



# Εφαρμογές Γραμμικής Άλγεβρας σε python με multiprocessing

ΟΜΑΔΙΚΗ ΕΡΓΑΣΙΑ 1<sup>ΟΥ</sup> ΕΞΑΜΗΝΟΥ

ECE\_Υ106 | Εισαγωγή στους Υπολογιστές | 2020-21

ΟΜΑΔΑ 18



Επιβλέπων Καθηγητής: Παλιουράς Βασίλης

## Περιεχόμενα

Οργάνωση Ομάδας – Μέλη .....	2
Αναθέσεις ρόλων – διαμοιρασμός του φορτίου .....	2
Θέμα Ομαδικής Εργασίας και Επεξήγηση .....	3
Βιβλιοθήκες που χρησιμοποιήθηκαν .....	3
Ανάλυση του κώδικα .....	4
Μετέπειτα διορθώσεις .....	4
Παρατηρήσεις <sup>3</sup> .....	5
Ανακατευθύνσεις .....	5
Εγκατάσταση του προγράμματος .....	6
Παραδείγματα χρήσης .....	7
Ο τελικός κώδικας .....	11
Βιβλιογραφία – Πηγές πληροφόρησης .....	64

## Οργάνωση Ομάδας – Μέλη

Η ομάδα αποτελείται από τους εξής φοιτητές του τμήματος:

Γιακουμέλου Αιμιλία με AM: up1083878

Ντάγκας Αλέξανδρος με AM: up1083874

Ντεν- Μπραμπερ Βερνάρδος- Αδριανός με AM: up1083808

Παπουτσάς Γεώργιος με AM: up1083738

Ροδόπουλος Γεώργιος με AM: up1083876

Ψημμένος Επαμεινώνδας με AM: up1083815

## Αναθέσεις ρόλων – διαμοιρασμός του φορτίου

Η ομάδα αρχικά έκρινε πως είναι σφώφρων ο διαμοιρασμός του προγράμματος να γίνει με τέτοιο τρόπο ώστε να είναι ξεκάθαρο το πεδίο με το οποίο θα ασχοληθεί κάθε φοιτητής ώστε η ανάπτυξη του κώδικα να χαρακτηρίζεται από συνοχή και από ευφράδεια (να μην υπάρχουν δηλαδή επαναλήψεις στον κώδικα καθώς και περιττές διεργασίες που το καθιστούν “βαρύτερο”). Έχοντας υπ’ όψη πρώτον τα κομμάτια της ύλης που καλύφθηκαν/επεξηγήθηκαν κατά την διάρκεια των διαλέξεων αλλά και των εργαστηρίων, δεύτερον τον χρόνο που παραδόθηκαν αλλά και τρίτον την πολυπλοκότητα αυτών, οι φοιτητές συμφώνησαν με την διανομή φορτίου – ανάθεση ρόλων. Η διανομή όμως δεν κατέστη ιδιαίτερα ξεκάθαρη αφού: βιβλιοθήκες όπως η multiprocessing δεν διδάχθηκαν και εν τέλει θεωρήθηκε δυσκολότερη - από αυτή που είχε προβλεφθεί - η διαχείρισή τους, ειδικότερα στον συνδυασμό αυτών. Οι διάφορες διεργασίες που δημιουργήθηκαν δεν ήταν απόλυτα συμβατές μεταξύ τους με αποτελέσματα πολλές φορές να καταλήγουν σε λάθος αποτελέσματα. Έτσι κρίθηκε απαραίτητη η διεύρυνση των ρόλων ώστε να λυθούν τέτοιου είδους προβλήματα.

Οι γραφικές απεικονίσεις, που αποτελούν το μεγαλύτερο μέρος του πηγαίου κώδικα, εκπονήθηκαν ως επί των πλείστων από τους: Γεώργιος Ροδόπουλος, Αλέξανδρος Ντάγκας & Γεώργιος Παπουτσάς . Σημαντική όμως ήταν και η συνδρομή των φοιτητών Αιμιλία Γιακουμέλου & Επαμεινώνδα Ψημμένου στο κομμάτι αυτό, για διάφορες λειτουργίες – διορθώσεις λ.χ. καταγραφή των errors και υλοποίηση προειδοποιητικών μηνυμάτων (error message-box) για την αποφυγή λαθών.

Η αρχική ανάλυση και χρήση των κατάλληλων συναρτήσεων της βιβλιοθήκης NumPy, καθώς και η συγγραφή – υλοποίηση των χωρίς multiprocessing συναρτήσεων πράξεων (κλάση SimpleCalculation και συνάρτηση calculate της κλάσης GUI), από τους φοιτητές Επαμεινώνδα Ψημμένο & Αιμιλία Γιακουμέλου.

Το σύνολο των φοιτητών ασχολήθηκε με τον συνδυασμό των συναρτήσεων που χρησιμοποιήθηκαν από την βιβλιοθήκη NumPy με την βιβλιοθήκη multiprocessing ώστε

να πετύχουν το βέλτιστο αποτέλεσμα (χρήση Παράλληλου Προγραμματισμού), με μεγάλο μέρος της διαδικασίας αυτής να αποτελεί εργασία των Αιμιλία Γιακουμέλου & Επαμεινώνδας Ψημμένο.

Το σύνολο των φοιτητών επίσης ασχοληθεί και με την Μετατροπή των εκάστοτε συναρτήσεων από Συναρτησιακό Προγραμματισμό (Functional Programming) σε Αντικειμενοστραφή (Object Oriented Programming), με μεγάλο μέρος της διαδικασίας αυτής να αποτελεί εργασία του Βερνάρδου- Αδριανού Ντεν- Μπραμπερ.

Κατά την διάρκεια των δοκιμών παρήχθησαν αρκετά αρχεία κώδικα, το καθένα με διαφορετική πολυπλοκότητα, ιδιαιτερότητα και χρόνο απόκρισης (π.χ. κάποια ήταν λειτουργικά για πράξεις μεταξύ μεγάλων διαστάσεων πινάκων ενώ για μικρά αποτύγχαναν). Έτσι στην τελική σύνθεση χρησιμοποιήθηκαν ολόκληρα αρχεία ή τμήματα αυτών.

Την συγγραφή της παρούσας έκθεσης ανέλαβε ο Επαμεινώνδας Ψημμένος.

## Θέμα Ομαδικής Εργασίας και Επεξήγηση

Το θέμα της ομαδικής εργασίας έχει ορισθεί ως «*Εφαρμογές γραμμικής άλγεβρας με multiprocessing και python*».

Η ομάδα κλήθηκε να δημιουργήσει πρόγραμμα σε python με το οποίο θα εκτελούνται πράξεις Γραμμικής Άλγεβρας Πινάκων (πολλαπλασιασμός, πρόσθεση, πολλαπλασιασμός πίνακα με αριθμό, εύρεση ορίζουσας, ύψωση πίνακα σε δύναμη κ.α.), με το οποίο θα γίνονται αισθητές οι χρονικές διαφορές στον υπολογισμό των πράξεων με χρήση multiprocessing και μη.

## Βιβλιοθήκες που χρησιμοποιήθηκαν

Κατά την υλοποίηση χρησιμοποιήθηκαν αρκετές βιβλιοθήκες της Python, με την χρήση των οποίων επιτεύχθηκαν ταχύτερα ορισμένες διαδικασίες (πχ χρήση της βιβλιοθήκης NumPy<sup>1</sup> για τις πράξεις πινάκων), κατέστη ευκολότερος ο προγραμματισμός για τις γραφικές απεικονίσεις (χρήση της βιβλιοθήκης tkinter) και μειώθηκε ο χρόνος υπολογισμού (χρήση της βιβλιοθήκης multiprocessing). Κατά την τελική σύνθεση του προγράμματος έγινε βελτιστοποίηση των εν χρήση βιβλιοθηκών ώστε η εκτέλεση του προγράμματος να απαιτεί τις ελάχιστες δυνατές<sup>2</sup>.

Οι βιβλιοθήκες που χρησιμοποιήθηκαν είναι:

1. Η βιβλιοθήκη **tkinter** που εξασφαλίζει την γραφική απεικόνιση του προγράμματος
2. Η βιβλιοθήκη **NumPy** με την χρήση της οποίας εκτελούνται οι πράξεις των Πινάκων
3. Η βιβλιοθήκη **time** με την οποία στο παρόν πρόγραμμα προσδιορίζεται ο πραγματικός χρόνος εκτέλεσης της εκάστοτε χρονομετρούμενης διεργασίας (π.χ.

- εκτέλεσης της πράξης του πολλαπλασιασμού με και χωρίς την χρήση της βιβλιοθήκης multiprocessing)
4. Η βιβλιοθήκη **PIL** με την οποία είναι εφικτή ή εισαγωγή του “Λογοτύπου” και του favicon.
  5. Η βιβλιοθήκη **tkinter.messagebox** με την οποία δημιουργούνται τα παράθυρα διαλόγου προβλήματος (error windows)
  6. Η βιβλιοθήκη **multiprocessing** με την οποία εξασφαλίζεται πως κατά την εκτέλεση του προγράμματος θα χρησιμοποιούνται πλέον του ενός επεξεργαστές (ή τμήματα επεξεργαστή/εικονικοί επεξεργαστές) ώστε να επιταχυνθούν οι διεργασίες

## Ανάλυση του κώδικα

Ο κώδικας του προγράμματος αποτελείται από 4 κλάσεις. Πιο συγκεκριμένα:

- **Κλάση GUI**: η κλάση στην οποία βασίζονται όλα τα γραφικά στοιχεία
- **Κλάση SimpleCalculation**: η κλάση αυτή περιέχει το σύνολο των πράξεων των πινάκων (χωρίς χρήση του παράλληλου προγραμματισμού)
- **Κλάση MultiprocessingCalculation**: σε αυτή την κλάση γίνονται όλες οι διεργασίες για τον παράλληλο προγραμματισμό των πράξεων πινάκων (χρησιμοποιώντας την κλάση Process)
- **Κλάση RandomMatrix**: εδώ εκτελούνται όλες οι διεργασίες που απαιτούνται για την διαμόρφωση των πινάκων, δημιουργία δεδομένων από αυτούς που χρειάζονται στην κλάση MultiprocessingCalculation

Αναλυτικότερα σχόλια για τον κώδικα και της συναρτήσεις υπάρχουν στον ίδιο τον κώδικα και στο κεφάλαιο « Ο Τελικός Κώδικας»

## Μετέπειτα διορθώσεις - εξέλιξη

Ως μετέπειτα διορθώσεις – εξέλιξη του κώδικα, αποτελεί η καθολική χρήση του παράλληλου προγραμματισμού ώστε να βελτιωθεί η τόσο των υπολογιστικών διεργασιών αλλά και των γραφικών διεπαφών, η δημιουργία executable αρχείου εγκατάστασης του προγράμματος αλλά και η διανομή αυτού. Επιπλέον ως εξέλιξη του προγράμματος αποτελεί και η ιδέα δημιουργίας ηλεκτρονικής σελίδας – ιστοτόπου, όπου το παρόν πρόγραμμα θα “τρέχει” online χωρίς την ανάγκη εγκατάστασης αυτού ή οποιοδήποτε άλλου λογισμικού.

## Παρατηρήσεις <sup>3</sup>

### Σχετικά με τον κώδικα

Οι φοιτητές κατανόησαν πως κατά τον πολλαπλασιασμό απαιτούνται περισσότεροι πόροι συστήματος, λόγω της πολυπλοκότητας αυτού και έτσι επικεντρώθηκαν σε αυτή την πράξη – διεργασία ώστε να πετύχουν τον βέλτιστο δυνατό χρόνο εκτέλεσης της διεργασίας.

Έτσι στην έκδοση του προγράμματος που κατατίθεται, μεγάλο μέρος των πράξεων δεν χρησιμοποιούν το παράλληλο προγραμματισμό καθώς δεν παρατηρείται σημαντική διαφορά στον χρόνο απόκρισης.

### Σχετικά με την λειτουργία της ομάδας

Λόγω των συνθηκών κατά των οποίων εκπονήθηκε η ομαδική εργασία (πανδημία, μη εφικτής δια ζώσης επικοινωνία, μη προσωπική γνωριμία μεταξύ των φοιτητών, διαφορετικές πόλεις κατοικίας) κατέστη αρκετά δύσκολη η εκπόνηση της εργασίας. Πάραυτα η συνεργασία μεταξύ των φοιτητών ήταν σίγουρα ικανοποιητική.

### Σχετικά με το παραχθέν πρόγραμμα

Οι φοιτητές παρήγαγαν αρκετά αρχεία κώδικα μέχρι την τελική σύνθεση, στην προσπάθεια το τελικό πρόγραμμα να χαρακτηρίζεται από ταχύτητα και απλότητα. Κάθε επιμέρους προσπάθεια συντέλεσε στην δημιουργία του τελικού προγράμματος. Έτσι μπορούμε να πούμε πως το τελικό πρόγραμμα δεν ανταποκρίνεται στην συνολική εργασία όλων των μελών καθώς αρκετή δουλειά αυτών δεν χρησιμοποιήθηκε ή η χρήση της δεν είναι άμεσα αντιληπτή (πχ συμβολή με την υπόδειξη κατάλληλων συναρτήσεων, εναλλακτικών προτάσεων κ.α.).

## Ανακατευθύνσεις

- 1: Η βιβλιοθήκη NumPy αποτελεί ένα λογισμικό ανοιχτού κώδικα (open-source software), γραμμένο σε Python και C ώστε να εξασφαλίσει τόσο ταχύτητα όσο και ευκολία στις πράξεις των Πινάκων
- 2: Για παράδειγμα για την γραφική απεικόνιση αρχικά είχαν χρησιμοποιηθεί οι Βιβλιοθήκες tkinter και turtle αλλά κατά την τελική σύνθεση διαπιστώθηκε πως δεν απαιτούνται και οι δύο, και έτσι επιλέχθηκε μόνο η tkinter
- 3: Οι παρατηρήσεις αυτές αποτελούν παρατηρήσεις του συντάκτη με τις οποίες ήταν σύμφωνοι όλοι οι εμπλεκόμενοι φοιτητές.

## Εγκατάσταση του προγράμματος

Το παρόν αποτελεί ομαδική εργασία των Γιακουμέλου Αιμιλία, Ντάγκας Αλέξανδρος, Ντεν- Μπραμπερ Βερνάρδος- Αδριανός, Παπουτσάς Γεώργιος, Ροδόπουλος Γεώργιος και Ψημμένος Επαμεινώνδας στα πλαίσια του μαθήματος “Εισαγωγή στους Υπολογιστές”. Επιβλέπων καθηγητής: Παλιουράς Βασίλης.

### Αναλυτικές οδηγίες Εγκατάστασης και εκτέλεσης του προγράμματος

Αρχικά αποθηκεύστε στον υπολογιστή σας το αρχείο 18.zip . Προχωρήστε έπειτα στην αποσυμπίεση ολόκληρου του φακέλου.

Το παρόν προϋποθέτει πως έχετε προ εγκατεστημένη στον υπολογιστή σας την Python 3.8 ή νεότερη. Αν δεν την έχετε ήδη εγκαταστημένη μπορείτε να κατεβάσετε την τελευταία έκδοση από εδώ: <https://www.python.org/downloads/>

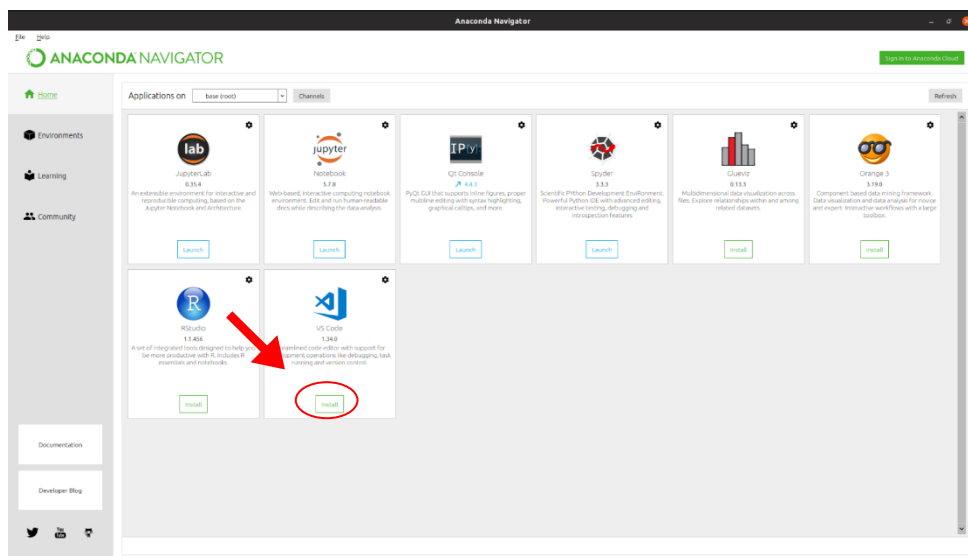
Στον κώδικα χρησιμοποιούνται βιβλιοθήκες της Python οι οποίες δεν είναι προ εγκατεστημένες με αυτή. Ως εκ τούτου, για την εκτέλεση του προγράμματος απαιτείται είτε να εγκαταστήσετε τις βιβλιοθήκες NumPy και PIL χειροκίνητα, είτε να χρησιμοποιήσετε εφαρμογές όπως το anaconda που τις περιέχουν.

Προτείνουμε την χρήση της εφαρμογής anaconda ως ευκολότερη. (Για την χειροκίνητη εγκατάσταση ανατρέξτε στο τέλος του κεφαλαίου).

Αν δεν έχετε εγκατεστημένο το anaconda, μπορείτε να το κατεβάσετε και να το εγκαταστήσετε ακολουθώντας τον σύνδεσμο <https://www.anaconda.com/products/individual#Downloads>.

Αφού εγκαταστήσετε το anaconda, εκτελέστε το πρόγραμμα Anaconda Navigator (εγκαθίσταται μαζί με το anaconda). Ανοίξτε όποιο code editor χρησιμοποιείτε (προτείνουμε και αναλύουμε στην συνέχεια το VS Code).

Αν το VS Code δεν είναι εγκατεστημένο στον υπολογιστή σας πατήστε Install όπως φαίνεται παρακάτω:



Εικόνα 9.1: Anaconda Navigator

Έπειτα εκτελέστε το VS Code μέσα από το Anaconda navigator. Ανοίξτε το αρχείο code18.py -που βρίσκετε στον φάκελο που αποσυμπιέσατε- (είτε με τον συνδυασμό των πλήκτρων Ctrl + o , είτε ακολουθώντας την διαδρομή File > Open File).

Τέλος εκτελέστε το πρόγραμμα με τον συνδυασμό πλήκτρων Ctrl + F5.

Με την εκτέλεση του προγράμματος ανοίγει ένα νέο παράθυρο στην οθόνη σας, και μπορείτε πλέον να χρησιμοποιήσετε το πρόγραμμα.

Αν θέλετε να εγκαταστήσετε χειροκίνητα τις απαιτούμενες βιβλιοθήκες, ακολουθήστε τα παρακάτω:

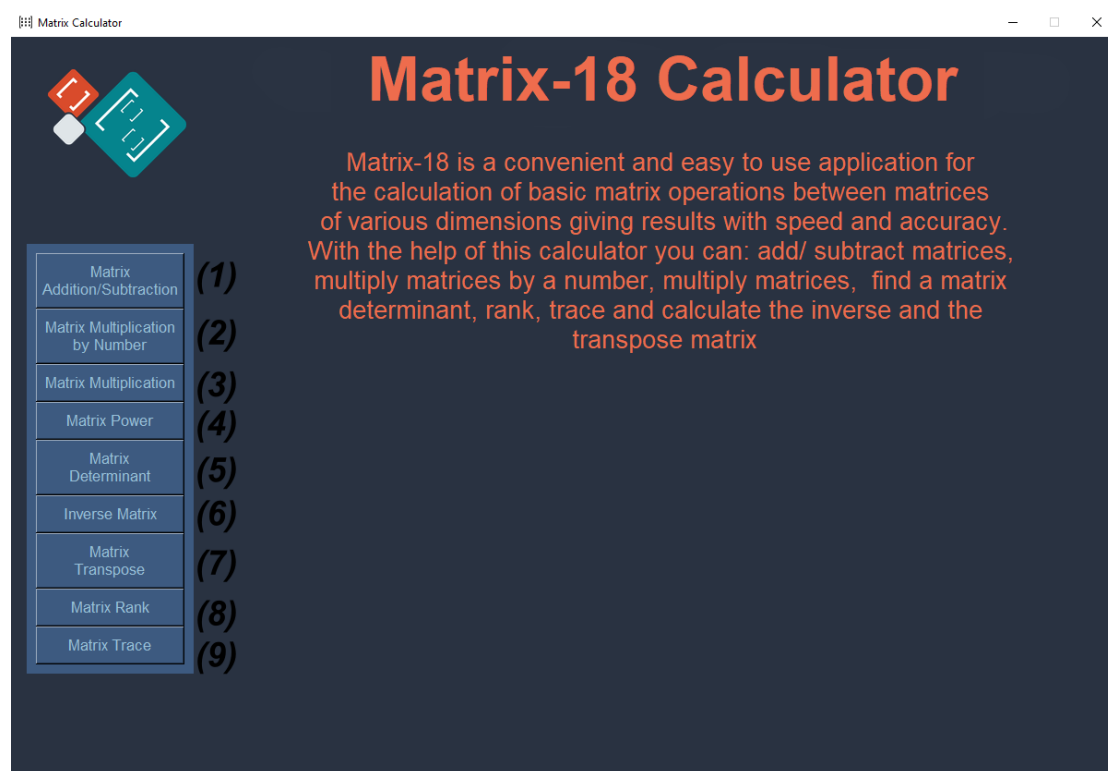
Στο τερματικό σας (terminal) (με την προϋπόθεση ότι χρησιμοποιείτε το pip), μπορείτε να εγκαταστήσετε τις βιβλιοθήκες NumPy, PIL, με την παρακάτω εντολή:

```
pip install numpy, Pillow
```

Μετά την εγκατάσταση των απαιτούμενων βιβλιοθηκών, μπορείτε να εκτελέσετε το πρόγραμμα code18.py από οποιονδήποτε code editor διαθέτετε.

## Παραδείγματα χρήσης

Με την εκτέλεση του προγράμματος από τον code editor, εμφανίζεται στην οθόνη του χρήστη το «Αρχικό Παράθυρο Πλοήγησης»





- (1) Πρόσθεση και Αφαίρεση Πινάκων
- (2) Πολλαπλασιασμός Πίνακα με Αριθμό
- (3) Πολλαπλασιασμός Πινάκων
- (4) Ύψωση Πίνακα σε Δύναμη
- (5) Εύρεση ορίζουσας Πίνακα
- (6) Εύρεση Αντίστροφου Πίνακα
- (7) Εύρεση Ανάστροφου Πίνακα
- (8) Εύρεση Τάξης Πίνακα
- (9) Εύρεση Ίχνους Πίνακα

Σε κάθε επιλογή εμφανίζεται, ανάλογα με την πράξη ένα παράθυρο (window) της μορφής:

The screenshot shows a web application titled "Matrix Multiplication Calculator". On the left is a sidebar menu with buttons for: Matrix Addition/Subtraction, Matrix Multiplication by Number, Matrix Multiplication, Matrix Power, Matrix Determinant, Inverse Matrix, Matrix Transpose, Matrix Rank, and Matrix Trace. The main area has a dark blue background. At the top left is a logo with three overlapping squares. To the right of the logo is the title "Matrix Multiplication Calculator" in orange. Below the title is a text block explaining matrix multiplication: "Matrix Multiplication is a binary operation that produces a matrix from two matrices. For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix. The resulting matrix, known as the matrix product, has the number of rows of the first and the number of columns of the second matrix." Below this text are two sections: "Custom Matrices" and "Random Matrices". Each section contains input fields for "Matrix A dimensions" and "Matrix B dimensions" (both set to 2x2) and a button ("Set matrices" for custom, "Calculate" for random).

Η ομάδα φοιτητών, θέλοντας τόσο να δείξει τις δυνατότητες του multiprocessing όσο και να προσφέρει ένα πρόγραμμα εύχρηστο στο ευρύ κοινό, αποφάσισε την δημιουργία διπλού μενού χρήστη.

Από την μία στο πλαίσιο “*Custom Matrices*” ο χρήστης μπορεί να εισάγει όλες τις πληροφορίες, μόνος του καθιστώντας έτσι το πρόγραμμα λειτουργικό. Από την άλλη μία στο πλαίσιο “*Random Matrices*”, ο χρήστης εισάγει μέρος των πληροφοριών και το πρόγραμμα εισάγει τυχαία στοιχεία (Υπάρχει δυνατότητα αποτυπώσεις αυτών στο Terminal).

Στο “*Custom Matrices*” ο χρήστης δύναται να επιλέξει μέσω αναπτυσσόμενου μενού (dropdown menu), τις διαστάσεις των πινάκων (ή αντίστοιχα του πίνακα, για διεργασίες που απαιτούν μόνο έναν πίνακα).

Για την αποφυγή λαθών το πρόγραμμα “επιβάλλει” να ισχύουν ορισμένες προϋποθέσεις ώστε να είναι εφικτές οι εκάστοτε διεργασίες. Παραδείγματος χάρη, στον πολλαπλασιασμό πινάκων απαιτείται ο αριθμός των στηλών του 1<sup>ου</sup> πίνακα να συνάδει με τον αριθμό των γραμμών του 2<sup>ου</sup>.

Αφού ο χρήστης εισάγει τις διαστάσεις του πίνακα (και όποια άλλη πληροφορία ζητείται -π.χ. στην διεργασία « (4) Ύψωση Πίνακα σε Δύναμη» απαιτείται η πληκτρολόγηση της δύναμης-), εμφανίζεται ένα νέο παράθυρο (window), προσαρμοσμένο κατάλληλα τόσο στην εκάστοτε διεργασία, όσο και στις διαστάσεις του πίνακα (ή των πινάκων) όπως αυτές δόθηκαν στο κύριο παράθυρο.

The screenshot shows the 'Matrix Calculator' application window. It features a dark blue theme. On the left, under 'Input matrix A:', there is a 2x8 grid of input fields. Below this grid are buttons for 'Clear', 'Fill with 0's', 'Fill with 1's', 'Save to memory', and 'Load from memory'. On the right, under 'Input matrix B:', there is an 8x2 grid of input fields. Below this grid are buttons for 'Clear', 'Fill with 0's', 'Fill with 1's', 'Save to memory', and 'Load from memory'. A large 'Calculate' button is positioned at the bottom center of the window.

#### Επεξήγηση δυνατοτήτων – buttons στο παράθυρο καταχώρησης πίνακα

Με το button “**Clear**” διαγράφονται όλες οι καταχωρήσεις στα κελιά του πίνακα

Με το button “**Fill with 0's**” σε κάθε κενό κελί του πίνακα εκχωρείται η τιμή 0 (δεν είναι αναγκαία η χρήση του καθώς από προεπιλογή τα κενά κελιά νοείται πως περιέχουν την τιμή 0)

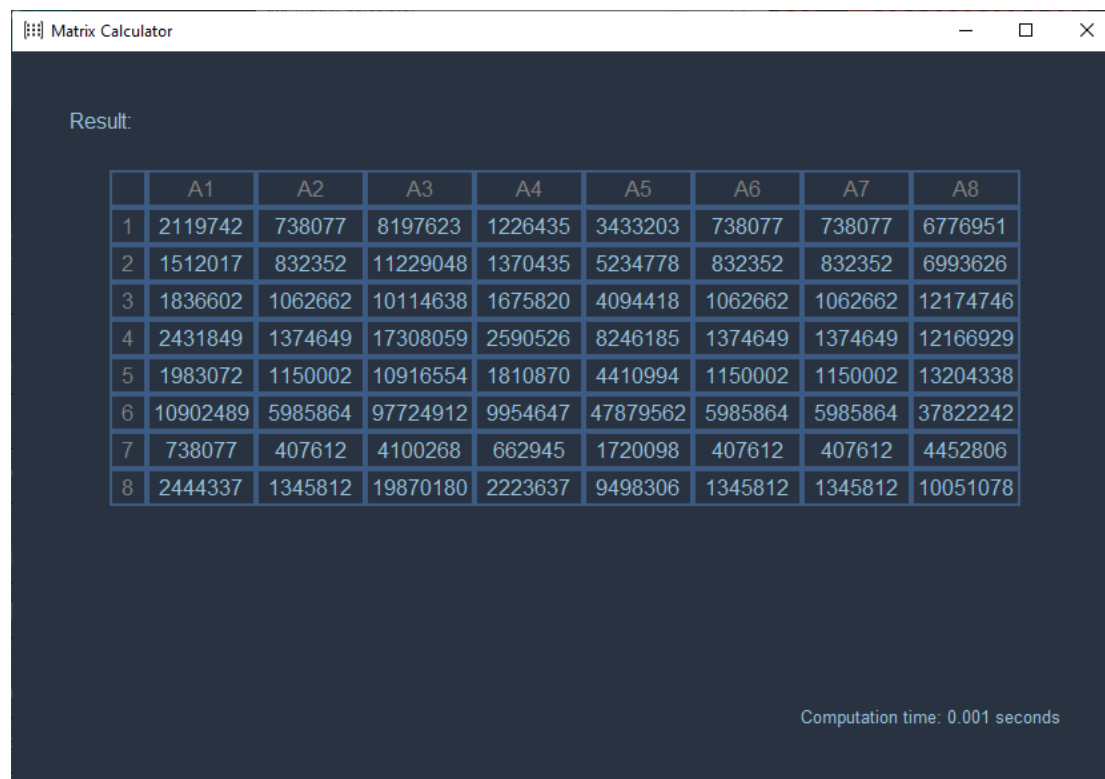
Με το button “**Fill with 1's**” σε κάθε κενό κελί του πίνακα εκχωρείται η τιμή 1

Με το button “**Save to memory**” ο πίνακας που έχει εισαχθεί, καταχωρείται στην μνήμη του προγράμματος.

Με το button **“Load from memory”** εισάγεται ο αποθηκευμένος στην μνήμη προγράμματος πίνακας, εφόσον αυτό είναι εφικτό.

Τέλος με το button **“Calculate”** εκτελείτε η επιλεγμένη διεργασία – πράξη.

Με την εκτέλεση της επιλεγμένης διεργασίας – πράξης δημιουργείτε ένα νέο παράθυρο στο οποίο φαίνεται το αποτέλεσμα της πράξης



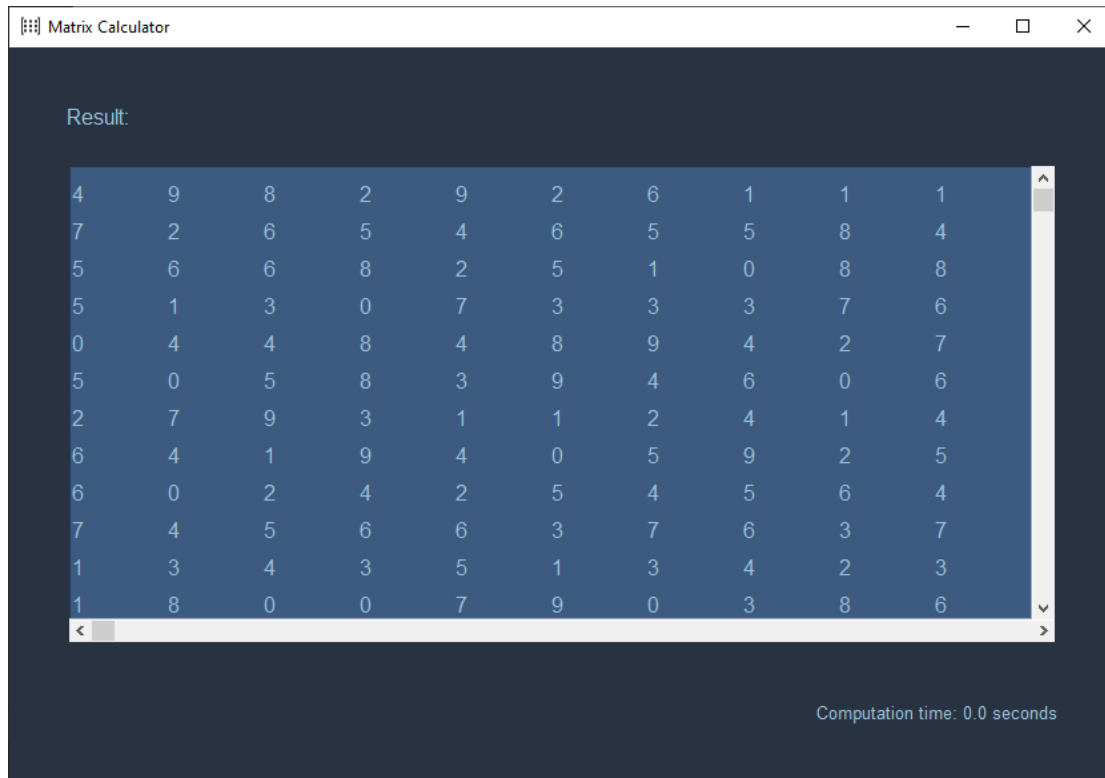
The screenshot shows a window titled "Matrix Calculator" with a dark blue background. Under the heading "Result:", there is an 8x8 matrix displayed in a table with a light blue grid. The columns are labeled A1 through A8, and the rows are numbered 1 through 8. The matrix contains numerical values. At the bottom right of the window, it says "Computation time: 0.001 seconds".

	A1	A2	A3	A4	A5	A6	A7	A8
1	2119742	738077	8197623	1226435	3433203	738077	738077	6776951
2	1512017	832352	11229048	1370435	5234778	832352	832352	6993626
3	1836602	1062662	10114638	1675820	4094418	1062662	1062662	12174746
4	2431849	1374649	17308059	2590526	8246185	1374649	1374649	12166929
5	1983072	1150002	10916554	1810870	4410994	1150002	1150002	13204338
6	10902489	5985864	97724912	9954647	47879562	5985864	5985864	37822242
7	738077	407612	4100268	662945	1720098	407612	407612	4452806
8	2444337	1345812	19870180	2223637	9498306	1345812	1345812	10051078

Στο κάτω μέρος του οποίου φαίνεται και ο χρόνος υπολογισμού της πράξης από το πρόγραμμα.

Η φιλοσοφία του **“Random Matrices”** είναι σχεδόν η ίδια με αυτή του **“Custom Matrices”**.

Αυτό που αλλάζει είναι ότι το παράθυρο όπου ο χρήστης εκχωρεί τα στοιχεία πίνακα δεν υπάρχει, καθώς ο πίνακας είναι - προκαθορισμένων από τον χρήστη διαστάσεων - τυχαίος. Έτσι εμφανίζεται κατευθείαν το αποτέλεσμα και ο βέλτιστος χρόνος υπολογισμού (σε περίπτωση που η πράξη εκτελεστεί με multiprocessing στο terminal τυπώνονται και οι δύο χρόνοι).



The screenshot shows a window titled "Matrix Calculator" with a dark blue background. It displays a 10x10 matrix of integers. The window has standard OS controls (minimize, maximize, close) in the top right corner. Below the matrix, it indicates "Computation time: 0.0 seconds".

Result:									
4	9	8	2	9	2	6	1	1	1
7	2	6	5	4	6	5	5	8	4
5	6	6	8	2	5	1	0	8	8
5	1	3	0	7	3	3	3	7	6
0	4	4	8	4	8	9	4	2	7
5	0	5	8	3	9	4	6	0	6
2	7	9	3	1	1	2	4	1	4
6	4	1	9	4	0	5	9	2	5
6	0	2	4	2	5	4	5	6	4
7	4	5	6	6	3	7	6	3	7
1	3	4	3	5	1	3	4	2	3
1	8	0	0	7	9	0	3	8	6

Computation time: 0.0 seconds

## Ο τελικός κώδικας

```
0001 from tkinter import *
0002 from tkinter import ttk
0003 from PIL import Image, ImageTk
0004 import numpy as np
0005 import multiprocessing as mp
0006 import time
0007 from tkinter import messagebox as mb
0008
0009
0010 class RandomMatrix:
0011     """This class contains methods that create either 1 or 2 random matrices
0012     with given dimensions whose elements are integers between 0 and 10"""
0013     @staticmethod
0014     def random_matrix(a, b=0):
0015         """Creates a random matrix with dimensions a x b"""
0016         matrix = np.random.randint(10, size=(a, b))
0017         return matrix
0018
0019     @staticmethod
0020     def two_random_matrices(a, b=0, c=0):
0021         """Creates a random matrix with dimensions a x b and another one with
0022         dimensions b x c"""
0023         matrix_A = np.random.randint(10, size=(a, b))
0024         matrix_B = np.random.randint(10, size=(b, c))
0025         return matrix_A, matrix_B
0026
0027
```

```
0028     class SimpleCalculation:
0029         """This class contains methods that perform basic linear algebra
0030         calculations (without the help of multiprocessing)"""
0031         @staticmethod
0032         def matrix_add(matrix_A, matrix_B):
0033             return matrix_A + matrix_B
0034
0035         @staticmethod
0036         def matrix_sub(matrix_A, matrix_B):
0037             return matrix_A - matrix_B
0038
0039         @staticmethod
0040         def matrix_mul_num(matrix, number):
0041             return matrix * float(number)
0042
0043         @staticmethod
0044         def matrix_mul(matrix_A, matrix_B):
0045             return np.matmul(matrix_A, matrix_B)
0046
0047         @staticmethod
0048         def matrix_power(matrix, power):
0049             return np.linalg.matrix_power(matrix, power)
0050
0051         @staticmethod
0052         def matrix_det(matrix):
0053             return np.linalg.det(matrix)
0054
0055         @staticmethod
0056         def matrix_inv(matrix):
0057             return np.linalg.inv(matrix)
0058
0059         @staticmethod
0060         def matrix_trans(matrix):
0061             return np.transpose(matrix)
0062
0063         @staticmethod
0064         def matrix_rank(matrix):
0065             return np.linalg.matrix_rank(matrix)
0066
0067         @staticmethod
0068         def matrix_trace(matrix):
0069             return np.trace(matrix)
0070
0071
0072     class MultiprocessingCalculation:
0073         """This class contains methods that calculate the product of two matrices
0074         as well as the matrix raised to some power using multiple simultaneous
0075         processes that decrease the computation time"""
0076         @staticmethod
0077         def block_shaped(arr):
0078             """Divides a square array into 4 equal arrays"""
0079             nrc = int(arr.shape[0] / 2)
0080             return arr.reshape(2, nrc, -1, nrc).swapaxes(1, 2).reshape(-1, nrc,
0081 nrc)
0082
0083         @staticmethod
0084         def simple_mul(arrA, arrB, pos, return_dict):
0085             """Calculates the product of two arrays and inserts it into a
dictionary"""
0085             rtn = np.matmul(arrA, arrB)
```

```
0086         return_dict[pos] = rtn
0087
0088     @staticmethod
0089     def array_sum(return_dict):
0090         """Calculates the sum of two products of arrays"""
0091         c11 = dict(return_dict)['11'] + dict(return_dict)['23']
0092         c12 = dict(return_dict)['12'] + dict(return_dict)['24']
0093         c21 = dict(return_dict)['31'] + dict(return_dict)['43']
0094         c22 = dict(return_dict)['32'] + dict(return_dict)['44']
0095         return c11, c12, c21, c22
0096
0097     @staticmethod
0098     def connect(c11, c12, c21, c22):
0099         """Concatenates the four sub-arrays into the final array"""
0100         above = np.concatenate((c11, c12), axis=1)
0101         below = np.concatenate((c21, c22), axis=1)
0102         final = np.concatenate((above, below), axis=0)
0103         return final
0104
0105     @staticmethod
0106     def zero_pad(arr):
0107         """Appends a row and a column of zeros into the array"""
0108         dim = arr.shape[0]
0109         zero1 = np.zeros((dim, 1), dtype='int32')
0110         arr = np.concatenate((arr, zero1), axis=1)
0111         zero2 = np.zeros((1, dim + 1), dtype='int32')
0112         arr = np.concatenate((arr, zero2), axis=0)
0113         return arr
0114
0115     @staticmethod
0116     def zero_pad_row(arr):
0117         """Appends a row of zeros into the array"""
0118         columns = arr.shape[1]
0119         zero = np.zeros((1, columns), dtype='int32')
0120         return np.concatenate((arr, zero), axis=0)
0121
0122     @staticmethod
0123     def zero_pad_col(arr):
0124         """Appends a column of zeros into the array"""
0125         rows = arr.shape[0]
0126         zero = np.zeros((rows, 1), dtype='int32')
0127         return np.concatenate((arr, zero), axis=1)
0128
0129     @staticmethod
0130     def a_split_mul(arrA, arrB):
0131         """Splits array A into two equal arrays and performs the
multiplication
0132         between the two equal arrays and array B"""
0133         a1, a2 = np.vsplit(arrA, 2)
0134
0135         manager = mp.Manager()
0136         return_dict = manager.dict()
0137
0138         pool = mp.Pool()
0139         pool.starmap(MultiprocessingCalculation.simple_mul, [(a1, arrB, '1',
return_dict),
0140                                                                (a2, arrB, '2',
return_dict), ])
0141         pool.close()
0142         pool.join()
```

```
0143
0144         return np.concatenate((dict(return_dict) ['1'],
dict(return_dict) ['2']), axis=0)
0145
0146     @staticmethod
0147     def b_split_mul(arrA, arrB):
0148         """Splits array B into two equal arrays and performs the
multiplication
0149         between the two equal arrays and array A"""
0150         b1, b2 = np.hsplit(arrB, 2)
0151
0152         manager = mp.Manager()
0153         return_dict = manager.dict()
0154
0155         pool = mp.Pool()
0156         pool.starmap(MultiprocessingCalculation.simple_mul, [(arrA, b1, '1',
return_dict),
0157                                                                 (arrA, b2, '2',
return_dict), ])
0158         pool.close()
0159         pool.join()
0160
0161         return np.concatenate((dict(return_dict) ['1'],
dict(return_dict) ['2']), axis=1)
0162
0163     @staticmethod
0164     def both_split_mul(arrA, arrB):
0165         """Splits both arrays into four equal arrays and performs the
multiplication between the four equal arrays"""
0166
0167         a1, a2 = np.hsplit(arrA, 2)
0168         b1, b2 = np.vsplit(arrB, 2)
0169
0170         manager = mp.Manager()
0171         return_dict = manager.dict()
0172
0173         pool = mp.Pool()
0174         pool.starmap(MultiprocessingCalculation.simple_mul, [(a1, b1, '1',
return_dict), (a2, b2, '2', return_dict), ])
0175         pool.close()
0176         pool.join()
0177
0178         return dict(return_dict) ['1'] + dict(return_dict) ['2']
0179
0180     @staticmethod
0181     def square_mul(arrA, arrB):
0182         """Splits both arrays into eight equal arrays and performs the
multiplication between the eight equal arrays"""
0183
0184         a11, a12, a21, a22 = MultiprocessingCalculation.block_shaped(arrA)
0185         b11, b12, b21, b22 = MultiprocessingCalculation.block_shaped(arrB)
0186
0187         manager = mp.Manager()
0188         return_dict = manager.dict()
0189
0190         pool = mp.Pool()
0191         pool.starmap(MultiprocessingCalculation.simple_mul,
0192                     [(a11, b11, '11', return_dict), (a12, b21, '23',
return_dict),
0193                     (a11, b12, '12', return_dict), (a12, b22, '24',
return_dict),
0194                     (a21, b11, '31', return_dict), (a22, b21, '43',
```

```
return_dict),
0195             (a21, b12, '32', return_dict), (a22, b22, '44',
return_dict), ])
0196     pool.close()
0197     pool.join()
0198
0199     c11, c12, c21, c22 = MultiprocessingCalculation.array_sum(return_dict)
0200     result = MultiprocessingCalculation.connect(c11, c12, c21, c22)
0201
0202     return result
0203
0204     @staticmethod
0205     def multiplication(arrA, arrB):
0206         """Carries out various checks and calls the according methods"""
0207         a = arrA.shape[0]
0208         b = arrA.shape[1]
0209         c = arrB.shape[1]
0210
0211         if a == b == c: # Square matrix
0212             if a % 2 != 0: # Check if Matrix's dimensions are odd numbers
0213                 arrA = MultiprocessingCalculation.zero_pad(arrA)
0214                 arrB = MultiprocessingCalculation.zero_pad(arrB)
0215                 result = MultiprocessingCalculation.square_mul(arrA, arrB)
0216                 result = np.delete(result, -1, 0)
0217                 result = np.delete(result, -1, 1)
0218                 return result
0219             else:
0220                 result = MultiprocessingCalculation.square_mul(arrA, arrB)
0221                 return result
0222
0223         elif max(a, b, c) == a: # Matrix A's rows is the largest dimension
0224             if a % 2 != 0: # Check if Matrix A's rows is an odd number
0225                 arrA = MultiprocessingCalculation.zero_pad_row(arrA)
0226                 result = MultiprocessingCalculation.a_split_mul(arrA, arrB)
0227                 result = np.delete(result, -1, 0)
0228                 return result
0229             else:
0230                 result = MultiprocessingCalculation.a_split_mul(arrA, arrB)
0231                 return result
0232
0233         elif max(a, b, c) == c: # Matrix B's columns is the largest dimension
0234             if c % 2 != 0: # Check if Matrix B's columns is an odd number
0235                 arrB = MultiprocessingCalculation.zero_pad_col(arrB)
0236                 result = MultiprocessingCalculation.b_split_mul(arrA, arrB)
0237                 result = np.delete(result, -1, 1)
0238                 return result
0239             else:
0240                 result = MultiprocessingCalculation.b_split_mul(arrA, arrB)
0241                 return result
0242
0243         elif max(a, b, c) == b: # Matrix A's columns (and Matrix B's rows) is
the largest dimension
0244             if b % 2 != 0: # Check if Matrix A's columns (and Matrix B's
rows) is an odd number
0245                 arrA = MultiprocessingCalculation.zero_pad_col(arrA)
0246                 arrB = MultiprocessingCalculation.zero_pad_row(arrB)
0247             if a % 2 != 0: # Check if Matrix A's rows is an odd number
0248                 arrA = MultiprocessingCalculation.zero_pad_row(arrA)
0249             if c % 2 != 0: # Check if Matrix B's columns is an odd number
0250                 arrB = MultiprocessingCalculation.zero_pad_col(arrB)
```



```
0251
0252         result = MultiprocessingCalculation.both_split_mul(arrA, arrB)
0253
0254         rows, columns = result.shape
0255         if rows == a + 1:
0256             result = np.delete(result, -1, 0)
0257         if columns == c + 1:
0258             result = np.delete(result, -1, 1)
0259
0260         return result
0261
0262     @staticmethod
0263     def matrix_power(matrix, power):
0264         """Calculates the matrix raised to some positive integer"""
0265         if power % 2 == 0: # Power is an even number
0266             matrix_square = MultiprocessingCalculation.multiplication(matrix,
matrix)
0267             matrix2 = matrix_square
0268             for i in range(0, power - 1, 2):
0269                 matrix2 =
MultiprocessingCalculation.multiplication(matrix_square, matrix2)
0270             return matrix2
0271
0272         elif power % 2 != 0: # Power is an odd number
0273             matrix_square = MultiprocessingCalculation.multiplication(matrix,
matrix)
0274             matrix2 = matrix_square
0275             for i in range(0, power - 2, 2):
0276                 matrix2 =
MultiprocessingCalculation.multiplication(matrix_square, matrix2)
0277             return MultiprocessingCalculation.multiplication(matrix, matrix2)
0278
0279
0280     class GUI:
0281         def __init__(self, root):
0282             self.root = root
0283             self.root.iconbitmap('matrix_ico.ico')
0284             self.color_bg1 = '#293241' #
0285             self.color_bg2 = '#3d5a80' #
0286             self.color_button1 = '#3d5a80' #
0287             self.color_button2 = '#d8bc66' # Color palette
0288             self.color_text1 = '#ee6c4d' #
0289             self.color_text2 = '#98c1d9' #
0290             self.color_text3 = '#ffffff' #
0291             self.root.geometry('1200x800')
0292             self.root.resizable(False, False)
0293             self.root.title('Matrix Calculator')
0294             self.dim_values = [2, 3, 4, 5, 6, 7, 8, 9, 10]
0295             self.f_left = Frame(self.root, bg=self.color_bg1) # Frame containing
button frame and logo
0296             self.f_left.pack(side='left', fill='y')
0297             self.logo = Canvas(self.f_left, bg=self.color_bg1,
highlightbackground=self.color_bg1, width=212, height=150) # Canvas containing upper
left logo
0298             self.logo.grid(row=0, column=0, sticky='N', pady=20)
0299             self.f_buttons = Frame(self.f_left, bg=self.color_bg2, padx=10,
pady=10) # Frame containing function buttons
0300             self.f_buttons.grid(row=1, column=0, pady=30)
0301             global photo
0302             photo = Image.open("logo.png").resize((150, 117), Image.ANTIALIAS)
```

```
0303         photo = ImageTk.PhotoImage(photo)
0304         self.logo.create_image(40, 15, image=photo, anchor=NW)
0305
0306         # Creating function buttons
0307         self.b_add_sub = Button(self.f_buttons,
text='Matrix\nAddition/Subtraction', font=('Arial', 13),
0308                                bg=self.color_button1, fg=self.color_text2,
activebackground=self.color_button2,
0309                                activeforeground=self.color_text2, pady=5,
command=lambda: GUI.add_sub(self))
0310         self.b_mul_num = Button(self.f_buttons, text='Matrix
Multiplication\nby Number', font=('Arial', 13),
0311                                bg=self.color_button1, fg=self.color_text2,
activebackground=self.color_button2,
0312                                activeforeground=self.color_text2, pady=5,
command=lambda: GUI.mul_num(self))
0313         self.b_mul = Button(self.f_buttons, text='Matrix Multiplication',
font=('Arial', 13), bg=self.color_button1,
0314                                fg=self.color_text2,
activebackground=self.color_button2, activeforeground=self.color_text2,
0315                                pady=5, command=lambda: GUI.mul(self))
0316         self.b_power = Button(self.f_buttons, text='Matrix Power',
font=('Arial', 13), bg=self.color_button1,
0317                                fg=self.color_text2,
activebackground=self.color_button2,
0318                                activeforeground=self.color_text2, pady=5,
command=lambda: GUI.power(self))
0319         self.b_det = Button(self.f_buttons, text='Matrix\nDeterminant',
font=('Arial', 13), bg=self.color_button1,
0320                                fg=self.color_text2,
activebackground=self.color_button2, activeforeground=self.color_text2,
0321                                pady=5, command=lambda: GUI.det(self))
0322         self.b_inv = Button(self.f_buttons, text='Inverse Matrix',
font=('Arial', 13), bg=self.color_button1,
0323                                fg=self.color_text2,
activebackground=self.color_button2, activeforeground=self.color_text2,
0324                                pady=5, command=lambda: GUI.inv(self))
0325         self.b_trans = Button(self.f_buttons, text='Matrix\nTranspose',
font=('Arial', 13), bg=self.color_button1,
0326                                fg=self.color_text2,
activebackground=self.color_button2,
0327                                activeforeground=self.color_text2, pady=5,
command=lambda: GUI.trans(self))
0328         self.b_rank = Button(self.f_buttons, text='Matrix Rank',
font=('Arial', 13), bg=self.color_button1,
0329                                fg=self.color_text2,
activebackground=self.color_button2,
0330                                activeforeground=self.color_text2, pady=5,
command=lambda: GUI.rank(self))
0331         self.b_trace = Button(self.f_buttons, text='Matrix Trace',
font=('Arial', 13), bg=self.color_button1,
0332                                fg=self.color_text2,
activebackground=self.color_button2,
0333                                activeforeground=self.color_text2, pady=5,
command=lambda: GUI.trace(self))
0334
0335         # Packing function buttons
0336         self.b_add_sub.pack(fill='x')
0337         self.b_mul_num.pack(fill='x')
0338         self.b_mul.pack(fill='x')
```

```
0339         self.b_power.pack(fill='x')
0340         self.b_det.pack(fill='x')
0341         self.b_inv.pack(fill='x')
0342         self.b_trans.pack(fill='x')
0343         self.b_rank.pack(fill='x')
0344         self.b_trace.pack(fill='x')
0345
0346         self.f_main = Frame(self.root, bg=self.color_bg1) # Frame containing
func descriptions and dimensions selection
0347         self.f_main.pack(side='left', expand=True, fill='both')
0348
0349         self.title = Label(self.f_main, text='Linear algebra applications',
font=('Arial', 50, 'bold'),
0350                                bg=self.color_bg1, fg=self.color_text1) # Main
title
0351         self.title.pack()
0352         self.info = Label(self.f_main, text=''
0353 Matrix-18 is a convenient and easy to use application for
0354 the calculation of basic matrix operations between matrices
0355 of various dimensions giving results with speed and accuracy.
0356 With the help of this calculator you can: add/ subtract matrices,
0357 multiply matrices by a number, multiply matrices, find a matrix
0358 determinant, rank, trace and calculate the inverse and the
0359 transpose matrix'', font=('Arial', 20), bg=self.color_bg1,
0360                                fg=self.color_text1) # Main app description
0361         self.info.pack()
0362
0363         """Each method below displays the correct widgets on the main (left)
0364 frame for entering the dimensions of the matrix (or matrices),
0365 according to the desired calculation"""
0366         def add_sub(self):
0367             GUI.clear_frame(self)
0368
0369             self.title = Label(self.f_main, text='Matrix Addition and Subtraction
Calculator', font=('Arial', 30, 'bold'),
0370                                bg=self.color_bg1, fg=self.color_text1)
0371             self.title.pack(pady=(30, 0))
0372
0373             self.desc = Label(self.f_main, text=''Matrix addition/subtraction is
the operation of adding/subtracting
0374 two matrices by adding/subtracting the corresponding elements together.'',
font=('Arial', 15),
0375                                bg=self.color_bg1, fg=self.color_text1)
0376             self.desc.pack(pady=(30, 0))
0377
0378             self.f_dims = LabelFrame(self.f_main, text='Custom Matrices',
padx=100, pady=10, bg=self.color_bg1,
0379                                fg=self.color_text3, relief=GROOVE)
0380             self.f_dims.pack(pady=(200, 50))
0381
0382             self.dim_text = Label(self.f_dims, text='Matrices dimension:',
bg=self.color_bg1, fg=self.color_text2)
0383             self.dim_text.grid(row=0, column=0)
0384
0385             self.dim_m = ttk.Combobox(self.f_dims, values=self.dim_values,
width=3, state='readonly')
0386             self.dim_m.current(0)
0387             self.dim_m.grid(row=0, column=1)
0388
0389             self.X_text = Label(self.f_dims, text='X', bg=self.color_bg1,
```

```
fg=self.color_text2)
0390         self.X_text.grid(row=0, column=2)
0391
0392         self.dim_n = ttk.Combobox(self.f_dims, values=self.dim_values,
width=3, state='readonly')
0393         self.dim_n.current(0)
0394         self.dim_n.grid(row=0, column=3)
0395
0396         self.op_text = Label(self.f_dims, text='Operation', bg=self.color_bg1,
fg=self.color_text2)
0397         self.op_text.grid(row=0, column=4, padx=(30, 2))
0398
0399         self.op = ttk.Combobox(self.f_dims, values=['+', '-'], width=2,
state='readonly')
0400         self.op.current(0)
0401         self.op.grid(row=0, column=5)
0402
0403         self.set = Button(self.f_dims, text='Set matrices', bg=self.color_bg1,
fg=self.color_text2,
0404                             activebackground=self.color_bg1,
0405                             activeforeground=self.color_text2, relief=RIDGE,
0406                             command=lambda: GUI.set_matrix(self, 'add_sub',
int(self.dim_m.get()), int(self.dim_n.get())))
0407         self.set.grid(row=0, column=6, padx=(100, 0))
0408
0409         self.f_rand_dims = LabelFrame(self.f_main, text='Random Matrices',
padx=100, pady=10, bg=self.color_bg1,
0410                                         fg=self.color_text3, relief=GROOVE)
0411         self.f_rand_dims.pack()
0412
0413         self.rand_dim_text = Label(self.f_rand_dims, text='Matrices
dimension:', bg=self.color_bg1, fg=self.color_text2)
0414         self.rand_dim_text.grid(row=0, column=0)
0415
0416         self.rand_dim_m = Entry(self.f_rand_dims, width=6)
0417         self.rand_dim_m.delete(0, END)
0418         self.rand_dim_m.insert(0, '2')
0419         self.rand_dim_m.grid(row=0, column=1)
0420
0421         self.rand_X_text = Label(self.f_rand_dims, text='X',
bg=self.color_bg1, fg=self.color_text2)
0422         self.rand_X_text.grid(row=0, column=2)
0423
0424         self.rand_dim_n = Entry(self.f_rand_dims, width=6)
0425         self.rand_dim_n.delete(0, END)
0426         self.rand_dim_n.insert(0, '2')
0427         self.rand_dim_n.grid(row=0, column=3)
0428
0429         self.rand_op_text = Label(self.f_rand_dims, text='Operation',
bg=self.color_bg1, fg=self.color_text2)
0430         self.rand_op_text.grid(row=0, column=4, padx=(30, 2))
0431
0432         self.rand_op = ttk.Combobox(self.f_rand_dims, values=['+', '-'],
width=2)
0433         self.rand_op.current(0)
0434         self.rand_op.grid(row=0, column=5)
0435
0436         self.rand_set = Button(self.f_rand_dims, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=9,
0437                             activebackground=self.color_bg1,
```

```
activeforeground=self.color_text2, relief=RIDGE,
0438             command=lambda: GUI.rand_calculate(self,
'add_sub', self.rand_dim_m.get(),
0439
self.rand_dim_n.get()))
0440         self.rand_set.grid(row=0, column=6, padx=(100, 0))
0441
0442         def mul_num(self):
0443             GUI.clear_frame(self)
0444
0445             self.title = Label(self.f_main, text='Matrix Multiplication By Number
Calculator', font=('Arial', 30, 'bold'),
0446                             bg=self.color_bg1, fg=self.color_text1)
0447             self.title.pack(pady=(30, 0))
0448
0449             self.desc = Label(self.f_main, text=''Matrix Multiplication By Number
is the operation of multiplying
0450             every element of the matrix by a certain number.'', font=('Arial', 15),
bg=self.color_bg1, fg=self.color_text1)
0451             self.desc.pack(pady=(30, 0))
0452
0453             self.f_dims = LabelFrame(self.f_main, text='Custom Matrix', padx=100,
pady=10, bg=self.color_bg1,
0454                                     fg=self.color_text3, relief=GROOVE)
0455             self.f_dims.pack(pady=(200, 50))
0456
0457             self.dim_text = Label(self.f_dims, text='Matrix dimension:',
bg=self.color_bg1, fg=self.color_text2)
0458             self.dim_text.grid(row=0, column=0)
0459
0460             self.dim_m = ttk.Combobox(self.f_dims, values=self.dim_values,
width=3, state='readonly')
0461             self.dim_m.current(0)
0462             self.dim_m.grid(row=0, column=1)
0463
0464             self.X_text = Label(self.f_dims, text='X', bg=self.color_bg1,
fg=self.color_text2)
0465             self.X_text.grid(row=0, column=2)
0466
0467             self.dim_n = ttk.Combobox(self.f_dims, values=self.dim_values,
width=3, state='readonly')
0468             self.dim_n.current(0)
0469             self.dim_n.grid(row=0, column=3)
0470
0471             self.num_text = Label(self.f_dims, text='Multiply by',
bg=self.color_bg1, fg=self.color_text2)
0472             self.num_text.grid(row=0, column=4, padx=(30, 2))
0473
0474             self.num_entry = Entry(self.f_dims, width=3)
0475             self.num_entry.insert(0, '1')
0476             self.num_entry.grid(row=0, column=5)
0477
0478             self.set = Button(self.f_dims, text='Set matrix', bg=self.color_bg1,
fg=self.color_text2,
0479                             activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
0480                             command=lambda: GUI.set_matrix(self, 'mul_num',
int(self.dim_m.get()), int(self.dim_n.get())))
0481             self.set.grid(row=0, column=6, padx=(100, 0))
0482
```

```
0483         self.f_rand_dims = LabelFrame(self.f_main, text='Random Matrix',
padx=100, pady=10, bg=self.color_bg1,
0484                                     fg=self.color_text3, relief=GROOVE)
0485         self.f_rand_dims.pack()
0486
0487         self.rand_dim_text = Label(self.f_rand_dims, text='Matrix dimension:',
bg=self.color_bg1, fg=self.color_text2)
0488         self.rand_dim_text.grid(row=0, column=0)
0489
0490         self.rand_dim_m = Entry(self.f_rand_dims, width=6)
0491         self.rand_dim_m.delete(0, END)
0492         self.rand_dim_m.insert(0, '2')
0493         self.rand_dim_m.grid(row=0, column=1)
0494
0495         self.rand_X_text = Label(self.f_rand_dims, text='X',
bg=self.color_bg1, fg=self.color_text2)
0496         self.rand_X_text.grid(row=0, column=2)
0497
0498         self.rand_dim_n = Entry(self.f_rand_dims, width=6)
0499         self.rand_dim_n.delete(0, END)
0500         self.rand_dim_n.insert(0, '2')
0501         self.rand_dim_n.grid(row=0, column=3)
0502
0503         self.rand_num_text = Label(self.f_rand_dims, text='Multiply by',
bg=self.color_bg1, fg=self.color_text2)
0504         self.rand_num_text.grid(row=0, column=4, padx=(30, 2))
0505
0506         self.rand_num_entry = Entry(self.f_rand_dims, width=3)
0507         self.rand_num_entry.insert(0, '1')
0508         self.rand_num_entry.grid(row=0, column=5)
0509
0510         self.rand_set = Button(self.f_rand_dims, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=3,
0511                               activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
0512                               command=lambda: GUI.rand_calculate(self,
'mul_num', self.rand_dim_m.get(),
0513 self.rand_dim_n.get()))
0514         self.rand_set.grid(row=0, column=6, padx=(100, 0))
0515
0516         def mul(self):
0517             GUI.clear_frame(self)
0518
0519             self.title = Label(self.f_main, text='Matrix Multiplication
Calculator', font=('Arial', 30, 'bold'),
0520                               bg=self.color_bg1, fg=self.color_text1)
0521             self.title.pack(pady=(30, 0))
0522
0523             self.desc = Label(self.f_main, text="'Matrix Multiplication is a
binary operation that produces
0524 a matrix from two matrices. For matrix multiplication,
0525 the number of columns in the first matrix must be equal
0526 to the number of rows in the second matrix. The resulting matrix,
0527 known as the matrix product, has the number of rows
0528 of the first and the number of columns of the second matrix.'",
font=('Arial', 15), bg=self.color_bg1,
0529                               fg=self.color_text1)
0530             self.desc.pack(pady=(30, 0))
0531
```

```
0532         self.f_dims = LabelFrame(self.f_main, text='Custom Matrices',
padx=100, pady=10, bg=self.color_bg1,
0533                                     fg=self.color_text3, relief=GROOVE)
0534         self.f_dims.pack(pady=(200, 50))
0535
0536         self.dimA_text = Label(self.f_dims, text='Matrix A dimensions:',
bg=self.color_bg1, fg=self.color_text2)
0537         self.dimA_text.grid(row=0, column=0, pady=10)
0538         self.dimB_text = Label(self.f_dims, text='Matrix B dimensions:',
bg=self.color_bg1, fg=self.color_text2)
0539         self.dimB_text.grid(row=1, column=0)
0540
0541         def callback(iv):
0542             iv.set(iv.get())
0543
0544         iv1 = IntVar()
0545         iv1.trace("w", lambda name, index, mode, iv1=iv1: callback(iv1))
0546
0547         self.dimA_m = ttk.Combobox(self.f_dims, values=self.dim_values,
width=3, state='readonly')
0548         self.dimA_m.current(0)
0549         self.dimA_m.grid(row=0, column=1)
0550
0551         self.X_text = Label(self.f_dims, text='X', bg=self.color_bg1,
fg=self.color_text2)
0552         self.X_text.grid(row=0, column=2)
0553
0554         self.dimA_n = ttk.Combobox(self.f_dims, textvariable=iv1,
values=self.dim_values, width=3, state='readonly')
0555         self.dimA_n.current(0)
0556         self.dimA_n.grid(row=0, column=3)
0557
0558         self.dimB_m = ttk.Combobox(self.f_dims, textvariable=iv1,
values=self.dim_values, width=3, state='readonly')
0559         self.dimB_m.current(0)
0560         self.dimB_m.grid(row=1, column=1)
0561
0562         self.X_text = Label(self.f_dims, text='X', bg=self.color_bg1,
fg=self.color_text2)
0563         self.X_text.grid(row=1, column=2)
0564
0565         self.dimB_n = ttk.Combobox(self.f_dims, values=self.dim_values,
width=3, state='readonly')
0566         self.dimB_n.current(0)
0567         self.dimB_n.grid(row=1, column=3)
0568
0569         self.set = Button(self.f_dims, text='Set matrices', bg=self.color_bg1,
fg=self.color_text2,
0570                           activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
0571                           command=lambda: GUI.set_matrix(self, 'mul',
int(self.dimA_m.get()), int(self.dimA_n.get()),
0572                           int(self.dimB_n.get())))
0573         self.set.grid(row=0, column=6, rowspan=2, padx=(100, 0))
0574
0575         self.f_rand_dims = LabelFrame(self.f_main, text='Random Matrices',
padx=100, pady=10, bg=self.color_bg1,
0576                                     fg=self.color_text3, relief=GROOVE)
0577         self.f_rand_dims.pack()
```

```
0578
0579         self.rand_dimA_text = Label(self.f_rand_dims, text='Matrix A
dimensions:', bg=self.color_bg1,
0580                                     fg=self.color_text2)
0581         self.rand_dimA_text.grid(row=0, column=0, pady=10)
0582         self.rand_dimB_text = Label(self.f_rand_dims, text='Matrix B
dimensions:', bg=self.color_bg1,
0583                                     fg=self.color_text2)
0584         self.rand_dimB_text.grid(row=1, column=0)
0585
0586         iv2 = StringVar()
0587         iv2.set('2')
0588         iv2.trace("w", lambda name, index, mode, iv2=iv2: callback(iv2))
0589
0590         self.rand_dimA_m = Entry(self.f_rand_dims, width=6)
0591         self.rand_dimA_m.insert(0, '2')
0592         self.rand_dimA_m.grid(row=0, column=1)
0593
0594         self.rand_X_text = Label(self.f_rand_dims, text='X',
bg=self.color_bg1, fg=self.color_text2)
0595         self.rand_X_text.grid(row=0, column=2)
0596
0597         self.rand_dimA_n = Entry(self.f_rand_dims, width=6, textvariable=iv2)
0598         self.rand_dimA_n.grid(row=0, column=3)
0599
0600         self.rand_dimB_m = Entry(self.f_rand_dims, textvariable=iv2, width=6)
0601         self.rand_dimB_m.grid(row=1, column=1)
0602
0603         self.rand_X_text = Label(self.f_rand_dims, text='X',
bg=self.color_bg1, fg=self.color_text2)
0604         self.rand_X_text.grid(row=1, column=2)
0605
0606         self.rand_dimB_n = Entry(self.f_rand_dims, width=6)
0607         self.rand_dimB_n.insert(0, '2')
0608         self.rand_dimB_n.grid(row=1, column=3)
0609
0610         self.rand_set = Button(self.f_rand_dims, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=9,
0611                                activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
0612                                command=lambda: GUI.rand_calculate(self, 'mul',
self.rand_dimA_m.get(),
0613                                self.rand_dimA_n.get(),
0614                                self.rand_dimB_n.get()))
0615         self.rand_set.grid(row=0, column=6, rowspan=2, padx=(100, 0))
0616
0617         def power(self):
0618             GUI.clear_frame(self)
0619
0620             self.title = Label(self.f_main, text='Matrix Power Calculator',
font=('Arial', 30, 'bold'),
0621                                bg=self.color_bg1, fg=self.color_text1)
0622             self.title.pack(pady=(30, 0))
0623
0624             self.desc = Label(self.f_main, text="'Matrix power is obtained by
multiplication matrix by itself 'n' times.
0625             The matrix must be square in order to raise it to a power.'", font=('Arial',
15), bg=self.color_bg1,
```



```
0626             fg=self.color_text1)
0627         self.desc.pack(pady=(30, 0))
0628
0629         self.f_dims = LabelFrame(self.f_main, text='Custom Matrix', padx=100,
0630         pady=10, bg=self.color_bg1,
0631             fg=self.color_text3, relief=GROOVE)
0632         self.f_dims.pack(pady=(200, 50))
0633         self.dim_text = Label(self.f_dims, text='Matrix dimension:',
0634         bg=self.color_bg1, fg=self.color_text2)
0635         self.dim_text.grid(row=0, column=0)
0636         self.dim = ttk.Combobox(self.f_dims, values=self.dim_values, width=3,
0637         state='readonly')
0638         self.dim.current(0)
0639         self.dim.grid(row=0, column=1)
0640         self.power_text = Label(self.f_dims, text='Power', bg=self.color_bg1,
0641         fg=self.color_text2)
0642         self.power_text.grid(row=0, column=4, padx=(30, 2))
0643         self.power_entry = Entry(self.f_dims, width=3)
0644         self.power_entry.insert(0, '1')
0645         self.power_entry.grid(row=0, column=5)
0646
0647         self.set = Button(self.f_dims, text='Set matrix', bg=self.color_bg1,
0648         fg=self.color_text2,
0649             activebackground=self.color_bg1,
0650             activeforeground=self.color_text2, relief=RIDGE,
0651             command=lambda: GUI.set_matrix(self, 'power',
0652             int(self.dim.get())))
0653         self.set.grid(row=0, column=6, padx=(100, 0))
0654
0655         self.f_rand_dims = LabelFrame(self.f_main, text='Random Matrix',
0656         padx=100, pady=10, bg=self.color_bg1,
0657             fg=self.color_text3, relief=GROOVE)
0658         self.f_rand_dims.pack()
0659         self.rand_dim_text = Label(self.f_rand_dims, text='Matrix dimension:',
0660         bg=self.color_bg1, fg=self.color_text2)
0661         self.rand_dim_text.grid(row=0, column=0)
0662         self.rand_dim = Entry(self.f_rand_dims, width=6)
0663         self.rand_dim.delete(0, END)
0664         self.rand_dim.insert(0, '2')
0665         self.rand_dim.grid(row=0, column=1)
0666         self.rand_power_text = Label(self.f_rand_dims, text='Power',
0667         bg=self.color_bg1, fg=self.color_text2)
0668         self.rand_power_text.grid(row=0, column=4, padx=(30, 2))
0669         self.rand_power_entry = Entry(self.f_rand_dims, width=3)
0670         self.rand_power_entry.insert(0, '1')
0671         self.rand_power_entry.grid(row=0, column=5)
0672         self.rand_set = Button(self.f_rand_dims, text='Calculate',
0673         bg=self.color_bg1, fg=self.color_text2, padx=3,
0674             activebackground=self.color_bg1,
0675             activeforeground=self.color_text2, relief=RIDGE,
0676             command=lambda: GUI.rand_calculate(self,
```

```
'power', self.rand_dim.get()))
0674         self.rand_set.grid(row=0, column=6, padx=(100, 0))
0675
0676     def det(self):
0677         GUI.clear_frame(self)
0678
0679         self.title = Label(self.f_main, text='Determinant Calculator',
font=('Arial', 30, 'bold'),
0680                             bg=self.color_bg1, fg=self.color_text1)
0681         self.title.pack(pady=(30, 0))
0682
0683         self.desc = Label(self.f_main, text='''the determinant is a scalar
value
0684         that can be computed from the elements
0685         of a square matrix and encodes certain properties
0686         of the linear transformation described by the matrix.
0687         The determinant of a matrix A is denoted det(A), det A, or |A|.''',
font=('Arial', 15), bg=self.color_bg1,
0688                             fg=self.color_text1)
0689         self.desc.pack(pady=(30, 0))
0690
0691         self.f_dims = LabelFrame(self.f_main, text='Custom Matrix', padx=100,
pady=10, bg=self.color_bg1,
0692                             fg=self.color_text3, relief=GROOVE)
0693         self.f_dims.pack(pady=(200, 50))
0694
0695         self.dim_text = Label(self.f_dims, text='Matrix dimension:',
bg=self.color_bg1, fg=self.color_text2)
0696         self.dim_text.grid(row=0, column=0)
0697
0698         self.dim = ttk.Combobox(self.f_dims, values=self.dim_values, width=3,
state='readonly')
0699         self.dim.current(0)
0700         self.dim.grid(row=0, column=1)
0701
0702         self.set = Button(self.f_dims, text='Set matrix', bg=self.color_bg1,
fg=self.color_text2,
0703                             activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
0704                             command=lambda: GUI.set_matrix(self, 'det',
int(self.dim.get()))))
0705         self.set.grid(row=0, column=6, padx=(100, 0))
0706
0707         self.f_rand_dims = LabelFrame(self.f_main, text='Random Matrix',
padx=100, pady=10, bg=self.color_bg1,
0708                             fg=self.color_text3, relief=GROOVE)
0709         self.f_rand_dims.pack()
0710
0711         self.rand_dim_text = Label(self.f_rand_dims, text='Matrix dimension:',
bg=self.color_bg1, fg=self.color_text2)
0712         self.rand_dim_text.grid(row=0, column=0)
0713
0714         self.rand_dim = Entry(self.f_rand_dims, width=6)
0715         self.rand_dim.insert(0, '2')
0716         self.rand_dim.grid(row=0, column=1)
0717
0718         self.rand_set = Button(self.f_rand_dims, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=3,
0719                             activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
```

```
0720             command=lambda: GUI.rand_calculate(self, 'det',
self.rand_dim.get()))
0721         self.rand_set.grid(row=0, column=6, padx=(100, 0))
0722
0723     def inv(self):
0724         GUI.clear_frame(self)
0725
0726         self.title = Label(self.f_main, text='Inverse Matrix Calculator',
font=('Arial', 30, 'bold'),
0727                             bg=self.color_bg1, fg=self.color_text1)
0728         self.title.pack(pady=(30, 0))
0729
0730         self.desc = Label(self.f_main, text='''In linear algebra, an n-by-n
square matrix A
0731         is called invertible (also nonsingular or nondegenerate),
0732         if there exists an n-by-n square matrix B such that AB=BA=I''', font=('Arial',
15), bg=self.color_bg1,
0733                             fg=self.color_text1)
0734         self.desc.pack(pady=(30, 0))
0735
0736         self.f_dims = LabelFrame(self.f_main, text='Custom Matrix', padx=100,
pady=10, bg=self.color_bg1,
0737                                 fg=self.color_text3, relief=GROOVE)
0738         self.f_dims.pack(pady=(200, 50))
0739
0740         self.dim_text = Label(self.f_dims, text='Matrix dimension:',
bg=self.color_bg1, fg=self.color_text2)
0741         self.dim_text.grid(row=0, column=0)
0742
0743         self.dim = ttk.Combobox(self.f_dims, values=self.dim_values, width=3,
state='readonly')
0744         self.dim.current(0)
0745         self.dim.grid(row=0, column=1)
0746
0747         self.set = Button(self.f_dims, text='Set matrix', bg=self.color_bg1,
fg=self.color_text2,
0748                             activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
0749                             command=lambda: GUI.set_matrix(self, 'inv',
int(self.dim.get()))))
0750         self.set.grid(row=0, column=6, padx=(100, 0))
0751
0752         self.f_rand_dims = LabelFrame(self.f_main, text='Random Matrix',
padx=100, pady=10, bg=self.color_bg1,
0753                                     fg=self.color_text3, relief=GROOVE)
0754         self.f_rand_dims.pack()
0755
0756         self.rand_dim_text = Label(self.f_rand_dims, text='Matrix dimension:',
bg=self.color_bg1, fg=self.color_text2)
0757         self.rand_dim_text.grid(row=0, column=0)
0758
0759         self.rand_dim = Entry(self.f_rand_dims, width=6)
0760         self.rand_dim.insert(0, '2')
0761         self.rand_dim.grid(row=0, column=1)
0762
0763         self.rand_set = Button(self.f_rand_dims, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=3,
0764                             activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
0765                             command=lambda: GUI.rand_calculate(self, 'inv',
```

```
self.rand_dim.get()))
0766         self.rand_set.grid(row=0, column=6, padx=(100, 0))
0767
0768     def trans(self):
0769         GUI.clear_frame(self)
0770
0771         self.title = Label(self.f_main, text='Matrix Transpose Calculator',
font=('Arial', 30, 'bold'),
0772                             bg=self.color_bg1, fg=self.color_text1)
0773         self.title.pack(pady=(30, 0))
0774
0775         self.desc = Label(self.f_main, text='''the transpose of a matrix is an
operator
0776         which flips a matrix over its diagonal;
0777         that is, it switches the row and
0778         column indices of the matrix A
0779         by producing another matrix,
0780         often denoted by AT (among other notations).''', font=('Arial', 15),
bg=self.color_bg1, fg=self.color_text1)
0781         self.desc.pack(pady=(30, 0))
0782
0783         self.f_dims = LabelFrame(self.f_main, text='Custom Matrix', padx=100,
pady=10, bg=self.color_bg1,
0784                                 fg=self.color_text3, relief=GROOVE)
0785         self.f_dims.pack(pady=(200, 50))
0786
0787         self.dim_text = Label(self.f_dims, text='Matrix dimension:',
bg=self.color_bg1, fg=self.color_text2)
0788         self.dim_text.grid(row=0, column=0)
0789
0790         self.dim_m = ttk.Combobox(self.f_dims, values=self.dim_values,
width=3, state='readonly')
0791         self.dim_m.current(0)
0792         self.dim_m.grid(row=0, column=1)
0793
0794         self.X_text = Label(self.f_dims, text='X', bg=self.color_bg1,
fg=self.color_text2)
0795         self.X_text.grid(row=0, column=2)
0796
0797         self.dim_n = ttk.Combobox(self.f_dims, values=self.dim_values,
width=3, state='readonly')
0798         self.dim_n.current(0)
0799         self.dim_n.grid(row=0, column=3)
0800
0801         self.set = Button(self.f_dims, text='Set matrix', bg=self.color_bg1,
fg=self.color_text2,
0802                           activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
0803                           command=lambda: GUI.set_matrix(self, 'trans',
int(self.dim_m.get()), int(self.dim_n.get())))
0804         self.set.grid(row=0, column=6, padx=(100, 0))
0805
0806         self.f_rand_dims = LabelFrame(self.f_main, text='Random Matrix',
padx=100, pady=10, bg=self.color_bg1,
0807                                       fg=self.color_text3, relief=GROOVE)
0808         self.f_rand_dims.pack()
0809
0810         self.rand_dim_text = Label(self.f_rand_dims, text='Matrix dimension:',
bg=self.color_bg1, fg=self.color_text2)
0811         self.rand_dim_text.grid(row=0, column=0)
```

```
0812
0813     self.rand_dim_m = Entry(self.f_rand_dims, width=6)
0814     self.rand_dim_m.insert(0, '2')
0815     self.rand_dim_m.grid(row=0, column=1)
0816
0817     self.rand_X_text = Label(self.f_rand_dims, text='X',
bg=self.color_bg1, fg=self.color_text2)
0818     self.rand_X_text.grid(row=0, column=2)
0819
0820     self.rand_dim_n = Entry(self.f_rand_dims, width=6)
0821     self.rand_dim_n.insert(0, '2')
0822     self.rand_dim_n.grid(row=0, column=3)
0823
0824     self.rand_set = Button(self.f_rand_dims, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=3,
0825                             activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
0826                             command=lambda: GUI.rand_calculate(self,
'trans', self.rand_dim_m.get(),
0827 self.rand_dim_n.get()))
0828     self.rand_set.grid(row=0, column=6, padx=(100, 0))
0829
0830     def rank(self):
0831         GUI.clear_frame(self)
0832
0833         self.title = Label(self.f_main, text='Matrix Rank Calculator',
font=('Arial', 30, 'bold'),
0834                             bg=self.color_bg1, fg=self.color_text1)
0835         self.title.pack(pady=(30, 0))
0836
0837         self.desc = Label(self.f_main, text='''the rank of a matrix A is the
dimension
0838 of the vector space generated
0839 (or spanned) by its columns.
0840 This corresponds to the maximal number
0841 of linearly independent columns of A.''' , font=('Arial', 15),
bg=self.color_bg1, fg=self.color_text1)
0842         self.desc.pack(pady=(30, 0))
0843
0844         self.f_dims = LabelFrame(self.f_main, text='Custom Matrix', padx=100,
pady=10, bg=self.color_bg1,
0845                                 fg=self.color_text3, relief=GROOVE)
0846         self.f_dims.pack(pady=(200, 50))
0847
0848         self.dim_text = Label(self.f_dims, text='Matrix dimension:',
bg=self.color_bg1, fg=self.color_text2)
0849         self.dim_text.grid(row=0, column=0)
0850
0851         self.dim_m = ttk.Combobox(self.f_dims, values=self.dim_values,
width=3, state='readonly')
0852         self.dim_m.current(0)
0853         self.dim_m.grid(row=0, column=1)
0854
0855         self.X_text = Label(self.f_dims, text='X', bg=self.color_bg1,
fg=self.color_text2)
0856         self.X_text.grid(row=0, column=2)
0857
0858         self.dim_n = ttk.Combobox(self.f_dims, values=self.dim_values,
width=3, state='readonly')
```

```
0859         self.dim_n.current(0)
0860         self.dim_n.grid(row=0, column=3)
0861
0862         self.set = Button(self.f_dims, text='Set matrix', bg=self.color_bg1,
fg=self.color_text2,
0863                             activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
0864                             command=lambda: GUI.set_matrix(self, 'rank',
int(self.dim_m.get()), int(self.dim_n.get())))
0865         self.set.grid(row=0, column=6, padx=(100, 0))
0866
0867         self.f_rand_dims = LabelFrame(self.f_main, text='Random Matrix',
padx=100, pady=10, bg=self.color_bg1,
0868                                     fg=self.color_text3, relief=GROOVE)
0869         self.f_rand_dims.pack()
0870
0871         self.rand_dim_text = Label(self.f_rand_dims, text='Matrix dimension:',
bg=self.color_bg1, fg=self.color_text2)
0872         self.rand_dim_text.grid(row=0, column=0)
0873
0874         self.rand_dim_m = Entry(self.f_rand_dims, width=6)
0875         self.rand_dim_m.insert(0, '2')
0876         self.rand_dim_m.grid(row=0, column=1)
0877
0878         self.rand_X_text = Label(self.f_rand_dims, text='X',
bg=self.color_bg1, fg=self.color_text2)
0879         self.rand_X_text.grid(row=0, column=2)
0880
0881         self.rand_dim_n = Entry(self.f_rand_dims, width=6)
0882         self.rand_dim_n.insert(0, '2')
0883         self.rand_dim_n.grid(row=0, column=3)
0884
0885         self.rand_set = Button(self.f_rand_dims, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=3,
0886                             activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
0887                             command=lambda: GUI.rand_calculate(self,
'rank', self.rand_dim_m.get(),
0888 self.rand_dim_n.get()))
0889         self.rand_set.grid(row=0, column=6, padx=(100, 0))
0890
0891         def trace(self):
0892             GUI.clear_frame(self)
0893
0894             self.title = Label(self.f_main, text='Matrix Trace Calculator',
font=('Arial', 30, 'bold'),
0895                                 bg=self.color_bg1, fg=self.color_text1)
0896             self.title.pack(pady=(30, 0))
0897             self.desc = Label(self.f_main, text='''In linear algebra, the trace of
a square matrix A, denoted tr(A)
0898 is defined to be the sum of elements on
0899 the main diagonal (from the upper left to the lower right) of A.
0900 The trace of a matrix is the sum of its eigenvalues
0901 and it is invariant with respect to a change of basis.
0902 This characterization can be used to define the trace of
0903 a linear operator in general.
0904 The trace is only defined for a square matrix (n × n).''', font=('Arial', 15),
bg=self.color_bg1, fg=self.color_text1)
0905             self.desc.pack(pady=(30, 0))
```

```
0906
0907         self.f_dims = LabelFrame(self.f_main, text='Custom Matrix', padx=100,
0908         pady=10, bg=self.color_bg1,
0909         fg=self.color_text3, relief=GROOVE)
0910         self.f_dims.pack(pady=(200, 50))
0911         self.dim_text = Label(self.f_dims, text='Matrix dimension:',
0912         bg=self.color_bg1, fg=self.color_text2)
0913         self.dim_text.grid(row=0, column=0)
0914         self.dim = ttk.Combobox(self.f_dims, values=self.dim_values, width=3,
0915         state='readonly')
0916         self.dim.current(0)
0917         self.dim.grid(row=0, column=1)
0918         self.set = Button(self.f_dims, text='Set matrix', bg=self.color_bg1,
0919         fg=self.color_text2,
0920         activebackground=self.color_bg1,
0921         activeforeground=self.color_text2, relief=RIDGE,
0922         command=lambda: GUI.set_matrix(self, 'trace',
0923         int(self.dim.get())))
0924         self.set.grid(row=0, column=6, padx=(100, 0))
0925
0926         self.f_rand_dims = LabelFrame(self.f_main, text='Random Matrix',
0927         padx=100, pady=10, bg=self.color_bg1,
0928         fg=self.color_text3, relief=GROOVE)
0929         self.f_rand_dims.pack()
0930         self.rand_dim_text = Label(self.f_rand_dims, text='Matrix dimension:',
0931         bg=self.color_bg1,
0932         fg=self.color_text2)
0933         self.rand_dim_text.grid(row=0, column=0)
0934         self.rand_dim = Entry(self.f_rand_dims, width=6)
0935         self.rand_dim.insert(0, '2')
0936         self.rand_dim.grid(row=0, column=1)
0937         self.rand_set = Button(self.f_rand_dims, text='Calculate',
0938         bg=self.color_bg1, fg=self.color_text2, padx=3,
0939         activebackground=self.color_bg1,
0940         activeforeground=self.color_text2, relief=RIDGE,
0941         command=lambda: GUI.rand_calculate(self,
0942         'trace', self.rand_dim.get()))
0943         self.rand_set.grid(row=0, column=6, padx=(100, 0))
0944
0945         def clear_frame(self):
0946             """This method clears all the widgets from the main (left) frame"""
0947             for widget in self.f_main.winfo_children():
0948                 widget.destroy()
0949
0950         def set_matrix(self, func, a, b=0, c=0):
0951             """This method opens a new window for the matrix (or matrices) input,
0952             according to the desired calculation"""
0953             self.input_win = Toplevel(root, bg=self.color_bg1)
0954             self.input_win.iconbitmap('matrix_ico.ico')
0955             self.input_win.title('Matrix Calculator')
0956
0957             if func == 'add_sub':
0958                 # According to the number of matrices needed to be inputted and
0959                 the dimensions of these
```

```
0954         # matrices, the correct numbers of frames are created (f_up,
f_down, f1, f1_grid etc.)
0955         # as well as entry boxes and some auxiliary buttons ("Clear",
"Fill with 1's" etc.)
0956         #
0957         # Similar process is done on all other 'elif' conditions,
0958         # so commenting will only be present on this 'if' block
0959
0960         self.input_win.resizable(False, False)
0961
0962         # Creating the frames and other widgets
0963         self.f_up = Frame(self.input_win, bg=self.color_bg1)
0964         self.f_up.pack(pady=(20, 0))
0965         self.f1 = Frame(self.f_up, bg=self.color_bg1)
0966         self.f1.pack(side='left', padx=25)
0967         self.separator = ttk.Separator(self.f_up, orient='vertical')
0968         self.separator.pack(side='left', padx=20, pady=20, fill='y')
0969         self.f2 = Frame(self.f_up, bg=self.color_bg1)
0970         self.f2.pack(side='left', padx=25)
0971         self.f_down = Frame(self.input_win, bg=self.color_bg1)
0972         self.f_down.pack(pady=(15, 20))
0973         self.calculate = Button(self.f_down, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=30,
0974                                pady=5, font=('Arial', 15),
activebackground=self.color_bg1,
0975                                activeforeground=self.color_text2,
relief=RIDGE,
0976                                command=lambda: GUI.calculate(self, func,
a, b))
0977         self.calculate.pack()
0978
0979         self.matrix_A_text = Label(self.f1, text='Input matrix A:',
bg=self.color_bg1, fg=self.color_text2,
0980                                font=('Arial', 10))
0981         self.matrix_A_text.pack()
0982         self.matrix_B_text = Label(self.f2, text='Input matrix B:',
bg=self.color_bg1, fg=self.color_text2,
0983                                font=('Arial', 10))
0984         self.matrix_B_text.pack()
0985
0986         self.f1_grid = Frame(self.f1, bg=self.color_bg1)
0987         self.f1_grid.pack()
0988         self.f1_buttons = Frame(self.f1, bg=self.color_bg1)
0989         self.f1_buttons.pack(pady=15)
0990
0991         self.f2_grid = Frame(self.f2, bg=self.color_bg1)
0992         self.f2_grid.pack()
0993         self.f2_buttons = Frame(self.f2, bg=self.color_bg1)
0994         self.f2_buttons.pack(pady=15)
0995
0996         # Creating empty 2D lists, which will later contain the entry
widgets
0997         self.matrix_A_entries = [[] for _ in range(a)]
0998         self.matrix_B_entries = [[] for _ in range(a)]
0999
1000         for j in range(b): # This 'for' loop displays numbers above entry
boxes
1001             Label(self.f1_grid, text=j+1, font=('Arial', 8),
bg=self.color_bg1,
1002                    fg=self.color_text2).grid(row=0, column=j + 1, padx=(3,
```



```
0))
1003         Label(self.f2_grid, text=j + 1, font=('Arial', 8),
bg=self.color_bg1,
1004         fg=self.color_text2).grid(row=0, column=j + 1, padx=(3,
0))
1005         for i in range(a): # This 'for' loop displays the entry boxes and
numbers left to them
1006         Label(self.f1_grid, text=i+1, font=('Arial', 8),
bg=self.color_bg1,
1007         fg=self.color_text2).grid(row=i + 1, column=0, padx=(3,
0))
1008         Label(self.f2_grid, text=i+1, font=('Arial', 8),
bg=self.color_bg1,
1009         fg=self.color_text2).grid(row=i + 1, column=0, padx=(3,
0))
1010         for j in range(b):
1011             x = Entry(self.f1_grid, width=5)
1012             x.grid(row=i+1, column=j+1, padx=3, pady=3)
1013             self.matrix_A_entries[i].append(x)
1014
1015             y = Entry(self.f2_grid, width=5)
1016             y.grid(row=i+1, column=j+1, padx=3, pady=3)
1017             self.matrix_B_entries[i].append(y)
1018
1019         # Creating and displaying the auxiliary buttons
1020         self.clear_button_A = Button(self.f1_buttons, text='Clear',
bg=self.color_bg1, fg=self.color_text2,
1021                                     font=('Arial', 8),
activebackground=self.color_bg1,
1022                                     activeforeground=self.color_text2,
relief=RIDGE, padx=33,
1023                                     command=lambda: GUI.clear_cells(self,
self.matrix_A_entries))
1024         self.clear_button_A.grid(row=0, column=0, columnspan=3, pady=3)
1025
1026         self.zeros_button_A = Button(self.f1_buttons, text="Fill with
0's", bg=self.color_bg1, fg=self.color_text2,
1027                                     font=('Arial', 8),
activebackground=self.color_bg1,
1028                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1029                                     command=lambda: GUI.fill_zeros(self,
self.matrix_A_entries))
1030         self.zeros_button_A.grid(row=1, column=0, pady=3)
1031
1032         self.ones_button_A = Button(self.f1_buttons, text="Fill with 1's",
bg=self.color_bg1, fg=self.color_text2,
1033                                     font=('Arial', 8),
activebackground=self.color_bg1,
1034                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1035                                     command=lambda: GUI.fill_ones(self,
self.matrix_A_entries))
1036         self.ones_button_A.grid(row=1, column=1, pady=3)
1037
1038         self.mem_sv_button_A = Button(self.f1_buttons, text=" Save to
memory ", bg=self.color_bg1,
1039                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1040                                     activeforeground=self.color_text2,
```

```
relief=RIDGE,
1041                                     command=lambda: GUI.mem_sv(self,
self.matrix_A_entries))
1042                                     self.mem_sv_button_A.grid(row=2, column=0, padx=3, pady=3)
1043
1044                                     self.mem_ld_button_A = Button(self.f1_buttons, text="Load from
memory", bg=self.color_bg1,
1045                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1046                                     activeforeground=self.color_text2,
relief=RIDGE,
1047                                     command=lambda: GUI.mem_ld(self,
self.matrix_A_entries))
1048                                     self.mem_ld_button_A.grid(row=2, column=1, padx=3, pady=3)
1049
1050                                     self.clear_button_B = Button(self.f2_buttons, text='Clear',
bg=self.color_bg1, fg=self.color_text2,
1051                                     font=('Arial', 8),
activebackground=self.color_bg1,
1052                                     activeforeground=self.color_text2,
relief=RIDGE, padx=33,
1053                                     command=lambda: GUI.clear_cells(self,
self.matrix_B_entries))
1054                                     self.clear_button_B.grid(row=0, column=0, columnspan=3, pady=3)
1055
1056                                     self.zeros_button_B = Button(self.f2_buttons, text="Fill with
0's", bg=self.color_bg1, fg=self.color_text2,
1057                                     font=('Arial', 8),
activebackground=self.color_bg1,
1058                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1059                                     command=lambda: GUI.fill_zeros(self,
self.matrix_B_entries))
1060                                     self.zeros_button_B.grid(row=1, column=0, pady=3)
1061
1062                                     self.ones_button_B = Button(self.f2_buttons, text="Fill with 1's",
bg=self.color_bg1, fg=self.color_text2,
1063                                     font=('Arial', 8),
activebackground=self.color_bg1,
1064                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1065                                     command=lambda: GUI.fill_ones(self,
self.matrix_B_entries))
1066                                     self.ones_button_B.grid(row=1, column=1, pady=3)
1067
1068                                     self.mem_sv_button_B = Button(self.f2_buttons, text=" Save to
memory ", bg=self.color_bg1,
1069                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1070                                     activeforeground=self.color_text2,
relief=RIDGE,
1071                                     command=lambda: GUI.mem_sv(self,
self.matrix_B_entries))
1072                                     self.mem_sv_button_B.grid(row=2, column=0, padx=3, pady=3)
1073
1074                                     self.mem_ld_button_B = Button(self.f2_buttons, text="Load from
memory", bg=self.color_bg1,
1075                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1076                                     activeforeground=self.color_text2,
```

```
relief=RIDGE,
1077                                     command=lambda: GUI.mem_ld(self,
self.matrix_B_entries))
1078         self.mem_ld_button_B.grid(row=2, column=1, padx=3, pady=3)
1079
1080         elif func == 'mul_num':
1081             is_error = False
1082
1083             if self.num_entry.get():
1084                 try:
1085                     float(self.num_entry.get())
1086                 except ValueError:
1087                     is_error = True
1088                     self.input_win.destroy()
1089                     self.errors('num')
1090
1091             if not is_error:
1092                 self.input_win.resizable(False, False)
1093
1094                 self.f = Frame(self.input_win, bg=self.color_bg1)
1095                 self.f.pack(pady=20)
1096
1097                 self.matrix_text = Label(self.f, text='Input matrix:',
bg=self.color_bg1, fg=self.color_text2,
1098                                     font=('Arial', 10))
1099                 self.matrix_text.pack()
1100
1101                 self.f_grid = Frame(self.f, bg=self.color_bg1)
1102                 self.f_grid.pack(padx=25)
1103                 self.f_buttons = Frame(self.f, bg=self.color_bg1)
1104                 self.f_buttons.pack(pady=(15, 0), padx=25)
1105
1106                 self.matrix_entries = [[] for _ in range(a)]
1107                 for j in range(b):
1108                     Label(self.f_grid, text=j+1, font=('Arial', 8),
bg=self.color_bg1,
1109                             fg=self.color_text2).grid(row=0, column=j + 1,
padx=(3, 0))
1110                 for i in range(a):
1111                     Label(self.f_grid, text=i + 1, font=('Arial', 8),
bg=self.color_bg1,
1112                             fg=self.color_text2).grid(row=i + 1, column=0,
padx=(3, 0))
1113                 for j in range(b):
1114                     x = Entry(self.f_grid, width=5)
1115                     x.grid(row=i+1, column=j+1, padx=3, pady=3)
1116                     self.matrix_entries[i].append(x)
1117
1118                 self.clear_button = Button(self.f_buttons, text='Clear',
bg=self.color_bg1, fg=self.color_text2,
1119                                             font=('Arial', 8),
activebackground=self.color_bg1,
1120                                             activeforeground=self.color_text2,
relief=RIDGE, padx=33,
1121                                             command=lambda:
GUI.clear_cells(self, self.matrix_entries))
1122                 self.clear_button.grid(row=0, column=0, columnspan=3, pady=3)
1123
1124                 self.zeros_button = Button(self.f_buttons, text="Fill with
0's", bg=self.color_bg1, fg=self.color_text2,
```

```
1125                                     font=('Arial', 8),
activebackground=self.color_bg1,
1126                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1127                                     command=lambda:
GUI.fill_zeros(self, self.matrix_entries))
1128                                     self.zeros_button.grid(row=1, column=0, pady=3)
1129
1130                                     self.ones_button = Button(self.f_buttons, text="Fill with
1131 's", bg=self.color_bg1, fg=self.color_text2,
1132                                     font=('Arial', 8),
activebackground=self.color_bg1,
activeforeground=self.color_text2, padx=20,
1133                                     relief=RIDGE, command=lambda:
GUI.fill_ones(self, self.matrix_entries))
1134                                     self.ones_button.grid(row=1, column=1, pady=3)
1135
1136                                     self.mem_sv_button = Button(self.f_buttons, text=" Save to
memory ", bg=self.color_bg1,
1137                                     fg=self.color_text2,
font=('Arial', 8), activebackground=self.color_bg1,
1138                                     activeforeground=self.color_text2,
relief=RIDGE,
1139                                     command=lambda: GUI.mem_sv(self,
self.matrix_entries))
1140                                     self.mem_sv_button.grid(row=2, column=0, padx=3, pady=3)
1141
1142                                     self.mem_ld_button = Button(self.f_buttons, text="Load from
memory", bg=self.color_bg1,
1143                                     fg=self.color_text2,
font=('Arial', 8), activebackground=self.color_bg1,
1144                                     activeforeground=self.color_text2,
relief=RIDGE,
1145                                     command=lambda: GUI.mem_ld(self,
self.matrix_entries))
1146                                     self.mem_ld_button.grid(row=2, column=1, padx=3, pady=3)
1147
1148                                     self.calculate = Button(self.f_buttons, text='Calculate',
bg=self.color_bg1, fg=self.color_text2,
1149                                     padx=30, pady=5, font=('Arial', 15),
activebackground=self.color_bg1,
1150                                     activeforeground=self.color_text2,
relief=RIDGE,
1151                                     command=lambda: GUI.calculate(self,
func, a, b))
1152                                     self.calculate.grid(row=3, column=0, columnspan=3, pady=(30,
0))
1153
1154                                     elif func == 'mul':
1155                                     self.input_win.resizable(False, False)
1156
1157                                     self.f_up = Frame(self.input_win, bg=self.color_bg1)
1158                                     self.f_up.pack(pady=(20, 0))
1159                                     self.f1 = Frame(self.f_up, bg=self.color_bg1)
1160                                     self.f1.pack(side='left', padx=25)
1161                                     self.separator = ttk.Separator(self.f_up, orient='vertical')
1162                                     self.separator.pack(side='left', padx=20, pady=20, fill='y')
1163                                     self.f2 = Frame(self.f_up, bg=self.color_bg1)
1164                                     self.f2.pack(side='left', padx=25)
1165                                     self.f_down = Frame(self.input_win, bg=self.color_bg1)
```

```
1166         self.f_down.pack(pady=(15, 20))
1167         self.calculate = Button(self.f_down, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=30,
1168                                 pady=5, font=('Arial', 15),
activebackground=self.color_bg1,
1169                                 activeforeground=self.color_text2,
relief=RIDGE,
1170                                 command=lambda: GUI.calculate(self, func,
a, b, c))
1171         self.calculate.pack()
1172
1173         self.matrix_A_text = Label(self.f1, text='Input matrix A:',
bg=self.color_bg1, fg=self.color_text2,
1174                                 font=('Arial', 10))
1175         self.matrix_A_text.pack()
1176         self.matrix_B_text = Label(self.f2, text='Input matrix B:',
bg=self.color_bg1, fg=self.color_text2,
1177                                 font=('Arial', 10))
1178         self.matrix_B_text.pack()
1179
1180         self.f1_grid = Frame(self.f1, bg=self.color_bg1)
1181         self.f1_grid.pack()
1182         self.f1_buttons = Frame(self.f1, bg=self.color_bg1)
1183         self.f1_buttons.pack(pady=15)
1184
1185         self.f2_grid = Frame(self.f2, bg=self.color_bg1)
1186         self.f2_grid.pack()
1187         self.f2_buttons = Frame(self.f2, bg=self.color_bg1)
1188         self.f2_buttons.pack(pady=15)
1189
1190         self.matrix_A_entries = [[] for _ in range(a)]
1191         self.matrix_B_entries = [[] for _ in range(b)]
1192
1193         for j in range(b):
1194             Label(self.f1_grid, text=j+1, font=('Arial', 8),
bg=self.color_bg1,
1195                     fg=self.color_text2).grid(row=0, column=j + 1, padx=(3,
0))
1196         for i in range(a):
1197             Label(self.f1_grid, text=i + 1, font=('Arial', 8),
bg=self.color_bg1,
1198                     fg=self.color_text2).grid(row=i + 1, column=0, padx=(3,
0))
1199         for j in range(b):
1200             x = Entry(self.f1_grid, width=5)
1201             x.grid(row=i+1, column=j+1, padx=3, pady=3)
1202             self.matrix_A_entries[i].append(x)
1203
1204         for j in range(c):
1205             Label(self.f2_grid, text=j+1, font=('Arial', 8),
bg=self.color_bg1,
1206                     fg=self.color_text2).grid(row=0, column=j + 1, padx=(3,
0))
1207         for i in range(b):
1208             Label(self.f2_grid, text=i + 1, font=('Arial', 8),
bg=self.color_bg1,
1209                     fg=self.color_text2).grid(row=i + 1, column=0, padx=(3,
0))
1210         for j in range(c):
1211             y = Entry(self.f2_grid, width=5)
```

```
1212         y.grid(row=i+1, column=j+1, padx=3, pady=3)
1213         self.matrix_B_entries[i].append(y)
1214
1215         self.clear_button_A = Button(self.fl_buttons, text='Clear',
bg=self.color_bg1, fg=self.color_text2,
1216                                     font=('Arial', 8),
activebackground=self.color_bg1,
1217                                     activeforeground=self.color_text2,
relief=RIDGE, padx=33,
1218                                     command=lambda: GUI.clear_cells(self,
self.matrix_A_entries))
1219         self.clear_button_A.grid(row=0, column=0, columnspan=3, pady=3)
1220
1221         self.zeros_button_A = Button(self.fl_buttons, text="Fill with
0's", bg=self.color_bg1, fg=self.color_text2,
1222                                     font=('Arial', 8),
activebackground=self.color_bg1,
1223                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1224                                     command=lambda: GUI.fill_zeros(self,
self.matrix_A_entries))
1225         self.zeros_button_A.grid(row=1, column=0, pady=3)
1226
1227         self.ones_button_A = Button(self.fl_buttons, text="Fill with 1's",
bg=self.color_bg1, fg=self.color_text2,
1228                                     font=('Arial', 8),
activebackground=self.color_bg1,
1229                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1230                                     command=lambda: GUI.fill_ones(self,
self.matrix_A_entries))
1231         self.ones_button_A.grid(row=1, column=1, pady=3)
1232
1233         self.mem_sv_button_A = Button(self.fl_buttons, text=" Save to
memory ", bg=self.color_bg1,
1234                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1235                                     activeforeground=self.color_text2,
relief=RIDGE,
1236                                     command=lambda: GUI.mem_sv(self,
self.matrix_A_entries))
1237         self.mem_sv_button_A.grid(row=2, column=0, padx=3, pady=3)
1238
1239         self.mem_ld_button_A = Button(self.fl_buttons, text="Load from
memory", bg=self.color_bg1,
1240                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1241                                     activeforeground=self.color_text2,
relief=RIDGE,
1242                                     command=lambda: GUI.mem_ld(self,
self.matrix_A_entries))
1243         self.mem_ld_button_A.grid(row=2, column=1, padx=3, pady=3)
1244
1245         self.clear_button_B = Button(self.f2_buttons, text='Clear',
bg=self.color_bg1, fg=self.color_text2,
1246                                     font=('Arial', 8),
activebackground=self.color_bg1,
1247                                     activeforeground=self.color_text2,
relief=RIDGE, padx=33,
1248                                     command=lambda: GUI.clear_cells(self,
```

```
self.matrix_B_entries))
1249         self.clear_button_B.grid(row=0, column=0, columnspan=3, pady=3)
1250
1251         self.zeros_button_B = Button(self.f2_buttons, text="Fill with
0's", bg=self.color_bg1, fg=self.color_text2,
1252                                     font=('Arial', 8),
activebackground=self.color_bg1,
1253                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1254                                     command=lambda: GUI.fill_zeros(self,
self.matrix_B_entries))
1255         self.zeros_button_B.grid(row=1, column=0, pady=3)
1256
1257         self.ones_button_B = Button(self.f2_buttons, text="Fill with 1's",
bg=self.color_bg1, fg=self.color_text2,
1258                                     font=('Arial', 8),
activebackground=self.color_bg1,
1259                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1260                                     command=lambda: GUI.fill_ones(self,
self.matrix_B_entries))
1261         self.ones_button_B.grid(row=1, column=1, pady=3)
1262
1263         self.mem_sv_button_B = Button(self.f2_buttons, text=" Save to
memory ", bg=self.color_bg1,
1264                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1265                                     activeforeground=self.color_text2,
relief=RIDGE,
1266                                     command=lambda: GUI.mem_sv(self,
self.matrix_B_entries))
1267         self.mem_sv_button_B.grid(row=2, column=0, padx=3, pady=3)
1268
1269         self.mem_ld_button_B = Button(self.f2_buttons, text="Load from
memory", bg=self.color_bg1,
1270                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1271                                     activeforeground=self.color_text2,
relief=RIDGE,
1272                                     command=lambda: GUI.mem_ld(self,
self.matrix_B_entries))
1273         self.mem_ld_button_B.grid(row=2, column=1, padx=3, pady=3)
1274
1275         elif func == 'power':
1276             is_error = False
1277             if self.power_entry.get():
1278                 if not self.power_entry.get().isdigit():
1279                     self.input_win.destroy()
1280                     is_error = True
1281                     self.errors('power')
1282
1283             if not is_error:
1284                 self.input_win.resizable(False, False)
1285
1286                 self.f = Frame(self.input_win, bg=self.color_bg1)
1287                 self.f.pack(pady=20)
1288
1289                 self.matrix_text = Label(self.f, text='Input matrix:',
bg=self.color_bg1, fg=self.color_text2,
1290                                     font=('Arial', 10))
```

```
1291         self.matrix_text.pack()
1292
1293         self.f_grid = Frame(self.f, bg=self.color_bg1)
1294         self.f_grid.pack(padx=25)
1295         self.f_buttons = Frame(self.f, bg=self.color_bg1)
1296         self.f_buttons.pack(pady=(15, 0), padx=25)
1297
1298         self.matrix_entries = [[] for _ in range(a)]
1299
1300         for j in range(a):
1301             Label(self.f_grid, text=j+1, font=('Arial', 8),
1302                 bg=self.color_bg1,
1303                 fg=self.color_text2).grid(row=0, column=j + 1,
1304                     padx=(3, 0))
1305             for i in range(a):
1306                 Label(self.f_grid, text=i + 1, font=('Arial', 8),
1307                     bg=self.color_bg1,
1308                     fg=self.color_text2).grid(row=i + 1, column=0,
1309                         padx=(3, 0))
1310                 for j in range(a):
1311                     x = Entry(self.f_grid, width=5)
1312                     x.grid(row=i+1, column=j+1, padx=3, pady=3)
1313                     self.matrix_entries[i].append(x)
1314
1315             self.clear_button = Button(self.f_buttons, text='Clear',
1316                 bg=self.color_bg1, fg=self.color_text2,
1317                 font=('Arial', 8),
1318                 activebackground=self.color_bg1,
1319                 activeforeground=self.color_text2,
1320                 relief=RIDGE, padx=33,
1321                 command=lambda:
1322                     GUI.clear_cells(self, self.matrix_entries))
1323             self.clear_button.grid(row=0, column=0, columnspan=3, pady=3)
1324
1325             self.zeros_button = Button(self.f_buttons, text="Fill with
1326                 0's", bg=self.color_bg1, fg=self.color_text2,
1327                 font=('Arial', 8),
1328                 activebackground=self.color_bg1,
1329                 activeforeground=self.color_text2,
1330                 relief=RIDGE, padx=20,
1331                 command=lambda:
1332                     GUI.fill_zeros(self, self.matrix_entries))
1333             self.zeros_button.grid(row=1, column=0, pady=3)
1334
1335             self.ones_button = Button(self.f_buttons, text="Fill with
1336                 1's", bg=self.color_bg1, fg=self.color_text2,
1337                 font=('Arial', 8),
1338                 activebackground=self.color_bg1,
1339                 activeforeground=self.color_text2,
1340                 relief=RIDGE, padx=20,
1341                 command=lambda:
1342                     GUI.fill_ones(self, self.matrix_entries))
1343             self.ones_button.grid(row=1, column=1, pady=3)
1344
1345             self.mem_sv_button = Button(self.f_buttons, text=" Save to
1346                 memory ", bg=self.color_bg1,
1347                 fg=self.color_text2,
1348                 font=('Arial', 8), activebackground=self.color_bg1,
1349                 activeforeground=self.color_text2,
1350                 relief=RIDGE,
1351                 command=lambda: GUI.mem_sv(self,
```



```
self.matrix_entries))
1333         self.mem_sv_button.grid(row=2, column=0, padx=3, pady=3)
1334
1335         self.mem_ld_button = Button(self.f_buttons, text="Load from
memory", bg=self.color_bg1,
1336                                     fg=self.color_text2,
font=('Arial', 8), activebackground=self.color_bg1,
1337                                     activeforeground=self.color_text2,
relief=RIDGE,
1338                                     command=lambda: GUI.mem_ld(self,
self.matrix_entries))
1339         self.mem_ld_button.grid(row=2, column=1, padx=3, pady=3)
1340
1341         self.calculate = Button(self.f_buttons, text='Calculate',
bg=self.color_bg1, fg=self.color_text2,
1342                                     padx=30, pady=5, font=('Arial', 15),
activebackground=self.color_bg1,
1343                                     activeforeground=self.color_text2,
relief=RIDGE,
1344                                     command=lambda: GUI.calculate(self,
func, a, b))
1345         self.calculate.grid(row=3, column=0, columnspan=3, pady=(30,
0))
1346
1347         elif func == 'det':
1348             self.input_win.resizable(False, False)
1349
1350             self.f = Frame(self.input_win, bg=self.color_bg1)
1351             self.f.pack(pady=20)
1352
1353             self.matrix_text = Label(self.f, text='Input matrix:',
bg=self.color_bg1, fg=self.color_text2,
1354                                     font=('Arial', 10))
1355             self.matrix_text.pack()
1356
1357             self.f_grid = Frame(self.f, bg=self.color_bg1)
1358             self.f_grid.pack(padx=25)
1359             self.f_buttons = Frame(self.f, bg=self.color_bg1)
1360             self.f_buttons.pack(pady=(15, 0), padx=25)
1361
1362             self.matrix_entries = [[] for _ in range(a)]
1363
1364             for j in range(a):
1365                 Label(self.f_grid, text=j+1, font=('Arial', 8),
bg=self.color_bg1,
1366                                     fg=self.color_text2).grid(row=0, column=j + 1, padx=(3,
0))
1367             for i in range(a):
1368                 Label(self.f_grid, text=i + 1, font=('Arial', 8),
bg=self.color_bg1,
1369                                     fg=self.color_text2).grid(row=i + 1, column=0, padx=(3,
0))
1370             for j in range(a):
1371                 x = Entry(self.f_grid, width=5)
1372                 x.grid(row=i+1, column=j+1, padx=3, pady=3)
1373                 self.matrix_entries[i].append(x)
1374
1375             self.clear_button = Button(self.f_buttons, text='Clear',
bg=self.color_bg1, fg=self.color_text2,
1376                                     font=('Arial', 8),
```

```
activebackground=self.color_bg1,
1377                                     activeforeground=self.color_text2,
relief=RIDGE, padx=33,
1378                                     command=lambda: GUI.clear_cells(self,
self.matrix_entries))
1379                                     self.clear_button.grid(row=0, column=0, columnspan=3, pady=3)
1380
1381                                     self.zeros_button = Button(self.f_buttons, text="Fill with 0's",
bg=self.color_bg1, fg=self.color_text2,
1382                                     font=('Arial', 8),
activebackground=self.color_bg1,
1383                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1384                                     command=lambda: GUI.fill_zeros(self,
self.matrix_entries))
1385                                     self.zeros_button.grid(row=1, column=0, pady=3)
1386
1387                                     self.ones_button = Button(self.f_buttons, text="Fill with 1's",
bg=self.color_bg1, fg=self.color_text2,
1388                                     font=('Arial', 8),
1389                                     activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
1390                                     padx=20, command=lambda:
GUI.fill_ones(self, self.matrix_entries))
1391                                     self.ones_button.grid(row=1, column=1, pady=3)
1392
1393                                     self.mem_sv_button = Button(self.f_buttons, text=" Save to memory
", bg=self.color_bg1,
1394                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1395                                     activeforeground=self.color_text2,
relief=RIDGE,
1396                                     command=lambda: GUI.mem_sv(self,
self.matrix_entries))
1397                                     self.mem_sv_button.grid(row=2, column=0, padx=3, pady=3)
1398
1399                                     self.mem_ld_button = Button(self.f_buttons, text="Load from
memory", bg=self.color_bg1,
1400                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1401                                     activeforeground=self.color_text2,
relief=RIDGE,
1402                                     command=lambda: GUI.mem_ld(self,
self.matrix_entries))
1403                                     self.mem_ld_button.grid(row=2, column=1, padx=3, pady=3)
1404
1405                                     self.calculate = Button(self.f_buttons, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=30,
1406                                     pady=5, font=('Arial', 15),
activebackground=self.color_bg1,
1407                                     activeforeground=self.color_text2,
relief=RIDGE,
1408                                     command=lambda: GUI.calculate(self, func,
a, b))
1409                                     self.calculate.grid(row=3, column=0, columnspan=3, pady=(30, 0))
1410
1411                                     elif func == 'inv':
1412                                     self.input_win.resizable(False, False)
1413
1414                                     self.f = Frame(self.input_win, bg=self.color_bg1)
```

```
1415         self.f.pack(pady=20)
1416
1417         self.matrix_text = Label(self.f, text='Input matrix:',
bg=self.color_bg1, fg=self.color_text2,
1418                                 font=('Arial', 10))
1419         self.matrix_text.pack()
1420
1421         self.f_grid = Frame(self.f, bg=self.color_bg1)
1422         self.f_grid.pack(padx=25)
1423         self.f_buttons = Frame(self.f, bg=self.color_bg1)
1424         self.f_buttons.pack(pady=(15, 0), padx=25)
1425
1426         self.matrix_entries = [[] for _ in range(a)]
1427
1428         for j in range(a):
1429             Label(self.f_grid, text=j+1, font=('Arial', 8),
bg=self.color_bg1,
1430                 fg=self.color_text2).grid(row=0, column=j + 1, padx=(3,
0))
1431         for i in range(a):
1432             Label(self.f_grid, text=i + 1, font=('Arial', 8),
bg=self.color_bg1,
1433                 fg=self.color_text2).grid(row=i + 1, column=0, padx=(3,
0))
1434         for j in range(a):
1435             x = Entry(self.f_grid, width=5)
1436             x.grid(row=i+1, column=j+1, padx=3, pady=3)
1437             self.matrix_entries[i].append(x)
1438
1439         self.clear_button = Button(self.f_buttons, text='Clear',
bg=self.color_bg1, fg=self.color_text2,
1440                                   font=('Arial', 8),
activebackground=self.color_bg1,
1441                                   activeforeground=self.color_text2,
relief=RIDGE, padx=33,
1442                                   command=lambda: GUI.clear_cells(self,
self.matrix_entries))
1443         self.clear_button.grid(row=0, column=0, columnspan=3, pady=3)
1444
1445         self.zeros_button = Button(self.f_buttons, text="Fill with 0's",
bg=self.color_bg1, fg=self.color_text2,
1446                                   font=('Arial', 8),
activebackground=self.color_bg1,
1447                                   activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1448                                   command=lambda: GUI.fill_zeros(self,
self.matrix_entries))
1449         self.zeros_button.grid(row=1, column=0, pady=3)
1450
1451         self.ones_button = Button(self.f_buttons, text="Fill with 1's",
bg=self.color_bg1, fg=self.color_text2,
1452                                   font=('Arial', 8),
activebackground=self.color_bg1,
1453                                   activeforeground=self.color_text2, relief=RIDGE,
padx=20, command=lambda:
1454         GUI.fill_ones(self, self.matrix_entries))
1455         self.ones_button.grid(row=1, column=1, pady=3)
1456
1457         self.mem_sv_button = Button(self.f_buttons, text=" Save to memory
", bg=self.color_bg1,
```

```
1458                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1459                                     activeforeground=self.color_text2,
relief=RIDGE,
1460                                     command=lambda: GUI.mem_sv(self,
self.matrix_entries))
1461         self.mem_sv_button.grid(row=2, column=0, padx=3, pady=3)
1462
1463         self.mem_ld_button = Button(self.f_buttons, text="Load from
memory", bg=self.color_bg1,
1464                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1465                                     activeforeground=self.color_text2,
relief=RIDGE,
1466                                     command=lambda: GUI.mem_ld(self,
self.matrix_entries))
1467         self.mem_ld_button.grid(row=2, column=1, padx=3, pady=3)
1468
1469         self.calculate = Button(self.f_buttons, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=30,
1470                                     pady=5, font=('Arial', 15),
activebackground=self.color_bg1,
1471                                     activeforeground=self.color_text2,
relief=RIDGE,
1472                                     command=lambda: GUI.calculate(self, func,
a, b))
1473         self.calculate.grid(row=3, column=0, columnspan=3, pady=(30, 0))
1474
1475         elif func == 'trans':
1476             self.input_win.resizable(False, False)
1477
1478             self.f = Frame(self.input_win, bg=self.color_bg1)
1479             self.f.pack(pady=20)
1480
1481             self.matrix_text = Label(self.f, text='Input matrix:',
bg=self.color_bg1, fg=self.color_text2,
1482                                     font=('Arial', 10))
1483             self.matrix_text.pack()
1484
1485             self.f_grid = Frame(self.f, bg=self.color_bg1)
1486             self.f_grid.pack(padx=25)
1487             self.f_buttons = Frame(self.f, bg=self.color_bg1)
1488             self.f_buttons.pack(pady=(15, 0), padx=25)
1489
1490             self.matrix_entries = [[] for _ in range(a)]
1491
1492             for j in range(b):
1493                 Label(self.f_grid, text=j+1, font=('Arial', 8),
bg=self.color_bg1,
1494                                     fg=self.color_text2).grid(row=0, column=j + 1, padx=(3,
0))
1495             for i in range(a):
1496                 Label(self.f_grid, text=i + 1, font=('Arial', 8),
bg=self.color_bg1,
1497                                     fg=self.color_text2).grid(row=i + 1, column=0, padx=(3,
0))
1498             for j in range(b):
1499                 x = Entry(self.f_grid, width=5)
1500                 x.grid(row=i+1, column=j+1, padx=3, pady=3)
1501                 self.matrix_entries[i].append(x)
```

```
1502
1503         self.clear_button = Button(self.f_buttons, text='Clear',
bg=self.color_bg1, fg=self.color_text2,
1504                                     font=('Arial', 8),
activebackground=self.color_bg1,
1505                                     activeforeground=self.color_text2,
relief=RIDGE, padx=33,
1506                                     command=lambda: GUI.clear_cells(self,
self.matrix_entries))
1507         self.clear_button.grid(row=0, column=0, columnspan=3, pady=3)
1508
1509         self.zeros_button = Button(self.f_buttons, text="Fill with 0's",
bg=self.color_bg1, fg=self.color_text2,
1510                                     font=('Arial', 8),
activebackground=self.color_bg1,
1511                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1512                                     command=lambda: GUI.fill_zeros(self,
self.matrix_entries))
1513         self.zeros_button.grid(row=1, column=0, pady=3)
1514
1515         self.ones_button = Button(self.f_buttons, text="Fill with 1's",
bg=self.color_bg1, fg=self.color_text2,
1516                                     font=('Arial', 8),
activebackground=self.color_bg1,
1517         activeforeground=self.color_text2, relief=RIDGE,
1518         padx=20, command=lambda:
GUI.fill_ones(self, self.matrix_entries))
1519         self.ones_button.grid(row=1, column=1, pady=3)
1520
1521         self.mem_sv_button = Button(self.f_buttons, text=" Save to memory
", bg=self.color_bg1,
1522                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1523                                     activeforeground=self.color_text2,
relief=RIDGE,
1524                                     command=lambda: GUI.mem_sv(self,
self.matrix_entries))
1525         self.mem_sv_button.grid(row=2, column=0, padx=3, pady=3)
1526
1527         self.mem_ld_button = Button(self.f_buttons, text="Load from
memory", bg=self.color_bg1,
1528                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1529                                     activeforeground=self.color_text2,
relief=RIDGE,
1530                                     command=lambda: GUI.mem_ld(self,
self.matrix_entries))
1531         self.mem_ld_button.grid(row=2, column=1, padx=3, pady=3)
1532
1533         self.calculate = Button(self.f_buttons, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=30,
1534                                     pady=5, font=('Arial', 15),
activebackground=self.color_bg1,
1535                                     activeforeground=self.color_text2,
relief=RIDGE,
1536                                     command=lambda: GUI.calculate(self, func,
a, b))
1537         self.calculate.grid(row=3, column=0, columnspan=3, pady=(30, 0))
1538
```

```
1539         elif func == 'rank':
1540             self.input_win.resizable(False, False)
1541
1542             self.f = Frame(self.input_win, bg=self.color_bg1)
1543             self.f.pack(pady=20)
1544
1545             self.matrix_text = Label(self.f, text='Input matrix:',
bg=self.color_bg1, fg=self.color_text2,
1546                                     font=('Arial', 10))
1547             self.matrix_text.pack()
1548
1549             self.f_grid = Frame(self.f, bg=self.color_bg1)
1550             self.f_grid.pack(padx=25)
1551             self.f_buttons = Frame(self.f, bg=self.color_bg1)
1552             self.f_buttons.pack(pady=(15, 0), padx=25)
1553
1554             self.matrix_entries = [[] for _ in range(a)]
1555
1556             for j in range(b):
1557                 Label(self.f_grid, text=j+1, font=('Arial', 8),
bg=self.color_bg1,
1558                     fg=self.color_text2).grid(row=0, column=j + 1, padx=(3,
0))
1559             for i in range(a):
1560                 Label(self.f_grid, text=i + 1, font=('Arial', 8),
bg=self.color_bg1,
1561                     fg=self.color_text2).grid(row=i + 1, column=0, padx=(3,
0))
1562             for j in range(b):
1563                 x = Entry(self.f_grid, width=5)
1564                 x.grid(row=i+1, column=j+1, padx=3, pady=3)
1565                 self.matrix_entries[i].append(x)
1566
1567             self.clear_button = Button(self.f_buttons, text='Clear',
bg=self.color_bg1, fg=self.color_text2,
1568                                     font=('Arial', 8),
activebackground=self.color_bg1,
1569                                     activeforeground=self.color_text2,
relief=RIDGE, padx=33,
1570                                     command=lambda: GUI.clear_cells(self,
self.matrix_entries))
1571             self.clear_button.grid(row=0, column=0, columnspan=3, pady=3)
1572
1573             self.zeros_button = Button(self.f_buttons, text="Fill with 0's",
bg=self.color_bg1, fg=self.color_text2,
1574                                     font=('Arial', 8),
activebackground=self.color_bg1,
1575                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1576                                     command=lambda: GUI.fill_zeros(self,
self.matrix_entries))
1577             self.zeros_button.grid(row=1, column=0, pady=3)
1578
1579             self.ones_button = Button(self.f_buttons, text="Fill with 1's",
bg=self.color_bg1, fg=self.color_text2,
1580                                     font=('Arial', 8),
activebackground=self.color_bg1,
activeforeground=self.color_text2, relief=RIDGE,
1582                                     padx=20, command=lambda:
GUI.fill_ones(self, self.matrix_entries))
```

```
1583         self.ones_button.grid(row=1, column=1, pady=3)
1584
1585         self.mem_sv_button = Button(self.f_buttons, text=" Save to memory
1586         ", bg=self.color_bg1,
1587         fg=self.color_text2, font=('Arial',
1588         8), activebackground=self.color_bg1,
1589         activeforeground=self.color_text2,
1590         relief=RIDGE,
1591         command=lambda: GUI.mem_sv(self,
1592         self.matrix_entries))
1593         self.mem_sv_button.grid(row=2, column=0, padx=3, pady=3)
1594
1595         self.mem_ld_button = Button(self.f_buttons, text="Load from
1596         memory", bg=self.color_bg1,
1597         fg=self.color_text2, font=('Arial',
1598         8), activebackground=self.color_bg1,
1599         activeforeground=self.color_text2,
1600         relief=RIDGE,
1601         command=lambda: GUI.mem_ld(self,
1602         self.matrix_entries))
1603         self.mem_ld_button.grid(row=2, column=1, padx=3, pady=3)
1604
1605         self.calculate = Button(self.f_buttons, text='Calculate',
1606         bg=self.color_bg1, fg=self.color_text2, padx=30,
1607         pady=5, font=('Arial', 15),
1608         activebackground=self.color_bg1,
1609         activeforeground=self.color_text2,
1610         relief=RIDGE,
1611         command=lambda: GUI.calculate(self, func,
1612         a, b))
1613         self.calculate.grid(row=3, column=0, columnspan=3, pady=(30, 0))
1614
1615         elif func == 'trace':
1616             self.input_win.resizable(False, False)
1617
1618             self.f = Frame(self.input_win, bg=self.color_bg1)
1619             self.f.pack(pady=20)
1620
1621             self.matrix_text = Label(self.f, text='Input matrix:',
1622             bg=self.color_bg1, fg=self.color_text2,
1623             font=('Arial', 10))
1624             self.matrix_text.pack()
1625
1626             self.f_grid = Frame(self.f, bg=self.color_bg1)
1627             self.f_grid.pack(padx=25)
1628             self.f_buttons = Frame(self.f, bg=self.color_bg1)
1629             self.f_buttons.pack(pady=(15, 0), padx=25)
1630
1631             self.matrix_entries = [[] for _ in range(a)]
1632
1633             for j in range(a):
1634                 Label(self.f_grid, text=j + 1, font=('Arial', 8),
1635                 bg=self.color_bg1,
1636                 fg=self.color_text2).grid(row=0, column=j + 1, padx=(3,
1637                 0))
1638             for i in range(a):
1639                 Label(self.f_grid, text=i + 1, font=('Arial', 8),
1640                 bg=self.color_bg1,
1641                 fg=self.color_text2).grid(row=i + 1, column=0, padx=(3,
1642                 0))
```

```
1626         for j in range(a):
1627             x = Entry(self.f_grid, width=5)
1628             x.grid(row=i + 1, column=j + 1, padx=3, pady=3)
1629             self.matrix_entries[i].append(x)
1630
1631             self.clear_button = Button(self.f_buttons, text='Clear',
bg=self.color_bg1, fg=self.color_text2,
1632                                     font=('Arial', 8),
activebackground=self.color_bg1,
1633                                     activeforeground=self.color_text2,
relief=RIDGE, padx=33,
1634                                     command=lambda: GUI.clear_cells(self,
self.matrix_entries))
1635             self.clear_button.grid(row=0, column=0, columnspan=3, pady=3)
1636
1637             self.zeros_button = Button(self.f_buttons, text="Fill with 0's",
bg=self.color_bg1, fg=self.color_text2,
1638                                     font=('Arial', 8),
activebackground=self.color_bg1,
1639                                     activeforeground=self.color_text2,
relief=RIDGE, padx=20,
1640                                     command=lambda: GUI.fill_zeros(self,
self.matrix_entries))
1641             self.zeros_button.grid(row=1, column=0, pady=3)
1642
1643             self.ones_button = Button(self.f_buttons, text="Fill with 1's",
bg=self.color_bg1, fg=self.color_text2,
1644                                     font=('Arial', 8),
activebackground=self.color_bg1,
1645                                     activeforeground=self.color_text2, relief=RIDGE,
1646                                     padx=20, command=lambda:
GUI.fill_ones(self, self.matrix_entries))
1647             self.ones_button.grid(row=1, column=1, pady=3)
1648
1649             self.mem_sv_button = Button(self.f_buttons, text=" Save to memory",
bg=self.color_bg1,
1650                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1651                                     activeforeground=self.color_text2,
relief=RIDGE,
1652                                     command=lambda: GUI.mem_sv(self,
self.matrix_entries))
1653             self.mem_sv_button.grid(row=2, column=0, padx=3, pady=3)
1654
1655             self.mem_ld_button = Button(self.f_buttons, text="Load from
memory", bg=self.color_bg1,
1656                                     fg=self.color_text2, font=('Arial',
8), activebackground=self.color_bg1,
1657                                     activeforeground=self.color_text2,
relief=RIDGE,
1658                                     command=lambda: GUI.mem_ld(self,
self.matrix_entries))
1659             self.mem_ld_button.grid(row=2, column=1, padx=3, pady=3)
1660
1661             self.calculate = Button(self.f_buttons, text='Calculate',
bg=self.color_bg1, fg=self.color_text2, padx=30,
1662                                     pady=5, font=('Arial', 15),
activebackground=self.color_bg1,
1663                                     activeforeground=self.color_text2,
relief=RIDGE,
```



```
1664         command=lambda: GUI.calculate(self, func,
a, b))
1665         self.calculate.grid(row=3, column=0, columnspan=3, pady=(30, 0))
1666
1667     def calculate(self, func, a, b=0, c=0):
1668         """This method passes the matrix values in NumPy arrays and
1669         proceeds with the according calculation"""
1670         is_error = False # 'True' if an error has occurred
1671
1672         if func == 'add_sub':
1673             self.matrix_A = np.zeros((a, b))
1674             self.matrix_B = np.zeros((a, b))
1675
1676             try:
1677                 for i in range(a):
1678                     for j in range(b):
1679                         if self.matrix_A_entries[i][j].get() == '':
1680                             self.matrix_A_entries[i][j].insert(0, '0')
1681                         if self.matrix_B_entries[i][j].get() == '':
1682                             self.matrix_B_entries[i][j].insert(0, '0')
1683
1684                         self.matrix_A[i, j] =
float(self.matrix_A_entries[i][j].get())
1685                         self.matrix_B[i, j] =
float(self.matrix_B_entries[i][j].get())
1686             except ValueError:
1687                 is_error = True
1688                 self.errors('alpha')
1689             except:
1690                 is_error = True
1691                 self.errors('unexpected')
1692
1693             if not is_error:
1694                 print("Matrix A:", self.matrix_A, sep="\n")
1695                 print()
1696                 print("Matrix B:", self.matrix_B, sep="\n")
1697                 print()
1698
1699                 if self.op.get() == '+':
1700                     start = time.perf_counter()
1701                     calc = SimpleCalculation.matrix_add(self.matrix_A,
self.matrix_B)
1702                     finish = time.perf_counter()
1703
1704                 elif self.op.get() == '-':
1705                     start = time.perf_counter()
1706                     calc = SimpleCalculation.matrix_sub(self.matrix_A,
self.matrix_B)
1707                     finish = time.perf_counter()
1708
1709                 self.time = round(finish - start, 3)
1710                 print("Result: ", calc, sep="\n")
1711                 print()
1712                 print("Time:", self.time)
1713                 print()
1714                 self.result_show(calc)
1715
1716             elif func == 'mul_num':
1717                 self.matrix = np.zeros((a, b))
1718
```

```
1719         try:
1720             for i in range(a):
1721                 for j in range(b):
1722                     if self.matrix_entries[i][j].get() == '':
1723                         self.matrix_entries[i][j].insert(0, '0')
1724
1725                 self.matrix[i, j] =
float(self.matrix_entries[i][j].get())
1726         except ValueError:
1727             is_error = True
1728             self.errors('alpha')
1729         except:
1730             is_error = True
1731             self.errors('unexpected')
1732
1733         if not is_error:
1734             print("Matrix:", self.matrix, sep="\n")
1735             print()
1736
1737             if not self.num_entry.get(): num = 1
1738             else: num = self.num_entry.get()
1739
1740             start = time.perf_counter()
1741             calc = SimpleCalculation.matrix_mul_num(self.matrix, num)
1742             finish = time.perf_counter()
1743
1744             self.time = round(finish - start, 3)
1745             print("Result:", calc, sep="\n")
1746             print()
1747             print("Time:", self.time)
1748             print()
1749             self.result_show(calc)
1750
1751         elif func == 'mul':
1752             self.matrix_A = np.zeros((a, b))
1753             self.matrix_B = np.zeros((b, c))
1754
1755             try:
1756                 for i in range(a):
1757                     for j in range(b):
1758                         if self.matrix_A_entries[i][j].get() == '':
1759                             self.matrix_A_entries[i][j].insert(0, '0')
1760
1761                         self.matrix_A[i, j] =
float(self.matrix_A_entries[i][j].get())
1762                 for i in range(b):
1763                     for j in range(c):
1764                         if self.matrix_B_entries[i][j].get() == '':
1765                             self.matrix_B_entries[i][j].insert(0, '0')
1766
1767                         self.matrix_B[i, j] =
float(self.matrix_B_entries[i][j].get())
1768             except ValueError:
1769                 is_error = True
1770                 self.errors('alpha')
1771             except:
1772                 is_error = True
1773                 self.errors('unexpected')
1774
1775             if not is_error:
```

```
1776         print("Matrix A:", self.matrix_A, sep="\n")
1777         print()
1778         print("Matrix B:", self.matrix_B, sep="\n")
1779         print()
1780
1781         start = time.perf_counter()
1782         calc = SimpleCalculation.matrix_mul(self.matrix_A,
self.matrix_B)
1783         finish = time.perf_counter()
1784
1785         self.time = round(finish - start, 3)
1786         print("Result:", calc, sep="\n")
1787         print()
1788         print("Time:", self.time)
1789         print()
1790         self.result_show(calc)
1791
1792     elif func == 'power':
1793         self.matrix = np.zeros((a, a))
1794
1795         try:
1796             for i in range(a):
1797                 for j in range(a):
1798                     if self.matrix_entries[i][j].get() == '':
1799                         self.matrix_entries[i][j].insert(0, '0')
1800
1801                         self.matrix[i, j] =
float(self.matrix_entries[i][j].get())
1802         except ValueError:
1803             is_error = True
1804             self.errors('alpha')
1805         except:
1806             is_error = True
1807             self.errors('unexpected')
1808
1809         if not is_error:
1810             print("Matrix:", self.matrix, sep="\n")
1811             print()
1812
1813             if not self.power_entry.get(): num = 1
1814             else: num = int(self.power_entry.get())
1815
1816             start = time.perf_counter()
1817             calc = SimpleCalculation.matrix_power(self.matrix, num)
1818             finish = time.perf_counter()
1819
1820             self.time = round(finish - start, 3)
1821             print("Result:", calc, sep="\n")
1822             print()
1823             print("Time:", self.time)
1824             print()
1825             self.result_show(calc)
1826
1827     elif func == 'det':
1828         self.matrix = np.zeros((a, a))
1829
1830         try:
1831             for i in range(a):
1832                 for j in range(a):
1833                     if self.matrix_entries[i][j].get() == '':
```

```
1834         self.matrix_entries[i][j].insert(0, '0')
1835
1836         self.matrix[i, j] =
float(self.matrix_entries[i][j].get())
1837     except ValueError:
1838         is_error = True
1839         self.errors('alpha')
1840     except:
1841         is_error = True
1842         self.errors('unexpected')
1843
1844     if not is_error:
1845         print("Matrix:", self.matrix, sep="\n")
1846         print()
1847
1848         start = time.perf_counter()
1849         calc = SimpleCalculation.matrix_det(self.matrix)
1850         finish = time.perf_counter()
1851
1852         self.time = round(finish - start, 3)
1853         print("Result:", calc, sep="\n")
1854         print()
1855         print("time:", self.time)
1856         print()
1857         self.result_show(calc, 'det')
1858
1859     elif func == 'inv':
1860         self.matrix = np.zeros((a, a))
1861
1862         try:
1863             for i in range(a):
1864                 for j in range(a):
1865                     if self.matrix_entries[i][j].get() == '':
1866                         self.matrix_entries[i][j].insert(0, '0')
1867
1868                     self.matrix[i, j] =
float(self.matrix_entries[i][j].get())
1869         except ValueError:
1870             is_error = True
1871             self.errors('alpha')
1872         except:
1873             is_error = True
1874             self.errors('unexpected')
1875
1876     if not is_error:
1877         print("Matrix:", self.matrix, sep="\n")
1878         print()
1879
1880         try:
1881             start = time.perf_counter()
1882             calc = SimpleCalculation.matrix_inv(self.matrix)
1883             finish = time.perf_counter()
1884         except np.linalg.LinAlgError:
1885             is_error = True
1886             self.errors('singular')
1887
1888     if not is_error:
1889         self.time = round(finish - start, 3)
1890         print("Result:", calc, sep="\n")
1891         print()
```

```
1892         print("Time:", self.time)
1893         print()
1894         self.result_show(calc)
1895
1896     elif func == 'trans':
1897         self.matrix = np.zeros((a, b))
1898         try:
1899             for i in range(a):
1900                 for j in range(b):
1901                     if self.matrix_entries[i][j].get() == '':
1902                         self.matrix_entries[i][j].insert(0, '0')
1903
1904                         self.matrix[i, j] =
float(self.matrix_entries[i][j].get())
1905         except ValueError:
1906             is_error = True
1907             self.errors('alpha')
1908         except:
1909             is_error = True
1910             self.errors('unexpected')
1911
1912         if not is_error:
1913             print("Matrix:", self.matrix, sep="\n")
1914             print()
1915
1916             start = time.perf_counter()
1917             calc = SimpleCalculation.matrix_trans(self.matrix)
1918             finish = time.perf_counter()
1919
1920             self.time = round(finish - start, 3)
1921             print("Result:", calc)
1922             print()
1923             print("Time:", self.time)
1924             print()
1925             self.result_show(calc)
1926
1927     elif func == 'rank':
1928         self.matrix = np.zeros((a, b))
1929         try:
1930             for i in range(a):
1931                 for j in range(b):
1932                     if self.matrix_entries[i][j].get() == '':
1933                         self.matrix_entries[i][j].insert(0, '0')
1934
1935                         self.matrix[i, j] =
float(self.matrix_entries[i][j].get())
1936         except ValueError:
1937             is_error = True
1938             self.errors('alpha')
1939         except:
1940             is_error = True
1941             self.errors('unexpected')
1942
1943         if not is_error:
1944             print("Matrix:", self.matrix, sep="\n")
1945             print()
1946
1947             start = time.perf_counter()
1948             calc = SimpleCalculation.matrix_rank(self.matrix)
1949             finish = time.perf_counter()
```

```
1950
1951         self.time = round(finish - start, 3)
1952         print("Result:", calc)
1953         print()
1954         print("Time:", self.time)
1955         print()
1956         self.result_show(calc, 'rank')
1957
1958     elif func == 'trace':
1959         self.matrix = np.zeros((a, a))
1960
1961         try:
1962             for i in range(a):
1963                 for j in range(a):
1964                     if self.matrix_entries[i][j].get() == '':
1965                         self.matrix_entries[i][j].insert(0, '0')
1966
1967                         self.matrix[i, j] =
float(self.matrix_entries[i][j].get())
1968         except ValueError:
1969             is_error = True
1970             self.errors('alpha')
1971         except:
1972             is_error = True
1973             self.errors('unexpected')
1974
1975         if not is_error:
1976             print("Matrix:", self.matrix, sep="\n")
1977             print()
1978
1979             start = time.perf_counter()
1980             calc = SimpleCalculation.matrix_trace(self.matrix)
1981             finish = time.perf_counter()
1982
1983             self.time = round(finish - start, 3)
1984             print("Result:", calc)
1985             print()
1986             print("Time:", self.time)
1987             print()
1988             self.result_show(calc, 'trace')
1989
1990     def rand_calculate(self, func, a, b=0, c=0):
1991         """This method creates random matrices of given dimensions and
proceeds
with the desired calculation"""
1992         is_error = False
1993
1994         if func == 'add_sub':
1995             try:
1996                 a = int(a)
1997                 b = int(b)
1998                 if a < 2 or b < 2:
1999                     is_error = True
2000                     self.errors('dims')
2001             except ValueError:
2002                 is_error = True
2003                 self.errors('dims')
2004
2005         if not is_error:
2006             try:
```

```
2008         matrix_A = RandomMatrix.random_matrix(a, b)
2009         matrix_B = RandomMatrix.random_matrix(a, b)
2010
2011         if self.rand_op.get() == '+':
2012             start = time.perf_counter()
2013             calc = SimpleCalculation.matrix_add(matrix_A,
matrix_B)
2014             finish = time.perf_counter()
2015
2016         elif self.rand_op.get() == '-':
2017             start = time.perf_counter()
2018             calc = SimpleCalculation.matrix_sub(matrix_A,
matrix_B)
2019             finish = time.perf_counter()
2020         except MemoryError:
2021             is_error = True
2022             self.errors('memory')
2023         except:
2024             is_error = True
2025             self.errors('unexpected')
2026
2027         if not is_error:
2028             print('Matrix A:', matrix_A, sep='\n')
2029             print()
2030             print('Matrix B:', matrix_B, sep='\n')
2031             print()
2032             self.time = round(finish - start, 3)
2033             print('Result:', calc, sep='\n')
2034             print()
2035             print('Time:', self.time)
2036             print()
2037             self.rand_result_show(calc)
2038
2039         elif func == 'mul_num':
2040             try:
2041                 a = int(a)
2042                 b = int(b)
2043                 if a < 2 or b < 2:
2044                     is_error = True
2045                     self.errors('dims')
2046             except ValueError:
2047                 is_error = True
2048                 self.errors('dims')
2049
2050         if not is_error:
2051             try:
2052                 matrix = RandomMatrix.random_matrix(a, b)
2053
2054                 if self.rand_num_entry.get():
2055                     try:
2056                         num = float(self.rand_num_entry.get())
2057                     except ValueError:
2058                         is_error = True
2059                         self.errors('num')
2060                 else: num = 1
2061
2062                 if not is_error:
2063                     start = time.perf_counter()
2064                     calc = SimpleCalculation.matrix_mul_num(matrix, num)
2065                     finish = time.perf_counter()
```

```
2066         except MemoryError:
2067             is_error = True
2068             self.errors('memory')
2069         except:
2070             is_error = True
2071             self.errors('unexpected')
2072
2073         if not is_error:
2074             print('Matrix:', matrix, sep='\n')
2075             print()
2076             self.time = round(finish - start, 3)
2077             print('Result:', calc, sep='\n')
2078             print()
2079             print('Time:', self.time)
2080             print()
2081             self.rand_result_show(calc)
2082
2083     elif func == 'mul':
2084         try:
2085             a = int(a)
2086             b = int(b)
2087             c = int(c)
2088             if a < 2 or b < 2 or c < 2:
2089                 is_error = True
2090                 self.errors('dims')
2091         except ValueError:
2092             is_error = True
2093             self.errors('dims')
2094
2095         if not is_error:
2096             try:
2097                 matrix_A, matrix_B = RandomMatrix.two_random_matrices(a,
2098 b, c)
2099
2100                 if min(a, b, c) > 1000:
2101                     start_mp = time.perf_counter()
2102                     calc =
MultiprocessingCalculation.multiplication(matrix_A, matrix_B)
2103                     finish_mp = time.perf_counter()
2104
2105                     self.time = round(finish_mp - start_mp, 3)
2106                     self.rand_result_show(calc)
2107                 else:
2108                     start_mp = time.perf_counter()
2109                     calc =
MultiprocessingCalculation.multiplication(matrix_A, matrix_B)
2110                     finish_mp = time.perf_counter()
2111
2112                     start_simple = time.perf_counter()
2113                     SimpleCalculation.matrix_mul(matrix_A, matrix_B)
2114                     finish_simple = time.perf_counter()
2115             except MemoryError:
2116                 is_error = True
2117                 self.errors('memory')
2118             except:
2119                 is_error = True
2120                 self.errors('unexpected')
2121
2122             self.time = round(min(finish_simple - start_simple,
finish_mp - start_mp), 3)
```



```
2122         if not is_error:
2123             print('Matrix A:', matrix_A, sep='\n')
2124             print()
2125             print('Matrix B:', matrix_B, sep='\n')
2126             print()
2127             self.time = round(min(finish_simple - start_simple,
finish_mp - start_mp), 3)
2128             print('Result:', calc, sep='\n')
2129             print('Time w/o multiprocessing:', finish_simple -
start_simple)
2130             print('Tim w/ multiprocessing:', finish_mp - start_mp)
2131             print()
2132             self.rand_result_show(calc)
2133
2134     elif func == 'power':
2135         try:
2136             a = int(a)
2137             if a < 2:
2138                 is_error = True
2139                 self.errors('dims')
2140         except ValueError:
2141             is_error = True
2142             self.errors('dims')
2143
2144         if not is_error:
2145             try:
2146                 matrix = RandomMatrix.random_matrix(a, a)
2147
2148                 if self.rand_power_entry.get():
2149                     if not self.rand_power_entry.get().isdigit():
2150                         is_error = True
2151                         self.errors('power')
2152                     else:
2153                         num = int(self.rand_power_entry.get())
2154                 else: num = 1
2155                 if not is_error:
2156                     start = time.perf_counter()
2157                     calc = SimpleCalculation.matrix_power(matrix, num)
2158                     finish = time.perf_counter()
2159             except MemoryError:
2160                 is_error = True
2161                 self.errors('memory')
2162             except:
2163                 is_error = True
2164                 self.errors('unexpected')
2165
2166         if not is_error:
2167             print('Matrix:', matrix, sep='\n')
2168             print()
2169             self.time = round(finish - start, 3)
2170             print('Result:', calc, sep='\n')
2171             print()
2172             print('Time:', self.time)
2173             print()
2174             self.rand_result_show(calc)
2175
2176     elif func == 'det':
2177         try:
2178             a = int(a)
2179             if a < 2:
```

```
2180         is_error = True
2181         self.errors('dims')
2182     except ValueError:
2183         is_error = True
2184         self.errors('dims')
2185
2186     if not is_error:
2187         try:
2188             matrix = RandomMatrix.random_matrix(a, a)
2189
2190             start = time.perf_counter()
2191             calc = SimpleCalculation.matrix_det(matrix)
2192             finish = time.perf_counter()
2193         except MemoryError:
2194             is_error = True
2195             self.errors('memory')
2196         except:
2197             is_error = True
2198             self.errors('unexpected')
2199
2200     if not is_error:
2201         print('Matrix:', matrix, sep='\n')
2202         print()
2203         self.time = round(finish - start, 3)
2204         print('Result:', calc, sep='\n')
2205         print()
2206         print('Time:', self.time)
2207         print()
2208         self.rand_result_show(calc, 'det')
2209
2210     elif func == 'inv':
2211         try:
2212             a = int(a)
2213             if a < 2:
2214                 is_error = True
2215                 self.errors('dims')
2216         except ValueError:
2217             is_error = True
2218             self.errors('dims')
2219
2220     if not is_error:
2221         try:
2222             matrix = RandomMatrix.random_matrix(a, a)
2223
2224             start = time.perf_counter()
2225             calc = SimpleCalculation.matrix_inv(matrix)
2226             finish = time.perf_counter()
2227         except MemoryError:
2228             is_error = True
2229             self.errors('memory')
2230         except:
2231             is_error = True
2232             self.errors('unexpected')
2233
2234     if not is_error:
2235         print('Matrix:', matrix, sep='\n')
2236         print()
2237         self.time = round(finish - start, 3)
2238         print('Result:', calc, sep='\n')
2239         print()
```

```
2240         print('Time:', self.time)
2241         print()
2242         self.rand_result_show(calc)
2243
2244     elif func == 'trans':
2245         try:
2246             a = int(a)
2247             b = int(b)
2248             if a < 2 or b < 2:
2249                 is_error = True
2250                 self.errors('dims')
2251         except ValueError:
2252             is_error = True
2253             self.errors('dims')
2254
2255     if not is_error:
2256         try:
2257             matrix = RandomMatrix.random_matrix(a, b)
2258
2259             start = time.perf_counter()
2260             calc = SimpleCalculation.matrix_trans(matrix)
2261             finish = time.perf_counter()
2262         except MemoryError:
2263             is_error = True
2264             self.errors('memory')
2265         except:
2266             is_error = True
2267             self.errors('unexpected')
2268
2269     if not is_error:
2270         print('Matrix:', matrix, sep='\n')
2271         print()
2272         self.time = round(finish - start, 3)
2273         print('Result:', calc, sep='\n')
2274         print()
2275         print('Time:', self.time)
2276         print()
2277         self.rand_result_show(calc)
2278
2279     elif func == 'rank':
2280         try:
2281             a = int(a)
2282             b = int(b)
2283             if a < 2 or b < 2:
2284                 is_error = True
2285                 self.errors('dims')
2286         except ValueError:
2287             is_error = True
2288             self.errors('dims')
2289
2290     if not is_error:
2291         try:
2292             matrix = RandomMatrix.random_matrix(a, b)
2293
2294             start = time.perf_counter()
2295             calc = SimpleCalculation.matrix_rank(matrix)
2296             finish = time.perf_counter()
2297         except MemoryError:
2298             is_error = True
2299             self.errors('memory')
```

```
2300         except:
2301             is_error = True
2302             self.errors('unexpected')
2303
2304         if not is_error:
2305             print('Matrix:', matrix, sep='\n')
2306             print()
2307             self.time = round(finish - start, 3)
2308             print('Result:', calc, sep='\n')
2309             print()
2310             print('Time:', self.time)
2311             print()
2312             self.rand_result_show(calc, 'rank')
2313
2314     elif func == 'trace':
2315         try:
2316             a = int(a)
2317             if a < 2:
2318                 is_error = True
2319                 self.errors('dims')
2320         except ValueError:
2321             is_error = True
2322             self.errors('dims')
2323
2324         if not is_error:
2325             try:
2326                 matrix = RandomMatrix.random_matrix(a, a)
2327
2328                 start = time.perf_counter()
2329                 calc = SimpleCalculation.matrix_trace(matrix)
2330                 finish = time.perf_counter()
2331             except MemoryError:
2332                 is_error = True
2333                 self.errors('memory')
2334             except:
2335                 is_error = True
2336                 self.errors('unexpected')
2337
2338         if not is_error:
2339             print('Matrix:', matrix, sep='\n')
2340             print()
2341             self.time = round(finish - start, 3)
2342             print('Result:', calc, sep='\n')
2343             print()
2344             print('Time:', self.time)
2345             print()
2346             self.rand_result_show(calc, 'trace')
2347
2348     def clear_cells(self, list):
2349         """This method clears all entry boxes"""
2350         for i in range(len(list)):
2351             for entry in list[i]:
2352                 entry.delete(0, 'end')
2353
2354     def fill_zeros(self, list):
2355         """This method fills all empty entry boxes with the number '0'"""
2356         for i in range(len(list)):
2357             for entry in list[i]:
2358                 if entry.get() == '':
2359                     entry.insert(0, '0')
```

```
2360
2361     def fill_ones(self, list):
2362         """This method fills all empty entry boxes with the number '1'"""
2363         for i in range(len(list)):
2364             for entry in list[i]:
2365                 if entry.get() == '':
2366                     entry.insert(0, '1')
2367
2368     def mem_sv(self, list):
2369         """This method saves the inputted matrix in the memory for future
2370 use"""
2371         is_error = False # this variable checks if there is error(element
isaplha)
2372         self.matrix_saved = np.zeros((len(list), len(list[0])))
2373         try:
2374             self.matrix_saved = np.zeros((len(list), len(list[0])))
2375             for i in range(len(list)):
2376                 for j in range(len(list[0])):
2377                     self.matrix_saved[i][j] = list[i][j].get()
2378         except ValueError:
2379             is_error = True
2380             self.errors('mem_save')
2381         except:
2382             is_error = True
2383             self.errors('unexpected')
2384
2385         if not is_error:
2386             print("Matrix in memory:", self.matrix_saved, sep="\n")
2387             print()
2388
2389     def mem_ld(self, list):
2390         """This method loads the matrix saved in the memory into the entry
2391 boxes"""
2392         try:
2393             if self.matrix_saved.shape[0] != len(list) or
self.matrix_saved.shape[1] != len(list[0]):
2394                 self.errors('mem_load_dims')
2395             else:
2396                 GUI.clear_cells(self, list)
2397                 for i in range(len(list)):
2398                     for j in range(len(list[0])):
2399                         list[i][j].insert(0, self.matrix_saved[i][j])
2400         except AttributeError:
2401             self.errors('mem_load_empty')
2402
2403     def result_show(self, result, func=''):
2404         """This method creates a new window displaying the result"""
2405         self.result_win = Toplevel(root, bg=self.color_bg1)
2406         self.result_win.iconbitmap('matrix_ico.ico')
2407         self.result_win.title('Matrix Calculator')
2408         self.result_win.geometry("600x400")
2409
2410         self.f_res_text = Frame(self.result_win, bg=self.color_bg1)
2411         self.f_res_text.pack(side='top', fill='x', padx=40, pady=(40, 20))
2412
2413         self.f_main_res = Frame(self.result_win, bg=self.color_bg1, padx=5,
pady=5)
2414         self.f_main_res.pack(side='top', padx=40)
```

```
2415
2416     self.f_time = Frame(self.result_win, bg=self.color_bg1)
2417     self.f_time.pack(side='bottom', fill='x', padx=40, pady=(10, 40))
2418
2419     self.time_text = Label(self.f_time, text=f'Computation time:
{self.time} seconds', font=('Arial', 10),
2420                             bg=self.color_bg1, fg=self.color_text2)
2421     self.time_text.pack(anchor='e')
2422
2423     self.result_text = Label(self.f_res_text, text='Result:',
font=('Arial', 12), bg=self.color_bg1,
2424                             fg=self.color_text2)
2425     self.result_text.pack(anchor='w')
2426
2427     labels = []
2428     width = 3
2429
2430     if func == 'det':
2431         self.result_win.geometry("600x300")
2432
2433         self.result = Label(self.f_main_res, text=f'Matrix determinant is
{round(result, 2)}', font=('Arial', 15),
2434                             bg=self.color_bg1, fg=self.color_text2)
2435         self.result.pack(anchor='n')
2436
2437     elif func == 'rank':
2438         self.result_win.geometry("600x300")
2439
2440         self.result = Label(self.f_main_res, text=f'Matrix rank is
{round(result, 2)}', font=('Arial', 15),
2441                             bg=self.color_bg1, fg=self.color_text2)
2442         self.result.pack(anchor='n')
2443
2444     elif func == 'trace':
2445         self.result_win.geometry("600x300")
2446
2447         self.result = Label(self.f_main_res, text=f'Matrix trace is
{round(result, 2)}', font=('Arial', 15),
2448                             bg=self.color_bg1, fg=self.color_text2)
2449         self.result.pack(anchor='n')
2450
2451     else:
2452         self.f_result = Frame(self.f_main_res, bg=self.color_bg2)
2453         self.f_result.pack(anchor='w')
2454
2455         self.result_win.geometry("830x550")
2456
2457         Label(self.f_result, text=' ', font=('Arial', 12),
bg=self.color_bg1, fg='grey',
2458             width=2).grid(row=0, column=0, padx=2, pady=2)
2459
2460         for j in range(len(result[0])):
2461             label = Label(self.f_result, text='A'+f'{j + 1}',
font=('Arial', 12), bg=self.color_bg1, fg='grey')
2462             label.grid(row=0, column=j+1, padx=2)
2463             labels.append(label)
2464
2465         for i in range(len(result)):
2466             Label(self.f_result, text=i + 1, font=('Arial', 12),
bg=self.color_bg1, fg='grey',
```

```
2467         width=2).grid(row=i+1, column=0, padx=2)
2468     for j in range(len(result[0])):
2469
2470         if result[i][j].is_integer():
2471             number = int(result[i][j])
2472         else:
2473             number = result[i][j]
2474
2475         if len(str(number)) > width:
2476             width = len(str(number))
2477
2478         label = Label(self.f_result, text=number, font=('Arial',
12), bg=self.color_bg1,
2479                        fg=self.color_text2)
2480         label.grid(row=i + 1, column=j + 1, padx=2, pady=2)
2481         labels.append(label)
2482
2483     for label in labels: label.configure(width=width)
2484
2485     def rand_result_show(self, result, func=''):
2486         """This method creates a new window displaying the result (from the
2487         random matrices)"""
2488         self.result_win = Toplevel(root, bg=self.color_bg1)
2489         self.result_win.iconbitmap('matrix_ico.ico')
2490         self.result_win.title('Matrix Calculator')
2491
2492         self.f_res_text = Frame(self.result_win, bg=self.color_bg1)
2493         self.f_res_text.pack(side='top', fill='x', padx=40, pady=(40, 20))
2494
2495         self.f_main_res = Frame(self.result_win, bg=self.color_bg1, padx=5,
pady=5)
2496         self.f_main_res.pack(side='top', padx=40)
2497
2498         self.f_time = Frame(self.result_win, bg=self.color_bg1)
2499         self.f_time.pack(side='bottom', fill='x', padx=40, pady=(10, 40))
2500
2501         self.time_text = Label(self.f_time, text=f'Computation time:
{self.time} seconds', font=('Arial', 10),
2502                                bg=self.color_bg1, fg=self.color_text2)
2503         self.time_text.pack(anchor='e')
2504
2505         self.result_text = Label(self.f_res_text, text='Result:',
font=('Arial', 12), bg=self.color_bg1,
2506                                fg=self.color_text2)
2507         self.result_text.pack(anchor='w')
2508
2509         if func == 'det':
2510             self.result_win.geometry("600x300")
2511
2512             self.result = Label(self.f_main_res, text=f'Matrix determinant is
{round(result, 2)}', font=('Arial', 15),
2513                                bg=self.color_bg1, fg=self.color_text2)
2514             self.result.pack(anchor='n')
2515
2516         elif func == 'rank':
2517             self.result_win.geometry("600x300")
2518
2519             self.result = Label(self.f_main_res, text=f'Matrix rank is
{round(result, 2)}', font=('Arial', 15),
2520                                bg=self.color_bg1, fg=self.color_text2)
```

```
2521         self.result.pack(anchor='n')
2522
2523     elif func == 'trace':
2524         self.result_win.geometry("600x300")
2525
2526         self.result = Label(self.f_main_res, text=f'Matrix trace is
2527 {round(result, 2)}', font=('Arial', 15),
2528                             bg=self.color_bg1, fg=self.color_text2)
2529         self.result.pack(anchor='n')
2530
2531     else:
2532         self.result_win.geometry("830x550")
2533
2534         self.scroll_x = Scrollbar(self.f_main_res, orient="horizontal")
2535         self.scroll_y = Scrollbar(self.f_main_res, orient="vertical")
2536         self.scroll_x.pack(side='bottom', fill='x')
2537         self.scroll_y.pack(side='right', fill='y')
2538
2539         self.t_result = Text(self.f_main_res, font=('Arial', 12),
2540                             bg=self.color_bg2, fg=self.color_text2,
2541                             spacing1=10, height=12, width=80,
2542                             relief=GROOVE,
2543                             xscrollcommand=self.scroll_x.set,
2544                             yscrollcommand=self.scroll_y.set, wrap='none')
2545         self.t_result['font'] = ('Arial', 12)
2546         self.t_result.pack(side='left')
2547
2548         if len(str(np.amax(result))) < 7: tabs = 1
2549         elif len(str(np.amax(result))) < 12: tabs = 2
2550         elif len(str(np.amax(result))) < 17: tabs = 3
2551         elif len(str(np.amax(result))) < 22: tabs = 4
2552         elif len(str(np.amax(result))) < 27: tabs = 5
2553         else: tabs = 6
2554
2555         for i in range(len(result)):
2556             for j in range(len(result[0])):
2557                 self.t_result.insert('end', result[i][j])
2558                 if j == len(result[0])-1:
2559                     self.t_result.insert('end', '\n')
2560                 else:
2561                     self.t_result.insert('end', '\t'*tabs)
2562
2563         self.scroll_x.config(command=self.t_result.xview)
2564         self.scroll_y.config(command=self.t_result.yview)
2565
2566     def errors(self, type):
2567         if type == 'alpha':
2568             mb.showerror(title='Error', message="Array's elements must be
2569 numbers")
2570         elif type == 'num':
2571             mb.showerror(title='Error', message='Multiplying number input must
2572 be a number')
2573         elif type == 'unexpected':
2574             mb.showerror(title='Error', message='An unexpected error has
2575 occurred')
2576         elif type == 'power':
2577             mb.showerror(title='Error', message='Power number input must be a
2578 positive integer')
2579         elif type == 'memory':
2580             mb.showerror(title='Error', message='Input values are too big. Try
```



```
smaller values')
2573         elif type == 'dims':
2574             mb.showerror(title='Error', message='Dimension inputs must be
integers larger or equal than 2')
2575         elif type == 'singular':
2576             mb.showerror(title='Error', message='Matrix is singular thus is
not invertible')
2577         elif type == 'mem_save':
2578             mb.showerror(title='Error', message="Array's elements in saved
matrix must be numbers")
2579         elif type == 'mem_load_empty':
2580             mb.showerror(title='Error', message='No matrix in memory')
2581         elif type == 'mem_load_dims':
2582             mb.showerror(title='Error', message='Matrix in memory does not
match current dimensions')
2583
2584
2585     if __name__ == '__main__':
2586         root = Tk()
2587         gui = GUI(root)
2588         root.mainloop()
```

## Βιβλιογραφία – Πηγές πληροφόρησης

1. Python - Εισαγωγή στους Υπολογιστές, Ν. Αβούρης, Μ. Κουκιάς, Β. Παλιουράς, Κ. Σγάρμπας, Πανεπιστημιακές Εκδόσεις Κρήτης, 2016
2. Διαφάνειες – προσφερόμενο υλικό από τις Διαλέξεις του μαθήματος
3. Επίσημες ιστοσελίδες των βιβλιοθηκών που χρησιμοποιήθηκαν:
  - <https://NumPy.org/doc/stable/>
  - <https://docs.python.org/3/library/multiprocessing.html>
  - <https://docs.python.org/3/library/tkinter.html>
4. Άλλες πηγές πληροφόρησης:
  - <https://www.geeksforgeeks.org/>
  - <https://stackoverflow.com/>
  - <https://www.w3schools.com/>
  - [https://en.wikipedia.org/wiki/Matrix\\_multiplication\\_algorithm](https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm)