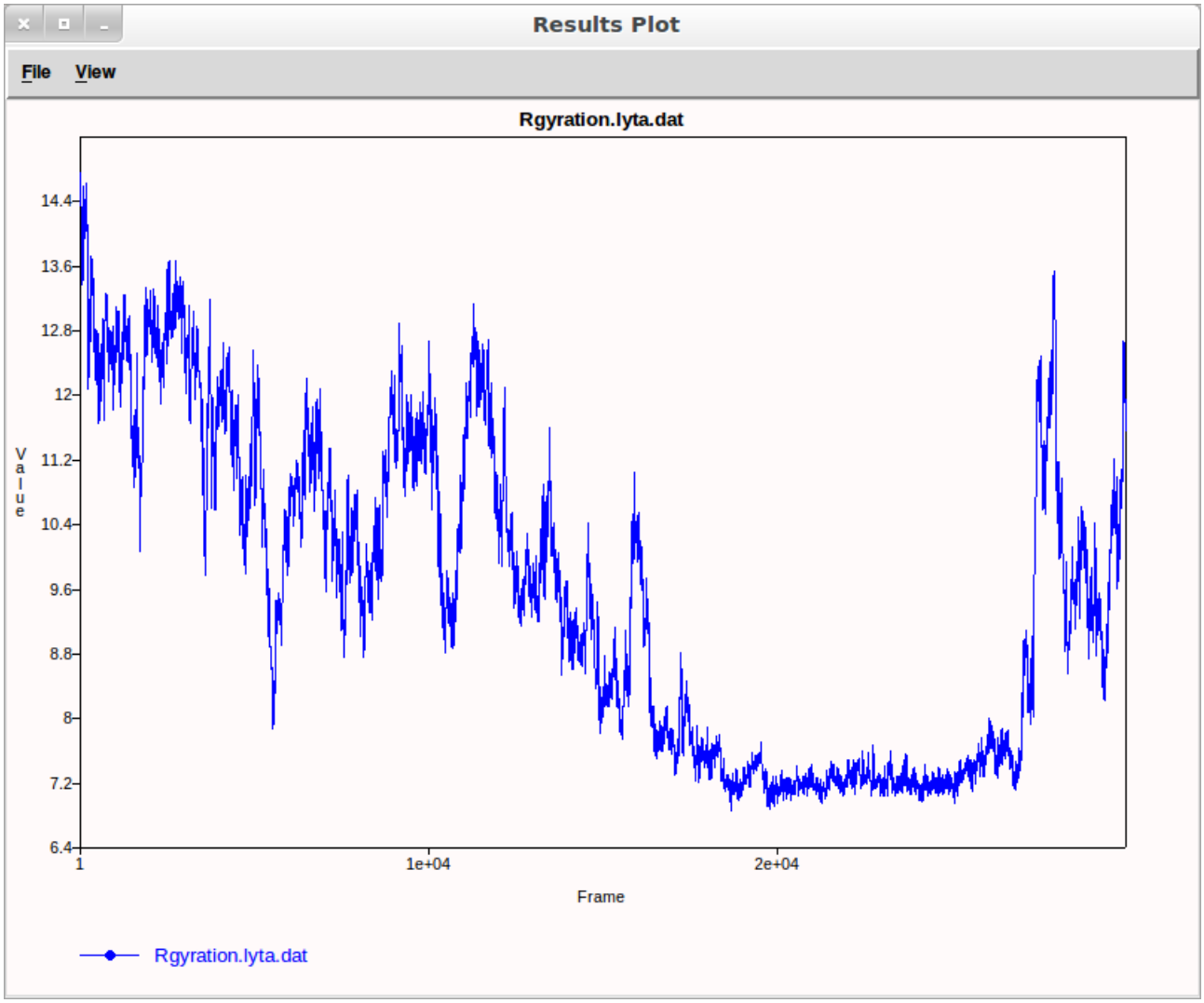
Στην ανάλυση αυτή υπολογίζονται οι πίνακες αποστάσεων ανάμεσα στα επιλεγμένα άτομα και δημιουργούνται 3 εικόνες postscript. Η πρώτη περιέχει έναν χάρτη των μέσων τιμών όλων των αποστάσεων CA-CA, η δεύτερη τις αντίστοιχες τυπικές αποκλίσεις, και η τρίτη η οποία αποτελεί συνένωση των δύο προηγούμενων για ευκολότερη σύγκριση. Και τα τρία αρχεία παρουσιάζονται στην εικόνα 11.



Εικόνα 11: Χάρτης των μέσων τιμών, των αποκλίσεων τους και σύνθεση των δύο για το εξαμερές BcZBP[25].

2.1.8 Radius of gyration

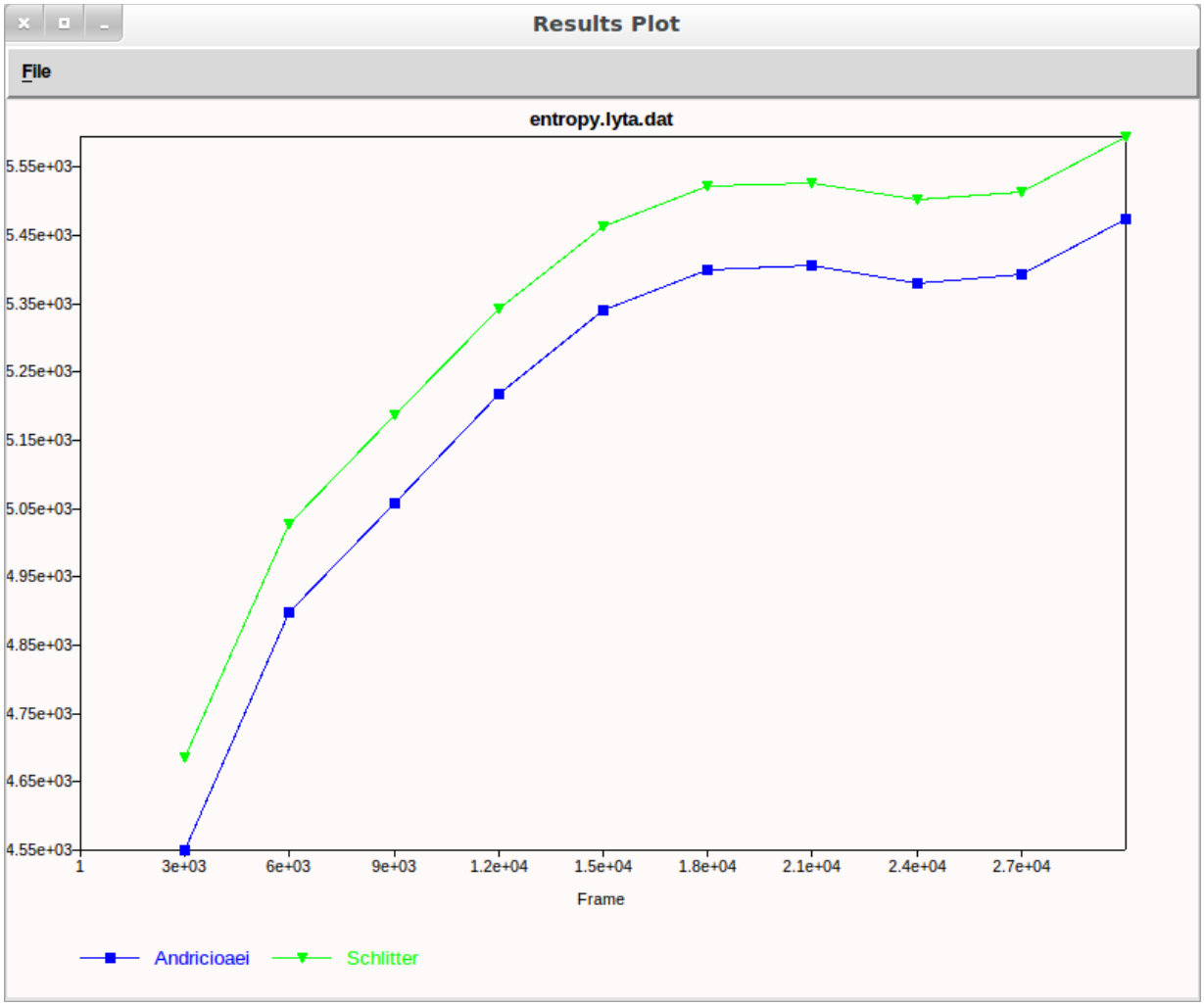
Στην ανάλυση αυτή υπολογίζεται η γυροσκοπική ακτίνα του μορίου και το αποτέλεσμα παρουσιάζεται μέσω γραφήματος όπως στην εικόνα 12.



Εικόνα 12: Γράφημα της γυροσκοπικής ακτίνας σε σχέση με τον χρόνο.

2.1.9 Entropy

Στην ανάλυση αυτή υπολογίζεται η εντροπία του μορίου με δύο τρόπους[23-24]. Αφού ο χρήστης προσδιορίσει την θερμοκρασία στην οποία θα πραγματοποιηθεί η προσομοίωση, έχει την επιλογή να ορίσει το βήμα με το οποίο θα αυξάνεται ο αριθμός των δομών που θα συμπεριλαμβάνονται στην ανάλυση έως ότου αναλύονται όλες οι δομές της προσομοίωσης. Η προεπιλεγμένη ρύθμιση θα έχει ως αποτέλεσμα την πραγματοποίηση 10 επαναλήψεων με κάθε καινούρια επανάληψη να περιέχει τις δομές της προηγούμενης επανάληψης (εκτός από την πρώτη) συν το επόμενο 10% των δομών. Τα αποτελέσματα παρουσιάζονται στην εικόνα 13.



Εικόνα 13: Γράφημα της εντροπίας σε σχέση με τον χρόνο.

2.1.10 Extract PDB(s)

Στην ανάλυση αυτή εξάγονται αρχεία PDB τα οποία εικονίζουν τα υπό ανάλυση μόρια στις στιγμές οι οποίες καθορίζονται από το αρχική και την τελευταία δομή καθώς και το βήμα της ανάλυσης.

2.1.11 Surface area

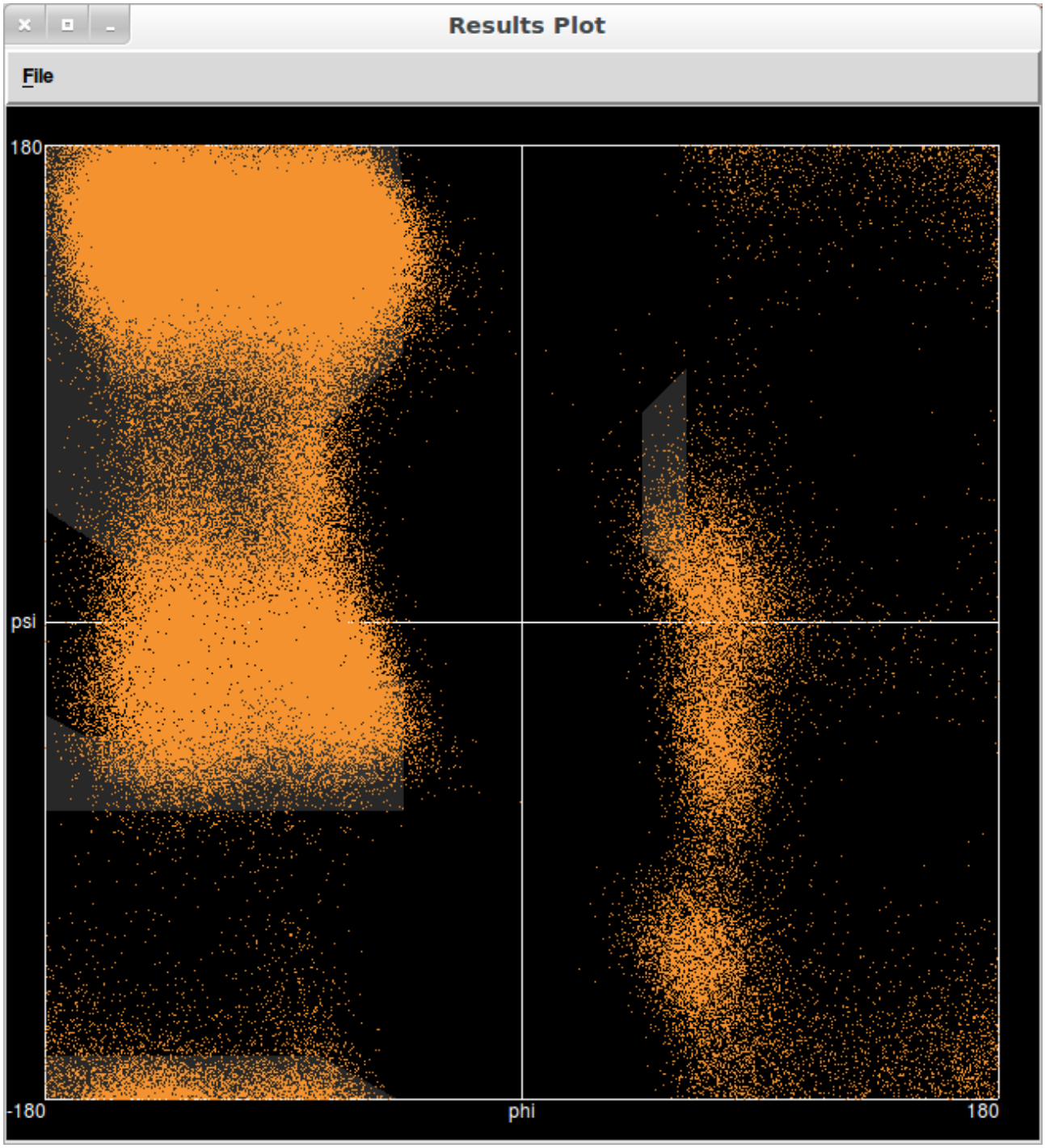
Στην ανάλυση αυτή υπολογίζεται ένα μέτρο της επιφάνειας του μορίου το οποίο ορίζεται από την επιλογή της αλυσίδας που έχει πραγματοποιήσει ο χρήστης, και το αποτέλεσμα είναι ένα γράφημα παρόμοιο με αυτά των εικόνων 3 και 12.

2.1.12 Distances, Bending angles & Torsion angles

Οι αναλύσεις υπολογισμού της απόστασης, της γωνίας και της δίεδρης γωνίας πραγματοποιούνται ανάμεσα σε 2, 3 και 4 άτομα αντίστοιχα και τα αποτελέσματα παρουσιάζονται σε διαγράμματα παρόμοια με αυτά των εικόνων 3 και 12.

2.1.13 phi/psi dihedral angles

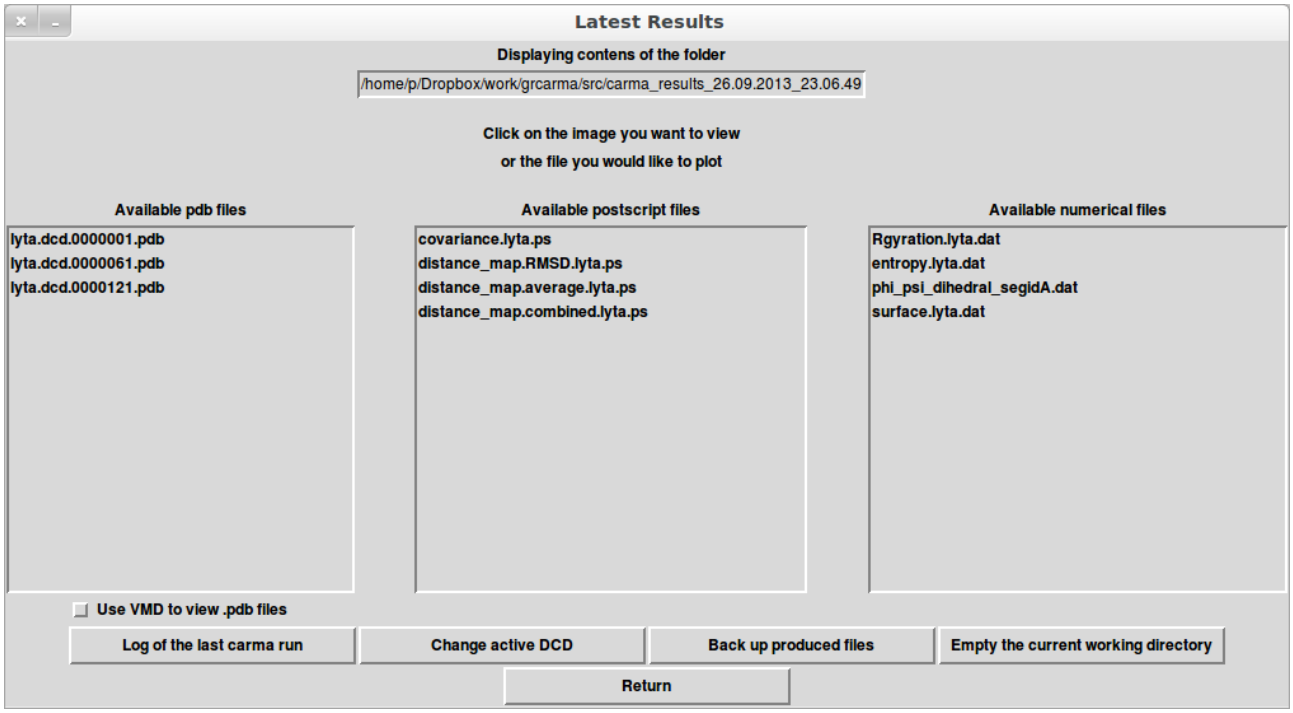
Η ανάλυση αυτή αποτελεί υποκατηγορία του υπολογισμού δίεδρων γωνιών, καθώς υπολογίζει τις δίεδρες γωνίες φ / ψ μίας αλυσίδας για όλη την διάρκεια της προσομοίωσης και προβάλλει τα αποτελέσματα σε ένα διάγραμμα Ramachandran, όπως φαίνεται στην εικόνα 14.



Εικόνα 14: Διάγραμμα Ramachandran.

2.1.14 View Results

Τέλος το κουμπί αυτό μεταφέρει τον χρήστη στο παράθυρο της εικόνας 15 από το οποίο μπορεί να επιλέξει τα αρχεία που θέλει να προβάλει, να επιλέξει άλλα αρχεία προς ανάλυση, να διαγράψει τα αρχεία τα οποία έχουν δημιουργηθεί μέχρι στιγμής, να αποθηκεύσει τα αρχεία που έχουν δημιουργηθεί μέχρι στιγμής σε κάποιον φάκελο και να προβάλει σε ένα παράθυρο το τελευταίο μήνυμα που εξέπεμψε το **carma**. Η λειτουργικότητα αυτή σκοπεύει στην απλοποίηση της διερεύνησης του τι μπορεί να απέτυχε σε κάποια ανάλυση και να επιτρέψει στον χρήστη να το διορθώσει χωρίς να κλείσει το πρόγραμμα.



Εικόνα 15: Το παράθυρο των αποτελεσμάτων.

Κεφάλαιο 3 – Πηγαίος κώδικας

Στην ενότητα αυτή παρατίθεται όλος ο πηγαίος κώδικας του προγράμματος. Πρέπει να σημειωθεί ότι η τελευταία έκδοση του προγράμματος είναι πάντα διαθέσιμη μέσω των ιστοσελίδων https://github.com/pkoukos/grcarma και http://sourceforge.net/projects/grcarma.

```
#!/usr/bin/env perl


=head1 NAME

grcarma - GUI to molecular dynamics trajectories analysis program carma

=head1 SYNOPSIS

grcarma [ PSF FILE ] [ DCD FILE ]

grcarma.exe [ PSF FILE ] [ DCD FILE ]

=head1 DESCRIPTION

grcarma is a GUI to molecular dynamics trajectories analysis program
B<carma>. It is written in Perl and makes use of the Tk module for
graphics. It is available for Linux and Windows, and requires the
carma executable in the same folder ( or in the PATH ). As seen in
the synopsis the program can be launched with a .psf / .dcd pair of
files as arguments. Alternatively, the program can be run without any
arguments and the user will be prompted to specify the files to use
for the analyses, through a graphical interface.

=head1 AUTHOR

grcarma has been developed by Panagiotis Koukos, under the supervision
of L<Prof. Nicholas M. Glykos|http://utopia.duth.gr/~glykos/Carma.html>
at the L<Department of Molecular Biology and Genetics|http://mbg.duth.gr/index.en.shtml>
of L<Democritus University of Thrace|http://www.duth.gr/index.en.sxhtml>.

=head1 SEE ALSO

For more information, see L<https://github.com/pkoukos/grcarma>

=head1 LICENSE

Copyright (c) 2012-2013, Panagiotis I. Koukos

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

=over

=item 1.

Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

=item 2.

Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following
disclaimer in the documentation and/or other materials
provided with the distribution.

=back

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

```
=cut

use strict;
use warnings;

# Use the Tk module for the GUI        #

use Tk;
use Tk::MsgBox;
require Tk::PlotDataset;
require Tk::LineGraphDataset;
require Tk::BrowseEntry;
require Tk::ROText;
require Tk::Dialog;

# Import the following core modules    #

use Cwd;
use Cwd 'abs_path';
use File::Path 'mkpath';
use File::Copy 'cp', 'mv';

use List::MoreUtils 'uniq';
use List::Util 'min', 'max';

# Get the system time and modify it so #
# that it is human readable            #
# Find the OS type and store it in a   #
# variable                             #

my $launch_dir = getcwd;

my @now = localtime();
my   $timeStamp   =   sprintf(   "carma_results_%02d.%02d.%04d_%02d.%02d.%02d",   $now[3],   $now[4]+1,
$now[5]+1900, $now[2], $now[1], $now[0] );

my ( $windows, $linux, $mac, ) = ( '', '', '', );

if ( $^O eq 'MSWin32' ) {
    $windows = 1;
}
elsif ( $^O eq 'linux' ) {
    $linux = 1;
}
elsif ( $^O eq 'darwin' ) {
    $mac = 1;
}

my $active_psf = '';
my $active_dcd = '';

# Declaration of the global scalars... #

our $VERSION = '0.1';

our $flag = '';
our $custom_id_flag = '';
our $seg_id_flag = '';
our $index_seg_id_flag = '';
our $custom_selection = '';
our $all_done = '';
#~ our $num_atoms = '';
our $psf_button = '';
our $dcd_button = '';
our $have_psf = 0;
our $have_dcd = 0;
our $have_files = '';
our $filetypes = '';
our $have_custom_psf = '';
our $dcd_count = -1;
our $atm_id_flag = '';
our $res_id_flag = '';
our $header = '';
our $atm_id = '';
```

```
our $new_psf;
our $active_run_buttons = '';
our $active_run_buttons_qfract = '';
our $eig_play = '';
our $eig_play_vector = '';
our $eig_art = '';
our $eig_art_vectors = '';
our $eig_art_frames = '';
our $top_eig = '';
our $frame_eig1 = '';
our $resid_bar_count = 0;
our $index_bar_count = '';
our $resid_row = '';
our $resid_column = '';
our $index_row = '';
our $index_column = '';
our $f4_b = '';
our $dpca_run_button = '';
our $surf_run_button = '';
our $index_num_atoms = '';
our $dpca_frame = '';
our $dpca_frame_1 = '';
our $frame_sur2 = '';
our $cpca_frame = '';
our $cpca_frame_1 = '';
our $qfract_run_button;
our $phi_psi_run_button;
our $frame_qfract2;
our $frame_phi_psi1;

my $cancel_all_segids = 0;

# and global arrays and hashes         #

our (
     @seg ids,           @unique chain ids,      @unique atom types,
     @dropdown,          @amplitudes,            @dropdown_value,
     @upper_res_limit,   @lower_res_limit,       @upper_fit_limit,
     @lower_fit_limit,   @frame_res1,            @frame_fit_index4,
     %num_residues,      %substitutions,         @frame_fit6,
     @fit_drop,          @fit_drop_value,        @frame_fit_bars,
);

# If the OS is *nix/unix-like and the  #
# the folder "carma_results exists in  #
# the current working directory get    #
# it's size and store it in a scalar   #

my $wd_size = '';

my $ps_viewer = '';
my $pdb_viewer ='';
my $vmd = 0;
my $stride = 0;
my $weblogo = 0;
my $seqlogo = 0;
my $terminal = 0;

my $count = 0;

if ( $linux || $mac ) {
    while ( my ( $key, $value ) = each(%ENV) ) {
        if ( $key eq 'GRCARMA_PS_VIEWER' ) {
            if ( `which $value 2> /dev/null` ) {
                $ps_viewer = $value;
            }
        }

        if ( $key eq 'GRCARMA_PDB_VIEWER' ) {
            if ( `which $value 2> /dev/null` ) {
                $pdb_viewer = $value;
            }
        }
    }
```

```
    # If any of the following programs #
    # is found in the /usr/bin folder  #
    # set it as the default .ps file   #
    # viewer                           #
    if ( $linux ) {
        if ( not $ps_viewer ) {
            if ( `which evince 2> /dev/null` ) {
                $ps_viewer = "evince";
            }
            elsif ( `which gv 2> /dev/null` ) {
                $ps_viewer = "gv";
            }
            elsif ( `which gs 2> /dev/null` ) {
                $ps_viewer = "gs";
            }
            elsif ( `which display 2> /dev/null` ) {
                $ps_viewer = "display";
            }
        }
    }
    else {
        if ( not $ps_viewer ) {
            if ( not `open -a Preview` ) {
                $ps_viewer = 'open -a Preview';
            }
            elsif ( `which gs` ) {
                $ps_viewer = 'gs';
            }
        }
    }

    chomp $ps_viewer if ( $ps_viewer );

    if ( not $pdb_viewer ) {
        if ( `which rasmol 2> /dev/null` ) {
            $pdb_viewer = 'rasmol';
        }
        elsif ( `which jmol 2> /dev/null` ) {
            $pdb_viewer = 'jmol';
        }
        elsif ( `which pymol 2> /dev/null` ) {
            $pdb_viewer = 'pymol';
        }
        elsif ( `which vmd 2> /dev/null` ) {
            $pdb_viewer = 'vmd';
        }
    }

    chomp $pdb_viewer if ( $pdb_viewer );

    if ( `which vmd 2> /dev/null` ) {
        $vmd = 1;
    }
    if ( `which stride 2> /dev/null` ) {
        $stride = 1;
    }
    if ( `which weblogo 2> /dev/null` ) {
        $weblogo = 1;
    }
    elsif ( `which seqlogo 2> /dev/null` ) {
        $seqlogo = 1;
    }

    if ( `which xterm 2> /dev/null` ) {
        $terminal = 'xterm';
    }
    elsif ( `which rxvt 2> /dev/null` ) {
        $terminal = 'rxvt';
    }
    elsif ( `which gnome-terminal 2> /dev/null` ) {
        $terminal = 'gnome-terminal';
    }

}
```

```
my $font_12 = "helvetica 13";
my $font_20 = "helvetica 15 bold";

# check for input from terminal        #
# if two files are specified           #
# and they are a DCD and a PSF file    #

my $run_from_terminal = 0;
my $ramachandran_popup = 0;
my $psf_file = '';
my $dcd_file = '';
my $dcd_name = '';
my $psf_name = '';
my $dcd_loc = '';

my $file_selection_window;
my $carma_version = '';

###############################################################################################
###   Main Window                                                                           ###
###############################################################################################

# Draw the main window                 #
my $mw = MainWindow -> new( -title => 'grcarma', );

if ( $linux or $mac ) {
    if ( not `which carma` and not `which carma64` ) {
        $mw -> messageBox( -font => "$font_12", -message => 'No carma executable found in the path.
Aborting', );
        exit;
    }
}
if ( $linux or $mac ) {
    if ( `carma` =~ /carma v.(\d.\d)/ ) {
        $carma_version = $1;
    }
}
else {
    if ( `carma.exe` =~ /carma v.(\d.\d)/ ) {
        $carma_version = $1;
    }
}

$mw -> maxsize( $mw -> screenwidth, $mw -> screenheight - 50, );
$mw -> minsize( 850, 705, );
$mw -> protocol( 'WM_DELETE_WINDOW' => sub { kill -2, $$; } );
$mw -> withdraw;

if ( @ARGV ) {
    if ( $linux or $mac ) {
        if ( not `which carma` and not `which carma64` ) {
            die "\nNo carma executable found in the path. Aborting.\n\n";
        }
        else {
            if ( `carma` =~ /carma v.(\d.\d)/ ) {
                $carma_version = $1;
            }
        }
    }

    if ( @ARGV == 2 ) {
        # regardless of which was specified    #
        # first store each of them in a file   #
        # and store the name of the .dcd file  #
        # in a variable                        #
        if ( $ARGV[0] =~ /.*\.psf/ && $ARGV[1] =~ /.*\.dcd/ ) {
            $psf_file = abs_path( $ARGV[0] );
            $dcd_file = abs_path( $ARGV[1] );
            if ( $dcd_file =~ /(.*)(\/|\\)(.*)\.dcd/ ) {
                $dcd_loc = $1;
                $dcd_name = $3;
                $active_dcd = $3 . '.dcd';
            }
```

```
            elsif ( $dcd_file ) {
                die "Unexpected character in the DCD name. Aborting.\n";
            }

            if ( $psf_file =~ /(.*)(\/|\\)(.*)\.psf/ ) {
                $psf_name = $3;
                $active_psf = $3 . '.psf';
            }
            elsif ( $psf_file ) {
                die "Unexpected character in the PSF name. Aborting.\n";
            }
        }
        elsif ( $ARGV[1] =~ /.*\.psf/ && $ARGV[0] =~ /.*\.dcd/ ) {
            $psf_file = abs_path( $ARGV[1] );
            $dcd_file = abs_path( $ARGV[0] );
            if ( $dcd_file =~ /(.*)(\/|\\)(.*)\.dcd/ ) {
                $dcd_loc = $1;
                $dcd_name = $3;
                $active_dcd = $3 . '.dcd';
            }
            if ( $psf_file =~ /(.*)(\/|\\)(.*)\.psf/ ) {
                $psf_name = $3;
                $active_psf = $3 . '.psf';
            }
        }
        # or terminate with a help message     #
        else {
            die "\nUsage: grcarma file.psf file.dcd\n\n";
        }

        # remember that files were specified   #
        # from STDIN and invoke the PSF parser #
        # subroutine                           #
        $run_from_terminal = 1;
        folder_size ( $launch_dir );
        parser();
    }
    else {
        # else if number of files specified is #
        # not 2 terminate with a help message  #
        die "\nPlease specify one .psf and one .dcd file\n\n";
    }
}
else {
    # Create the frame for the selection   #
    # of files                             #
    $file_selection_window = MainWindow -> new( -title => 'File Selection', );
     $file_selection_window -> protocol( 'WM_DELETE_WINDOW' => sub { $file_selection_window ->
destroy; $mw -> destroy; kill -2, $$; } );
     my $width_position = int ( ( $file_selection_window -> screenwidth / 2 ) - (
$file_selection_window -> width / 2 ) );
    my $height_position = int ( ( ( $file_selection_window -> screenheight - 80 ) / 2 ) - (
$file_selection_window -> height / 2 ) );
    my $file_selection_window_position = "+" . $width_position . "+" . $height_position;

    $file_selection_window -> geometry( "$file_selection_window_position" );

    $file_selection_window -> Label( -text => 'Please select a .psf and a .dcd file', ) -> pack;

    # Draw the button for psf selection    #
    $psf_button = $file_selection_window -> Button( -text => 'Browse for a .psf file',
                                    -command => sub {
                                        open_file ( "psf" );
                                    },
                                    -font => "$font_12", )
                                    -> pack( -side => 'left' );

    # Draw the button for dcd selection    #
    $dcd_button = $file_selection_window -> Button( -text => 'Browse for a .dcd file',
                                    -command => sub {
                                        open_file ( "dcd" );
                                    },
                                    -font => "$font_12", )
                                    -> pack( -side => 'right' );
```

```
    $mw -> waitVariable(\$have_files);
}

###############################################################################################################
###   Container Frame                                                                                       ###
###############################################################################################################

# Create the first frame ( container ) #
my $f0 = $mw -> Frame();

###############################################################################################################
###   File Menu                                                                                             ###
###############################################################################################################

# Create the menubar                   #
$mw -> configure( -menu => my $menubar = $mw -> Menu );

# Create the menubutton "File" and the #
# menubutton "Help"                    #
my $file = $menubar -> cascade( -font => "$font_12", -label => '~File');
my $help = $menubar -> cascade( -font => "$font_12", -label => '~Help');

# Draw a separating line               #
$file -> separator();

# Create a command of the "file" menu- #
# button which terminates the program  #
$file -> command( -label => "Exit",
                  -underline => 1,
                  -font => "$font_12",
                  -command => [ $mw => 'destroy' ], );
$help -> command( -label => 'About',
                  -font => "$font_12",
                  -command => \&about, );

###############################################################################################################
###   Menubutton Frame                                                                                      ###
###############################################################################################################

# Draw the second frame ( menubuttons) #
# on top of the first one              #
my $f1 = $f0 -> Frame () -> pack ( qw/ -side left -expand 1 -fill y/ );

# ... the fitting menu                 #
my $fitting_menu = $f1 -> Button( -text => 'Fitting',
                                  -command => \&fit_window,
                                  -width => 24,
                                  -font => "$font_12", ) -> pack;

#Draw the button for the rmsd menu...  #
my $rmsd_menu = $f1 -> Button( -text => 'RMSD Matrix',
                               -command => \&rmsd_window,
                               -width => 24,
                               -font => "$font_12", ) -> pack;

# ... the dpca menu                    #
my $dpca_menu = $f1 -> Button( -text => 'Dihedral PCA',
                               -command => \&dpca_window,
                               -width => 24,
                               -font => "$font_12", ) -> pack;

# ... the cpca menu                    #
my $cpca_menu = $f1 -> Button( -text => 'Cartesian PCA',
                               -command => \&cpca_window,
                               -width => 24,
                               -font => "$font_12", ) -> pack;

# ... the var_covar matrix menu        #
my $varcov_menu = $f1 -> Button( -text => "Covariance, average and\nrepresentative structures",
                                 -command => \&cov_avg_rep_window,
                                 -width => 24,
                                 -font => "$font_12", ) -> pack;
```

```
# ... the secondary structure menu     #
my $stride_menu;
if ( $linux or $mac ) {
    $stride_menu = $f1 -> Button( -text => "Secondary structure",
                                  -command => \&stride_window,
                                  -state => 'disabled',
                                  -width => 24,
                                  -font => "$font_12", ) -> pack;
}

#Draw the button for the qfract menu...#
my $qfract_menu = $f1 -> Button( -text => 'Fraction of native contacts',
                                 -command => \&qfract_window,
                                 -width => 24,
                                 -font => "$font_12", ) -> pack;

# ... the average distances menu      #
my $rms_menu = $f1 -> Button( -text => 'Distance maps',
                              -command => \&rms_window,
                              -width => 24,
                              -font => "$font_12", ) -> pack;

# ... the gyration menu               #
my $rgr_menu = $f1 -> Button( -text => 'Radius of gyration',
                              -command => \&rgr_window,
                              -width => 24,
                              -font => "$font_12", ) -> pack;

# ... the entropy menu                #
my $entropy_menu = $f1 -> Button( -text => 'Entropy',
                                  -command => \&entropy_window,
                                  -width => 24,
                                  -font => "$font_12", ) -> pack;

# ... the pdb menu                    #
my $pdb_menu = $f1 -> Button( -text => 'Extract PDB(s)',
                              -command => \&pdb_window,
                              -width => 24,
                              -font => "$font_12", ) -> pack;

# ... the surface area menu           #
my $sur_menu = $f1 -> Button( -text => 'Surface area',
                              -command => \&sur_window,
                              -width => 24,
                              -font => "$font_12", ) -> pack;

# ... the map menu                    # implemented but not in use
#my $map_menu = $f1 -> Button( -text => 'Ion-water distribution',
#                              -command => \&map_window,
#                              -width => 24,
#                              -font => "$font_12", ) -> pack;

# ... the distances menu              #
my $dis_menu = $f1 -> Button( -text => 'Distances',
                              -command => \&dis_window,
                              -width => 24,
                              -font => "$font_12", ) -> pack;

# ... the bending angles menu         #
my $ang_menu = $f1 -> Button( -text => 'Bending Angles',
                              -command => \&bnd_window,
                              -width => 24,
                              -font => "$font_12", ) -> pack;

# ... the torsion angles menu         #
my $tor_menu = $f1 -> Button( -text => 'Torsion Angles ( general )',
                              -command => \&tor_window,
                              -width => 24,
                              -font => "$font_12", ) -> pack;

# ... the phi-psi angles menu         #
my $phi_psi_menu = $f1 -> Button( -text => 'phi/psi dihedral angles',
                                  -command => \&phi_psi_window,
                                  -width => 24,
```

```
                                -font => "$font_12", ) -> pack;

$f1 -> Label( -text => "\n", ) -> pack( -side => 'top', );

# and the exit menu                     #
my $exit_menu = $f1 -> Button( -text => 'EXIT',
                               -width => 24,
                               -command => \&exit,
                               -font => "$font_12", ) -> pack( -side => 'bottom', );

# ... the image menu                    #
our $image_menu = $f1 -> Button( -text => 'View Results',
                                 -command => [ \&image_window ],
                                 -width => 24,
                                 -font => "$font_12",
                                 -state => 'disabled', ) -> pack( -side => 'bottom', );

###############################################################################################
###   Textbox Frame                                                                         ###
###############################################################################################

# Draw the sixth frame(textbox) on top #
# of the first one and immediately     #
# after the fifth frame is drawn       #
my $f5 = $f0 -> Frame() -> pack( -after => $f1, -side => 'top', -fill => 'both', -expand => 1, );

our $text = $f5 -> ROText(
                          -bg => 'black',
                          -fg => 'white',
                          -font => "courier 12",
                          -wrap => 'word',
                          -width => 90,
                          -height => 32, ) -> pack();

#~ $text -> configure( -height => 41, ) if ( $mac || $linux );
#~ $text -> configure( -width => 85, ) if ( $windows );

# Define colored text tags to be used for the text displayed
# in the textbox
$text -> tagConfigure( "error", -foreground => "red3" );
$text -> tagConfigure( "valid", -foreground => "green3" );
$text -> tagConfigure( "info" , -foreground => "orange1" );
$text -> tagConfigure( "cyan" , -foreground => "cyan" );

$text -> tagConfigure( 'center', -justify => 'center', -foreground => 'red3', );

# Also tie the STDOUT to the textbox   #
#~ tie *STDOUT, 'Tk::Text', $text;

if ( $carma_version and $carma_version ne '1.3' ) {
    $text -> insert( 'end', "\nDetected old carma version ($carma_version).\ngrcarma is" .
                            " designed to run with the latest version of carma.\nPlease" .
                            " consider upgrading.\n", 'center' );
}

$text -> insert( 'end', "\n\nSELECT A TASK FROM THE LEFT PANEL\n\n\n" );

# If OS is *nix/unix-like insert a     #
# line informing the user of the prog  #
# selected for .ps viewing             #
if ( $linux || $mac  ) {
    if ( $pdb_viewer ) {
        $text -> insert( 'end', "* The program selected for PDB viewing is \"$pdb_viewer\".\n",
'info' );
    }
    else {
        $text -> insert( 'end', "* No PDB viewer detected. PDB file viewing disabled.\n", 'error' );
    }

    if ( $ps_viewer ) {
         $text   ->   insert(   'end',   "*   The   program   selected   for   postscript   viewing
is \"$ps_viewer\".\n", 'info' );
    }
    else {
```

```
        $text -> insert( 'end', "* No postscript viewer detected. Postscript file viewing
disabled.\n", 'error' );
    }

    if ( $stride ) {
        $text -> insert( 'end', "* stride executable located. Secondary structure analysis
enabled.\n", 'info' );
        $stride_menu -> configure( -state => 'normal', );

        if ( $seqlogo or $weblogo ) {
            $text -> insert( 'end', "* Weblogo executable located. Representations of sec. structure
enabled.\n", 'info' );
        }
        else {
            $text -> insert( 'end', "* No weblogo executable found. Representations of sec.
structure disabled.\n", 'error' );
        }
    }
    else {
        $text -> insert( 'end', "stride executable not located. Secondary structure analysis
disabled.\n", 'error' );
    }

    if ( $terminal and $linux ) {
        $text -> insert( 'end', "* $terminal executable located. Carma will run in a separate
terminal.\n", 'info' );
    }
    elsif ( $linux ) {
        $text -> insert( 'end', "* No x terminal emulator located. Carma will not run in a separate
terminal.\n", 'error' );
    }

    if ( $vmd and $pdb_viewer ne 'vmd' ) {
        $text -> insert( 'end', "* VMD executable located. Plotting of CNS map files enabled.\n",
'info' );
    }
    elsif ( not $vmd ) {
        $text -> insert( 'end', "* VMD executable not located. Plotting of CNS map files
disabled.\n", 'error' );
    }
}

$text -> insert( 'end', "\nWarning : The total size of carma_results\nfolder(s) in the current
directory is $wd_size MB\n", 'center', ) if ( $wd_size > 100 );

###############################################################################################
###   Active file frame                                                                     ###
###############################################################################################

# Draw the seventh frame(active files) #
# on top of the first one immediately  #
# after the fifth frame is drawn       #
my $f6 = $f0 -> Frame() -> pack( -after => $f1, -side => 'bottom', -fill => 'none', -expand => 0, );
my $f7 = $f0 -> Frame() -> pack( -after => $f1, -side => 'bottom', -fill => 'none', -expand => 0, );

# Create the labels displaying the     #
# active .psf & .dcd files and update  #
# the mainwindow to include them       #
our $active_psf_dcd_label = $f6 -> Label( -text => "Active PSF-DCD : ", -font => "$font_12", )
                                        -> pack( -side => 'left', );
our $active_psf_label = $f6 -> Label( -text => "$active_psf ", -fg => 'blue', -font => "$font_12", )
                                    -> pack( -side => 'left', );
our $active_dcd_label = $f6 -> Label( -text => "$active_dcd", -fg => 'blue', -font => "$font_12", )
                                    -> pack( -side => 'left', );

my $go_back_button = $f7 -> Button( -text => 'Go back to original PSF/DCD', -state => 'disabled',
-command => sub {
    our @other;
    $other[0] -> invoke if ( Exists( $other[0] ) );

    $active_psf = $psf_name . '.psf';
    $active_dcd = $dcd_name . '.dcd';

    ( $atm_id_flag, $res_id_flag, $custom_id_flag, $count, ) = ( '', '', '', 0, );
```

```
    undef @unique_chain_ids;
    undef @seg_ids;
    undef %num_residues;

    my $x = $mw -> width;
    my $y = $mw -> height;

    $f0 -> packForget;
    parser( $active_psf, $active_dcd, );
    $f0 -> pack( -side => 'top', -fill => 'both', -expand => 1, );
    $mw -> geometry( "$x" . 'x' . "$y" );
    $mw -> update;

    $active_psf_label -> configure( -text => "$active_psf ", );
    $active_dcd_label -> configure( -text => "$active_dcd", );

}, ) -> pack( -side => 'bottom', );

$f0 -> pack( -side => 'top', -fill => 'both', -expand => 1, );
$mw -> update();

# Get the resolution of the screen     #
# and position the mainwindow centered #
# and every other window to it's right #
my $x_position = int ( ( $mw -> screenwidth / 2 ) - ( $mw -> width / 2 ) );
my $y_position = int ( ( ( $mw -> screenheight - 80 ) / 2 ) - ( $mw -> height / 2 ) );

my $mw_position = "+" . $x_position . "+" . $y_position;#print $mw_position;
my $toplevel_position = "+" . ( $x_position + 150 ) . "+" . ( $y_position + 100 );

$mw -> geometry('1024x745');
$mw -> geometry ("$mw_position");

#~ my $plot_step = ( $header / ( $mw -> screenwidth ) );

# This is due to a windows-exlusive    #
# bug that forces the window to the    #
# background                           #
$mw -> focusForce if ( $^O ne 'linux' );
$f5 -> bind( '<Configure>', sub {
    $text -> configure( -width => $f5 -> width, -height => int( ( $f5 -> height ) / 17 ), );
    $mw -> update;
}, );

#############################################################################################################
###   End of main program                                                                                 ###
#############################################################################################################

MainLoop;

#############################################################################################################
###   Open files from GUI                                                                                 ###
#############################################################################################################

sub open_file {
    # Depending on the argument that this  #
    # subroutine is passed, the $filetypes #
    # variable will be set to psf or dcd   #
    # restricting the files viewed with    #
    # the getOpenMethod to the extension   #
    # currently in $filetypes              #

    if ( $linux || $mac ) {
        if ( $ [0] eq 'psf' ) {
            $filetypes = [ ['PSF Files', '.psf'] ];
        }
        elsif ( $_[0] eq 'dcd' ) {
            $filetypes = [ ['DCD Trajectory Files', '.dcd'] ];
        }
    }
    # Again, due to another bug on windows #
    # the variable needs to be defined     #
    # differently                          #
    else {
```

```
        if ( $_[0] eq 'psf' ) {
            $filetypes = [ ['PSF FIles',  '.psf'], ['PSF FIles',  '.psf'] ];
        }
        elsif ( $_[0] eq 'dcd' ) {
            $filetypes = [ ['DCD Trajectory Files', '.dcd'], ['DCD Trajectory Files', '.dcd'] ];
        }
    }

    my $file = $mw -> getOpenFile( -filetypes => $filetypes, -initialdir => getcwd, );
    # If the file selected through the     #
    # getOpen method is a .psf file        #
    if ( $file and $file =~ /(.*)(\/|\\)(.*)\.psf/ ) {
        if ( $linux || $mac ) {
            # If on *nix invoke the abs_path       #
            # subroutine and store it's result in  #
            # a scalar. This is nessecary because  #
            # normally relative paths will be used #
            # rendering the data in $psf_file      #
            # obsolete every time the working      #
            # directory is changed                 #
            $psf_file = abs_path ( $file );
            $psf_name = $3;
            $active_psf = $3 . '.psf';
        }
        else {
            $psf_name = $3;
            $active_psf = $3 . '.psf';
            # else substitute the '/' for '\' in   #
            # $file as windows uses a backward     #
            # slash & the getOpen method returns   #
            # the absolute path to the file unix-  #
            # style                                #
            $file =~ s/\//\\/g;
            if ( $file =~ /\s/ ) {
                $psf_file = "\"$file\"";
            }
            else {
                $psf_file = $file;
            }
        }
        $have_psf = 1;

        $psf_button -> configure( -state => 'disabled', );
    }
    elsif ( $file and $file =~ /psf$/ ) {
        if ( $linux or $mac ) {
            $file_selection_window -> messageBox( -message => "Unexpected character in the PSF name.
Aborting.",
                                                  -icon => "warning",
                                                  -font => $font_12, );
        }
        else {
            $file_selection_window -> messageBox( -message => "Unexpected character in the PSF name.
Aborting.",
                                                  -icon => "warning", );
        }

        exit;
    }

    # Do the same for .dcd files and add a #
    # scalar which contains the the name   #
    # of the dcd file                      #
    if ( $file and $file =~ /(.*)(\/|\\)(.*)\.dcd/ ) {
        if ( $linux || $mac ) {
            $dcd_file = abs_path ( $file );
            $dcd_loc = $1;
            $dcd_name = $3;
            $active_dcd = $3 . '.dcd';
        }
        else {
            $dcd_loc = $1;
            $dcd_name = $3;
            $active_dcd = $3 . '.dcd';
```

```
            $file =~ s/\///\\/g;
            $dcd_loc =~ s/\///\\/g;
            if ( $file =~ /\s/ ) {
                $dcd_file = "\"$file\"";
            }
            else {
                $dcd_file = $file;
            }
        }
        $have_dcd = 1;

        $dcd_button -> configure( -state => 'disabled', );
        folder_size ( $launch_dir );
    }
    elsif ( $file and $file =~ /dcd$/ ) {
        if ( $linux or $mac ) {
            $file_selection_window -> messageBox( -message => "Unexpected character in the DCD name.
Aborting.",
                                                  -icon => "warning",
                                                  -font => $font_12, );
        }
        else {
            $file_selection_window -> messageBox( -message => "Unexpected character in the DCD name.
Aborting.",
                                                  -icon => "warning", );
        }

        exit;
    }
    # If the file selected is not a psf or #
    # a dcd, then display a window with a  #
    # warning                              #

    if ( $have_psf && $have_dcd ) {
        parser();

        $have_psf = 0;
        $have_dcd = 0;
    }
}

#############################################################################################################
###   Parse the .psf file to extract info for the protein(s)                                              ###
#############################################################################################################

sub parser {
    ( $psf_file, $dcd_file, ) = @_ if ( @_ );

    if ( $windows and not @_ ) {
        my @psf_path = split '', $psf_file;
        my @dcd_path = split '', $dcd_file;

        my $bad_character_test = 0;

        foreach ( @psf_path ) {
            if ( ord ( $_ ) > 127 ) {
                $bad_character_test = 1;
            }
        }

        foreach ( @dcd_path ) {
            if ( ord ( $_ ) > 127 ) {
                $bad_character_test = 1;
            }
        }

        if ( $bad_character_test ) {
            $file_selection_window -> messageBox( -message => "Sorry. Non-ASCII characters detected
in the path of the specified DCD/PSF files. " .
                                                              "Please  place  your  files  in  a
directory whose path does not contain special characters " .
                                                               "(for  example,  something  like
C:\\MD\\myproject\\dcdpsf\\).",
                                                  -icon => 'warning', );
```

```
            exit( 1 );
        }
    }

    $psf_file =~ s/\"//g if ( $windows );

    open PSF_FILE, '<', $psf_file || die "Cannot open $psf_file for reading: $!\n";

    # Extract the number of atoms found    #
    # in the .psf file                     #
    my $num_atoms = 0;
    my $psf_line;
    my $psf_pos;

    our $is_ext_psf = 0;

    while ( <PSF_FILE> ) {
        if ( /EXT/i and $. == 1 ) {
            $is_ext_psf = 1;
        }

        if ( /(\d*)\s\!NATOM/ ) {
            $num_atoms = $1;
            $psf_pos = tell;
            $psf_line = $.;
            last;
        }
    }

    my @psf_file = <PSF_FILE>;
    close PSF_FILE;

    my $no_segid_index;
    if ( $is_ext_psf ) {
        $no_segid_index = 11;
    }
    else {
        $no_segid_index = 9;
    }

    my @test_line = split '', $psf_file[$psf_line];
    if ( $test_line[$no_segid_index] =~ / / ) {
        my $temporary_mw = MainWindow -> new( -title => 'test', );
        $temporary_mw -> withdraw;
        my $response;

        if ( $linux or $mac ) {
            $response = $temporary_mw -> messageBox( -message => "It seems the PSF file you
specified " .
            "lacks chain ids. If this is a single chain molecule-without waters and ions- you can "
.
            "assign chain id A to all atoms and procced normally. Do you want to do so now? Click "
.
            "'no' to open a window to the PSF file and specify the ranges for each chain.",
                                                -icon => 'warning',
                                                -type => 'yesno',
                                                -font => "$font_12", );
        }
        else {
            $response = $temporary_mw -> messageBox( -message => "It seems the PSF file you
specified " .
            "lacks chain ids. If this is a single chain molecule-without waters and ions- you can "
.
            "assign chain id A to all atoms and procced normally. Do you want to do so now? Click "
.
            "'no' to open a window to the PSF file and specify the ranges for each chain.",
                                                -icon => 'warning',
                                                -type => 'yesno', );
        }

        if ( $response =~ /yes/i ) {
            $new_psf = "$psf_name.mod.psf";

            open PSF_FILE, '<', $psf_file || die "Cannot open $psf_file for reading: $!\n";
```

```
            open NEW_PSF, '>', $new_psf || die "Cannot open $new_psf for writing: $!\n";

            my $line_count = 0;
            while ( <PSF_FILE> ) {
                if ( /\d*\s\!NATOM/ ) {
                    print NEW_PSF $_;
                    last;
                }
                else {
                    print NEW_PSF $_;
                }
            }

            while ( <PSF_FILE> ) {
                if ( $line_count < $num_atoms && /^(\s+\d+ ) (\s+.+?)$/ ) {
                    print NEW_PSF $1 . 'A' . $2 . "\n";
                }
                else {
                    print NEW_PSF $_;
                }

                $line_count++;
            }

            close NEW_PSF;
            close PSF_FILE;

            $psf_file = abs_path ($new_psf);# if ( $linux or $mac );
            $active_psf = $new_psf;
            $temporary_mw -> destroy;
            parser( $psf_file, $dcd_file, );
        }
        elsif ( $response =~ /no/i ) {
            $new_psf = "$psf_name.mod.psf";
            my @allowed_segids = ( 'A' .. 'Z' );

            open BAD_PSF, '<', $psf_file || die "Cannot open $psf_file for reading: $!\n";

            my $temporary_window = $temporary_mw ->  Toplevel();
            $temporary_window -> Label( -text => 'Specify the range for each chain using the atom
number ( first column )', ) -> pack;

            my $text_frame = $temporary_window -> Frame() -> pack;

            my @bars_frame;
            $bars_frame[0] = $temporary_window -> Frame() -> pack;

            my $temporary_text = $text_frame -> ROText( -height => 25, -width => 90, ) -> pack;
            while ( <BAD_PSF> ) {
                $temporary_text -> insert( 'end', $_ );
            }

            my @lower;
            my @upper;

            my $temporary_row = 1;
            my $temporary_column = 1;

             $bars_frame[$temporary_row-1] -> Label( -text => 'From atom ', ) -> grid( -row =>
$temporary_row, -column => $temporary_column, );
            $bars_frame[$temporary_row-1] -> Entry( -textvariable => \$lower[$temporary_row-1], ) ->
grid( -row => $temporary_row, -column => ++$temporary_column, );
             $bars_frame[$temporary_row-1] -> Label( -text => ' to atom ', ) -> grid( -row =>
$temporary row, -column => ++$temporary column, );
            $bars_frame[$temporary_row-1] -> Entry( -textvariable => \$upper[$temporary_row-1], ) ->
grid( -row => $temporary_row, -column => ++$temporary_column, );
            $bars_frame[$temporary_row-1] -> Label( -text => 'assign chain identifier ', ) -> grid(
-row => $temporary_row, -column => ++$temporary_column, );
             $bars_frame[$temporary_row-1]      ->      Entry(      -textvariable      =>      \
$allowed_segids[$temporary_row-1], -state => 'readonly', ) -> grid( -row => $temporary_row, -column
=> ++$temporary_column, );
            $bars_frame[$temporary_row-1] -> Button( -text => 'Add row', -command => sub {
                $temporary_row++;
                $temporary_column = 1;
```

```
                $bars_frame[$temporary_row-1] = $temporary_window -> Frame() -> pack;

                $bars_frame[$temporary_row-1] -> Label( -text => 'From atom ', ) -> grid( -row =>
$temporary_row, -column => $temporary_column, );
                $bars_frame[$temporary_row-1] -> Entry( -textvariable => \$lower[$temporary_row-1],
) -> grid( -row => $temporary_row, -column => ++$temporary_column, );
                $bars_frame[$temporary_row-1] -> Label( -text => ' to atom ', ) -> grid( -row =>
$temporary_row, -column => ++$temporary_column, );
                $bars_frame[$temporary_row-1] -> Entry( -textvariable => \$upper[$temporary_row-1],
) -> grid( -row => $temporary_row, -column => ++$temporary_column, );
                $bars_frame[$temporary_row-1] -> Label( -text => 'assign chain identifier ', ) ->
grid( -row => $temporary_row, -column => ++$temporary_column, );
                    $bars_frame[$temporary_row-1]     ->     Entry(     -textvariable     =>     \
$allowed_segids[$temporary_row-1], -state => 'readonly', ) -> grid( -row => $temporary_row, -column
=> ++$temporary_column, );
                $bars_frame[$temporary_row-1] -> Button( -text => 'Remove row', -command => sub {
                    $temporary_row--;
                    $bars_frame[$temporary_row] -> destroy;
                    pop @bars_frame;
                }, ) -> grid( -row => $temporary_row, -column => ++$temporary_column, );
            }, ) -> grid( -row => $temporary_row, -column => ++$temporary_column, );

            $temporary_window -> Button( -text => 'Create PSF', -command => sub {
                seek BAD_PSF, 0, 0;
                open NEW_PSF, '>', $new_psf || die "Cannot open $new_psf for writing: $!\n";

                while ( <BAD_PSF> ) {
                    print NEW_PSF $_;
                    if ( /\!NATOM/ ) {
                        last;
                    }
                }

                my $i;
                my $check = 0;
                for ( $i = 0 ; $i < scalar(@bars_frame) ; $i++ ) {
                    if ( $lower[$i] and $upper[$i] ) {
                        $check++;
                    }
                }

                if ( $check == $i ) {
                    $psf_line = '';
                    my $previous_line;
                    my $line_count = 0;
                    my $remember_psf = 0;
                    for ( $i = 0 ; $i < scalar(@bars_frame ) ; $i++ ) {
                        if ( $psf_line ) {
                            seek BAD PSF, $psf line, 0;
                            if ( $previous_line =~ /^(\s+)(\d+)  (\s+.+?)$/ ) {
                                print NEW_PSF $1 . $2 . " $allowed_segids[$i]" . $3 . "\n";
                            }
                        }

                        while ( <BAD_PSF> ) {
                            if ( $line_count < $num_atoms && /^(\s+)(\d+)  (\s+.+?)$/ ) {
                                if ( $2 == $upper[$i] + 1 ) {
                                    $psf_line = tell;
                                    $previous_line = $_;
                                    last;
                                }

                                if ( $2 >= $lower[$i] and $2 <= $upper[$i] ) {
                                    print NEW_PSF $1 . $2 . " $allowed_segids[$i]" . $3 . "\n";
                                }
                            }
                            else {
                                $previous_line = $_;
                                $psf_line = tell;
                                last;
                            }

                            $line_count++;
```

```
                    }
                }

                seek BAD_PSF, $psf_line, 0;
                print NEW_PSF $previous_line;

                while ( <BAD_PSF> ) {
                    print NEW_PSF $_;
                }

                close NEW_PSF;
                close BAD_PSF;

                $psf_file = abs_path( $new_psf );
                $active_psf = $new_psf;
                $temporary_mw -> destroy;
                parser( $psf_file, $dcd_file, );
            }
            else {
                close NEW_PSF;
                if ( $mac or $linux ) {
                    $temporary_mw -> messageBox( -message => 'All of the boxes must be filled in
order to create a new PSF', -font => "$font_12", );
                }
                else {
                    $temporary_mw -> messageBox( -message => 'All of the boxes must be filled in
order to create a new PSF', -font => "$font_12", );
                }
            }
        }, ) -> pack( -side => 'bottom', );
    }
    else {
        if ( $run_from_terminal ) {
            exit;
        }
        else {
            $mw -> destroy;
            kill -2, $$ or die ($!);
        }
    }
}
else {
    $psf_file =~ s/\"//g if ( $windows );

    open PSF_FILE, '<', $psf_file || die "Cannot open $psf_file for reading: $!\n";

    seek PSF_FILE, $psf_pos, 0;

    my $i = 0;

    my @atom_types;
    my @chain_ids;

    # Continue parsing through the .psf    #
    # file storing the various atmids and  #
    # segids as well as the number of      #
    # residues in each chain               #
    while ( <PSF_FILE> ) {
        if ( $i < $num_atoms ) {
            my $temp_chain = substr $_, $no_segid_index, 4;
            $temp_chain =~ s/ //g;

            my $temp_atom_type;
            $temp_atom_type = substr $_ , 38, 4 if ( $is_ext_psf );
            $temp_atom_type = substr $_, 24, 4 if ( not $is_ext_psf );

            $temp_atom_type =~ s/ //g;

            push @chain_ids, $temp_chain;
            push @atom_types, $temp_atom_type;

            if ( /^\s*\d+\s*(\D+)\s*(\d+)\s*\w+\s*\w+/ ) {
                my $temp_segid = $1;
                my $temp_resid = $2;
```

```
                $temp_segid =~ s/ //g;
                $num_residues{$temp_segid} = $temp_resid;
            }
        }

        $i++;
    }

    close ( PSF_FILE );

    $psf_file = "\"$psf_file\"" if ( $windows );

    # Substitute every proton atmid for H  #
    foreach ( @atom_types ) {
        if ( $_ =~ /^H.*/ ) {
            $_ =~ s/^H.*/H/g;
        }
    }

    # Sort the atmids - segids and remove  #
    # any and all multiple entries         #
    my @sorted_atom_types = sort ( @atom_types );
    our @unique_atom_types = uniq ( @sorted_atom_types );

    my @sorted_chain_ids = sort ( @chain_ids );
    our @unique_chain_ids = uniq ( @sorted_chain_ids);

    # Use carma to check the validity by   #
    # parsing the output of the following  #
    # carma run searching for the presence #
    # of the word 'Abort'                  #
    my $valid_psf_dcd_pair = '';
    if ( $linux || $mac ) {
        $valid_psf_dcd_pair = `carma -v -last 1 $psf_file $dcd_file`;
    }
    else {
        $valid_psf_dcd_pair = `carma.exe -v -last 1 $psf_file $dcd_file`;
    }

    # If found create a help message or    #
    # a window prompting the user to retry #
    if ( $valid_psf_dcd_pair =~ /Abort./i ) {
        if ( $run_from_terminal ) {
            die "\nNumber of atoms in PSF and DCD do not match.\n\n";
        }
        else {
            #~ $file_selection_window -> packForget;
            my $response;
            if ( $linux or $mac ) {
                $response = $file selection window -> messageBox( -message => "Number of atoms
in PSF and DCD do not match.\nWould you like to retry?",
                                                                  -type => 'yesno',
                                                                  -icon => 'warning',
                                                                  -font => "$font_12", );
            }
            else {
                $response = $file_selection_window -> messageBox( -message => "Number of atoms
in PSF and DCD do not match.\nWould you like to retry?",
                                                                  -type => 'yesno',
                                                                  -icon => 'warning', );
            }

            if ( $response eq "Yes" ) {
                #~ $gui -> pack;
                $dcd_button -> configure( -state => 'normal', );
                $psf_button -> configure( -state => 'normal', );
            }
            else {
                $mw -> destroy;
                $file_selection_window -> destroy;
                kill -2, $$ || die ( $! );
            }
        }
    }
```

```
        # If not found proceed parsing the dcd #
        # header                               #
        else {
            dcd_header_parser();
        }
    }
}

#############################################################################################
###   Parse the STDOUT of carma to extract the number of frames in the header of the .dcd file  ###
#############################################################################################

sub dcd_header_parser {
    my $input;
    $input = shift if ( $_[0] and $_[0] !~ /dcd|psf/ );

    if ( $input ) {
        create_dir();

        if ( $linux || $mac ) {
            `carma -v -last 1 $active_psf $active_dcd > carma.out`;
        }
        else {
            `carma.exe -v -last 1 $active_psf $active_dcd > carma.out`;
        }
    }
    else {
        if ( $linux || $mac ) {
            `carma -v -last 1 $psf_file $dcd_file > carma.out`;
        }
        else {
            `carma.exe -v -last 1 $psf_file $dcd_file > carma.out`;
        }
    }

    unlink ( "carma.fitted.dcd" );

    opendir CWD, getcwd or die $!;

    while ( my $file_to_delete = readdir CWD ) {
        if ( $file_to_delete =~ /.0000001.ps$/ ) {
            unlink $file_to_delete;
        }
    }

    closedir CWD;

    # Extract the number of frames found   #
    # in the .dcd header                   #
    open OUTPUT, '<', "carma.out" || die "Cannot open carma.out for reading: $!";
    while ( <OUTPUT> ) {
        if ( /Number of coordinate sets is (\d+)/ ) {
            $header = $1;
        }
    }

    close (OUTPUT);
    unlink ( "carma.out", );

    if ( $input ) {
        return ( $header );
    }

    # At this point every check has been   #
    # succesful and unless the program was #
    # run from the terminal, the container #
    # frame is drawn and at the same time  #
    # the window for file selection is     #
    # withdrawn                            #

    $have_files = 1;

    $file_selection_window -> destroy if ( Exists($file_selection_window) );
```

```
    $mw -> deiconify;
    $mw -> raise;
}

##############################################################################################
###   Run carma with the selected parameters                                               ###
##############################################################################################

sub carma {
    # Set the variable used for reporting  #
    # success to the rest of the program   #
    # to zero and substitute any multiple  #
    # spaces in the $flag scalar with a    #
    # single space                         #

    $all_done = 0;
    $flag =~ s/[\s]\s+/ /g;
    $flag = 'carma.exe' . $flag . " $active_psf $active_dcd" if ( $windows );
    $flag = 'carma' . $flag . " $active_psf $active_dcd" if ( $linux || $mac );

    if ( $windows ) {
        $text -> insert( 'end', "$flag\n", 'info' );
        $text -> see( 'end', );
        $mw -> update;

        system ( "$flag $active_psf $active_dcd > last_carma_run.log" );
    }
    elsif ( $linux or $mac ) {
        $text -> insert( 'end', "$flag\n", 'info' );
        $text -> see( 'end', );
        $mw -> update;

        if ( $terminal ) {
            if ( $terminal eq 'xterm' ) {
                 system  ( "$terminal -fg white -bg black -geometry 80x25+800+200 -e \"$flag
$active psf $active_dcd | tee last_carma_run.log\"" );
            }
            elsif ( $terminal eq 'rxvt' ) {
                system ( "$terminal -fg white -bg black -geometry 80x25+800+200 -e sh -c \"$flag
$active_psf $active_dcd | tee last_carma_run.log\"" );
            }
            else {
                 system  ( "$terminal --geometry 80x25+800+200 -x sh -c \"$flag $active_psf
$active_dcd | tee last_carma_run.log\"" );
            }
        }
        else {
            system ( "$flag $active_psf $active_dcd | tee last_carma_run.log" );
        }
    }

    # When the run is completed open the   #
    # file 'last_carma_run.log' in the CWD and #
    # parse through it one line at a time  #
    # searching for the pattern 'All done' #
    open TEMP_OUT, "last_carma_run.log" || die "Cannot open last_carma_run.log for reading: $!\n";
    while ( <TEMP_OUT> ) {
        # If a match is found set the value of #
        # $all_done to 1                       #
        if ( /All done/i ) {
            $all_done = 1;
        }
        else {
            $all_done = 0;
        }
    }

    close TEMP_OUT;

    $flag = '';
    $index_seg_id_flag = '';
}

##############################################################################################
```

```
###   Draw the window containing every atmid in the PSF file                                    ###
#################################################################################################

sub raise_custom_window {
    # The same format is used throughout   #
    # the program for the subroutines that #
    # are used in order to cover the many  #
    # different keywords used by carma so  #
    # they will be described in detail in  #
    # the comments of this subroutine and  #
    # are valid for every subroutine which #
    # follows                              #

    # First, the variables, arrays, hashes #
    # are imported ( when necessary ) and  #
    # initialised                          #
    my $x = 1;
    my $y = 1;

    $custom_id_flag = '';
    $custom_selection = '';

    our @custom_atom_ids;

    # Second, if the toplevel window does  #
    # not exist, it is created and it's    #
    # elements are defined                 #
    if ( !Exists( my $top_custom ) ) {
        # The toplevel is drawn, titled and    #
        # positioned. The 'X' button in the    #
        # upper right ( or left depending on   #
        # your window manager ) corner doesn't #
        # destroy the window but withdraws it  #
        my $top_custom = $mw -> Toplevel();
        $top_custom -> geometry("$toplevel_position");
        $top_custom -> protocol( 'WM DELETE WINDOW' => sub { $top_custom -> withdraw });
        $top_custom -> title( "Custom selection" );

        my $frame_custom1 = $top_custom -> Frame() -> pack( -side => 'bottom', );
        my $frame_custom2 = $top_custom -> Frame() -> pack( -side => 'top', );

        $frame_custom1 -> Button( -text => 'Submit',
                                  -command => sub {
                                   $top_custom -> withdraw;

                                   # Foreach active element of the array  #
                                   # @custom_atom_ids add its atmid to a  #
                                   # scalar and use it for the flag       #
                                   foreach ( @custom_atom_ids ) {
                                       if ( defined ) {
                                           if ( /(\w+)/ ) {
                                                $custom_id_flag = $custom_id_flag . " -atmid " . $1;
                                                $custom_selection = 1;
                                                $atm_id_flag = '';
                                            }
                                       }
                                   }
                                 }, )
                                 -> pack( -side => 'right', );

        $frame_custom1 -> Button( -text => 'Return',
                                  -command => [ $top_custom => 'withdraw' ], )
                                  -> pack(-side => 'left', );

        # For every atmid draw a checkbutton   #
        # and name it after itself. If it is   #
        # active store the atmid in the array  #
        # @custom_atom_ids                     #
        our @unique_atom_types;
        for my $i ( 0 .. $#unique_atom_types ) {
            $frame_custom2 -> Checkbutton( -text => $unique_atom_types[$i],
                                           -offvalue => '',
                                           -onvalue => $unique_atom_types[$i],
                                           -variable => \$custom_atom_ids[$i], )
```

```
                                        -> grid( -row => "$x", -column => "$y", -sticky => 'w',
);

            $x++;
            if ( $x == 5 ) {
                $y++;
                $x = 1;
            }
        }
    }

    # Finally, if the window has already   #
    # been created it is brought in the    #
    # foreground                           #
    else {
        $custom_id_flag = '';
        $atm_id_flag = '';
        $top_custom -> deiconify;
        $top_custom -> raise;
    }
}

#########################################################################################################
###   Draw the window for the RMSD matrix calculation                                                 ###
#########################################################################################################

sub rmsd_window {
    my $rmsd_first = 1;
    my $rmsd_first_flag = '';
    my $rmsd_last = dcd_header_parser( "rmsd" );
    my $rmsd_last_flag = '';
    my $rmsd_step;
    my $rmsd_step_flag = '';
    my $rmsd_min = '';
    my $rmsd_min_flag = '';
    my $rmsd_max = '';
    my $rmsd_max_flag = '';
    my $rmsd_reverse = '';
    my $rmsd_plot = '';
    my $rmsd_top;
    my $matrix_size = 0;

    if ( !Exists( $rmsd_top ) ) {
        # If the number or frames is greater   #
        # than 3k set the value of $rmsd_step  #
        # to $header/3000 rounded up to the    #
        # nearest integer, otherwise set it to #
        # 1                                    #
        if ( $rmsd_last <= 3000 ) {
            $rmsd_step = 1;
        }
        elsif ( $rmsd_last > 3000 ) {
            $rmsd_step = int ( ( $rmsd_last / 3000 ) + 0.5 );
        }

        $rmsd_top = $mw -> Toplevel( -title => 'RMSD matrix' );
        $rmsd_top -> geometry("$toplevel_position");

        my $frame_rmsd1 = $rmsd_top -> Frame() -> pack( -expand => 1, -fill => 'x', );
        our $frame_rmsd2 = $rmsd_top -> Frame() -> pack( -expand => 1, -fill => 'x', );
        my $frame_rmsd3 = $rmsd_top -> Frame() -> pack( -expand => 1, -fill => 'x', );
        my $frame_rmsd6 = $rmsd_top -> Frame() -> pack( -expand => 1, -fill => 'x', );
        my $frame_rmsd4 = $rmsd_top -> Frame( -borderwidth => 2, -relief => 'groove', ) -> pack(
-expand => 1, -fill => 'x', );

        radiobuttons ( $frame_rmsd1 );
        checkbuttons ( $frame_rmsd2 );
        otherbuttons ( $frame_rmsd3 );

        $frame_rmsd6 -> Label( -text => "\nVarious options", -font => $font_20, ) -> pack;

        # $frame_rmsd4 -> Label( -text => "\nVarious options", ) -> grid ( -row => 0, -column =>
2, );
```

```
        # Create entry boxes for user input    #
        $frame_rmsd4 -> Label( -text => 'First frame to use: ', )
                                -> grid( -row => 1, -column => 1, -sticky => 'w', );
        $frame_rmsd4 -> Entry( -textvariable => \$rmsd_first, )
                                -> grid( -row => 1, -column => 2, );
        $frame_rmsd4 -> Label( -text => 'Last frame to use: ', )
                                -> grid( -row => 2, -column => 1, -sticky => 'w', );
        $frame_rmsd4 -> Entry( -textvariable => \$rmsd_last, )
                                -> grid( -row => 2, -column => 2, );
        $frame_rmsd4 -> Label( -text => 'Stride (step) between frames: ', )
                                -> grid( -row => 3, -column => 1, -sticky => 'w', );
        $frame_rmsd4 -> Entry( -textvariable => \$rmsd_step, )
                                -> grid( -row => 3, -column => 2, );

        my $temp_dim = int ( ( $rmsd_last - $rmsd_first ) / $rmsd_step );
        $matrix_size = int ( ( $temp_dim * $temp_dim * 6 ) / ( 1024 * 1024 ) );

        $frame_rmsd4 -> Label( -text => "The matrix that will be produced will be $matrix_size MB",
-fg => 'red', )
                                -> grid( -row => 7, -column => 1, );

        $frame_rmsd4 -> Label( -text => 'Minimum value for postscript plot (dark blue): ', )
                                -> grid( -row => 4, -column => 1, -sticky => 'w', );
        $frame_rmsd4 -> Entry( -textvariable => \$rmsd_min, )
                                -> grid( -row => 4, -column => 2, );
        $frame_rmsd4 -> Label( -text => 'Maximum value for postscript plot (dark red): ', )
                                -> grid( -row => 5, -column => 1, -sticky => 'w', );
        $frame_rmsd4 -> Entry( -textvariable => \$rmsd_max, )
                                -> grid( -row => 5, -column => 2, );

        $frame_rmsd4 -> Checkbutton( -text => 'Reverse color gradient',
                                     -variable => \$rmsd_reverse,
                                     -offvalue => '',
                                     -onvalue => " -reverse", )
                                     -> grid( -row => 6, -column => 1, -sticky => 'w', );

        my $frame_rmsd5 = $rmsd_top -> Frame() -> pack();

        # For every variable used for input that is active create a flag and add
        # it to the flag used to run carma
        $frame_rmsd5 -> Button( -text => 'Run',
                                -command => sub {
            $rmsd_first_flag = ( ( $rmsd_first != 1 ) ? " -first $rmsd_first" : '' );
            $rmsd_last_flag = ( ( $rmsd_last != $header ) ? " -last $rmsd_last" : '' );
            $rmsd_step_flag = ( ( $rmsd_step != 1 ) ? " -step $rmsd_step" : '' );
            $rmsd_min_flag = ( $rmsd_min ? " -min $rmsd_min" : '' );
            $rmsd_max_flag = ( $rmsd_max ? " -max $rmsd_max" : '' );

            $rmsd_top -> destroy();
            $mw -> update;

            # If a segid has been specified add it to the flag as well

            $seg_id_flag = '' if $seg_id_flag;

            foreach ( @seg_ids ) {
                if ( defined ) {
                    $seg_id_flag = $seg_id_flag . $_;
                }
            }

            if ( $res_id_flag ) {
                 $flag  =  "  -v  -w  -cross  $rmsd_first_flag  $rmsd_last_flag  $rmsd_step_flag
$rmsd min flag $rmsd_max_flag $atm_id_flag $custom_id_flag $res_id_flag";
            }
            elsif ( $seg_id_flag ) {
                 $flag  =  "  -v  -w  -cross  $rmsd_first_flag  $rmsd_last_flag  $rmsd_step_flag
$rmsd_min_flag $rmsd_max_flag $seg_id_flag $atm_id_flag $custom_id_flag";
            }
            else {
                 $flag  =  "  -v  -w  -cross  $rmsd_first_flag  $rmsd_last_flag  $rmsd_step_flag
$rmsd_min_flag $rmsd_max_flag $atm_id_flag $custom_id_flag";
            }
```

```
            create_dir();

            $text -> insert( 'end', "\n\nNow calculating RMSD matrix.\n\n", 'cyan', );
            $text -> see( 'end', );
            $mw -> update;

            carma();

            if ( $all_done ) {
                $text -> insert( 'end', "\nCalculation finished. Now creating a postcript plot of
carma.RMSD.matrix\n", 'cyan' );
                $text -> see( 'end', );

                $mw -> update;
                sleep 1;

                my $coloring;
                $coloring = `carma.exe $rmsd_reverse -col - < carma.RMSD.matrix` if ( $windows );
                $coloring = `carma $rmsd_reverse -col - < carma.RMSD.matrix` if ( $linux || $mac );

                if ( $coloring =~ /(\d*\.\d*) to (\d*\.\d*)/ ) {
                    my $from = sprintf ("%5.2f", $1);
                    my $to   = sprintf ("%5.2f", $2);

                    if ( $rmsd_reverse ) {
                        $text -> insert( 'end', "\nThe color gradient for the postscript image
ranges from $from (dark red), through yellow, to $to (dark blue)\n", );
                    }
                    else {
                        $text -> insert( 'end', "\nThe color gradient for the postscript image
ranges from $from (dark blue), through yellow, to $to (dark red)\n", );
                    }

                    $text -> insert( 'end', "\nUse \"View Results\"\n", 'valid' );
                    $text -> see( 'end', );
                    $image_menu -> configure( -state => 'normal', );

                    if ( $active_dcd =~ /(.+)\.dcd/ ) {
                        mv ( "carma.stdin.ps", "RMSD_matrix.$1.ps" );
                    }
                }
            }
            else {
                $text -> insert( 'end', "\nSomething went wrong. For details check
last_carma_run.log located in :\n", 'error', );
                $text -> insert( 'end', getcwd . "\n", 'info', );
                $text -> see( 'end', );
            }
            }, )
            -> grid( -row => 2, -column => 2, );

        $frame_rmsd5 -> Button( -text => 'Return',
                                -command => [ $rmsd_top => 'withdraw' ], )
                                -> grid( -row => 2, -column => 1, );

    }
    else {
        $rmsd_top -> deiconify;
        $rmsd_top -> raise;
    }

}

#############################################################################################
###   Draw the window for the native contact calculation                                  ###
#############################################################################################

sub qfract_window {
    my $qfract_first = 1;
    my $qfract_first_flag = '';
    my $qfract_last = dcd_header_parser( "qfract" );
    my $qfract_last_flag = '';
    my $qfract_step = 1;
    my $qfract_step_flag = '';
```

```
    my $qfract_cutoff = 8;
    my $qfract_dist = 2;
    my $qfract_plot = '';
    my $qfract_ref = '';
    my $qfract_nat = '';
    my $qfract_ref_num = '';
    my $top_qfract;
    my $nat_pdb_file = '';
    my $nat_button = '';

    if ( !Exists ( $top_qfract ) ) {
        $top_qfract = $mw -> Toplevel( -title => 'Native Contacts', );
        $top_qfract -> geometry("$toplevel_position");
        $top_qfract -> protocol( 'WM_DELETE_WINDOW' => sub { $top_qfract -> withdraw }, );

        my $frame_qfract1 = $top_qfract -> Frame() -> pack( -expand => 1, -fill => 'x', );
        $frame_qfract2 = $top_qfract -> Frame() -> pack( -expand => 1, -fill => 'x', );
        my $frame_qfract3 = $top_qfract -> Frame() -> pack( -expand => 1, -fill => 'x', );
        my $frame_qfract4 = $top_qfract -> Frame() -> pack( -fill => 'x', );
        my $frame_qfract5 = $top_qfract -> Frame( -relief => 'groove', -borderwidth => 2, ) -> pack(
-expand => 1, -fill => 'x', );
        my $frame_qfract6 = $top_qfract -> Frame() -> pack();

        $frame_qfract5 -> Label( -text => 'Distance Cutoff: ', )
                               -> grid( -row => 1, -column => 1, -sticky => 'w', );
        $frame_qfract5 -> Entry( -textvariable => \$qfract_cutoff, )
                               -> grid( -row => 1, -column => 2, );
        $frame_qfract5 -> Label( -text => 'Residue Separation: ', )
                               -> grid( -row => 2, -column => 1, -sticky => 'w', );
        $frame_qfract5 -> Entry( -textvariable => \$qfract_dist, )
                               -> grid( -row => 2, -column => 2, );

        radiobuttons ( $frame_qfract1 );
        checkbuttons ( $frame_qfract2 );
        otherbuttons ( $frame_qfract3 );

        $frame_qfract4 -> Label( -text => "\nVarious Options", -font => $font_20, ) -> pack;

         $nat_button = $frame_qfract5 -> Checkbutton( -text => 'Use pdb file to define the native
structure',
                                                      -variable => \$qfract_nat,
                                                      -offvalue => '',
                                                      -onvalue => ' -nat',
                                                      -command => sub {
             $nat_pdb_file = $top_qfract -> getOpenFile( -filetypes => [ [ 'Native structure PDB
file', '.pdb' ] ], );
            unless ( $nat_pdb_file ) {
                $nat_button -> toggle;
            }
        }, )
        -> grid( -row => 4, -column => 1, -sticky => 'w', );

        $frame_qfract5 -> Checkbutton( -text => 'Use frame ',
                                       -variable => \$qfract_ref,
                                       -offvalue => '',
                                       -onvalue => ' -ref', )
                                       -> grid( -row => 5, -column => 1, -sticky => 'w', );
        $frame_qfract5 -> Entry( -textvariable => \$qfract_ref_num, )
                               -> grid( -row => 5, -column => 2, );
        $frame_qfract5 -> Label( -text => 'as the native structure', )
                               -> grid( -row => 5, -column => 3, );

        $frame_qfract5 -> Label( -text => 'First frame to use: ', )
                               -> grid( -row => 6, -column => 1, -sticky => 'w', );
        $frame_qfract5 -> Entry( -textvariable => \$qfract_first, )
                               -> grid( -row => 6, -column => 2, );
        $frame_qfract5 -> Label( -text => 'Last frame to use: ', )
                               -> grid( -row => 7, -column => 1, -sticky => 'w', );
        $frame_qfract5 -> Entry( -textvariable => \$qfract_last, )
                               -> grid( -row => 7, -column => 2, );
        $frame_qfract5 -> Label( -text => 'Stride (step) between frames: ', )
                               -> grid( -row => 8, -column => 1, -sticky => 'w', );
        $frame_qfract5 -> Entry( -textvariable => \$qfract_step, )
                               -> grid( -row => 8, -column => 2, );
```

```
        $frame_qfract6 -> Button( -text => 'Return',
                                  -command => [ $top_qfract => 'withdraw' ], )
                                  -> pack( -side => 'left', );

        $qfract_run_button = $frame_qfract6 -> Button( -text => 'Run',
                                                       -state => 'disabled',
                                                       -command => sub {
            $qfract_first_flag = ( ( $qfract_first != 1 ) ? " -first $qfract_first" : '' );
            $qfract_last_flag = ( ( $qfract_last != $header ) ? " -last $qfract_last" : '' );
            $qfract_step_flag = ( ( $qfract_step != 1 ) ? " -step $qfract_step" : '' );

            $top_qfract -> withdraw;

            $seg_id_flag = '' if $seg_id_flag;

            foreach ( @seg_ids ) {
                if ( defined ( $_ ) ) {
                    $seg_id_flag = $seg_id_flag . $_;
                }
            }

            if ( $res_id_flag ) {
                $flag = " -v -qf $qfract_cutoff $qfract_dist $qfract_nat $nat_pdb_file $qfract_ref
$qfract_ref_num $atm_id_flag $res_id_flag $qfract_first_flag $qfract_last_flag $qfract_step_flag";
            }
            elsif ( $seg_id_flag ) {
                $flag = " -v -qf $qfract_cutoff $qfract_dist $qfract_nat $nat_pdb_file $qfract_ref
$qfract_ref_num $atm_id_flag $seg_id_flag $qfract_first_flag $qfract_last_flag $qfract_step_flag";
            }
            else {
                $flag = " -v -qf $qfract_cutoff $qfract_dist $qfract_nat $nat_pdb_file $qfract_ref
$qfract_ref_num $atm_id_flag $qfract_first_flag $qfract_last_flag $qfract_step_flag";
            }

            create_dir();

            $text -> insert( 'end', "\n\nNow calculating fraction of native contacts.\n\n", 'cyan',
);
            $text -> see( 'end', );
            $mw -> update;

            carma();

            if ( $all_done ) {
                if ( $active_dcd =~ /(.+)\.dcd/ ) {
                    mv ( 'carma.Qfraction.dat', "Qfraction.$1.dat" );
                }

                $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n", 'valid' );
                $text -> see( 'end', );
                $image_menu -> configure( -state => 'normal', );
            }
            else {
                 $text   ->   insert(   'end',   "\nSomething   went   wrong.   For   details   check
last_carma_run.log located in :\n", 'error', );
                $text -> insert( 'end', getcwd . "\n", 'info', );
                $text -> see( 'end', );
            }
        }, )
        -> pack( -side => 'right', );

    if ( $active_run_buttons_qfract ) {
        $qfract_run_button -> configure( -state => 'normal', );
    }
    else {
        $qfract_run_button -> configure( -state => 'disabled', );
    }

    }
    else {
        $top_qfract -> deiconify;
        $top_qfract -> raise;
    }
```

```
}

##########################################################################################################
###   Draw the window for the dPCA calculation                                                         ###
##########################################################################################################

sub dpca_window {
    $atm_id_flag = '' if ( $atm_id_flag );
    $custom_id_flag = '' if ( $custom_id_flag );

    my $dpca_top;
    my $dpca_eigenvectors = '';
    my $dpca_combinations = '';
    my $dpca_temp = '';
    my $dpca_cutoff = '';
    my $dpca_dgwidth = '';
    my $dpca_dgwidth_num = '';
    my $dpca_3d = '';
    my $dpca_first = 1;
    my $dpca_first_flag = '';
    my $dpca_last = dcd_header_parser( "dpca" );
    my $dpca_last_flag = '';
    my $dpca_step = 1;
    my $dpca_step_flag = '';
    our $chi1 = '';
    our $include_segid = '';
    our $dpca_auto_entry;
    our $dpca_auto_entry_num = 10;

    our $res_id_flag;
    our @cluster_stats = '';

    if ( !Exists( $dpca_top ) ) {
        unless ( $dpca_temp ) {
            $dpca_temp = 298;
            $dpca_eigenvectors = 5;
            $dpca_combinations = 3;
        }

        $dpca_top = $mw -> Toplevel( -title => 'Dihedral PCA ' );
        $dpca_top -> geometry("$toplevel_position");

        $dpca_frame = $dpca_top -> Frame( -borderwidth => 2, -relief => 'groove', );

        $dpca_frame -> Label( -text => 'Temperature(K): ',)
                            -> grid( -row => 1, -column => 1, -sticky => 'w', );
        $dpca_frame -> Entry( -textvariable => \$dpca_temp,)
                            -> grid( -row => 1, -column => 3, );
        $dpca_frame -> Label( -text => 'Total principal components to calculate: ',
                            -anchor => 'e',)
                            -> grid( -row => 2, -column => 1, -sticky => 'w', );
        $dpca_frame -> Entry( -textvariable => \$dpca_eigenvectors,)
                            -> grid( -row => 2, -column => 3, );
        $dpca_frame -> Label( -text => 'Number of components to use in PC(i)-PC(j) plots: ',
                            -anchor => 'e', )
                            -> grid( -row => 3, -column => 1, -sticky => 'w', );
        $dpca_frame -> Entry( -textvariable => \$dpca_combinations,)
                            -> grid( -row => 3, -column => 3, );
        $dpca_frame -> Label( -text => "Sigma cutoff ( automatically determined if left undefined ):
",
                            -anchor => 'e',)
                            -> grid( -row => 4, -column => 1, -sticky => 'w', );
        $dpca_frame -> Entry( -textvariable => \$dpca_cutoff,)
                            -> grid( -row => 4, -column => 3, );
        $dpca_frame -> Label( -text => 'First frame to use: ',
                            -anchor => 'w', )
                            -> grid( -row => 5, -column => 1, -sticky => 'w',);
        $dpca_frame -> Entry( -textvariable => \$dpca_first, )
                            -> grid( -row => 5, -column => 3, );
        $dpca_frame -> Label( -text => 'Last frame to use: ',
                            -anchor => 'w', )
                            -> grid( -row => 6, -column => 1, -sticky => 'w',);
        $dpca_frame -> Entry( -textvariable => \$dpca_last, )
                            -> grid( -row => 6, -column => 3, );
```

```
        $dpca_frame -> Label( -text => 'Stride (step) between frames: ',
                              -anchor => 'w', )
                              -> grid( -row => 7, -column => 1, -sticky => 'w',);
        $dpca_frame -> Entry( -textvariable => \$dpca_step, )
                              -> grid( -row => 7, -column => 3, );

        $dpca_frame -> Checkbutton( -text => 'Set limits for PC(i)-PC(j) plots: ',
                                    -anchor => 'e',
                                    -variable => \$dpca_dgwidth,
                                    -onvalue => " -dgwidth",
                                    -offvalue => '', )
                                    -> grid( -row => 8, -column => 1, -sticky => 'w', );
        $dpca_frame -> Entry( -textvariable => \$dpca_dgwidth_num,)
                              -> grid( -row => 8, -column => 3, );

        #~ $dpca_frame -> Label( -text => "For dPCA your atom type selection will be ignored.", )
                              #~ -> grid( -row => 9, -column => 2, );

        $dpca_frame_1 = $dpca_top -> Frame() -> pack( -fill => 'x', );
        my $dpca_frame_2 = $dpca_top -> Frame() -> pack( -fill => 'x', );

        checkbuttons ( $dpca_frame_1 );
        otherbuttons ( $dpca_frame_2 );

        my $dpca_frame_3 = $dpca_top -> Frame() -> pack( -side => 'top', -expand => 1, -fill => 'x',
);
        my $dpca_frame_4 = $dpca_top -> Frame() -> pack( -side => 'top', -expand => 1, -fill => 'x',
);

        $dpca_frame -> pack( -side => 'top', -expand => 1, -fill => 'both', );
         my $dpca_frame_5 = $dpca_top -> Frame() -> pack( -side => 'top', -expand => 1, -fill =>
'none', );

        $dpca_frame_3 -> Label( -text => "\nVarious Options\n", -font => $font_20, )
                               -> grid( -row => 1, -column => 1, );

        $dpca_auto_entry = $dpca_frame_4 -> Entry( -textvariable => \$dpca_auto_entry_num,
                                                   -state => 'disabled', )
                                                   -> grid( -row => 2, -column => 2, );

        $dpca_frame_4 -> Label( -text => 'clusters', ) -> grid( -row => 2, -column => 3, );

        my $dpca_clustering_b = $dpca_frame_4 -> Checkbutton( -text => 'Automatically isolate max:
',
                                                              -command => sub {
                                        if ( $dpca_auto_entry -> cget( -state, ) eq 'disabled' ) {
                                            $dpca_auto_entry -> configure( -state => 'normal', );
                                        }
                                        else {
                                            $dpca auto entry -> delete( 0, 'end', );
                                             $dpca_auto_entry -> configure( -state => 'disabled',
);
                                        }
                                        }, )
                                    -> grid( -row => 2, -column => 1, -sticky => 'w', );
        $dpca_clustering_b -> invoke;

        my $dpca_3d_button = $dpca_frame_4 -> Checkbutton( -text => 'Create 3D landscapes',
                                                           -variable => \$dpca_3d,
                                                           -offvalue => '',
                                                           -onvalue => " -3d", )
                                                           -> grid( -row => 3, -column => 1, -sticky
=> 'w', );

        $dpca_3d_button -> invoke if ( $vmd );

          $dpca_frame_4    ->    Checkbutton(    -text    =>    'Include    chi1    angles    in    the
analysis                          ',
                                -variable => \$chi1,
                                -offvalue => '',
                                -onvalue => " -chi1", )
                                -> grid( -row => 4, -column => 1, -sticky => 'w', );

        $dpca_frame_5 -> Button( -text => 'Return',
```

```
                                -command => sub {
                               $dpca_top -> withdraw;
                               }, )
                              -> pack( -side => 'left', );

        $dpca_run_button = $dpca_frame_5 ->Button( -text => 'Run',
                                                   -command => sub {
            $dpca_first_flag = ( ( $dpca_first != 1 ) ? " -first $dpca_first" : '' );
            $dpca_last_flag = ( ( $dpca_last != $header ) ? " -last $dpca_last" : '' );
            $dpca_step_flag = ( ( $dpca_step != 1 ) ? " -step $dpca_step" : '' );

             open  PCA_FIRST_STEP,  '>',  'pca_first_step'  or  die  "Cannot  open  pca_first_step  for
writing : $!\n";

            printf PCA_FIRST_STEP "%8d", $dpca_last;

            close PCA_FIRST_STEP;

            $dpca_top -> withdraw;

            $seg_id_flag = '' if $seg_id_flag;

            foreach ( @seg_ids ) {
                if ( defined ( $_ ) ) {
                    $seg_id_flag = $seg_id_flag . $_;
                }
            }

            if ( $res_id_flag ) {
                if ( $dpca_auto_entry -> cget( -state, ) eq 'normal' ) {
                    my $response;
                    if ( $linux or $mac ) {
                        $response = $mw -> messageBox( -message => 'You have specified a residue
selection. Would you ' .
                                                       'like  the  output  PDB  files  to
include all atoms ?',
                                                   -type => 'yesno',
                                                   -icon => 'question',
                                                   -font => "$font_12", );
                    }
                    else {
                        $response = $mw -> messageBox( -message => 'You have specified a residue
selection. Would you ' .
                                                       'like  the  output  PDB  files  to
include all atoms ?',
                                                   -type => 'yesno',
                                                   -icon => 'question', );
                    }

                    if ( $response =~ /yes/i ) {
                        $include_segid = 1;
                    }
                    else {
                        $include_segid = 0;
                        $seg_id_flag = '';
                    }

                     $flag = " -v -w -col $dpca_first_flag $dpca_last_flag $dpca_step_flag $dpca_3d
$dpca_dgwidth  $dpca_dgwidth_num  $chi1  $res_id_flag  -dPCA  $dpca_eigenvectors  $dpca_combinations
$dpca_temp $dpca_cutoff";
                }
                elsif ( $seg_id_flag ) {
                    my $response;
                    if ( $mac or $linux ) {
                        $response = $mw -> messageBox( -message => 'You have specified a residue
selection. Would you ' .
                                                       'like  the  principal  component
analysis to include' .
                                                      ' all atoms ?',
                                                   -type => 'yesno',
                                                   -icon => 'question',
                                                   -font => "$font_12", );
                    }
                    else {
```

```
                        $response = $mw -> messageBox( -message => 'You have specified a residue
selection. Would you ' .
                                                                    'like the principal component
analysis to include' .
                                                                   ' all atoms ?',
                                                          -type => 'yesno',
                                                          -icon => 'question', );
                    }

                    if ( $response =~ /yes/i ) {
                        $flag = " -v -w -col $dpca_first_flag $dpca_last_flag $dpca_step_flag
$dpca_3d $dpca_dgwidth $dpca_dgwidth_num $chi1 $res_id_flag $seg_id_flag -dPCA $dpca_eigenvectors
$dpca_combinations $dpca_temp $dpca_cutoff";
                    }
                    else {
                        $flag = " -v -w -col $dpca_first_flag $dpca_last_flag $dpca_step_flag
$dpca_3d $dpca_dgwidth $dpca_dgwidth_num $chi1 $res_id_flag -dPCA $dpca_eigenvectors
$dpca_combinations $dpca_temp $dpca_cutoff";
                    }
                }
                else {
                    $flag = " -v -w -col $dpca_first_flag $dpca_last_flag $dpca_step_flag $dpca_3d
$dpca_dgwidth $dpca_dgwidth_num $chi1 $res_id_flag -dPCA $dpca_eigenvectors $dpca_combinations
$dpca_temp $dpca_cutoff";
                }
            }
            else {
                $flag = " -v -w -col $dpca_first_flag $dpca_last_flag $dpca_step_flag $dpca_3d
$dpca_dgwidth $dpca_dgwidth_num $chi1 $seg_id_flag -dPCA $dpca_eigenvectors $dpca_combinations
$dpca_temp $dpca_cutoff";
            }

            my $mess_check = 1;

            create_dir();
            if ( $dpca_auto_entry -> cget( -state, ) eq 'normal' ) {
                if ( $active_dcd =~ /dPCA.fitted.cluster_\d\d.dcd/ ) {
                    $mess_check = 0;

                    my $response;
                    if ( $mac or $linux ) {
                        $response = $dpca_top -> messageBox( -message => 'This operation may remove
and/or overwrite the active DCD/PSF pair. Doing' .
                                                                    ' so will either lead to
the calculation failing to complete, or worse, to ' .
                                                                   'corrupting your files. Are
you sure you want to proceed ? Select "No" '.
                                                                    'to abort this operation
(which will give you the chance to copy and/or rename' .
                                                                    ' the active DCD/PSF pair
to something different).',
                                                          -icon => 'warning',
                                                          -type => 'yesno',
                                                          -font => "$font_12",
                                                          -fg => 'red', );
                    }
                    else {
                        $response = $dpca_top -> messageBox( -message => 'This operation may remove
and/or overwrite the active DCD/PSF pair. Doing' .
                                                                    ' so will either lead to
the calculation failing to complete, or worse, to ' .
                                                                   'corrupting your files. Are
you sure you want to proceed ? Select "No" '.
                                                                    'to abort this operation
(which will give you the chance to copy and/or rename' .
                                                                    ' the active DCD/PSF pair
to something different).',
                                                          -icon => 'warning',
                                                          -type => 'yesno', );
                    }


                    if ( $response =~ /yes/i ) {
                        $mess_check = 1;
```

```
                }
            }
            elsif ( -f 'dPCA.fitted.cluster_01.dcd.varcov.dat' ) {
                my $response;
                if ( $mac or $linux ) {
                    $response = $dpca_top -> messageBox( -message => 'The analysis you are about
to perform will produce files ' .
                                                                    'that already exist in the
working directory. Would you like to' .
                                                                     ' delete the old files
before proceeding ?',
                                                        -icon => 'question',
                                                        -type => 'yesno',
                                                        -font => "$font_12", );
                }
                else {
                    $response = $dpca_top -> messageBox( -message => 'The analysis you are about
to perform will produce files ' .
                                                                    'that already exist in the
working directory. Would you like to' .
                                                                     ' delete the old files
before proceeding ?',
                                                        -icon => 'question',
                                                        -type => 'yesno', );
                }

                if ( $response =~ /yes/i ) {
                    opendir CWD_AGAIN, getcwd || die "Cannot open " . getcwd . ": $!";
                    while ( my $dh1 = readdir CWD_AGAIN ) {
                        if ( $dh1 =~ /\w*dPCA\w*/ and $dh1 ne $active_psf and $dh1 ne
$active_dcd ) {
                            unlink ( $dh1 );
                        }
                    }

                    closedir CWD_AGAIN;
                }
            }
        }

        if ( $mess_check ) {
            $text -> insert( 'end', "\n\nNow performing dPCA.\n\n", 'cyan', );
            $text -> see( 'end', );
            $mw -> update;

            carma ( "pca" );
            if ( $all_done ) {
                if ( -f "carma.variance_explained.dat" ) {
                    mv ( "carma.variance_explained.dat", "dPCA.variance_explained.dat" );

                    open VARIANCE_IN, '<', "dPCA.variance_explained.dat" or die "Cannot open
dPCA.variance_explained.dat for reading : $!\n";
                    open VARIANCE_OUT1, '>', "dPCA.clusters_vs_variance_explained.dat" or die
"Cannot open clusters_vs_variance_explained.dat for reading : $!\n";
                    open VARIANCE_OUT2, '>', "dPCA.clusters_vs_rms_cutoff.dat" or die "Cannot
open rms_cutoff_vs_variance_explained.dat for reading : $!\n";

                    while ( <VARIANCE_IN> ) {
                        chomp;

                        if ( /\s*(\S+)\s+(\S+)\s+(\S+)/ ) {
                            print VARIANCE_OUT1 "$1 $2\n";
                            print VARIANCE_OUT2 "$1 $3\n";
                        }
                    }

                    close VARIANCE_OUT2;
                    close VARIANCE_OUT1;
                    close VARIANCE_IN;
                }

                if ( -f "carma.3d_landscape.cns" ) {
                    mv ( "carma.3d_landscape.cns", "dPCA.3d_landscape.cns" );
                }
```

```
                    opendir CWD, getcwd || die "Cannot open " . getcwd . ": $!";
                    while ( my $dh = readdir CWD ) {
                        if ( $dh =~ /carma.dPCA.DG_(\d+)_(\d+).(\w+)/ ) {
                            mv ( "$dh", "dPCA.PC$1_vs_$2.$3" );
                        }
                        elsif ( $dh =~ /\w+.dcd.dPCA.varcov.(\w+)/ ) {
                            mv ( "$dh", "dPCA.covariance.$1" );
                        }
                        elsif ( $dh =~ /carma.dPCA.eigenvalues.dat/ ) {
                            mv ( "$dh", "dPCA.eigenvalues.dat" );
                        }
                        elsif ( $dh =~ /carma.clusters.dat/ ) {
                            mv ( "$dh", "dPCA.clusters.dat" );
                        }
                    }
                    closedir CWD;

                    unless ( $dpca_auto_entry -> cget( -state, ) eq 'normal' ) {
                         $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n",
'valid' );
                        $text -> see( 'end', );
                        $image_menu -> configure( -state => 'normal', );
                        $all_done = '';
                    }

                    if ( $dpca_auto_entry -> cget( -state, ) eq 'normal' ) {
                        auto_window ( 'dPCA' ) ;

                        if ( $all_done ) {
                            foreach ( @cluster_stats ) {
                                $text -> insert( 'end', "$_\n", );
                                $text -> see( 'end', );
                            }

                              $text -> insert( 'end', "\nCalculation finished. Use \"View
Results\"\n", 'valid' );
                            $text -> see( 'end', );
                            $image_menu -> configure( -state => 'normal', );
                            $all_done = '';
                        }
                        else {
                             $text -> insert( 'end', "\nSomething went wrong. For details check
last_carma_run.log located in :\n", 'error', );
                            $text -> insert( 'end', getcwd . "\n", 'info', );
                            $text -> see( 'end', );
                        }
                    }
                }
                else {
                     $text -> insert( 'end', "\nSomething went wrong. For details check
last_carma_run.log located in :\n", 'error', );
                    $text -> insert( 'end', getcwd . "\n", 'info', );
                    $text -> see( 'end', );
                }
            }
        }, ) -> pack( -side => 'right', );

        if ( $active_run_buttons ) {
            $dpca_run_button -> configure( -state => 'normal', );
        }
        else {
            $dpca_run_button -> configure( -state => 'disabled', );
        }
    }
    else {
        $dpca_top -> deiconify;
        $dpca_top -> raise;
    }

}

###########################################################################################################
###   Draw the window for the cPCA calculation                                                          ###
```

```
##########################################################################################################

sub cpca_window {
    my $cpca_eigenvectors = '';
    my $cpca_combinations = '';
    my $cpca_temp = '';
    my $cpca_cutoff = '';
    my $cpca_dgwidth = '';
    my $cpca_dgwidth_num = '';
    my $cpca_first = 1;
    my $cpca_first_flag = '';
    my $cpca_last = dcd_header_parser( "cpca" );
    my $cpca_last_flag = '';
    my $cpca_step = 1;
    my $cpca_step_flag = '';
    my $cpca_3d = '';
    my $chi1 = '';
    my $cpca_mass = '';
    my $cpca_use = '';
    my $cpca_top;
    our $include_segid;
    our $cpca_auto_entry;
    our $cpca_auto_entry_num = 10;
    our @cluster_stats = '';

    if ( !Exists( $cpca_top ) ) {
        unless ( $cpca_temp ) {
            $cpca_temp = 298;
            $cpca_eigenvectors = 5;
            $cpca_combinations = 3;
        }

        $cpca_top = $mw -> Toplevel( -title => 'Cartesian PCA ' );
        $cpca_top -> geometry("$toplevel_position");

        $cpca_frame = $cpca_top -> Frame( -borderwidth => 2, -relief => 'groove', );

        $cpca_frame -> Label( -text => 'Temperature(K): ',)
                            -> grid( -row => 1, -column => 1, -sticky => 'w', );
        $cpca_frame -> Entry( -textvariable => \$cpca_temp,)
                            -> grid( -row => 1, -column => 3, );

        $cpca_frame -> Label( -text => 'Total principal components to calculate: ',
                            -anchor => 'e',)
                            -> grid( -row => 2, -column => 1, -sticky => 'w', );
        $cpca_frame -> Entry( -textvariable => \$cpca_eigenvectors,)
                            -> grid( -row => 2, -column => 3, );
        $cpca_frame -> Label( -text => 'Number of components to use in PC(i)-PC(j) plots: ',
                            -anchor => 'e',)
                            -> grid( -row => 3, -column => 1, -sticky => 'w', );
        $cpca_frame -> Entry( -textvariable => \$cpca_combinations,)
                            -> grid( -row => 3, -column => 3, );
        $cpca_frame -> Label( -text => 'Sigma cutoff ( automatically determined if left undefined ):
',
                            -anchor => 'e',)
                            -> grid( -row => 4, -column => 1, -sticky => 'w', );
        $cpca_frame -> Entry( -textvariable => \$cpca_cutoff,)
                            -> grid( -row => 4, -column => 3, );
        $cpca_frame -> Label( -text => 'First frame to use: ', )
                            -> grid( -row => 5, -column => 1, -sticky => 'w',);
        $cpca_frame -> Entry( -textvariable => \$cpca_first, )
                            -> grid( -row => 5, -column => 3, );
        $cpca_frame -> Label( -text => 'Last frame to use: ', )
                            -> grid( -row => 6, -column => 1, -sticky => 'w',);
        $cpca_frame -> Entry( -textvariable => \$cpca_last, )
                            -> grid( -row => 6, -column => 3, );
        $cpca_frame -> Label( -text => 'Stride (step) between frames: ', )
                            -> grid( -row => 7, -column => 1, -sticky => 'w',);
        $cpca_frame -> Entry( -textvariable => \$cpca_step, )
                            -> grid( -row => 7, -column => 3, );
        $cpca_frame -> Checkbutton( -text => 'Set limits for PC(i)-PC(j) plots: ',
                                    -anchor => 'e',
                                    -variable => \$cpca_dgwidth,
                                    -onvalue => " -dgwidth",
```

```
                                -offvalue => '', )
                                -> grid( -row => 8, -column => 1, -sticky => 'w', );
        $cpca_frame -> Entry( -textvariable => \$cpca_dgwidth_num,)
                              -> grid( -row => 8, -column => 3, );

        $cpca_frame_1 = $cpca_top -> Frame() ->pack( -fill => 'x', );
        my $cpca_frame_2 = $cpca_top -> Frame() -> pack( -fill => 'x', );
        my $cpca_frame_3 = $cpca_top -> Frame() -> pack( -fill => 'x', );
        my $cpca_frame_5 = $cpca_top -> Frame() -> pack( -side => 'top', -expand => 1, -fill => 'x',
);
        my $cpca_frame_4 = $cpca_top -> Frame() -> pack( -side => 'top', -expand => 1, -fill =>
'both', );

        radiobuttons ( $cpca_frame_1 );
        checkbuttons ( $cpca_frame_2 );
        otherbuttons ( $cpca_frame_3 );

        $cpca_frame -> pack( -side => 'top', -expand => 1, -fill => 'both', );
        my $cpca_frame_6 = $cpca_top -> Frame()-> pack( -side => 'top', );

        $cpca_frame_5 -> Label( -text => "\nVarious Options", -font => $font_20, ) -> pack;

        $cpca_auto_entry = $cpca_frame_4 -> Entry( -textvariable => \$cpca_auto_entry_num,
                                                   -state => 'disabled', )
                                                   -> grid( -row => 2, -column => 1, -sticky => 'w',
);

        $cpca_frame_4 -> Label( -text => 'clusters', ) -> grid( -row => 2, -column => 2, -sticky =>
'w', );

        my $cpca_clustering_b = $cpca_frame_4 -> Checkbutton( -text => 'Automatically isolate max:
',
                                                              -command => sub {
          if ( $cpca_auto_entry -> cget( -state, ) eq 'disabled' ) {
              $cpca_auto_entry -> configure( -state => 'normal', );
          }
          else {
              $cpca_auto_entry -> delete( 0, 'end', );
              $cpca_auto_entry -> configure( -state => 'disabled', );
          }
          }, )
        -> grid( -row => 2, -column => 0, -sticky => 'w', );

        $cpca_clustering_b -> invoke;

        $cpca_frame_4 -> Checkbutton( -text => 'Use mass weighting',
                                      -variable => \$cpca_mass,
                                      -offvalue => '',
                                      -onvalue => " -mass", )
                                      -> grid( -row => 3, -column => 0, -sticky => 'w', );
        my $cpca_3d_button = $cpca_frame_4 -> Checkbutton( -text => 'Create 3D landscapes',
                                                           -variable => \$cpca_3d,
                                                           -offvalue => '',
                                                           -onvalue => " -3d", )
                                                           -> grid( -row => 4, -column => 0, -sticky
=> 'w', );

        $cpca_3d_button -> invoke if ( $vmd );

        $cpca_frame_4 -> Checkbutton( -text => 'Use previously calculated eigenvalues             ',
                                      -variable => \$cpca_use,
                                      -offvalue => '',
                                      -onvalue => " -use", )
                                      -> grid( -row => 5, -column => 0, -sticky => 'w', );

        $cpca_frame_6 ->Button( -text => 'Return',
                                -command => sub {
                                $cpca_top -> withdraw;
                                }, )
                               -> pack( -side => 'left', );

        $cpca_frame_6 ->Button( -text => 'Run',
                                -command => sub {
            $cpca_first_flag = ( ( $cpca_first != 1 ) ? " -first $cpca_first" : '' );
```

```
            $cpca_last_flag = ( ( $cpca_last != $header ) ? " -last $cpca_last" : '' );
            $cpca_step_flag = ( ( $cpca_step != 1 ) ? " -step $cpca_step" : '' );

             open  PCA_FIRST_STEP,  '>',  'pca_first_step'  or  die  "Cannot  open  pca_first_step  for
writing : $!\n";

            printf PCA_FIRST_STEP "%8d", $cpca_last;

            close PCA_FIRST_STEP;

            $cpca_top -> withdraw;

            $seg_id_flag = '' if $seg_id_flag;

            foreach ( @seg_ids ) {
                if ( defined ( $_ ) ) {
                    $seg_id_flag = $seg_id_flag . $_;
                }
            }

            if ( $res_id_flag ) {
                if ( $cpca_auto_entry -> cget( -state, ) eq 'normal' ) {
                    my $response;
                    if ( $linux or $mac ) {
                        $response = $mw -> messageBox( -message => 'You have specified a residue
selection. Would you ' .
                                                       'like  the  output  PDB  files  to
include all atoms ?',
                                                   -type => 'yesno',
                                                   -icon => 'question',
                                                   -font => "$font_12", );
                    }
                    else {
                        $response = $mw -> messageBox( -message => 'You have specified a residue
selection. Would you ' .
                                                       'like  the  output  PDB  files  to
include all atoms ?',
                                                   -type => 'yesno',
                                                   -icon => 'question', );
                    }

                    if ( $response =~ /yes/i ) {
                        $include_segid = 1;
                    }
                    else {
                        $include_segid = 0;
                        $seg_id_flag = '';
                    }

                     $flag  =  "  -v  -w  -col  -cov  $cpca_first_flag  $cpca_last_flag  $cpca_step_flag
$res_id_flag   $cpca_dgwidth   $cpca_dgwidth_num   $atm_id_flag   $custom_id_flag   -eigen   -proj
$cpca_eigenvectors $cpca_combinations $cpca_temp $cpca_cutoff $cpca_mass $cpca_3d $cpca_use";
                }
                elsif ( $seg_id_flag ) {
                    my $response;
                    if ( $linux or $mac ) {
                        $response = $mw -> messageBox( -message => 'You have specified a residue
selection. Would you ' .
                                                       'like  the  principal  component
analysis to include' .
                                                       ' all atoms ?',
                                                   -type => 'yesno',
                                                   -icon => 'question',
                                                   -font => "$font_12", );
                    }
                    else {
                        $response = $mw -> messageBox( -message => 'You have specified a residue
selection. Would you ' .
                                                       'like  the  principal  component
analysis to include' .
                                                       ' all atoms ?',
                                                   -type => 'yesno',
                                                   -icon => 'question', );
                    }
```

```
                    if ( $response =~ /yes/i ) {
                        $flag = " -v -w -col -cov $cpca_first_flag $cpca_last_flag $cpca_step_flag
$res_id_flag $cpca_dgwidth $cpca_dgwidth_num $atm_id_flag $seg_id_flag $custom_id_flag -eigen -proj
$cpca_eigenvectors $cpca_combinations $cpca_temp $cpca_cutoff $cpca_mass $cpca_3d $cpca_use";
                    }
                    else {
                        $flag = " -v -w -col -cov $cpca_first_flag $cpca_last_flag $cpca_step_flag
$res_id_flag   $cpca_dgwidth   $cpca_dgwidth_num   $atm_id_flag   $custom_id_flag   -eigen   -proj
$cpca_eigenvectors $cpca_combinations $cpca_temp $cpca_cutoff $cpca_mass $cpca_3d $cpca_use";
                    }
                }
                else {
                    $flag = " -v -w -col -cov $cpca_first_flag $cpca_last_flag $cpca_step_flag
$res_id_flag   $cpca_dgwidth   $cpca_dgwidth_num   $atm_id_flag   $custom_id_flag   -eigen   -proj
$cpca_eigenvectors $cpca_combinations $cpca_temp $cpca_cutoff $cpca_mass $cpca_3d $cpca_use";
                }
            }
            else {
                $flag = " -v -w -col -cov $cpca_first_flag $cpca_last_flag $cpca_step_flag
$cpca_dgwidth   $cpca_dgwidth_num   $atm_id_flag   $seg_id_flag   $custom_id_flag   -eigen   -proj
$cpca_eigenvectors $cpca_combinations $cpca_temp $cpca_cutoff $cpca_mass $cpca_3d $cpca_use";
            }

            create_dir();

            my $mess_check = 1;

            if ( $cpca_auto_entry -> cget( -state, ) eq 'normal' ) {
                if ( $active_dcd =~ /cPCA.fitted.cluster_\d\d.dcd/ ) {
                    $mess_check = 0;

                    my $response;
                    if ( $mac or $linux ) {
                        $response = $cpca_top -> messageBox( -message => 'This operation may remove
and/or overwrite the active DCD/PSF pair. Doing' .
                                                             ' so will either lead to
the calculation failing to complete, or worse, to ' .
                                                             'corrupting your files. Are
you sure you want to proceed ? Select "No" '.
                                                             'to abort this operation
(which will give you the chance to copy and/or rename' .
                                                             ' the active DCD/PSF pair
to something different).',
                                                    -icon => 'warning',
                                                    -type => 'yesno',
                                                    -font => "$font_12",
                                                    -fg => 'red', );
                    }
                    else {
                        $response = $cpca_top -> messageBox( -message => 'This operation may remove
and/or overwrite the active DCD/PSF pair. Doing' .
                                                             ' so will either lead to
the calculation failing to complete, or worse, to ' .
                                                             'corrupting your files. Are
you sure you want to proceed ? Select "No" '.
                                                             'to abort this operation
(which will give you the chance to copy and/or rename' .
                                                             ' the active DCD/PSF pair
to something different).',
                                                    -icon => 'warning',
                                                    -type => 'yesno', );
                    }


                    if ( $response =~ /yes/i ) {
                        $mess_check = 1;
                    }
                }
                elsif ( -f 'cPCA.fitted.cluster_01.dcd.varcov.dat' ) {
                    my $response;
                    if ( $linux or $mac ) {
                        $response = $cpca_top -> messageBox( -message => 'The analysis you are about
to perform will produce files ' .
```

```
                                                                    'that already exist in the
working directory. Would you like to' .
                                                                    '  delete  the  old  files
before proceeding ?',
                                                          -icon => 'question',
                                                          -type => 'yesno',
                                                          -font => "$font_12", );
                    }
                    else {
                        $response = $cpca_top -> messageBox( -message => 'The analysis you are about
to perform will produce files ' .
                                                                    'that already exist in the
working directory. Would you like to' .
                                                                    '  delete  the  old  files
before proceeding ?',
                                                          -icon => 'question',
                                                          -type => 'yesno', );
                    }
                    if ( $response =~ /yes/i ) {
                        opendir CWD_AGAIN, getcwd || die "Cannot open " . getcwd . ": $!";
                        while ( my $dh1 = readdir CWD_AGAIN ) {
                             if  (  $dh1  =~  /\w*cPCA\w*/  and  $dh1  ne  $active_psf  and  $dh1  ne
$active_dcd ) {
                                 unlink ( $dh1 );
                            }
                        }

                        closedir CWD_AGAIN;
                    }
                }
            }

            if ( $mess_check ) {
                $text -> insert( 'end', "\n\nNow performing cPCA.\n\n", 'cyan', );
                $text -> see( 'end', );
                $mw -> update;

                carma ( "pca" );

                if ( $all_done ) {
                    if ( -f "carma.variance_explained.dat" ) {
                        mv ( "carma.variance_explained.dat", "cPCA.variance_explained.dat" );

                         open  VARIANCE_IN,  '<',  "cPCA.variance_explained.dat"  or  die  "Cannot  open
cPCA.variance_explained.dat for reading : $!\n";
                         open  VARIANCE_OUT1,  '>',  "cPCA.clusters_vs_variance_explained.dat"  or  die
"Cannot open clusters_vs_variance_explained.dat for reading : $!\n";
                         open  VARIANCE_OUT2,  '>',  "cPCA.clusters_vs_rms_cutoff.dat"  or  die  "Cannot
open rms_cutoff_vs_variance_explained.dat for reading : $!\n";

                        while ( <VARIANCE_IN> ) {
                            chomp;

                            if ( /\s*(\S+)\s+(\S+)\s+(\S+)/ ) {
                                print VARIANCE_OUT1 "$1 $2\n";
                                print VARIANCE_OUT2 "$1 $3\n";
                            }
                        }

                        close VARIANCE_OUT2;
                        close VARIANCE_OUT1;
                        close VARIANCE_IN;
                    }

                    if ( -f "carma.3d_landscape.cns" ) {
                        mv ( "carma.3d_landscape.cns", "cPCA.3d_landscape.cns" );
                    }

                    opendir CWD, getcwd || die "Cannot open " . getcwd . ": $!";
                    while ( my $dh = readdir CWD ) {
                        if ( $dh =~ /carma.PCA.DG_(\d+)_(\d+).(\w+)/ ) {
                            mv ( "$dh", "cPCA.PC$1_vs_$2.$3" );
                        }
                        elsif ( $dh =~ /\w+.dcd.varcov.(\w+)/ ) {
```

```
                        mv ( "$dh", "cPCA.covariance.$1" );
                    }
                    elsif ( $dh =~ /carma.PCA.eigenvalues.dat/ ) {
                        mv ( "$dh", "cPCA.eigenvalues.dat" );
                    }
                    elsif ( $dh =~ /carma.clusters.dat/ ) {
                        mv ( "$dh", "cPCA.clusters.dat" );
                    }
                }
                closedir CWD;

                unless ( $cpca_auto_entry -> cget( -state, ) eq 'normal' ) {
                    $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n",
'valid' );
                    $text -> see( 'end', );
                    $image_menu -> configure( -state => 'normal', );
                    $all_done = '';
                }

                if ( $cpca_auto_entry -> cget( -state, ) eq 'normal' ) {
                    auto_window ( 'cPCA' );
                    if ( $all_done ) {
                        foreach ( @cluster_stats ) {
                            $text -> insert( 'end', "$_\n", );
                            $text -> see( 'end', );
                        }

                          $text -> insert( 'end', "\nCalculation finished. Use \"View
Results\"\n", 'valid' );
                        $text -> see( 'end', );
                        $image_menu -> configure( -state => 'normal', );
                        $all_done = '';
                    }
                    else {
                         $text -> insert( 'end', "\nSomething went wrong. For details check
last carma run.log located in :\n", 'error', );
                        $text -> insert( 'end', getcwd . "\n", 'info', );
                        $text -> see( 'end', );
                    }
                }
            }
            else {
                 $text -> insert( 'end', "\nSomething went wrong. For details check
last_carma_run.log located in :\n", 'error', );
                $text -> insert( 'end', getcwd . "\n", 'info', );
                $text -> see( 'end', );
            }
        }
    }, ) -> pack( -side => 'right', );
}
else {
    $cpca_top -> deiconify;
    $cpca_top -> raise;
}
}

##############################################################################################
###   Automated cluster analysis                                                           ###
##############################################################################################

sub auto_window {
    our $prev_psf;

    my $input = shift;

    our $dpca_auto_entry;
    our $dpca_auto_entry_num;
    our $cpca_auto_entry;
    our $cpca_auto_entry_num;

    our $chi1;
    our $include_segid;

    our @cluster_stats;
```

```
    my $cluster_number;
    my $cluster_size = 0;

    # The psf/dcd files that are in use at the time the subroutine is called
    # This is necessary because the active files will change several times
    # as the subroutine is executed, and they need to revert to the original
    # ones when it is done
    my ( $remember_psf, $remember_dcd, ) = ( $active_psf, $active_dcd, );

    # $clusters is the number of clusters the user defined in the dpca or cpca panels
    # @clusters is the number of clusters contained in the file 'clusters.dat'
    my $clusters;
    my @clusters;

    my $fit_check = 0;
    my $super_check = 0;

    create_dir();

     open CLUSTERS, "$input\.clusters.dat" || die "Cannot open $input\.clusters.dat for reading:
$!\n";

    my $i = 0;
    while ( <CLUSTERS> ) {
        if ( /(\s*\d+\s*)(\d*)(.*)/ ) {
            $clusters[$i] = $2;
            $i++;
        }
    }
    close CLUSTERS;

    @clusters = uniq ( @clusters );
    $clusters = @clusters;

    # If the number of clusters the user desires the cluster analysis to
    # be performed on, exceeds that of the number of clusters detailed in
    # the file 'clusters.dat'
    if ( $input eq 'cPCA' && $clusters > $cpca_auto_entry_num ) {
        $clusters = $cpca_auto_entry_num;
    }
    elsif ( $input eq 'dPCA' && $clusters > $dpca_auto_entry_num ) {
        $clusters = $dpca_auto_entry_num;
    }

    for ( $i = 1 ; $i <= $clusters ; $i++ ) {
         open CLUSTERS, '<', "$input\.clusters.dat" || die "Cannot open $input\.clusters.dat for
reading: $!\n";

        $i = sprintf ( "%02d", $i );

        my $file = "C_$i.dat";
        open OUT, '>', $file || die "Cannot open $file for writing\n: $!";

        while ( <CLUSTERS> ) {
            if ( /^(\s*\d*\s*)(\d*)(.*)/ ) {
                if ( $2 == $i ) {
                    print OUT "$1$2\n";
                    $cluster_size++;
                }
            }
        }

        close OUT;
        close CLUSTERS;

        $cluster_number = sprintf ( "%3d", $i, );
        $cluster_size = sprintf ( "%7d", $cluster_size, );
        push ( @cluster_stats, "Cluster $cluster_number : $cluster_size frames (out of $header)" );

        $cluster_size = 0;

        if ( $i == 1 ) {
            $text -> insert( 'end', "\nNow performing cluster extraction.\n", 'cyan', );
            $text -> see( 'end', );
```

```
            $mw -> update;
        }

        $text -> insert( 'end', "\nCluster $i :\n", 'cyan', );

        if ( $linux || $mac ) {
            $text -> insert( 'end', "\ncarma -v -sort $file $remember_dcd\n", 'info', );
            $text -> see( 'end', );
            $mw -> update;

            `carma -v -sort $file $remember_dcd`;
            `mv carma.reordered.dcd $input.cluster_$i.dcd`;
        }
        else {
            $text -> insert( 'end', "\ncarma.exe -v -sort $file $remember_dcd\n", 'info', );
            $text -> see( 'end', );
            $mw -> update;

            `carma.exe -v -sort $file $remember_dcd`;
            `move carma.reordered.dcd $input.cluster_$i.dcd`;
        }

        my $backbone = 'C|CA|N|O';

        my (
             $seg_custom, $seg_atm, $seg, $res_custom, $res_atm,
             $res,        $custom,  $atm, $nothing,    $seg_res,
        );

        if ( $include_segid ) {
            my @selected_atoms = split ' -atmid ', $custom_id_flag if ( $custom_id_flag );
            shift @selected_atoms if ( @selected_atoms );

            my $line_count = 1;
            my $regex_var = '';

            {               local $" = '|';
                if ( $input eq 'cPCA' ) {
                    if ( $custom_id_flag ) {
                        $regex_var = qr{^(\s*\d+)(\s*)(Z)(\s*\d+\s*\w+\s+)(@selected_atoms)(\s+.*)};
                    }
                    elsif ( $atm_id_flag =~ /heavy/i ) {
                        $regex_var = qr{^(\s*\d+)(\s*)(Z)(\s*\d+\s*\w+\s+)([^H].*)(\s+.*)};
                    }
                    elsif ( $atm_id_flag =~ /allid/i ) {
                        $regex_var = qr{^(\s*\d+)(\s*)(Z)(\s*\d+\s*\w+\s+)(\w+)(\s+.*)};
                    }
                    else {
                        $regex_var = qr{^(\s*\d+)(\s*)(Z)(\s*\d+\s*\w+\s+)($backbone)(\s+.*)};
                    }
                }
                else {
                     $regex_var  =  qr{^(\s*\d+)(\s*)(Z)(\s*\d+\s*\w+\s+)($backbone)(\s+.*)}  if  (
$input eq 'dPCA' );
                }
            }

             open PSF, '<', "selected_residues.psf" || die "Cannot open selected_residues.psf for
reading: $!";
            open OUT, '>', "fit.index" || die "Cannot open fit.index for writing: $!";

            while ( <PSF> ) {
                if ( /!N(BOND|THETA|PHI|IMPHI|DON|ACC|NBB|GRP)/ ) {
                    last;
                }
                elsif ( /$regex_var/i ) {
                    print OUT "$1$2$3$4$5$6\n";
                }
            }

            close OUT;
            close PSF;

            $active_dcd = "$input.cluster_$i.dcd";
```

```
            $active_psf = $prev_psf;

            $flag = " -v -w -fit -index -atmid ALLID";

            carma ( 'auto' );

            if ( $all_done ) {
                $seg_res = 1;
                $fit_check = 1;
            }
        }
        elsif ( $seg_id_flag ) {
            my @chains = split ' -segid ', $seg_id_flag;
            shift @chains;

            if ( $custom_id_flag ) {
                my @selected_atoms = split ' -atmid ', $custom_id_flag;
                shift @selected_atoms;

                my $line_count = 1;
                my $regex_var = '';

                {                   local $" = '|';
                      $regex_var    =    qr{^(\s*\d+)(\s*)(@chains)(\s*\d+\s*\w+\s+)(@selected_atoms)
(\s+.*)};
                }

                open PSF, '<', $active_psf || die "Cannot open $active_psf for reading: $!";
                open OUT, '>', "fit.index" || die "Cannot open fit.index for writing: $!";

                while ( <PSF> ) {
                    if ( /!N(BOND|THETA|PHI|IMPHI|DON|ACC|NBB|GRP)/ ) {
                        last;
                    }
                    elsif ( /$regex_var/i ) {
                        print OUT "$1$2$3$4$5$6\n";
                    }
                }

                close OUT;
                close PSF;

                $active_dcd = "$input.cluster_$i.dcd";

                $flag = " -v -w -fit -index -atmid ALLID";

                carma ( 'auto' );

                if ( $all_done ) {
                    $seg custom = 1;
                    $fit_check = 1;
                }
            }
            elsif ( $atm_id_flag ) {
                my $heavy = 0;
                $heavy = 1 if ( $atm_id_flag =~ /HEAVY/ );
                my $allid = 0;
                $allid = 1 if ( $atm_id_flag =~ /ALLID/ );

                my $line_count = 1;
                my $regex_var = '';

                {                   local $" = '|';
                    $regex_var = qr{^(\s*\d+)(\s*)(@chains)(\s*\d+\s*\w+\s+)($backbone)(\s+.*)} if (
$atm_id =~ /backbone/i );
                      $regex_var    =    qr{^(\s*\d+)(\s*)(@chains)(\s*\d+\s*\w+\s+)(\w+)(.*)}    if    (
$atm_id_flag =~ /(HEAVY|ALLID)/ );
                }

                open PSF, '<', $active_psf || die "Cannot open $active_psf for reading: $!";
                open OUT, '>', "fit.index" || die "Cannot open fit.index for writing: $!";

                while ( <PSF> ) {
                    if ( /!N(BOND|THETA|PHI|IMPHI|DON|ACC|NBB|GRP)/ ) {
```

```
                last;
            }
            elsif ( /$regex_var/i ) {
                my $line = $1 . $2 . $3 . $4 . $5 . $6;

                if ( $heavy && $5 !~ /^H/ ) {
                    print OUT "$line\n";
                }
                elsif ( $allid || $atm_id =~ /backbone/i ) {
                    printf OUT ( "%s\n", $line, );
                }
            }
        }

        close OUT;
        close PSF;

        $active_dcd = "$input.cluster_$i.dcd";

        $flag = " -v -w -fit -index -atmid ALLID";
        carma ( 'auto' );

        if ( $all_done ) {
            $seg_atm = 1;
            $fit_check = 1;
        }

        $heavy = 0;
        $allid = 0;
    }
    else {
        my $line_count = 1;
        my $regex_var = '';

        {                       local $" = '|';
            $regex_var = qr{^(\s*\d+)(\s*)(@chains)(\s*\d+\s*\w+\s+)($backbone)(\s+.*)};
        }

        open PSF, '<', $active_psf || die "Cannot open $active_psf for reading: $!";
        open OUT, '>', "fit.index" || die "Cannot open fit.index for writing: $!";

        while ( <PSF> ) {
            if ( /!N(BOND|THETA|PHI|IMPHI|DON|ACC|NBB|GRP)/ ) {
                last;
            }
            elsif ( /$regex_var/i ) {
                print OUT "$1$2$3$4$5$6\n";
            }
        }

        close OUT;
        close PSF;

        $active_dcd = "$input.cluster_$i.dcd";

        $flag = " -v -w -fit -index -atmid ALLID";
        carma ( 'auto' );

        if ( $all_done ) {
            $seg = 1;
            $fit_check = 1;
        }
    }
}
elsif ( $res_id_flag ) {
    if ( $custom_id_flag ) {
        $active_dcd = "$input.cluster_$i.dcd";
        $active_psf = "selected_residues.psf";

        if ( $input eq 'dPCA' ) {
            $flag = " -v -w -fit $res_id_flag";
        }
        else {
            $flag = " -v -w -fit $res_id_flag $custom_id_flag";
```

```
        }

        carma ( 'auto' );

        if ( $all_done ) {
            $res_custom = 1;
            $fit_check = 1;
        }
    }
    elsif ( $atm_id_flag ) {
        $active_dcd = "$input.cluster_$i.dcd";
        $active_psf = "selected_residues.psf";

        if ( $input eq 'dPCA' ) {
            $flag = " -v -w -fit $res_id_flag";
        }
        else {
            $flag = " -v -w -fit $res_id_flag $atm_id_flag";
        }

        carma ( 'auto' );

        if ( $all_done ) {
            $res_atm = 1;
            $fit_check = 1;
        }
    }
    else {
        $active_dcd = "$input.cluster_$i.dcd";
        $active_psf = "selected_residues.psf";

        if ( $input eq 'dPCA' and $chi1 ) {
            $flag = " -v -w -fit $res_id_flag -atmid ALLID";
        }
        else {
            $flag = " -v -w -fit $res_id_flag -atmid C -atmid CA -atmid N -atmid O";
        }

        carma ( 'auto' );

        if ( $all_done ) {
            $res = 1;
            $fit_check = 1;
        }
    }
}
elsif ( $custom_id_flag ) {
    my @selected_atoms = split ' -atmid ', $custom_id_flag;
    shift @selected_atoms;

    my $line_count = 1;
    my $regex_var = '';

    {               local $" = '|';
        $regex_var = qr{^(\s*\d+)(\s*)(\w+)(\s*\d+\s*\w+\s+)(@selected_atoms)(\s+.*)};
    }

    open PSF, '<', $active_psf || die "Cannot open $active_psf for reading: $!";
    open OUT, '>', "fit.index" || die "Cannot open fit.index for writing: $!\n";

    while ( <PSF> ) {
        if ( /!N(BOND|THETA|PHI|IMPHI|DON|ACC|NBB|GRP)/ ) {
            last;
        }
        elsif ( /$regex_var/i ) {
            print OUT "$1$2$3$4$5$6\n";
        }
    }

    close PSF;
    close OUT;

    $active_dcd = "$input.cluster_$i.dcd";
```

```
            $flag = " -v -w -fit -atmid ALLID -index";
            carma ( 'auto' );

            if ( $all_done ) {
                $custom = 1;
                $fit_check = 1;
            }
        }
        elsif ( $atm_id_flag ) {
            my $heavy = 0;
            $heavy = 1 if ( $atm_id_flag =~ /HEAVY/i );
            my $allid = 0;
            $allid = 1 if ( $atm_id_flag =~ /ALLID/i );

            my $line_count = 1;
            my $regex_var = '';

            $regex_var = qr{^(\s*\d+)(\s*)(\w+)(\s*\d+\s*\w+\s+)($backbone)(\s+.*)} if ( $atm_id =~
/BACKBONE/i );
            $regex_var = qr{^(\s*\d+)(\s*\w+)(\s*\d+)(\s*\w+\s+)(\w+)(.*)} if ( $atm_id_flag =~ /
(HEAVY|ALLID)/ );

            open PSF, '<', $active_psf || die "Cannot open $active_psf for reading: $!";
            open OUT, '>', "fit.index" || die "Cannot open fit.index for writing: $!";

            while ( <PSF> ) {
                if ( /!N(BOND|THETA|PHI|IMPHI|DON|ACC|NBB|GRP)/ ) {
                    last;
                }
                elsif ( /$regex_var/i ) {
                    my $line = $1 . $2 . $3 . $4 . $5 . $6;

                    if ( $heavy && $5 !~ /^H/ ) {
                        print OUT "$line\n";
                    }
                    elsif ( $allid || $atm id =~ /BACKBONE/i ) {
                        print OUT "$line\n";
                    }
                }
            }

            close OUT;
            close PSF;

            $active_dcd = "$input.cluster_$i.dcd";

            $flag = " -v -w -fit -index -atmid ALLID";
            carma ( 'auto' );

            if ( $all done ) {
                $atm = 1;
                $fit_check = 1;
            }

            $heavy = 0;
            $allid = 0;
        }
        else {
            my $line_count = 1;
            my $regex_var = '';

            $regex_var = qr{^(\s*\d+)(\s*)(\w+)(\s*\d+\s*\w+\s+)($backbone)(\s+.*)};

            open PSF, '<', $active psf || die "Cannot open $active psf for reading: $!";
            open OUT, '>', "fit.index" || die "Cannot open fit.index for writing: $!";

            while ( <PSF> ) {
                if ( /!N(BOND|THETA|PHI|IMPHI|DON|ACC|NBB|GRP)/ ) {
                    last;
                }
                elsif ( /$regex_var/i ) {
                    print OUT "$1$2$3$4$5$6\n";
                }
            }
```

```
            close OUT;
            close PSF;

            $active_dcd = "$input.cluster_$i.dcd";

            $flag = " -v -w -fit -index -atmid ALLID";
            carma ( 'auto' );

            if ( $all_done ) {
                $nothing = 1;
                $fit_check = 1;
            }
        }

        mv ( "carma.fit-rms.dat", "$input.rms_from_first_frame.cluster_$i.dat" );

        if ( $fit_check ) {
            mv ( "carma.fitted.dcd", "$input.fitted.cluster_$i.dcd" );
            mv ( "carma.selected_atoms.psf", "$input.fitted.cluster_$i.psf" );

            $active_dcd = "$input.fitted.cluster_$i.dcd";

            if ( $seg_res ) {
                if ( $input eq 'dPCA' ) {
                    if ( $chi1 ) {
                        $flag = " -v -w -col -cov -dot -norm -super -atmid HEAVY";
                    }
                    else {
                        $flag = " -v -w -col -cov -dot -norm -super -atmid C -atmid CA -atmid N
-atmid O";
                    }
                }
                else {
                    if ( $atm_id_flag ) {
                        $flag = " -v -w -col -cov -dot -norm -super $atm_id_flag";
                    }
                    else {
                        $flag = " -v -w -col -cov -dot -norm -super -atmid CA";
                    }
                }

                carma ( 'auto' );

                $seg_res = 0;
            }
            elsif ( $seg_custom || $seg_atm ) {
                 $flag = " -v -w -col -cov -dot -norm -super $seg_id_flag $custom_id_flag
$atm_id_flag";
                carma ( 'auto' );

                $seg_custom = 0;
                $seg_atm = 0;
            }
            elsif ( $seg ) {
                if ( $input eq 'dPCA' ) {
                    if ( $chi1 ) {
                        $flag = " -v -w -col -cov -dot -norm -super $seg_id_flag -atmid HEAVY";
                    }
                    else {
                        $flag = " -v -w -col -cov -dot -norm -super $seg_id_flag -atmid CA -atmid C
-atmid N -atmid O";
                    }
                }
                else {
                    $flag = " -v -w -col -cov -dot -norm -super $seg_id_flag -atmid CA";
                }
                carma ( 'auto' );

                $seg = 0;
            }
            elsif ( $res_custom || $res_atm || $res ) {
                if ( $linux or $mac ) {
                    if ( $input eq 'dPCA' ) {
```

```
                        if ( $chi1 ) {
                            `carma -v -w -last 1 -atmid ALLID -segid Z $input.cluster_$i.dcd
selected_residues.psf`;
                        }
                        else {
                            `carma -v -w -last 1 -atmid C -atmid CA -atmid N -atmid O -segid Z
$input.cluster_$i.dcd selected_residues.psf`;
                        }
                    }
                    else {
                        if ( $custom_id_flag ) {
                            `carma -v -w -last 1 $custom_id_flag -segid Z $input.cluster_$i.dcd
selected_residues.psf`;
                        }
                        elsif ( $atm_id_flag ) {
                            `carma -v -w -last 1 $atm_id_flag -segid Z $input.cluster_$i.dcd
selected_residues.psf`;
                        }
                        else {
                            `carma -v -w -last 1 -atmid C -atmid CA -atmid N -atmid O -segid Z
$input.cluster_$i.dcd selected_residues.psf`;
                        }
                    }
                }
                else {
                    if ( $input eq 'dPCA' ) {
                        if ( $chi1 ) {
                            `carma.exe -v -w -last 1 -atmid ALLID -segid Z $input.cluster_$i.dcd
selected_residues.psf`;
                        }
                        else {
                            `carma.exe -v -w -last 1 -atmid C -atmid CA -atmid N -atmid O -segid Z
$input.cluster_$i.dcd selected_residues.psf`;
                        }
                    }
                    else {
                        if ( $custom_id_flag ) {
                            `carma.exe -v -w -last 1 $custom_id_flag -segid Z $input.cluster_$i.dcd
selected_residues.psf`;
                        }
                        elsif ( $atm_id_flag ) {
                            `carma.exe -v -w -last 1 $atm_id_flag -segid Z $input.cluster_$i.dcd
selected_residues.psf`;
                        }
                        else {
                            `carma.exe -v -w -last 1 -atmid C -atmid CA -atmid N -atmid O -segid Z
$input.cluster_$i.dcd selected_residues.psf`;
                        }
                    }
                }

                unlink ( "$input.cluster_$i.dcd.0000001.ps" );

                   open    IN,    '<',    "carma.selected_atoms.psf"    or    die    "Cannot    open
carma.selected_atoms.psf for reading: $!\n";
                  open    OUT,    '>',    "new.selected_residues.psf"    or    die    "Cannot    open
new.selected_residues.psf for writing: $!\n";

                while ( <IN> ) {
                    print OUT $_;
                }

                close ( OUT );
                close ( IN );

                $active_psf = 'new.selected_residues.psf';
                $active_dcd = "$input.fitted.cluster_$i.dcd";

                if ( $input eq 'cPCA' ) {
                    if ( $atm_id_flag or $res_id_flag ) {
                        $flag = " -v -w -col -cov -dot -norm -super -segid Z $custom_id_flag
$atm_id_flag";
                    }
                    else {
```

```
                    $flag = " -v -w -col -cov -dot -norm -super -segid Z -atmid CA";
                }
            }
            else {
                if ( $chi1 ) {
                    $flag = " -v -w -col -cov -dot -norm -super -segid Z -atmid HEAVY";
                }
                else {
                    $flag = " -v -w -col -cov -dot -norm -super -segid Z -atmid CA -atmid C
-atmid N -atmid O";
                }
            }

            carma ( 'auto' );

            $res_custom = 0;
            $res_atm = 0;
            $res = 0;
        }
        elsif ( $custom || $atm ) {
            $active_dcd = "$input.fitted.cluster_$i.dcd";

            $flag = " -v -w -col -cov -dot -norm -super $custom_id_flag $atm_id_flag";
            carma ( 'auto' );

            $custom = 0;
            $atm = 0;
        }
        elsif ( $nothing ) {
            $active_dcd = "$input.fitted.cluster_$i.dcd";

            if ( $input eq 'dPCA' ) {
                $flag = " -v -w -col -cov -dot -norm -super -atmid C -atmid CA -atmid N -atmid
O";
            }
            else {
                $flag = " -v -w -col -cov -dot -norm -super -atmid CA";
            }
            carma ( 'auto' );

            $nothing = 0;
        }

        if ( $all_done ) {
            mv ( "carma.superposition.pdb", "$input.superposition.cluster_$i.pdb" );
            mv ( "carma.average.pdb", "$input.average.cluster_$i.pdb" );

            opendir CWD, getcwd || die "Cannot open " . getcwd . ": $!";
            while ( my $dh = readdir CWD ) {
                if ( $dh =~ /carma.\w+.fitted.cluster (\d+).dcd.varcov.ps/ ) {
                    mv ( "$dh", "$input.covariance.cluster_$1.ps" );
                }
            }
            closedir CWD;

            $fit_check = 0;
            $super_check = 1;
        }
        else {
              $text -> insert( 'end', "\nSomething went wrong. For details check
last_carma_run.log located in :\n", 'error', );
            $text -> insert( 'end', getcwd . "\n", 'info', );
        }

         open IN, '<', "carma.rms-average.dat" || die "Cannot open carma.rms-average.dat for
reading: $!";

        my $smallest = 1000;
        my $frame;
        while ( <IN> ) {
            if ( /^\s+(\d+)\s+(\d+\.\d+).*?$/ ) {
                if ( $2 < $smallest ) {
                    $smallest = $2;
                    $frame = $1;
```

```
                }
            }
        }

        close IN;
        mv ( "carma.rms-average.dat", "$input.rms_from_average_structure.cluster_$i.dat" );

            if ( $linux or $mac ) {
                if ( $input eq 'dPCA' ) {
                    if ( $chi1 ) {
                        $text -> insert( 'end', "carma -v -w -atmid HEAVY -first $frame -last
$frame -pdb $active_dcd $active_psf\n", 'info' );
                        `carma -v -w -atmid HEAVY -first $frame -last $frame -pdb $active_dcd
$active_psf`;
                    }
                    else {
                        $text -> insert( 'end', "carma -v -w -atmid C -atmid CA -atmid N -atmid
O -first $frame -last $frame -pdb $active_dcd $active_psf\n", 'info' );
                        `carma -v -w -atmid C -atmid CA -atmid N -atmid O -first $frame -last
$frame -pdb $active_dcd $active_psf`;
                    }
                }
                else {
                    if ( $include_segid or $atm_id_flag ) {
                        $text -> insert( 'end', "carma -v -w $atm_id_flag -first $frame -last
$frame -pdb $active_dcd $active_psf\n", 'info' );
                        `carma -v -w $atm_id_flag -first $frame -last $frame -pdb $active_dcd
$active_psf`;
                    }
                    elsif ( $custom_id_flag ) {
                        $text -> insert( 'end', "carma -v -w $custom_id_flag -first $frame -last
$frame -pdb $active_dcd $active_psf\n", 'info' );
                        `carma -v -w $custom_id_flag -first $frame -last $frame -pdb $active_dcd
$active_psf`;
                    }
                    else {
                        $text -> insert( 'end', "carma -v -w -atmid CA -first $frame -last
$frame -pdb $active_dcd $active_psf\n", 'info' );
                        `carma -v -w -atmid CA -first $frame -last $frame -pdb $active_dcd
$active_psf`;
                    }
                }
            }
            else {
                if ( $input eq 'dPCA' ) {
                    if ( $chi1 ) {
                        $text -> insert( 'end', "carma.exe -v -w -atmid HEAVY -first $frame
-last $frame -pdb $active_dcd $active_psf\n", 'info' );
                        `carma.exe -v -w -atmid HEAVY -first $frame -last $frame -pdb
$active_dcd $active_psf`;
                    }
                    else {
                        $text -> insert( 'end', "carma.exe -v -w -atmid C -atmid CA -atmid N
-atmid O -first $frame -last $frame -pdb $active_dcd $active_psf\n", 'info' );
                        `carma.exe -v -w -atmid C -atmid CA -atmid N -atmid O -first $frame
-last $frame -pdb $active_dcd $active_psf`;
                    }
                }
                else {
                    if ( $include_segid or $atm_id_flag ) {
                        $text -> insert( 'end', "carma.exe -v -w $atm_id_flag -first $frame
-last $frame -pdb $active_dcd $active_psf\n", 'info' );
                        `carma.exe -v -w $atm_id_flag -first $frame -last $frame -pdb
$active_dcd $active_psf`;
                    }
                    elsif ( $custom_id_flag ) {
                        $text -> insert( 'end', "carma.exe -v -w $custom_id_flag -first $frame
-last $frame -pdb $active_dcd $active_psf\n", 'info' );
                        `carma.exe -v -w $custom_id_flag -first $frame -last $frame -pdb
$active_dcd $active_psf`;
                    }
                    else {
                        $text -> insert( 'end', "carma.exe -v -w -atmid CA -first $frame -last
$frame -pdb $active_dcd $active_psf\n", 'info' );
```

```
                            `carma.exe -v -w -atmid CA -first $frame -last $frame -pdb $active_dcd
$active_psf`;
                        }
                    }
                }

            $text -> see( 'end', );

            $frame = sprintf ( "%.7d", $frame, );
             mv  (  "$input.fitted.cluster_$i.dcd.$frame.pdb",  "$input.representative.cluster_$i.pdb"
);
        }
        else {
            $text -> insert( 'end', "\nSomething went wrong. For details check last_carma_run.log
located in :\n", 'error', );
            $text -> insert( 'end', getcwd . "\n", 'info', );
        }
    }
    $active_dcd = $remember_dcd;
    $active_psf = $remember_psf;
}

#############################################################################################################
###   Draw the window for covariance, average and representative structure calculation          ###
#############################################################################################################

sub cov_avg_rep_window {
    my $avg_dot = '';
    my $avg_norm = '';
    my $avg_mass = '';
    my $avg_first = 1;
    my $avg_last = dcd_header_parser( "avg" );
    my $avg_step = 1;
    my $avg_reverse = '';
    my $avg_first_flag = '';
    my $avg_last_flag = '';
    my $avg_step_flag = '';
    my $top_avg;

    if ( !Exists ( $top_avg ) ) {
        $top_avg = $mw -> Toplevel( -title => 'Average and representative structures', );
        $top_avg -> geometry("$toplevel_position");
        $top_avg -> protocol( 'WM_DELETE_WINDOW' => sub { $top_avg -> withdraw }, );

        my $frame_avg1 = $top_avg -> Frame() -> pack( -expand => 1, -fill => 'x', );
        my $frame_avg2 = $top_avg -> Frame() -> pack( -expand => 1, -fill => 'x', );
        my $frame_avg3 = $top_avg -> Frame() -> pack( -expand => 1, -fill => 'x', );

        radiobuttons ( $frame_avg1 );
        checkbuttons ( $frame_avg2 );
        otherbuttons ( $frame_avg3 );

        my $frame_avg4 = $top_avg -> Frame() -> pack( -fill => 'x', );
         my  $frame_avg5  =  $top_avg  ->  Frame(  -borderwidth  =>  2,  -relief  =>  'groove',  )->  pack(
-expand => 0, );
        $frame_avg4 -> Label( -text => "\nVarious Options", -font => $font_20, ) -> pack( -side =>
'top', );

        my $avg_dot_b = $frame_avg5 -> Checkbutton( -text => 'Use dot product (needed for average
structures)',
                                    -variable => \$avg_dot,
                                    -offvalue => '',
                                    -onvalue => " -dot", )
                                    -> grid( -row => 0, -column => 0, -sticky => 'w', );#pack( -side
=> 'top', -anchor => 'w', );
        $avg_dot_b -> select;
        $frame_avg5 -> Checkbutton( -text => 'Calculate normalised matrices',
                                    -variable => \$avg_norm,
                                    -offvalue => '',
                                    -onvalue => " -norm", )
                                    -> grid( -row => 1, -column => 0, -sticky => 'w', );#pack( -side
=> 'top', -anchor => 'w', );
        $frame_avg5 -> Checkbutton( -text => 'Calculate mass-weighted matrices',
                                    -variable => \$avg_mass,
```

```
                                -offvalue => '',
                                -onvalue => " -mass", )
                                -> grid( -row => 2, -column => 0, -sticky => 'w', );#pack( -side
=> 'top', -anchor => 'w', );

        $frame_avg5 -> Label( -text => 'First frame to use: ', )
                               -> grid( -row => 3, -column => 0, -sticky => 'w', );
        $frame_avg5 -> Entry( -textvariable => \$avg_first, )
                               -> grid( -row => 3, -column => 1, );
        $frame_avg5 -> Label( -text => 'Last frame to use: ', )
                               -> grid( -row => 4, -column => 0, -sticky => 'w', );
        $frame_avg5 -> Entry( -textvariable => \$avg_last, )
                               -> grid( -row => 4, -column => 1, );
        $frame_avg5 -> Label( -text => 'Stride (step) between frames: ', )
                               -> grid( -row => 5, -column => 0, -sticky => 'w', );
        $frame_avg5 -> Entry( -textvariable => \$avg_step, )
                               -> grid( -row => 5, -column => 1, );

        my $frame_avg6 = $top_avg -> Frame() -> pack( -expand => 0, );

        $frame_avg6 -> Button( -text => 'Return',
                               -command => [ $top_avg => 'withdraw' ], )
                               -> pack( -side => 'left', );

        $frame_avg6 -> Button( -text => 'Run',
                               -command => sub {
            $top_avg -> destroy;

            $avg_first_flag = ( ( $avg_first != 1 ) ? " -first $avg_first" : '' );
            $avg_last_flag = ( ( $avg_last != $header ) ? " -last $avg_last" : '' );
            $avg_step_flag = ( ( $avg_step != 1 ) ? " -step $avg_step" : '' );

            $seg_id_flag = '' if $seg_id_flag;

            foreach ( @seg_ids ) {
                if ( defined ( $  ) ) {
                    $seg_id_flag = $seg_id_flag . $_;
                }
            }

            create_dir();

              $text   ->   insert(   'end',   "\n\nNow   calculating   average   and   representative
structures.\n\n", 'cyan', );
            $text -> see( 'end', );
            $mw -> update;

            if ( $res_id_flag ) {
                $flag = " -v -w -col -cov $avg_dot $avg_norm $avg_mass $avg_step_flag $avg_last_flag
$avg first flag $avg_reverse $atm_id_flag $res_id_flag -super";
            }
            elsif ( $seg_id_flag ) {
                $flag = " -v -w -col -cov $avg_dot $avg_norm $avg_mass $avg_step_flag $avg_last_flag
$avg_first_flag $avg_reverse $atm_id_flag $seg_id_flag -super";
            }
            else {
                $flag = " -v -w -col -cov $avg_dot $avg_norm $avg_mass $avg_step_flag $avg_last_flag
$avg_first_flag $avg_reverse $atm_id_flag -super";
            }

            carma();

            if ( $all_done ) {
                open CARMA_OUT, '<', 'last_carma_run.log' or die "Cannot open last_carma_run.log for
reading : $!\n";

                while ( <CARMA_OUT> ) {
                    if ( /(Maximum of variance-covariance matrix is [+|-]?)(\S+)/ ) {
                        chomp;
                         $text -> insert( 'end', sprintf "\n%s %f (dark red)\n", $1, $2) if ( not
$avg_reverse );
                         $text -> insert( 'end', sprintf "\n%s %f (dark blue)\n", $1, $2) if (
$avg_reverse );
                    }
```

```
                    if ( /(Minimum of variance-covariance matrix is [+|-]?)(\S+)/ ) {
                        chomp;
                        $text -> insert( 'end', sprintf "%s%f (dark blue)\n", $1, $2 ) if ( not
$avg_reverse );
                        $text -> insert( 'end', sprintf "%s%f (dark red)\n", $1, $2 ) if (
$avg_reverse );
                    }
                }

                close CARMA_OUT;

                if ( $avg_dot ) {
                    open IN, '<', "carma.rms-average.dat" || die "Cannot open carma.rms-average.dat
for reading: $!";

                    my $smallest = 1000;
                    my $frame;
                    while ( <IN> ) {
                        if ( /^\s+(\d+)\s+(\d+\.\d+).*?$/ ) {
                            if ( $2 < $smallest ) {
                                $smallest = $2;
                                $frame = $1;
                            }
                        }
                    }

                    close IN;

                    if ( $linux || $mac ) {
                        if ( $res_id_flag ) {
                            `carma -v -w -atmid ALLID $res_id_flag -first $frame -last $frame -pdb
$active_dcd $active_psf`;
                        }
                        else {
                            `carma -v -w -atmid ALLID -first $frame -last $frame -pdb $active_dcd
$active_psf`;
                        }
                    }
                    else {
                        if ( $res_id_flag ) {
                            `carma.exe -v -w -atmid ALLID $res_id_flag -first $frame -last $frame
-pdb $active_dcd $active_psf`;
                        }
                        else {
                            `carma.exe -v -w -atmid ALLID -first $frame -last $frame -pdb
$active_dcd $active_psf`;
                        }
                    }

                    $frame = sprintf ( "%.7d", $frame, );

                    if ( $active_dcd =~ /(.+)\.dcd/ ) {
                        mv ( "$active_dcd.$frame.pdb", "representative.$1.pdb" );
                        mv ( "carma.superposition.pdb", "superposition.$1.pdb" );
                        mv ( "carma.average.pdb", "average.$1.pdb" );
                        mv ( "$1.dcd.varcov.ps", "covariance.$1.ps" );
                        mv ( "carma.rms-average.dat", "rms_from_average.$1.dat" );
                    }
                }

                $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n", 'valid' );
                $text -> see( 'end', );
                $image_menu -> configure( -state => 'normal', );
            }
            else {
                $text -> insert( 'end', "\nSomething went wrong. For details check
last_carma_run.log located in :\n", 'error', );
                $text -> insert( 'end', getcwd . "\n", 'info', );
                $text -> see( 'end', );
            }
        }, )
        -> pack( -side => 'right', );
    }
    else {
```

```
        $top_avg -> deiconify;
        $top_avg -> raise;
    }
}

#############################################################################################
###   Calculate secondary structure with stride                                           ###
#############################################################################################

sub stride_window {
    my $top_stride;
    my $stride_first = 1;
    my $stride_first_flag;
    my $stride_last = dcd_header_parser ( 'rmsd' );
    my $stride_last_flag;
    my $stride_step;
    my $stride_step_flag;

    if ( !Exists ( $top_stride ) ) {
        if ( $stride_last <= 30000 ) {
            $stride_step = 1;
        }
        elsif ( $stride_last > 30000 ) {
            $stride_step = int ( ( $stride_last / 30000 ) + 0.5 );
        }

        $top_stride = $mw -> Toplevel( -title => 'Secondary structure calculation', );
        $top_stride -> geometry("$toplevel_position");
        $top_stride -> protocol( 'WM_DELETE_WINDOW' => sub { $top_stride -> withdraw }, );

        my $frame_stride2 = $top_stride -> Frame() -> pack( -fill => 'x', );
        my $frame_stride3 = $top_stride -> Frame() -> pack( -fill => 'x', );
        my $frame_stride5 = $top_stride -> Frame() -> pack( -fill => 'x', );
         our $frame_stride1 = $top_stride -> Frame( -borderwidth => 2, -relief => 'groove', ) ->
pack( -fill => 'x', );

        $frame_stride1 -> Label( -text => 'First frame to use: ', )
                               -> grid( -row => 1, -column => 1, -sticky => 'w', );
        $frame_stride1 -> Entry( -textvariable => \$stride_first, )
                               -> grid( -row => 1, -column => 2, );
        $frame_stride1 -> Label( -text => 'Last frame to use: ', )
                               -> grid( -row => 2, -column => 1, -sticky => 'w', );
        $frame_stride1 -> Entry( -textvariable => \$stride_last, )
                               -> grid( -row => 2, -column => 2, );
        $frame_stride1 -> Label( -text => 'Stride (step) between frames: ', )
                               -> grid( -row => 3, -column => 1, -sticky => 'w', );
        $frame_stride1 -> Entry( -textvariable => \$stride_step, )
                               -> grid( -row => 3, -column => 2, );

        checkbuttons ( $frame stride2 );
        otherbuttons ( $frame_stride3 );

        $frame_stride5 -> Label( -text => "\nVarious options", -font => $font_20, ) -> pack;

        my $frame_stride4 = $top_stride -> Frame() -> pack( -expand => 0, );

        $frame_stride4 -> Button( -text => 'Return',
                                  -command => [ $top_stride => 'withdraw' ], )
                                  -> pack( -side => 'left', );

        $frame_stride4 -> Button( -text => 'Run',
                                  -command => sub {
        $stride_first_flag = ( ( $stride_first != 1 ) ? " -first $stride_first" : '' );
        $stride last flag = ( ( $stride last != $header ) ? " -last $stride last" : '' );
        $stride_step_flag = ( ( $stride_step != 1 ) ? " -step $stride_step" : '' );

        if ( $stride_first != 1 or $stride_step != 1 ) {
            open STRIDE_FIRST_STEP, '>', 'stride_first_step' or die $!;

            printf STRIDE_FIRST_STEP "%8d %8d", $stride_first, $stride_step;

            close STRIDE_FIRST_STEP;
        }
```

```
        $top_stride -> destroy;

        $seg_id_flag = '' if $seg_id_flag;

        foreach ( @seg_ids ) {
            if ( defined ( $_ ) ) {
                $seg_id_flag = $seg_id_flag . $_;
            }
        }

        if ( $res_id_flag ) {
             $flag  =  "  -w  -v  -pdb  -stride  -atmid  HEAVY  $stride_first_flag  $stride_last_flag
$stride_step_flag $res_id_flag $custom_id_flag";
        }
        elsif ( $seg_id_flag ) {
             $flag  =  "  -w  -v  -pdb  -stride  -atmid  HEAVY  $stride_first_flag  $stride_last_flag
$stride_step_flag $seg_id_flag $custom_id_flag";
        }
        else {
             $flag  =  "  -w  -v  -pdb  -stride  -atmid  HEAVY  $stride_first_flag  $stride_last_flag
$stride_step_flag $custom_id_flag";
        }

        create_dir( 'stride' );

        opendir DIR, '.' or die "Cannot open .. for reading : $!\n";
        while ( my $pdb_file = readdir DIR ) {
            mv ( "$pdb_file", "tmp" ) if ( $pdb_file =~ /pdb$/ );
        }

        closedir DIR;

        $text -> insert( 'end', "\n\nNow performing secondary structure analysis using stride.\n\n",
'cyan', );
        $text -> see( 'end', );
        $mw -> update;

        carma();

        if ( $all_done ) {
            my $dir = getcwd;

            opendir TMPDIR, "tmp" or die "Cannot open . for reading : $!\n";
            while ( my $dh = readdir TMPDIR ) {
                $dh = getcwd . "/tmp/$dh";

                mv ( $dh, $dir ) unless ( $dh =~ /\.\./ );
            }

            closedir TMPDIR;
            rmdir ( "tmp" );

            open PSF, '<', 'carma.selected_atoms.psf' or die "Cannot open carma.selected_atoms.psf
for reading : $!\n";

            my @residues;
            my $residues;
            my %residues;
            my $columns = 0;
            my @columns;

            my $i = 0;

            if ( ( $res id flag and not $seg_id_flag ) or ( $res_id_flag and $seg_id_flag ) ) {
                while ( <PSF> ) {
                    if ( /\!NATOM/ ) {
                        last;
                    }
                }

                while ( <PSF> ) {
                    if ( /\d+\s+Z\s+(\d+)\s+\w+\s+\w+/ ) {
                        $residues[$i] = $1;
                    }
```

```
                $i++;
            }
        }
        else {
            while ( <PSF> ) {
                if ( /\!NATOM/ ) {
                    last;
                }
            }

            while ( <PSF> ) {
                if ( /\d+\s+([A-Z])\s+(\d+)\s+\w+\s+\w+/ ) {
                    $residues{$1} = $2;
                }
            }
        }

        close PSF;

        $residues = scalar(uniq(@residues)) if ( @residues );

        if ( $residues or scalar ( keys %residues ) == 1 ) {
            if ( $residues ) {
                $columns = $residues;
            }
            else {
                 foreach ( keys %residues ) {
                    $columns = $residues{$_};
                }
            }

             open STRIDE, '<', "carma.stride.dat" or die "Cannot open carma.stride.dat for
reading : $!\n";
            open OUT, '>', "temp.dat" or die "Cannot open temp.dat for writing : $!\n";
            open OUT1, '>', "stride_plot.dat" or die "Cannot open temp.dat for writing : $!\n";

            my $line;
            my $new_line;
            while ( my $line = <STRIDE> ) {
                if ( $line =~ /No hydrogen bonds/ ) {
                    print OUT ">\n" . 'C' x $columns . "\n";
                    print OUT1 'C' x $columns . "\n";
                }
                else {
                    $new_line = substr( $line, 0, $columns, '', );
                    $new_line =~ s/ /C/g;
                    $new_line =~ s/b/B/g;
                    print OUT ">\n$new_line\n";
                    print OUT1 "$new_line\n";
                }
            }

            close STRIDE;
            close OUT;
            close OUT1;

            if ( $weblogo or $seqlogo ) {
                $text -> insert( 'end', "\nNow running weblogo/seqlogo on carma.stride.dat\n",
'cyan' );
                $text -> see( 'end', );
                $mw -> update;
                sleep 1;

                if ( $weblogo ) {
                    `weblogo -C '#600080' I 'pi helix' -C '#6080FF' B 'b turn' -C '#6080FF' T 'b
turn' -C '#A00080' G '3-10 helix' -C '#FF0080' H 'a-helix' -C 'green' C 'coil' -C '#FFC800' E
'sheet' --composition none -a 'GTCHEBI' < temp.dat > weblogo.sec_structure_graph.eps`;
                }
                elsif ( $seqlogo ) {
                    `seqlogo -Y -C 40 -w 20 -f temp.dat > weblogo.sec_structure_graph.eps`;
                }
            }
```

```
                unlink ( "temp.dat" );
            }
            elsif ( scalar ( keys %residues ) > 1 ) {
                my $j = 0;
                my $offset = 0;
                foreach ( keys %residues ) {
                    open STRIDE, '<', "carma.stride.dat" or die "Cannot open carma.stride.dat for
reading : $!\n";
                    open OUT, '>', "stride_chain$_.dat" or die "Cannot open stride_chain$_.dat for
reading : $!\n";
                    open OUT1, '>', "stride_plot_chain$_.dat" or die "Cannot open stride_chain$_.dat
for reading : $!\n";

                    if ( $j != 0 ) {
                        if ( $columns <= 50 ) {
                            $offset = 50;
                        }
                        elsif ( $columns > 50 and $columns <= 100 ) {
                            $offset = 100;
                        }
                        elsif ( $columns > 100 and $columns <= 150 ) {
                            $offset = 150;
                        }
                        elsif ( $columns > 150 and $columns <= 200 ) {
                            $offset = 200;
                        }

                        $columns = $residues{$_};
                    }
                    else {
                        $columns = $residues{$_};
                    }

                    my $line;
                    my $new_line;
                    while ( $line = <STRIDE> ) {
                        $new_line = substr( $line, $offset, $columns, '', );

                        if ( $new_line =~ /No hydrogen bonds/ ) {
                            print OUT ">\n" . 'C' x $columns . "\n";
                            print OUT1 'C' x $columns . "\n";
                        }
                        else {
                            $new_line =~ s/ /C/g;
                            $new_line =~ s/b/B/g;
                            print OUT ">\n$new_line\n";
                            print OUT1 "$new_line\n";
                        }
                    }

                    close OUT;
                    close OUT1;
                    close STRIDE;

                    if ( $weblogo or $seqlogo ) {
                          $text    ->    insert(    'end',    "\nNow    running    weblogo/seqlogo    on
carma.stride.dat\n", 'cyan' ) if ( $j == 0 );
                        $text -> see( 'end', );
                        $mw -> update;
                        sleep 1;

                        if ( $weblogo ) {
                            `weblogo -C '#600080' I 'pi helix' -C '#6080FF' B 'b turn' -C '#6080FF'
T 'b turn' -C '#A00080' G '3-10 helix' -C '#FF0080' H 'a-helix' -C 'green' C 'coil' -C '#FFC800' E
'sheet'      --composition      none      -a      'GTCHEBI'      <      stride_chain$_.dat      >
weblogo.sec_structure_graph_chain$_.eps`;
                        }
                        elsif ( $seqlogo ) {
                              `seqlogo    -Y    -C    40    -w    20    -f    stride_chain$_.dat    >
weblogo.sec_structure_graph_chain$_.eps`;
                        }
                    }

                    unlink ( "stride_chain$_.dat" );
```

```
                $j++;
            }
        }

        if ( $active_dcd =~ /(.+)\.dcd/ ) {
            my $cur_dcd_name = $1;

            mv ( "carma.stride.dat", "stride_text.$1.dat" );

            opendir CWD, getcwd or die $!;
            while ( my $file = readdir CWD ) {
                if ( $file =~ /stride_plot.dat/ ) {
                    mv ( "$file", "stride_plot.$cur_dcd_name.dat" );
                }
                elsif ( $file =~ /stride_plot_(.+).dat/ ) {
                    mv ( $file, "stride_plot_$1.$cur_dcd_name.dat" );
                }

                if ( $file =~ /weblogo.sec_structure_graph.eps/ ) {
                    mv ( $file, "weblogo_graph.$cur_dcd_name.eps" );
                }
                elsif ( $file =~ /weblogo.sec_structure_graph_(.+).eps/ ) {
                    mv ( $file, "weblogo_graph_$1.$cur_dcd_name.eps" );
                }
            }

            closedir CWD;
        }

        $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n", 'valid' );
        $text -> see( 'end', );

        $image_menu -> configure( -state => 'normal', );
    }
    else {
        $text -> insert( 'end', "\nSomething went wrong. For details check last_carma_run.log
located in :\n", 'error', );
        $text -> insert( 'end', getcwd . "\n", 'info', );
        $text -> see( 'end', );
    }
    }, )-> pack( -side => 'right', );
}
else {
    $top_stride -> deiconify;
    $top_stride -> raise;
}
}

##########################################################################################################
###   Draw the window for viewing the .ps images                                                       ###
##########################################################################################################

sub image_window {
    our $prev_psf;
    our @other;

    my $i = 0;
    my $j = 0;
    my ( @pdb, @dat, @ps, );

    my $files =
        'Qfraction.*.dat|' .
        'Rgyration.*dat|' .
        'surface.*dat|' .
        'distances.*|' .
        '.PCA.eigenvalues.*|' .
        '.*rms_from_*.*.dat|' .
        'torsions.*|' .
        'phi_psi_dihedral_segid.*.dat|' .
        'bendangles.*|' .
        '.PCA.rms_from_*.*.dat|' .
        'entropy.*dat|' .
        '.*.clusters.dat|' .
```

```
        'entropy_andricioaei.*|' .
        'entropy_schlitter.*|' .
        '.*.clusters_vs_variance_explained.dat|' .
        '.*.clusters_vs_rms_cutoff.dat|' .
        'stride.*dat';

    if ( $vmd ) {
        $files .= '|.*.cns';
    }

    our $image_top = $mw -> Toplevel( -title => 'Latest Results', );
    $image_top -> geometry("$toplevel_position");
    $image_top -> resizable( 0, 0 );

    opendir IMAGE_DIR, getcwd || die "Cannot open " . getcwd . ": $!";
    while ( my $dh = readdir IMAGE_DIR ) {
        if ( $dh =~ /.*.ps$/ ) {
            push @ps, $dh;
        }
        elsif ( $dh =~ /$files/ ) {
            push @dat, $dh;
        }
        elsif ( $dh =~ /.*\.pdb$/ ) {
            push @pdb, $dh;
        }
    }
    closedir IMAGE_DIR;

    @pdb = sort ( @pdb );
    @dat = sort ( @dat );
    @ps = sort ( @ps );

    my $dir = getcwd;

    my $frame_image1 = $image_top -> Frame() -> grid( -row => 5, -column => 1, );
    my $frame_image2 = $image_top -> Frame() -> grid( -row => 5, -column => 3, );
    my $frame_image3 = $image_top -> Frame() -> grid( -row => 5, -column => 2, );
    my $frame_image4 = $image_top -> Frame() -> grid( -row => 6, -column => 1, );
    my $frame_image5 = $image_top -> Frame() -> grid( -row => 7, -column => 1, -columnspan => 4, );
    my $frame_image6 = $image_top -> Frame() -> grid( -row => 8, -column => 1, -columnspan => 4, );

    $image_top -> Label( -text => "Displaying contens of the folder", ) -> grid( -row => 1, -column
=> 2, );
    $image_top -> Entry( -text => $dir, -width => 0, ) -> grid( -row => 2, -column => 2, );
    $image_top -> Label( -text => "\nClick on the image you want to view", ) -> grid( -row => 3,
-column => 2, );
    $image_top -> Label( -text => "or the file you would like to plot\n", ) -> grid( -row => 4,
-column => 2, );

    $frame_image1 -> Label( -text => 'Available pdb files', ) -> pack unless ( ( $linux or $mac )
and not $pdb_viewer );
    $frame_image2 -> Label( -text => 'Available numerical files', ) -> pack;
    $frame_image3 -> Label( -text => 'Available postscript files', ) -> pack unless ( ( $linux or
$mac ) and not $ps_viewer );

    my $lb1 = $frame_image1 -> Scrolled( "Listbox", -scrollbars => 'oe', -selectmode => "single",
-width => 35, -height => 15, ) -> pack unless ( ( $linux or $mac ) and not $pdb_viewer );
    my $lb2 = $frame_image2 -> Scrolled( "Listbox", -scrollbars => 'oe', -selectmode => "single",
-width => 43, -height => 15, ) -> pack;
    my $lb3 = $frame_image3 -> Scrolled( "Listbox", -scrollbars => 'oe', -selectmode => "single",
-width => 40, -height => 15, ) -> pack unless ( ( $linux or $mac ) and not $ps_viewer );

    my $vmd_check;
    my $vmd_check_var;
    if ( $vmd ) {
        $vmd_check = $frame_image4 -> Checkbutton( -text => 'Use VMD to view .pdb files',
                                                   -variable => \$vmd_check_var, )
                                                   -> pack( -side => 'left', );
    }

    if ( $terminal and $pdb_viewer ne 'vmd' ) {
        $frame_image5 -> Button( -text => 'Log of the last carma run',
                                 -width => 28,
                                 -command => sub {
```

```
            if ( $terminal eq 'xterm' ) {
                system ( "$terminal -fg white -bg black -geometry 80x25+800+200 -e \"sleep 30 | cat
last_carma_run.log\"" );
            }
            elsif ( $terminal eq 'rxvt' ) {
                system ( "$terminal -fg white -bg black -geometry 80x25+800+200 -e sh -c \"sleep 30
| cat last_carma_run.log\"" );
            }
            elsif ( $terminal eq 'gnome-terminal' ) {
                 system  (  "$terminal  --geometry  80x25+800+200  -x  sh  -c  'sleep  30  |  cat
last_carma_run.log'" );
            }
        }, ) -> pack( -side => 'left', );
    }

    $frame_image5 -> Button( -text => 'Change active DCD',
                             -width => 28,
                             -command => sub {
        my @dcd_files_in_the_folder;
        my @candidate_dcd;
        opendir CUR_DIR, '.' or die $!;

        while ( my $new_file = readdir CUR_DIR ) {
            if ( $new_file =~ /(.+)\.dcd$/ ) {
                push ( @dcd_files_in_the_folder, $1 );
            }
        }

        closedir CUR_DIR;

        foreach ( @dcd_files_in_the_folder ) {
            if ( -f "$_.psf" ) {
                push ( @candidate_dcd, $_ );
            }
        }

        my $change_dcd_toplevel = $mw -> Toplevel();

        $change_dcd_toplevel -> Label( -text => "Select the DCD file that will become the new active
DCD\n" .
                                               " It will automatically be paired with the correct
PSF.", ) -> pack;

        my $listbox = $change_dcd_toplevel -> Listbox( -width => 45, ) -> pack;

        $listbox -> insert( 'end', sort @candidate_dcd );

        $listbox -> bind( '<Button-1>', sub {
            my $selection = $listbox -> get( $listbox -> curselection() );

            $other[0] -> invoke if ( Exists( $other[0] ) );

            $active_psf = $selection . '.psf';
            $active_dcd = $selection . '.dcd';

            $prev_psf = $active_psf;

            $active_psf_label -> configure( -text => "$active_psf ", );
            $active_dcd_label -> configure( -text => "$active_dcd", );

            $go_back_button -> configure( -state => 'normal', );

            $change_dcd_toplevel -> destroy;
            $image_top -> destroy;

            ( $atm_id_flag, $res_id_flag, $custom_id_flag, $count, ) = ( '', '', '', 0, );
            undef @unique_chain_ids;
            undef @seg_ids;
            undef %num_residues;

            my $x = $mw -> width;
            my $y = $mw -> height;

            $f0 -> packForget;
```

```
            parser ( $active_psf, $active_dcd );

            $f0 -> pack( -side => 'top', -fill => 'both', -expand => 1, );
            $mw -> geometry( "$x" . 'x' . "$y" );
            $mw -> update;
        } );
    }, ) -> pack( -side => 'left', );

    $frame_image5 -> Button( -text => 'Back up produced files',
                             -width => 28,
                             -command => sub {
        my $i = 0;
        my $dir_name = "grcarma_backup_$i";
        my $current_dir = getcwd;
        my $backup_toplevel = $mw -> Toplevel();

        $backup_toplevel -> Label( -text => 'Select a name for the subdirectory that will contain
all results up to now', -font => $font_12, ) -> pack;
        $backup_toplevel -> Label( -text => 'It will be created in the directory the program was
launched from.', -font => $font_12, ) -> pack;
        $backup_toplevel -> Label( -text => 'After the process is completed the active PSF-DCD files
will be the ones that were originally specified', -font => $font_12, ) -> pack;
        $backup_toplevel -> Entry( -textvariable => \$dir_name,
                                   -width => 20, ) -> pack;
        $backup_toplevel -> Button( -text => 'Confirm',
                                    -command => sub {
            if ( not $dir_name ) {
                if ( $linux or $mac ) {
                    $backup_toplevel -> messageBox( -message => 'Please specify a name for the back
up directory', -font => $font_12, );
                }
                else {
                    $backup_toplevel -> messageBox( -message => 'Please specify a name for the back
up directory', );
                }
            }
            else {
                $backup_toplevel -> destroy;

                $dir_name = "$launch_dir/$dir_name";
                mkpath ( "$dir_name" );

                opendir OLD_DIR, $current_dir or die $!;

                $active_psf = $psf_name . '.psf';
                $active_dcd = $dcd_name . '.dcd';

                while ( my $file_to_copy = readdir OLD_DIR ) {
                    unless ( $file_to_copy =~ /^$active_psf$|^$active_dcd$/ ) {
                        mv ( "$file_to_copy", "$dir_name" );
                    }
                }

                closedir OLD_DIR;

                $image_top -> destroy;
                $image_menu -> configure( -state => 'disabled', );

                ( $atm_id_flag, $res_id_flag, $custom_id_flag, $count, ) = ( '', '', '', 0, );
                undef @unique_chain_ids;
                undef @seg_ids;
                undef %num_residues;

                my $x = $mw -> width;
                my $y = $mw -> height;

                $f0 -> packForget;
                parser ( $active_psf, $active_dcd );

                $f0 -> pack( -side => 'top', -fill => 'both', -expand => 1, );
                $mw -> geometry( "$x" . 'x' . "$y" );
                $mw -> update;

                $active_psf_label -> configure( -text => "$active_psf ", );
```

```
                $active_dcd_label -> configure( -text => "$active_dcd", );

                $i++;
            }
        }, ) -> pack;
    }, ) -> pack( -side => 'left', );
    $frame_image5 -> Button( -text => 'Empty the current working directory',
                             -width => 28,
                             -command => sub {
        my $response;
        if ( $linux or $mac ) {
            $response = $frame_image5 -> messageBox( -message => "Are you sure? All non psf/dcd
files will be permanently deleted.",
                                                     -type => 'yesno',
                                                     -icon => 'question',
                                                     -font => "$font_12", );
        }
        else {
            $response = $frame_image5 -> messageBox( -message => "Are you sure? All non psf/dcd
files will be permanently deleted.",
                                                     -type => 'yesno',
                                                     -icon => 'question', );
        }

        if ( $response =~ /yes/i ) {
            $image_top -> destroy;
            $image_menu -> configure( -state => 'disabled', );

            opendir CWD, getcwd or die "Cannot open cwd: $!";
            while ( my $cwd = readdir CWD ) {
                unless ( $cwd =~ /psf$|dcd$/ ) {
                    unlink $cwd;
                }
            }

            closedir CWD;
        }
    }, ) -> pack( -side => 'left', );

    $frame_image6 -> Button( -text => 'Return',
                             -command => [ $image_top => 'withdraw' ],
                             -width => 28, )
                             -> pack;

    $lb1 -> insert( 'end', @pdb, ) unless ( ( $linux or $mac ) and not $pdb_viewer );
    $lb2 -> insert( 'end', @dat, );
    $lb3 -> insert( 'end', @ps, ) unless ( ( $linux or $mac ) and not $ps_viewer );

    unless ( ( $linux or $mac ) and not $pdb_viewer ) {
        $lb1 -> bind( '<Button-1>', sub {
            my $selection = $lb1 -> get( $lb1 -> curselection() );

            if ( $vmd_check_var ) {
                system ( "vmd $selection" ) if ( $linux || $mac );
            }
            else {
                system ( "$pdb_viewer $selection &" ) if ( ( $linux || $mac ) && $pdb_viewer );
                `start $selection` if ( $windows );
            }
        } );
    }
    $lb2 -> bind( '<Button-1>', sub {
        my $selection = $lb2 -> get( $lb2 -> curselection() );

        if ( $selection =~ /cns$/ ) {
            system ( "vmd $selection" );
        }
        elsif ( $selection =~ /stride_plot/ ) {
            {               my $coords_offset = 0;
                my $coords_step = 1;
                if ( -f 'stride_first_step' ) {
                    open TMP_FILE, '<', 'stride_first_step' or die $!;

                    while ( <TMP_FILE> ) {
```

```
            if ( /\s+(\d+)\s+(\d+)/ ) {
                $coords_offset = $1 if ( $1 != 1 );
                $coords_step = $2 if ( $2 != 1 );
            }
        }

        #~ unlink ( 'stride_first_step' );
        close TMP_FILE;
    }

    local $/ = \1;

    my $temp_w = MainWindow -> new( -title => 'Secondary structure plot', );
    $temp_w -> geometry( '1018x670' );
    $temp_w -> resizable( 0, 0, );
    $temp_w -> configure( -menu => my $menubar = $temp_w -> Menu );

    my $file = $menubar -> cascade( -label => '~File' );

    my $nofres = length ( `head -1 $selection` ) - 1;
    my $frames;

    if ( `wc -l $selection` =~ /(\d+)/ ) {
        $frames = $1;
    }

    my $height = 600;
    my $width = 1010;
    my $color = '';

    my $y_step = $height / $nofres;
    my $x_step = $width / $frames;

    our $canvas = $temp_w -> Canvas( -cursor=>"crosshair",
                                     -height => $height,
                                     -width => $width+8,
                                     -bg => 'white', ) -> pack( -anchor => 'n', );

    $canvas -> CanvasBind("<$_>", sub {
        $canvas -> yviewMoveto( 0 )
    } ) for ( 4, 5, );

    my $temp_frame1 = $temp_w -> Frame() -> pack( -side => 'left', );
    my $temp_frame2 = $temp_w -> Frame() -> pack( -side => 'right', );

     my $coord_label = $temp_frame1 -> Label( -text => 'Coordinates ( residue,
frame ) :', ) -> pack( -side => 'left', );

    my $legend_canvas = $temp_frame2 -> Canvas( -height => 60, -width => 400, ) -> pack(
-side => 'right', );

    $legend_canvas -> createRectangle( 3, 3, 17, 17, -fill => '#FF0080', );
    $legend_canvas -> createText( 68, 10, -text => 'A Helix', );
    $legend_canvas -> createRectangle( 3, 23, 17, 37, -fill => '#A00080', );
    $legend_canvas -> createText( 68, 30, -text => '3-10 Helix', );
    $legend_canvas -> createRectangle( 3, 43, 17, 57, -fill => '#600080', );
    $legend_canvas -> createText( 68, 50, -text => 'Pi Helix', );

    $legend_canvas -> createRectangle( 203, 3, 217, 17, -fill => '#FFC800', );
    $legend_canvas -> createText( 268, 10, -text => 'B Sheet', );
    $legend_canvas -> createRectangle( 203, 23, 217, 37, -fill => '#6080FF', );
    $legend_canvas -> createText( 268, 30, -text => 'B/G Turn', );
    $legend_canvas -> createRectangle( 203, 43, 217, 57, -fill => '#FFFFFF', );
    $legend_canvas -> createText( 268, 50, -text => 'Coil/Unassigned', );

    #~ $canvas -> bind("<Button-1>", [ \&print_xy, Ev('x'), Ev('y'), $x_step, $y_step,
$nofres, $coordinates, $coords_offset, $coords_step, ] );
     $canvas -> CanvasBind("<Button-1>", [ \&print_xy, Ev('x'), Ev('y'), $x_step,
$y_step, $nofres, $coord_label, $coords_offset, $coords_step, ] );

    sub print_xy {
         my ($canv, $x, $y, $x_step, $y_step, $nofres, $coord_label, $coords_offset,
$coords_step, ) = @_;
```

```
                    my $coords = sprintf "(%4d, %7d)", ($nofres-($canv->canvasy($y)/$y_step))+1,
$coords_offset+1+($coords_step*sprintf"%7d",(($canv->canvasx($x)-10)/$x_step));
                    $coord_label -> configure( -text => "Coordinates ( residue, frame ) : $coords",
);
                }

                $temp_w -> update;
                my $currentSize = $temp_w -> reqwidth . "x" . $temp_w -> reqheight;
                my $newsize;

                $temp_w -> update;

                $file -> command(
                    -label       => 'Save As Postscript',
                    -accelerator => 'Ctrl-s',
                    -underline   => 0,
                    -command     => sub {
                        $canvas -> update;
                        $canvas -> postscript( -file => "$selection.eps", -colormode => 'color', );
                    }
                );
                $file->separator;
                $file->command(
                    -label       => "Close",
                    -accelerator => 'Ctrl-q',
                    -underline   => 0,
                    -command     => sub { $temp_w -> destroy; },
                );

                $temp_w -> bind( $temp_w, "<Control-s>" => sub {
                        $canvas -> update;
                        $canvas -> postscript( -file => "$selection.eps", -colormode => 'color', );
                    }
                );

                $temp_w -> bind( $temp_w, "<Control-q>" => sub { $temp_w -> destroy; }
                );

                open FILE, '<', $selection or die "Cannot open $selection for reading : $!\n";

                my ( $x, $y, ) = ( 0, 0, );
                while ( <FILE> ) {
                    if ( $_ eq 'H' ) {
                        $color = '#FF0080';
                    }
                    elsif ( $_ eq 'G' ) {
                        $color = '#A00080';
                    }
                    elsif ( $_ eq 'I' ) {
                        $color = '#600080';
                    }
                    elsif ( $_ eq 'E' ) {
                        $color = '#FFC800';
                    }
                    elsif ( $_ eq 'B' or $_ eq 'T' ) {
                        $color = '#6080FF';
                    }
                    elsif ( $_ eq 'C' ) {
                        $color = '#FFFFFF';
                    }

                    $canvas -> createRectangle( ( $x * $x_step ) + 10, ( $height - ( $y_step * ( $y
+ 1 ) ) ), ( ( $x + 1 ) * $x_step ) + 10, ( $height - ( $y_step * $y ) ), -fill => "$color",
-outline => undef, ) unless ( $color eq '#FFFFFF' );

                    if ( $y == $nofres ) {
                        $y = 0;
                        $x++;
                    }
                    else {
                        $y++;
                    }
                }
```

```
                close FILE;

                my $counter = 1;
                for ( my $temp_res = $y_step ; $temp_res < $height ; $temp_res += $y_step ) {
                    if ( $counter % 10 ) {
                        $canvas -> createLine( 4, $height-$temp_res, 8, $height-$temp_res, -fill =>
'black', );
                    }
                    else {
                        $canvas -> createLine( 0, $height-$temp_res, 8, $height-$temp_res, -fill =>
'black', -width => 2, );
                    }

                    $counter++;
                }

                $canvas -> createLine( 8, 600, 8, 0, -fill => 'black', -width => 2, );
                $canvas -> createLine( 0, 600, 1018, 600, -fill => 'black', -width => 2, );
            }
        }
        elsif ( $selection =~ /stride.*.dat/ ) {
            my $temp_w = MainWindow -> new( -title => 'carma.stride.dat', );
            my $temp_t = $temp_w -> Scrolled( 'ROText', -wrap => 'none', ) -> pack;

            open STRIDE, '<', $selection or die "Cannor open $selection for reading : $!\n";

            while ( <STRIDE> ) {
                $temp_t -> insert( 'end', "$_" );
            }
            close STRIDE;
        }
        elsif ( $selection =~ /phi_psi/ ) {
            my $response = 'yes' if ( $ramachandran_popup > 0 );

            our ( $first_residue, $last_residue, $number_of_residues, $correction, $temp_segid );

            if ( $ramachandran_popup == 0 ) {
                if ( $linux or $mac ) {
                    $response = $mw -> messageBox(

                    -message => "The file $selection contains the phi/psi angles" .
                                " for residues $first_residue to $last_residue of chain $temp_segid"
.
                                ". Would you like to view these results in a Ramachandran plot ?",
                    -icon => 'question',
                    -type => 'yesno',
                    -font => "$font_12", );
                }
                else {
                    $response = $mw -> messageBox(

                    -message => "The file $selection contains the phi/psi angles" .
                                " for residues $first_residue to $last_residue of chain $temp_segid"
.
                                ". Would you like to view these results in a Ramachandran plot ?",
                    -icon => 'question',
                    -type => 'yesno', );
                }

                $ramachandran_popup++;
            }

            if ( $response =~ /yes/i ) {
                $response = 'yes';

                my $entry_var = "$first_residue-$last_residue;";
                my $top = $mw -> Toplevel( -title => 'phi/psi dihedral angles' );
                 $top -> Label( -text => "$number_of_residues pairs of phi/psi angles detected. In
case you would\n" .
                                        "like to plot only a subset or subsets of those residues,\n"
.
                                        "please define them using the textbox below", ) -> pack;

                $top -> Entry( -textvariable => \$entry_var, ) -> pack;
```

```
                my $semicolon_count = 0;
                my @range_low;
                my @range_high;
                $top -> Button( -text => 'Plot',
                                -command => sub {
                    $top -> destroy;

                    if ( $entry_var =~ /;/ ) {
                        $semicolon_count = $entry_var =~ tr/;//;
                    }

                    $entry_var =~ s/;/\n/g;

                    open FH, '<', \$entry_var or die "Cannot open $entry_var for reading : $!\n";

                    my $j = 0;
                    while ( <FH> ) {
                        if ( /(\d+).?(\d*)/ ) {
                            $range_low[$j] = $1;
                            $range_high[$j] = $2 if $2;
                        }

                        $j++;
                    }

                    close FH;

                    open TORSIONS, '<', "phi_psi_dihedral_segid$temp_segid.dat" or die "Cannot open
phi_psi_dihedral_segid$temp_segid.dat for reading : $!\n";

                    my ( @phi_angles, @psi_angles, );

                    for ( my $k = 0 ; $k < scalar ( @range_low ) ; $k++ ) {
                        if ( $range_high[$k] ) {
                            for ( my $index = $range_low[$k] ; $index <= $range_high[$k] ; $index++
) {#print DEBUG $index;

                                my $other_index = 0;

                                while ( <TORSIONS> ) {
                                    $phi_angles[$other_index] = substr $_, 10 + ( ( $index - ( 2 +
$correction ) ) * 20 ), 9;
                                    $psi_angles[$other_index] = substr $_, 20 + ( ( $index - ( 2 +
$correction ) ) * 20 ), 9;
                                    $other_index++;
                                }

                                seek TORSIONS, 0, 0;
                                $index = sprintf ( "%03d", $index );

                                open OUT, '>>', "phi_psi_temp_angles" or die $!;

                                while ( my $phi_line = shift @phi_angles, my $psi_line = shift
@psi_angles, ) {
                                    chomp ( $phi_line, $psi_line, );

                                    print OUT "$phi_line $psi_line\n";
                                }

                                close OUT;
                            }
                        }
                        else {
                            my $index = $range_low[$k];
                            my $other_index = 0;

                            while ( <TORSIONS> ) {
                                $phi_angles[$other_index] = substr $_, 10 + ( ( $index - ( 2 +
$correction ) ) * 20 ), 9;
                                $psi_angles[$other_index] = substr $_, 20 + ( ( $index - ( 2 +
$correction ) ) * 20 ), 9;
                                $other_index++;
                            }
```

```
                            seek TORSIONS, 0, 0;
                            $index = sprintf ( "%03d", $index );

                            open OUT, '>>', "phi_psi_temp_angles" or die $!;

                             while  ( my  $phi_line  =  shift  @phi_angles,  my  $psi_line  =  shift
@psi_angles, ) {
                                chomp ( $phi_line, $psi_line, );

                                print OUT "$phi_line $psi_line\n";
                            }

                            close OUT;
                        }
                    }

                    close TORSIONS;

                    scatter_plot ( 'ramachandran' );
                }, ) -> pack;
            }
        }
        elsif ( $selection =~ /clusters.dat/ ) {
            my $noframes;

            if ( -f 'pca_first_step' ) {
                open PCA_FIRST_STEP, '<', 'pca_first_step' or die $!;

                while ( <PCA_FIRST_STEP> ) {
                    if ( /(\d+)/ ) {
                        $noframes = $1;
                    }
                }

                close PCA_FIRST_STEP;
            }

            open CLUSTERS, '<', $selection or die "Cannot open $selection for reading : $!\n";

            my @frames;
            my @clusters;
            my @frames_clusters;
            my @sorted_frames_clusters;

            while ( <CLUSTERS> ) {
                if ( /^\s*(\d+)\s+(\d+)/ ) {
                    push @frames_clusters, sprintf ( "%8d %3d", $1, $2 );
                    push @clusters, $2;
                    push @frames, $1;
                }
            }

            close CLUSTERS;

            my $nofclusters = uniq(@clusters);

            my $height = ( $nofclusters * 20 ) + 2;
            my $width = 770;

            my @colors = (
                            'blue', 'red', 'green', 'magenta', 'cyan', 'orange', 'purple', 'brown',
'maroon', 'orangered',
                            'blue2', 'red2', 'green2', 'magenta2', 'cyan2', 'orange2', 'purple2',
'brown2', 'maroon2', 'orangered2',
                            'blue3', 'red3', 'green3', 'magenta3', 'cyan3', 'orange3', 'purple3',
'brown3', 'maroon3', 'orangered3',
                            'blue4', 'red4', 'green4', 'magenta4', 'cyan4', 'orange4', 'purple4',
'brown4', 'maroon4', 'orangered4',
                            'blue', 'red', 'green', 'magenta', 'cyan', 'orange', 'purple', 'brown',
'maroon', 'orangered',
                            'blue2', 'red2', 'green2', 'magenta2', 'cyan2', 'orange2', 'purple2',
'brown2', 'maroon2', 'orangered2',
                            'blue3', 'red3', 'green3', 'magenta3', 'cyan3', 'orange3', 'purple3',
```

```
'brown3', 'maroon3', 'orangered3',
                        'blue4', 'red4', 'green4', 'magenta4', 'cyan4', 'orange4', 'purple4',
'brown4', 'maroon4', 'orangered4',
            );

            my $temp_w = MainWindow -> new( -title => 'Cluster plot', );
            $temp_w -> geometry( $width . 'x' . ( $height + 30 ) );
            $temp_w -> resizable( 0, 0, );
            $temp_w -> configure( -menu => my $menubar = $temp_w -> Menu );

            my $file = $menubar -> cascade( -label => '~File' );

            my $canvas_frame = $temp_w -> Frame() -> pack;
            my $coords_frame = $temp_w -> Frame() -> pack;

            my $canvas = $canvas_frame -> Canvas( -cursor=>"crosshair",
                                        -height => $height,
                                        -width => $width,
                                        -bg => 'white', ) -> pack( -anchor => 'n', );

            my $coord_label = $coords_frame -> Label( -text => 'Coordinates ( Cluster, frame ) :', )
-> pack( -side => 'left', );

              $canvas   ->   CanvasBind("<Button-1>",   [   \&print_xy_clusters,   Ev('x'),   Ev('y'),
$nofclusters, $noframes, $width, $coord_label, $height, ] );

            sub print_xy_clusters {
                my ($canv, $x, $y, $nofclusters, $noframes, $width, $coord_label, $height, ) = @_;

                if ( $x >= 58 ) {
                    my $coords = sprintf "(%2d, %7d)", ( ( $height - $y ) / 20 ) + 1, ( ( $noframes
* ( $x - 58 ) ) / ( $width - 58 ) );
                    $coord_label -> configure( -text => "Coordinates ( Cluster, frame ) : $coords",
);
                }
            }

            $file -> command(
                -label       => 'Save As Postscript',
                -accelerator => 'Ctrl-s',
                -underline   => 0,
                -command     => sub {
                    $canvas -> update;
                    $canvas -> postscript( -file => "$selection.eps", -colormode => 'color', );
                }
            );
            $file->separator;
            $file->command(
                -label       => "Close",
                -accelerator => 'Ctrl-q',
                -underline   => 0,
                -command     => sub { $temp_w -> destroy; },
            );

            $temp_w -> bind( $temp_w, "<Control-s>" => sub {
                    $canvas -> update;
                    $canvas -> postscript( -file => "$selection.eps", -colormode => 'color', );
                }
            );

            $temp_w -> bind( $temp_w, "<Control-q>" => sub { $temp_w -> destroy; }
            );

            $canvas -> CanvasBind("<$ >", sub {
                $canvas -> yviewMoveto( 0 )
            } ) for ( 4, 5, );

            foreach ( @frames_clusters ) {
                 if ( /(\d+)\s*(\d+)/ ) {
                     $canvas  ->  createRectangle( 58  +  ( ( $1  /  $noframes  )  *  ( $width  -  58  )
)                       , 20 * ( $nofclusters - $2 ),
                                                   58 + ( ( $1 / $noframes ) * ( $width - 58 ) ) + (
$width / $noframes ), 20 * ( 1 + $nofclusters - $2 ),
                                                   -fill => $colors[$2-1], -outline => undef, );
```

```
                }
            }

            for ( my $i = 1 ; $i <= $nofclusters ; $i++ ) {
                $canvas -> createText( 27, ( $nofclusters * 20 ) - ( 20 * $i )  + 10, -fill => ,
$colors[$i-1], -text => sprintf ( "Cluster %2d", $i), );
                 $canvas  -> createLine( 0,  ( $nofclusters  * 20  ) - ( 20  * $i  ), $width,  (
$nofclusters * 20 ) - ( 20 * $i ), -fill => 'black', -width => 1, );
            }

            $canvas -> createLine( 56, $height, 56, 0, -fill => 'black', -width => 2, ); #y_axis
             $canvas  -> createLine( 0, $height, $width, $height, -fill => 'black', -width => 2, );
#x_axis
        }
        else {
            plot ( $selection );
        }
    } );
    unless ( ( $linux or $mac ) and not $ps_viewer ) {
        $lb3 -> bind( '<Button-1>', sub {
            my $selection = $lb3 -> get( $lb3 -> curselection() );

            if ( ( $linux || $mac ) && $ps_viewer ) {
                if ( $ps_viewer eq 'gs' ) {
                    system ( "$ps_viewer $selection" );
                }
                else {
                    system ( "$ps_viewer $selection &" );
                }
            }
            else {
                `start $selection`;
            }
        } );
    }
}

#############################################################################################################
###   Draw the window for residue selection                                                               ###
#############################################################################################################

sub select_residues {
    my @lines_to_modify;

    our $is_ext_psf;

    my $pos = '';
    my $prev_line = '';
    our $prev_psf = $active_psf unless ( $active_psf eq 'selected_residues.psf' );

    create_dir();

    if ( $active_psf eq 'selected_residues.psf' ) {
        open PSF_FILE, '<', $prev_psf || die "Cannot open $prev_psf for reading\n";
    }
    else {
        open PSF_FILE, '<', $active_psf || die "Cannot open $active_psf for reading\n";
    }

    open OUT, '>', "selected_residues.psf" || die "Cannot open selected_residues.psf for writing\n";

    # As soon as '!NATOM' is met, reading #
    # of the .psf file and output to the  #
    # custom psf file is stopped          #
    my $nofatoms = 0;
    while ( <PSF_FILE> ) {
        print OUT $_;
        if ( /(\d+) \!NATOM/ ) {
            $nofatoms = $1;
            last;
        }
    }

    # For every resid bar                  #
```

```
    for ( my $i = 0 ; $i <= $resid_bar_count ; $i++ ) {
        # If the $pos variable exists move to  #
        # the point of the filehandle defined  #
        # by it                                #
        if ( $pos ) {
            seek PSF_FILE, $pos, 0;

            while ( my $line = <PSF_FILE> ) {
                if ( $line !~ /^(\s*\d+\s+)($dropdown_value[$i])(\s+)(\d+)(.+)$/ ) {
                    print OUT $line;
                }
                else {
                    print OUT $line;
                    $pos = tell;
                    last;
                }
            }

            seek PSF_FILE, $pos, 0;
        }

        # Else continue reading the .psf file  #
        # from the next line after the one     #
        # containing '!NATOM'                  #
        while ( <PSF_FILE> ) {
            # If the pattern is met                #
            if ( /^(\s*)(\d+)(\s+)($dropdown_value[$i])(\s+)(\d+)(.+)$/ ) {
                # And the residue number equals the    #
                # upper limit set by the user store in #
                # $pos the location in the filehandle  #
                # and in $prev_line the line just read #
                # and exit the while loop              #
                if ( $6 == $upper_res_limit[$i] + 1 ) {
                    $pos = tell;
                    print OUT $_;
                    last;
                }
                elsif ( $6 == $num_residues{$dropdown_value[$i]} ) {
                    printf OUT "%s%s%s%-4s %s%s\n", $1, $2, $3, 'Z', $6, $7 if ( not $is_ext_psf );
                    printf OUT "%s%s%s%-4s     %s%s\n", $1, $2, $3, 'Z', $6, $7 if ( $is_ext_psf );

                    while ( <PSF_FILE> ) {
                        if ( /^(\s*)(\d+)(\s+)($dropdown_value[$i])(\s+)(\d+)(.+)$/ ) {
                            printf OUT "%s%s%s%-4s  %s%s\n", $1, $2, $3, 'Z', $6, $7 if ( not
$is_ext_psf );
                            printf OUT "%s%s%s%-4s     %s%s\n", $1, $2, $3, 'Z', $6, $7 if (
$is_ext_psf );
                        }
                        else {
                            print OUT $ ;
                            $pos = tell;
                            last;
                        }
                    }

                    last;
                }

                # If the line contains a residue whose #
                # number falls between the limits set  #
                # by the user export that line to the  #
                # custom .psf file while changing the  #
                # chain id to 'Z'                      #
                # Otherwise export the line as is      #
                if ( $6 >= $lower_res_limit[$i] && $6 <= $upper_res_limit[$i] ) {
                    printf OUT "%s%s%s%-4s %s%s\n", $1, $2, $3, 'Z', $6, $7 if ( not $is_ext_psf );
                    printf OUT "%s%s%s%-4s     %s%s\n", $1, $2, $3, 'Z', $6, $7 if ( $is_ext_psf );
                }
                else {
                    print OUT $_;
                }
            }
            else {
                print OUT $_;
```

```
            }
        }
    }

    # Set the position of the filehandle   #
    # to the one specified by $pos, print  #
    # the line which would have been       #
    # skipped if not for $prev_line, and   #
    # print the rest of the .psf to the    #
    # custom file                          #
    seek PSF_FILE, $pos, 0;
    #~ print OUT $prev_line;

    while ( <PSF_FILE> ) {
        print OUT $_;
    }

    close OUT;
    close PSF_FILE;

    $active_psf = 'selected_residues.psf';

    $active_psf_label -> configure( -text => "$active_psf ", );
    $active_dcd_label -> configure( -text => "$active_dcd", );

    #~ $count++;

    $dpca_run_button -> configure( -state => 'normal', ) if ( $dpca_run_button );
    $surf_run_button -> configure( -state => 'normal', ) if ( $surf_run_button );
    $qfract_run_button -> configure( -state => 'normal', ) if ( $qfract_run_button );
    $phi_psi_run_button -> configure( -state => 'normal', ) if ( $phi_psi_run_button );

    $text -> insert ( 'end', "\nYou have submitted a residue selection which " .
                             "resulted in the creation of a new .psf file. " .
                             "While the \"Change\" radiobutton is selected all" .
                             " the calculations will be made with the custom " .
                             "PSF file. By selecting the \"All\" radiobutton" .
                             " the selected PSF file reverts to the previous " .
                             "active PSF.\n", 'info' );
    $text -> see( 'end', );

    #~ $seg_id_flag = '' if ( $seg_id_flag );

    $res_id_flag = " -segid Z";
    $active_run_buttons = 1;

    $go_back_button -> configure( -state => 'normal', );
}

#############################################################################################
###   Create fit.index                                                                    ###
#############################################################################################

sub create_fit_index {
    my ( $atmid, $selection, $segids, ) = @_;

    my $fit_regex = '';
    my @atom_selection;
    my @segids;

    if ( $segids  ) {
        @segids = split ' ', $segids;
    }

    if ( @segids ) {
        local $" = '|';

        if ( $atm_id_flag ) {
            if ( $atmid =~ /backbone/i ) {
                $fit_regex = qr{^(\s+\d+\s+)(@segids)(\s+)(\d+)(\s+\w+\s+)(C|CA|N|O)(\s+.*)$};
            }
            elsif ( $atmid =~ /heavy/i ) {
                $fit_regex = qr{^(\s+\d+\s+)(@segids)(\s+)(\d+)(\s+\w+\s+)([^H].*)(\s+.*)$};
            }
```

```
            elsif ( $atmid =~ /allid/i ) {
                $fit_regex = qr{^(\s+\d+\s+)(@segids)(\s+)(\d+)(\s+\w+\s+)(\w+)(\s+.*)$};
            }
            else {print '4';

                $fit_regex = qr{^(\s+\d+\s+)(@segids)(\s+)(\d+)(\s+\w+\s+)(CA)(\s+.*)$};
            }
        }
        elsif ( $custom_id_flag ) {
            @atom_selection = split ' -atmid', $custom_id_flag;
            shift @atom_selection;

            $fit_regex = qr{^(\s+\d+\s+)(@segids)(\s+)(\d+)(\s+\w+\s+)(@atom_selection)(\s+.*)$};
        }
        else {
            $fit_regex = qr{^(\s+\d+\s+)(@segids)(\s+)(\d+)(\s+\w+\s+)(CA)(\s+.*)$};
        }
    }
    else {
        if ( $atm_id_flag ) {
            if ( $atmid =~ /backbone/i ) {
                $fit_regex = qr{^(\s+\d+\s+)(\w+)(\s+)(\d+)(\s+\w+\s+)(C|CA|N|O)(\s+.*)$};
            }
            elsif ( $atmid =~ /heavy/i ) {
                $fit_regex = qr{^(\s+\d+\s+)(\w+)(\s+)(\d+)(\s+\w+\s+)([^H].*)(\s+.*)$};
            }
            elsif ( $atmid =~ /allid/i ) {
                $fit_regex = qr{^(\s+\d+\s+)(\w+)(\s+)(\d+)(\s+\w+\s+)(\w+)(\s+.*)$};
            }
            else {
                $fit_regex = qr{^(\s+\d+\s+)(\w+)(\s+)(\d+)(\s+\w+\s+)(CA)(\s+.*)$};
            }
        }
        elsif ( $custom_id_flag ) {
            @atom_selection = split ' -atmid', $custom_id_flag;
            shift @atom_selection;

            $fit_regex = qr{^(\s+\d+\s+)(\w+)(\s+)(\d+)(\s+\w+\s+)(@atom_selection)(\s+.*)$};
        }
        else {
            $fit_regex = qr{^(\s+\d+\s+)(\w+)(\s+)(\d+)(\s+\w+\s+)(CA)(\s+.*)$};
        }
    }

    # The same as above but for the index  #
    # subroutine                           #
     open PSF, '<', "carma.selected_atoms.psf" || die "Cannot open carma.selected_atoms.psf for
reading:$!\n";
    open OUT, '>', "fit.index" || die "Cannot open fit.index for writing: $!\n";

    $index_num_atoms = 0;
    while ( <PSF> ) {
        if ( /(\d*) \!NATOM/ ) {
            $index_num_atoms = $1;
            last;
        }
    }

    if ( $selection =~ /radio/ ) {
        while ( <PSF> ) {
            if ( /$fit_regex/ ) {
                print OUT $_;
            }
        }
    }
    else {
        #~ my $fit_atom_count = 1;
        my $index_pos = '';
        my $line_count = 1;
        for ( my $i = 0 ; $i < $index_bar_count ; $i++ ) {
            if ( $index_pos ) {
                seek PSF, $index_pos, 0;
            }
```

```
            while ( my $index_line = <PSF> ) {
                if ( $index_line =~ /$fit_regex/ ) {
                    if ( $4 > $upper_fit_limit[$i] ) {
                        $index_pos = tell;
                        last;
                    }

                    if ( $4 >= $lower_fit_limit[$i] && $4 <= $upper_fit_limit[$i] ) {
                        print OUT "$index_line";
                    }
                }
                $line_count++;
            }
        }
    }

    close OUT;
    close PSF_FILE;

    $text -> insert ( 'end', "\nYou have submitted a residue selection which " .
                             "resulted in the creation of a fit.index file.", 'info' );
    $text -> see( 'end', );

}

##########################################################################################################
###   Create new 'select residues bar'                                                                 ###
##########################################################################################################

sub resid_window {
    my $top_res;

    if ( !Exists ( $top_res ) ) {
        foreach ( keys %num_residues ) {
            $text -> insert( 'end', "\n$num_residues{$_} residues in chain $_\n", 'info' );
            $text -> see( 'end', );
        }

        $top_res = $mw -> Toplevel( -title => 'Residue Selection', );
        $top_res -> geometry("$toplevel_position");
        $top_res -> protocol( 'WM_DELETE_WINDOW' => sub { $top_res -> withdraw });

        my $frame_res0 = $top_res -> Frame( -borderwidth => 3,
                                            -relief => 'groove',)
                                            -> pack( -fill => 'x', );

        $frame_res1[$resid_bar_count] = $top_res -> Frame() -> pack();

        $frame_res1[$resid_bar_count] -> Button( -text => 'Add..',
                                                 -width => 10,
                                                 -command => sub {
            $resid_bar_count++;
            $frame_res1[$resid_bar_count] = $top_res -> Frame() -> pack() unless ( $resid_bar_count
== 0 );

            add_resid_bar();

            if ( $resid_bar_count >= 1 ) {
                $frame_res1[$resid_bar_count] -> Button( -text => 'Remove',
                                                         -width => 10,
                                                         -command => sub {
                                                        $frame_res1[$resid_bar_count] -> destroy;
                                                        $resid_bar_count--;
                                                    }, )
                                                    -> grid( -row => "$resid_row", -column =>
"$resid_column" + 5, );
            }
            }, )
        -> grid( -row => 2, -column => 6, );

        my $frame_res2 = $top_res -> Frame() -> pack( -side => 'bottom', -expand => 0, );

        $frame_res2 -> Button( -text => 'Return',
                               -command => [ $top_res => 'withdraw' ], )
```

```
                                -> pack( -side => 'left', );

        $frame_res2 -> Button( -text => 'Submit',
                                -command => sub {
            my $i;
            my $check = 0;
            for ( $i = 0 ; $i <= $resid_bar_count ; $i++ ) {
                if ( $lower_res_limit[$i] and $upper_res_limit[$i] and $dropdown_value[$i] ) {
                    $check++;
                }
            }

            if ( $check == $i ) {
                if ( $seg_id_flag ) {
                    $seg_id_flag = '';
                }
                select_residues();

                $top_res -> destroy;
                $resid_bar_count = 0;
            }
            else {
                if ( $linux or $mac ) {
                    $top_res -> messageBox( -message => 'All of the boxes must be filled in order to
submit a residue selection',
                                            -icon => 'warning',
                                            -font => "$font_12", );
                }
                else {
                    $top_res -> messageBox( -message => 'All of the boxes must be filled in order to
submit a residue selection',
                                            -icon => 'warning',
                                            -font => "$font_12", );
                }
            }
        }, )
        -> pack( -side => 'right', );

        $resid_row = 1;
        $resid_column = 4;

        $frame_res0 -> Label( -text => "\nPlease specify the selections in ascending order for each
chain\n", ) -> pack;

        add_resid_bar();
    }
    else {
        $top_res -> deiconify;
        $top_res -> raise;
    }
}

#############################################################################################
###   Create a new bar for residue selection                                              ###
#############################################################################################

sub add_resid_bar {
    # Create a new frame everytime a new   #
    # bar is vreated and insert the bar in #
    # that frame                           #
    $resid_row++;
    $resid_column = 1;

    $frame_res1[$resid_bar_count] -> Label( -text => 'From: ', )
                                            ->   grid(   -row   =>   "$resid_row",   -column   =>
"$resid_column", );
    $frame_res1[$resid_bar_count] -> Entry( -textvariable => \$lower_res_limit[$resid_bar_count], )
                                            ->   grid(   -row   =>   "$resid_row",   -column   =>
"$resid_column" + 1, );
    $frame_res1[$resid_bar_count] -> Label( -text => 'To: ', )
                                            ->   grid(   -row   =>   "$resid_row",   -column   =>
"$resid_column" + 2, );
    $frame_res1[$resid_bar_count] -> Entry( -textvariable => \$upper_res_limit[$resid_bar_count], )
                                            ->   grid(   -row   =>   "$resid_row",   -column   =>
```

```
"$resid_column" + 3, );

    $dropdown[$resid_bar_count] = $frame_res1[$resid_bar_count] -> BrowseEntry( -label => "in chain:
",
                                                                                -variable   =>   \
$dropdown_value[$resid_bar_count], )
                                                                               -> grid( -row  =>
"$resid_row", -column => "$resid_column" + 4, );
    my $chain_counter = 0;
    foreach ( keys %num_residues ) {
        $dropdown[$resid_bar_count] -> insert( 'end', $_ );
        $chain_counter++;
    }

    if ( $chain_counter == 1 ) {
        foreach ( keys %num_residues ) {
            $dropdown_value[$resid_bar_count] = $_;
        }
    }
}

#############################################################################################################
###   Create a new bar for fit.index creation                                                             ###
#############################################################################################################

sub add_index_bar {
    $index_row = 1;
    $index_column = 1;

    $frame_fit_bars[$index_bar_count] -> Label( -text => 'From: ', )
                                          ->   grid(   -row   =>   "$index_row",   -column   =>
"$index_column", );
          $frame_fit_bars[$index_bar_count]       ->       Entry(       -textvariable       =>       \
$lower_fit_limit[$index_bar_count], )
                                          ->   grid(   -row   =>   "$index_row",   -column   =>
"$index column" + 1, );
    $frame_fit_bars[$index_bar_count] -> Label( -text => 'To: ', )
                                          ->   grid(   -row   =>   "$index_row",   -column   =>
"$index_column" + 2, );
          $frame_fit_bars[$index_bar_count]       ->       Entry(       -textvariable       =>       \
$upper_fit_limit[$index_bar_count], )
                                          ->   grid(   -row   =>   "$index_row",   -column   =>
"$index_column" + 3, );

    $fit_drop[$index_bar_count] = $frame_fit_bars[$index_bar_count] -> BrowseEntry( -label => "in
chain: ",
                                                                                -variable   =>   \
$fit_drop_value[$index_bar_count], )
                                                                               -> grid( -row  =>
"$index_row", -column => "$index_column" + 4, );

    my $chain_counter = 0;
    foreach ( keys %num_residues ) {
        $fit_drop[$index_bar_count] -> insert( 'end', $_ );
        $chain_counter++;
    }

    if ( $chain_counter == 1 ) {
        foreach ( keys %num_residues ) {
            $fit_drop_value[$index_bar_count] = $_;
        }
    }
}

#############################################################################################################
###   Draw the window for solute entropy calculation                                                      ###
#############################################################################################################

sub entropy_window {
    my $ent_first = 1;
    my $ent_first_flag = '';
    my $ent_last = dcd_header_parser( "entropy" );
    my $ent_last_flag = '';
    my $ent_step = '';
```

```
my $ent_temp = '';
my $lower_ent_limit = '';
my $upper_ent_limit = '';
my $top_ent;
my $entropy_plot;

my @a_entropy;
my @s_entropy;

if ( !Exists ( $top_ent ) ) {
    # Divide the number of frames in the   #
    # .dcd header by 10 and round it up    #
    if ( $ent_last ) {
        $ent_step = int ( ( $ent_last / 10 ) );
    }

    $top_ent = $mw -> Toplevel( -title => 'Solute entropy calculation', );
    $top_ent -> geometry("$toplevel_position");
    $top_ent -> protocol( 'WM_DELETE_WINDOW' => sub { $top_ent -> withdraw }, );

    my $frame_ent1 = $top_ent -> Frame( -borderwidth => 2, -relief => 'groove', );

    $frame_ent1 -> Label( -text => 'Temperature (K, required parameter): ', )
                        -> grid( -row => 1, -column => 1, -sticky => 'w', );
    $frame_ent1 -> Entry( -textvariable => \$ent_temp, )
                        -> grid( -row => 1, -column => 2, );
    $frame_ent1 -> Label( -text => 'First frame to use : ', )
                        -> grid( -row => 2, -column => 1, -sticky => 'w', );
    $frame_ent1 -> Entry( -textvariable => \$ent_first, )
                        -> grid( -row => 2, -column => 2, );
    $frame_ent1 -> Label( -text => 'Last frame to use : ', )
                        -> grid( -row => 3, -column => 1, -sticky => 'w', );
    $frame_ent1 -> Entry( -textvariable => \$ent_last, )
                        -> grid( -row => 3, -column => 2, );
    $frame_ent1 -> Label( -text => 'Step (in number of frames) for calculating entropy : ', )
                        -> grid( -row => 4, -column => 1, -sticky => 'w', );
    $frame_ent1 -> Entry( -textvariable => \$ent_step, )
                        -> grid( -row => 4, -column => 2, );

    our $frame_ent2 = $top_ent -> Frame() -> pack( -fill => 'x', );
    my $frame_ent3 = $top_ent -> Frame() -> pack( -fill => 'x', );
    my $frame_ent4 = $top_ent -> Frame() -> pack( -fill => 'x', );
    my $frame_ent6 = $top_ent -> Frame() -> pack( -fill => 'x', );
    $frame_ent1 -> pack( -fill => 'x', );

    radiobuttons ( $frame_ent2 );
    checkbuttons ( $frame_ent3 );
    otherbuttons ( $frame_ent4 );

    $frame_ent6 -> Label( -text => "\nVarious options", -font => $font_20, ) -> pack;

    my $frame_ent5 = $top_ent -> Frame() -> pack( -expand => 0, );

    $frame_ent5 -> Button( -text => 'Return',
                           -command => [ $top_ent => 'withdraw' ], )
                           -> pack( -side => 'left', );

    $frame_ent5 -> Button( -text => 'Run',
                           -command => sub {
            if ( $ent_step and $ent_temp ) {
                $ent_first_flag = ( ( $ent_first != 1 ) ? " -first $ent_first" : '' );
                $ent_last_flag = ( ( $ent_last != $header ) ? " -last $ent_last" : '' );

                open ENTROPY_FIRST_STEP, '>', 'entropy_first_step' or die $!;

                printf ENTROPY_FIRST_STEP "%8d", $ent_step;

                close ENTROPY_FIRST_STEP;

                # Make ten repeat runs each time using #
                # $ent_step more steps. This means in  #
                # the first run the first tenth of the #
                # frames will be used, in the second   #
                # the first fifth...                   #
```

```
                    $top_ent -> destroy;

                    $text -> insert( 'end', "\n\nNow calculating entropy.\n\n", 'cyan', );
                    $text -> see( 'end', );
                    $mw -> update;

                    # The result of the $i * $ent_step     #
                    # multiplication is the number of the  #
                    # frame that will be used after the    #
                    # ' -last' flag                        #
                    for ( my $i = 1 ; ( $i * $ent_step ) <= $ent_last ; $i++ ) {
                        # If that number exceeds the number of #
                        # frames in the .dcd header then that  #
                        # number will be used instead          #

                        $lower_ent_limit = $ent_first;
                        $upper_ent_limit = ( $i * $ent_step );
                        $text -> insert( 'end', "Calculating entropy for frames $lower_ent_limit -
$upper_ent_limit :\n\n", 'cyan' );
                        $text -> see( 'end', );

                        $seg_id_flag = '' if $seg_id_flag;

                        foreach ( @seg_ids ) {
                            if ( defined ( $_ ) ) {
                                $seg_id_flag = $seg_id_flag . $_;
                            }
                        }

                        if ( $res_id_flag ) {
                             $flag = " -v -cov -col -eigen -mass -temp $ent_temp $atm_id_flag
$res_id_flag -first $lower_ent_limit -last $upper_ent_limit";
                        }
                        elsif ( $seg_id_flag ) {
                             $flag = " -v -cov -col -eigen -mass -temp $ent_temp $atm_id_flag
$seg id flag -first $lower_ent_limit -last $upper_ent_limit";
                        }
                        else {
                            $flag = " -v -cov -col -eigen -mass -temp $ent_temp $atm_id_flag -first
$lower_ent_limit -last $upper_ent_limit";
                        }

                        create_dir();
                        carma();

                         open READ_ENTROPY, '<' , "last_carma_run.log" || die "Cannot open
last_carma_run.log for reading:$!";
                          open WRITE_ENTROPY, '>>', "carma_entropy.dat" || die "Cannot open
carma_entropy.dat for writing:$!";

                        # Parse the output file for the lines  #
                        # containing the results and save them #
                        # in a file named 'carma_entropy.dat'  #
                        # and two arrays, one for every type   #
                        # of entropy calculated by carma       #
                        while ( <READ_ENTROPY> ) {
                            if ( /Entropy \(Andricioaei\)(\s*)is (\d*\.\d*) (\(J\/molK\))/ ) {
                                $a_entropy[$i] = $2;
                                printf WRITE_ENTROPY ("%4d %15.5f\n", $i + 1, $2, );
                                $text -> insert( 'end', "\n$_", );
                                $text -> see( 'end', );
                            }
                            if ( /Entropy \(Schlitter\)(\s*)is (\d*\.\d*) (\(J\/molK\))/ ) {
                                $s entropy[$i] = $2;
                                printf WRITE_ENTROPY ("%4d %15.5f\n", $i + 1, $2, );
                                $text -> insert( 'end', "$_\n", );
                                $text -> see( 'end', );
                            }
                        }
                        close READ_ENTROPY;
                        $upper_ent_limit = 0;
                    }

                    if ( $all_done ) {
```

```
                        shift @a_entropy if ( @a_entropy );
                        shift @s_entropy if ( @s_entropy );

                        my $andrici_test = 1;
                        my $schlitter_test = 1;

                        close WRITE_ENTROPY;

                        if ( @a_entropy and @s_entropy ) {
                            my $i = 0;
                            while ( $i <= $#a_entropy ) {
                                if ( not $a_entropy[$i] ) {
                                    $a_entropy[$i] = -100;
                                    $andrici_test = 0;
                                }
                                else {
                                    $i++;
                                }
                            }

                            $i = 0;
                            while ( $i <= $#s_entropy ) {
                                if ( not $s_entropy[$i] ) {
                                    $s_entropy[$i] = -100;
                                    $schlitter_test = 0;
                                }
                                else {
                                    $i++;
                                }
                            }
                        }

                        # If arrays for both entropies exist   #
                        # overwrite the entropy file with the  #
                        # contents of those arrays             #
                        if ( ( @a_entropy and @s_entropy ) ) {
                             open WRITE_ENTROPY, '>', "carma_entropy.dat" || die "Cannot open
carma_entropy.dat for writing";
                            my $k = 0;
                            foreach ( @s_entropy ) {
                                $k++;
                            }

                            for ( my $j = 0 ; $j < $k ; $j++ ) {
                                 printf WRITE_ENTROPY ( "%4d %15.5f %15.5f\n", $j + 1,
$a_entropy[$j], $s_entropy[$j], );
                            }
                            close WRITE_ENTROPY;

                            if ( $andrici_test and $schlitter_test ) {
                                 $text -> insert( 'end', "\nCalculation finished. Use \"View
Results\"\n", 'valid' );
                                $text -> see( 'end', );
                                $image_menu -> configure( -state => 'normal', );
                            }
                            elsif ( $andrici_test and not $schlitter_test ) {
                                 $text -> insert( 'end', "\nCalculation finished. Sclitter entropy
could not be calculated for all frames. Use \"View Results\"\n", 'valid' );
                                $text -> see( 'end', );
                                $image_menu -> configure( -state => 'normal', );
                            }
                            elsif ( not $andrici_test and $schlitter_test ) {
                                $text -> insert( 'end', "\nCalculation finished. Andricioaei entropy
could not be calculated for all frames. Use \"View Results\"\n", 'valid' );
                                $text -> see( 'end', );
                                $image_menu -> configure( -state => 'normal', );
                            }
                        }
                        else {
                            if ( @a_entropy ) {
                                open WRITE_ENTROPY, '>', "carma_entropy_andricioaei" || die "Cannot
open carma_entropy_andricioaei.dat for writing";

                                my $k = 0;
```

```
                                foreach ( @a_entropy ) {
                                    $k++;
                                }

                                for ( my $j = 0 ; $j < $k ; $j++ ) {
                                    printf WRITE_ENTROPY ( "%4d %15.5f\n", $j + 1, $a_entropy[$j],
);
                                }
                                close WRITE_ENTROPY;

                                 $text -> insert( 'end', "\nCalculation finished but Schlitter
entropy could not be calculated. Use \"View Results\"\n", 'valid' );
                                $text -> see( 'end', );
                                $image_menu -> configure( -state => 'normal', );
                            }
                            if ( @s_entropy ) {
                                 open WRITE_ENTROPY, '>', "carma_entropy_schlitter" || die "Cannot
open carma_entropy_schlitter.dat for writing";

                                my $k = 0;
                                foreach ( @s_entropy ) {
                                    $k++;
                                }

                                for ( my $j = 0 ; $j < $k ; $j++ ) {
                                    printf WRITE_ENTROPY ( "%4d %15.5f\n", $j + 1, $s_entropy[$j],
);
                                }
                                close WRITE_ENTROPY;

                                 $text -> insert( 'end', "\nCalculation finished but Andricioaei
entropy could not be calculated. Use \"View Results\"\n", 'valid' );
                                $text -> see( 'end', );
                                $image_menu -> configure( -state => 'normal', );
                            }

                            unlink ( 'carma_entropy.dat' );
                        }

                        if ( $active_dcd =~ /(.+).dcd/ ) {
                            mv ( 'carma.PCA.eigenvalues.dat', "entropy.$1.eigenvalues" );
                            mv ( "$1.dcd.varcov.ps", "covariance.$1.ps" );

                            if ( -f 'carma_entropy.dat' ) {
                                mv ( 'carma_entropy.dat', "entropy.$1.dat" );
                            }
                            if ( -f 'carma_entropy_andricioaei' ) {
                                mv ( 'carma_entropy_andricioaei', "andricioaei_entropy.$1.dat" );
                            }
                            if ( -f 'carma entropy schlitter' ) {
                                mv ( 'carma_entropy_schlitter', "schlitter_entropy.$1.dat" );
                            }
                        }

                    }
                    else {
                         $text -> insert( 'end', "\nSomething went wrong. For details check
last_carma_run.log located in :\n", 'error', );
                        $text -> insert( 'end', getcwd . "\n", 'info', );
                        $text -> see( 'end', );
                    }
                }
                else {
                    if ( $linux or $mac ) {
                        $top_ent -> messageBox( -message => 'Both step and temperature need to be
defined for entropy calculation',
                                                -icon => 'warning',
                                                -font => "$font_12", );
                    }
                    else {
                        $top_ent -> messageBox( -message => 'Both step and temperature need to be
defined for entropy calculation',
                                                -icon => 'warning', );
                    }
```

```
                }
        }, )
        -> pack( -side => 'right', );
    }
    else {
        $top_ent -> deiconify;
        $top_ent -> raise;
    }
}

#############################################################################################################
###   Create PDB files for the specified frames                                                           ###
#############################################################################################################

sub pdb_window {
    my $top_pdb;
    my $pdb_step;
    my $pdb_step_flag;
    my $pdb_first = 1;
    my $pdb_first_flag = '';
    my $pdb_last = dcd_header_parser( "pdb" );
    my $pdb_last_flag = '';

    if ( !Exists ( $top_pdb ) ) {
        if ( $pdb_last > 500 ) {
            $pdb_step = int ( ( $pdb_last / 500 ) + 0.5 );
        } else {
            $pdb_step = 1;
        }

        $top_pdb = $mw -> Toplevel( -title => 'Extract selected PDB files', );
        $top_pdb -> geometry("$toplevel_position");
        $top_pdb -> protocol( 'WM_DELETE_WINDOW' => sub { $top_pdb -> withdraw }, );

        my $frame_pdb2 = $top_pdb -> Frame() -> pack( -fill => 'x', );
        my $frame_pdb3 = $top_pdb -> Frame() -> pack( -fill => 'x', );
        my $frame_pdb4 = $top_pdb -> Frame() -> pack( -fill => 'x', );
        my $frame_pdb6 = $top_pdb -> Frame() -> pack( -fill => 'x', );
        my $frame_pdb1 = $top_pdb -> Frame( -borderwidth => 2, -relief => 'groove', ) -> pack( -fill
=> 'x', );

        radiobuttons ( $frame_pdb2 );
        checkbuttons ( $frame_pdb3 );
        otherbuttons ( $frame_pdb4 );

        $frame_pdb6 -> Label( -text => "\nVarious options", -font => $font_20, ) -> pack;

        $frame_pdb1 -> Label( -text => 'First frame to use: ', )
                        -> grid( -row => 1, -column => 1, -sticky => 'w', );
        $frame_pdb1 -> Entry( -textvariable => \$pdb_first, )
                        -> grid( -row => 1, -column => 2, );
        $frame_pdb1 -> Label( -text => 'Last frame to use: ', )
                        -> grid( -row => 2, -column => 1, -sticky => 'w', );
        $frame_pdb1 -> Entry( -textvariable => \$pdb_last, )
                        -> grid( -row => 2, -column => 2, );
        $frame_pdb1 -> Label( -text => 'Stride (step) between frames: ' . ' ' x 27, )
                        -> grid( -row => 3, -column => 1, -sticky => 'w', );
        $frame_pdb1 -> Entry( -textvariable => \$pdb_step, )
                        -> grid( -row => 3, -column => 2, );

        my $frame_pdb5 = $top_pdb -> Frame() -> pack( -expand => 0, );

        $frame_pdb5 -> Button( -text => 'Return',
                               -command => [ $top_pdb => 'withdraw' ], )
                               -> pack( -side => 'left', );

        $frame_pdb5 -> Button( -text => 'Run',
                               -command => sub {
        $pdb_first_flag = ( ( $pdb_first != 1 ) ? " -first $pdb_first" : '' );
        $pdb_last_flag = ( ( $pdb_last != $header ) ? " -last $pdb_last" : '' );
        $pdb_step_flag = ( ( $pdb_step != 1 ) ? " -step $pdb_step" : '' );

        $top_pdb -> destroy;
```

```
        $seg_id_flag = '' if $seg_id_flag;

        foreach ( @seg_ids ) {
            if ( defined ( $_ ) ) {
                $seg_id_flag = $seg_id_flag . $_;
            }
        }

        if ( $res_id_flag ) {
             $flag  =  "  -v  -w  -pdb  $pdb_step_flag  $pdb_first_flag  $pdb_last_flag  $atm_id_flag
$custom_id_flag $res_id_flag";
        }
        elsif ( $seg_id_flag ) {
             $flag  =  "  -v  -w  -pdb  $pdb_step_flag  $pdb_first_flag  $pdb_last_flag  $atm_id_flag
$custom_id_flag $seg_id_flag";
        }
        else {
             $flag  =  "  -v  -w  -pdb  $pdb_step_flag  $pdb_first_flag  $pdb_last_flag  $atm_id_flag
$custom_id_flag";
        }

        create_dir();

        $text -> insert( 'end', "\n\nNow extracting pdb files.\n\n", 'cyan', );
        $text -> see( 'end', );
        $mw -> update;

        carma();

        if ( $all_done ) {
            $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n", 'valid' );
            $text -> see( 'end', );
            $image_menu -> configure( -state => 'normal', );
        }
        else {
             $text -> insert( 'end', "\nSomething went wrong. For details check last_carma_run.log
located in :\n", 'error', );
            $text -> insert( 'end', getcwd . "\n", 'info', );
            $text -> see( 'end', );
        }
        }, )
        -> pack( -side => 'right', );

    }
    else {
        $top_pdb -> deiconify;
        $top_pdb -> raise;
    }

}

##########################################################################################################
###   Draw the window for distance maps                                                                ###
##########################################################################################################

sub rms_window {
    my $rms_first = 1;
    my $rms_first_flag = '';
    my $rms_last = dcd_header_parser( "rms" );
    my $rms_last_flag = '';
    my $rms_step = 1;
    my $rms_step_flag = '';
    my $rms_min = '';
    my $rms_min_flag = '';
    my $rms_max = '';
    my $rms_max_flag = '';
    my $rms_mrms = '';
    my $rms_mrms_flag = '';
    my $rms_reverse = '';
    my $average_ps_file = '';
    my $rmsdev_ps_file = '';
    my $top_rms;

    if ( !Exists ( $top_rms ) ) {
```

```
        $top_rms = $mw -> Toplevel( -title => 'Average distance and rms deviation from them', );
        $top_rms -> geometry("$toplevel_position");
        $top_rms -> protocol( 'WM_DELETE_WINDOW' => sub { $top_rms -> withdraw }, );

        my $frame_rms1 = $top_rms -> Frame() -> pack( -fill => 'x', );
        my $frame_rms2 = $top_rms -> Frame() -> pack( -fill => 'x', );
        my $frame_rms3 = $top_rms -> Frame() -> pack( -fill => 'x', );
        my $frame_rms6 = $top_rms -> Frame() -> pack( -fill => 'x', );

        radiobuttons ( $frame_rms1 );
        checkbuttons ( $frame_rms2 );
        otherbuttons ( $frame_rms3 );

        $frame_rms6 -> Label( -text => "\nVarious Options", -font => $font_20, ) -> pack;

        my $frame_rms4 = $top_rms -> Frame( -borderwidth => 2, -relief => 'groove', ) -> pack(
-expand => 0, );

        $frame_rms4 -> Label( -text => 'First frame to use: ', )
                    -> grid( -row => 1, -column => 1, -sticky => 'w', );
        $frame_rms4 -> Entry( -textvariable => \$rms_first, )
                    -> grid( -row => 1, -column => 2, );
        $frame_rms4 -> Label( -text => 'Last frame to use: ', )
                    -> grid( -row => 2, -column => 1, -sticky => 'w', );
        $frame_rms4 -> Entry( -textvariable => \$rms_last, )
                    -> grid( -row => 2, -column => 2, );
        $frame_rms4 -> Label( -text => 'Stride (step) between frames: ', )
                    -> grid( -row => 3, -column => 1, -sticky => 'w', );
        $frame_rms4 -> Entry( -textvariable => \$rms_step, )
                    -> grid( -row => 3, -column => 2, );

        $frame_rms4 -> Label( -text => 'Minimum value for postscript plot (dark blue): ', )
                    -> grid( -row => 4, -column => 1, -sticky => 'w', );
        $frame_rms4 -> Entry( -textvariable => \$rms_min, )
                    -> grid( -row => 4, -column => 2, );
        $frame_rms4 -> Label( -text => 'Maximum value for postscript plot (dark red): ', )
                    -> grid( -row => 5, -column => 1, -sticky => 'w', );
        $frame_rms4 -> Entry( -textvariable => \$rms_max, )
                    -> grid( -row => 5, -column => 2, );

        $frame_rms4 -> Checkbutton( -text => 'Reverse color gradient',
                                    -variable => \$rms_reverse,
                                    -offvalue => '',
                                    -onvalue => " -reverse", )
                                    -> grid( -row => 6, -column => 1, -sticky => 'w', );

        my $frame_rms5 = $top_rms -> Frame() -> pack( -expand => 0, );

        $frame_rms5 -> Button( -text => 'Return',
                               -command => [ $top_rms => 'withdraw' ], )
                               -> pack( -side => 'left', );

        $frame_rms5 -> Button( -text => 'Run',
                               -command => sub {
        $rms_first_flag = ( ( $rms_first != 1 ) ? " -first $rms_first" : '' );
        $rms_last_flag = ( ( $rms_last != $header ) ? " -last $rms_last" : '' );
        $rms_step_flag = ( ( $rms_step != 1 ) ? " -step $rms_step" : '' );
        $rms_min_flag = ( $rms_min ? " -min $rms_min" : '' );
        $rms_max_flag = ( $rms_max ? " -max $rms_max" : '' );
        $rms_mrms_flag = ( $rms_mrms ? " -mrms $rms_mrms" : '' );

        $top_rms -> destroy;

        if ( $rms_first && $rms_last && $rms_first > 0 && $rms_first == $rms_last ) {
            $seg_id_flag = '' if $seg_id_flag;

            foreach ( @seg_ids ) {
                if ( defined ( $_ ) ) {
                    $seg_id_flag = $seg_id_flag . $_;
                }
            }

            if ( $seg_id_flag ) {
                $flag = " -v -w -col $rms_first_flag $rms_last_flag $atm_id_flag $custom_id_flag
```

```
$res_id_flag $seg_id_flag";
            }
            else {
                $flag = " -v -w -col $rms_first_flag $rms_last_flag $atm_id_flag $custom_id_flag
$res_id_flag";
            }
        }
        else {
            $seg_id_flag = '' if $seg_id_flag;

            foreach ( @seg_ids ) {
                if ( defined ( $_ ) ) {
                    $seg_id_flag = $seg_id_flag . $_;
                }
            }

            if ( $res_id_flag ) {
                 $flag = " -v -w -col -rms $rms_first_flag $rms_last_flag $rms_step_flag
$rms_min_flag $rms_max_flag $rms_mrms_flag $rms_reverse $atm_id_flag $custom_id_flag $res_id_flag";
            }
            elsif ( $seg_id_flag ) {
                 $flag = " -v -w -col -rms $rms_first_flag $rms_last_flag $rms_step_flag
$rms_min_flag $rms_max_flag $rms_mrms_flag $rms_reverse $atm_id_flag $custom_id_flag $seg_id_flag";
            }
            else {
                 $flag = " -v -w -col -rms $rms_first_flag $rms_last_flag $rms_step_flag
$rms_min_flag $rms_max_flag $rms_mrms_flag $rms_reverse $atm_id_flag $custom_id_flag";
            }
        }

        create_dir();

        $text -> insert( 'end', "\n\nNow calculating average Ca - Ca distances.\n\n", 'cyan', );
        $text -> see( 'end', );
        $mw -> update;

        carma();

        if ( $all_done ) {
            my ( $max_rms, $max_avg, );

            open RMS_OUT, "last_carma_run.log" || die "Cannot open last_carma_run.log for reading:
$!";
            while ( <RMS_OUT> ) {
                if ( /Writing postscript file (\w*\.dcd\.averag.ps)/ ) {
                    $average_ps_file = $1;
                }
                if ( /Writing postscript file (\w*\.dcd\.rmsdev.ps)/ ) {
                    $rmsdev_ps_file = $1;
                }
                if ( /Maximum of observed average distances is (\d+.?\d*)/ ) {
                    $max_avg = sprintf "%.2f", $1;
                }
                if ( /Maximum rms deviation from average distances is (\d+.?\d*)/ ) {
                    $max_rms = sprintf "%.2f", $1;
                }
            }
            close RMS_OUT;

            if ( $linux || $mac ) {
                `carma $average_ps_file $rmsdev_ps_file`;
            }
            else {
                `carma.exe $average_ps_file $rmsdev_ps_file`;
            }

            if ( $active_dcd =~ /(.+).dcd/ ) {
                mv ( "$1.dcd.averag.ps", "distance_map.average.$1.ps" );
                mv ( "$1.dcd.rmsdev.ps", "distance_map.RMSD.$1.ps" );
                mv ( "carma.merged.ps", "distance_map.combined.$1.ps" );
            }

            if ( $rms_reverse ) {
                $text -> insert( 'end', "\nAverage distances range from 0.0 (dark red) to " .
```

```
$max_avg . "A (dark blue)\n", ) if ( $max_avg );
                $text -> insert( 'end', "RMSDs of distances range from 0.0 (dark blue) to " .
$max_rms . "A (dark red)\n\n", ) if ( $max_rms );
            }
            else {
                $text -> insert( 'end', "\nAverage distances range from 0.0 (dark blue) to " .
$max_avg . "A (dark red)\n", ) if ( $max_avg );
                $text -> insert( 'end', "RMSDs of distances range from 0.0 (dark red) to " .
$max_rms . "A (dark blue)\n\n", ) if ( $max_rms );
            }

            $text -> insert( 'end', "Calculation finished. Use \'View Results\'\n", 'valid' );
            $text -> see( 'end', );
            $image_menu -> configure( -state => 'normal', );
        }
        else {
            $text -> insert( 'end', "\nSomething went wrong. For details check last_carma_run.log
located in :\n", 'error', );
            $text -> insert( 'end', getcwd . "\n", 'info', );
            $text -> see( 'end', );
        }
        }, )-> pack( -side => 'right', );
    }
    else {
        $top_rms -> deiconify;
        $top_rms -> raise;
    }
}

#############################################################################################
###   Draw the window for radius of gyration                                              ###
#############################################################################################

sub rgr_window {
    my $top_rgr;
    my $rgr_first = 1;
    my $rgr_first_flag = '';
    my $rgr_last = dcd_header_parser ( 'rgr' );
    my $rgr_last_flag = '';
    my $rgr_step = 1;
    my $rgr_step_flag = '';

    if ( !Exists ( $top_rgr ) ) {
        $top_rgr = $mw -> Toplevel( -title => 'Radius of gyration', );
        $top_rgr -> geometry("$toplevel_position");
        $top_rgr -> protocol( 'WM_DELETE_WINDOW' => sub { $top_rgr -> withdraw }, );

        our $frame_rgr1 = $top_rgr -> Frame() -> pack( -fill => 'x', );
        my $frame_rgr2 = $top_rgr -> Frame() -> pack( -fill => 'x', );
        my $frame_rgr3 = $top_rgr -> Frame() -> pack( -fill => 'x', );
        my $frame_rgr5 = $top_rgr -> Frame() -> pack( -fill => 'x', );
        my $frame_rgr6 = $top_rgr -> Frame( -borderwidth => 2, -relief => 'groove', ) -> pack( -fill
=> 'both', -expand => 1, );

        radiobuttons ( $frame_rgr1 );
        checkbuttons ( $frame_rgr2 );
        otherbuttons ( $frame_rgr3 );

        $frame_rgr5 -> Label( -text => "\nVarious options", ) -> pack;

        $frame_rgr6 -> Label( -text => "First frame to use :", ) -> grid( -row => 1, -column => 1,
-sticky => 'w', );
        $frame_rgr6 -> Entry( -textvariable => \$rgr_first, ) -> grid( -row => 1, -column => 2, );
        $frame_rgr6 -> Label( -text => "Last frame to use :", ) -> grid( -row => 2, -column => 1,
-sticky => 'w', );
        $frame_rgr6 -> Entry( -textvariable => \$rgr_last, ) -> grid( -row => 2, -column => 2, );
        $frame_rgr6 -> Label( -text => "Stride ( step ) between frames :" . ' ' x 27, ) -> grid(
-row => 3, -column => 1, -sticky => 'w', );
        $frame_rgr6 -> Entry( -textvariable => \$rgr_step, ) -> grid( -row => 3, -column => 2, );

        my $frame_rgr4 = $top_rgr -> Frame() -> pack( -expand => 0, );

        $frame_rgr4 -> Button( -text => 'Return',
                               -command => [ $top_rgr => 'withdraw' ], )
```

```
                                -> pack( -side => 'left', );

        $frame_rgr4 -> Button( -text => 'Run',
                               -command => sub {
            $rgr_first_flag = ( ( $rgr_first != 1 ) ? " -first $rgr_first" : '' );
            $rgr_last_flag = ( ( $rgr_last != $header ) ? " -last $rgr_last" : '' );
            $rgr_step_flag = ( ( $rgr_step != 1 ) ? " -step $rgr_step" : '' );

            $top_rgr -> destroy;

            $seg_id_flag = '' if $seg_id_flag;

            foreach ( @seg_ids ) {
                if ( defined ( $_ ) ) {
                    $seg_id_flag = $seg_id_flag . $_;
                }
            }

            if ( $res_id_flag ) {
                 $flag  =  "  -v  -rg  $atm_id_flag  $res_id_flag  $custom_id_flag  $rgr_first_flag
$rgr_last_flag $rgr_step_flag";
            }
            elsif ( $seg_id_flag ) {
                 $flag  =  "  -v  -rg  $atm_id_flag  $custom_id_flag  $seg_id_flag  $rgr_first_flag
$rgr_last_flag $rgr_step_flag";
            }
            else {
                 $flag  =  "  -v  -rg  $atm_id_flag  $custom_id_flag  $rgr_first_flag  $rgr_last_flag
$rgr_step_flag";
            }

            create_dir();

            $text -> insert( 'end', "\n\nNow calculating radious of gyration.\n\n", 'cyan', );
            $text -> see( 'end', );
            $mw -> update;

            carma();

            if ( $all_done ) {
                if ( $active_dcd =~ /(.+).dcd/ ) {
                    mv ( 'carma.Rgyration.dat', "Rgyration.$1.dat" );
                }

                $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n", 'valid' );
                $text -> see( 'end', );
                $image_menu -> configure( -state => 'normal', );
            }
            else {
                 $text  ->  insert(  'end',  "\nSomething  went  wrong.  For  details  check
last_carma_run.log located in :\n", 'error', );
                $text -> insert( 'end', getcwd . "\n", 'info', );
                $text -> see( 'end', );
            }
        }, )-> pack( -side => 'right', );
    }
    else {
        $top_rgr -> deiconify;
        $top_rgr -> raise;
    }
}

#############################################################################################
###   Draw the window for distances                                                       ###
#############################################################################################

sub dis_window {
    my $dis_atom1 = '';
    my $dis_atom2 = '';
    my $top_dis;
    my $dis_first = 1;
    my $dis_first_flag = '';
    my $dis_last = dcd_header_parser ( 'rgr' );
    my $dis_last_flag = '';
```

```
    my $dis_step = 1;
    my $dis_step_flag = '';

    my ( @list, ) = helper_function();

    if ( !Exists ( $top_dis ) ) {
        $top_dis = $mw -> Toplevel( -title => 'Distances', );
        $top_dis -> geometry("$toplevel_position");
        $top_dis -> protocol( 'WM_DELETE_WINDOW' => sub { $top_dis -> withdraw }, );

        my $frame_dis0 = $top_dis -> Frame() -> pack;
        my $frame_dis1 = $top_dis -> Frame() -> pack;

        $frame_dis0 -> Label( -text => 'Define atoms by selecting resiue number, chain identifier
and atom type', ) -> pack;

        $frame_dis1 -> Label( -text => 'Atom 1 :', ) -> grid( -row => 2, -column => 1, );
        our $dist_list1 = $frame_dis1 -> BrowseEntry( -variable => \$dis_atom1,
                                                      -listwidth => 50,
                                                      -autolistwidth => 0,
                                                      -browsecmd => sub {
            my $selection = our $dist_list1 -> get( 'active' );

            if ( $selection =~ /(\d+)$/ ) {
                $dis_atom1 = $1;
            }
        }, ) -> grid( -row => 2, -column => 2, );
        $frame_dis1 -> Label( -text => 'Atom 2 :', ) -> grid( -row => 2, -column => 3, );
        our $dist_list2 = $frame_dis1 -> BrowseEntry( -variable => \$dis_atom2,
                                                      -listwidth => 50,
                                                      -autolistwidth => 0,
                                                      -browsecmd => sub {
            my $selection = our $dist_list2 -> get( 'active' );

            if ( $selection =~ /(\d+)$/ ) {
                $dis_atom2 = $1;
            }
        }, ) -> grid( -row => 2, -column => 4, );

        $dist_list1 -> insert( 'end', @list, );
        $dist_list2 -> insert( 'end', @list, );

        our $frame_dis2 = $top_dis -> Frame() -> pack( -fill => 'x', );
        my $frame_dis3 = $top_dis -> Frame() -> pack( -fill => 'x', );
        my $frame_dis4 = $top_dis -> Frame() -> pack( -fill => 'x', );
        my $frame_dis6 = $top_dis -> Frame() -> pack( -fill => 'x', );
        my $frame_dis7 = $top_dis -> Frame( -borderwidth => 2, -relief => 'groove', ) -> pack( -fill
=> 'both', -expand => 1, );

        radiobuttons ( $frame_dis2 );
        checkbuttons ( $frame_dis3 );
        otherbuttons ( $frame_dis4 );

        $frame_dis6 -> Label( -text => "\nVarious options", -font => $font_20, ) -> pack;

        $frame_dis7 -> Label( -text => "First frame to use :", ) -> grid( -row => 1, -column => 1,
-sticky => 'w', );
        $frame_dis7 -> Entry( -textvariable => \$dis_first, ) -> grid( -row => 1, -column => 2, );
        $frame_dis7 -> Label( -text => "Last frame to use :", ) -> grid( -row => 2, -column => 1,
-sticky => 'w', );
        $frame_dis7 -> Entry( -textvariable => \$dis_last, ) -> grid( -row => 2, -column => 2, );
        $frame_dis7 -> Label( -text => "Stride ( step ) between frames :" . ' ' x 27, ) -> grid(
-row => 3, -column => 1, -sticky => 'w', );
        $frame_dis7 -> Entry( -textvariable => \$dis_step, ) -> grid( -row => 3, -column => 2, );

        my $frame_dis5 = $top_dis -> Frame() -> pack( -expand => 0, );

        $frame_dis5 -> Button( -text => 'Return',
                               -command => [ $top_dis => 'withdraw' ], )
                               -> pack( -side => 'left', );

        $frame_dis5 -> Button( -text => 'Run',
                               -command => sub {
        $dis_first_flag = ( ( $dis_first != 1 ) ? " -first $dis_first" : '' );
```

```
        $dis_last_flag = ( ( $dis_last != $header ) ? " -last $dis_last" : '' );
        $dis_step_flag = ( ( $dis_step != 1 ) ? " -step $dis_step" : '' );

        $top_dis -> destroy;

        $seg_id_flag = '' if $seg_id_flag;

        foreach ( @seg_ids ) {
            if ( defined ( $_ ) ) {
                $seg_id_flag = $seg_id_flag . $_;
            }
        }

        if ( $res_id_flag ) {
             $flag = " -v -dist $dis_atom1 $dis_atom2 $atm_id_flag $custom_id_flag $res_id_flag
$dis_first_flag $dis_last_flag $dis_step_flag";
        }
        elsif ( $seg_id_flag ) {
             $flag = " -v -dist $dis_atom1 $dis_atom2 $atm_id_flag $custom_id_flag $seg_id_flag
$dis_first_flag $dis_last_flag $dis_step_flag";
        }
        else {
            $flag = " -v -dist $dis_atom1 $dis_atom2 $atm_id_flag $custom_id_flag $dis_first_flag
$dis_last_flag $dis_step_flag";
        }

        create_dir();

        $text -> insert( 'end', "\n\nNow calculating distances.\n\n", 'cyan', );
        $text -> see( 'end', );
        $mw -> update;

        carma();

        if ( $all_done ) {
            if ( $active dcd =~ /(.+).dcd/ ) {
                mv ( 'carma.distances', "distances.$1.dat" );
            }

            $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n", 'valid' );
            $text -> see( 'end', );
            $image_menu -> configure( -state => 'normal', );
        }
        else {
            $text -> insert( 'end', "\nSomething went wrong. For details check last_carma_run.log
located in :\n", 'error', );
            $text -> insert( 'end', getcwd . "\n", 'info', );
            $text -> see( 'end', );
        }
        }, )-> pack( -side => 'right', );
    }
    else {
        $top_dis -> deiconify;
        $top_dis -> raise;
    }
}

###############################################################################################################
###   Draw the window for bending angles                                                                    ###
###############################################################################################################

sub bnd_window {
    my $bnd_atom1 = '';
    my $bnd_atom2 = '';
    my $bnd_atom3 = '';
    my $top_bnd;
    my $bnd_first = 1;
    my $bnd_first_flag = '';
    my $bnd_last = dcd_header_parser ( 'bnd' );
    my $bnd_last_flag = '';
    my $bnd_step = 1;
    my $bnd_step_flag = '';

    my ( @list, ) = helper_function();
```

```
    if ( !Exists ( $top_bnd ) ) {
        $top_bnd = $mw -> Toplevel( -title => 'Bending angles', );
        $top_bnd -> geometry("$toplevel_position");
        $top_bnd -> protocol( 'WM_DELETE_WINDOW' => sub { $top_bnd -> withdraw }, );

        my $frame_bnd0 = $top_bnd -> Frame() -> pack;
        my $frame_bnd1 = $top_bnd -> Frame() -> pack;

        $frame_bnd0 -> Label( -text => 'Define atoms by selecting resiue number, chain identifier
and atom type', ) -> pack;

        $frame_bnd1 -> Label( -text => 'Atom 1 :', ) -> grid( -row => 2, -column => 1, );
        our $bend_list1 = $frame_bnd1 -> BrowseEntry( -variable => \$bnd_atom1,
                                                      -listwidth => 50,
                                                      -autolistwidth => 0,
                                                      -browsecmd => sub {
            my $selection = our $bend_list1 -> get( 'active' );

            if ( $selection =~ /(\d+)$/ ) {
                $bnd_atom1 = $1;
            }
        }, ) -> grid( -row => 2, -column => 2, );

        $frame_bnd1 -> Label( -text => 'Atom 2 :', ) -> grid( -row => 2, -column => 3, );
        our $bend_list2 = $frame_bnd1 -> BrowseEntry( -variable => \$bnd_atom2,
                                                      -listwidth => 50,
                                                      -autolistwidth => 0,
                                                      -browsecmd => sub {
            my $selection = our $bend_list2 -> get( 'active' );

            if ( $selection =~ /(\d+)$/ ) {
                $bnd_atom2 = $1;
            }
        }, ) -> grid( -row => 2, -column => 4, );

        $frame_bnd1 -> Label( -text => 'Atom 3 :', ) -> grid( -row => 3, -column => 1, );
        our $bend_list3 = $frame_bnd1 -> BrowseEntry( -variable => \$bnd_atom3,
                                                      -listwidth => 50,
                                                      -autolistwidth => 0,
                                                      -browsecmd => sub {
            my $selection = our $bend_list3 -> get( 'active' );

            if ( $selection =~ /(\d+)$/ ) {
                $bnd_atom3 = $1;
            }
        }, ) -> grid( -row => 3, -column => 2, );

        $bend_list1 -> insert( 'end', @list, );
        $bend_list2 -> insert( 'end', @list, );
        $bend_list3 -> insert( 'end', @list, );

        our $frame_bnd2 = $top_bnd -> Frame() -> pack( -fill => 'x', );
        my $frame_bnd3 = $top_bnd -> Frame() -> pack( -fill => 'x', );
        my $frame_bnd4 = $top_bnd -> Frame() -> pack( -fill => 'x', );
        my $frame_bnd6 = $top_bnd -> Frame() -> pack( -fill => 'x', );
        my $frame_bnd7 = $top_bnd -> Frame( -borderwidth => 2, -relief => 'groove', ) -> pack(
-expand => 1, -fill => 'x', );

        radiobuttons ( $frame_bnd2 );
        checkbuttons ( $frame_bnd3 );
        otherbuttons ( $frame_bnd4 );

        $frame_bnd6 -> Label( -text => "\nVarious options", -font => $font_20, ) -> pack;

        $frame_bnd7 -> Label( -text => "First frame to use :", ) -> grid( -row => 1, -column => 1,
-sticky => 'w', );
        $frame_bnd7 -> Entry( -textvariable => \$bnd_first, ) -> grid( -row => 1, -column => 2, );
        $frame_bnd7 -> Label( -text => "Last frame to use :", ) -> grid( -row => 2, -column => 1,
-sticky => 'w', );
        $frame_bnd7 -> Entry( -textvariable => \$bnd_last, ) -> grid( -row => 2, -column => 2, );
        $frame_bnd7 -> Label( -text => "Stride ( step ) between frames :" . ' ' x 27, ) -> grid(
-row => 3, -column => 1, -sticky => 'w', );
        $frame_bnd7 -> Entry( -textvariable => \$bnd_step, ) -> grid( -row => 3, -column => 2, );
```

```
        my $frame_bnd5 = $top_bnd -> Frame() -> pack( -expand => 0, );

        $frame_bnd5 -> Button( -text => 'Return',
                               -command => [ $top_bnd => 'withdraw' ], )
                               -> pack( -side => 'left', );

        $frame_bnd5 -> Button( -text => 'Run',
                               -command => sub {
        $bnd_first_flag = ( ( $bnd_first != 1 ) ? " -first $bnd_first" : '' );
        $bnd_last_flag = ( ( $bnd_last != $header ) ? " -last $bnd_last" : '' );
        $bnd_step_flag = ( ( $bnd_step != 1 ) ? " -step $bnd_step" : '' );

        $top_bnd -> destroy;

        $seg_id_flag = '' if $seg_id_flag;

        foreach ( @seg_ids ) {
            if ( defined ( $_ ) ) {
                $seg_id_flag = $seg_id_flag . $_;
            }
        }

        if ( $res_id_flag ) {
             $flag  =  "  -v  -bend  $bnd_atom1  $bnd_atom2  $bnd_atom3  $atm_id_flag  $custom_id_flag
$res_id_flag $bnd_first_flag $bnd_last_flag $bnd_step_flag";
        }
        elsif ( $seg_id_flag ) {
             $flag  =  "  -v  -bend  $bnd_atom1  $bnd_atom2  $bnd_atom3  $atm_id_flag  $custom_id_flag
$seg_id_flag $bnd_first_flag $bnd_last_flag $bnd_step_flag";
        }
        else {
             $flag  =  "  -v  -bend  $bnd_atom1  $bnd_atom2  $bnd_atom3  $atm_id_flag  $custom_id_flag
$bnd_first_flag $bnd_last_flag $bnd_step_flag";
        }

        create_dir();

        $text -> insert( 'end', "\n\nNow calculating bend angles.\n\n", 'cyan', );
        $text -> see( 'end', );
        $mw -> update;

        carma();

        if ( $all_done ) {
            if ( $active_dcd =~ /(.+).dcd/ ) {
                mv ( 'carma.bendangles', "bendangles.$1.dat" );
            }

            $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n", 'valid' );
            $text -> see( 'end', );
            $image_menu -> configure( -state => 'normal', );
        }
        else {
             $text -> insert( 'end', "\nSomething went wrong. For details check last_carma_run.log
located in :\n", 'error', );
            $text -> insert( 'end', getcwd . "\n", 'info', );
            $text -> see( 'end', );
        }
        }, )-> pack( -side => 'right', );
    }
    else {
        $top_bnd -> deiconify;
        $top_bnd -> raise;
    }
}

#############################################################################################
###   Draw the window for torsion angles                                                  ###
#############################################################################################

sub tor_window {
    my $tor_atom1 = '';
    my $tor_atom2 = '';
```

```
    my $tor_atom3 = '';
    my $tor_atom4 = '';
    my $top_tor;
    my $tor_first = 1;
    my $tor_first_flag = '';
    my $tor_last = dcd_header_parser ( 'bnd' );
    my $tor_last_flag = '';
    my $tor_step = 1;
    my $tor_step_flag = '';

    my @list = helper_function();

    if ( !Exists ( $top_tor ) ) {
        $top_tor = $mw -> Toplevel( -title => 'Torsion angles', );
        $top_tor -> geometry("$toplevel_position");
        $top_tor -> protocol( 'WM_DELETE_WINDOW' => sub { $top_tor -> withdraw }, );

        my $frame_tor0 = $top_tor -> Frame() -> pack;
        my $frame_tor1 = $top_tor -> Frame() -> pack;

        $frame_tor0 -> Label( -text => 'Define atoms by selecting resiue number, chain identifier
and atom type', ) -> pack;

        $frame_tor1 -> Label( -text => 'Atom 1 :', ) -> grid( -row => 2, -column => 1, );
        our $tors_list1 = $frame_tor1 -> BrowseEntry( -variable => \$tor_atom1,
                                                      -listwidth => 50,
                                                      -autolistwidth => 0,
                                                      -browsecmd => sub {
            my $selection = our $tors_list1 -> get( 'active' );

            if ( $selection =~ /(\d+)$/ ) {
                $tor_atom1 = $1;
            }
        }, ) -> grid( -row => 2, -column => 2, );

        $frame_tor1 -> Label( -text => 'Atom 2 :', ) -> grid( -row => 2, -column => 3, );
        our $tors_list2 = $frame_tor1 -> BrowseEntry( -variable => \$tor_atom2,
                                                      -listwidth => 50,
                                                      -autolistwidth => 0,
                                                      -browsecmd => sub {
            my $selection = our $tors_list2 -> get( 'active' );

            if ( $selection =~ /(\d+)$/ ) {
                $tor_atom2 = $1;
            }
        }, ) -> grid( -row => 2, -column => 4, );

        $frame_tor1 -> Label( -text => 'Atom 3 :', ) -> grid( -row => 3, -column => 1, );
        our $tors_list3 = $frame_tor1 -> BrowseEntry( -variable => \$tor_atom3,
                                                      -listwidth => 50,
                                                      -autolistwidth => 0,
                                                      -browsecmd => sub {
            my $selection = our $tors_list3 -> get( 'active' );

            if ( $selection =~ /(\d+)$/ ) {
                $tor_atom3 = $1;
            }
        }, ) -> grid( -row => 3, -column => 2, );

        $frame_tor1 -> Label( -text => 'Atom 4 :', ) -> grid( -row => 3, -column => 3, );
        our $tors_list4 = $frame_tor1 -> BrowseEntry( -variable => \$tor_atom4,
                                                      -listwidth => 50,
                                                      -autolistwidth => 0,
                                                      -browsecmd => sub {
            my $selection = our $tors_list4 -> get( 'active' );

            if ( $selection =~ /(\d+)$/ ) {
                $tor_atom4 = $1;
            }
        }, ) -> grid( -row => 3, -column => 4, );

        $tors_list1 -> insert( 'end', @list, );
        $tors_list2 -> insert( 'end', @list, );
        $tors_list3 -> insert( 'end', @list, );
```

```
        $tors_list4 -> insert( 'end', @list, );

        our $frame_tor2 = $top_tor -> Frame() -> pack( -fill => 'x', );
        my $frame_tor3 = $top_tor -> Frame() -> pack( -fill => 'x', );
        my $frame_tor4 = $top_tor -> Frame() -> pack( -fill => 'x', );
        my $frame_tor6 = $top_tor -> Frame() -> pack( -fill => 'x', );
         my $frame_tor7 = $top_tor -> Frame( -borderwidth => 2, -relief => 'groove', ) -> pack(
-expand => 1, -fill => 'x', );

        radiobuttons ( $frame_tor2 );
        checkbuttons ( $frame_tor3 );
        otherbuttons ( $frame_tor4 );

        $frame_tor6 -> Label( -text => "\nVarious options", -font => $font_20, ) -> pack;

        $frame_tor7 -> Label( -text => "First frame to use :", ) -> grid( -row => 1, -column => 1,
-sticky => 'w', );
        $frame_tor7 -> Entry( -textvariable => \$tor_first, ) -> grid( -row => 1, -column => 2, );
        $frame_tor7 -> Label( -text => "Last frame to use :", ) -> grid( -row => 2, -column => 1,
-sticky => 'w', );
        $frame_tor7 -> Entry( -textvariable => \$tor_last, ) -> grid( -row => 2, -column => 2, );
        $frame_tor7 -> Label( -text => "Stride ( step ) between frames :" . ' ' x 27, ) -> grid(
-row => 3, -column => 1, -sticky => 'w', );
        $frame_tor7 -> Entry( -textvariable => \$tor_step, ) -> grid( -row => 3, -column => 2, );

        my $frame_tor5 = $top_tor -> Frame() -> pack( -expand => 0, );

        $frame_tor5 -> Button( -text => 'Return',
                               -command => [ $top_tor => 'withdraw' ], )
                               -> pack( -side => 'left', );

        $frame_tor5 -> Button( -text => 'Run',
                               -command => sub {
        $tor_first_flag = ( ( $tor_first != 1 ) ? " -first $tor_first" : '' );
        $tor_last_flag = ( ( $tor_last != $header ) ? " -last $tor_last" : '' );
        $tor_step_flag = ( ( $tor_step != 1 ) ? " -step $tor_step" : '' );

        $top_tor -> destroy;

        $seg_id_flag = '' if $seg_id_flag;

        foreach ( @seg_ids ) {
            if ( defined ( $_ ) ) {
                $seg_id_flag = $seg_id_flag . $_;
            }
        }

        if ( $res_id_flag ) {
             $flag  =  "  -v  -torsion  $tor_atom1  $tor_atom2  $tor_atom3  $tor_atom4  $atm_id_flag
$res id flag $custom_id_flag $tor_first_flag $tor_last_flag $tor_step_flag";
        }
        elsif ( $seg_id_flag ) {
             $flag  =  "  -v  -torsion  $tor_atom1  $tor_atom2  $tor_atom3  $tor_atom4  $atm_id_flag
$custom_id_flag $seg_id_flag $tor_first_flag $tor_last_flag $tor_step_flag";
        }
        else {
             $flag  =  "  -v  -torsion  $tor_atom1  $tor_atom2  $tor_atom3  $tor_atom4  $atm_id_flag
$custom_id_flag $tor_first_flag $tor_last_flag $tor_step_flag";
        }

        create_dir();

        $text -> insert( 'end', "\n\nNow calculating torsion angles.\n\n", 'cyan', );
        $text -> see( 'end', );
        $mw -> update;

        carma();

        if ( $all_done ) {
            if ( $active_dcd =~ /(.+).dcd/ ) {
                mv ( 'carma.torsions', "torsions.$1.dat" );
            }

            $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n", 'valid' );
```

```
            $text -> see( 'end', );
            $image_menu -> configure( -state => 'normal', );
        }
        else {
            $text -> insert( 'end', "\nSomething went wrong. For details check last_carma_run.log
located in :\n", 'error', );
            $text -> insert( 'end', getcwd . "\n", 'info', );
            $text -> see( 'end', );
        }
        }, )-> pack( -side => 'right', );
    }
    else {
        $top_tor -> deiconify;
        $top_tor -> raise;
    }
}

#############################################################################################
###   Helper function for the distances, bending and torsion angles functions             ###
#############################################################################################

sub helper_function {
    create_dir();

    my ( @atom_data, );

    if ( $linux or $mac ) {
        `carma -w -last 1 $atm_id_flag $custom_id_flag $active_psf $active_dcd`;
    }
    else {
        `carma.exe -w -last 1 $atm_id_flag $custom_id_flag $active_psf $active_dcd`;
    }

    unlink ( "$active_dcd.0000001.dat", "$active_dcd.0000001.ps" );

     open  IN,  '<',  "carma.selected_atoms.psf"  or  die  "Cannot  open  carma.selected_atoms.psf  for
reading: $!";

    my $i = 0;
    my $num_of_atoms = 0;
    while ( <IN> ) {
        if ( /(\d+) !NATOM/ ) {
            $num_of_atoms = $1;
            last;
        }
    }

    while ( <IN> ) {
        my ( $segid, $atmid, $resid, $atmnum, $resname, );

        $resid = substr $_, 14, 4;
        $segid = substr $_, 9, 1;
        $resname = substr $_, 19, 3;
        $atmid = substr $_, 24, 4;
        $atmnum = substr $_, ( 8 - length $num_of_atoms ), length $num_of_atoms;

         $atom_data[$i] = sprintf ( "%04d %s %3s % 3s             %7d", $resid, $segid, $resname,
$atmid, $atmnum, );
        $i++;
    }

    close IN;

    return ( @atom_data, );
}

#############################################################################################
###   Draw the window for phi/psi dihedral angles                                         ###
#############################################################################################

sub phi_psi_window {
    my $top_phi_psi;
    my $phi_psi_first = 1;
    my $phi_psi_first_flag;
```

```
    my $phi_psi_last = dcd_header_parser( 'phi_psi' );
    my $phi_psi_last_flag;
    my $phi_psi_step = 1;
    my $phi_psi_step_flag;

    if ( !Exists ( $top_phi_psi ) ) {
        $top_phi_psi = $mw -> Toplevel( -title => 'phi/psi dihedral angles', );
        $top_phi_psi -> geometry("$toplevel_position");
        $top_phi_psi -> protocol( 'WM_DELETE_WINDOW' => sub { $top_phi_psi -> withdraw }, );

        $frame_phi_psi1 = $top_phi_psi -> Frame() -> pack( -fill => 'x', );
        my $frame_phi_psi4 = $top_phi_psi -> Frame() -> pack( -fill => 'x', );
        my $frame_phi_psi5 = $top_phi_psi -> Frame() -> pack( -fill => 'x', );
        my $frame_phi_psi2 = $top_phi_psi -> Frame( -borderwidth => 2, -relief => 'groove', ) ->
pack( -fill => 'x', );
        my $frame_phi_psi3 = $top_phi_psi -> Frame() -> pack( -expand => 0, );

        checkbuttons ( $frame_phi_psi1 );
        otherbuttons ( $frame_phi_psi4 );

        $frame_phi_psi5 -> Label( -text => "\nVarious Options", -font => $font_20, ) -> pack;

        $frame_phi_psi2 -> Label( -text => 'First frame to use: ', )
                        -> grid( -row => 1, -column => 1, -sticky => 'w', );
        $frame_phi_psi2 -> Entry( -textvariable => \$phi_psi_first, )
                        -> grid( -row => 1, -column => 2, );
        $frame_phi_psi2 -> Label( -text => 'Last frame to use: ', )
                        -> grid( -row => 2, -column => 1, -sticky => 'w', );
        $frame_phi_psi2 -> Entry( -textvariable => \$phi_psi_last, )
                        -> grid( -row => 2, -column => 2, );
        $frame_phi_psi2 -> Label( -text => 'Stride (step) between frames: ', )
                        -> grid( -row => 3, -column => 1, -sticky => 'w', );
        $frame_phi_psi2 -> Entry( -textvariable => \$phi_psi_step, )
                        -> grid( -row => 3, -column => 2, );

        $frame phi psi3 -> Button( -text => 'Return',
                         -command => [ $top_phi_psi => 'withdraw' ], )
                         -> pack( -side => 'left', );

        $phi_psi_run_button = $frame_phi_psi3 -> Button( -text => 'Run',
                                                         -state => 'disabled',
                                                         -command => sub {
            $phi_psi_first_flag = ( ( $phi_psi_first != 1 ) ? " -first $phi_psi_first" : '' );
            $phi_psi_last_flag = ( ( $phi_psi_last != $header ) ? " -last $phi_psi_last" : '' );
            $phi_psi_step_flag = ( ( $phi_psi_step != 1 ) ? " -step $phi_psi_step" : '' );

            $top_phi_psi -> withdraw;

            $seg_id_flag = '' if $seg_id_flag;

            foreach ( @seg_ids ) {
                if ( defined ( $_ ) ) {
                    $seg_id_flag = $seg_id_flag . $_;
                }
            }

            create_dir();

            our $temp_segid;
            if ( $res_id_flag ) {
                $temp_segid = 'Z';
            }
            elsif ( $seg_id_flag =~ /([A-Z])/ ) {
                $temp_segid = $1;
            }

            our ( $first_residue, $last_residue, $number_of_residues, $correction, ) = indeces (
"$temp_segid" );

            if ( not -f 'indeces.dat' ) {
                $text -> insert( 'end', 'No \'indeces.dat\' file present in the working directory.
Aborting.', 'error' );
                $text -> see( 'end', );
            }
```

```
            elsif ( -z 'indeces.dat' ) {
                $text -> insert( 'end', 'The file containing the indeces of the atoms to be used for
the calculation of' .
                                        ' the phi/psi angles seems to be empty. Maybe you need to
review the residue selection ?', 'error' );
                $text -> see( 'end', );
            }
            else {
                $flag = " -v -atmid ALLID $phi_psi_first_flag $phi_psi_last_flag $phi_psi_step_flag
-torsion indeces.dat";

                $text -> insert( 'end', "\n\nNow calculating phi/psi dihedral angles.\n\n", 'cyan',
);
                $text -> see( 'end', );
                $mw -> update;

                carma();

                if ( $all_done ) {
                    if ( $active_dcd =~ /(.+).dcd/ ) {
                        mv ( "carma.torsions", "phi_psi_dihedral_segid$temp_segid.dat" );
                    }

                     $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n",
'valid' );
                    $text -> see( 'end', );
                    $image_menu -> configure( -state => 'normal', );
                }
                else {
                     $text -> insert( 'end', "\nSomething went wrong. For details check
last_carma_run.log located in :\n", 'error', );
                    $text -> insert( 'end', getcwd . "\n", 'info', );
                    $text -> see( 'end', );
                }
            }
        }, )-> pack( -side => 'right', );

        if ( $active_run_buttons_qfract ) {
            $phi_psi_run_button -> configure( -state => 'normal', );
        }
        else {
            $phi_psi_run_button -> configure( -state => 'disabled', );
        }
    }
    else {
        $top_phi_psi -> deiconify;
        $top_phi_psi -> raise;
    }
}

#############################################################################################
### Get the atom indeces for calculating phi/psi angles                                   ###
### Contributed by N. M. Glykos                                                           ###
#############################################################################################

sub indeces {
    my $input = shift;

    my (
        $nofatm, $tot, @at,
        $at1_num, $at1_res, $at1_nam,
        $at2_num, $at2_res, $at2_nam,
        $at3_num, $at3_res, $at3_nam,
        $at4_num, $at4_res, $at4_nam,
        $is_C, $is_CA, $is_N,
        $index1, $index2, );

    open OUT, '+>', 'temp.dat' or die "cannot open temp.dat for writing : $!\n";
    open IN, '<', $active_psf or die "cannot open $active_psf for reading : $!\n";

    while ( <IN> ) {
        if ( /(\d+) !NATOM/ ) {
            $nofatm = $1;
            last;
```

```
        }
    }

    $tot = 0;
    for ( my $i = 0 ; $i < $nofatm  ; $i++ ) {
        my $line = <IN>;
        if ( $line =~ /(\d+)\s+$input\s+(\d+)\s+\w+\s+(\w+)/ ) {
            if ( $3 eq "N" || $3 eq "C" || $3 eq "CA" ) {
                $at[ $tot ] = $1 . " " . $2 . " " . $3;
                $tot++;
            }
        }
    }

    if ( $tot % 3 != 0 ) {
        my $response;
        if ( $linux or $mac ) {
            $response = $mw -> messageBox( -message => "Total number of C CA N atoms read is not a
multiple of 3. " .
                                                       "  Could  it  be  that  the  peptide  is  N/C
capped?. If so please" .
                                                       "  use  residue  selection  to  exclude  the
relevant segments from the analysis.",
                                           -icon => 'warning',
                                           -font => $font_12, );
        }
        else {
            $response = $mw -> messageBox( -message => "Total number of C CA N atoms read is not a
multiple of 3. " .
                                                       "  Could  it  be  that  the  peptide  is  N/C
capped?. If so please" .
                                                       "  use  residue  selection  to  exclude  the
relevant segments from the analysis.",
                                           -icon => 'warning', );
        }
    }
    elsif ( $tot == 0 ) {
        my $response;
        if ( $linux or $mac ) {
            $response = $mw -> messageBox( -message => "Total number of C CA N atoms read is 0.
Aborting",
                                           -icon => 'warning',
                                           -font => $font_12, );
        }
        else {
            $response = $mw -> messageBox( -message => "Total number of C CA N atoms read is 0.
Aborting",
                                           -icon => 'warning', );
        }
    }
    else {
        for ( my $i=0 ; $i < $tot ; $i += 3 ) {
            if ( $at[ $i ] =~ /(\d+) (\d+) (\w+)/ ) {
                $at1_num = $1;
                $at1_res = $2;
                $at1_nam = $3;
            }
            else {
                die "Something is wrong. Get help.\n";
            }

            if ( $at[ $i+1 ] =~ /(\d+) (\d+) (\w+)/ ) {
                $at2_num = $1;
                $at2_res = $2;
                $at2_nam = $3;
            }
            else {
                die "Something is wrong. Get help.\n";
            }

            if ( $at[ $i+2 ] =~ /(\d+) (\d+) (\w+)/ ) {
                $at3_num = $1;
                $at3_res = $2;
                $at3_nam = $3;
```

```
    }
    else {
        die "Something is wrong. Get help.\n";
    }

    if ( $at2_res ne $at1_res || $at2_res ne $at3_res || $at3_res ne $at1_res ) {
        die "Panic. Get help.\n";
    }

    if ( $at1_nam ne "N" || $at2_nam ne "CA" || $at3_nam ne "C" ) {

        if ( $at1_nam eq "N" ) {
            $is_N = 0;
        }
        if ( $at1_nam eq "CA" ) {
            $is_CA = 0;
        }
        if ( $at1_nam eq "C" ) {
            $is_C = 0;
        }

        if ( $at2_nam eq "N" ) {
            $is_N = 1;
        }
        if ( $at2_nam eq "CA" ) {
            $is_CA = 1;
        }
        if ( $at2_nam eq "C" ) {
            $is_C = 1;
        }

        if ( $at3_nam eq "N" ) {
            $is_N = 2;
        }
        if ( $at3_nam eq "CA" ) {
            $is_CA = 2;
        }
        if ( $at3_nam eq "C" ) {
            $is_C = 2;
        }

        my $N_line = $at[ $i + $is_N ];
        my $C_line = $at[ $i + $is_C ];
        my $CA_line = $at[ $i + $is_CA ];

        $at[ $i   ] = $N_line;
        $at[ $i+1 ] = $CA_line;
        $at[ $i+2 ] = $C_line;
    }
}

for ( my $i=0 ; $i <= ($tot / 3 - 2) ; $i++ ) {
    $index1 = $i * 3;
    $index2 = $i * 3 + 2;

    if ( $at[ $index1 ] =~ /(\d+) (\d+) (\w+)/ ) {
        $at1_num = $1;
        $at1_res = $2;
        $at1_nam = $3;
    }
    else {
        die "Something is wrong. Get help.\n";
    }

    if ( $at[ $index1+1 ] =~ /(\d+) (\d+) (\w+)/ ) {
        $at2_num = $1;
        $at2_res = $2;
        $at2_nam = $3;
    }
    else {
        die "Something is wrong. Get help.\n";
    }

    if ( $at[ $index1+2 ] =~ /(\d+) (\d+) (\w+)/ ) {
```

```
            $at3_num = $1;
            $at3_res = $2;
            $at3_nam = $3;
        }
        else {
            die "Something is wrong. Get help.\n";
        }

        if ( $at[ $index1+3 ] =~ /(\d+) (\d+) (\w+)/ ) {
            $at4_num = $1;
            $at4_res = $2;
            $at4_nam = $3;
        }
        else {
            die "Something is wrong. Get help.\n";
        }

        printf OUT "%6d %6d %6d %6d  # psi for residue %4d\n", $at1_num, $at2_num, $at3_num,
$at4_num, $at3_res;

        if ( $at[ $index2 ] =~ /(\d+) (\d+) (\w+)/ ) {
            $at1_num = $1;
            $at1_res = $2;
            $at1_nam = $3;
        }
        else {
            die "Something is wrong. Get help.\n";
        }
        if ( $at[ $index2+1 ] =~ /(\d+) (\d+) (\w+)/ ) {
            $at2_num = $1;
            $at2_res = $2;
            $at2_nam = $3;
        }
        else {
            die "Something is wrong. Get help.\n";
        }
        if ( $at[ $index2+2 ] =~ /(\d+) (\d+) (\w+)/ ) {
            $at3_num = $1;
            $at3_res = $2;
            $at3_nam = $3;
        }
        else {
            die "Something is wrong. Get help.\n";
        }
        if ( $at[ $index2+3 ] =~ /(\d+) (\d+) (\w+)/ ) {
            $at4_num = $1;
            $at4_res = $2;
            $at4_nam = $3;
        }
        else {
            die "Something is wrong. Get help.\n";
        }

        printf OUT "%6d %6d %6d %6d  # phi for residue %4d\n", $at1_num, $at2_num, $at3_num,
$at4_num, $at4_res;
    }
    seek OUT, 0, 0;

    open OUT2, '+>', 'indeces.dat' or die "Cannot open indeces.dat for writing : $!\n";

    while ( <OUT> ) {
        print OUT2 $_ unless ( $. == 1 or eof );
    }
    unlink ( 'temp.dat', );

    seek OUT2, 0, 0;

    my @resid_span;
    my $i = 0;
    while ( <OUT2> ) {
        if ( /p.i for residue\s+(\d+)/ ) {
            $resid_span[$i] = $1;
            $i++;
        }
```

```
        }

        my $first_residue = $resid_span[0];
        my $last_residue = $resid_span[scalar @resid_span - 1];
        my $num_residues = ( scalar @resid_span / 2 );
        my $correction = 0;
        if ( $first_residue != 2 ) {
            $correction = $first_residue - 2;
        }

        close OUT2;
        close IN;
        close OUT;

        return ( $first_residue, $last_residue, $num_residues, $correction, );
    }
}

#############################################################################################
###   Draw the window for ion and water distribution maps                                 ###
#############################################################################################

sub map_window {
    my $map_ang_x = '';
    my $map_ort_x = '';
    my $map_ang_y = '';
    my $map_ort_y = '';
    my $map_ang_z = '';
    my $map_ort_z = '';
    my $map_grid_spacing = '';
    my $top_map;

    if ( -e "fit.index" ) {
        if ( !Exists ( $top_map ) ) {
            $top_map = $mw -> Toplevel( -title => 'Ion and water distribution maps', );
            $top_map -> geometry("$toplevel position");
            $top_map -> protocol( 'WM_DELETE_WINDOW' => sub { $top_map -> withdraw }, );

            my $frame_map1 = $top_map -> Frame( -borderwidth => 3,
                                                -relief => 'groove',)
                                                -> pack( -fill => 'x', );

            $frame_map1 -> Label( -text => 'Limits', ) -> pack;

            my $frame_map2 = $top_map -> Frame() -> pack( -expand => 0, );

            $frame_map2 -> Label( -text => 'Angstrom' )
                                -> grid( -row => 2, -column => 1, );
            $frame_map2 -> Label( -text => 'Orthogonal' )
                                -> grid( -row => 3, -column => 1, );
            $frame_map2 -> Label( -text => 'X' )
                                -> grid( -row => 1, -column => 2, -columnspan => 2, );
            $frame_map2 -> Entry( -textvariable => \$map_ang_x )
                                -> grid( -row => 2, -column => 2, -columnspan => 2, );
            $frame_map2 -> Entry( -textvariable => \$map_ort_x )
                                -> grid( -row => 3, -column => 2, -columnspan => 2, );
            $frame_map2 -> Label( -text => 'Y' )
                                -> grid( -row => 1, -column => 4, -columnspan => 2, );
            $frame_map2 -> Entry( -textvariable => \$map_ang_y )
                                -> grid( -row => 2, -column => 4, -columnspan => 2, );
            $frame_map2 -> Entry( -textvariable => \$map_ort_y )
                                -> grid( -row => 3, -column => 4, -columnspan => 2, );
            $frame_map2 -> Label( -text => 'Z' )
                                -> grid( -row => 1, -column => 6, -columnspan => 2, );
            $frame_map2 -> Entry( -textvariable => \$map_ang_z )
                                -> grid( -row => 2, -column => 6, -columnspan => 2, );
            $frame_map2 -> Entry( -textvariable => \$map_ort_z )
                                -> grid( -row => 3, -column => 6, -columnspan => 2, );
            $frame_map2 -> Label( -text => 'Grid spacing' )
                                -> grid( -row => 1, -column => 8, );
            $frame_map2 -> Entry( -textvariable => \$map_grid_spacing )
                                -> grid( -row => 2, -column => 8, );

            my $frame_map3 = $top_map -> Frame() -> pack( -fill => 'x', );
```

```
            my $frame_map4 = $top_map -> Frame() -> pack( -fill => 'x', );
            my $frame_map5 = $top_map -> Frame() -> pack( -fill => 'x', );

            radiobuttons ( $frame_map3 );
            checkbuttons ( $frame_map4 );
            otherbuttons ( $frame_map5 );

            my $frame_map6 = $top_map -> Frame() -> pack( -expand => 0, );

            $frame_map6 -> Button( -text => 'Return',
                                   -command => [ $top_map => 'withdraw' ], )
                                   -> pack( -side => 'left', );

            $frame_map6 -> Button( -text => 'Run',
                                   -command => sub {
            $top_map -> destroy;

            $seg_id_flag = '' if $seg_id_flag;

            foreach ( @seg_ids ) {
                if ( defined ( $_ ) ) {
                    $seg_id_flag = $seg_id_flag . $_;
                }
            }

            if ( $seg_id_flag ) {
                 $flag  =  "  -v  -w  -map  $map_ang_x  $map_ort_x  $map_ang_y  $map_ort_y  $map_ang_z
$map_ort_z $map_grid_spacing $atm_id_flag $custom_id_flag $seg_id_flag $res_id_flag";
            }
            else {
                 $flag  =  "  -v  -w  -map  $map_ang_x  $map_ort_x  $map_ang_y  $map_ort_y  $map_ang_z
$map_ort_z $map_grid_spacing $atm_id_flag $custom_id_flag $res_id_flag";
            }

            &create_dir;

            $text -> insert( 'end', "\n\nNow mapping water and ions.\n\n", 'cyan', );
            $text -> see( 'end', );
            $mw -> update;

            &carma;

            if ( $all_done ) {
                $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n", 'valid' );
                $text -> see( 'end', );
                $image_menu -> configure( -state => 'normal', );
            }
            else {
                 $text   ->   insert(   'end',   "\nSomething   went   wrong.   For   details   check
last carma run.log located in :\n", 'error', );
                $text -> insert( 'end', getcwd . "\n", 'info', );
                $text -> see( 'end', );
            }
            }, )-> pack( -side => 'right', );
        }
        else {
            $top_map -> deiconify;
            $top_map -> raise;
        }
    }
    else {
        my $response;
        if ( $linux or $mac ) {
            $response = $mw -> messageBox( -message => "No \'fit.index\' file found in the working
directory. Would you like to load one?",
                                           -type => 'yesno',
                                           -icon => 'question',
                                           -font => "$font_12", );
        }
        else {
            $response = $mw -> messageBox( -message => "No \'fit.index\' file found in the working
directory. Would you like to load one?",
                                           -type => 'yesno',
                                           -icon => 'question', );
```

```
        }

        my $filetype = undef;
        if ( $response eq 'Yes' ) {
            if ( $^O eq 'linux' ) {
                $filetype = [ ['Fitting index files', '.index'] ];
            }
            else {
                $filetype = [ ['Fitting index files', '.index'], ['Fitting index files',  '.index']
];
            }

            my $file = $mw -> getOpenFile( -filetypes => $filetype, );

            if ( $file =~ /\w*\.index/ ) {
                $file = abs_path ( $file );
                create_dir();

                if ( $linux || $mac ) {
                    `cp $file .`;
                }
                else {
                    `copy $file .`;
                }
            }
        }
    }
}

###############################################################################################################
###   Draw the window for the surface area analysis                                                         ###
###############################################################################################################

sub sur_window {
    my $top_sur;
    my $sur_plot = '';
    my $sur_first = 1;
    my $sur_last = dcd_header_parser( "rms" );
    my $sur_step = 1;
    my $sur_first_flag;
    my $sur_last_flag;
    my $sur_step_flag;

    if ( !Exists ( $top_sur ) ) {
        $top_sur = $mw -> Toplevel( -title => 'Surface area', );
        $top_sur -> geometry("$toplevel_position");
        $top_sur -> protocol( 'WM_DELETE_WINDOW' => sub { $top_sur -> withdraw }, );

        our $frame_sur1 = $top_sur -> Frame() -> pack( -fill => 'x', );
        $frame_sur2 = $top_sur -> Frame() -> pack( -fill => 'x', );
        my $frame_sur3 = $top_sur -> Frame() -> pack( -fill => 'x', );
        my $frame_sur6 = $top_sur -> Frame() -> pack( -fill => 'x', );
        my $frame_sur4 = $top_sur -> Frame( -borderwidth => 2, -relief => 'groove', ) -> pack( -fill
=> 'x', );
        my $frame_sur5 = $top_sur -> Frame() -> pack( -expand => 0, );

        radiobuttons ( $frame_sur1 );
        checkbuttons ( $frame_sur2 );
        otherbuttons ( $frame_sur3 );

        $frame_sur6 -> Label( -text => "\nVarious options", -font => $font_20, ) -> pack;

        $frame_sur4 -> Label( -text => 'First frame to use: ', )
                    -> grid( -row => 1, -column => 1, -sticky => 'w', );
        $frame_sur4 -> Entry( -textvariable => \$sur_first, )
                    -> grid( -row => 1, -column => 2, );
        $frame_sur4 -> Label( -text => 'Last frame to use: ', )
                    -> grid( -row => 2, -column => 1, -sticky => 'w', );
        $frame_sur4 -> Entry( -textvariable => \$sur_last, )
                    -> grid( -row => 2, -column => 2, );
        $frame_sur4 -> Label( -text => 'Stride (step) between frames: ', )
                    -> grid( -row => 3, -column => 1, -sticky => 'w', );
        $frame_sur4 -> Entry( -textvariable => \$sur_step, )
                    -> grid( -row => 3, -column => 2, );
```

```
        $frame_sur5 -> Button( -text => 'Return',
                               -command => [ $top_sur => 'withdraw' ], )
                               -> pack( -side => 'left', );

        $surf_run_button = $frame_sur5 -> Button( -text => 'Run',
                                                  -state => 'disabled',
                                                  -command => sub {
        $sur_first_flag = ( ( $sur_first != 1 ) ? " -first $sur_first" : '' );
        $sur_last_flag = ( ( $sur_last != $header ) ? " -last $sur_last" : '' );
        $sur_step_flag = ( ( $sur_step != 1 ) ? " -step $sur_step" : '' );

        $top_sur -> destroy;

        $seg_id_flag = '' if $seg_id_flag;

        foreach ( @seg_ids ) {
            if ( defined ( $_ ) ) {
                $seg_id_flag = $seg_id_flag . $_;
            }
        }

        if ( $res_id_flag ) {
             $flag  =  "  -v  -surf  $sur_first_flag  $sur_last_flag  $sur_step_flag  $atm_id_flag
$custom_id_flag $res_id_flag";
        }
        elsif ( $seg_id_flag ) {
             $flag  =  "  -v  -surf  $sur_first_flag  $sur_last_flag  $sur_step_flag  $atm_id_flag
$custom_id_flag $seg_id_flag";
        }
        else {
             $flag  =  "  -v  -surf  $sur_first_flag  $sur_last_flag  $sur_step_flag  $atm_id_flag
$custom_id_flag";
        }

        create_dir();

        $text -> insert( 'end', "\n\nNow calculating surface area.\n\n", 'cyan', );
        $text -> see( 'end', );
        $mw -> update;

        carma();

        if ( $all_done ) {
            if ( $active_dcd =~ /(.+).dcd/ ) {
                mv ( 'carma.surface.dat', "surface.$1.dat" );
            }

            $text -> insert( 'end', "\nCalculation finished. Use \"View Results\"\n", 'valid' );
            $text -> see( 'end', );
            $image_menu -> configure( -state => 'normal', );

            if ( $sur_plot ) {
                plot ( 'carma.surface.dat' );
                $sur_plot = 0;
            }
        }
        else {
            $text -> insert( 'end', "\nSomething went wrong. For details check last_carma_run.log
located in :\n", 'error', );
            $text -> insert( 'end', getcwd . "\n", 'info', );
            $text -> see( 'end', );
        }
        }, )-> pack( -side => 'right', );

        if ( $active_run_buttons ) {
            $surf_run_button -> configure( -state => 'normal', );
        }
        else {
            $surf_run_button -> configure( -state => 'disabled', );
        }
    }
    else {
        $top_sur -> deiconify;
```

```
        $top_sur -> raise;
    }
}

#############################################################################################################
###   Draw the window for fitting                                                                         ###
#############################################################################################################

sub fit_window {

    my $no_fit = '';
    my $ref = 1;
    my $ref_flag = '';
    my $fit_plot = '';
    my $index = '';
    my $index_confirmation = '';
    my $fit_run_button;
    my $resid_active = 0;

    my $index_atm_id;
    my $index_atm_id_flag;
    my $index_seg_id_flag;
    my $index_res_id_flag;
    my @index_seg_ids;

    our $top_fit;

    if ( !Exists ( $top_fit ) ) {
        unless ( $dcd_count > -1 ) {
            $dcd_count = -1;
        }

        $top_fit = $mw -> Toplevel( -title => 'Fitting', );
        $top_fit -> geometry("$toplevel_position");
        $top_fit -> protocol( 'WM_DELETE_WINDOW' => sub { #close STDERR;
                                                          $top_fit -> withdraw });

        my $frame_fit1 = $top_fit -> Frame() -> pack( -fill => 'x', );
        my $frame_fit2 = $top_fit -> Frame() -> pack( -fill => 'x', );
        my $frame_fit3 = $top_fit -> Frame() -> pack( -fill => 'x', );
        my $frame_fit5 = $top_fit -> Frame() -> pack( -fill => 'x', );

        $index_bar_count = 0;
        $frame_fit6[$index_bar_count] = $top_fit -> Frame( -borderwidth => 2, -relief => 'groove', )
-> pack( -fill => 'x', -expand => 1, );
        $frame_fit_bars[$index_bar_count] = $top_fit -> Frame();

        my $frame_fit6a = $frame_fit6[0] -> Frame() -> grid( -row => 0, -column => 2, -columnspan =>
6, );

         $frame_fit6a  -> Label( -text  =>  'Atom  Selection' )  -> grid( -row  =>  1,  -column  =>  0,
-columnspan => 5, );

        my @index_radiobuttons = ( 'CA', 'Backbone', 'Heavy', 'All atoms', 'Custom selection', );
        my @index_radio_b;

        for my $i ( 0 .. $#index_radiobuttons ) {
            $index_radio_b[$i] = $frame_fit6a -> Radiobutton( -text => $index_radiobuttons[$i],
                                                              -value => $index_radiobuttons[$i],
                                                              -variable => \$index_atm_id,
                                                              -command => sub {
                if ( $index_atm_id eq 'CA' ) {
                    $index_atm_id_flag = "";
                    $custom_selection = 0;
                }
                elsif ( $index_atm_id eq 'Backbone' ) {
                    $index_atm_id_flag = " -atmid C -atmid CA -atmid N -atmid O";
                    $custom_selection = 0;
                }
                elsif ( $index_atm_id eq 'Heavy' ) {
                    $index_atm_id_flag = " -atmid HEAVY";
                    $custom_selection = 0;
                }
                elsif ( $index_atm_id eq 'All atoms' ) {
```

```
                $index_atm_id_flag = " -atmid ALLID";
                $custom_selection = 0;
            }

            $index = ' -index';
            $index_confirmation = 1;

            $fit_run_button -> configure( -state => 'normal', );
        }, );

        $index_radio_b[$i] -> grid( -row => 2, -column => $i, );
    }

    $index_radio_b[4] -> configure( -command => \&raise_custom_window, );

    $frame_fit6a -> Label( -text => "\nChain Selection", ) -> grid( -row => 3, -column => 0,
-columnspan => 5, );

    my @index_check_b;

    my $index_count = 0;

    for my $j ( 0 .. $#unique_chain_ids ) {
        $index_check_b[$j] = $frame_fit6a -> Checkbutton( -text => $unique_chain_ids[$j],
                                                          -variable => \$index_seg_ids[$j],
                                                          -offvalue => '',
                                                          -onvalue => "$unique_chain_ids[$j]",
                                                          -command => sub {            }, );

        $index_check_b[$j] -> grid( -row => 4, -column => $j, );
    }

    my @index_otherbuttons = ( 'All', 'Change', );
    my @index_other;

    $frame_fit6a -> Label( -text => "Residue Selection\n", ) -> grid( -row => 5, -column => 0,
-columnspan => 5, );

    for my $k ( 0 .. $#index_otherbuttons ) {
        $index_other[$k] = $frame_fit6a -> Radiobutton( -text => "$index_otherbuttons[$k]",
                                                        -value => $index_otherbuttons[$k],
                                                        -variable => \$index_res_id_flag, );

        $index_other[$k] -> grid( -row => 6, -column => $k, );
    }

    $frame_fit_bars[0] -> Button( -text => 'Confirm',
                                  -width => 10,
                                  -command => sub {
        my $i;
        my $check = 0;
        for ( $i = 1 ; $i <= $index_bar_count ; $i++ ) {
            if ( $lower_fit_limit[$i-1] and $upper_fit_limit[$i-1] and $fit_drop_value[$i-1] ) {
                $check++;
            }
        }

        if ( $check == $i - 1 ) {
            create_dir();

            if ( $linux || $mac ) {
                $seg_id_flag = '' if $seg_id_flag;

                foreach ( @seg_ids ) {
                    if ( defined ( $_ ) ) {
                        $seg_id_flag = $seg_id_flag . $_;
                    }
                }

                if ( $seg_id_flag ) {
                    `carma -v -w -last 1 $atm_id_flag $res_id_flag $custom_id_flag $seg_id_flag
$active_psf $active_dcd`;
                }
                else {
```

```
                    `carma -v -w -last 1 $atm_id_flag $custom_id_flag $res_id_flag $active_psf
$active_dcd`;
                }
            }
            else {
                $seg_id_flag = '' if $seg_id_flag;

                foreach ( @seg_ids ) {
                    if ( defined ( $_ ) ) {
                        $seg_id_flag = $seg_id_flag . $_;
                    }
                }

                if ( $seg_id_flag ) {
                    `carma.exe -v -w -last 1 $atm_id_flag $res_id_flag $custom_id_flag
$seg_id_flag $active_psf $active_dcd"`;
                }
                else {
                    `carma.exe -v -w -last 1 $atm_id_flag $res_id_flag $custom_id_flag
$active_psf $active_dcd`;
                }
            }

            unlink ( "$dcd_name.dcd.0000001.dat" );
            unlink ( "$dcd_name.dcd.0000001.ps" );

            $index_seg_id_flag = '';

            foreach ( @index_seg_ids ) {
                if ( defined ( $_ ) ) {
                    $index_seg_id_flag = $index_seg_id_flag . ' ' . $_;
                }
            }

            if ( $index_seg_id_flag ) {
                create_fit_index( $index_atm_id, 'from_confirm', $index_seg_id_flag, );
            }
            else {
                create_fit_index( $index_atm_id, 'from_confirm', );
            }

            $index = ' -index';
            $index_confirmation = 1;
            $resid_active = 1;

            $fit_run_button -> configure( -state => 'normal', );
        }
        else {
            if ( $linux or $mac ) {
                $top_fit -> messageBox( -message => 'All of the boxes must be filled in order to
submit a residue selection',
                                        -icon => 'warning',
                                        -font => "$font_12", );
            }
            else {
                $top_fit -> messageBox( -message => 'All of the boxes must be filled in order to
submit a residue selection',
                                        -icon => 'warning', );
            }
        }
    }, ) -> grid( -row => 1, -column => 7, );

    $frame_fit6[0] -> packForget;

    $frame_fit_bars[$index_bar_count] -> Button( -text => 'Add..',
                                                 -width => 10,
                                                 -command => sub {
        $frame_fit_bars[$index_bar_count] = $top_fit -> Frame() -> pack( -anchor => 'w', );#
unless ( $index_bar_count == 0 );

        add_index_bar();

        if ( $index_bar_count >= 1 ) {
            $frame_fit_bars[$index_bar_count] -> Button( -text => 'Remove',
```

```
                                                    -width => 10,
                                                    -command => sub {
                    $frame_fit_bars[$index_bar_count-1] -> destroy;
                    $index_bar_count--;
                }, ) -> grid( -row => "$index_row", -column => "$index_column" + 5, );
            }

            $index_bar_count++;
        }, ) -> grid( -row => 1, -column => 6, );

        radiobuttons ( $frame_fit1 );
        checkbuttons ( $frame_fit2 );
        otherbuttons ( $frame_fit3 );

        my $frame_fit2a = $top_fit -> Frame() -> pack( -fill => 'x', );
        my $frame_fit4 = $top_fit -> Frame() -> pack( -fill => 'x', );

        $frame_fit2a -> Label( -text => "\nOptional settings", -font => "$font_20", ) -> pack;

        my $index_bool;
        my $index_button = $frame_fit4 -> Checkbutton( -text => 'Use a subset of the residues for
the fitting',
                                                    -variable => \$index_bool,
                                                    -command => sub {
            if ( $fit_run_button -> cget( -state, ) eq 'normal' and not $index_confirmation ) {
                $fit_run_button -> configure( -state => 'disabled', );
            }
            elsif ( $fit_run_button -> cget( -state, ) eq 'disabled' ) {
                $fit_run_button -> configure( -state => 'normal', );
            }

            if ( $index_bool ) {
                $frame_fit6[0] -> pack;
            }
            else {
                for ( my $i = $index_bar_count ; $i >= 0 ; $i-- ) {

                    $frame_fit6[$i-1] -> packForget;
                }
            }
        }, ) -> grid( -row => 4, -column => 1, -sticky => 'w', );

        $index_other[0] -> configure( -command => sub {
            if ( $index_bar_count != 0 ) {
                for ( my $i = $index_bar_count ; $i > 0 ; $i-- ) {
                    $frame_fit_bars[$i-1] -> packForget;
                    $index_bar_count--;
                }

                $index_button -> configure( -state => 'normal', );
            }
        } );
        $index_other[0] -> invoke;

        $index_other[1] -> configure( -command => sub {
            if ( $index_bar_count == 0 ) {
                $index_button -> configure( -state => 'disabled', );

                add_index_bar();
                $frame_fit_bars[$index_bar_count] -> pack( -anchor => 'w', );
                $index_bar_count++;
            }
        }, );

        my $index_row = 1;
        my $index_column = 4;

        $frame_fit4 -> Label( -text => "Use frame as reference: ", ) -> grid( -row => 2, -column =>
1, -sticky => 'w', );
        $frame_fit4 -> Entry( -textvariable => \$ref,  ) -> grid( -row => 2, -column => 2, -sticky
=> 'w', );

        $frame_fit4 -> Checkbutton( -text => 'No fit',
                                    -variable => \$no_fit,
```

```
                                -offvalue => '',
                                -onvalue => " -nofit", )
                                -> grid( -row => 3, -column => 1, -sticky => 'w', );

        my $frame_fit7 = $top_fit -> Frame() -> pack( -side => 'bottom', -expand => 0, );

        $frame_fit7 -> Button( -text => 'Return',
                               -command => sub { $top_fit -> withdraw; }, )
                               -> pack( -side => 'left', );

        $fit_run_button = $frame_fit7 -> Button( -text => 'Run',
                                                 -command => sub {
            $ref_flag = ( ( $ref != 1 ) ? " -ref $ref" : '' );

            create_dir();

            if ( $index and not $resid_active ) {
                if ( $linux || $mac ) {
                    $seg_id_flag = '' if $seg_id_flag;

                    foreach ( @seg_ids ) {
                        if ( defined ( $_ ) ) {
                            $seg_id_flag = $seg_id_flag . $_;
                        }
                    }

                    if ( $seg_id_flag ) {
                        `carma -v -w -last 1 $atm_id_flag $res_id_flag $custom_id_flag $seg_id_flag
$active_psf $active_dcd`;
                    }
                    else {
                        `carma -v -w -last 1 $atm_id_flag $custom_id_flag $res_id_flag $active_psf
$active_dcd`;
                    }
                }
                else {
                    $seg_id_flag = '' if $seg_id_flag;

                    foreach ( @seg_ids ) {
                        if ( defined ( $_ ) ) {
                            $seg_id_flag = $seg_id_flag . $_;
                        }
                    }

                    if ( $seg_id_flag ) {
                         `carma.exe  -v  -w  -last  1  $atm_id_flag  $res_id_flag  $custom_id_flag
$seg_id_flag $active_psf $active_dcd"`;
                    }
                    else {
                         `carma.exe  -v  -w  -last  1  $atm_id_flag  $res_id_flag  $custom_id_flag
$active_psf $active_dcd`;
                    }
                }

                unlink ( "$dcd_name.dcd.0000001.dat" );
                unlink ( "$dcd_name.dcd.0000001.ps" );

                $index_seg_id_flag = '';

                foreach ( @index_seg_ids ) {
                    if ( defined ( $_ ) ) {
                        $index_seg_id_flag = $index_seg_id_flag . ' ' . $_;
                    }
                }

                if ( $index_seg_id_flag ) {
                    create_fit_index( $index_atm_id, 'from_radiobutton', $index_seg_id_flag, );
                }
                else {
                    create_fit_index( $index_atm_id, 'from_radiobutton', );
                }

                $index = ' -index';
                $index_confirmation = 1;
```

```
            $fit_run_button -> configure( -state => 'normal', );
        }

        $top_fit -> destroy;

        $seg_id_flag = '' if $seg_id_flag;

        foreach ( @seg_ids ) {
            if ( defined ( $_ ) ) {
                $seg_id_flag = $seg_id_flag . $_;
            }
        }

        if ( $seg_id_flag ) {
             $flag  =  "  -w  -v  -fit  $ref_flag  $index  $no_fit  $atm_id_flag  $custom_id_flag
$res_id_flag $seg_id_flag";
        }
        else {
             $flag  =  "  -w  -v  -fit  $ref_flag  $index  $no_fit  $atm_id_flag  $custom_id_flag
$res_id_flag";
        }

        $text -> insert( 'end', "\n\nNow performing fitting.\n\n", 'cyan', );
        $text -> see( 'end', );
        $mw -> update;

        carma ( "fit" );
        if ( $all_done ) {
            if ( $active_dcd =~ /(.+).dcd/ ) {
                mv ( 'carma.fit-rms.dat', "rms_from_frame_$ref.$1.dat" );
            }

            $text -> insert( 'end', "\nFitting complete. Use \"View Results\"\n", 'valid' );
            $text -> see( 'end', );
            $image_menu -> configure( -state => 'normal', );

            my $response;
            if ( $linux or $mac ) {
                $response = $mw -> messageBox( -message => "Would you like to use this PSF - DCD
pair in other calculations?",
                                               -type => 'yesno',
                                               -icon => 'question',
                                               -font => "$font_12", );
            }
            else {
                $response = $mw -> messageBox( -message => "Would you like to use this PSF - DCD
pair in other calculations?",
                                               -type => 'yesno',
                                               -icon => 'question', );
            }

            if ( $response eq "Yes" ) {
                $dcd_count++;
                $active_dcd = "carma.fitted_$dcd_count.dcd";
                $active_psf = "carma.fitted_$dcd_count.psf";

                if ( $linux || $mac ) {
                    `mv carma.fitted.dcd carma.fitted_$dcd_count.dcd`;
                    `mv carma.selected_atoms.psf carma.fitted_$dcd_count.psf`;
                }
                else {
                    `move carma.fitted.dcd carma.fitted_$dcd_count.dcd`;
                    `move carma.selected_atoms.psf carma.fitted_$dcd_count.psf`;
                }

                ( $atm_id_flag, $res_id_flag, $custom_id_flag, $count, ) = ( '', '', '', 0, );
                undef @unique_chain_ids;
                undef @seg_ids;
                undef %num_residues;

                my $x = $mw -> width;
                my $y = $mw -> height;

                $f0 -> packForget;
```

```
                    parser( $active_psf, $active_dcd, );
                    $f0 -> pack( -side => 'top', -fill => 'both', -expand => 1, );
                    $mw -> geometry( "$x" . 'x' . "$y" );
                    $mw -> update;

                    $active_psf_label -> configure( -text => "$active_psf ", );
                    $active_dcd_label -> configure( -text => "$active_dcd", );
                    $go_back_button -> configure( -state => 'normal', );
                }
                else {
                    if ( $linux or $mac ) {
                        $mw -> messageBox( -type => "ok",
                                           -message  =>  "These  files  will  not  be  used  in  any
calculations and will be overwritten next time you perform a fitting",
                                          -font => "$font_12", );
                    }
                    else {
                        $mw -> messageBox( -type => "ok",
                                           -message  =>  "These  files  will  not  be  used  in  any
calculations and will be overwritten next time you perform a fitting", );
                    }
                }

                if ( $fit_plot ) {
                    plot ( 'carma.fit-rms.dat' );
                    $fit_plot = 0;
                }
            }
            else {
                $top_fit -> destroy;
                  $text   ->   insert(   'end',   "\nSomething   went   wrong.   For   details   check
last_carma_run.log located in :\n", 'error', );
                $text -> insert( 'end', getcwd . "\n", 'info', );
                $text -> see( 'end', );
            }
        }, ) -> pack( -side => 'right', );
    }
    else {
        $top_fit -> deiconify;
        $top_fit -> raise;
    }
}

#############################################################################################################
###   Create a subfolder in the CWD for the result files                                                  ###
#############################################################################################################

sub create_dir {
    my $input;

    if ( $_[0] and ( $_[0] eq 'stride' or $_[0] eq 'backup' ) ) {
        $input = shift;
    }

    # If the string returned by the getcwd #
    # function contains the name of the    #
    # folder used for storing the results  #
    # of the program then terminate the    #
    # subroutine with a success status     #
    if ( getcwd =~ /carma_results/ ) {
        unless ( $input and ( $input eq 'stride' or $input eq 'backup' ) ) {
            return(0);
        }
    }
    # If the folder does not exist in the  #
    # Cwd it is created and a subfolder    #
    # with the current time as it's name   #
    # will be created as well. This folder #
    # will serve as the storing point for  #
    # every grcarma session                #
    # If the folder exists then only the   #
    # subfolder of every session is made   #

    if ( ( -d -w $launch_dir and not $windows ) or $windows ) {
```

```
        if ( $input and $input eq 'stride' ) {
            mkpath ( "$launch_dir/$timeStamp/tmp", 0, 0755, );
            chdir ( "$launch_dir/$timeStamp/" );
        }
        else {
            mkpath ( "$launch_dir/$timeStamp", 0, 0755, );
            chdir ( "$launch_dir/$timeStamp" );
        }

        # After the folder(s) have been made   #
        # they are made the Cwd and links to   #
        # specified .psf and .dcd files are    #
        # created                              #
        if ( $linux || $mac ) {
            unless ( $input and $input eq 'stride' ) {
                `ln -s $psf_file .`;
                `ln -s $dcd_file .`;
            }
        }
        else {
            $psf_file =~ s/\"//g;
            $dcd_file =~ s/\"//g;

            link ( $psf_file, "$psf_name.psf", );
            link ( $dcd_file, "$dcd_name.dcd", );
            # link ( $new_psf, "$psf_name.psf", );

            $psf_file = "\"$psf_file\"";
            $dcd_file = "\"$dcd_file\"";
        }
    }
    else {
        if ( $run_from_terminal ) {
             die "\nSeems like you don't have write privileges for the folder the program was
launched from: $!\n\n";
        }
        else {
            if ( $linux or $mac ) {
                $mw -> messageBox( -message => "Seems like you don't have write privileges for the
folder the program was launched from: $!\n\n",
                                   -type => 'ok',
                                   -icon => 'warning',
                                   -font => "$font_12", );
            }
            else {
                $mw -> messageBox( -message => "Seems like you don't have write privileges for the
folder the program was launched from: $!\n\n",
                                   -type => 'ok',
                                   -icon => 'warning', );
            }

            $mw -> destroy;
            exit 1;
        }
    }
}

#############################################################################################################
###   Create the atmid radiobutton bar                                                                    ###
#############################################################################################################

sub radiobuttons {
    our $dist_list1;
    our $dist_list2;
    our $frame_dis2;

    our $bend_list1;
    our $bend_list2;
    our $bend_list3;
    our $frame_bnd2;

    our $tors_list1;
    our $tors_list2;
    our $tors_list3;
```

```
our $tors_list4;
our $frame_tor2;

our $frame_stride1;
our $frame_rgr1;
our $frame_ent2;
our $frame_sur1;

my $input = shift;
$input -> Label ( -text => 'Atom Selection', -font => "$font_20", ) -> pack;

my @radiobuttons = ( 'CA', 'Backbone', 'Heavy', 'All atoms', 'Custom selection', );
my @radio_b;

# For every item of the radiobuttons   #
# array draw a radiobutton named after #
# each item and if the radiobutton is  #
# active store it's name in a variable #
for my $i ( 0 .. $#radiobuttons ) {
    $radio_b[$i] = $input -> Radiobutton( -text => $radiobuttons[$i],
                                          -value => $radiobuttons[$i],
                                          -variable => \$atm_id,
                                          -command => sub {
        # If the above variable equals any of  #
        # the @radiobuttons entries specify    #
        # atmid flags for each entry           #
        if ( $atm_id eq 'CA' ) {
            $atm_id_flag = "";
            $custom_id_flag = '';
            $custom_selection = 0;
        }
        elsif ( $atm_id eq 'Backbone' ) {
            $atm_id_flag = " -atmid C -atmid CA -atmid N -atmid O";
            $custom_id_flag = '';
            $custom_selection = 0;
        }
        elsif ( $atm_id eq 'Heavy' ) {
            $atm_id_flag = " -atmid HEAVY";
            $custom_id_flag = '';
            $custom_selection = 0;
        }
        elsif ( $atm_id eq 'All atoms' ) {
            $atm_id_flag = " -atmid ALLID";
            $custom_id_flag = '';
            $custom_selection = 0;
        }

        if ( ( $frame_dis2 ) and $input eq $frame_dis2 ) {
            my ( @temp_list, ) = helper_function();

            $dist_list1 -> delete( 0, 'end' );
            $dist_list2 -> delete( 0, 'end' );

            $dist_list1 -> insert( 'end', @temp_list );
            $dist_list2 -> insert( 'end', @temp_list );
        }
        if ( ( $frame_bnd2 ) and $input eq $frame_bnd2 ) {
            my ( @temp_list, ) = helper_function();

            $bend_list1 -> delete( 0, 'end' );
            $bend_list2 -> delete( 0, 'end' );
            $bend_list3 -> delete( 0, 'end' );

            $bend list1 -> insert( 'end', @temp list );
            $bend_list2 -> insert( 'end', @temp_list );
            $bend_list3 -> insert( 'end', @temp_list );
        }
        if ( ( $frame_tor2 ) and $input eq $frame_tor2 ) {
            my ( @temp_list, ) = helper_function();

            $tors_list1 -> delete( 0, 'end' );
            $tors_list2 -> delete( 0, 'end' );
            $tors_list3 -> delete( 0, 'end' );
            $tors_list4 -> delete( 0, 'end' );
```

```
                $tors_list1 -> insert( 'end', @temp_list );
                $tors_list2 -> insert( 'end', @temp_list );
                $tors_list3 -> insert( 'end', @temp_list );
                $tors_list4 -> insert( 'end', @temp_list );
            }
        }, );

        $radio_b[$i] -> pack( -side => 'left', -anchor => 'w', );
    }

    if ( ( $frame_stride1 ) and $input eq $frame_stride1 ) {
        $radio_b[2] -> invoke();
    }
    elsif ( ( $frame_rgr1 ) and $input eq $frame_rgr1 ) {
        $radio_b[2] -> invoke();
    }
    elsif ( ( $frame_ent2 ) and $input eq $frame_ent2 ) {
        $radio_b[2] -> invoke();
    }
    elsif ( ( $frame_sur1 ) and $input eq $frame_sur1 ) {
        $radio_b[2] -> invoke();
    }
    else {
        $radio_b[0] -> invoke();
    }

    $radio_b[4] -> configure( -command => \&raise_custom_window);
}

#############################################################################################################
###   Create the segid checkbutton bar                                                                    ###
#############################################################################################################

sub checkbuttons {
    my $input = shift;

    our $frame_rmsd2;

    if ( ( $dpca_frame_1 && $input eq $dpca_frame_1 ) or ( $frame_sur2 && $input eq $frame_sur2 ) )
{
        $input -> Label( -text => "\nAt least one chain selection required", -font => "$font_20", )
-> pack;
    }
     elsif ( ( $frame_qfract2 && $input eq $frame_qfract2 ) or ( $frame_phi_psi1 and $input eq
$frame_phi_psi1 ) ) {
        $input -> Label( -text => "\nYou must select only one chain", -font => "$font_20", ) ->
pack;
    }
    else {
        $input -> Label( -text => "\nChain Selection", -font => "$font_20", ) -> pack;
    }

    my @check_b;

    for my $i ( 0 .. $#unique_chain_ids ) {
        $check_b[$i] = $input -> Checkbutton( -text => $unique_chain_ids[$i],
                                              -variable => \$seg_ids[$i],
                                              -offvalue => '',
                                              -onvalue => " -segid $unique_chain_ids[$i]",
                                              -command => sub {
            if ( $seg_ids[$i] ne '' ) {
                $count++;
            }
            else {
                $count--;
            }

            if ( $seg_ids[$i] ne '' ) {
                $active_run_buttons = 1;
                $dpca_run_button -> configure( -state => 'normal', ) if ( $dpca_run_button );
                $surf_run_button -> configure( -state => 'normal', ) if ( $surf_run_button );
            }
            elsif ( $count < 1 ) {
```

```
                $active_run_buttons = 0;
                $dpca_run_button -> configure( -state => 'disabled', ) if ( $dpca_run_button );
                $surf_run_button -> configure( -state => 'disabled', ) if ( $surf_run_button );
            }

            if (  $count == 1 ) {
                $active_run_buttons_qfract = 1;
                $qfract_run_button -> configure( -state => 'normal', ) if ( $qfract_run_button );
                $phi_psi_run_button -> configure( -state => 'normal', ) if ( $phi_psi_run_button );
            }
            elsif ( $count != 1 ) {
                $active_run_buttons_qfract = 0;
                $qfract_run_button -> configure( -state => 'disabled', ) if ( $qfract_run_button );
                $phi_psi_run_button -> configure( -state => 'disabled', ) if ( $phi_psi_run_button
);
            }
         }, );

        $check_b[$i] -> pack( -side => 'left', -anchor => 'w', );
    }

    if ( ( $frame_rmsd2 ) and $input eq $frame_rmsd2 ) {
        foreach my $element ( @check_b ) {
            if ( $element -> cget( -onvalue, ) =~ /segid (\w+)/ ) {
                $element -> select;
                $count++ unless ( $count == scalar ( @check_b ) );

                if ( $count < 1 ) {
                    $active_run_buttons = 0;
                }
                else {
                    $active_run_buttons = 1;
                }

                if (  $count == 1 ) {
                    $active_run_buttons_qfract = 1;
                }
                elsif ( $count != 1 ) {
                    $active_run_buttons_qfract = 0;
                }
            }
        }
    }

    if ( @unique_chain_ids == 1 ) {
        if ( ( $dpca_frame_1 and $input eq $dpca_frame_1 ) or
             ( $frame_sur2 and $input eq $frame_sur2 ) or
             ( $frame_qfract2 and $input eq $frame_qfract2 ) or
             ( $frame_phi_psi1 and $input eq $frame_phi_psi1 ) ) {
            foreach my $element ( @check_b ) {
                $element -> select;
                $count++ unless ( $count == scalar ( @check_b ) );

                if ( $count < 1 ) {
                    $active_run_buttons = 0;
                }
                else {
                    $active_run_buttons = 1;
                }

                if (  $count == 1 ) {
                    $active_run_buttons_qfract = 1;
                }
                elsif ( $count != 1 ) {
                    $active_run_buttons_qfract = 0;
                }
            }
        }
    }
}

##########################################################################################################
###   Create the resid radiobutton bar                                                                 ###
##########################################################################################################
```

```
sub otherbuttons {
    our $prev_psf;

    my @otherbuttons = ( 'All', 'Change', );
    our @other;

    $_[0] -> Label( -text => "\nResidue Selection", -font => "$font_20", ) -> pack;

    for my $i ( 0 .. $#otherbuttons ) {
        $other[$i] = $_[0] -> Radiobutton( -text => "$otherbuttons[$i]",
                                           -value => $otherbuttons[$i], );

        $other[$i] -> pack( -side => 'left', -anchor => 'w', );
    }

    $other[0] -> configure( -command => sub {
        $cancel_all_segids = 1;

        if ( $res_id_flag ) {
            $res_id_flag = '';
        }
        $have_custom_psf = 0;
        $f4_b -> destroy if ( $f4_b );

        $active_psf_label -> configure( -text => $prev_psf, ) if ( $prev_psf );
        $active_psf = $prev_psf if ( $prev_psf );
    }, );
    $other[1] -> configure( -command => [ \&resid_window ], );
}

#############################################################################################################
###   Create the scatter plot of the input file                                                           ###
#############################################################################################################

sub scatter plot {
    open IN, '<', 'phi_psi_temp_angles' or die $!;
    open CONVERTED_ANGLES, '>', "converted_angles" or die $!;

    my $j = 0;
    while ( my $line = <IN> ) {
        if ( $line =~ /(\s*)(\+|\-)(\d+.\d+)\s+(\s*)(\+|\-)(\d+.\d+)/ ) {
            chomp $line;

            my $phi_space = $1;
            my $phi_sign = $2;
            my $phi_value = "$2$3";

            my $psi_space = $4;
            my $psi_sign = $5;
            my $psi_value = "$5$6";

             $line = sprintf "%8.4f %8.4f\n", ( ( ( $phi_value + 180 ) / 0.6 ) + 25, ( ( $psi_value
- 180 ) / -0.6 ) + 25, );

             print CONVERTED_ANGLES $line;
             $j++;
         }
     }

    close IN;
    close CONVERTED_ANGLES;
    unlink ( "phi_psi_temp_angles" ,);

    my $mw = MainWindow -> new( -title => "Results Plot", );

    $mw -> configure( -menu => my $menubar = $mw -> Menu );
    my $file = $menubar -> cascade( -label => '~File' );

    my $size = 650;

    my $canvas = $mw->Canvas( -cursor=>"crosshair", -background=>"black",
                  -width=>$size, -height=>$size )->pack;
```

```
    my $kill_var= 0;
    $canvas -> CanvasBind("<$_>", sub {
        $canvas -> yviewMoveto( 0 )
    } ) for ( 4, 5, );

    $file -> command(
        -label       => 'Save As Postscript',
        -accelerator => 'Ctrl-s',
        -underline   => 0,
        -command     => sub {
            $canvas -> update;
            $canvas -> postscript( -file => "ramaplot.eps", -colormode => 'color', );
        }
    );
    $file->separator;
    $file->command(
        -label       => "Close",
        -accelerator => 'Ctrl-q',
        -underline   => 0,
        -command     => sub { $kill_var = 1; $mw -> destroy; },
    );

    $mw -> bind( $mw, "<Control-s>" => sub {
            $canvas -> update;
            $canvas -> postscript( -file => "ramaplot.eps", -colormode => 'color', );
        }
    );

    $mw -> bind( $mw, "<Control-q>" => sub { $kill_var = 1; $mw -> destroy; }
    );

    $canvas -> createRectangle( 0, 0, 650, 650, -fill => 'black', );

    $canvas -> createPolygon( 25, 253.5, 90.33, 298.17, 181.67, 298.17, 201.17, 249, 201.17, 204.17,
251.17, 155, 251.17, 56.5, 246.83, 25.17, 25, 25, -fill => '#282828', );
     $canvas -> createPolygon( 64.17, 172.83, 207.67, 172.83, 233.83, 137, 233.83, 36.33, 96.75,
36.33, 96.83, 65.33, 64.17, 99, 64.17, 172.83, -fill => '#353535', );
     $canvas -> createPolygon( 25, 383.17, 51.17, 396.5, 103.33, 396.5, 142.5, 378.67, 146.83,
360.67, 251.17, 360.67, 251.17, 443.5, 25, 443.5, -fill => '#282828', );
    $canvas -> createPolygon( 64.17, 425.67, 233.63, 425.67, 233.83, 392, 157.67, 392, 118.5, 410,
64.17, 410, 64.17, 425.67, -fill => '#353535', );
    $canvas -> createPolygon( 25, 598, 199, 598, 246.83, 625, 25, 625, -fill => '#282828', );
    $canvas -> createPolygon( 429.33, 300.5, 429.33, 163.83, 401, 193, 401, 280.33, 429.33, 300.5,
-fill => '#282828', );

    $canvas -> createText( 12, 325, -fill => 'white', -text => 'psi', ); # y axis
    $canvas -> createText( 13, 27, -fill => 'white', -text => '180', ); # y axis
    $canvas -> createText( 325, 633, -fill => 'white', -text => 'phi', ); # x axis
    $canvas -> createText( 13, 633, -fill => 'white', -text => '-180', ); # x axis
    $canvas -> createText( 615, 633, -fill => 'white', -text => '180', ); # x axis

    $canvas -> createLine( 25, 325, 625, 325, -fill => 'white', ); # y axis
    $canvas -> createLine( 325, 25, 325, 625, -fill => 'white', ); # x axis

    # Bounding box for the plot
    $canvas -> createLine( 25, 25, 625, 25, -fill => 'white', );
    $canvas -> createLine( 25, 25, 25, 625, -fill => 'white', );
    $canvas -> createLine( 25, 625, 625, 625, -fill => 'white', );
    $canvas -> createLine( 625, 25, 625, 625, -fill => 'white', );

    my $i = $canvas -> Photo( -width => 650, -height => 650, );
    $canvas -> createImage( 0, 0, -image => $i, -anchor => 'nw', );

    open CONVERTED_ANGLES, '<', "converted_angles" or die $!;

    $mw -> protocol( 'WM_DELETE_WINDOW' => sub { $kill_var = 1; $mw -> destroy; }, );

    CANVAS: while ( <CONVERTED_ANGLES> ) {
        last CANVAS if ( $kill_var );

        my $phi_angle = substr $_, 0, 8;
        my $psi_angle = substr $_, 9, 8;

        $phi_angle = int ( $phi_angle + 0.5 );
```

```
        $psi_angle = int ( $psi_angle + 0.5 );

        $i -> put('#f3922e', -to => ( $phi_angle, $psi_angle ) );

        if ( $j < 1000000 ) {
            $mw -> update unless ( $. % 100 );
        }
        elsif ( $j < 10000000 ) {
            $mw -> update unless ( $. % 1000 );
        }
        elsif ( $j < 100000000 ) {
            $mw -> update unless ( $. % 10000 );
        }
    }

    close CONVERTED_ANGLES;
    #~ close OUT;
    unlink ( "converted_angles", );
}

##############################################################################################################
###   Create the plot of the input matrix                                                                  ###
##############################################################################################################

sub plot {
    my $input = shift;
    my $header = dcd_header_parser( 'plot' );
    my $graph;

    my ( @frames,    @Q,    @Qs,    @q,
         @values,    @data, @step,  @legends,
         @Schlitter, @Andricioaei,  @clusters,
         @cutoffs, );

    our $image_top;

    my $mw = MainWindow -> new( -title => "Results Plot", );
    $mw -> configure( -menu => my $menubar = $mw -> Menu );
    my $file = $menubar -> cascade( -label => '~File' );
    my $view;

    my $lines = 0;

    open INPUT, '<', $input or die $!;
    while ( <INPUT> ) {
        $lines++;
    }

    close INPUT;

    my $entropy_total_min;
    my $entropy_total_max;
    my $entropy_step;

    my $negative_entropy_test = 0;

    if ( $input =~ /entropy/ and $input !~ /eigen/ ) {
        if ( -f 'entropy_first_step' ) {
            open ENTROPY_FIRST_STEP, '<', 'entropy_first_step' or die $!;
            while ( <ENTROPY_FIRST_STEP> ) {
                if ( /(\d+)/ ) {
                    $entropy_step = $1;
                }
            }

            close ENTROPY_FIRST_STEP;
        }

        my $entropy_counter = 0;
        my @entropy_min_max;

        if ( $input !~ /andrici|schlitter/ ) {
            open ENTROPY, '<', $input or die "Cannot open $input for reading : $!\n";
```

```
        while ( <ENTROPY> ) {
            if ( /\s*\d+\s+(\S+)\s+(\S+)/ ) {
                my $entropy_column_1 = $1;
                my $entropy_column_2 = $2;

                if ( $entropy_column_1 > 0 and $entropy_column_2 > 0 ) {
                    $entropy_min_max[$entropy_counter] = $entropy_column_1;
                    $entropy_min_max[$entropy_counter+1] = $entropy_column_2;
                }
                elsif ( $entropy_column_1 > 0 and $entropy_column_2 < 0 ) {
                    $entropy_min_max[$entropy_counter] = $entropy_column_1;
                    $negative_entropy_test = 1;
                }
                elsif ( $entropy_column_1 < 0 and $entropy_column_2 > 0 ) {
                    $entropy_min_max[$entropy_counter] = $entropy_column_2;
                    $negative_entropy_test = 1;
                }

                $entropy_counter++;
            }
        }

        my @entropy_min_max_sorted = sort { $a <=> $b } @entropy_min_max;

        $entropy_total_max = $entropy_min_max_sorted[scalar(@entropy_min_max_sorted)-1];
        $entropy_total_min = $entropy_min_max_sorted[0];

        close ENTROPY;
    }
    else {
        open ENTROPY, '<', $input or die "Cannot open $input for reading : $!\n";

        while ( <ENTROPY> ) {
            if ( /\s*\d+\s+(\S+)/ ) {
                my $entropy_column_1 = $1;

                if ( $entropy_column_1 > 0 ) {
                    $entropy_min_max[$entropy_counter] = $entropy_column_1;
                }
                elsif ( $entropy_column_1 < 0 ) {
                    $entropy_min_max[$entropy_counter] = $entropy_column_1;
                    $negative_entropy_test = 1;
                }

                $entropy_counter++;
            }
        }

        my @entropy_min_max_sorted = sort { $a <=> $b } @entropy_min_max;

        $entropy_total_max = $entropy_min_max_sorted[scalar(@entropy_min_max_sorted)-1];
        $entropy_total_min = $entropy_min_max_sorted[0];

        close ENTROPY;
    }
}
else {
    $view= $menubar -> cascade( -label => '~View' );
}

# menubar configuration

$file -> command(
    -label       => 'Save As Postscript',
    -accelerator => 'Ctrl-s',
    -underline   => 0,
    -command     => sub {
        $graph -> update;
        $graph -> postscript( -file => "$input.eps", -colormode => 'color', );
    }
);
$file->separator;
$file->command(
    -label       => "Close",
```

```
        -accelerator => 'Ctrl-q',
        -underline   => 0,
        -command     => sub { $mw -> destroy; },
    );

    $mw -> bind( $mw, "<Control-s>" => sub {
            $graph -> update;
            $graph -> postscript( -file => "$input.eps", -colormode => 'color', );
        }
    );

    $mw -> bind( $mw, "<Control-q>" => sub { $mw -> destroy; }
    );

    my $screenwidth = $mw -> screenwidth;

    my $line_skip = 0;
    if ( $lines < $screenwidth ) {
        $line_skip = 1;
    }
    elsif ( $lines / 5000 <= 1.0 ) {
        $line_skip = 1;
    }
    else {
        $line_skip = int ( $lines / 5000 );
    }

    open IN, '<', $input || die "Cannot open $input for reading: $!";

    if ( -z $input ) {
        close IN;
        if ( $linux or $mac ) {
            $image_top -> messageBox( -message => 'The file you are trying to plot seems to be
empty. The file is located in ' . getcwd,
                                      -icon => 'warning',
                                      -font => "$font_12", );
        }
        else {
            $image_top -> messageBox( -message => 'The file you are trying to plot seems to be
empty. The file is located in ' . getcwd,
                                      -icon => 'warning', );
        }

        $image_top -> destroy;
    }
    else {
        my $i = 0;
        while ( my $line = <IN> ) {
            if ( $. == $line_skip * $i+1 && $input =~ /qfract/i && $line =~ /\s+(\d+)\s+(\S+)\s+
(\S+)\s+(\S+)/ ) {
                $frames[$i] = $1;
                $Q[$i] = $2;
                $Qs[$i] = $3;
                $q[$i] = $4;
                $i++;
            }
            elsif ( $. == $line_skip * $i+1 && $input =~ /rms_from|rgyr|bend|tors|dist/i && $line =~
/\s+(\d+)\s+([-+]?\d+\.?\d*)/ ) {
                $frames[$i] = $1;
                $values[$i] = $2;
                $i++;
            }
            elsif ( $input =~ /surf/i ) {
                if ( $. == $line skip * $i+1 and $line =~ /\s+(\d+)\s+(\d+\.?\d*)/ ) {
                    $frames[$i] = $1;
                    $values[$i] = $2;
                    $i++;
                }
            }
            elsif ( $input =~ /entropy/ and $input !~ /andrici|schlitter/ && $line =~ /\s+(\S+?)\s+
(\S+)\s*(\S*)/ ) {
                $frames[$i] = $1*$entropy_step;
                $Andricioaei[$i] = $2;
                $Schlitter[$i] = $3;
```

```
            $i++;
        }
        elsif ( $input =~ /andrici|schlitter/i and $line =~ /\s+(\S+?)\s+(\S+)/ ) {
            $frames[$i] = $1*$entropy_step;
            $values[$i] = $2;
            $i++;
        }
        elsif ( $input =~ /clusters_vs_variance/ && $line =~ /^\s*(\d+)\s+(\S+)/ ) {
            $frames[$i] = $1;
            $values[$i] = $2;
            $i++;
        }
        elsif ( $input =~ /clusters_vs_rms_cutoff/ && $line =~ /^\s*(\d+)\s+(\S+)/ ) {
            $frames[$i] = $1;
            $values[$i] = $2;
            $i++;
        }
        elsif ( $input =~ /eigenvalues/ && $line =~ /^\s*(\S+)/ ) {
            $frames[$i] = $i;
            $values[$i] = $1;
            $i++;
        }
    }

    close IN;

    my $tick;
    if ( $frames[$i-1] <= 10 ) {
        $tick = 1;
    }
    elsif ( $frames[$i-1] <= 100 ) {
        $tick = 10;
    }
    elsif ( $frames[$i-1] <= 1000 ) {
        $tick = 100;
    }
    elsif ( $frames[$i-1] <= 10000 ) {
        $tick = 1000;
    }
    elsif ( $frames[$i-1] <= 100000 ) {
        $tick = 10000;
    }
    elsif ( $frames[$i-1] <= 1000000 ) {
        $tick = 100000;
    }
    elsif ( $frames[$i-1] <= 10000000 ) {
        $tick = 1000000;
    }

    my ( $dataset1, $dataset2, $dataset3, $dataset4, $dataset5, $dataset6, );

    unless ( $input =~ /entropy.*dat/ ) {
        $view -> command(
        -label       => "Show as lines",
        #~ -accelerator => 'Ctrl-l',
        -underline   => 8,
        -command     => sub {
            $graph -> clearDatasets();
            $graph -> plot;

            if ( $input =~ /qfract/i ) {
                $dataset1 = LineGraphDataset -> new( -name => 'Q',
                                                     -xData => \@frames,
                                                     -yData => \@Q,
                                                     -color => 'blue', );
                $dataset2 = LineGraphDataset -> new( -name => 'Qs',
                                                     -xData => \@frames,
                                                     -yData => \@Qs,
                                                     -color => 'green', );
                $dataset3 = LineGraphDataset -> new( -name => 'q',
                                                     -xData => \@frames,
                                                     -yData => \@q,
                                                     -color => 'purple', );
```

```
            $graph -> addDatasets( $dataset1, $dataset2, $dataset3, );
        }
        else {
            $dataset6 = LineGraphDataset -> new( -name => $input,
                                                 -xData => \@frames,
                                                 -yData => \@values,
                                                 -color => 'blue', );

            $graph -> addDatasets( $dataset6, );
        }

        $graph -> plot;
    },
    );
    $view -> separator;
    $view -> command(
    -label       => "Show as dots",
    #~ -accelerator => 'Ctrl-d',
    -underline   => 8,
    -command     => sub {
        $graph -> clearDatasets();
        $graph -> plot;

        if ( $input =~ /qfract/i ) {
            $dataset1 = LineGraphDataset -> new( -name => 'Q',
                                                 -xData => \@frames,
                                                 -yData => \@Q,
                                                 -lineStyle => 'none',
                                                 -pointStyle => 'circle',
                                                 -pointSize => 1,
                                                 -color => 'blue', );
            $dataset2 = LineGraphDataset -> new( -name => 'Qs',
                                                 -xData => \@frames,
                                                 -yData => \@Qs,
                                                 -lineStyle => 'none',
                                                 -pointStyle => 'circle',
                                                 -pointSize => 1,
                                                 -color => 'green', );
            $dataset3 = LineGraphDataset -> new( -name => 'q',
                                                 -xData => \@frames,
                                                 -yData => \@q,
                                                 -lineStyle => 'none',
                                                 -pointStyle => 'circle',
                                                 -pointSize => 1,
                                                 -color => 'purple', );

            $graph -> addDatasets( $dataset1, $dataset2, $dataset3, );
        }
        else {
            if ( $input =~ /eigenvalues/ ) {
                $dataset6 = LineGraphDataset -> new( -name => $input,
                                                     -xData => \@frames,
                                                     -yData => \@values,
                                                     -lineStyle => 'none',
                                                     -pointStyle => 'circle',
                                                     -pointSize => 3,
                                                     -color => 'blue', );
            }
            else {
                $dataset6 = LineGraphDataset -> new( -name => $input,
                                                     -xData => \@frames,
                                                     -yData => \@values,
                                                     -lineStyle => 'none',
                                                     -pointStyle => 'circle',
                                                     -pointSize => 1,
                                                     -color => 'blue', );
            }

            $graph -> addDatasets( $dataset6, );
        }

        $graph -> plot;
    }, );
}
```

```
        if ( $input =~ /variance|rms_cutoff/ ) {
            open VARIANCE, '<', $input or die "Cannot open $input for reading : $!\n";

            my $i = 0;
            while ( <VARIANCE> ) {
                $i++;
            }
            close VARIANCE;

            if ( $i < 2 ) {
                if ( $linux or $mac ) {
                    $mw -> messageBox( -message => 'Only one line read from the file $input. Will
not plot', -font => "$font_12", );
                    $mw -> destroy;
                }
                else {
                    $mw -> messageBox( -message => 'Only one line read from the file $input. Will
not plot', );
                    $mw -> destroy;
                }
                return;
            }

            if ( $input =~ /variance/ ) {
                $graph = $mw -> PlotDataset( -width => 800,
                                             -height => 600,
                                             -background => 'snow',
                                             -xlabel => 'Cluster',
                                             -autoScaleX => 'Off',
                                             -scale => [ '1', $i, '1', '', '', '', '', '', '', ],
                                             -ylabel => 'Variance explained',
                                             -plotTitle => [ $input, 20, ] )
                                             -> pack( qw/ -fill both -expand 1/ );
            }
            else {
                $graph = $mw -> PlotDataset( -width => 800,
                                             -height => 600,
                                             -background => 'snow',
                                             -xlabel => 'Cluster',
                                             -autoScaleX => 'Off',
                                             -scale => [ '1', $i, '1', '', '', '', '', '', '', ],
                                             -ylabel => 'RMS cutoff',
                                             -plotTitle => [ $input, 20, ] )
                                             -> pack( qw/ -fill both -expand 1/ );
            }
        }
        elsif ( $input =~ /entropy.*dat/ ) {
            $graph = $mw -> PlotDataset( -width => 800,
                                         -height => 600,
                                         -background => 'snow',
                                         -ylabel => '',
                                         -xlabel => 'Frame',
                                         -autoScaleY => 'Off',
                                         -autoScaleX => 'Off',
                                           -scale   =>   [   '1',   $i*$entropy_step,   $entropy_step,
$entropy_total_min-1, $entropy_total_max+1, '100', '', '', '', ],
                                         -ylabelPos => 50,
                                         -plotTitle => [ $input, 20, ] )
                                         -> pack( qw/ -fill both -expand 1/ );
        }
        elsif ( $input =~ /surface/ ) {
            $graph = $mw -> PlotDataset( -width => 800,
                                         -height => 600,
                                         -background => 'snow',
                                         -xlabel => 'Frame',
                                         -ylabel => '',
                                         -autoScaleX => 'Off',
                                         -scale => [ '0', $frames[$i-1], $tick, '', '', '', '', '',
'', ],
                                         -plotTitle => [ $input, 20, ] )
                                         -> pack( qw/ -fill both -expand 1/ );
        }
        else {
```

```
            if ( $input =~ /eigenvalues/ ) {
                $graph = $mw -> PlotDataset( -width => 800,
                                             -height => 600,
                                             -background => 'snow',
                                             -xlabel => 'Frame',
                                             -autoScaleX => 'Off',
                                             -scale => [ '0', $frames[$i-1], $tick, '', '', '', '',
'', '', ],
                                             -ylabel => 'Eigenvectors',
                                             -plotTitle => [ $input, 20, ] )
                                             -> pack( qw/ -fill both -expand 1/ );
            }
            else {
                $graph = $mw -> PlotDataset( -width => 800,
                                             -height => 600,
                                             -background => 'snow',
                                             -xlabel => 'Frame',
                                             -autoScaleX => 'Off',
                                              -scale => [ $frames[0], $frames[$i-1], $tick, '', '',
'', '', '', '', ],
                                             -ylabel => 'Value',
                                             -plotTitle => [ $input, 20, ] )
                                             -> pack( qw/ -fill both -expand 1/ );
            }
        }

        if ( $input =~ /qfract/i ) {
            $dataset1 = LineGraphDataset -> new( -name => 'Q',
                                                 -xData => \@frames,
                                                 -yData => \@Q,
                                                 -color => 'blue', );
            $dataset2 = LineGraphDataset -> new( -name => 'Qs',
                                                 -xData => \@frames,
                                                 -yData => \@Qs,
                                                 -color => 'green', );
            $dataset3 = LineGraphDataset -> new( -name => 'q',
                                                 -xData => \@frames,
                                                 -yData => \@q,
                                                 -color => 'purple', );

            $graph -> addDatasets( $dataset1, $dataset2, $dataset3, );
        }
        elsif ( $input =~ /entropy.*dat/i ) {
            if ( $negative_entropy_test ) {
                if ( $input !~ /andrici|schlitter/ ) {
                    $dataset4 = LineGraphDataset -> new( -name => 'Andricioaei',
                                                         -xData => \@frames,
                                                         -yData => \@Andricioaei,
                                                         -lineStyle => 'none',
                                                         -pointStyle => 'circle',
                                                         -color => 'blue', );
                    $dataset5 = LineGraphDataset -> new( -name => 'Schlitter',
                                                         -xData => \@frames,
                                                         -yData => \@Schlitter,
                                                         -lineStyle => 'none',
                                                         -pointStyle => 'diamond',
                                                         -color => 'green', );

                    $graph -> addDatasets( $dataset4, $dataset5, );
                }
                elsif ( $input !~ /schlitter/ ) {
                    $dataset4 = LineGraphDataset -> new( -name => 'Andricioaei',
                                                         -xData => \@frames,
                                                         -yData => \@values,
                                                         -lineStyle => 'none',
                                                         -pointStyle => 'circle',
                                                         -color => 'blue', );

                    $graph -> addDatasets( $dataset4, );
                }
                elsif ( $input !~ /andrici/ ) {
                    $dataset5 = LineGraphDataset -> new( -name => 'Schlitter',
                                                         -xData => \@frames,
                                                         -yData => \@values,
```

```
                                                    -lineStyle => 'none',
                                                    -pointStyle => 'circle',
                                                    -color => 'blue', );

                    $graph -> addDatasets( $dataset5, );
                }
            }
            else {
                if ( $input !~ /andrici|schlitter/ ) {
                    $dataset4 = LineGraphDataset -> new( -name => 'Andricioaei',
                                                    -xData => \@frames,
                                                    -yData => \@Andricioaei,
                                                    -color => 'blue', );
                    $dataset5 = LineGraphDataset -> new( -name => 'Schlitter',
                                                    -xData => \@frames,
                                                    -yData => \@Schlitter,
                                                    -color => 'green', );

                    $graph -> addDatasets( $dataset4, $dataset5, );
                }
                elsif ( $input !~ /schlitter/ ) {
                    $dataset4 = LineGraphDataset -> new( -name => 'Andricioaei',
                                                    -xData => \@frames,
                                                    -yData => \@values,
                                                    -color => 'blue', );

                    $graph -> addDatasets( $dataset4, );
                }
                elsif ( $input !~ /andrici/ ) {
                    $dataset5 = LineGraphDataset -> new( -name => 'Schlitter',
                                                    -xData => \@frames,
                                                    -yData => \@values,
                                                    -color => 'blue', );

                    $graph -> addDatasets( $dataset5, );
                }
            }
        }
        else {
            if ( $input =~ /tors|bend/ ) {
                $dataset6 = LineGraphDataset -> new( -name => $input,
                                                -xData => \@frames,
                                                -yData => \@values,
                                                -lineStyle => 'none',
                                                -pointStyle => 'circle',
                                                -pointSize => 1,
                                                -color => 'blue', );
            }
            elsif ( $input =~ /eigenvalues/ ) {
                $dataset6 = LineGraphDataset -> new( -name => $input,
                                                -xData => \@frames,
                                                -yData => \@values,
                                                -lineStyle => 'none',
                                                -pointStyle => 'circle',
                                                -pointSize => 3,
                                                -color => 'blue', );
            }
            else {
                $dataset6 = LineGraphDataset -> new( -name => $input,
                                                -xData => \@frames,
                                                -yData => \@values,
                                                -color => 'blue', );
            }

            $graph -> addDatasets( $dataset6, );
        }

        $graph -> plot;

        $graph -> CanvasBind("<$_>", sub {
            $graph -> yviewMoveto( 0 )
        } ) for ( 4, 5, );
    }
}
```

```
#############################################################################################################
###   Create the help window                                                                              ###
#############################################################################################################

sub about {
    my $top = $mw -> Toplevel( -title => "Help", );
    $top -> geometry( "$toplevel_position" );

    $top -> Label( -text => "This is grcarma v$VERSION", ) -> pack;
    $top -> Button( -text => "View online documentation",
                    -command => sub {
        if ( $linux ) {
            system ( "x-www-browser https://github.com/pkoukos/grcarma/wiki" );
        }
        elsif ( $mac ) {
            system ( "open https://github.com/pkoukos/grcarma/wiki" );
        }
        elsif ( $windows ) {
            system ( "start https://github.com/pkoukos/grcarma/wiki" );
        }
    }, ) -> pack;
}

#############################################################################################################
###   The subroutine that determines the cumulative size of the various carma_results folders   ###
#############################################################################################################

sub folder_size {
    no warnings;

    my $size;

    foreach ( glob ( "carma_results*/*" ) ) {
        my @filestats = stat $_;
        $size += $filestats[7] if ( not -l );
    }

    $wd_size = sprintf( "%d", $size / 1024 / 1024 );
}
```

Βιβλιογραφικές Αναφορές

1. Dror R. O., Dirks R. M., Grossman J. P., Xu H., and Shaw D. E., *Annu. Rev. Biophys.* **2012**, *41*, 429-452.
2. Adcock S. A. and McCammon J. A., *Chemical Reviews* **2006,** *106*, 1589-1615.
3. Karplus M., Petsko G. A., *Nature* **1990**, *347*, 631-639.
4. McCammon J. A., Gelin B. R. & Karplus M. *Nature* **1977,** *267*, 585–590.
5. Cornell W.D., Cieplak P., Bayly C.I., Gould I.R., Merz Jr. K.M., Ferguson D.M., Spellmeyer D.C., Fox T., Caldwell J.W., Kollman P.A., *J. Am. Chem. Soc.* **1995**, *117,* 5179–5197.
6. MacKerell Jr. A.D., Bashford D., Bellott M., Dunbrack Jr. R.L., Evanseck J.D., Field M.J., Fischer S., Gao J., Guo H., Ha S., Joseph-McCarthy D., Kuchnir L., Kuczera K., Lau F.T.K., Mattos C., Michnick S., Ngo T., Nguyen D.T., Prodhom B., Reiher III W.E., Roux B., Schlenkrich M., Smith J.C., Stote R., Straub J., Watanabe M., Wiorkiewicz-Kuczera J., Yin D., Karplus M., *J. Phys. Chem. B* **1998**, *102,* 3586–3616.
7. Scott W.R.P., Hunenberger P.H., Tironi I.G., Mark A.E., Billeter S.R., Fennen J., Torda A.E., Huber T., Kruger P., W.F. van Gunsteren, *J. Phys. Chem. A* **1999,** *103,* 3596–3607.
8. Humphrey, W., Dalke, A. and Schulten, K., *J. Molec. Graphics* **1996**, *14*, 33-38.
9. Phillips J. C., Braun R., Wang W., Gumbart J., Tajkhorshid E., Villa E., Chipot C., Skeel R. D., Kale L., and Schulten K., *J. Comput. Chem.* **2005**, *26*, 1781-1802.
10. Glykos, N. M., *J. Comput. Chem.* **2006,** *27*, 1765-1768.
11. Stamatakis M. and Vlachos D. G., *ACS Catalysis* **2012,** *2,* 2648-2663.

12. Jayaram B., Dhingra P., Lakhani B. and Shekhar S., *J. Chem. Sci.* **2012,** *124*, 83–91.

13. Wan, S., Coveney, P.V., *J. Comput. Chem.* **2011,** *32,* 2843-2852.

14. Friedman R., Boye K., Flatmark K., *Biochim. Biophys. Acta, Rev. Cancer* **2013,** *1836*, 1-14.

15. Csermely P., Korcsmáros T., Kiss H. J. M., London G., Nussinov R., *Pharmacol. Ther.* **2013,** *138*, 333-408.

16. Hirotaka O., Masaaki N., Shingo K., Wataru S., Hironori S., *Front. Microbiol.* **2012,** *3,* 258.

17. Lindorff-Larsen, K., Maragakis, P., Piana, S., Eastwood, M.P., Dror, R.O., Shaw, D.E., *PLoS ONE* **2012,** *7*, e32131.

18. Patmanidis I. & Glykos N.M., *J. Mol. Graph. Model.* **2013,** *41*, 68-71.

19. Sayle R. and Milner-White E. J., *Trends Biochem. Sci.* **1995**, *20*, 374.

20. D. Frishman, P. Argos, *Proteins: Struct., Funct., Genet.* **1995**, *23*, 566-579.

21. Schneider T. D., Stephens R. M., *Nucleic Acids Res.* **1990**, *18,* 6097-6100.

22. S. S. Cho, Y. Levy, P. G. Wolynes , *Proc. Natl. Acad. Sci. U. S. A.* **2006**, 103, 586-591.

23. J. Schlitter*, Chem. Phys. Lett.* **1993**, *215*, 6.

24. I. Andricioaei, M. Karplus, *J. Chem. Phys*. **2001**, *115*, 6289.

25. Fadouloglou V.E., Stavrakoudis S., Bouriotis V., Kokkinidis M., & Glykos N.M., *J. Chem. Theory Comput.* **2009**, *5*, 3299-3311.