Οικονόμου Πολυχρόνης
Παπαδόπουλος Πολύκαρπος

Εισαγωγή στον προγραμματισμό

MATLAB

ΚΑΛΛΙΠΟΣ ανοικτές ακαδημαϊκές εκδόσεις

Εθνικό Πρόγραμμα Ανάπτυξης 2021-2025

Εισαγωγή στον προγραμματισμό MATLAB

Τίτλος πρωτοτύπου: <<*Εισαγωγή στον προγραμματισμό*>>

Εισαγωγή στον προγραμματισμό

MATLAB

Πολυχρόνης Οικονόμου
Αναπληρωτής Καθηγητής
Πανεπιστήμιο Πατρών

Πολύκαρπος Παπαδόπουλος
Επίκουρος Καθηγητής
Πανεπιστήμιο Πατρών

ΚΑΛΛΙΠΟΣ
ΑΝΟΙΚΤΕΣ ΑΚΑΔΗΜΑΪΚΕΣ ΕΚΔΟΣΕΙΣ
www.kallipos.gr

Συντελεστές έκδοσης

Γλωσσική επιμέλεια: Βασιλική Τυραϊδή

Τεχνική επεξεργασία: Πολυχρόνης Οικονόμου

Κεντρική Ομάδα Υποστήριξης

Γλωσσικός Έλεγχος: Γεωργία Τριανταφυλλίδου

Γραφιστικός Έλεγχος: Χρήστος Κεντρωτής

Βιβλιοθηκονομική Επεξεργασία: Αλέξανδρος Ηλιάκης

ΚΑΛΛΙΠΟΣ

Εθνικό Μετσόβιο Πολυτεχνείο

Ηρώων Πολυτεχνείου 9

15780 Ζωγράφου

www.kallipos.gr

Βιβλιογραφική αναφορά: Οικονόμου, Π., & Παπαδόπουλος, Π. (2023). *Εισαγωγή στον προγραμματισμό – MATLAB* [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις.

Διαθέσιμο στο: http://dx.doi.org/10.57713/kallipos-145

ISBN: **978-618-5667-86-3**

Στη Βασιλική και τη Σοφιάννα

Π.Ο.

Στην Ελένη και τον Ντίνο

Π.Π.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΡΟΛΟΓΟΣ

Τις τελευταίες δεκαετίες, η διείσδυση του προγραμματισμού σε διάφορες επιστημονικές και τεχνολογικές δραστηριότητες του σύγχρονου τρόπου ζωής αυξάνεται ραγδαία. Η διαδικασία επεξεργασίας και ανάλυσης μεγάλου όγκου δεδομένων και η ανάγκη για τη μελέτη περίπλοκων φυσικών συστημάτων και σύνθετων τεχνολογικών εφαρμογών απαιτούν την επέκταση ή και την ανάπτυξη νέων, ευέλικτων και αξιόπιστων προγραμμάτων, δηλαδή τη δημιουργία κώδικα. Στην κατεύθυνση αυτή, γίνεται φανερό ότι η εξοικείωση, σε μικρότερο ή μεγαλύτερο βαθμό, με τεχνικές προγραμματισμού είναι ένα πολύτιμο εργαλείο, ειδικά για κατηγορίες επιστημόνων, όπως οι μηχανικοί, οι οικονομολόγοι και οι ιατροί, οι οποίοι απαιτείται να λαμβάνουν κρίσιμες αποφάσεις, συχνά σε πραγματικό χρόνο, και να προβαίνουν σε εξειδικευμένες ενέργειες, χωρίς να βασίζονται αποκλειστικά σε άλλους.

Στη λογική αυτή, στόχος του παρόντος βιβλίου είναι η εξοικείωση των αναγνωστών με βασικά στοιχεία ανάπτυξης προγραμματιστικών δεξιοτήτων και, συνεπώς, μπορεί να χρησιμοποιηθεί ως ένα εισαγωγικό σύγγραμμα στη λογική του προγραμματισμού. Μέσο για την επίτευξη του σκοπού αυτού είναι το Matlab, το οποίο είναι ένα λογισμικό για την ανάπτυξη κώδικα και προγραμμάτων, δηλαδή είναι μια γλώσσα προγραμματισμού.

Η ύλη ανά κεφάλαιο του βιβλίου έχει ως ακολούθως. Στο **Κεφάλαιο 1** παρουσιάζονται οι βασικές αρχές του προγραμματισμού και το περιβάλλον του Matlab, καθώς και τα βασικά εργαλεία και λειτουργίες του Matlab. Στο **Κεφάλαιο 2** παρουσιάζονται οι βασικές μαθηματικές και λογικές συναρτήσεις, οι οποίες αποτελούν τα βασικά εργαλεία για την ανάπτυξη προγραμμάτων. Στη συνέχεια, στο **Κεφάλαιο 3** παρουσιάζονται οι εντολές με τις οποίες μπορούν να υλοποιηθούν δομές ελέγχου της ροής ενός προγράμματος, ενώ στο **Κεφάλαιο 4** παρατίθενται και αναλύονται οι εντολές με τις οποίες μπορούν να υλοποιηθούν και να ελεγχθούν επαναληπτικές δομές.

Στα **Κεφάλαια 5 και 6** παρουσιάζονται οι βασικοί τρόποι κατασκευής και διαμόρφωσης διδιάστατων και τριδιάστατων γραφικών παραστάσεων, αντίστοιχα. Στο **Κεφάλαιο 7** η προσοχή στρέφεται στα πολυώνυμα και στους μιγαδικούς αριθμούς, ενώ στο **Κεφάλαιο 8** στους πίνακες και στον χειρισμό τους από το Matlab.

Στο **Κεφάλαιο 9** παρουσιάζεται η διαδικασία κατασκευής συναρτήσεων από τον ίδιο τον χρήστη με στόχο την ανάπτυξη κώδικα επίλυσης πιο σύνθετων προβλημάτων. Στο **Κεφάλαιο 10** παρουσιάζονται οι βασικές εντολές που αφορούν τα συμβολικά μαθηματικά στο Matlab. Συγκεκριμένα, παρουσιάζονται οι τρόποι ορισμού και χειρισμού συμβολικών μεταβλητών και συμβολικών συναρτήσεων. Στο τελευταίο κεφάλαιο, το

Κεφάλαιο 11, αναπτύσσονται ολοκληρωμένα προγράμματα με στόχο την επίλυση σύνθετων και απαιτητικών προβλημάτων, χρησιμοποιώντας συνδυαστικά τις γνώσεις όλων των προηγούμενων κεφαλαίων. Το παρόν σύγγραμμα ολοκληρώνεται με **δύο παραρτήματα** και το **ευρετήριο όρων**.

Αξίζει να επισημανθεί ότι σε κάθε κεφάλαιο του παρόντος συγγράμματος περιέχονται βιβλιογραφικές αναφορές για περαιτέρω μελέτη, ενώ υπάρχουν ασκήσεις αυτοαξιολόγησης. Η συγγραφική ομάδα προτείνει αυτές οι ασκήσεις να λύνονται παράλληλα με τη μελέτη του κάθε κεφαλαίου. Η αυτοαξιολόγηση του αναγνώστη, στην κατανόηση του υλικού που έχει προηγηθεί, μπορεί να επιτευχθεί με βάση τις αναλυτικές λύσεις που δίνονται στο τέλος του κάθε κεφαλαίου.

Εν αντιθέσει, η συλλογή των άλυτων ασκήσεων που παρατίθεται στο τέλος του κάθε κεφαλαίου, εκτός του Κεφαλαίου 11, προτείνεται να αποτελεί αντικείμενο μελέτης μετά την ολοκλήρωση κάθε κεφαλαίου και για αυτόν τον λόγο οι ασκήσεις αυτές δεν ακολουθούν τη σειρά εμφάνισης των εννοιών στο κάθε κεφάλαιο.

Οι συγγραφείς επιθυμούν να ευχαριστήσουν, από τη θέση αυτή, τους πανεπιστημιακούς τους δασκάλους που τους μετέδωσαν την αγάπη για τον Προγραμματισμό και τα Μαθηματικά αλλά και το πάθος τους για τη διδασκαλία. Ιδιαίτερες, όμως, ευχαριστίες αποδίδονται στις οικογένειές τους για την υπομονή και την υποστήριξη που έδειξαν καθ' όλη τη διάρκεια της συγγραφής του βιβλίου.

Ευχόμαστε το παρόν σύγγραμμα να αποτελέσει έναν πλήρη οδηγό για όσους διδάσκουν ή διδάσκονται ή απλώς επιθυμούν να έρθουν σε μια πρώτη επαφή με τον Προγραμματισμό. Επίσης, ο τρόπος συγγραφής του συγγράμματος με (α) την παράθεση προαπαιτούμενων γνώσεων, (β) την καταγραφή των προσδοκώμενων μαθησιακών αποτελεσμάτων, (γ) την αναφορά επιστημονικών όρων στην αρχή κάθε κεφαλαίου, καθώς και (δ) τη χρήση πλήθους ασκήσεων αυτοαξιολόγησης αλλά και άλυτων, επιτρέπει τη χρήση του και για την εξ αποστάσεως εκπαίδευση.

Τέλος, είναι απαραίτητο να αναφερθεί ότι παρότι έγινε προσπάθεια ελαχιστοποίησης λαθών, αστοχιών και παραλείψεων, είναι σχεδόν σίγουρο ότι τέτοια έχουν παραμείνει στο παρόν κείμενο. Για κάθε παράλειψη ή λάθος, υπεύθυνα είναι τα μέλη της συγγραφικής ομάδας και κάθε παρατήρηση και σχόλιο για βελτίωση του παρόντος συγγράμματος είναι ευπρόσδεκτα.

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ ΚΑΙ ΣΤΟ MATLAB

Σύνοψη:

Σε αυτό το κεφάλαιο παρουσιάζονται οι βασικές αρχές του προγραμματισμού και το περιβάλλον του Matlab. Επιπρόσθετα, παρατίθενται τα βασικά εργαλεία και οι κύριες λειτουργίες του Matlab, τα οποία θα μας επιτρέψουν στα επόμενα κεφάλαια να εκμεταλλευτούμε τις δυνατότητές του και να επιλύσουμε σύνθετα προβλήματα.

Προαπαιτούμενη γνώση: Βασικές γνώσεις χειρισμού ηλεκτρονικού υπολογιστή.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του κεφαλαίου αυτού, θα γνωρίζετε:

- καλές πρακτικές προγραμματισμού,
- το περιβάλλον του Matlab,
- τη δημιουργία μεταβλητών,
- τη δημιουργία αρχείων script,
- τον τρόπο ανάκτησης πληροφοριών για τις εντολές του Matlab.

Γλωσσάριο επιστημονικών όρων

- Command Window
- Αρχείο script
- Επιστημονική μορφή αριθμού
- Πίνακας - διάνυσμα
- Τελεστές αριθμητικών πράξεων

1.1 Εισαγωγή

Ο όγκος των πληροφοριών και των δεδομένων, που ο σύγχρονος άνθρωπος καλείται να επεξεργαστεί και να διαχειριστεί στην καθημερινότητά του αυξάνεται συνεχώς και η συνεχής ροή πληροφορίας διέπει σχεδόν όλες τις όψεις της καθημερινότητας. Από τη βέλτιστη δρομολόγηση ενός οχήματος από το ένα σημείο στο άλλο, μέχρι την παρακολούθηση, σε πραγματικό χρόνο, της λειτουργίας ενός αεροπλάνου, όλες οι δραστηριότητες περιλαμβάνουν δεδομένα και πληροφορίες στις οποίες πρέπει να γίνει η κατάλληλη επεξεργασία, έτσι ώστε να παραχθούν χρήσιμες πληροφορίες για τον τελικό χρήστη.

Η διαδικασία επεξεργασίας και ανάλυσης της διαθέσιμης πληροφορίας απαιτεί την επέκταση ή και την ανάπτυξη νέων ευέλικτων προγραμμάτων, δηλαδή κώδικα. Μάλιστα, η απαίτηση αυτή έχει γίνει σήμερα πιο επιτακτική εξαιτίας του όγκου των δεδομένων που προέρχονται από διάφορες πηγές. Φυσικά, η μη εξοικείωση του συνόλου των ανθρώπων με τον προγραμματισμό δεν είναι τόσο κρίσιμη όσο ο ψηφιακός αναλφαβητισμός. Βασικός λόγος για αυτό είναι ότι τα περισσότερα προγράμματα είναι δομημένα έτσι ώστε να είναι φιλικά προς τον τελικό χρήστη. Παρ΄ όλα αυτά, δεν πρέπει να παραγνωρίσουμε το γεγονός ότι «οι άνθρωποι που δεν έχουν γνώσεις προγραμματισμού θα χρειαστεί να βασίζονται (συνεχώς) σε άλλους για να τους βοηθήσουν στην πορεία της ζωής τους», όπως έγραψε και η Vee (2017).

Η τελευταία διαπίστωση γίνεται πιο προβληματική όταν αφορά σε κατηγορίες ατόμων, όπως οι μηχανικοί, οι οικονομολόγοι και οι ιατροί, οι κατηγορίες δηλαδή των επιστημόνων οι οποίοι απαιτείται να μην βασίζονται αποκλειστικά σε άλλους, αφού οφείλουν να λαμβάνουν κρίσιμες, συχνά σε πραγματικό χρόνο, αποφάσεις σε ειδικές, ξεχωριστές περιπτώσεις και να προβαίνουν σε εξειδικευμένες ενέργειες. Πολύτιμος σύμμαχός τους σε αυτήν τη διαδικασία είναι η εξοικείωσή τους, σε μικρότερο ή μεγαλύτερο βαθμό, με τεχνικές προγραμματισμού, έτσι ώστε να μπορούν να αναπτύσσουν αλλά και να τροποποιούν εργαλεία προσαρμόζοντάς τα στις δικές τους ανάγκες και στις πληροφορίες/δεδομένα που έχουν στη διάθεσή τους.

Προτού γίνει η παρουσίαση του περιβάλλοντος του Matlab, το οποίο θα χρησιμοποιηθεί για την εισαγωγή στον προγραμματισμό, κρίνεται απαραίτητο να συζητήσουμε κάποιες καλές πρακτικές προγραμματισμού. Οι καλές αυτές πρακτικές ευελπιστούμε να βοηθήσουν τον αναγνώστη όχι μόνο στο να παρακολουθήσει τα επόμενα κεφάλαια του συγγράμματος αυτού αλλά και να τον κατευθύνουν στην ανάπτυξη των προγραμματιστικών του δεξιοτήτων.

1.2 Καλές πρακτικές προγραμματισμού

Η ανάπτυξη προγραμμάτων και κώδικα δεν είναι απλώς ο μετασχηματισμός της ανθρώπινης λογικής σε μια γλώσσα που μπορεί να επεξεργαστεί ένας ηλεκτρονικός υπολογιστής. Πρέπει να έχουμε πάντα κατά νου ότι οι υπολογιστές δεν είναι εξυπνότεροι από τους ανθρώπους και μπορούν να εκτελέσουν μόνο προγράμματα και εντολές που έχουν δημιουργήσει οι άνθρωποι. Η διαφορά και το πλεονέκτημα των υπολογιστών είναι ότι αυτοί είναι εξαιρετικά πιο γρήγοροι από εμάς.

Η παραπάνω διαπίστωση μας οδηγεί στην πρώτη και βασικότερη αρχή για την ανάπτυξη ενός προγράμματος. Κάθε πρόγραμμα στοχεύει στην επίλυση ενός προβλήματος, του οποίου τη λύση ή έστω την αλγοριθμική επίλυσή του πρέπει να έχουμε ήδη σκεφτεί και περιγράψει αναλυτικά προτού τη μετατρέψουμε σε κώδικα. Η λύση πρέπει να έχει περιγραφτεί με διακριτά και ξεκάθαρα βήματα. Τα βήματα αυτά θα αποτελούν τη ραχοκοκαλιά γύρω από την οποία θα αναπτύξουμε τα επιμέρους κομμάτια του κώδικά μας.

Το σπάσιμο του κώδικα σε επιμέρους κομμάτια μας οδηγεί στη δεύτερη αρχή για την ανάπτυξη ενός καλού κώδικα. Πρέπει να σκεφτόμαστε απλά, σταδιακά, επικεντρώνοντας την προσοχή μας σε ένα προς ένα τα βήματα επίλυσης, χωρίς ωστόσο να χάνουμε τη συνολική εικόνα. Χρειάζονται μικρές νίκες, προτού

κατακτήσουμε πλήρως τον στόχο.

Πέρα όμως από τις προαναφερθείσες γενικές αρχές, πρέπει να αναφέρουμε και τις ακόλουθες καλές πρακτικές, οι οποίες μπορούν να βοηθήσουν τον αναγνώστη:

- στο να κάνει τα πρώτα βήματά του στον προγραμματισμό πιο γρήγορα και πιο απλά αλλά και
- στο να προβεί στη δημιουργία προγραμμάτων, τα οποία να μπορούν να επεκταθούν ή να επαναχρησιμοποιηθούν χωρίς δυσκολία, ενώ να υπάρχει ταυτόχρονα και η δυνατότητα να καθίσταται ο κώδικας κατανοητός και εύκολα χειρίσιμος από τρίτους.

Οι καλές αυτές πρακτικές μπορούν να συνοψιστούν στα ακόλουθα σημεία:

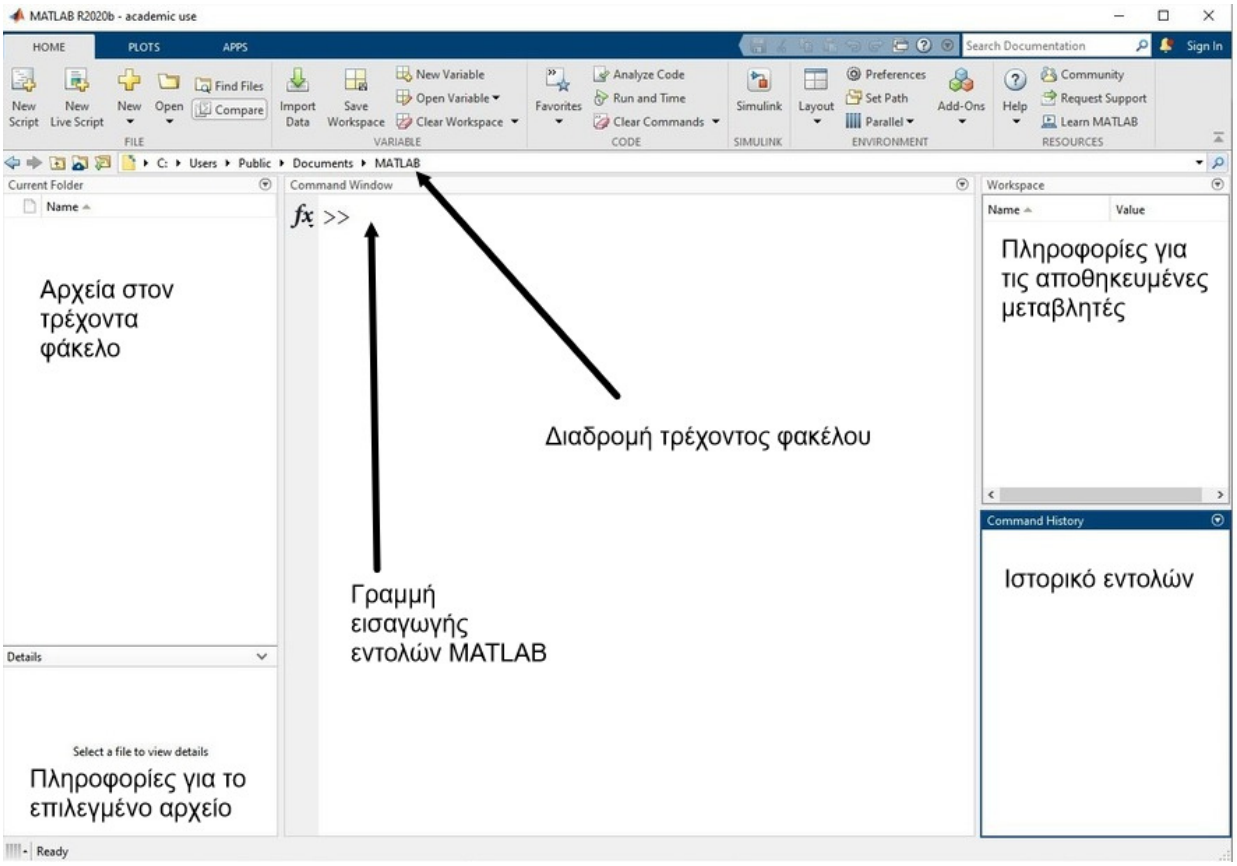
- κρατήστε τον κώδικα όσο πιο απλό γίνεται,
- ελέγξτε ότι ο κώδικάς σας πραγματοποιεί αυτό που πρέπει να υλοποιήσει, εξετάζοντας διεξοδικά όλα τα πιθανά σενάρια,
- μην γράφετε τμήματα κώδικα που δεν εξυπηρετούν την επίλυση του προβλήματος που αντιμετωπίζετε,
- στοχεύστε στην επαναχρησιμοποίηση του κώδικά σας, μην επαναλαμβάνετε ξανά και ξανά τον ίδιο κώδικα,
- εισάγετε σχόλια στον κώδικά σας, χωρίς όμως να τον υπερφορτώνετε,
- δώστε τον κώδικά σας να το ελέγξει και κάποιος άλλος.

Οι παραπάνω καλές πρακτικές ισχύουν για όλες τις γλώσσες προγραμματισμού και, φυσικά, και για το Matlab, το περιβάλλον του οποίου θα παρουσιαστεί στην ακόλουθη ενότητα.

1.3 Το περιβάλλον του MATLAB

Το Matlab είναι ένα λογισμικό για την ανάπτυξη κώδικα και προγραμμάτων, δηλαδή είναι μια γλώσσα προγραμματισμού, με κύριο προσανατολισμό την υλοποίηση αριθμητικών μεθόδων και υπολογισμών. Το όνομα Matlab προέρχεται από τις λέξεις MATrix LABoratotry, δηλαδή εργαστήριο πινάκων, μιας και η αρχική έκδοσή του, το 1967, από τον Μαθηματικό και Προγραμματιστή Cleve Moler, αφορούσε την επίλυση προβλημάτων Γραμμικής Άλγεβρας. Η πορεία του Matlab από εκείνη την πρώτη, πρώιμη έκδοση, είναι μακρά και συνεχής. Σήμερα το Matlab πρακτικά αποτελεί μια ισχυρή, ολοκληρωμένη γλώσσα προγραμματισμού με δυνατότητα ανάπτυξης εφαρμογών σε πλήθος επιστημονικών περιοχών, όπως οι Φυσικές Επιστήμες, η Μηχανική και η Ιατρική. Το Matlab περιλαμβάνει μεταξύ άλλων:

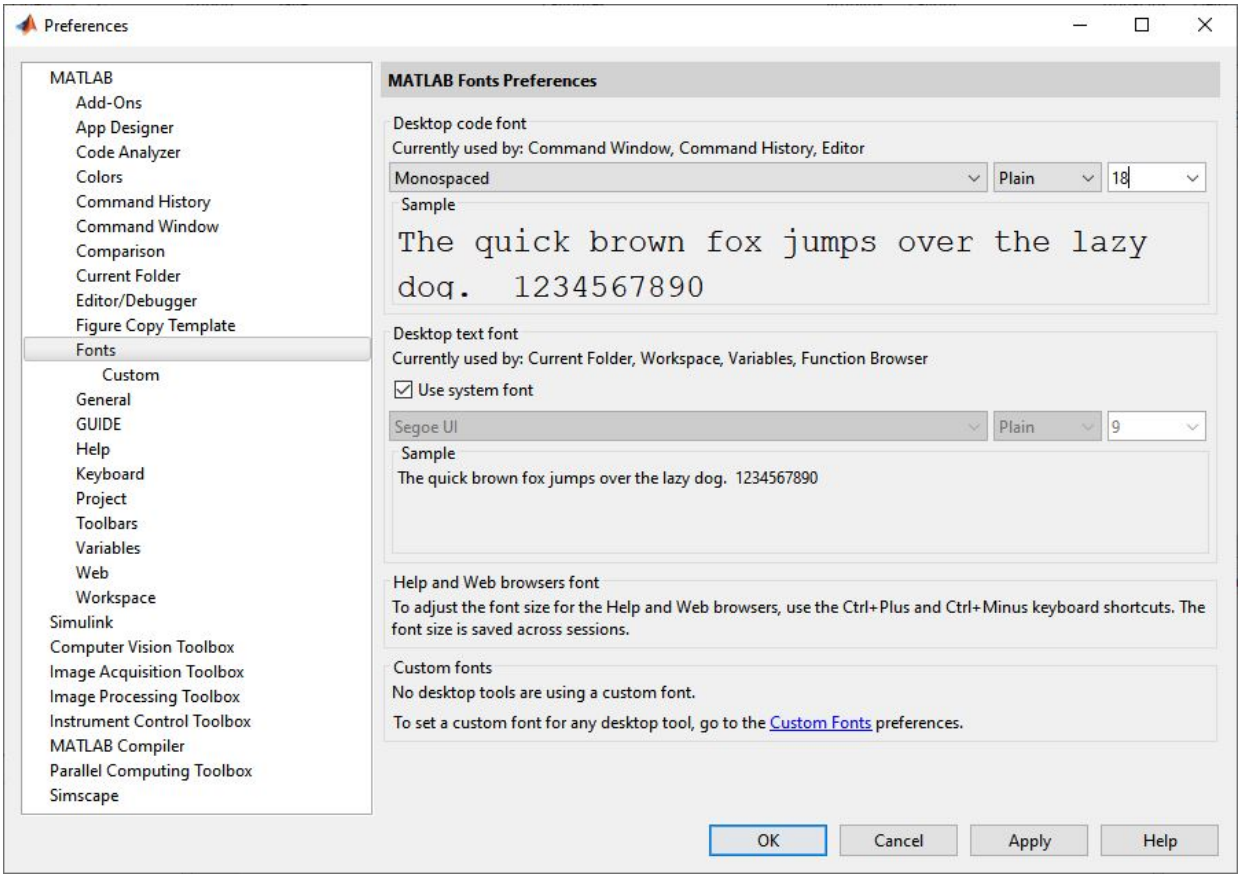
- πλήθος αριθμητικών μεθόδων για την επίλυση εξισώσεων και συστημάτων,
- μεθόδους επεξεργασίας και ανάλυσης μεγάλου όγκου δεδομένων,
- εντολές οπτικοποίησης των δεδομένων και των αποτελεσμάτων μέσω δημιουργίας γραφικών παραστάσεων,
- μεθόδους τέλεσης συμβολικών πράξεων,
- πλήθος έτοιμων, πλήρως παραμετροποιημένων, εφαρμογών για την επίλυση προβλημάτων,
- πλήθος εργαλείων και εντολών που επιτρέπουν τη δημιουργία προγραμμάτων από τον ίδιο τον χρήστη.



Εικόνα 1.1: Περιβάλλον εργασίας του Matlab.

Το Matlab μπορεί να τρέξει σε διάφορα λειτουργικά περιβάλλοντα, όπως στα Microsoft Windows, macOS και σε διάφορες εκδοχές του Linux, όπως το Ubuntu. Σε όλα τα λειτουργικά συστήματα το Matlab εκκινεί όπως και όλα τα υπόλοιπα εγκατεστημένα προγράμματα. Εκκινώντας το Matlab εμφανίζεται το περιβάλλον εργασίας, όπως δείχνεται στην Εικόνα 1.1. Σημειώνεται ότι στο Εικόνα 1.1 εμφανίζεται το περιβάλλον της έκδοσης R2020b του Matlab. Οι προηγούμενες εκδόσεις του Matlab δεν διαφέρουν σημαντικά από την έκδοση που χρησιμοποιείται στο παρόν σύγγραμμα. Επίσης, θα πρέπει να αναφερθεί ότι στόχος του συγγράμματος δεν είναι να παρουσιάσει μία συγκεκριμένη έκδοση του Matlab, αλλά να εξοικειώσει τον/την αναγνώστη/στρια με τη φιλοσοφία και τη λογική του προγραμματισμού, έτσι ώστε αυτός/ή να μπορεί να προσαρμοστεί σε οποιαδήποτε γλώσσα προγραμματισμού. Το Matlab αποτελεί μόνο το όχημα για αυτόν τον σκοπό.

Στο περιβάλλον εργασίας του Matlab κυριαρχεί το Command Window στο κέντρο του παραθύρου. Στο Command Window πληκτρολογούνται οι εντολές (δεξιά από το >>), οι οποίες εκτελούνται πατώντας `enter`. Στο αριστερό μέρος εμφανίζονται τα αρχεία (προγράμματα, σύνολα δεδομένων κ.ά.) που είναι διαθέσιμα στον τρέχοντα φάκελο, ενώ, αν επιλέξουμε κάποιο από αυτά, τότε εμφανίζονται πιθανές διαθέσιμες πληροφορίες για το αρχείο αυτό στο κάτω αριστερό παράθυρο. Στο δεξί μέρος του περιβάλλοντος εργασίας εμφανίζονται το ιστορικό των εντολών που έχουμε πληκτρολογήσει (κάτω δεξιά) και οι μεταβλητές (βλ. επόμενη ενότητα) που έχουμε ορίσει και έχει αποθηκεύσει στη μνήμη του το Matlab. Τέλος, στην κορυφή του περιβάλλοντος εργασίας εμφανίζεται το menu και τα διάφορα tabs του Matlab. Στις ακόλουθες ενότητες και στα επόμενα κεφάλαια του παρόντος συγγράμματος θα αναφερθούμε σε κάποια από τα παραπάνω στοιχεία του Matlab αναφέροντας τις λειτουργίες και τα χαρακτηριστικά τους. Επί του σημείου αυτού, θα αναφερθούμε στην επιλογή Preferences από το tab “HOME”, μέσω της οποίας μπορούμε να αλλάξουμε πολλές από τις παραμέτρους του Matlab, όπως παραδείγματος χάριν τη γραμματοσειρά (μέγεθος κ.ά.) των εντολών στο Command Window (βλ. Εικόνα 1.2).



Εικόνα 1.2: Αλλαγή γραμματοσειράς στο Matlab.

1.4 Το MATLAB ως υπολογιστής χειρός

Το Matlab εκτελεί τις αριθμητικές πράξεις όπως ακριβώς εκτελούνται και στα μαθηματικά, χρησιμοποιώντας δηλαδή τη σειρά προτεραιότητας των πράξεων. Οι βασικοί τελεστές αριθμητικών πράξεων είναι:

- το σύμβολο " + ", με το οποίο εκτελούμε την πρόσθεση δύο αριθμών. Παραδείγματος χάριν, αν πληκτρολογήσουμε

```
>> x=2+3
```

στο Command Window, το Matlab επιστρέφει το `x=5`.

- το σύμβολο " - ", με το οποίο εκτελούμε την αφαίρεση δύο αριθμών. Παραδείγματος χάριν, αν πληκτρολογήσουμε

```
>> x=2-3
```

το Matlab επιστρέφει το `x=-1`.

- το σύμβολο " * ", με το οποίο εκτελούμε τον πολλαπλασιασμό δύο αριθμών. Παραδείγματος χάριν, αν πληκτρολογήσουμε

```
>> x=2*3
```

το Matlab επιστρέφει το `x=6`.

- το σύμβολο " / ", με το οποίο εκτελούμε τη διαίρεση δύο αριθμών. Παραδείγματος χάριν, αν πληκτρολογήσουμε

```
>> x=2/3
```

το Matlab επιστρέφει το `x=0.6667`.

- το σύμβολο " ^ ", με το οποίο εκτελούμε την ύψωση ενός αριθμού σε κάποιον άλλον. Παραδείγματος χάριν, αν πληκτρολογήσουμε

```
>> x=2^3
```

το Matlab επιστρέφει το `x=8`.

Το Matlab, όπως ειπώθηκε, διατηρεί τη σειρά προτεραιότητας των πράξεων και, επομένως, η χρήση παρενθέσεων είναι απαραίτητη μόνο αν θέλουμε να προηγηθεί μια πρόσθεση (ή αφαίρεση) από έναν πολλαπλασιασμό (ή διαίρεση) και μια δύναμη. Παραδείγματος χάριν, πληκτρολογώντας

```
>> x=-2+2*3
```

στο Command Window, το Matlab επιστρέφει σωστά το `x=4`, αφού εκτελεί πρώτα τον πολλαπλασιασμό και, στη συνέχεια, την αφαίρεση. Αν επιθυμούμε να εκτελεστεί πρώτα η αφαίρεση και, στη συνέχεια, το αποτέλεσμα να πολλαπλασιαστεί με το δύο, τότε πρέπει να πληκτρολογήσουμε

```
>> x=(-2+3)*2
```

Η εντολή αυτή επιστρέφει την τιμή 2 για το `x`.

Αξίζει να σημειωθεί ότι το Matlab μπορεί να χειριστεί και πράξεις με μιγαδικούς αριθμούς, όπως αναλύεται στο Κεφάλαιο 7, αλλά και με το άπειρο. Το άπειρο στο Matlab συμβολίζεται με `inf` ή `Inf`. Παραδείγματα χρήσης του απείρου είναι οι ακόλουθες εντολές

```
>> 2+Inf
>> Inf-Inf
```

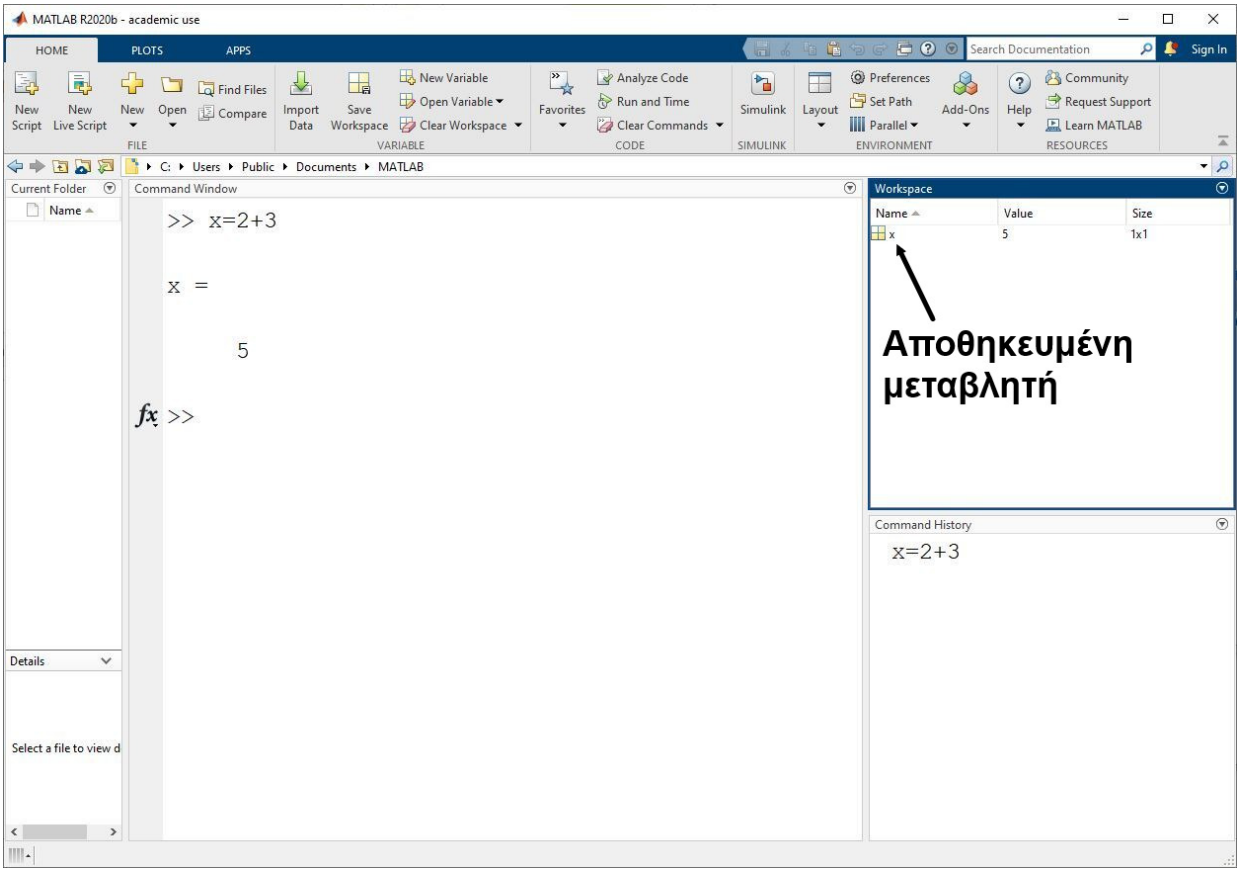
οι οποίες επιστρέφουν `Inf` και `NaN`, αντίστοιχα. Η δεύτερη εντολή επιστρέφει το `NaN`, δηλαδή το `Not a Number`, αφού αναπαριστά μια απροσδιόριστη μορφή.

1.5 Μεταβλητές

Στο Matlab, όπως και σε όλες τις γλώσσες προγραμματισμού, η πραγματοποίηση αριθμητικών πράξεων μπορεί να γίνει με τη βοήθεια αριθμητικών μεταβλητών. Στην προηγούμενη ενότητα παρουσιάστηκαν οι βασικοί τελεστές των αριθμητικών πράξεων μέσω παραδειγμάτων στα οποία το αποτέλεσμα αποθηκευόταν σε μια μεταβλητή με το όνομα `x`. Πιο συγκεκριμένα, καθεμία από αυτές τις εντολές υλοποιεί δύο λειτουργίες

- δημιουργεί με το " = " μια μεταβλητή, στην προκειμένη περίπτωση με το όνομα `x`, και
- αναθέτει σε αυτήν το αποτέλεσμα της πράξης που βρίσκεται δεξιά του " = ".

Η δημιουργία μιας μεταβλητής αποτυπώνεται στο Matlab με την εμφάνιση των στοιχείων της στο παράθυρο του Workspace, όπως φαίνεται στο Εικόνα 1.3. Κάθε μεταβλητή που εμφανίζεται στο Workspace μπορεί να χρησιμοποιηθεί



Εικόνα 1.3: Δημιουργία και ανάθεση τιμής σε μεταβλητή.

- για τη δημιουργία μιας νέας μεταβλητής, όπως στο παρακάτω παράδειγμα

```
>> x=5
>> y=x+3
```

στο όποιο, αρχικά, δημιουργούμε τη μεταβλητή `x` στην οποία αναθέτουμε την τιμή 5, την οποία χρησιμοποιούμε για να δημιουργήσουμε τη μεταβλητή `y`, στην οποία αναθέτουμε την τιμή 5+3, δηλαδή το 8.

- για την ανάθεση μιας νέας τιμής σε μια υπάρχουσα μεταβλητή, όπως στο παρακάτω παράδειγμα

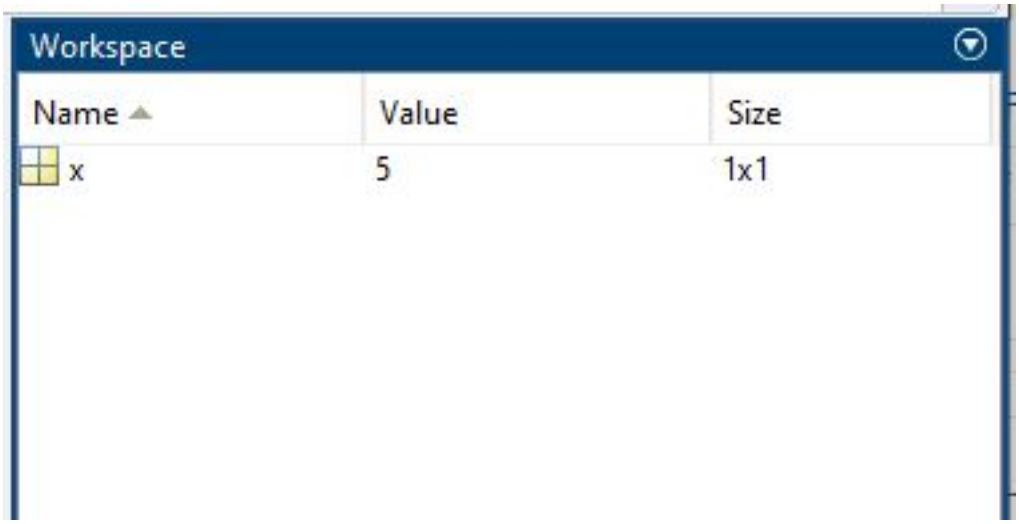
```
>> x=5
>> x=x^2
```

Στην πρώτη εντολή δημιουργούμε ξανά τη μεταβλητή `x` στην οποία αναθέτουμε την τιμή 5 και, στη συνέχεια, αναθέτουμε την τιμή 5^2, δηλαδή το 25, στη `x`. Επομένως, μετά την εκτέλεση των παραπάνω εντολών η τελική τιμή του `x` θα ισούται με 25.

Σημειώνεται ότι το όνομα μιας μεταβλητής μπορεί να αποτελείται από έναν ή περισσότερους χαρακτήρες. Σε κάθε περίπτωση, όμως, θα πρέπει να ικανοποιεί τους παρακάτω κανόνες:

- ο πρώτος χαρακτήρας πρέπει να είναι γράμμα,
- να αποτελείται από λατινικούς χαρακτήρες, αριθμούς ή/και κάτω παύλες (“_”).

Στα ονόματα των μεταβλητών γίνεται διάκριση πεζών-κεφαλαίων και, επομένως, οι μεταβλητές `x1` και `X1` θεωρούνται διαφορετικές στο Matlab.



Εικόνα 1.4: Στοιχεία της μεταβλητής στο Workspace του Matlab.

Τέλος, συμβουλεύουμε να μην χρησιμοποιούνται ως ονόματα για μεταβλητές λέξεις και σύμβολα που χρησιμοποιούνται ήδη από το Matlab για την αναπαράσταση γνωστών μεταβλητών ή συναρτήσεων (π.χ. `inf, log, …`).

1.6 Διανύσματα - Πίνακες

Το Matlab αναπαριστά όλες τις αριθμητικές μεταβλητές ως πίνακες. Παρατηρώντας στην Εικόνα 1.3 και στη μεγέθυνση του Workspace στην Εικόνα 1.4 τα στοιχεία της μεταβλητής που έχουμε δημιουργήσει, μπορούμε να διαπιστώσουμε ότι οι διαστάσεις (size) της μεταβλητής x είναι 1×1.

Αν και οι Πίνακες θα παρουσιαστούν αναλυτικά στο Κεφάλαιο 8, στο σημείο αυτό θα αναφερθούμε σε κάποιες βασικές έννοιες για τους Πίνακες. Οι Πίνακες, στην ουσία, αποτελούν ορθογώνιες διατάξεις αριθμών, συμβόλων ή ακόμα και εκφράσεων, διατεταγμένα σε σειρές και στήλες. Παραδείγματος χάριν, ο

$$\begin{bmatrix} 1 & 2 & 3 \\ -1 & 2 & 0 \end{bmatrix}$$

είναι ένας πίνακας που αποτελείται από 6 στοιχεία διατεταγμένα σε 2 σειρές και 3 στήλες. Ένας τέτοιος πίνακας λέμε ότι είναι ένας 2×3 πίνακας. Ένας πίνακας που αποτελείται από μία μόνο γραμμή ή μόνο μία στήλη λέμε ότι έχει διαστάσεις $1 \times n$ ή $n \times 1$ και αναφέρονται ως διάνυσμα γραμμή ή, απλώς, διάνυσμα και διάνυσμα στήλη, αντίστοιχα. Ένας αριθμός μπορεί να εκληφθεί ως ένας πίνακας με ένα μόνο στοιχείο, δηλαδή με μία μόνο γραμμή και μία μόνο στήλη. Για τον λόγο αυτό, το Matlab δηλώνει στο Workspace τις διαστάσεις της αριθμητικής μεταβλητής x ως 1×1 (βλ. Εικόνα 1.4).

Στο Matlab τα διανύσματα εισάγονται με τον ακόλουθο τρόπο:

```
>> x=[3, 4, 6, -1]
```

η οποία ορίζει το διάνυσμα $[3,4,6,-1]$ μίας γραμμής και τεσσάρων στηλών. Τα διανύσματα στήλες εισάγονται ως ακολούθως:

```
>> x=[3; 0; 21]
```

η οποία αναθέτει στη μεταβλητή x το 3×1 διάνυσμα στήλη $\begin{bmatrix} 3 \\ 0 \\ 21 \end{bmatrix}$.

Διανύσματα που έχουν συγκεκριμένη δομή, όπως παραδείγματος χάριν το $[1,3,5,7,9,11]$, μπορούν να εισαχθούν πληκτρολογώντας εντολές της μορφής

$$start : step : finish$$

Έτσι, το προαναφερθέν διάνυσμα μπορεί να εισαχθεί στο Matlab και ως ακολούθως:

```
>> x=1:2:11
x =
     1     3     5     7     9    11
```

ή και ως:

```
>> x=1:2:12
x =
     1     3     5     7     9    11
```

αφού το επόμενο στοιχείο, μετά το 11 που απέχει 2 μονάδες από αυτό, είναι το 13, το οποίο είναι μεγαλύτερο από το 12 και για αυτό δεν συμπεριλαμβάνεται στο διάνυσμα.

Παρατήρηση 1.1

Σημειώνεται ότι μπορούμε να παραλείψουμε το *step* στην εντολή της μορφής $start : step : finish$. Σε αυτήν την περίπτωση, το Matlab θέτει μόνο του το *step* ίσο με ένα. Επομένως, η εντολή

```
>> x=0:3
```

επιστρέφει το διάνυσμα $[0,1,2,3]$.

Οι πίνακες στο Matlab εισάγονται με τα στοιχεία ανά γραμμή, όπως φαίνεται παρακάτω:

```
>> x=[1, 2, 3 ; -1, 2, 0]
```

Η παραπάνω εντολή αναθέτει στη μεταβλητή x τον 2×3 πίνακα

$$\begin{bmatrix} 1 & 2 & 3 \\ -1 & 2 & 0 \end{bmatrix}$$

Παρατήρηση 1.2

Οι πίνακες μπορούν να προστεθούν (αφαιρεθούν), αρκεί να έχουν τις ίδιες διαστάσεις. Σε μια τέτοια περίπτωση, το άθροισμα (αφαίρεση) 2 πινάκων ορίζει έναν πίνακα ίδιας διάστασης με τους αρχικούς, του οποίου τα στοιχεία προκύπτουν από την προσθήκη (αφαίρεση) των στοιχείων στις αντίστοιχες θέσεις. Παραδείγματος χάριν, έχουμε ότι

$$\begin{bmatrix} 1 & 2 & 3 \\ -1 & 2 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 2 \\ -2 & 3 & -1 \end{bmatrix}.$$

Ο πολλαπλασιασμός πινακών είναι μια πιο περίπλοκη πράξη και θα παρουσιαστεί αναλυτικά στο Κεφάλαιο 8, ενώ ένα εγχειρίδιο με τα βασικά στοιχεία των Πινάκων μπορεί να βρεθεί και στο Παράρτημα Α΄. Στο σημείο αυτό, θα αναφερθούμε μόνο στον πολλαπλασιασμό στοιχείο προς στοιχείο των πινάκων,

το αποκαλούμενο γινόμενο Hadamard (Hadamard product). Το γινόμενο Hadamard είναι διαφορετικό από αυτό που αποκαλείται πολλαπλασιασμός πινάκων και ορίζεται για πίνακες ίδιας διάστασης. Το γινόμενο Hadamard επιστρέφει έναν πίνακα διάστασης ίδιας με τη διάσταση των αρχικών, του οποίου εν λόγω πίνακα τα στοιχεία προκύπτουν από τον πολλαπλασιασμό των στοιχείων στις αντίστοιχες θέσεις. Στο Matlab ο πολλαπλασιασμός πινάκων στοιχείο προς στοιχείο γίνεται με την εντολή .*. Παράδειγμα εφαρμογής του πολλαπλασιασμού πινάκων στοιχείο προς στοιχείο στο Matlab εμφανίζεται στη συνέχεια

```
>> [1, 2 , 3 ; -1, 2, 0].*[1, 2 , 3 ; -1, 1, -1]
ans=
     1     4     9
     1     2     0
```

Παρατήρηση 1.3

Αντίστοιχα με τον πολλαπλασιασμό πινάκων στοιχείο προς στοιχείο ορίζονται και

- η διαίρεση πινάκων στοιχείο προς στοιχείο, η οποία εκτελείται με την εντολή ./ μεταξύ δύο πινάκων ίδιας διάστασης,
- η δύναμη πινάκων στοιχείο προς στοιχείο, η οποία εκτελείται με την εντολή .^ μεταξύ δύο πινάκων ίδιας διάστασης ή χρησιμοποιώντας απλώς έναν πίνακα περνώντας τη δύναμη σε καθένα στοιχείο του πίνακα ξεχωριστά, όπως φαίνεται από τα ακόλουθα δύο παραδείγματα:

```
>> [1, 2 , 3 ; -1, 2, 0].^[1, 2 , 3 ; -1, 1, -1]
ans =
     1     4    27
    -1     2   Inf
>> [1,2,3;4,5,6].^2
ans =
     1     4     9
    16    25    36
```

1.6.1 Οι εντολές size και length

Η διάσταση (size) ενός πίνακα, έστω A, μπορεί να ανακτηθεί στο Matlab με την εντολή size(A). Η εντολή size(A) επιστρέφει ένα 1×2 διάνυσμα, του οποίου το πρώτο στοιχείο αντιπροσωπεύει το πλήθος των γραμμών του πίνακα και το δεύτερο το πλήθος των στηλών. Στη συνέχεια, εμφανίζονται παραδείγματα-εφαρμογές της εντολής *size*.

```
>> A=[1, 2 , 3 ; -1, 2, 0]
A =
     1     2     3
    -1     2     0
>> size(A)
ans =
     2     3
>> B=[1, 2 ; 3  4 ; 2, 0 ; 0 1]
```

```
B =
     1     2
     3     4
     2     0
     0     1
>> size(B)
ans =
     4     2
```

Η εντολή `length`, από την άλλη, επιστρέφει τη μέγιστη διάσταση ενός πίνακα. Επομένως, για τους πίνακες A και B των προηγούμενων εντολών, οι εντολές `length(A)` και `length(B)` επιστρέφουν την τιμή 3 για τον πίνακα A (δηλαδή το πλήθος των στηλών του A που είναι μεγαλύτερο από το πλήθος των γραμμών του) και την τιμή 4 για τον πίνακα B (δηλαδή το πλήθος των γραμμών του B που είναι μεγαλύτερο από το πλήθος των στηλών του).

Ενδιαφέρον παρουσιάζει η εφαρμογή της εντολής `length` σε ένα διάνυσμα, αφού σε αυτή την περίπτωση επιστρέφει το πλήθος των στοιχείων του. Παραδείγματος χάριν, το πλήθος των στοιχείων του διανύσματος

```
>> x=0:0.01:10^3;
```

μπορεί να υπολογιστεί άμεσα με την εφαρμογή της εντολής *length* ως ακολούθως:

```
>> length(x)
ans =
      100001
```

1.7 Εντολές μορφοποίησης

Το Matlab διαθέτει εντολές που μορφοποιούν τον τρόπο που τυπώνονται τα αποτελέσματα ή μεταβάλλουν την αναπαράσταση των αριθμών στο Command Window. Για να γίνει αυτό πιο εύκολα κατανοητό, θα ορίσουμε τη μεταβλητή `y`, στην οποία θα αναθέσουμε τον άρρητο αριθμό $10\sqrt{2}$, και, στη συνέχεια, θα εφαρμόσουμε με τη σειρά τις ακόλουθες εντολές:

1. `format compact`
2. `format long`
3. `format short`
4. `format longe`
5. `format shorte`
6. `format bank`
7. `format rat`
8. `format loose`
9. `format short`

για να αναπαραστήσουμε τη μεταβλητή `y`. Οι δύο τελευταίες εντολές αποκαθιστούν την προεπιλεγμένη επιλογή του Matlab για τον τρόπο παρουσίασης των αποτελεσμάτων και των αριθμών. Σημειώνεται ότι η εντολή που υπολογίζει την τετραγωνική ρίζα ενός αριθμού στο Matlab είναι η sqrt. Οι εντολές και τα αντίστοιχα αποτελέσματα εμφανίζονται στη συνέχεια:

```
>> y=10*sqrt(2)

y =

   14.1421

>> format compact
>> y
y =
   14.1421
>> format long
>> y
y =
  14.142135623730951
>> format short
>> y
y =
   14.1421
>> format longe
>> y
y =
     1.414213562373095e+01
>> format shorte
>> y
y =
   1.4142e+01
>> format bank
>> y
y =
         14.14
>> format rat
>> y
y =
    2786/197
>> format loose
>> y

y =

    2786/197

>> format short
>> y

y =

   14.1421
```

Από τα παραπάνω, διαπιστώνουμε ότι οι εντολές

- `format compact`
- `format loose`

διαμορφώνουν τον τρόπο που τυπώνονται τα αποτελέσματα, χρησιμοποιώντας μια συμπαγή ή χαλαρή (προσθέτοντας κενές γραμμές) μορφοποίηση στην εκτύπωση των αποτελεσμάτων, αντίστοιχα. Οι υπόλοιπες εντολές επηρεάζουν τον τρόπο αναπαράστασης των αριθμών στο Command Window, χωρίς όμως να αλλοιώνουν τις τιμές των αριθμών που χρησιμοποιεί και έχει αποθηκευμένες στη μνήμη του το Matlab.

Από τους παραπάνω τρόπους αναπαράστασης των αριθμών, αξίζει να αναφερθούμε:

1. στην επιστημονική μορφή των αριθμών, την οποία ζητάμε από το Matlab να υιοθετήσει υποχρεωτικώς για την αναπαράσταση όλων των αριθμών με τις εντολές `format longe` και `format shorte`
2. στην προσεγγιστική αναπαράσταση όλων των αριθμών με τη βοήθεια ακέραιων αριθμών, η οποία επιτυγχάνεται με την εντολή `format rat`.

Υιοθετώντας την επιστημονική γραφή των αριθμών, οι αριθμοί εκφράζονται αποτυπώνοντας μέρος μόνο των σημαντικών ψηφίων του αριθμού, ακολουθούμενο από το `e+n` ή `e-n`, το οποίο υποδηλώνει τον πολλαπλασιασμό του προηγούμενου αριθμού με το 10^n ή 10^{-n}, αντίστοιχα. Έτσι, επί παραδείγματι, ο αριθμός 43.232454 στην επιστημονική μορφή γράφεται ως 4.3232e+01, ενώ ο αριθμός 0.0020201 ως 2.0201e-03. Σημειώνεται ότι το Matlab θα τυπώσει έναν πολύ μεγάλο αριθμό ή έναν αριθμό κοντά στο μηδέν με την επιστημονική του μορφή, ακόμα και αν δεν έχουμε επιλέξει κάποια από τις εντολές `format longe` και `format shorte`.

Αναφορικά, τώρα, με την προσεγγιστική αναπαράσταση όλων των αριθμών με τη βοήθεια ακέραιων αριθμών, το Matlab με την εντολή `format rat` προσπαθεί να χρησιμοποιήσει όσο το δυνατόν μικρότερους ακέραιους για να αναπαραστήσει έναν αριθμό ως το πηλίκο τους, χωρίς όμως να αλλοιώσει σημαντικά την ακρίβεια του αποτελέσματος. Έτσι, στο παραπάνω παράδειγμα, ο αριθμός $10\sqrt{2}$ προσεγγίζεται από το πηλίκο 2786/197. Υπολογίζοντας τη διαφορά $2786/197 - 10\sqrt{2}$, το Matlab τυπώνει την τιμή -3.6440e-06, δηλαδή τον αριθμό -0.000003644, ο οποίος είναι ένας εξαιρετικά μικρός αριθμός.

Κλείνοντας την ενότητα αυτή, πρέπει να αναφερθούμε και στη χρήση του ελληνικού ερωτηματικού “ ; ” (semicolon) στο τέλος των εντολών στο Matlab. Πληκτρολογώντας το “ ; ” στο τέλος των εντολών, αυτές εκτελούνται κανονικά, αλλά εμποδίζεται η εκτύπωση των αποτελεσμάτων. Ένα παράδειγμα της χρήσης του “ ; ” φαίνεται στις επόμενες εντολές στις οποίες ανατίθεται τιμή στη μεταβλητή `x`, η οποία μάλιστα χρησιμοποιείται στη συνέχεια, χωρίς όμως να τυπώνεται το αποτέλεσμά της.

```
>> x=3;
>> y=x+2
y =
     5
```

1.8 Αρχεία MATLAB- script files

Τις περισσότερες φορές η επίλυση ενός προβλήματος με την ανάπτυξη ενός κώδικα απαιτεί περισσότερες από μία προσπάθειες - βελτιώσεις. Ο κώδικας συνήθως αποτελείται από (σχετικά) μεγάλο πλήθος εντολών

και είναι συχνά επιθυμητό να έχει τη δυνατότητα παραμετροποίησης, έτσι ώστε να μπορεί να επαναχρησιμοποιηθεί για την επίλυση παρόμοιων προβλημάτων. Προς την κατεύθυνση αυτή, χρήσιμο εργαλείο αποτελούν τα αρχεία script, ή αλλιώς, αρχεία εντολών script m-files. Τα αρχεία script περιέχουν μια ακολουθία εντολών που πληκτρολογούνται στον κειμενογράφο του Matlab και μπορούν με εύκολο και απλό τρόπο να εκτελεστούν από το Command Window.

Στην ενότητα αυτή, δίνεται μια αρχική περιγραφή των αρχείων script, τα οποία θα χρησιμοποιήσουμε σε μεγάλο βαθμό στα επόμενα κεφάλαια. Η δημιουργία ενός αρχείου script μπορεί να γίνει είτε πληκτρολογώντας `edit` και το όνομα του αρχείου με την επέκταση `.m` στο Command Window είτε κάνοντας κλικ στην επιλογή New Script από το menu του Matlab, όπως φαίνεται στην Εικόνα 1.5. Η επιλογή αυτή ανοίγει (σε ένα νέο παράθυρο) τον κειμενογράφο του Matlab, ο οποίος φαίνεται στην Εικόνα 1.6.

Στο παράθυρο που ανοίγει μπορούμε να πληκτρολογήσουμε όποιες εντολές θέλουμε. Παραδείγματος χάριν, το παρακάτω αρχείο script, το οποίο μπορούμε να σώσουμε, επί παραδείγματι, με το όνομα fahtocel, μετατρέπει τους 30° βαθμούς Fahrenheit σε βαθμούς Celsius.

```
1 F=30;
2 C=(F - 32)*5/9
```

Αν μετά την αποθήκευση του αρχείου πληκτρολογήσουμε στο Command Window το όνομα του αρχείου (χωρίς την επέκταση `.m`), τότε το Matlab θα εκτελέσει ακολουθιακά τις εντολές και θα τυπώσει την τιμή του C, η οποία στην προκειμένη περίπτωση ισούται με -1.1112.

Το παραπάνω αρχείο script είναι ένα πρώτο, απλό παράδειγμα αρχείου script, το οποίο όμως είναι αρκετά στατικό. Στη συνέχεια, παρουσιάζουμε πώς τα αρχεία script μπορούν να γίνουν πιο δυναμικά με χρήση:

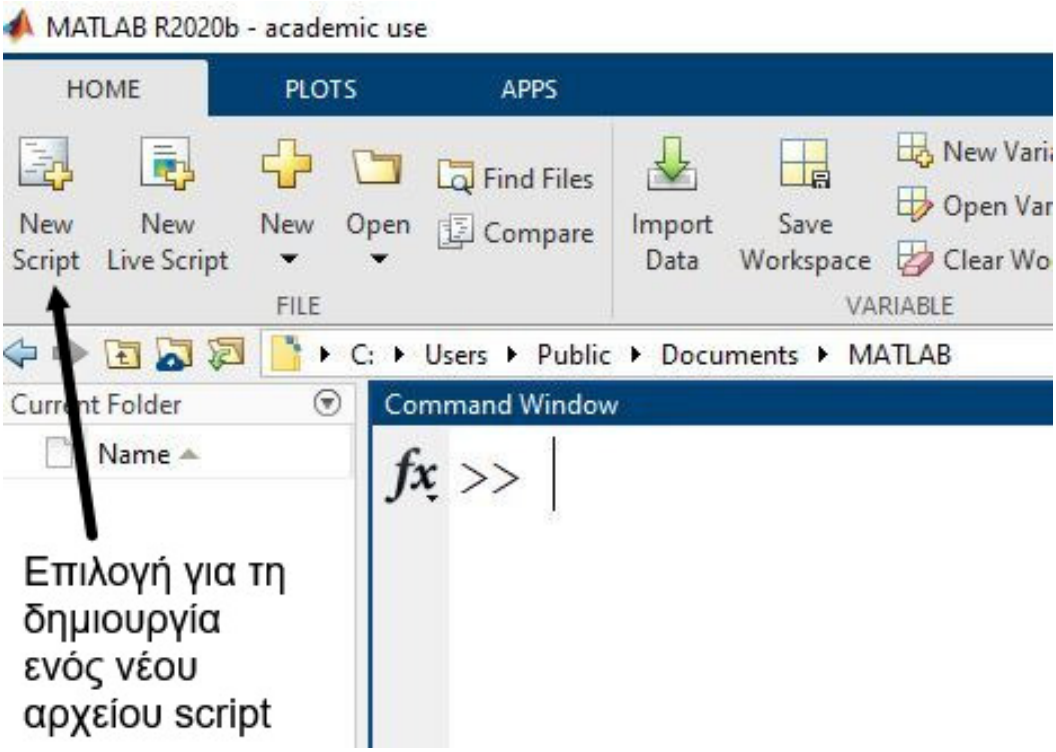
- Της δυνατότητας πληκτρολόγησης των τιμών εισόδου από τον χρήστη. Στο παραπάνω παράδειγμα, η τιμή εισόδου ήταν η τιμή του F, δηλαδή των βαθμών Fahrenheit, η οποία δεν είναι απαραίτητο να θέλουμε να είναι πάντα ίση με 30.
- Της επιστροφής συνοδευτικού κειμένου, που θα πληροφορεί τον χρήστη για το τι είναι η τιμή που υπολόγισε το πρόγραμμα.
- Της ενσωμάτωσης πληροφοριών και σχολίων στο αρχείο script, που θα διευκολύνουν τον χρήστη να καταλάβει ποιο πρόβλημα επιλύει ο συγκεκριμένος κώδικας.

Στη προσπάθειά μας, χρήσιμες είναι οι παρακάτω εντολές-λειτουργίες του Matlab:

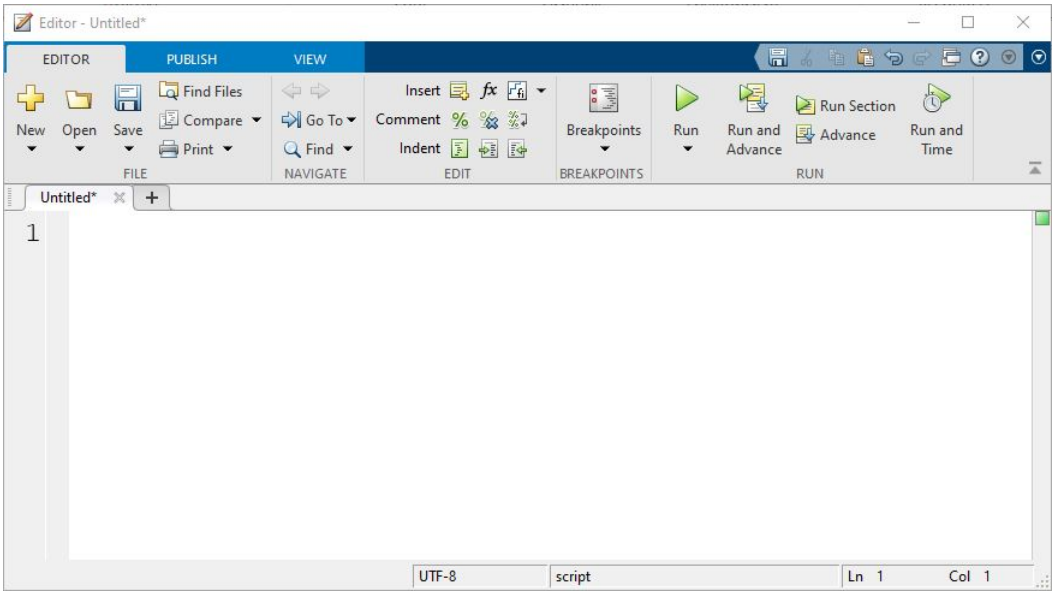
1. `input` για την ανάθεση τιμών σε μεταβλητές από τον χρήστη,
2. `disp` για την επιστροφή κειμένου πληροφοριών,
3. `%` για την εισαγωγή μη εκτελέσιμων σχολίων στον κώδικα.

Στη συνέχεια, εμφανίζεται ένα πιο δυναμικό και επεξηγηματικό αρχείο script για τη μετατροπή των βαθμών Fahrenheit σε βαθμούς Celsius:

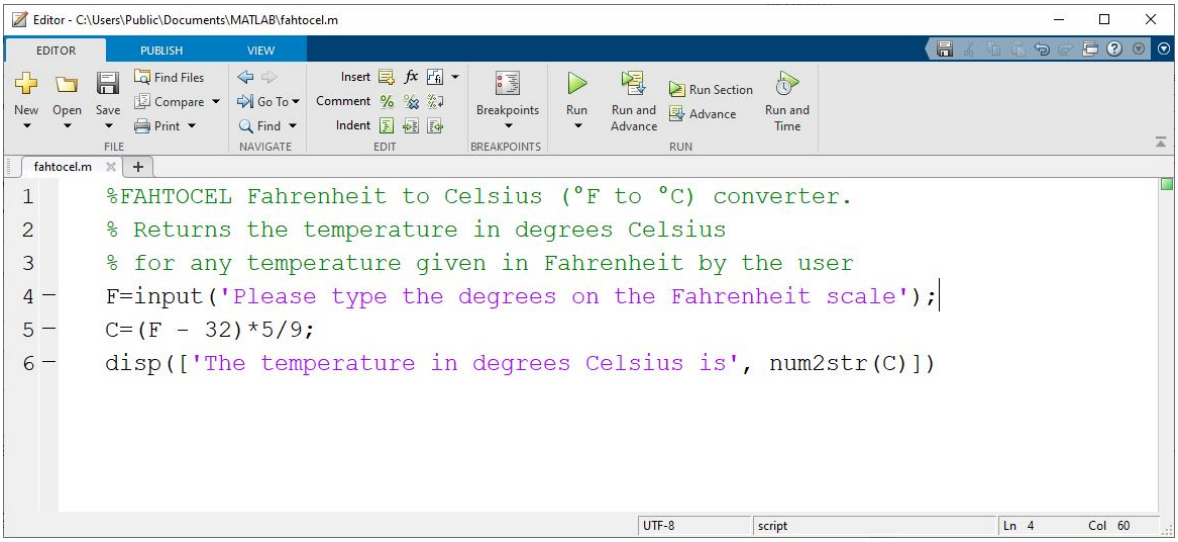
```
1 %FAHTOCEL Fahrenheit to Celsius (F to C) converter.
2 % Returns the temperature in degrees Celsius
3 % for any temperature given in Fahrenheit by the user
4 F = input('Please type the degrees on the Fahrenheit scale');
5 C = (F - 32)*5/9;
6 disp(['The temperature in degrees Celsius is', num2str(C)])
```



Εικόνα 1.5: Δημιουργία νέου αρχείου script από το menu του Matlab.



Εικόνα 1.6: Περιβάλλον δημιουργίας αρχείων script του Matlab.



Εικόνα 1.7: Αρχείο script για τη μετατροπή των βαθμών Fahrenheit σε βαθμούς Celsius.

το οποίο παρουσιάζεται, όπως φαίνεται στο Matlab, στην Εικόνα 1.7.

Οι τρεις πρώτες γραμμές, οι οποίες ξεκινούν με το σύμβολο " % ", περιλαμβάνουν βοηθητικές πληροφορίες για το πρόγραμμα. Η χρησιμότητα των γραμμών αυτών και η ειδική μορφή της πρώτης γραμμής (με τα κεφαλαία γράμματα του ονόματος του αρχείου) θα φανεί στην ενότητα 1.11.

Στην τέταρτη γραμμή εμφανίζεται η εντολή `input`. Η εντολή `input` εκτελεί τις παρακάτω λειτουργίες:

1. εμφανίζει στο Command Window το μήνυμα που είναι πληκτρολογημένο μέσα στις παρενθέσεις,
2. θέτει το Matlab σε αναμονή των πληροφοριών (π.χ. αριθμού) που θα πληκτρολογήσει ο χρήστης,
3. τις οποίες αναθέτει στη μεταβλητή, που εμφανίζεται αριστερά του " = ".

Έτσι, στην προκειμένη περίπτωση, όταν εκτελεστεί η εντολή στη γραμμή 4, θα εμφανιστεί το μήνυμα `Please type the degrees on the Fahrenheit scale` στο Command Window και το Matlab θα περιμένει ο χρήστης να πληκτρολογήσει τους βαθμούς Fahrenheit που επιθυμεί να μετατραπούν σε βαθμούς Celsius. Η τιμή που θα πληκτρολογήσει ο χρήστης θα ανατεθεί στη μεταβλητή `F`, αφού πατήσει το enter, και το πρόγραμμα θα συνεχίσει στις επόμενες γραμμές εντολών.

Στη γραμμή 5 του κώδικα δημιουργείται η μεταβλητή `C`, στην οποία ανατίθεται η τιμή $(F - 32) * 5/9$. Στην τελευταία γραμμή του κώδικα η εντολή `disp` εμφανίζει το μήνυμα που έχουμε πληκτρολογήσει και τη θερμοκρασία σε βαθμούς Celsius. Σημειώνεται ότι η εντολή `disp` μπορεί να χειριστεί είτε μόνο κείμενο είτε μόνο αριθμούς. Για τον λόγο αυτό, χρησιμοποιείται και η εντολή `num2str`, η οποία αναπαριστά τους αριθμούς ως χαρακτήρες και, ως εκ τούτου, μπορεί να τους εμφανίσει η εντολή `disp` μαζί με το συνοδευτικό κείμενο.

Κλείνοντας την παρουσίαση των αρχείων script, πρέπει να σημειωθεί ότι:

- για να είναι ένα αρχείο script προσβάσιμο, πρέπει να βρίσκεται αποθηκευμένο στον φάκελο εργασίας, δηλαδή τον τρέχοντα φάκελο,
- το όνομα του αρχείου οφείλει να ακολουθεί τους κανόνες που ισχύουν και για τα ονόματα των μεταβλητών, αλλά δεν πρέπει να περιλαμβάνει κεφαλαία γράμματα, αφού τα λειτουργικά συστήματα, όπως επί παραδείγματι τα Windows, δεν κάνουν διάκριση κεφαλαίων και μικρών στα ονόματα των αρχείων,

- οποιαδήποτε μεταβλητή δημιουργείται κατά την εκτέλεση του αρχείου ή είχε δημιουργηθεί πριν την εκτέλεσή του, είναι διαθέσιμη από αυτό και παραμένει ενεργή στη μνήμη του Matlab και μετά την ολοκλήρωση των εντολών του αρχείου.

Αναφορικά με το όνομα των αρχείων script (αλλά και των μεταβλητών), για να είμαστε σίγουροι ότι το όνομα που θέλουμε να χρησιμοποιήσουμε δεν χρησιμοποιείται από το Matlab, μπορούμε να πληκτρολογήσουμε την εντολή

```
>> which ....
```

στο Command Window, αντικαθιστώντας τις τελείες με το όνομα που επιθυμούμε. Αν το όνομα χρησιμοποιείται από κάποια εντολή του Matlab, η πληροφορία αυτή θα τυπωθεί στο Command Window, αλλιώς το Matlab θα μας ενημερώσει ότι δεν βρέθηκε κάποια εντολή με το όνομα αυτό.

Τέλος, αξίζει να σημειωθούν δύο δυνατότητες που έχουν τα αρχεία script, τις οποίες θα δούμε αναλυτικά στα επόμενα κεφάλαια. Τα αρχεία script μπορούν:

1. να καλέσουν άλλα αρχεία script, αρκεί αυτά να βρίσκονται στον ίδιο φάκελο με αυτά,
2. να καλέσουν τον ίδιο τον εαυτό τους.

Άσκηση αυτοαξιολόγησης 1.1

Δημιουργήστε ένα αρχείο script που θα υπολογίζει και θα επιστρέφει την περίμετρο και το εμβαδόν ενός κύκλου, την ακτίνα του οποίου θα εισάγει ο χρήστης. Σημειώνεται ότι το π στο Matlab συμβολίζεται με `pi`.

Άσκηση αυτοαξιολόγησης 1.2

Δημιουργήστε ένα αρχείο script που θα υπολογίζει και θα επιστρέφει την περίμετρο και το εμβαδόν ενός ορθογωνίου, του οποίου το μήκος των πλευρών θα εισάγει ο χρήστης.

1.9 Βασικές εντολές διαχείρισης

Ιδιαίτερα χρήσιμες είναι οι εντολές:

- clc,
- clear all,
- close all.

Η πρώτη εντολή (clc) καθαρίζει το Command Window από τις εντολές, που έχουμε πληκτρολογήσει νωρίτερα, σβήνοντας οτιδήποτε έχει τυπωθεί. Η δεύτερη εντολή (clear all) διαγράφει από τη μνήμη οποιαδήποτε μεταβλητή έχει οριστεί νωρίτερα, ενώ η τρίτη, close all, κλείνει τα παράθυρα των γραφικών παραστάσεων που έχουμε πιθανώς δημιουργήσει[1].

Παρατήρηση 1.4

Αν πληκτρολογήσουμε τις εντολές

[1] Με τη δημιουργία γραφικών παραστάσεων θα ασχοληθούμε στα Κεφάλαια 5 και 6.

```
>> close all
>> clear all
>> clc
```

στην ουσία είναι σαν να κάνουμε επανεκκίνηση του Matlab και να έχουμε ένα καινούργιο περιβάλλον εργασίας, το οποίο δεν εξαρτάται απ΄ ό,τι έχουμε κάνει νωρίτερα.

1.10 Βοήθεια στο Matlab

Το Matlab διαθέτει τέτοιο πλήθος εντολών, ώστε είναι αδύνατον κάποιος να τις γνωρίζει όλες. Επιπροσθέτως, αρκετά συχνά, ακόμα και για εντολές που γνωρίζουμε χρειάζεται να θυμηθούμε το συντακτικό τους ή το τι ακριβώς κάνουν. Για τους παραπάνω λόγους, το Matlab έχει ενσωματωμένες πληροφορίες για όλες τις εντολές. Οι πληροφορίες αυτές μπορούν να ανακτηθούν με την εντολή `help`, ακολουθούμενη από το όνομα της εντολής. Παραδείγματος χάριν, η εντολή

```
>> help disp
```

επιστρέφει τις βασικές πληροφορίες για την εντολή `disp`, το συντακτικό της αλλά και μια λίστα από εντολές με παρόμοια λειτουργία.

Με την εντολή `help` μπορούμε να δούμε και τις πληροφορίες που έχουμε γράψει στα αρχεία script που έχουμε δημιουργήσει. Παραδείγματος χάριν, πληκτρολογώντας

```
>> help fahtocel
```

στο Command Window το Matlab επιστρέφει τις τρεις πρώτες γραμμές, δηλαδή τις γραμμές στην αρχή του αρχείου που ξεκινάμε με “ % ”, με το όνομα του αρχείου fahtocel τυπωμένο με έντονα γράμματα.

Στην περίπτωση που δεν γνωρίζουμε το όνομα της εντολής που χρειαζόμαστε, μπορούμε να χρησιμοποιήσουμε την εντολή `lookfor`. Η εντολή `lookfor`, ακολουθούμενη από μια λέξη, επιστρέφει μια λίστα με όλες τις εντολές που περιέχουν τη συγκεκριμένη λέξη στην πρώτη γραμμή των πληροφοριών, που μπορούν να ανακτηθούν με την εντολή `help`. Παραδείγματος χάριν, πληκτρολογώντας

```
>> lookfor display
```

στο Command Window, το Matlab επιστρέφει όλες τις εντολές με τη λέξη display στην πρώτη γραμμή των πληροφοριών τους. Φυσικά, μέσα σε αυτήν τη μακροσκελή λίστα βρίσκεται και η εντολή `disp`.

1.11 Σφάλματα στο MATLAB

Αν κατά την πληκτρολόγηση εντολών ή την εκτέλεση αρχείων script το Matlab εντοπίσει ένα συντακτικό σφάλμα στον κώδικα, τότε το πρόγραμμα διακόπτεται και το Matlab επιστρέφει ένα μήνυμα σφάλματος, υποδεικνύοντας το πιθανό σημείο του κώδικα στο οποίο εντοπίζεται το σφάλμα. Επιπροσθέτως, επιστρέφει και πληροφορίες για το πιθανό λάθος. Τις περισσότερες φορές οι πληροφορίες αυτές είναι αρκετά ακριβείς και η προσεκτική ανάγνωσή τους μπορεί να μας οδηγήσει στον γρήγορο εντοπισμό του σφάλματος.

1.12 Ασκήσεις

Άσκηση 1.1. *Δώστε την κατάλληλη απάντηση (ΣΩΣΤΟ ή ΛΑΘΟΣ) στις κάτωθι προτάσεις. Αιτιολογήστε σύντομα τις απαντήσεις σας.*

1. *Οι μεταβλητές που δημιουργούνται κατά την εκτέλεση ενός αρχείου script μπορούν να χρησιμοποιηθούν μόνο από το συγκεκριμένο αρχείο.*
2. *Ο αριθμός 3.1111e-02 ισούται με τον αριθμό 0.0031111.*
3. *Η εντολή* `format long` *αναγκάζει το* Matlab *να τυπώνει τον αριθμό* π *ως 3.14.*
4. *Το σύμβολο " ; " έχει διπλή χρήση στο* Matlab*. Εμποδίζεται η εκτύπωση των αποτελεσμάτων αλλά και αλλάζει τη γραμμή κατά την εισαγωγή των στοιχείων ενός πίνακα.*

Άσκηση 1.2. *Δημιουργήστε ένα αρχείο script που θα υπολογίζει και θα επιστρέφει το άθροισμα και τη διαφορά δύο αριθμών που θα εισάγει ο χρήστης.*

Άσκηση 1.3. *Δημιουργήστε ένα αρχείο script που θα υπολογίζει και θα επιστρέφει το γινόμενο και το πηλίκο δύο αριθμών που θα εισάγει ο χρήστης.*

Άσκηση 1.4. *Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη την εισαγωγή ενός πίνακα και θα επιστρέφει το πλήθος των γραμμών και των στηλών του πίνακα που πληκτρολόγησε ο χρήστης.*

Άσκηση 1.5. *Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη την εισαγωγή ενός πίνακα και θα επιστρέφει τη μέγιστη διάσταση του πίνακα που πληκτρολόγησε ο χρήστης.*

Άσκηση 1.6. *Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη την εισαγωγή ενός πίνακα και θα επιστρέφει έναν πίνακα με στοιχεία το τετράγωνο των στοιχείων του πίνακα που πληκτρολόγησε ο χρήστης.*

Άσκηση 1.7. *Δημιουργήστε ένα αρχείο script που θα υπολογίζει και θα επιστρέφει το εμβαδόν και τον όγκο ενός κύβου, του οποίου την πλευρά θα εισάγει ο χρήστης.*

Άσκηση 1.8. *Δημιουργήστε ένα αρχείο script που θα υπολογίζει και θα επιστρέφει το εμβαδόν και τον όγκο μιας σφαίρας, της οποίας την ακτίνα θα εισάγει ο χρήστης.*

1.13 Απαντήσεις στις ασκήσεις αυτοαξιολόγησης

Λύση άσκησης αυτοαξιολόγησης 1.1

Για να δημιουργήσουμε ένα αρχείο script που υπολογίζει και επιστρέφει την περίμετρο και το εμβαδόν ενός κύκλου, την ακτίνα του οποίου θα εισάγει ο χρήστης, χρειάζεται να πληκτρολογήσουμε τις ακόλουθες εντολές και να τις αποθηκεύσουμε σε ένα αρχείο `m`.

```
1 %CIRCLEPA Calculates the Perimeter and the Area of a circle
2 % for a radius given by the user
3 r=input('Please type the radius of the circle');
4 P=2*pi*r;
5 A=pi*r^2;
6 disp(['The Perimeter of the circle is', num2str(P)])
7 disp(['and the Area of the circle is', num2str(A)])
```

Λύση άσκησης αυτοαξιολόγησης 1.2

Για να δημιουργήσουμε ένα αρχείο script που υπολογίζει και επιστρέφει την περίμετρο και το εμβαδόν ενός ορθογωνίου, του οποίου το μήκος των πλευρών θα εισάγει ο χρήστης, χρειάζεται να πληκτρολογήσουμε τις ακόλουθες εντολές και να τις αποθηκεύσουμε σε ένα αρχείο `m`.

```
1 %RECTANGLEPA Calculates the Perimeter and the Area of a rectangle
2 % for a length and a width given by the user.
3 L=input('Please type the length of the rectangle');
4 W=input('Please type the width of the rectangle');
5 P=2*L+2*W;
6 A=L*W;
7 disp(['The Perimeter of the rectangle is', num2str(P)])
8 disp(['and the Area of the rectangle is', num2str(A)])
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ξενόγλωσση

Vee, A. (2017). *Coding Literacy: How Computer Programming is Changing Writing*. The MIT Press.

ΚΕΦΑΛΑΙΟ 2

ΒΑΣΙΚΕΣ ΜΑΘΗΜΑΤΙΚΕΣ ΚΑΙ ΛΟΓΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

Σύνοψη:

Σε αυτό το κεφάλαιο παρουσιάζονται οι βασικές μαθηματικές και λογικές συναρτήσεις, οι οποίες αποτελούν τα βασικά εργαλεία για την ανάπτυξη προγραμμάτων. Πιο συγκεκριμένα, αρχικά παρουσιάζεται ο τρόπος κλήσης των τριγωνομετρικών, εκθετικών και λογαριθμικών συναρτήσεων. Στη συνέχεια, γίνεται αναφορά στον τρόπο υπολογισμού της τετραγωνικής ρίζας αλλά και στον υπολογισμό γενικά της n ρίζας ενός αριθμού. Η παρουσίαση των βασικών συναρτήσεων ολοκληρώνεται με την παρουσίαση των συναρτήσεων στρογγυλοποίησης του Matlab και άλλων βοηθητικών συναρτήσεων, όπως το άθροισμα και το γινόμενο. Έπειτα η προσοχή στρέφεται στις λογικές συναρτήσεις και τους λογικούς τελεστές, ενώ το κεφάλαιο ολοκληρώνεται με μια αναφορά στην προτεραιότητα των πράξεων.

Προαπαιτούμενη γνώση: Εξοικείωση με το περιβάλλον του Matlab και βασικές γνώσεις μαθηματικών.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του κεφαλαίου αυτού, θα γνωρίζετε:

- τις βασικές μαθηματικές συναρτήσεις και τον τρόπο σύνταξής τους στο Matlab,
- τις βασικές λογικές πράξεις,
- τις λογικές συναρτήσεις και τους λογικούς τελεστές αλλά και τον τρόπο σύνταξή τους στο Matlab,
- τη σειρά προτεραιότητας εκτέλεσης των μαθηματικών και λογικών πράξεων.

Γλωσσάριο επιστημονικών όρων

- Ακέραιο μέρος
- Άρνηση
- Διάζευξη
- Εκθετικές συναρτήσεις
- Λογαριθμικές συναρτήσεις
- Λογικά διανύσματα
- Λογική πρόταση
- Σύζευξη
- Συναρτήσεις στρογγυλοποίησης
- Τριγωνομετρικές συναρτήσεις

2.1 Εισαγωγή

Το Matlab, όπως είδαμε στο προηγούμενο κεφάλαιο, μπορεί να χρησιμοποιηθεί ως υπολογιστής χειρός για την τέλεση μαθηματικών πράξεων. Πέρα από τις βασικές αυτές λειτουργίες, το Matlab ενσωματώνει και μια σειρά εντολών για τον υπολογισμό βασικών συναρτήσεων, όπως, παραδείγματος χάριν, οι τριγωνομετρικές συναρτήσεις. Στις επόμενες ενότητες παρουσιάζονται αρχικά κάποιες βασικές μαθηματικές συναρτήσεις και κάποιες σταθερές που είναι ενσωματωμένες στο Matlab. Στη συνέχεια, δίνεται ιδιαίτερη προσοχή στους λογικούς τελεστές και στις λογικές συναρτήσεις, ενώ το κεφάλαιο ολοκληρώνεται με μια αναφορά στην προτεραιότητα των εντολών και των πράξεων στο Matlab.

2.2 Βασικές μαθηματικές συναρτήσεις

Οι εκθετικές, οι λογαριθμικές και οι τριγωνομετρικές συναρτήσεις είναι μερικές από τις πιο δημοφιλείς συναρτήσεις σε διάφορα επιστημονικά πεδία. Για τον λόγο αυτό, το Matlab ενσωματώνει έτοιμα προγράμματα/εντολές για τον υπολογισμό των τιμών των συναρτήσεων αυτών, χωρίς να απαιτείται η σύνταξη κώδικα κάθε φορά που απαιτείται ο υπολογισμός των τιμών τους. Μάλιστα, οι εντολές που χρησιμοποιεί το Matlab για τις συναρτήσεις αυτές είναι απλές και μοιάζουν πολύ με τις αγγλικές ονομασίες των συναρτήσεων αυτών, έτσι ώστε να διευκολύνεται η απομνημόνευσή τους.

2.2.1 Τριγωνομετρικές συναρτήσεις

Οι τριγωνομετρικές συναρτήσεις είναι περιοδικές συναρτήσεις που ορίζονται πάνω σε γωνίες και είναι σημαντικές στη μελέτη όχι μόνο τριγώνων και γενικά γεωμετρικών σχημάτων αλλά και, μεταξύ άλλων, για τη μοντελοποίηση περιοδικών φαινομένων.

Οι τρεις βασικές τριγωνομετρικές συναρτήσεις είναι:

- το ημίτονο (sine),
- το συνημίτονο (cosine),
- η εφαπτομένη (tangent).

Συχνά, οι ακόλουθες τρεις συναρτήσεις αναφέρονται ως “δευτερεύουσες”:

- η συντέμνουσα (cosecant),
- η τέμνουσα (secant),
- η συνεφαπτομένη (cotangent).

Η εφαπτομένη αλλά και οι τρεις δευτερεύουσες συναρτήσεις μπορούν να οριστούν με βάση το ημίτονο και το συνημίτονο μέσω των σχέσεων

$$\text{εφ}\theta = \frac{\text{ημ}\,\theta}{\text{συν}\theta},\ \text{τεμ}\theta = \frac{1}{\text{ημ}\theta},\ \text{στεμ}\theta = \frac{1}{\text{συν}\theta},\ \text{σφ}\theta = \frac{\text{συν}\theta}{\text{ημ}\theta}.$$

Παρ΄ όλα αυτά, το Matlab έχει ενσωματωμένες εντολές για όλες αυτές τις συναρτήσεις. Έτσι, παραδείγματος χάριν, η παρακάτω εντολή επιστρέφει τις τιμές των συναρτήσεων αυτών για $\theta = 0$.

```
>> [sin(0),cos(0),tan(0),cot(0),csc(0),sec(0)]
ans =
     0     1     0   Inf   Inf     1
```

Σημειώνεται ότι οι γωνία θ στις συναρτήσεις αυτές στο Matlab εκφράζεται σε ακτίνια. Επομένως, η εντολή

```
>> cos(pi)
ans =
    -1
```

επιστρέφει την τιμή του συνημιτόνου για γωνία π, δηλαδή 180 μοιρών. Η εντολή pi δηλώνει στο Matlab τη σταθερά π.

Παρατήρηση 2.1

Το Matlab δίνει τη δυνατότητα να καλέσουμε τις τριγωνομετρικές συναρτήσεις χρησιμοποιώντας και μοίρες εκτός από ακτίνια. Πιο συγκεκριμένα, οι εντολές

$$sind(x),\ cosd(x),\ tand(x),\ cotd(x),\ cscd(x),\ secd(x)]$$

δέχονται ως όρισμα τη γωνία x εκφραζόμενη σε μοίρες.

Παρατήρηση 2.2

Όλες οι τριγωνομετρικές συναρτήσεις μπορούν να εφαρμοστούν και σε διανύσματα ή πίνακες, επιστρέφοντας διανύσματα ή πίνακες αντίστοιχης διάστασης. Παραδείγματος χάριν, η εντολή

```
>> cos([0,pi/2,pi])
```

επιστρέφει τις τιμές 1, 0 και -1, δηλαδή τις τιμές του συν0, συν$\frac{\pi}{2}$ και συνπ, αντίστοιχα.

Ολοκληρώνοντας την παρουσίαση των τριγωνομετρικών συναρτήσεων, αξίζει να αναφερθούν και οι εντολές που επιστρέφουν τις αντίστροφες τριγωνομετρικές συναρτήσεις. Οι εντολές αυτές είναι οι ακόλουθες:

- τόξο ημίτονου: asin,
- τόξο συνημίτονου: acos,
- τόξο εφαπτομένης: atan,
- τόξο συντέμνουσας: acot,
- τόξο τέμνουσας: acsc,
- τόξο συνεφαπτομένης: asec.

Έτσι, η εντολή

```
>> acos(-1)
```

επιστρέφει την τιμή 3.1416, που στην ουσία ισούται με τη σταθερά π.

2.2.2 Εκθετικές και λογαριθμικές συναρτήσεις

Οι εκθετικές και λογαριθμικές συναρτήσεις αποτελούν βασικές συναρτήσεις για την περιγραφή πολλών φυσικών φαινομένων. Μάλιστα, η τιμή y του λογαρίθμου με βάση τον αριθμό α ενός αριθμού x συμβολίζεται με $y = \log_\alpha x$ και είναι τέτοια, ώστε να ικανοποιεί τη σχέση

$$\alpha^y = x.$$

Από τα παραπάνω, είναι φανερό ότι η συνάρτηση $y = \log_\alpha x$ είναι η αντίστροφη της $\alpha^y = x$.

Στο Matlab οι τιμές των εκθετικών συναρτήσεων υπολογίζονται με τη βοήθεια του συμβόλου των δυνάμεων, που είδαμε στο προηγούμενο κεφάλαιο. Πιο συγκεκριμένα, οι εντολές

```
>> x=-1;
>> 2.1^x
```

επιστρέφουν την τιμή 0.4762, που είναι η τιμή της συνάρτηση 2.1^x για $x = 2.1$.

Σχετικά με τις λογαριθμικές συναρτήσεις πρέπει να αναφέρουμε ότι οι εντολές `log2(x)`, `log10(x)` και `log(x)` επιστρέφουν τον λογάριθμο με βάση το 2, το 10 και το e του αριθμού x, αντίστοιχα. Παραδείγματος χάριν, ο υπολογισμός των τιμών των `log2(x)`, `log10(x)` και `log(x)` για `x=10` επιτυγχάνεται με την εκτέλεση της εντολής

```
>> [log2(10),log10(10),log(10)]
ans =
    3.3219    1.0000    2.3026
```

Παρατήρηση 2.3

Το Matlab δεν έχει ενσωματωμένη συνάρτηση για την αντίστροφη συνάρτηση οποιασδήποτε εκθετικής συνάρτησης, δηλαδή για οποιαδήποτε βάση για τον λογάριθμο. Παρ΄ όλα αυτά, μπορούμε να υπολογίσουμε την τιμή του λογαρίθμου με οποιαδήποτε βάση χρησιμοποιώντας

- την ιδιότητα αλλαγής βάσης του λογαρίθμου

$$\log_b(x) = \frac{\log_k(x)}{\log_k(b)}$$

- μια από τις ενσωματωμένες εντολές του Matlab για τον υπολογισμό της τιμής της συνάρτησης του λογαρίθμου με βάση το 2, το 10 ή τον φυσικό Νεπέριο αριθμό e.

Παρατήρηση 2.4

Οι τιμές της συνάρτησης e^x υπολογίζονται στο Matlab με την εντολή `exp(x)`. Επομένως, ο φυσικός Νεπέριος αριθμός e μπορεί να υπολογιστεί πληκτρολογώντας την εντολή `exp(1)`.

Παρατήρηση 2.5

Σημειώνεται ότι οποτεδήποτε στη συνέχεια του βιβλίου χρησιμοποιείται ο όρος $\log(x)$, αυτός θα αναφέρεται στον λογάριθμο με βάση το e. Ο όρος $ln(x)$ μπορεί να χρησιμοποιείται και αυτός χωρίς καμία διάκριση από τον όρο $\log(x)$.

2.2.3 Τετραγωνική ρίζα και n-οστή ρίζα

Στο προηγούμενο κεφάλαιο αναφερθήκαμε στην εντολή που επιστρέφει την τετραγωνική ρίζα ενός αριθμού. Η εντολή αυτή είναι η `sqrt`. Αξίζει να σημειωθεί ότι η εντολή *sqrt* δεν λαμβάνει όρισμα μόνο μη αρνητικούς αριθμούς. Έτσι, η εντολή

```
>> sqrt(-1)
```

δεν επιστρέφει σφάλμα, αλλά το αποτέλεσμα

```
ans =
   0.0000 + 1.0000i
```

δηλαδή τον μιγαδικό αριθμό $0 + i$. Με τους μιγαδικούς αριθμούς θα ασχοληθούμε αναλυτικά στο Κεφάλαιο 7.

Μια γενίκευση της εντολής *sqrt* είναι η εντολή `nthroot`, η οποία επιστρέφει τη n-οστή ρίζα ενός αριθμού. Το συντακτικό της εμφανίζεται στο επόμενο παράδειγμα:

```
>> nthroot(8,3)
ans =
     2
```

στο οποίο έχουμε λάβει την $\sqrt[3]{8}$, η οποία ισούται με 2.

2.2.4 Συναρτήσεις στρογγυλοποίησης

Μια σημαντική οικογένεια συναρτήσεων είναι οι συναρτήσεις στρογγυλοποίησης. Στο Matlab υπάρχουν τέσσερις συναρτήσεις, οι οποίες εκτελούν τις ακόλουθες διαδικασίες στρογγυλοποίησης ενός αριθμού x.

- round: στρογγυλοποιεί τον αριθμό x προς τον πλησιέστερο ακέραιο αριθμό.
- ceil: στρογγυλοποιεί τον αριθμό x προς τον πλησιέστερο ακέραιο προς το $+\infty$.
- floor: στρογγυλοποιεί τον αριθμό x προς τον πλησιέστερο ακέραιο προς το $-\infty$.
- fix: στρογγυλοποιεί τον αριθμό x προς τον πλησιέστερο ακέραιο προς το μηδέν. Ο αριθμός αυτός αναφέρεται και ως το ακέραιο μέρος του αριθμού.

Παράδειγμα 2.1

Να εξετάσετε ποια από τα $y1, y2, y3$ και $y4$, που ορίζονται με τις ακόλουθες εντολές, είναι ίσα μεταξύ τους και να εξηγήσετε γιατί συμβαίνει αυτό.

```
>> x=-log(2.1);
>> y1=round(x);
>> y2=ceil(x);
>> y3=floor(x);
>> y4=fix(x);
```

Λύση Παραδείγματος 2.1

Η τιμή της μεταβλητής x ισούται με -0.7419. Επομένως, τα $y1$ και $y3$ ισούνται με -1, ενώ τα $y2$ και $y4$ ισούνται με μηδέν. Αυτό συμβαίνει διότι

- ο πλησιέστερος ακέραιος αριθμός στο -0.7419 είναι το -1,
- ο πλησιέστερος ακέραιος αριθμός προς το $-\infty$ για το -0.7419 είναι πάλι το -1

και, επομένως, τα $y1$ και $y3$ ισούνται και τα δύο με -1. Από την άλλη, ο πλησιέστερος ακέραιος αριθμός προς το $+\infty$ ή το μηδέν είναι το 0 και για αυτό τον λόγο τα $y2$ και $y4$ είναι ίσα μεταξύ τους.

Άσκηση αυτοαξιολόγησης 2.1

Να προσδιορίσετε το ακέραιο και το δεκαδικό μέρος της πέμπτης ρίζας του e^3.

Παρατήρηση 2.6

Η εντολή round μπορεί να συνταχθεί και ως round(X,N), όπου το N είναι ένας ακέραιος αριθμός. Αν το N είναι ένας θετικός ακέραιος, τότε ο αριθμός X στρογγυλοποιείται στο N-οστό ψηφίο μετά την υποδιαστολή. Αν το N είναι ένας αρνητικός ακέραιος, τότε ο X στρογγυλοποιείται στο N-οστό ψηφίο αριστερά της υποδιαστολής, ενώ, αν ισούται με μηδέν, ο X στρογγυλοποιείται στον πλησιέστερο ακέραιο. Τα αποτελέσματα από τη χρήση της εντολής round(X,N) παρουσιάζονται στα επόμενα δύο παραδείγματα:

```
>> round(sqrt(2), 2)
ans =
    1.4100
>> round(121323, -3)
ans =
      121000
```

2.2.5 Άλλες συναρτήσεις

Ολοκληρώνοντας την παρουσίαση των βασικών συναρτήσεων και των εντολών με τις οποίες μπορούμε να υπολογίσουμε τις τιμές στο Matlab, αξίζει να αναφερθούμε σε μια σειρά επιπρόσθετων συναρτήσεων και εντολών, οι οποίες είναι ιδιαίτερα χρήσιμες στις διάφορες εφαρμογές. Στον πίνακα 2.1 εμφανίζονται οι συναρτήσεις αυτές, καθώς και κάποια ενδεικτικά παραδείγματα για τον τρόπο σύνταξής τους στο Matlab.

2.2.6 Σταθερές

Το Matlab περιέχει ενσωματωμένες μια σειρά από σημαντικές σταθερές. Από τις πιο συχνά χρησιμοποιούμενες είναι το π, το οποίο, όπως έχουμε δει, δηλώνεται στο Matlab με το pi. Άλλες σταθερές που υπάρχουν ενσωματωμένες στο Matlab είναι:

- η φανταστική μονάδα, η οποία δηλώνεται με το i ή το j,
- το γ του Euler (Havil, 2003), η οποία δηλώνεται με το όνομα `eulergamma` στο Matlab.

Αξίζει να σημειωθεί ότι οι σταθερές π και γ είναι άρρητοι αριθμοί, Παρ΄ όλα αυτά, το Matlab, επειδή είναι ένα πρόγραμμα που χρησιμοποιείται για την τέλεση αριθμητικών και όχι συμβολικών πράξεων, χρησιμοποιεί

Συνάρτηση	Εντολή	Παραδείγματα

Απόλυτη τιμή	abs	`>> abs(-3)`
		`ans =`
		`3`
		`>> abs([-3, 0, 2])`
		`ans =`
		`3 0 2`

Πρόσημο	sign	`>> sign([-3, 0, 2])`
		`ans =`
		`-1 0 1`

Άθροισμα	sum	`>> sum([-3, 0, 2])`
		`ans =`
		`-1`

Γινόμενο	prod	`>> prod([-3, 0, 2])`
		`ans =`
		`0`

Ελάχιστο	*min*	`>> min([-3, 0, 2])`
		`ans =`
		`-3`

Μέγιστο	*max*	`>> max([-3, 0, 2])`
		`ans =`
		`2`

Πίνακας 2.1: Βασικές συναρτήσεις και παραδείγματα για τον τρόπο σύνταξής τους στο Matlab.

πεπερασμένο πλήθος ψηφίων για τον υπολογισμό τους κατά τις διάφορες πράξεις. Αυτό έχει ως συνέπεια να εμφανίζονται διάφορα μικρά σφάλματα, όπως το ακόλουθο για το ημπ, το οποίο, ως γνωστόν, ισούται με μηδέν, αλλά από το Matlab επιστρέφεται ως:

```
>> sin(pi)
ans =
   1.2246e-16
```

Το λάθος αυτό οφείλεται στο γεγονός ότι το Matlab χρησιμοποιεί ένα μεγάλο αλλά πεπερασμένο αριθμό ψηφίων, για να αναπαριστά το π (αλλά και τους άλλους άρρητους) στις πράξεις που εκτελεί εσωτερικά, με συνέπεια το π που έχει στη μνήμη ο υπολογιστής να μην είναι ακριβώς ίσο με την πραγματική τιμή του π. Το λάθος, βεβαίως, που προκύπτει από αυτήν την τακτική είναι εξαιρετικά μικρό, αφού, ακόμα και αν στρογγυλοποιήσουμε στο 15^o δεκαδικό ψηφίο, το αποτέλεσμα θα είναι μηδέν, όπως φαίνεται και στο αποτέλεσμα της επόμενης εντολής στο Matlab.

```
>> round(sin(pi),15)
ans =
     0
```

Αυτό δεν σημαίνει ότι δεν πρέπει να είμαστε προσεκτικοί όταν έχουμε να κάνουμε πράξεις με άρρητους αριθμούς. Αυτό το μικρό σφάλμα, αν επαναληφθεί πολλές φορές, μπορεί να δώσει σημαντικά εσφαλμένα αποτελέσματα.

Κλείνοντας την αναφορά στις σταθερές που έχει το Matlab και πώς αυτό χειρίζεται τους άρρητους αριθμούς, αξίζει να αναφέρουμε ότι μπορούμε να τυπώσουμε ένα μεγάλο πλήθος των ψηφίων που χρησιμοποιεί το Matlab για την αναπαράσταση των άρρητων αριθμών χρησιμοποιώντας την εντολή vpa. Παραδείγματος χάριν, αν θέλουμε να τυπώσουμε το π με τα πρώτα 780 ψηφία του (συμπεριλαμβανομένου του 3 στο 3.14....), θα πρέπει να πληκτρολογήσουμε την εντολή

```
>> vpa(pi,780)
```

η οποία τυπώνει την τιμή του π στρογγυλοποιημένη στο 779^o δεκαδικό ψηφίο.

2.3 Λογικές συναρτήσεις - Λογικοί τελεστές

Οι λογικές συναρτήσεις εφαρμόζονται πάνω στις αποκαλούμενες λογικές μεταβλητές και μπορούν να λάβουν μόνο δύο τιμές, το μηδέν και το ένα, ανάλογα με το αν το αποτέλεσμα της συνάρτησης είναι ψευδές ή αληθές. Οι λογικές μεταβλητές προκύπτουν συχνά ως αποτέλεσμα λογικών προτάσεων, οι οποίες εκφράζονται με χρήση των αποκαλούμενων λογικών τελεστών ή, αλλιώς, λογικών συνδέσμων. Οι λογικές προτάσεις είναι προτάσεις για τις οποίες μπορούμε να αποφασίσουμε αν είναι αληθείς ή όχι. Οι λογικές προτάσεις αποτελούν βασικά εργαλεία για την ανάπτυξη κώδικα και την επίλυση απλών αλλά και σύνθετων προβλημάτων με τη βοήθεια των υπολογιστών.

Στη συνέχεια, παρουσιάζονται οι βασικές λογικές συναρτήσεις και τελεστές. Στο τέλος της ενότητας παρουσιάζονται και μια σειρά από συναρτήσεις ελέγχων που έχει ενσωματωμένες το Matlab και βασίζονται στις λογικές προτάσεις.

Για μια πιο αναλυτική παρουσίαση των λογικών συναρτήσεων και του κλάδου των μαθηματικών (άλγεβρα Boole) που ασχολείται με τις λογικές μεταβλητές και τις λογικές συναρτήσεις ο/η αναγνώστης/στρια παραπέμπεται στα βιβλία των Givant και Halmos (2008) και Zohuri και Moghaddam (2017).

2.3.1 Βασικές λογικές συναρτήσεις και τελεστές

Κάθε λογική πρόταση είναι είτε αληθής είτε ψευδής. Στα μαθηματικά και στον προγραμματισμό, όπως προαναφέρθηκε, κάθε λογική πρόταση ισούται με ένα, αν είναι αληθής, και με μηδέν, αν είναι ψευδής. Με βάση αυτήν την παρατήρηση, είναι εύκολο να κατανοηθούν τα αποτελέσματα των εντολών που παρουσιάζονται στη συνέχεια, οι οποίες συγκρίνουν τις τιμές τριών μεταβλητών με τη βοήθεια

- του **μεγαλύτερου** (“>”),
- του **μικρότερου** (“<”),
- του **μεγαλύτερου ή ίσου** (“>=”),
- του **μικρότερου ή ίσου** (“<=”),

- του **ίσου** (“==”),
- του **όχι ίσου - διάφορου** (“~=”),

τα οποία αποτελούν τα δομικά υλικά των λογικών προτάσεων.

```
>> x=4;y=5;z=0:5;
>> x>y % megalitero
ans =
     0
>> x<y % mikrotero
ans =
     1
>> z>=y % megalitero i iso
ans =
     0     0     0     0     0     1
>> z<=x % mikrotero i iso
ans =
     1     1     1     1     1     0
>> x==y % iso
ans =
     0
>> x==z % iso
ans =
     0     0     0     0     1     0
>> x~=x % iso
ans =
     0
>> x~=z % iso
ans =
     1     1     1     1     0     1
```

Για παράδειγμα, η εντολή $x > y$ επιστρέφει την τιμή 0, δηλαδή ψευδές, αφού το x που ισούται με 4 δεν είναι μεγαλύτερο από το y που ισούται με 5. Ενδιαφέρον παρουσιάζουν οι εντολές με τις οποίες συγκρίνεται το x με το διάνυσμα $z = [0,1,2,3,4,5]$. Σε αυτές τις περιπτώσεις το Matlab συγκρίνει την τιμή του x με καθένα στοιχείο του z και επιστρέφει ένα λογικό διάνυσμα με μηδενικά και άσους, διάστασης ίσης με το z και αποτέλεσμα της εφαρμογής του αντίστοιχου λογικού τελεστή σε καθένα ξεχωριστό στοιχείο του z. Έτσι η εντολή $x == z$ επιστρέφει ένα διάνυσμα με όλα τα στοιχεία ίσα με το μηδέν εκτός από το προτελευταίο που είναι ίσο με ένα, επειδή το προτελευταίο στοιχείο του z είναι πράγματι ίσο με τη τιμή του x, δηλαδή το 4.

Οι παραπάνω λογικοί τελεστές μπορούν να χρησιμοποιηθούν για να κατασκευαστούν πιο σύνθετες λογικές προτάσεις. Οι λογικές αυτές προτάσεις κατασκευάζονται με βάση τις τρεις κύριες πράξεις της άλγεβρας Boole:

- τη σύζευξη (σύμβολο Matlab: &&),
- τη διάζευξη (σύμβολο Matlab: ||),
- την άρνηση (σύμβολο Matlab: ~).

Οι τρεις αυτές πράξεις μπορούν να ορίσουν όλες τις λογικές πράξεις μεταξύ λογικών προτάσεων. Τα

αποτελέσματά τους περιγράφονται στον παρακάτω πίνακα αλήθειας

p	q	$p\&\&q$	$p\|\|q$	$\sim p$
T	T	T	T	F
T	F	F	T	
F	T	F	T	T
F	F	F	F	

στον οποίο έχουμε συμβολίσει με p και q δύο λογικές προτάσεις, ενώ με T και F την αλήθεια (True) και το ψέμα (False), αντίστοιχα.

Έτσι, η (p && q) που εκφράζει το αποτέλεσμα της λογικής πρότασης p **ΚΑΙ** q επιστρέφει 1, δηλαδή αλήθεια, μόνο αν και οι δύο προτάσεις είναι αληθείς. Από την άλλη, η (p || q), η οποία εκφράζει το αποτέλεσμα της λογικής πρότασης p **Ή** q, επιστρέφει 1 στις περιπτώσεις που τουλάχιστον μία από τις προτάσεις είναι αληθής. Αν και οι δύο προτάσεις είναι ψευδείς, τότε η (p || q) επιστρέφει την τιμή μηδέν. Τέλος, η άρνηση μιας πρότασης ($\sim p$) επιστρέφει το μηδέν, αν η πρόταση είναι αληθής, και το ένα, αν είναι ψευδής.

Παράδειγμα 2.2

Να ορίσετε στο Matlab τις μεταβλητές x και y και να αναθέσετε σε αυτές τις τιμές 4 και 5, αντίστοιχα. Στη συνέχεια, να υπολογίσετε τα αποτελέσματα των λογικών προτάσεων (x<y&&2*x>y) και (x<y || y<3). Δικαιολογήστε τις τιμές που επιστρέφει το Matlab για καθεμία από αυτές τις προτάσεις.

Λύση Παραδείγματος 2.2

Ο ορισμός των μεταβλητών x και y, η ανάθεση σε αυτές των τιμών 4 και 5, αντίστοιχα, αλλά και η πραγματοποίηση των λογικών προτάσεων της άσκησης γίνεται με την εκτέλεση των επόμενων εντολών στο Matlab:

```
>> x=4;y=5;
>> x<y && 2*x>y
ans =
     1
>> x<y || y<3
ans =
     1
```

Παρατηρούμε ότι και οι δύο λογικές προτάσεις επιστρέφουν την τιμή ένα, δηλαδή και οι δύο προτάσεις είναι αληθείς.
Πράγματι, η πρόταση (x<y&&2*x>y) είναι αληθής, αφού και οι δύο επιμέρους προτάσεις είναι αληθείς. Πιο συγκεκριμένα,

- η x<y είναι αληθής, αφού το 4 είναι πράγματι μικρότερο του 5,
- η 2*x>y είναι αληθής, αφού το 2*4=8 είναι μεγαλύτερο του 5

και, επομένως, αφού οι δύο προτάσεις συνδέονται με το **ΚΑΙ** (&&), η (x<y&&2*x>y) είναι αληθής και ως εκ τούτου το Matlab επιστρέφει το ένα.
Η πρόταση (x<y || y<3) είναι αληθής και για αυτό το Matlab επιστρέφει το ένα, αφού, ναι μεν η (y<3) είναι ψευδής, αλλά η (x<y) είναι αληθής. Επομένως, αφού οι δύο προτάσεις συνδέονται με το (||), αρκεί να είναι τουλάχιστον μία από τις επιμέρους προτάσεις αληθής, όπως πράγματι είναι, για να είναι αληθής η συνολική πρόταση.

Άσκηση αυτοαξιολόγησης 2.2

Για τις μεταβλητές x και y και τις τιμές που ανατέθηκαν σε αυτές στην άσκηση 2.2 να υπολογίσετε τα αποτελέσματα των λογικών προτάσεων (x>y && y<3) και (x>y || (y<3)) και να δικαιολογήσετε τις τιμές που επιστρέφει το Matlab.

Πέρα από τις τρεις κύριες πράξεις της άλγεβρας Boole, που αναφέρθηκαν νωρίτερα, μπορούν να οριστούν και μια σειρά από άλλες, οι οποίες όμως χαρακτηρίζονται ως δευτερεύουσες, με την έννοια ότι μπορούν να εκφραστούν με τη βοήθεια των τριών κύριων πράξεων. Η βασικότερη εξ αυτών είναι η λεγόμενη αποκλειστική διάζευξη (εντολή Matlab: `xor`), η οποία εφαρμόζεται σε δύο επιμέρους προτάσεις και επιστρέφει την τιμή ένα, δηλαδή χαρακτηρίζεται ως αληθής, μόνο αν ακριβώς μία από τις προτάσεις αυτές είναι αλήθεια. Ο πίνακας αλήθειας της αποκλειστικής διάζευξης εμφανίζεται στη συνέχεια.

p	q	p `xor` q
T	T	F
T	F	T
F	T	T
F	F	F

Το συντακτικό της `xor` στο Matlab εμφανίζεται στις επόμενες εντολές. Στις εντολές αυτές εμφανίζεται και η ισοδύναμη έκφραση με τη χρήση αποκλειστικά των τριών κύριων πράξεων.

```
>> x=4;y=5;
>> xor(x<y,2*x>y)
ans =
     0
>> x<y && ~(2*x>y) || ~(x<y) && 2*x>y
ans =
     0
```

Σημειώνεται ότι η πρόταση `xor(x<y,2*x>y)` είναι ψευδής, διότι και οι δύο επιμέρους προτάσεις είναι αληθείς.

Παρατήρηση 2.7

Στο Matlab, όπως έχουμε αναφέρει, με μηδέν δηλώνεται η ψευδής πρόταση, ενώ με ένα η αληθής. Με βάση αυτό και παρατηρώντας ότι:

- η άρνηση του ψέματος έχει ως αποτέλεσμα την αλήθεια,
- η άρνηση της αλήθειας έχει ως αποτέλεσμα το ψέμα,

είναι εύκολο να κατανοήσουμε τα αποτελέσματα που επιστρέφει το Matlab στην παρακάτω εντολή.

```
>> ~[0, 1]
ans =
   1   0
```

Επεκτείνοντας την παραπάνω λογική και θεωρώντας ότι κανένας άλλος αριθμός πέρα από το μηδέν δεν μπορεί να δοθεί σε μια ψευδή πρόταση, μπορούμε να κατανοήσουμε τα λογικά διανύσματα (δηλαδή τα διανύσματα με μηδενικά και άσους) που επιστρέφει το Matlab ως αποτελέσματα στις παρακάτω εντολές:

```
>> ~[1,2,3,4,0,1,0]
ans =
     0     0     0     0     1     0     1

>> ~(~[1,2,3,4,0,1,0])
ans =
     1     1     1     1     0     1     0
```

2.3.2 Λογικές συναρτήσεις ελέγχων

Το Matlab έχει ενσωματωμένες διάφορες εντολές που εκτελούν συγκεκριμένους ελέγχους με βάση τις λογικές προτάσεις που παρουσιάστηκαν νωρίτερα για τα στοιχεία διανυσμάτων ή πινάκων. Μερικές από αυτές τις εντολές, το συντακτικό τους, καθώς και η περιγραφή για το τι ακριβώς κάνουν εμφανίζονται στον Πίνακα 2.2.

Στη συνέχεια, εμφανίζονται κάποια ενδεικτικά παραδείγματα από την εφαρμογή των εντολών αυτών.

```
>> A=[1,3,5,7];
>> B=1:2:8;
>> isequal(A,B)
ans =
  logical
   1

>> x=[1:0];
>> isempty(x)
ans =
  logical
   1

>> isinf([pi NaN 2 Inf -Inf])
ans =
  1x5 logical array
   0   0   0   1   1

>>  isnan([pi NaN 2 Inf -Inf])
ans =
  1x5 logical array
   0   1   0   0   0
```

Σε πολλές περιπτώσεις χρήσιμες μπορεί να αποδειχτούν και οι εντολές:

- `any`,
- `all`,

όταν εφαρμόζουμε λογικές εντολές σε λογικά διαστήματα, όπως αυτά που επιστρέφουν οι 2 τελευταίες εντολές στα παραπάνω παραδείγματα.

Εντολή	Συντακτικό και περιγραφή
`isequal`	Η εντολή `isequal(A,B)` επιστρέφει • τη λογική τιμή 1, δηλαδή αλήθεια, αν τα A και B είναι ίδιας διάστασης και τα στοιχεία τους είναι ίδια μεταξύ τους και • τη λογική τιμή 0, δηλαδή ψέμα, σε διαφορετική περίπτωση.
`isempty`	Η εντολή `isempty(A)` επιστρέφει • τη λογική τιμή 1, αν το A είναι ένα κενό διάνυσμα και • τη λογική τιμή 0, αν περιέχει στοιχεία.
isinf	Η εντολή `isinf(A)` επιστρέφει έναν πίνακα μηδενικών και άσων, διάστασης ίσης με τη διάσταση του A. Η τιμή 1 εμφανίζεται στις θέσεις, όπου ο πίνακας A περιέχει τα $-\infty$ ή $+\infty$ και το μηδέν στις υπόλοιπες θέσεις.
`isnan`	Η εντολή `isnan(A)` επιστρέφει έναν πίνακα μηδενικών και άσων, διάστασης ίσης με τη διάσταση του A. Η τιμή 1 εμφανίζεται στις θέσεις, όπου ο πίνακας A περιέχει το `NaN` (μη αριθμός) και το μηδέν στις υπόλοιπες θέσεις.

Πίνακας 2.2: Λογικές συναρτήσεις ελέγχων στο Matlab.

Η εντολή `any` επιστρέφει την τιμή 1, αν υπάρχει τουλάχιστον ένα μη μηδενικό στοιχείο σε ένα λογικό διάνυσμα, και μηδέν, διαφορετικά. Επομένως, η εντολή `any` επιστρέφει την τιμή 1, αν υπάρχει τουλάχιστον ένα στοιχείο που αντιπροσωπεύει μια αληθή πρόταση. Σημειώνεται ότι η εντολή `any` αγνοεί τα στοιχεία που είναι `NaN`. Από την άλλη, η εντολή `all` επιστρέφει την τιμή 1, αν όλα τα στοιχεία σε ένα λογικό διάνυσμα είναι μη μηδενικά, δηλαδή αν όλα τα στοιχεία αντιπροσωπεύουν αληθείς προτάσεις.

Στη συνέχεια, εμφανίζονται τα αποτελέσματα από την εφαρμογή των εντολών `any` και `all` στο αποτέλεσμα της λογικής συνάρτησης ελέγχου `isinf` στο διάνυσμα [pi, NaN, 2, Inf, -Inf].

```
>> any(isinf([pi NaN 2 Inf -Inf]))
ans =
  logical
   1

>> all(isinf([pi NaN 2 Inf -Inf]))
ans =
  logical
   0
```

Από τα παραπάνω αποτελέσματα βλέπουμε ότι το Matlab επιστρέφει:

- την τιμή ένα στην πρώτη εντολή, αφού το διάνυσμα [pi, NaN, 2, Inf, -Inf] έχει τουλάχιστον ένα στοιχείο που ισούται με $-\infty$ ή $+\infty$,
- την τιμή μηδέν στη δεύτερη εντολή, αφού το διάνυσμα [pi, NaN, 2, Inf, -Inf] περιέχει και στοιχεία που δεν ισούται με $-\infty$ ή $+\infty$.

Παράδειγμα 2.3

Οι εντολές `zeros(n,k)` και `ones(n,k)` δημιουργούν $n \times k$ πίνακες μηδενικών και άσων, αντίστοιχα. Με βάση τις μεταβλητές

```
>> z=0:5; w=[0,1,22,-3,0,1]; q=[0,1,2,3,4,5]
>> zeromat=zeros(1,3); nonzeromat=[2,-2,3]; onemat=ones(1,3);
```

να προσδιορίσετε τα αποτελέσματα των εντολών:

```
>> isequal(z,q)
>> ~isequal(z,q)
>> [any(z),any(w),any(zeromat),any(nonzeromat),any(onemat)]
>> [all(z),all(w),all(zeromat),all(nonzeromat),all(onemat)]
```

Στη συνέχεια, να επιβεβαιώσετε τα αποτελέσματα, εκτελώντας τις εντολές αυτές στο Matlab.

Λύση Παραδείγματος 2.3

Επειδή η μεταβλητή z αναπαριστά το διάνυσμα $[0,1,2,3,4,5]$, δηλαδή το q, η εντολή `isequal(z,q)` επιστρέφει την τιμή 1, αφού τα δύο διανύσματα ταυτίζονται. Η εντολή `~isequal(z,q)`, η οποία είναι η άρνηση της αληθούς πρότασης `isequal(z,q)`, επιστρέφει την τιμή 0, ως άρνηση της αλήθειας.
Η τρίτη εντολή αφορά την εφαρμογή της εντολής `any` σε όλες τις μεταβλητές, όπου έχουμε ορίσει. Αφού η εντολή `any` επιστρέφει το ένα στις περιπτώσεις που υπάρχει τουλάχιστον ένα μη μηδενικό στοιχείο, η τρίτη εντολή θα επιστρέψει το διάνυσμα $[1,1,0,1,1]$, αφού μόνο η μεταβλητή `zeromat` έχει όλα τα στοιχεία της ίσα με μηδέν.
Η εφαρμογή της εντολής `all` σε όλες τις μεταβλητές, που έχουμε ορίσει στην τέταρτη εντολή, επιστρέφει το διάνυσμα $[0,0,0,1,1]$, αφού μόνο οι μεταβλητές `nonzeromat` και `onemat` έχουν όλα τα στοιχεία τους διαφορετικά από το μηδέν.
Τα παραπάνω επιβεβαιώνονται και από την εκτέλεση των ακόλουθων εντολών στο Matlab.

```
>> isequal(z,q)
ans =
  logical
   1

>> ~isequal(z,q)
ans =
  logical
   0

>> [any(z),any(w),any(zeromat),any(nonzeromat),any(onemat)]
ans =
  1x5 logical array
   1   1   0   1   1

>> [all(z),all(w),all(zeromat),all(nonzeromat),all(onemat)]
ans =
  1x5 logical array
   0   0   0   1   1
```

Άσκηση αυτοαξιολόγησης 2.3

Να προσδιορίσετε τα αποτελέσματα των εντολών:

```
>> any([x ~= x + 1, x >= 0, x == x])
>> all([x ~= x + 1, x >= 0, x == x])
```

για τη μεταβλητή

```
>> x=sin(exp(1))
```

2.3.3 Η εντολή find

Με την εντολή `find(X)` το Matlab επιστρέφει ένα διάνυσμα με τις θέσεις στις οποίες το διάνυσμα X παίρνει μη μηδενικές τιμές. Μια ενδιαφέρουσα εφαρμογή της εντολής `find` είναι όταν το X είναι ένα λογικό διάνυσμα. Στην περίπτωση αυτή, η εντολή `find` επιστρέφει τις θέσεις των μη μηδενικών στοιχείων του X, δηλαδή των στοιχείων που αντιπροσωπεύουν αληθείς λογικές προτάσεις.

Η εντολή `find` μπορεί να συνταχθεί και με διαφορετικούς τρόπους, όπως οι ακόλουθοι:

- `find(X,K)`: επιστρέφει τις K πρώτες θέσεις των μη μηδενικών στοιχείων του X. Αν υπάρχουν λιγότερα από K μη μηδενικά στοιχεία, επιστρέφει όλες τις θέσεις των στοιχείων αυτών,
- `find(X,K,'first')`: το ίδιο με την εντολή `find(X,K)`,
- `find(X,K,'last')` επιστρέφει τις K τελευταίες θέσεις των μη μηδενικών στοιχείων του X. Αν υπάρχουν λιγότερα από K μη μηδενικά στοιχεία, επιστρέφει όλες τις θέσεις των στοιχείων αυτών.

Παράδειγμα 2.4

Χρησιμοποιώντας την εντολή `find`, να βρείτε τα πολλαπλάσια του 0.5 μεταξύ του 1 και του 10, για τα οποία η απόλυτη τιμή της συνάρτησης $cos(x) \cdot exp(-x)$ είναι μεγαλύτερη του 0.02.

Λύση Παραδείγματος 2.4

Εκτελώντας τις εντολές:

```
>> x=1:0.5:10;
>> y=cos(x).*exp(-x);
>> positions=find(abs(y)>0.02)
positions =
     1     3     4     5     6
>> x(positions)
ans =
    1.0000    2.0000    2.5000    3.0000    3.5000
```

ορίζουμε

- το διάνυσμα x με τιμές $[1, 1.5, 2, \ldots, 9.5, 10]$,
- το διάνυσμα y με τις τιμές της $cos(x) \cdot exp(-x)$ πάνω από τα x,
- προσδιορίζουμε και αποθηκεύουμε τις θέσεις του y, στις οποίες η απόλυτη τιμή του $cos(x) \cdot exp(-x)$ είναι μεγαλύτερη του 0.02 και

- επιστρέφουμε τις τιμές του x στις προαναφερθείσες θέσεις.

Από την παραπάνω διαδικασία διαπιστώνουμε ότι τα πολλαπλάσια του 0.5 μεταξύ του 1 και του 10, για τα οποία η απόλυτη τιμή του $cos(x) \cdot exp(-x)$ είναι μεγαλύτερη του 0.02, είναι τα 1, 2, 2.5, 3 και 3.5.

Άσκηση αυτοαξιολόγησης 2.4

Να βρεθεί με ακρίβεια 0.001, αν υπάρχει, η πρώτη ρίζα της εξίσωσης $cos(x) \cdot sin(x) \cdot exp(-x/2) = 0$, που είναι μεγαλύτερη ή ίση από το 5 και μικρότερη ή ίση από το 10.

2.4 Προτεραιότητα πράξεων

Κλείνοντας το κεφάλαιο αυτό, πρέπει να τονίσουμε τη σειρά των αριθμητικών και λογικών πράξεων. Η παρακάτω λίστα περιλαμβάνει τη σειρά με την οποία εκτελούνται οι διάφορες πράξεις στο Matlab:

1. παρενθέσεις
2. δυνάμεις (από αριστερά προς στα δεξιά)
3. άρνηση (“~”)
4. πολλαπλασιασμοί και διαιρέσεις (από αριστερά προς στα δεξιά)
5. προσθέσεις και αφαιρέσεις (από αριστερά προς στα δεξιά)
6. (“:”)
7. λογικές προτάσεις (“>”,“<”,“>=”,“<=”,“==”,“ =”)
8. και (“&&”)
9. ή (“||”)

Παράδειγμα 2.5

Μια εταιρεία αναζητά εργαζόμενους με βαθμό πτυχίου (*degree*) τουλάχιστον 8.0 ή με εργασιακή εμπειρία (*experience*) μεγαλύτερη των 5 ετών. Σε κάθε περίπτωση ο υποψήφιος εργαζόμενος θα πρέπει να πετύχει, επίσης, και βαθμολογία μεγαλύτερη του 7 στο τεστ (*test*) που διεξάγει η εταιρεία για την αξιολόγηση των ικανοτήτων των υποψήφιων.
Ποια ή ποιες από τις παρακάτω λογικές προτάσεις επιλέγει σωστά τους πιθανούς υποψήφιους εργαζόμενους, σύμφωνα με τα κριτήρια της εταιρείας;

1. degree>=8 || experience>2 && test>7
2. (degree>=8 || experience>2) && test>7
3. degree>=8 || (experience>2 && test>7)

Δικαιολογήστε την απάντησή σας.

Λύση Παραδείγματος 2.5

Σωστή απάντηση είναι η δεύτερη, επειδή η εταιρεία ζητά υποχρεωτικά ο εργαζόμενος να έχει βαθμολογία μεγαλύτερη του 7 στο τεστ (*test*) που διεξάγει και να ικανοποιεί τουλάχιστον μια από τις άλλες δύο απαιτήσεις. Επομένως, η παρένθεση που υπάρχει στη δεύτερη εντολή εξασφαλίζει ότι θα εξεταστούν πρώτα οι υποθέσεις με το ή (“||”) και το αποτέλεσμά τους θα εξεταστεί στη συνέχεια, αν αληθεύει ταυτόχρονα (με το και - `&&”) με την άλλη λογική πρόταση.
Η πρώτη είναι εσφαλμένη, διότι το και (“&&”) προηγείται από το ή και (“||”) και, επομένως, θα επιστρέφει το ένα, αν ένας υποψήφιος

- έχει βαθμό πτυχίου μεγαλύτερο του 8.0 ή
- αν έχει εργασιακή εμπειρία (*experience*) μεγαλύτερη των 5 ετών και έχει πετύχει βαθμολογία μεγαλύτερη του 7 στο τεστ (*test*).

Το ΊΔΙΟ ακριβώς εκφράζει και η τρίτη επιλογή, στην οποία η παρένθεση είναι περιττή, αφού, και να έλειπε, οι πράξεις θα γινόντουσαν με τον ίδιο ακριβώς τρόπο.

2.5 Ασκήσεις

Άσκηση 2.1. *Δημιουργήστε ένα script αρχείο το οποίο να επιστρέφει την τιμή των συναρτήσεων*

$$exp\ (x + y^2) \text{ και } \log(\sqrt{|x + y|})$$

για x και y που εισάγει ο χρήστης (input).

Άσκηση 2.2. *Δημιουργήστε ένα αρχείο script, το οποίο θα υπολογίζει τον πίνακα τιμών της συνάρτησης $f(x) = sin(x)/x$ για ένα διάνυσμα τιμών* **x** $= (x_1, x_2, \dots, x_n)$ *που θα εισάγει ο χρήστης. Το αρχείο πρέπει να επιστρέφει τον πίνακα στην ακόλουθη μορφή.*

$x1$	$x2$	...	x_n
$f(x1)$	$f(x2)$	...	$f(x_n)$

Στην περίπτωση που το διάνυσμα x περιέχει την τιμή μηδέν, θα πρέπει να επιστρέφει το όριο της $f(x)$ στο μηδέν, δηλαδή το 1.

Άσκηση 2.3. *Ποιο είναι το αποτέλεσμα των παρακάτω εντολών;*

```
>> x=-2;y=-3;
>> x>-3&&(y<=-3||x==2&&y>3)
```

Άσκηση 2.4. *Δημιουργήστε ένα αρχείο script, το οποίο να πληροφορεί τον χρήστη αν η n-οστή ρίζα ενός θετικού αριθμού x είναι μεγαλύτερη από 2 ή όχι για x και n που εισάγει ο χρήστης.*

Άσκηση 2.5. *Προσδιορίστε και δικαιολογήστε τα αποτελέσματα των εντολών:*

```
>> isempty(a)
>> isinf(b)
>> isnan(b)
```

για τις μεταβλητές

```
>> a=[]; b=[Inf,-3,0,NaN];
```

Άσκηση 2.6. *Ποια θα είναι η τιμή του h μετά την εκτέλεση των παρακάτω εντολών;*

```
>> A=[3.2 2.6 -0.3 8.1 3.9];
>> h=find(A>4)
```

Άσκηση 2.7. *Να βρεθεί με ακρίβεια 0.001, αν υπάρχει, η πρώτη ρίζα της εξίσωσης $cos(x) \cdot sin(x) = 0$ που είναι μεγαλύτερη ή ίση από το 6.*

2.6 Απαντήσεις στις ασκήσεις αυτοαξιολόγησης

Λύση άσκησης αυτοαξιολόγησης 2.1

Το ακέραιο μέρος, yi, και το δεκαδικό μέρος, yd, της πέμπτης ρίζας του e^3 μπορούν να υπολογιστούν με τις εντολές:

```
>> x=nthroot(exp(3),5);
>> yi=fix(x);
>> yd=x-yi;
>> [yi, yd]
```

Η τελευταία εντολή επιστρέφει το ακόλουθο αποτέλεσμα:

```
ans =
    1.0000    0.8221
```

που είναι το ακέραιο και το δεκαδικό μέρος, αντίστοιχα, της πέμπτης ρίζας του e^3.

Λύση άσκησης αυτοαξιολόγησης 2.2

Η εκτέλεση των λογικών προτάσεων της άσκησης γίνεται με την εκτέλεση των επόμενων εντολών στο Matlab:

```
>> x=4;y=5;
>> x>y && y<3
ans =
     0
>> x>y || ~(y<3)
ans =
     1
```

Παρατηρούμε ότι η πρώτη λογική πρόταση επιστρέφει την τιμή μηδέν, δηλαδή είναι ψευδής, ενώ η δεύτερη είναι αληθής και για αυτό επιστρέφει την τιμή ένα.
Ο λόγος που η πρώτη πρόταση είναι ψευδής είναι επειδή και οι δύο επιμέρους προτάσεις είναι ψευδείς, αφού το x δεν είναι μεγαλύτερο από το y και το y δεν είναι μικρότερο από το 3. Επειδή οι δύο προτάσεις συνδέονται με το &&, αρκούσε μόνο μία από τις δύο επιμέρους προτάσεις να είναι ψευδής έτσι ώστε η συνολική πρόταση να χαρακτηριστεί και αυτή ψευδής.
Από την άλλη, η δεύτερη πρόταση είναι αληθής, διότι αρκεί μία από τις δύο επιμέρους προτάσεις να είναι αληθής για να χαρακτηριστεί η πρόταση ως αληθής. Αυτό συμβαίνει επειδή η $(y < 3)$ είναι ψευδής, αλλά με την άρνηση που είναι μπροστά της, καθίσταται αληθής, με συνέπεια ολόκληρη η πρόταση να είναι αληθής.

Λύση άσκησης αυτοαξιολόγησης 2.3

Η εντολή

```
>> any([x ~= x + 1, x >= 0, x == x])
```

επιστρέφει σίγουρα 1, αφού οι λογικές προτάσεις $x \sim= x + 1$ και $x == x$ είναι αληθείς για κάθε πραγματικό αριθμό x, άρα και για το $x = sin(exp(1))$.
Η εντολή

```
>> all([x ~= x + 1, x >= 0, x == x])
```

από την άλλη, θα επιστρέψει 1, μόνο αν και η πρόταση $x >= 0$ είναι αληθής, δηλαδή μόνο αν το x είναι μεγαλύτερο ή ίσο με το μηδέν. Επειδή η τιμή της μεταβλητής x ισούται με 0.4108, η πρόταση $x >= 0$ αληθεύει και, επομένως, η εντολή με το all επιστρέφει το ένα.
Τα παραπάνω επιβεβαιώνονται και με τη βοήθεια του Matlab, όπως καταδεικνύεται στη συνέχεια.

```
>> x=sin(exp(1))
x =
    0.4108

>> any([x ~= x + 1, abs(x) >= 0, x == x])
ans =
  logical
   1

>> all([x ~= x + 1, abs(x) >= 0, x == x])
ans =
  logical
   1
```

Λύση άσκησης αυτοαξιολόγησης 2.4

Με τις εντολές

```
>> x=5:0.001:10;
>> y=cos(x).*sin(x).*exp(-x/2);
```

ορίζουμε ένα διάνυσμα x με όλες τις τιμές από 5 ως 10 με βήμα 0.001 και τις τιμές της συνεχούς συνάρτησης $cos(x) \cdot sin(x) \cdot exp(-x/2)$ πάνω από αυτά τα x. Στη συνέχεια, παρατηρώντας ότι για $x = 5$, η $cos(x) \cdot sin(x) \cdot exp(-x/2)$ ισούται με -0.0223, αρκεί να εκτελέσουμε την εντολή

```
>> position=find(y>0,1)
position =
        1285
```

για να επιστραφεί η πρώτη θέση στην οποία η y λαμβάνει για πρώτη φορά θετική τιμή. Επομένως, η συνάρτηση $cos(x) \cdot sin(x) \cdot exp(-x/2)$ αλλάζει πρόσημο μεταξύ του $x(1284)$ και του $x(1285)$, τις τιμές των οποίων λαμβάνουμε με την εντολή

```
>> x(1284:1285)
ans =
    6.2830    6.2840
```

Επομένως, σύμφωνα με το θεώρημα Bolzano, υπάρχει τουλάχιστον μία ρίζα της συνάρτησης $cos(x) \cdot sin(x) \cdot exp(-x/2)$ στο διάστημα $[6.2830, 6.2840]$. Στην πραγματικότητα, στο διάστημα αυτό υπάρχει μοναδική ρίζα, την τιμή της οποίας έχουμε προσδιορίσει σε ένα διάστημα μήκους 0.001.
Επομένως, η πρώτη ρίζα της εξίσωσης $cos(x) \cdot sin(x) \cdot exp(-x/2)$ που είναι μεγαλύτερη ή ίση από το 5 και μικρότερη ή ίση από το 10 ανήκει στο διάστημα $[6.2830, 6.2840]$.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ξενόγλωσση

Givant, S. and Halmos, P. (2008). *Introduction to Boolean Algebras*. Undergraduate Texts in Mathematics. Springer, New York.

Havil, J. (2003). *Gamma: Exploring Euler's Constant*. Princeton University Press.

Zohuri, B. and Moghaddam, M. (2017). What Is Boolean Logic and How It Works. In: pp. 183–198.

ΚΕΦΑΛΑΙΟ 3

ΕΛΕΓΧΟΙ ΡΟΗΣ

Σύνοψη:

Σε αυτό το κεφάλαιο παρουσιάζονται οι εντολές με τις οποίες μπορούν να υλοποιηθούν δομές ελέγχου της ροής ενός προγράμματος. Πιο συγκεκριμένα, παρουσιάζεται η χρήση και τα στοιχεία διαφορετικών δομών ελέγχου ροής (if, if-else, if-elseif-else, switch), καθώς και οι διαφορές τους. Ιδιαίτερη αναφορά γίνεται και στη χρήση εμφωλευμένων δομών ελέγχου ροής για την επίλυση πιο σύνθετων προβλημάτων.

Προαπαιτούμενη γνώση: Τα κεφάλαια 1 και 2 του παρόντος συγγράμματος.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του κεφαλαίου αυτού, θα γνωρίζετε:

- τον τρόπο με τον οποίο συντάσσονται οι βασικές εντολές ελέγχου ροής,
- τις διαφορές μεταξύ των διαφορετικών εντολών ελέγχου ροής,
- τη χρήση εμφωλευμένων δομών ελέγχου ροής.

Γλωσσάριο επιστημονικών όρων

- `if-elseif-else`
- `switch`
- Δομή ελέγχου ροής
- Εμφωλευμένη δομή
- Χειρισμός λαθών, `error`

3.1 Εισαγωγή

Τα περιεχόμενα ενός προγράμματος εκτελούνται σειριακά, δηλαδή ξεκινώντας από την πρώτη εντολή, που βρίσκεται στην αρχή του κώδικά μας, και συνεχίζοντας προς τα κάτω. Έτσι, υλοποιούνται με τη σειρά όλες οι εκτελέσιμες εντολές μέχρι να τερματιστεί ο κώδικας. Ωστόσο, υπάρχουν περιπτώσεις όπου χρειάζεται κάποιες εντολές να παραληφθούν ή να επιλέγεται ένα συγκεκριμένο υποσύνολο εντολών από ένα δεδομένο πλήθος δυνατών επιλογών. Αυτό συνήθως προκύπτει έπειτα από έναν αριθμητικό ή λογικό έλεγχο κάποιας τιμής, που θα καθορίσει το πώς πρέπει να συνεχιστεί η εκτέλεση των εντολών. Για παράδειγμα, αν ο έλεγχος μιας μεταβλητής δείξει ότι η τιμή της είναι μηδέν, τότε θα πρέπει να αποτραπούν υπολογισμοί που θα χρησιμοποιούν τη μεταβλητή αυτή ως παρονομαστή σε αριθμητικές πράξεις.

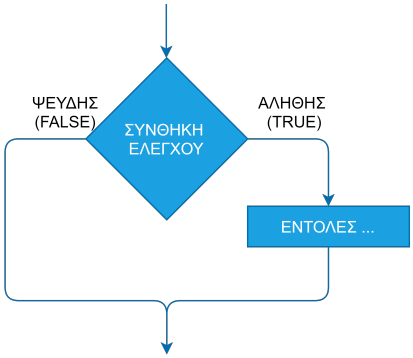
Η ανάπτυξη σύνθετων προγραμμάτων απαιτεί συχνά τη χρήση λογικών προτάσεων και συναρτήσεων για την πραγματοποίηση διάφορων ελέγχων και την επιλογή των εντολών που θα εκτελεστούν. Θα δούμε, λοιπόν, τις εντολές του λογισμικού Matlab που πραγματοποιούν αυτούς τους ελέγχους και μας επιτρέπουν να επιλέγουμε τις κατάλληλες εντολές, ανάλογα με τις συνθήκες.

3.2 Η εντολή if

Όπως αναφέρθηκε, υπάρχουν περιπτώσεις όπου θέλουμε να εκτελείται μια ομάδα εντολών, μόνο αν αληθεύει κάποια συνθήκη. Σε αυτή την περίπτωση, χρησιμοποιείται η εντολή `if` για την υλοποίηση μιας δομής απλής επιλογής, όπως φαίνεται στο Σχήμα 3.1. Η γενική μορφή με την οποία συντάσσεται η εντολή `if` εμφανίζεται στον Κώδικα 3.1.

```
1  if Συνθήκη
2
3          ... Εντολές ...
4
5  end
```

Κώδικας 3.1: Σύνταξη απλής `if`.



Σχήμα 3.1: Η εντολή `if`.

Ας δούμε πώς λειτουργεί η εντολή `if`. Αρχικά ελέγχεται η συνθήκη που βρίσκεται στο `if`. Ο έλεγχος αυτός έχει δύο δυνατές λογικές εξόδους: είτε η συνθήκη θα είναι αληθής είτε θα είναι ψευδής. Αν η συνθήκη είναι αληθής, εκτελούνται οι εντολές που βρίσκονται κάτω από τον έλεγχο, ανάμεσα στο `if` και το `end`, και, στη συνέχεια, το πρόγραμμα προχωράει στις διαδικασίες που βρίσκονται αμέσως μετά το `end`. Σε αντίθετη περίπτωση, οι εντολές εντός της δομής επιλογής `if` *δεν* εκτελούνται και το πρόγραμμα συνεχίζει με τις

διαδικασίες που βρίσκονται αμέσως μετά το `end`.

Παράδειγμα 3.1

Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη να εισάγει έναν πραγματικό αριθμό x και θα εκτυπώνει την απόλυτη τιμή του, δηλαδή το $|x|$, χωρίς τη χρήση της εντολής abs.

Λύση Παραδείγματος 3.1

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μία μεταβλητή στην οποία θα αποθηκεύεται η τιμή που εισήγαγε ο χρήστης (π.χ. x), καθώς και μία μεταβλητή στην οποία θα αποθηκεύεται η απόλυτη τιμή (π.χ. xAbs).

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να ελέγξει τη μεταβλητή x και, αν είναι μικρότερη του μηδενός, να της αλλάξει πρόσημο.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %ABSOLUT_X Find the absolut value of x
2  % Initialization
3  clear all
4  x = input('Give a real number: ');
5
6  % Main code
7  xAbs = x;
8  if x<0
9    xAbs = -x;
10 end
11 disp(['The absolute value of ',num2str(x),' is ',num2str(xAbs)])
```

Ο έλεγχος του αν η μεταβλητή x είναι μικρότερη του μηδενός πραγματοποιείται στη γραμμή 8 του παραπάνω κώδικα με τη βοήθεια της συνθήκης `x<0`. Αν ικανοποιείται η συνθήκη αυτή, τότε εκτελείται η εντολή της γραμμής 9, η οποία αλλάζει την τιμή της μεταβλητής xAbs με το -x.
Σημειώνεται ότι στις δύο πρώτες γραμμές εμφανίζεται η βοήθεια για το συγκεκριμένο αρχείο script, ενώ στη γραμμή τρία εμφανίζεται η εντολή `clear all`, η οποία διαγράφει οποιαδήποτε μεταβλητή έχει δημιουργηθεί νωρίτερα στο Matlab. Τα δύο προαναφερθέντα στοιχεία δεν αποτελούν απαραίτητα στοιχεία για την εκτέλεση του προγράμματος αλλά αποτελούν καλές πρακτικές προγραμματισμού, οι οποίες ακολουθούνται στο υπόλοιπο μέρος του συγγράμματος.

Παρατήρηση 3.1

Το όνομα κάθε μεταβλητής μπορείτε να το επιλέγετε τυχαία, καλό είναι όμως να περιγράφει συνοπτικά το περιεχόμενο της μεταβλητής αυτής, ώστε να διευκολύνεται η ανάγνωση του κώδικα!
Έτσι, στο παραπάνω παράδειγμα, τη μεταβλητή, στην οποία αποθηκεύεται η απόλυτη τιμή του x, την ονομάσαμε xAbs.

Παράδειγμα 3.2

Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη να εισάγει τρεις αριθμούς και θα εκτυπώνει τον μεγαλύτερο, χωρίς τη χρήση της ενσωματωμένης εντολής `max` του Matlab.

Λύση Παραδείγματος 3.2

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε τρεις μεταβλητές όπου θα αποθηκευτούν οι αριθμοί που θα δώσει ο χρήστης (π.χ. x1, x2, x3), καθώς και μία μεταβλητή στην οποία θα υποθηκευτεί η μέγιστη τιμή (π.χ. xMax).

Κύριος κώδικας: Αρχικά, κάνουμε τη θεώρηση ότι η πρώτη μεταβλητή που εισήγαγε ο χρήστης έχει τη μεγαλύτερη τιμή. Το πρόγραμμα θα πρέπει να συγκρίνει τη δεύτερη μεταβλητή με τις άλλες δύο και, αν είναι μεγαλύτερη, να την καταχωρίσει στη xMax. Στη συνέχεια, θα πρέπει να επαναλάβει τη διαδικασία και για την τρίτη μεταβλητή. Στο τέλος, πρέπει να εμφανίσει στην οθόνη την τιμή της xMax.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %MAXIMUM Find the maximum of three numbers
2  % Initialization
3   clear all
4   x1 = input('Give a number: ');
5   x2 = input('Give another number: ');
6   x3 = input('Give a third number: ');
7
8   % Main code
9   xMax = x1;
10
11   if x2>x1 && x2>x3
12     xMax = x2;
13   end
14
15   if x3>x1 && x3>x2
16     xMax = x3;
17   end
18
19   disp(['The maximum number is ',num2str(xMax)])
```

Παρατήρηση 3.2

Κάθε πρόβλημα μπορεί να έχει πάνω από μία σωστές λύσεις!
Συνήθως, καλύτερος τρόπος επίλυσης είναι αυτός που οδηγεί στον συντομότερο κώδικα.

Άσκηση αυτοαξιολόγησης 3.1

Δοκιμάστε να φτιάξετε έναν εναλλακτικό αλγόριθμο επίλυσης του Παραδείγματος 3.2, όπου οι εντολές `if` θα συνοδεύονται από μία απλή συνθήκη και όχι από μία διπλή, όπως στη λύση που παρουσιάσαμε.
Υπόδειξη: Χρησιμοποιήστε στους ελέγχους τη μεταβλητή xMax.

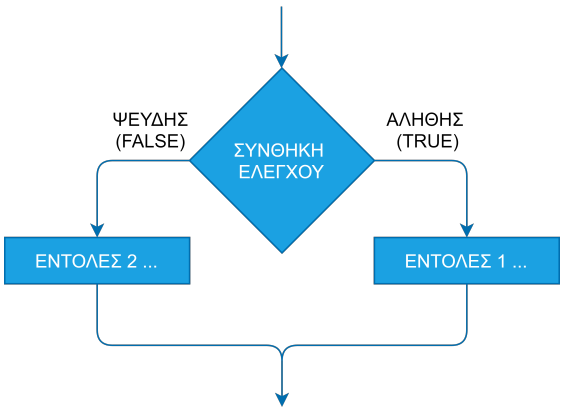
3.3 Η εντολή if-else

Στις περιπτώσεις όπου θέλουμε η επιλογή να αφορά δύο σύνολα εντολών, χρησιμοποιείται μια δομή σύνθετης επιλογής η οποία υλοποιείται από την εντολή `if-else`, όπως φαίνεται στο Σχήμα 3.2.

Η γενική μορφή με την οποία συντάσσεται η `if-else` παρουσιάζεται στον Κώδικα 3.2

```
1  if Συνθήκη
2
3          ... Εντολές 1 ...
4
5  else
6
7          ... Εντολές 2 ...
8
9  end
```

Κώδικας 3.2: Σύνταξη σύνθετης `if-else`.



Σχήμα 3.2: Η εντολή `if-else`.

Κατά την εκτέλεση του κώδικα, αρχικά ελέγχεται η συνθήκη που συνοδεύει την εντολή `if`. Αν αυτή επαληθεύεται, εκτελούνται οι Εντολές 1, οι οποίες βρίσκονται κάτω από το πρώτο σκέλος της δομής `if-else`. Αν δεν επαληθεύεται, τότε εκτελείται το τμήμα κώδικα που βρίσκεται μετά την εντολή `else`, δηλαδή οι Εντολές 2. Μια σημαντική παρατήρηση είναι ότι η εντολή `else` δεν ακολουθείται από λογική συνθήκη, αλλά περιλαμβάνει όλες τις περιπτώσεις που δεν επαληθεύουν τη συνθήκη στο `if`.

Παράδειγμα 3.3

Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη να εισάγει έναν Αριθμό Μητρώου (Α.Μ.) φοίτησης και θα πληροφορεί τον χρήστη αν ο αριθμός είναι έγκυρος ή όχι. Οι έγκυροι Α.Μ. πρέπει να βρίσκονται μέσα στο διάστημα από 1000 μέχρι 9999.

Λύση Παραδείγματος 3.3

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μόνο μία μεταβλητή στην οποία θα αποθηκεύεται η τιμή που εισήγαγε ο χρήστης (π.χ. ΑΜ).

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να ελέγχει τη μεταβλητή ΑΜ και, αν δεν είναι μέσα

στο προβλεπόμενο εύρος, να εκτυπώνει μήνυμα μη-εγκυρότητας, ενώ, σε αντίθετη περίπτωση, να εκτυπώνει ότι είναι έγκυρο Α.Μ.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %AM_CHECK Check if the A.M. is valid
2  % Initialization
3   clear all
4   AM = input('Give a valid A.M.: ');
5
6   % Main code
7   if AM<1000 || AM>9999
8     disp('Error: The A.M. you entered is not valid!')
9   else
10    error(['The A.M. is valid.'])
11  end
```

Παρατήρηση 3.3

Η εντολή `error`, η οποία εμφανίζεται στην γραμμή 7 του παραπάνω κώδικα, χρησιμοποιείται όταν θέλουμε να διακόψουμε άμεσα την εκτέλεση του προγράμματος με κατάλληλο συνοδευτικό μήνυμα. Η σύνταξή της είναι παρόμοια με αυτή της εντολής `disp` με τη διαφορά ότι τυπώνει το κείμενο με κόκκινο χρώμα, όπως κάνει και το Matlab όταν παρουσιάζεται ένα σφάλμα στον κώδικα.

Παράδειγμα 3.4

Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη να εισάγει έναν θετικό ακέραιο αριθμό n. Αν ο αριθμός n δεν είναι θετικός ακέραιος, θα εκτυπώνει μήνυμα λάθους, αλλιώς θα εκτυπώνει την τιμή του 2 υψωμένη εις τη n-οστή δύναμη.

Λύση Παραδείγματος 3.4

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μόνο μία μεταβλητή στην οποία θα αποθηκεύεται η τιμή που εισήγαγε ο χρήστης (π.χ. n).

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να ελέγχει αν η μεταβλητή n είναι θετικός ακέραιος. Ένας τρόπος για να εξετάσουμε αν μια μεταβλητή είναι θετικός ακέραιος, είναι να συνδυάσουμε δύο ελέγχους μέσα στη λογική συνθήκη:
- έναν για το αν το n είναι θετικό ή αρνητικό,
- και έναν για το αν το n είναι ακέραιο ή όχι.

Ο τρόπος για να βρούμε αν ένας αριθμός είναι ακέραιος ή όχι, είναι να τον στρογγυλοποιήσουμε με την εντολή round και να ελέγξουμε αν παραμένει ο ίδιος μετά τη στρογγυλοποίησή του. Για παράδειγμα, ο αριθμός 5 δεν αλλάζει όταν τον στρογγυλοποιήσουμε, άρα είναι ακέραιος. Αντίθετα, ο αριθμός 5.2 αλλάζει, δηλαδή γίνεται 5 όταν τον στρογγυλοποιήσουμε, άρα *δεν* είναι ακέραιος.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %POS_INT Find if the value is a positive integer
2  % Initialization
```

```
3   clear all
4   n = input('Give a positive integer: ');
5
6   % Main code
7   if n<0 || round(n)~=n
8     error('The number you entered is not a positive integer!')
9   else
10    disp(['The ',num2str(n),'-th power of 2 is ',num2str(2^n)])
11  end
```

Παρατήρηση 3.4

Έλεγχοι, όπως αυτός που κάναμε στην προηγούμενη άσκηση, θα είναι πολύ χρήσιμοι για τον ορισμό πινάκων που θα δούμε σε επόμενα κεφάλαια, καθώς οι διαστάσεις των πινάκων πρέπει να είναι απαραίτητα θετικοί ακέραιοι.
Συνήθως, οι έλεγχοι αυτοί εισάγονται στο αρχικό κομμάτι του κώδικα, μετά την αρχικοποίηση των μεταβλητών και πριν το κύριο μέρος.

Άσκηση αυτοαξιολόγησης 3.2

Δοκιμάστε να φτιάξετε έναν εναλλακτικό αλγόριθμο επίλυσης του Παραδείγματος 3.4, όπου η εντολή `if` θα συνοδεύεται από μία συνθήκη με && και όχι από μία συνθήκη με ||, όπως στη λύση που παρουσιάσαμε.

3.4 Η εντολή if-elseif-else

Στην περίπτωση που έχουμε περισσότερες από μία συνθήκες τις οποίες θέλουμε να ελέγξουμε, μπορούμε να χρησιμοποιήσουμε μία δομή πολλαπλής επιλογής `if-elseif-else`, όπως φαίνεται στο Σχήμα 3.3. Με τη δομή αυτή υπάρχει η δυνατότητα διαδοχικών ελέγχων σε μια σειρά λογικών συνθηκών, έως ότου βρεθεί κάποια που να είναι αληθής. Όταν βρεθεί μία αληθής συνθήκη, τότε εκτελούνται οι εντολές που βρίσκονται κάτω από αυτή και, στη συνέχεια, η εκτέλεση συνεχίζεται με τις διαδικασίες που βρίσκονται αμέσως μετά το `end`. Για να υλοποιήσουμε τη δομή πολλαπλής επιλογής με την εντολή `if-elseif-else`, χρησιμοποιούμε τη γενική της μορφή, όπως φαίνεται στον Κώδικα 3.3 και στο Σχήμα 3.3.

```
1   if Συνθήκη 1
2
3         ... Εντολές 1 ...
4
5   elseif Συνθήκη 2
6
7         ... Εντολές 2 ...
8       .
9       .
10      .
```

```
11
12    elseif Συνθήκη   n-1
13
14           ... Εντολές n-1 ...
15
16    else
17
18           ... Εντολές n ...
19
20    end
```

Κώδικας 3.3: Σύνταξη πολλαπλής `if-elseif-else`.

Η εντολή `if-elseif-else` εκτελείται σειριακά:

- Αρχικά ελέγχεται η συνθήκη 1. Στην περίπτωση που είναι αληθής, εκτελούνται οι εντολές 1 και η δομή επιλογής τερματίζεται, οδηγώντας στο τμήμα του προγράμματος μετά το `end` και τον κώδικα που ακολουθεί.
- Αν η συνθήκη 1 είναι ψευδής, τότε οδηγούμαστε στον έλεγχο της αμέσως επόμενης συνθήκης, δηλαδή της συνθήκης 2. Αν αυτή είναι αληθής, τότε με την ίδια λογική εκτελούνται οι εντολές 2 και έπειτα η δομή τερματίζεται, οδηγώντας στο τμήμα του προγράμματος μετά το `end`.
- Στην περίπτωση όπου *και* η συνθήκη 2 είναι ψευδής, ελέγχεται η αμέσως επόμενη συνθήκη κ.ο.κ. Η διαδικασία συνεχίζεται μέχρις ότου βρεθεί κάποια συνθήκη η οποία είναι αληθής, πράγμα που θα έχει ως αποτέλεσμα και την εκτέλεση του αντίστοιχου κώδικα.
- Αν αυτό δεν συμβεί, δηλαδή καμία από τις συνθήκες 1 έως n-1 δεν είναι αληθής, τότε οδηγούμαστε στην εντολή `else`. Σε αυτήν την περίπτωση, θα εκτελεστούν οι εντολές n και, στη συνέχεια, η δομή επιλογής τερματίζεται, οδηγώντας στο τμήμα του προγράμματος μετά το `end`.

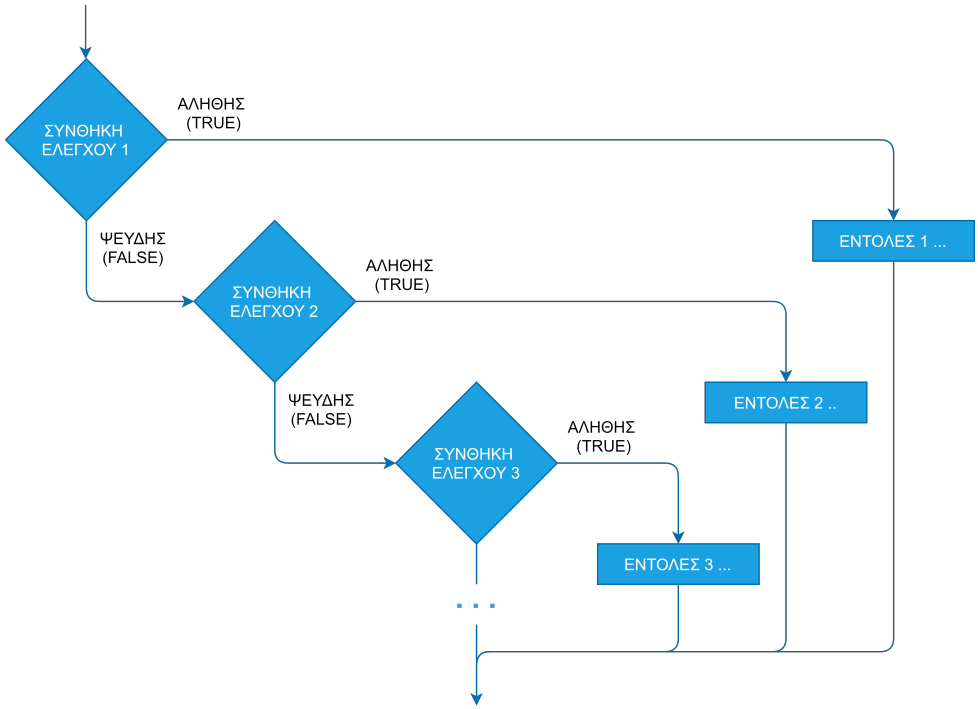
Παράδειγμα 3.5

Στην Αριθμητική Ανάλυση υπάρχουν διάφορες μέθοδοι για την εύρεση της ρίζας μίας εξίσωσης με πιο γνωστές τη μέθοδο της διχοτόμου, τη μέθοδο των διαδοχικών προσεγγίσεων και τη μέθοδο Newton-Raphson (Hoffman, 2001). Σε πολλά προγράμματα Αριθμητικής Ανάλυσης ο χρήστης έχει τη δυνατότητα να επιλέξει ποια από τις μεθόδους αυτές θέλει να χρησιμοποιήσει για την επίλυση μίας εξίσωσης.
Θεωρήστε ότι κατασκευάζετε εσείς ένα τέτοιο πρόγραμμα, το οποίο πρέπει να ζητάει από τον χρήστη να επιλέξει ποια μέθοδο θέλει να χρησιμοποιήσει. Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη να εισάγει έναν αριθμό, πληροφορώντας τον ότι το 1 αντιστοιχεί στη μέθοδο της διχοτόμου, το 2 στη μέθοδο των διαδοχικών προσεγγίσεων και το 3 στη μέθοδο Newton-Raphson.
Ο κώδικας θα ελέγχει την επιλογή του χρήστη και θα τον πληροφορεί ποια μέθοδο έχει επιλέξει. Στην περίπτωση που ο χρήστης έχει εισάγει κάποιον αριθμό που δεν ανήκει στις παραπάνω επιλογές, θα εμφανίζεται κατάλληλο μήνυμα σφάλματος.

Λύση Παραδείγματος 3.5

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μόνο μία μεταβλητή στην οποία θα αποθηκεύεται η τιμή που εισήγαγε ο χρήστης (π.χ. method).

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να ελέγχει τη μεταβλητή method και να εκτυπώνει το κατάλληλο μήνυμα.



Σχήμα 3.3: Η εντολή `if-elseif-else`.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %METHOD_ROOT Choose the method to find a root
2  % Initialization
3  clear all
4  method = input('Choose a method: 1 - Dihotomos, 2 - Proseggiseis, 3
       - Newton-Raphson ');
5
6  % Main code
7  if method == 1
8    disp('Solution method: Dihotomos')
9  elseif method == 2
10   disp('Solution method: Diadohikes Proseggiseis')
11 elseif method == 3
12   disp('Solution method: Newton-Raphson')
13 else
14   error('The input number is not valid!')
15 end
```

Παρατήρηση 3.5

Η χρήση της εντολής `else` στον τελευταίο έλεγχο μίας δομής `if-elseif-else` δεν είναι υποχρεωτική, μας βοηθάει όμως να συμπεριλάβουμε στον έλεγχο όλες τις εναπομείνασες περιπτώσεις, χωρίς να τις ορίσουμε με ξεχωριστές συνθήκες. Θα μπορούσαμε, δηλαδή, στο προηγούμενο Παράδειγμα να αντικαταστήσουμε την εντολή `else` με τη συνθήκη

```
1 elseif method~=1 && method~=2 && method~=3
```

και το αποτέλεσμα θα ήταν ακριβώς το ίδιο!
Ωστόσο, η ολοκλήρωση μίας δομής `if-elseif-else` με την εντολή `else` είναι πολύ πιο σύντομη και ασφαλής, αφού δεν μπορεί να ξεχάσουμε κάποια περίπτωση και για αυτό την προτιμούμε.

Παράδειγμα 3.6

Ας προσπαθήσουμε να προσθέσουμε κάποια χαρακτηριστικά στη λύση του Παραδείγματος 3.5. Στο Παράδειγμα αυτό κατασκευάσαμε μία δομή επιλογής που πληροφορεί τον χρήστη για τη μέθοδο που έχει επιλέξει για να βρει τη ρίζα μίας εξίσωσης, μεταξύ τριών επιλογών οι οποίες είναι η μέθοδος της διχοτόμου, η μέθοδος των διαδοχικών προσεγγίσεων και η μέθοδος Newton-Raphson. Ωστόσο, καθεμία από αυτές τις μεθόδους απαιτεί διαφορετικά δεδομένα από τον χρήστη, προκειμένου να λειτουργήσει. Συγκεκριμένα:

- η μέθοδος της διχοτόμου απαιτεί το αρχικό και τελικό σημείο ενός διαστήματος, μέσα στο οποίο εκτιμούμε ότι βρίσκεται η ρίζα,
- η μέθοδος διαδοχικών προσεγγίσεων απαιτεί ένα αρχικό σημείο εκκίνησης, καθώς και έναν συντελεστή, ο οποίος καθορίζει την ταχύτητα εύρεσης της ρίζας και
- τέλος, η μέθοδος Newton-Raphson απαιτεί μόνο ένα αρχικό σημείο, προκειμένου να υπολογίσει τη ρίζα μιας εξίσωσης.

Προσπαθήστε να μετατρέψετε τον κώδικα του Παραδείγματος 3.5, έτσι ώστε να ζητάει από τον χρήστη τα κατάλληλα δεδομένα, ανάλογα με τη μέθοδο που έχει επιλέξει.

Λύση Παραδείγματος 3.6

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε αρκετές μεταβλητές, καθώς είναι πολλές οι δυνατές επιλογές που θα έχει ο χρήστης. Συγκεκριμένα, θα χρειαστούμε μία μεταβλητή στην οποία θα αποθηκεύεται η τιμή που εισήγαγε ο χρήστης για την επιλογή της μεθόδου (το 1 αντιστοιχεί στη μέθοδο της διχοτόμου, το 2 στη μέθοδο των διαδοχικών προσεγγίσεων και το 3 στη μέθοδο Newton-Raphson), ενώ θα χρειαστούμε επιπλέον:

- δύο μεταβλητές για το αρχικό και το τελικό σημείο του διαστήματος, στη μέθοδο της διχοτόμου (x_ar_dix και x_tel_dix, αντίστοιχα),
- μία μεταβλητή για το σημείο εκκίνησης και μία για τον συντελεστή της μεθόδου, στην περίπτωση των διαδοχικών προσεγγίσεων (x_ar_dp και c_dp, αντίστοιχα),
- μία μεταβλητή για το σημείο εκκίνησης της μεθόδου Newton-Raphson (x_ar_NR).

Κύριος κώδικας: Μέσα σε κάθε συνθήκη του εξωτερικού ελέγχου, θα πρέπει να προστεθούν οι εντολές που θα ζητούν από τον χρήστη τα κατάλληλα δεδομένα για κάθε μέθοδο.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %METHOD_ROOT Choose the method to find a root
2  % Initialization
3  clear all
4  method = input('Choose a method: 1 - Dihotomos, 2 - Proseggiseis, 3
       - Newton-Raphson :');
5  x_ar_dix = 0; % Arxiko shmeio methodou dixotomou
6  x_tel_dix = 0; % Teliko shmeio methodou dixotomou
7  x_ar_dp = 0;  % Arxiko shmeio Methodou diadohikwn proseggisewn
8  c_dp = 0; % Syntelestis diadohikwn proseggisewn
9  x_ar_NR = 0;  % Arxiko shmeio Methodou N-R
10
11 % Main code
12 if method == 1
13   disp('Solution method: Dihotomos')
14   x_ar_dix = input('Starting point: ');
15   x_tel_dix = input('Ending point: ');
16 elseif method == 2
17   disp('Solution method: Diadohikes Proseggiseis')
18   x_ar_dp = input('Starting point: ');
19   c_dp = input('Coefficient: ');
20 elseif method == 3
21   disp('Solution method: Newton-Raphson')
22   x_ar_NR = input('Starting point: ');
23 else
24   error('The input number is not valid!')
25 end
```

Παρατήρηση 3.6

Παρατηρούμε ότι στο παραπάνω παράδειγμα χρειάστηκε να ορίσουμε και να αρχικοποιήσουμε πολλές μεταβλητές. Οι μεταβλητές αυτές παίρνουν τιμή από τον χρήστη μέσω των αντίστοιχων εντολών `input`, οπότε δημιουργείται η απορία ποιος είναι ο λόγος που τις αρχικοποιούμε.

Είναι γεγονός ότι, στη συγκεκριμένη περίπτωση, η αρχικοποίηση δεν παίζει κάποιον ρόλο, αφού η «εικονική» τιμή 0 χάνεται μόλις ο χρήστης δώσει τα δεδομένα. Με άλλα λόγια, ο κώδικας θα λειτουργούσε εξίσου καλά, αν έλειπαν οι γραμμές 3-7. Ωστόσο, είναι καλή προγραμματιστική πρακτική να αρχικοποιούνται όλες οι μεταβλητές στην αρχή του κώδικα, ώστε ο προγραμματιστής να έχει εποπτεία όλων των ονομάτων των μεταβλητών που έχει χρησιμοποιήσει, καθώς και των αρχικών τους τιμών.

Αυτό γίνεται ιδιαίτερα χρήσιμο όσο αυξάνεται η πολυπλοκότητα ενός κώδικα, καθώς αποτρέπει λάθη σύγχυσης λόγω επανάληψης κάποιου ονόματος μεταβλητής. Είναι, επίσης, πολύ βοηθητικό στην περίπτωση που θέλουμε να εμπλουτίσουμε κάποιον κώδικα με νέα στοιχεία και νέες λειτουργίες. Σε αυτή την περίπτωση, είναι πολύ χρήσιμη και η σύντομη περιγραφή υπό τη μορφή σχολίου της λειτουργίας που επιτελεί η κάθε μεταβλητή.

Παράδειγμα 3.7: Young *et al.* (2010)

Η πτήση ενός αεροσκάφους χαρακτηρίζεται υποηχητική, διηχητική, υπερηχητική ή υπερ-υπερηχητική, ανάλογα με την τιμή του αδιάστατου αριθμού Mach, ο οποίος εκφράζει τον λόγο της ταχύτητας του αεροσκάφους u προς την ταχύτητα του ήχου c, $M = u/c$.
Συγκεκριμένα:

- όταν $M \leq 0.8$, έχουμε πτήση στην υποηχητική περιοχή (Subsonic),
- όταν $0.8 < M \leq 1.2$, έχουμε πτήση στη διηχητική περιοχή (Transonic),
- όταν $1.2 < M \leq 5$, έχουμε πτήση στην υπερηχητική περιοχή (Supersonic),
- όταν $5 < M \leq 10$, έχουμε πτήση στην υπερ-υπερηχητική περιοχή (Hypersonic),
- όταν $10 < M \leq 25$, έχουμε πτήση στην υψηλή υπερ-υπερηχητική περιοχή (High Hypersonic),
- μεγαλύτερες ταχύτητες με $M > 25$ επιτυγχάνονται κατά την επανείσοδο διαστημικών οχημάτων στην ατμόσφαιρα (Re-entry speeds).

Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη την ταχύτητα του αεροσκάφους σε km/h και θα εκτυπώνει το είδος της πτήσης. Θεωρήστε ότι η ταχύτητα του ήχου είναι 1235 km/h.

Λύση Παραδείγματος 3.7

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε δύο μεταβλητές: μία στην οποία θα αποθηκεύεται η ταχύτητα του αεροσκάφους (π.χ. u) που θα δίνει ο χρήστης και μία που θα έχει αποθηκευμένη την ταχύτητα του ήχου (π.χ. c).

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να υπολογίζει τον αδιάστατο αριθμό Mach, να βρίσκει σε ποια περιοχή ανήκει και να εκτυπώνει το κατάλληλο μήνυμα.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%MACH Find the Mach number
% Initialization
clear all
u = input('Aircraft speed [in km/h]:');
c = 1235;

% Main code
M=u/c;
if M > 0 && M <= 0.8
  disp('Subsonic')
elseif M > 0.8 && M <= 1.2
  disp('Transonic')
elseif M > 1.2 && M <= 5
  disp('Supersonic')
elseif M > 5 && M <= 10
  disp('Hypersonic')
elseif M > 10 && M <= 25
  disp('High Hypersonic')
```

```
19    elseif M > 25
20      disp('Re-entry speeds')
21    else
22      error('The velocity must be positive!')
23    end
```

Σημειώνεται ότι η δομή ελέγχου ολοκληρώνεται με τη χρήση της εντολής `error` ύστερα από το `else` για τον χειρισμό περιπτώσεων που ο χρήστης εισάγει μια αρνητική ταχύτητα.

3.5 Εμφωλευμένες επιλογές

Οι εφαρμογές των εντολών ελέγχου που παρουσιάστηκαν στις προηγούμενες ενότητες αφορούσαν απλές περιπτώσεις προβλημάτων, όπου ανάλογα με την τιμή μίας ή περισσότερων μεταβλητών εκτελούνταν συγκεκριμένα τμήματα κώδικα. Σε πολλά προβλήματα, όμως, η επιλογή αυτή είναι πολύ πιο σύνθετη και εξαρτάται από άλλους ελέγχους οι οποίοι έχουν προηγηθεί. Ας δούμε ένα θεωρητικό παράδειγμα για να κατανοήσουμε αυτούς τους σύνθετους ελέγχους.

Υποθέτουμε ότι κατασκευάζουμε ένα πρόγραμμα ελέγχου ποιότητας για ένα κατάστημα φρούτων. Το ζητούμενο σε αυτή την περίπτωση είναι ο χρήστης να εισάγει πληροφορίες για το φρούτο που έχει αγοράσει και, συγκεκριμένα, το χρώμα του. Ωστόσο, οι επιλογές που θα πρέπει να δίνονται στον χρήστη πρέπει να διαφέρουν ανάλογα με το φρούτο. Για παράδειγμα, αν ο έλεγχος αφορά μήλα, τότε οι δυνατές επιλογές πρέπει να είναι «κόκκινο» ή «πράσινο», ενώ αν ο έλεγχος αφορά μπανάνες, τότε οι δυνατές επιλογές πρέπει να είναι «κίτρινη» ή «πράσινη». Είναι σαφές ότι μία δυνατότητα επιλογής κόκκινου χρώματος για τις μπανάνες, δεν θα είχε νόημα. Σε αυτή την περίπτωση, λοιπόν, θα πρέπει, μετά την επιλογή φρούτου από τον χρήστη, να υπάρχει μία εσωτερική δομή ελέγχου που θα αφορά αποκλειστικά το συγκεκριμένο φρούτο.

Η περίπτωση αυτή, κατά την οποία μια δομή επιλογής βρίσκεται στο εσωτερικό μιας άλλης ονομάζεται *εμφωλευμένη*. Είναι προφανές ότι μια εμφωλευμένη δομή μπορεί να περιέχει οποιαδήποτε από τις εντολές που αναφέρθηκαν στις προηγούμενες ενότητες ή συνδυασμό τους. Ας δούμε ένα παράδειγμα σύνταξης εμφωλευμένης δομής: αν η εξωτερική δομή ελέγχου αποτελείται από τις συνθήκες A(1) και A(2), ενώ εσωτερικά υπάρχουν δύο εμφωλευμένες δομές με συνθήκες B(1), B(2) και C(1), C(2), αντίστοιχα, ο κώδικας θα έχει τη μορφή του Κώδικα 3.4.

```
1   if Συνθήκη A(1)
2
3       if Συνθήκη B(1)
4
5           ... Εντολές 1 ...
6
7       elseif Συνθήκη B(2)
8
9           ... Εντολές 2 ...
10
11      end
12
13  elseif Συνθήκη A(2)
14
```

```
15      if Συνθήκη  C(1)
16
17          ... Εντολές 3 ...
18
19      elseif Συνθήκη  C(2)
20
21          ... Εντολές 4 ...
22
23      end
24
25  end
```

Κώδικας 3.4: Σύνταξη εμφωλευμένης `if`.

Ένα πράγμα στο οποίο θα πρέπει να δίνεται ιδιαίτερη προσοχή είναι ο τερματισμός των δομών επιλογής. Είναι βασικό μια εμφωλευμένη δομή να τερματίζεται σε σημείο που βρίσκεται εντός της εξωτερικής δομής επιλογής και να μην διασταυρώνεται με άλλες εμφωλευμένες δομές. Με άλλα λόγια, κάθε εμφωλευμένη δομή πρέπει να τερματίζεται προτού ξεκινήσει η επόμενη, ενώ όλες οι εμφωλευμένες δομές επιλογής πρέπει να τερματίζονται εντός της συνολικής δομής ελέγχου.

Για να μπορούμε να ελέγξουμε πιο εύκολα ότι τηρείται αυτός ο κανόνας σε περιπτώσεις με συνδυασμό πολλών ελέγχων, είναι χρήσιμο να δημιουργούμε μία «κλιμακωτή» μορφή στη γραφή του κώδικά μας, δηλαδή να αφήνουμε μεγαλύτερο περιθώριο στο αριστερό μέρος του κειμενογράφου για κάθε ομάδα εντολών που βρίσκεται στο εσωτερικό μίας δομής. Αυτός είναι ο λόγος για τον οποίο στο παραπάνω παράδειγμα σύνταξης όλες οι εντολές της δομής Β βρίσκονται «πιο μέσα» σε σχέση με τις αντίστοιχες της δομής Α. Το ίδιο συμβαίνει και με τις εντολές της δομής ελέγχου Γ. Παρατηρείστε, επίσης, ότι οι εντολές των δομών Β και Γ βρίσκονται στο ίδιο επίπεδο, καθώς είναι και οι δύο εμφωλευμένες δομές εντός της Α.

Πρέπει να σημειωθεί ότι ο τρόπος αυτός σύνταξης του κώδικα δεν είναι υποχρεωτικός, αφού το πρόγραμμα θα μπορούσε να εκτελεστεί χωρίς σφάλμα ακόμα κι αν όλες οι εντολές του ξεκινούσαν από την πρώτη στήλη, χωρίς περιθώρια από το αριστερό άκρο του κειμενογράφου. Ωστόσο, είναι πολύ χρήσιμο να τηρούμε τη γραφή με τα κατάλληλα περιθώρια, καθώς μας επιτρέπει να εντοπίσουμε με μια ματιά ποιες δομές βρίσκονται εμφωλευμένες μέσα σε άλλες και να ελέγξουμε αν έχουν τερματιστεί στο σωστό σημείο. Για τον λόγο αυτό, ο κειμενογράφος του Matlab αφήνει αυτόματα μεγαλύτερο περιθώριο στις εντολές που ακολουθούν στην επόμενη γραμμή από μία `if`, elseif ή else, ενώ επαναφέρει αυτόματα το αρχικό περιθώριο μόλις πληκτρολογήσουμε την εντολή τερματισμού `end`. Με τον τρόπο αυτό διασφαλίζει την κλιμακωτή διάταξη στις εμφωλευμένες δομές, χωρίς να απαιτείται παρέμβαση του προγραμματιστή.

Παράδειγμα 3.8

Δημιουργήστε ένα αρχείο script που να υπολογίζει την επιφάνεια ή τον όγκο ενός τρισδιάστατου σχήματος. Ο χρήστης θα επιλέγει το είδος του σχήματος, το οποίο μπορεί να είναι κύβος ή κύλινδρος, το μέγεθος που θέλει να υπολογιστεί (επιφάνεια ή όγκος) και θα εκτυπώνεται το αντίστοιχο αποτέλεσμα. Οι τύποι που χρειάζονται για τους υπολογισμούς δίνονται παρακάτω:

- Εμβαδό κύβου πλευράς L: $E = 6L^2$
- Όγκος κύβου πλευράς L: $V = L^3$
- Εμβαδό κυλίνδρου ακτίνας R και ύψους H: $E = 2\pi RH + 2\pi R^2$
- Όγκος κυλίνδρου ακτίνας R και ύψους H: $V = 2\pi R^2 H$

Λύση Παραδείγματος 3.8

Αρχικοποίηση μεταβλητών: Το πρόβλημα που έχουμε να λύσουμε περιλαμβάνει αρκετά δεδομένα. Συγκεκριμένα, θα πρέπει να δημιουργήσουμε τις μεταβλητές για τις επιλογές (α) κύβος ή κύλινδρος (π.χ. shape) και (β) υπολογισμός εμβαδού ή όγκου (π.χ. Calc). Επίσης,

- αν ο χρήστης επιλέξει τον κύβο, θα χρειαστεί να δώσει το μήκος της πλευράς (π.χ. L),
- αν επιλέξει κύλινδρο, τότε θα χρειαστεί να δώσει την ακτίνα (π.χ. R) και το ύψος (π.χ. H).

Για τους λόγους που αναφέρθηκαν στην Παρατήρηση 3.6, θα αρχικοποιήσουμε όλες τις μεταβλητές στην αρχή του κώδικα.

Κύριος κώδικας: Αρχικά, θα πρέπει να ζητήσουμε από τον χρήστη να επιλέξει ποιο σχήμα επιθυμεί να υπολογιστεί. Στη συνέχεια, θα ζητάμε από τον χρήστη να επιλέξει αν θα υπολογιστεί το εμβαδόν ή ο όγκος. Ανάλογα με τις επιλογές, θα πρέπει να κατευθύνεται ο κώδικας στις κατάλληλες εντολές για εισαγωγή των δεδομένων και, στη συνέχεια, θα εκτυπώνεται το αποτέλεσμα, σύμφωνα με τον αντίστοιχο μαθηματικό τύπο.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %AREA_VOLUME Find area or the volume of 3D shapes
2  % Initialization
3   clear all
4   shape = 0;  % Type of 3D shape
5   Calc = 0; % Calculation of area or volume
6   L = 0; % Length of cube side
7   R = 0; % Radius of cylinder
8   H = 0; % Height of cylinder
9
10  % Main code
11  disp('Choose 3D shape:');
12  shape = input('1 - Cube, 2 - Cylinder: ');
13  disp('Choose calculation:');
14  Calc = input('1 - Area, 2 - Volume: ');
15
16  % Code for cube
17  if shape == 1
18      L = input('Length of the side of the cube: ');
19      if Calc == 1
20          disp(['Result: ', num2str(6*L^2)]) % Area
21      elseif Calc == 2
22          disp(['Result: ', num2str(L^3)]) % Volume
23      else
24          error('Wrong calculation choice') % Wrong calculation
25      end
26
27  % Code for cylinder
28  elseif shape == 2
29      R = input('Radius of cylinder: ');
30      H = input('Height of cylinder: ');
```

```
31      if Calc == 1
32          disp(['Result: ', num2str(2*pi*R*H + 2*pi*R^2)]) % Area
33      elseif Calc == 2
34          disp(['Results: ', num2str(pi*R^2*H)]) % Volume
35      else
36          error('Wrong calculation choice') % Wrong calculation
37      end
38
39  % Wrong shape input
40  else
41      error('Wrong shape choice')
42  end
```

Άσκηση αυτοαξιολόγησης 3.3

Δοκιμάστε να επεκτείνετε τον αλγόριθμο του Παραδείγματος 3.8, προσθέτοντας την επιλογή για υπολογισμό εμβαδού ή όγκου σφαίρας. Οι επιπλέον τύποι που θα χρειαστείτε είναι:

- Εμβαδόν σφαίρας ακτίνας R: $E = 4\pi R^2$,
- Όγκος σφαίρας ακτίνας R: $V = \frac{4}{3}\pi R^3$.

Παράδειγμα 3.9

Μία πολύ χρήσιμη λειτουργία των υπολογιστών είναι η επίλυση γραμμικών συστημάτων, τα οποία προκύπτουν από τη μοντελοποίηση φυσικών προβλημάτων. Η επίλυση μπορεί να γίνει είτε με άμεσες μεθόδους, δηλαδή μεθόδους που βασίζονται σε πράξεις πινάκων και οριζουσών (π.χ. η μέθοδος Cramer και η απαλοιφή Gauss), είτε με επαναληπτικές μεθόδους, οι οποίες μέσα από διαδοχικές επαναλήψεις καταφέρνουν να πλησιάσουν πολύ κοντά στην ακριβή λύση του συστήματος. Τέτοιες επαναληπτικές μέθοδοι είναι η μέθοδος Jacobi, η μέθοδος Gauss-Seidel και η μέθοδος των συζυγών κλίσεων.
Θεωρήστε ότι κατασκευάζετε εσείς ένα πρόγραμμα επίλυσης γραμμικών συστημάτων, το οποίο θα δίνει τη δυνατότητα στον χρήστη να επιλέξει ποια μέθοδο θέλει να χρησιμοποιήσει. Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη να επιλέξει αν θέλει να χρησιμοποιήσει άμεση ή επαναληπτική μέθοδο. Στη συνέχεια, ανάλογα με την επιλογή που έκανε, θα του δίνει τη δυνατότητα να χρησιμοποιήσει μία από τις μεθόδους που προαναφέραμε, δηλαδή:

- αν ο χρήστης επιλέξει άμεση μέθοδο, θα μπορεί να χρησιμοποιήσει είτε τη μέθοδο Cramer είτε την απαλοιφή Gauss,
- αν ο χρήστης επιλέξει επαναληπτική μέθοδο, θα μπορεί να χρησιμοποιήσει είτε τη μέθοδο Jacobi, είτε τη μέθοδο Gauss-Seidel, είτε τη μέθοδο των συζυγών κλίσεων.

Ο κώδικας θα πληροφορεί στο τέλος ποια μέθοδος έχει επιλεχθεί. Στην περίπτωση που ο χρήστης έχει εισάγει αριθμό που δεν ανήκει στις προκαθορισμένες επιλογές, θα εμφανίζεται κατάλληλο μήνυμα σφάλματος.

Λύση Παραδείγματος 3.9

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μία μεταβλητή στην οποία θα αποθηκεύεται η αρχική επιλογή για άμεσο ή επαναληπτικό τρόπο επίλυσης (π.χ. method) και άλλες δύο μεταβλητές για τις επιμέρους επιλογές που θα έχει (π.χ. method_amesi και method_epanal).

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να ελέγχει πρώτα αν θα ακολουθηθεί άμεση ή επαναληπτική επίλυση και, στη συνέχεια, να ζητάει την ακριβή μέθοδο μέσα από τις προκαθορισμένες επιλογές.
Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %LIN_SYS_METHOD Choose the method to solve a linear system
2  % Initialization
3  clear all
4  method = 0; % Tropos epilysis
5  method_amesi = 0; % Eidos amesis methodou
6  method_epanal = 0; % Eidos epanaliptikis methodou
7
8  % Main code
9
10 method = input('Epilekse tropo lysis: 1 - Amesos, 2 - Epanaliptikos:
       ');
11 if method == 1
12    method_amesi = input('Epilekse methodo: 1 - Cramer, 2 - Gauss: ')
          ;
13    if method_amesi == 1
14       disp('Methodos Cramer')
15    elseif method_amesi == 2
16       disp('Methodos Gauss')
17    else
18       error('Lathos methodos!')
19    end
20 elseif method == 2
21    method_epanal = input('Epilekse methodo: 1 - Jacobi, 2 - GS, 3 -
         SK: ');
22    if method_epanal == 1
23       disp('Methodos Jacobi')
24    elseif method_epanal == 2
25       disp('Methodos GS')
26    elseif method_epanal == 3
27       disp('Methodos SK')
28    else
29       error('Lathos methodos!')
30    end
31 else
32   error('Lathos tropos lysis!')
33 end
```

3.6 Η εντολή switch

Η εντολή `switch`, όπως γίνεται κατανοητό και από το όνομά της, λειτουργεί σαν «διακόπτης» μεταξύ διαφορετικών «θέσεων», δηλαδή μεταξύ διαφορετικών τμημάτων κώδικα. Συγκεκριμένα, η εντολή `switch` ελέγχει την τιμή που λαμβάνει μία μεταβλητή ή μία ολόκληρη παράσταση και ανάλογα οδηγεί στις κατάλληλες εντολές που θέλουμε να εκτελεστούν. Ας δούμε ένα απλό παράδειγμα σύνταξης (Κώδικας 3.5), όπου ελέγχεται μία μεταβλητή και ανάλογα με την τιμή που έχει, οδηγείται ο κώδικας στις κατάλληλες εντολές:

```
1  switch ( Μεταβλητή )
2
3      case Τιμή  1
4
5          ... Εντολές 1 ...
6
7      case   { Τιμή 2 , Τιμή 3 } ...
8
9          ... Εντολές 2 ...
10
11     . . .
12
13     otherwise
14
15         ... Εντολές n ...
16
17 end
```

Κώδικας 3.5: Παράδειγμα σύνταξης switch.

Παρατηρώντας το παραπάνω παράδειγμα σύνταξης, μπορούμε να συμπεράνουμε ότι η εντολή `switch` λειτουργεί ως εξής:

- Δίπλα στην εντολή `switch` ορίζεται η μεταβλητή (ή ολόκληρη παράσταση), η οποία θα ελεγχθεί στη συνέχεια.
- Δίπλα στις εντολές `case` ορίζονται οι πιθανές τιμές (περιπτώσεις) που μπορεί να έχει η προκαθορισμένη μεταβλητή.
- Σε κάθε περίπτωση μπορεί να περιλαμβάνονται μία ή περισσότερες τιμές, ανάλογα με τη φύση του προβλήματος.
- Κάτω από το κάθε `case` περιλαμβάνονται οι εντολές που θα εκτελεστούν, αν ισχύσει η συγκεκριμένη περίπτωση.
- Αν έχουμε συμπεριλάβει όλες τις επιθυμητές περιπτώσεις στα επιμέρους `case`, τότε μπορούμε να τερματίσουμε τη δομή `switch` με την εντολή `end`. Αν όμως θέλουμε να εισάγουμε μία τελευταία περίπτωση που να περιλαμβάνει όλες τις υπόλοιπες ενδεχόμενες τιμές που μπορεί να έχει η

ελεγχόμενη μεταβλητή, τότε χρησιμοποιούμε την εντολή `otherwise`. Με την εντολή αυτή καλύπτουμε την περίπτωση που δεν επαληθεύεται κάποιο από τα προκαθορισμένα `case`, και ορίζουμε τις εντολές που θα πρέπει να εκτελεστούν.

- Η δομή `switch` τερματίζεται με την εντολή `end`.

Παράδειγμα 3.10

Δημιουργήστε σε ένα αρχείο script τον κώδικα για ένα τυχερό παιχνίδι με ρίψεις ζαριού. Στην περίπτωση που το ζάρι έχει την τιμή 1, θα εκτυπώνεται μήνυμα ότι ο παίκτης έχει χάσει, αν έχει τιμή 2, 3 ή 4, θα εκτυπώνεται μήνυμα ότι ο γύρος αυτός δεν μετράει, ενώ, αν έχει τιμή 5 ή 6, θα εκτυπώνεται μήνυμα ότι έχει κερδίσει.

Λύση Παραδείγματος 3.10

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μία μεταβλητή στην οποία θα αποθηκεύεται η τυχαία τιμή του ζαριού (π.χ. zari), η οποία θα είναι ένας θετικός ακέραιος από το 1 μέχρι το 6. Η τυχαία αυτή τιμή μπορεί να ληφθεί με την εντολή `randi(6)`, η οποία επιστρέφει τυχαία έναν θετικό ακέραιο από το 1 μέχρι το 6.

Κύριος κώδικας: Στο πρόγραμμα θα πρέπει να οριστεί η μεταβλητή που θα χρησιμοποιηθεί ως «διακόπτης» `(switch)`, που στην περίπτωσή μας είναι η zari. Στη συνέχεια, θα πρέπει να οριστούν οι διαφορετικές περιπτώσεις `(case)`, δηλαδή οι τιμές που μπορεί να έχει η μεταβλητή zari, καθώς και η εντολή εκτύπωσης κατάλληλου μηνύματος για κάθε περίπτωση.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %DICE_GAME Game with dice
2  % Initialization
3  clear all
4  zari = randi(6);
5  disp(['The dice is ',num2str(zari)]);
6
7  % Main code
8  switch (zari)
9    case 1
10     disp('You loose!')
11   case {2,3,4}
12     disp('This turn doesn not count!')
13   case {5,6}
14     disp('You win!')
15 end
```

Παρατήρηση 3.7

Παρατηρείστε ότι στο παραπάνω Παράδειγμα έχουμε τη δυνατότητα να μην χρησιμοποιήσουμε την εντολή `otherwise`, καθώς οι τιμές που μπορεί να πάρει το ζάρι είναι αυστηρά καθορισμένες από την εντολή randi(6). Ο λόγος που επιλέξαμε το `case {5,6}` έναντι του `otherwise` είναι πρακτικός: με το `case` μπορούμε να δούμε με μία ματιά στον κώδικα ποιες είναι οι τιμές του ζαριού που κερδίζουν. Αν

επιθυμούσαμε, ωστόσο, θα μπορούσαμε να αντικαταστήσουμε τη γραμμή 12 του κώδικα με την εντολή `otherwise` και η λειτουργία του προγράμματος θα παρέμενε αμετάβλητη.

Από την ανάλυση της εντολής `switch` προκύπτει εύλογα το ερώτημα ποια είναι η διαφορά μεταξύ μιας δομής `if` και μιας δομής `switch`. Στην πραγματικότητα, οι διαφορές δεν είναι μεγάλες, αφού μια δομή `switch` μπορεί εύκολα να μετατραπεί σε δομή `if` με εισαγωγή των κατάλληλων ελέγχων. Ωστόσο, η δομή `if` είναι πιο γενική, καθώς σε κάθε σκέλος της οι έλεγχοι μπορεί να αφορούν διαφορετικές μεταβλητές ή συνδυασμό τους, ενώ η δομή `switch` συνήθως περιορίζεται από τη μεταβλητή που έχουμε ορίσει ως ελεγκτή, δίπλα στην εντολή `switch`.

Η πιο σημαντική, όμως, διαφορά έγκειται στην πρακτικότητα του `switch`, αφού μας επιτρέπει να βλέπουμε με μια ματιά ποιες εντολές εκτελούνται σε κάθε `case`. Το είδαμε αυτό στο Παράδειγμα 3.10, όπου η έκβαση του παιχνιδιού, ανάλογα με την τιμή του ζαριού, γίνεται εύκολα αντιληπτή. Αν υλοποιούσαμε τη λύση με χρήση δομής `if`, θα έπρεπε να χρησιμοποιήσουμε πολλαπλούς ελέγχους με χρήση &&, οι οποίοι θα καθιστούσαν τον κώδικα πιο δυσανάγνωστο.

Η ευκολία που προσφέρει η δομή `switch` είναι πολύ πιο εμφανής στην περίπτωση που ο έλεγχος αφορά χαρακτήρες. Ας δούμε το παρακάτω παράδειγμα που είναι μια παραλλαγή του Παραδείγματος 3.5.

Παράδειγμα 3.11

Είδαμε στο Παράδειγμα 3.5 ότι η ρίζα μιας εξίσωσης μπορεί να βρεθεί με τη μέθοδο της διχοτόμου, τη μέθοδο των διαδοχικών προσεγγίσεων και τη μέθοδο Newton-Raphson.
Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη να πληκτρολογήσει 'DIX' αν επιθυμεί να εφαρμοστεί η μέθοδος της διχοτόμου, 'DP' για μέθοδο των διαδοχικών προσεγγίσεων και 'NR' για μέθοδο Newton-Raphson. Ο κώδικας θα ελέγχει την επιλογή του χρήστη και θα τον πληροφορεί ποια μέθοδο έχει επιλέξει. Στην περίπτωση που ο χρήστης έχει εισάγει κάτι που δεν ανήκει στις παραπάνω επιλογές, θα εμφανίζεται κατάλληλο μήνυμα σφάλματος.

Λύση Παραδείγματος 3.11

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μόνο μία μεταβλητή στην οποία θα αποθηκεύεται η τιμή που εισήγαγε ο χρήστης (π.χ. method).

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να ελέγχει τη μεταβλητή method με μία δομή switch και να εκτυπώνει το κατάλληλο μήνυμα.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %METHOD_ROOTS Choose the method to find a root
2  % Initialization
3  clear all
4  method = input('Choose a method: ''DIX'' - Dihotomos, \''DP'' -
       Proseggiseis, ''NR'' - Newton-Raphson: ');
5
6  % Main code
7  switch (method)
8    case 'DIX'
9      disp('Solution method: Dihotomos')
10   case 'DP'
11     disp('Solution method: Diadohikes Proseggiseis')
12   case 'NR'
```

```
13        disp('Solution method: Newton-Raphson')
14      otherwise
15        error('The input is not valid!')
16    end
```

Παρατήρηση 3.8

Για να πραγματοποιήσουμε ελέγχους σε μεταβλητές χαρακτήρων με χρήση δομής `if`, πρέπει να χρησιμοποιήσουμε την εντολή `strcmpi`. Για παράδειγμα, αν θέλουμε να ελέγξουμε αν η μεταβλητή method του Παραδείγματος 3.11 αποτελείται από τη σειρά χαρακτήρων DIX, θα πρέπει να χρησιμοποιήσουμε τη συνθήκη

```
1  if strcmpi(method,'DIX')
```

η οποία δίνει αποτέλεσμα 1, όταν η μεταβλητή method περιέχει τη σειρά χαρακτήρων DIX, και 0 στην αντίθετη περίπτωση.

3.7 Ασκήσεις

Άσκηση 3.1. *Σε ένα ηλεκτρικό κύκλωμα δύο αντιστάσεις $R1$ και $R2$ μπορεί να είναι συνδεδεμένες είτε σε σειρά είτε παράλληλα. Στην πρώτη περίπτωση, η ολική αντίσταση R_{ol} είναι το άθροισμά τους, $R_{ol} = R1 + R2$, ενώ στη δεύτερη η ολική αντίσταση δίνεται από τη σχέση $1/R_{ol} = 1/R1 + 1/R2$.*

Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη τις τιμές των δύο ηλεκτρικών αντιστάσεων, καθώς και την πληροφορία αν είναι συνδεδεμένες σε σειρά ή παράλληλα και θα υπολογίζει την ολική αντίσταση, την οποία θα εμφανίζει με κατάλληλο μήνυμα.

Άσκηση 3.2. *Με βάση τη λύση του Παραδείγματος 3.6, δημιουργήστε ένα αρχείο script που θα υπολογίζει την περιοχή του αριθμού Mach με βάση την ταχύτητα του αεροσκάφους u που θα δίνει ο χρήστης σε m/sec. Στη συνέχεια, ο αλγόριθμος θα πρέπει να μετατρέπει την ταχύτητα σε km/h ώστε να υπολογίζει τον αριθμό Mach από τη σχέση $M = u/c$, όπου c είναι η ταχύτητα του ήχου 1235 km/h.*
Υπόδειξη: *Μπορείτε να χρησιμοποιήσετε τη λύση του Παραδείγματος 3.6 σαν βάση για να χτίσετε τον κώδικά σας. Συγκεκριμένα, μπορείτε να προσθέσετε τον έλεγχο και τη μετατροπή των μονάδων της ταχύτητας πριν τον υπολογισμό του αριθμού Mach.*

Άσκηση 3.3. *Η βαθμολογία σε ένα μάθημα από το πρόγραμμα Erasmus κυμαίνεται από 0 έως 10 (grade) και μπορεί κατατάσσεται στις παρακάτω κατηγορίες:*

- *FAIL όταν είναι μικρότερη του 5,*
- *PASS όταν είναι ακριβώς 5,*
- *GOOD όταν είναι μεγαλύτερη από 5 έως και 6.5,*
- *VERY GOOD όταν είναι μεγαλύτερη από 6.5 έως και 8.5,*
- *EXCELLENT όταν είναι μεγαλύτερη από 8.5.*

Αν ο βαθμός στο συγκεκριμένο μάθημα διαμορφώνεται από τον βαθμό εργαστηρίου με συντελεστή 0.4 και από τον τελικό βαθμό εξέτασης με συντελεστή 0.6, δημιουργήστε ένα αρχείο script το οποίο θα ζητάει από τον χρήστη να εισάγει τον βαθμό εργαστηρίου και τον τελικό βαθμό εξέτασης και θα υπολογίζει την κατηγορία κατάταξης.
Υπόδειξη: *Θα πρέπει να συμπεριλάβετε στους ελέγχους και την περίπτωση κατά την οποία ο χρήστης εισήγαγε βαθμολογίες εκτός των προβλεπόμενων ορίων. Στην περίπτωση αυτή, θα πρέπει να εκτυπώνεται κατάλληλο μήνυμα σφάλματος.*

Άσκηση 3.4. *Θεωρούμε την εφαρμογή που παρουσιάζεται στο Παράδειγμα 3.9, που αφορά την επιλογή μεθόδου εύρεσης ρίζας, μέσα από ένα σύνολο τριών μεθόδων (μέθοδος της διχοτόμου, μέθοδος των διαδοχικών προσεγγίσεων και Newton-Raphson) και την εισαγωγή κατάλληλων δεδομένων. Συγκεκριμένα, αν επιλεχθεί η μέθοδος της διχοτόμου, τότε ο χρήστης καλείται να εισάγει δύο αριθμούς που αντιστοιχούν στο αρχικό και στο τελικό σημείο ενός διαστήματος, μέσα στο οποίο εκτιμάει ότι βρίσκεται η ρίζα. Υπάρχει, όμως περίπτωση, ο χρήστης να εισάγει εκ παραδρομής ανάποδα τα σημεία του διαστήματος ή να εισάγει δύο ίδια σημεία με αποτέλεσμα το διάστημα να είναι μηδενικό.*

Για να αποφευχθούν αυτά τα πιθανά λάθη, προσθέστε στον κώδικα του Παραδείγματος 3.9 κατάλληλο έλεγχο, ώστε, αν παρουσιαστεί κάποια από αυτές τις περιπτώσεις κατά την εισαγωγή των δεδομένων στη μέθοδο της διχοτόμου, να εμφανίζεται στον χρήστη κατάλληλο μήνυμα σφάλματος.

Άσκηση 3.5. *Θεωρούμε έναν θετικό ακέραιο αριθμό a. Ο αριθμός αυτός μπορεί να είναι είτε περιττός είτε άρτιος. Στην περίπτωση που είναι θετικός, ο αριθμός αυτός μπορεί να είναι πρώτος, δηλαδή να διαιρείται μόνο με τον εαυτό του και τη μονάδα. Επίσης, αν ο αριθμός είναι πρώτος, τότε υπάρχει πιθανότητα να είναι δίδυμος πρώτος αριθμός, δηλαδή να είναι πρώτος και ο $a+2$ ή ο $a-2$ (για παράδειγμα δίδυμοι πρώτοι αριθμοί είναι το 11 και το 13, καθώς είναι και οι δύο πρώτοι και η διαφορά τους είναι 2).*

Άσκηση 3.6. *Δημιουργήστε ένα αρχείο script το οποίο θα ζητάει από τον χρήστη να εισάγει έναν θετικό ακέραιο και, στη συνέχεια, αν αυτός έχει εισάγει πράγματι θετικό ακέραιο, θα κάνει τους ακόλουθους ελέγχους και θα ενημερώνει ανάλογα τον χρήστη:*

- *Αν ο αριθμός είναι άρτιος θα τυπώνει το μήνυμα 'Artios'.*
- *Αν ο αριθμός είναι περιττός, θα ελέγχει αν είναι ταυτόχρονα και πρώτος. Αν δεν είναι πρώτος, θα τυπώνει μήνυμα 'Perittos'.*
- *Στην περίπτωση που ο περιττός αριθμός είναι πρώτος, θα κάνει επιπλέον έλεγχο για το αν είναι δίδυμος πρώτος, δηλαδή θα ελέγχει και τους $a+2$ και $a-2$. Αν δεν είναι κάποιος από αυτούς πρώτος, τότε θα τυπώνει μήνυμα 'Prwtos'.*
- *Τέλος, αν προκύψει ότι κάποιος εκ των $a+2$ και $a-2$ είναι πρώτος, τότε θα τυπώνει μήνυμα 'Didymos Prwtos'.*

Υπόδειξη 1: *Για να βρείτε αν ένας αριθμός είναι άρτιος ή περιττός μπορείτε να ελέγξετε το υπόλοιπο της διαίρεσής του με το 2, χρησιμοποιώντας την εντολή* `rem(a,2)`. *Για να ελέγξετε αν ένας αριθμός είναι πρώτος, μπορείτε να χρησιμοποιήσετε την εντολή* `isprime(a)`, *η οποία επιστρέφει 1 αν ο a είναι πρώτος και 0 αν δεν είναι.*

Υπόδειξη 2: *Σημειώνεται ότι όλοι οι πρώτοι αριθμοί είναι περιττοί εκτός από το 2. Κατασκευάστε τον κώδικά σας έτσι, ώστε να επιστρέφει το μήνυμα 'Artios Prwtos' στην περίπτωση που ο χρήστης εισάγει το 2.*

Άσκηση 3.7. *Εμπλουτίστε τον κώδικα της προηγούμενης άσκησης, ελέγχοντας αν ο αριθμός που εισήγαγε ο χρήστης είναι όντως θετικός ακέραιος. Αν είναι, τότε η διαδικασία θα συνεχίζεται κανονικά, ενώ σε αντίθετη περίπτωση θα εμφανίζεται κατάλληλο μήνυμα σφάλματος και το πρόγραμμα θα τερματίζεται.*
Υπόδειξη: *Για τον τρόπο με τον οποίο ελέγχουμε αν ένας αριθμός είναι θετικός ακέραιος, μπορείτε να συμβουλευτείτε τη λύση του Παραδείγματος 3.4.*

3.8 Απαντήσεις στις ασκήσεις αυτοαξιολόγησης

Λύση άσκησης αυτοαξιολόγησης 3.1

Τα αρχικά βήματα του εναλλακτικού αλγορίθμου είναι ίδια με αυτά του Παραδείγματος 3.2 και περιλαμβάνουν τη δημιουργία τριών μεταβλητών στις οποίες θα αποθηκευτούν οι αριθμοί που θα δώσει ο χρήστης. Στη συνέχεια, κάνουμε την αρχική θεώρηση ότι η x1 έχει τη μεγαλύτερη τιμή και την καταχωρίζουμε στη μεταβλητή xMax. Το ερώτημα που πρέπει να απαντήσουμε είναι αν όντως η xMax είναι μεγαλύτερη από τις x2 και x3. Για να βρούμε την απάντηση, συγκρίνουμε πρώτα τη x2 με τη xMax:

- Αν x2<xMax, τότε η xMax περιέχει τη μεγαλύτερη τιμή από τις μεταβλητές που έχουμε ελέγξει μέχρι τώρα (δηλαδή τις x1 και x2), οπότε δεν χρειάζεται να κάνουμε κάτι άλλο σε αυτή τη φάση.
- Αν, όμως, x2>xMax, τότε πρέπει να καταχωρίσουμε τη x2 στην xMax, ώστε η τελευταία να περιέχει τη μέγιστη τιμή από τις x1 και x2.

Μετά τον έλεγχο αυτό, έχουμε καταφέρει να καταχωρίσουμε στη xMax τη μεγαλύτερη από τις x1 και x2. Πρέπει, λοιπόν, να προχωρήσουμε στον έλεγχο της x3, κάτι το οποίο γίνεται με αντίστοιχο τρόπο:

- Αν x3<xMax, τότε η xMax περιέχει τη μεγαλύτερη τιμή από όλες τις μεταβλητές, οπότε αυτή είναι και η λύση της άσκησης.
- Αν, όμως, x3>xMax, τότε πρέπει να καταχωρίσουμε τη x3 στην xMax, και με αυτόν τον τρόπο έχουμε βρει τη μέγιστη από τις τρεις τιμές.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %MAXIMUM Find the maximum of three numbers
2  % Initialization
3   clear all
4   x1 = input('Give a number: ');
5   x2 = input('Give another number: ');
6   x3 = input('Give a third number: ');
7
8   % Main code
9   xMax = x1;
10
11   if xMax<x2
12     xMax = x2;
13   end
14
15   if xMax<x3
16     xMax = x3;
17   end
18
19   disp(['The maximum number is ',num2str(xMax)])
```

Λύση άσκησης αυτοαξιολόγησης 3.2

Το πρώτο βήμα για τη λύση της άσκησης είναι η αρχικοποίηση της μεταβλητής n, αντίστοιχα με το Παράδειγμα 3.4. Στη συνέχεια, θα ελέγξουμε αν ικανοποιούνται οι δύο προϋποθέσεις που χρειαζόμαστε για τη λύση, δηλαδή:

- αν η τιμή του n είναι θετικός αριθμός και
- αν το n είναι ακέραιος (γίνεται με χρήση της εντολής `round`, βλ. Παράδειγμα 3.4).

Αν ισχύουν *και* οι δύο αυτές προϋποθέσεις, τοτε μπορούμε να υπολογίσουμε και να εκτυπώσουμε τη ζητούμενη τιμή, δηλαδή την τιμή του 2 υψωμένη εις τη n-οστή δύναμη.
Σε αντίθετη περίπτωση, το πρόγραμμα θα εκτυπώνει μήνυμα σφάλματος.
Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %POS_INT Find if the value is a positive integer
2  % Initialization
3   clear all
4   n = input('Give a positive integer: ');
5
6   % Main code
7   if n>0 && round(n)==n
8     disp(['The ',num2str(n),'-th power of 2 is ',num2str(2^n)])
9   else
10    error('The number you entered is not a positive integer!')
11  end
```

Λύση άσκησης αυτοαξιολόγησης 3.3

Η δομή της λύσης του Παραδείγματος 3.8 μας επιτρέπει να προσθέσουμε το επιπλέον σχήμα εισάγοντας ένα "block" εντολών, αντίστοιχο με αυτά που ήδη υπάρχουν για τον κύβο και τον κύλινδρο. Συγκεκριμένα, δεν θα χρειαστούμε κάποια επιπλέον μεταβλητή, αφού η ακτίνα της σφαίρας μπορεί να αποθηκευτεί στη μεταβλητή R, ενώ οι μεταβλητές για την επιλογή σχήματος και είδους υπολογισμού υπάρχουν ήδη.
Αυτό που πρέπει να γίνει είναι να προστεθεί στο κείμενο της εντολής `input` της μεταβλητής shape (γραμμή 12) η επιπλέον δυνατότητα για τους υπολογισμούς της σφαίρας (... 3 - sphere). Στη συνέχεια, θα πρέπει να προστεθεί μία εμφωλευμένη δομή επιλογής για την περίπτωση της σφαίρας, η οποία θα περιλαμβάνει την εισαγωγή της ακτίνας, τον έλεγχο για εμβαδόν ή όγκο, καθώς και τους κατάλληλους υπολογισμούς.
Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %AREA_VOLUME Find area or the volume of 3D shapes
2  % Initialization
3   clear all
4   shape = 0;  % Type of 3D shape
5   Calc = 0; % Calculation of area or volume
6   L = 0; % Length of cube side
7   R = 0; % Radius of cylinder or sphere
8   H = 0; % Height of cylinder
9
```

```
% Main code
disp('Choose 3D shape:');
shape = input('1 - Cube, 2 - Cylinder, 3 - Sphere: ');
disp('Choose calculation:');
Calc = input('1 - Area, 2 - Volume: ');

% Code for cube
if shape == 1
    L = input('Length of the side of the cube: ');
    if Calc == 1
        disp(['Result: ', num2str(6*L^2)]) % Area
    elseif Calc == 2
        disp(['Result: ', num2str(L^3)]) % Volume
    else
        error('Wrong calculation choice') % Wrong calculation
    end


% Code for cylinder
elseif shape == 2
    R = input('Radius of cylinder: ');
    H = input('Height of cylinder: ');
    if Calc == 1
        disp(['Result: ', num2str(2*pi*R*H + 2*pi*R^2)]) % Area
    elseif Calc == 2
        disp(['Results: ', num2str(pi*R^2*H)]) % Volume
    else
        error('Wrong calculation choice') % Wrong calculation
    end

% Code for sphere
elseif shape == 3
    R = input('Radius of cphere: ');
    if Calc == 1
        disp(['Result: ', num2str(4*pi*R^2)]) % Area
    elseif Calc == 2
        disp(['Results: ', num2str(4/3*pi*R^3)]) % Volume
    else
        error('Wrong calculation choice') % Wrong calculation
    end

% Wrong shape input
else
    error('Wrong shape choice')
end
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ξενόγλωσση

Hoffman, J. D. (2001). *Numerical Methods for Engineers and Scientists, (2nd ed.)* CRC Press, New York.

Young, D. F., Munson, B. R., Okiishi, T. H. and Huebsch, W. W. (2010). *A Brief Introduction to Fluid Mechanics (5th ed.)* John Wiley & Sons, Jefferson City.

ΚΕΦΑΛΑΙΟ 4

ΕΠΑΝΑΛΗΠΤΙΚΕΣ ΔΙΑΔΙΚΑΣΙΕΣ

Σύνοψη:

Σε αυτό το κεφάλαιο παρουσιάζονται οι εντολές με τις οποίες μπορούν να υλοποιηθούν και να ελεγχθούν επαναληπτικές δομές. Πιο συγκεκριμένα, παρουσιάζονται οι διαδικασίες δημιουργίας επαναληπτικών διαδικασιών, δηλαδή βρόχων, με χρήση των εντολών `for` και `while`, καθώς και ο τρόπος δημιουργίας εμφωλευμένων επαναλήψεων. Τέλος, γίνεται παρουσίαση των εντολών `break` και continue, οι οποίες επιτρέπουν τον εσωτερικό χειρισμό των επαναληπτικών διαδικασιών.

Προαπαιτούμενη γνώση: Τα κεφάλαια 1, 2 και 3 του παρόντος συγγράμματος.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του κεφαλαίου αυτού, θα γνωρίζετε:

- τις βασικές εντολές για την υλοποίηση επαναληπτικών διαδικασιών,
- τη χρήση εμφωλευμένων επαναληπτικών δομών,
- τις βασικές εντολές διαχείρισης επαναληπτικών διαδικασιών.

Γλωσσάριο επιστημονικών όρων

- Ατέρμονος βρόχος
- Βρόχος
- Επαναληπτική διαδικασία
- Εμφωλευμένη επαναληπτική δομή
- Μεταβλητή μετρητής

4.1 Εισαγωγή

Οι ηλεκτρονικοί υπολογιστές αποτελούν πολύ χρήσιμα εργαλεία για πολλές σύγχρονες τεχνολογικές εφαρμογές και αυτό διότι έχουν τη δυνατότητα να εκτελούν απλές πράξεις και διαδικασίες με πολύ μεγάλη ταχύτητα. Για να εκμεταλλευτούμε, όμως, τη δυνατότητα αυτή, πρέπει να καταφέρουμε να μετατρέψουμε τα πολύπλοκα προβλήματα που συναντάμε στον πραγματικό κόσμο σε μια σειρά από επαναλαμβανόμενες απλές εντολές και πράξεις. Η διαδικασία αυτή περιλαμβάνει πολλά στάδια, μερικά από τα οποία είναι η μοντελοποίηση των φυσικών προβλημάτων, η κατάστρωση των εξισώσεων που περιγράφουν τους φυσικούς μηχανισμούς και η επίλυση των εξισώσεων αυτών. Η συνεισφορά των ηλεκτρονικών υπολογιστών βρίσκεται κυρίως στο τελευταίο στάδιο, αφού μας βοηθούν να λύσουμε με ακρίβεια και ταχύτητα προβλήματα ή επίλυση γραμμικών συστημάτων που αποτελούνται από τεράστιο πλήθος αγνώστων.

Οι τεχνικές για την επίλυση σύνθετων εξισώσεων με χρήση αριθμητικών τεχνικών είναι γνωστές εδώ και δεκαετίες. Η εφαρμογή, όμως, αυτών των τεχνικών χωρίς τη βοήθεια των μηχανών είναι μια διαδικασία επίπονη και, σε πολλές περιπτώσεις, αδύνατη. Για να επιλυθεί ένα γραμμικό σύστημα που έχει αρκετές χιλιάδες αγνώστων, χρειάζεται τεράστιο ανθρώπινο δυναμικό που να κάνει τις απαραίτητες πράξεις. Όσο και αν αυτό φαντάζει απίθανο στις ημέρες μας, υπήρχε μία εποχή που ήταν η μοναδική λύση, και αυτό μπορεί κανείς να το αντιληφθεί αν μελετήσει τις πρώτες προσπάθειες της Εθνικής Διοίκησης Αεροναυτικής και Διαστήματος των ΗΠΑ (NASA) τη δεκαετία του '50 να εκτελέσει διαστημικές πτήσεις. Για να υπολογιστούν οι τροχιές των διαστημικών οχημάτων, χρειαζόταν η συντονισμένη εργασία πλήθους «ανθρώπινων υπολογιστών», δηλαδή επιστημόνων επιφορτισμένων με την εκτέλεση πράξεων και τον έλεγχο αποτελεσμάτων (Εικόνα 4.1). Κάτι τέτοιο στη σύγχρονη εποχή είναι αδιανόητο, αφού με τις σημερινές δυνατότητες των ηλεκτρονικών υπολογιστών αντίστοιχα προβλήματα μπορούν να λυθούν μέσα σε λίγα λεπτά, χρησιμοποιώντας κατάλληλες εντολές και, βέβαια, επαναληπτικές διαδικασίες.

Εικόνα 4.1: Στις πρώτες αποστολές της Εθνικής Διοίκησης Αεροναυτικής και Διαστήματος των ΗΠΑ (NASA) οι υπολογισμοί εκτελούνταν από «ανθρώπινους υπολογιστές» (Πηγή: https://en.wikipedia.org/wiki/Computer_(occupation)).

Οι επαναληπτικές διαδικασίες χρησιμοποιούνται σε περιπτώσεις που θέλουμε την επανειλημμένη εκτέλεση μιας ομάδας εντολών. Ας δούμε ένα πολύ απλό παράδειγμα: έστω ότι θέλουμε να υπολογίσουμε τον μέσο όρο της βαθμολογίας που λάβαμε σε 10 εργαστήρια Matlab, τα οποία περιλαμβάνονται στο πλαίσιο του αντίστοιχου μαθήματος. Η τιμή του μέσου όρου προκύπτει αν προσθέσουμε τους 10 βαθμούς και, στη συνέχεια, διαιρέσουμε το άθροισμα με το πλήθος τους. Για να εκτελέσουμε αυτήν τη διαδικασία στο Matlab πρέπει να εισάγουμε μία-μία τις τιμές αυτών των 10 βαθμών μέσω της γνωστής μας εντολής `input`, να τις προσθέσουμε και, στη συνέχεια, να βρούμε τον μέσο όρο. Με τα εργαλεία που έχουμε στη διάθεσή μας έως εδώ θα έπρεπε να τοποθετήσουμε στον κώδικά μας 10 φορές την εντολή `input`, προσθέτοντας κάθε φορά τον νέο βαθμό στο τρέχον άθροισμα, όπως φαίνεται στο παρακάτω πρόγραμμα του Κώδικα 4.1.

```
1  %MEAN_GRADE Calculate the mean grade of a student
2  % Initialization
3  clear all
4  x = 0;
5  athr = 0;
6
7  % Main code
8  x = input('Dose to vathmo: ');
9  athr = athr + x;
10 x = input('Dose to vathmo: ');
11 athr = athr + x;
12
13 ...
14
15 x = input('Dose to vathmo: ');
16 athr = athr + x;
17 disp(athr/10)
```

Κώδικας 4.1: Υπολογισμός μέσου όρου χωρίς τη χρήση δομής επανάληψης.

Είναι προφανές ότι ο Κώδικας 4.1 δεν αποτελεί πρακτική λύση στο πρόβλημά μας. Είναι, όμως, χρήσιμος ως προς το ότι μας επιτρέπει να παρατηρήσουμε ποιες εντολές επαναλαμβάνονται στην υλοποίηση της συγκεκριμένης λύσης. Οι εντολές αυτές είναι (i) η `input` με την οποία εισάγεται κάθε φορά ο νέος βαθμός και (ii) η πράξη της πρόσθεσης του νέου βαθμού στο τρέχον άθροισμα, δηλαδή οι εντολές

```
x = input('Dose to vathmo: ');
athr = athr + x;
```

Μπορούμε, λοιπόν, να εντάξουμε τις δύο αυτές γραμμές κώδικα μέσα σε μία δομή επανάληψης, η οποία θα εκτελέσει τις αντίστοιχες εντολές όσες φορές ορίσουμε. Η πιο απλή δομή επανάληψης ή, αλλιώς, βρόχου που μπορεί να δώσει το επιθυμητό αποτέλεσμα, προκύπτει με χρήση της εντολής `for`, την οποία θα περιγράψουμε αναλυτικά στη συνέχεια. Προς το παρόν, ας δούμε πώς μετατρέπεται ο Κώδικας 4.1, αν χρησιμοποιήσουμε μία δομή επανάληψης `for`:

```
1  %MEAN_GRADE Calculate the mean grade of a student
2  % Initialization
3  clear all
4  x = 0;
5  athr = 0;
6
7  % Main code
```

```
 8   for i=1:10
 9     x = input('Dose to vathmo: ');
10     athr = athr + x;
11   end
12   disp(athr/10)
```

Κώδικας 4.2: Υπολογισμός μέσου όρου με τη χρήση της δομής επανάληψης for.

Παρατηρούμε ότι ο νέος κώδικας είναι πολύ πιο σύντομος και ευανάγνωστος από τον αρχικό. Ο αριθμός των επαναλήψεων φαίνεται από την έκφραση «ένα έως δέκα» (1:10) που ακολουθεί την εντολή `for`, ενώ οι εντολές που θα εκτελεστούν επαναλαμβανόμενα, αυτές δηλαδή που βρίσκονται μέσα στη δομή `for`, γίνονται εύκολα αντιληπτές από το αυξημένο περιθώριο στον κειμενογράφο.

Στις επόμενες παραγράφους θα παρουσιαστούν οι δύο βασικές επαναληπτικές δομές που βασίζονται στις εντολές `for` και `while`. Θα δούμε ότι στην πρώτη περίπτωση η επανάληψη βασίζεται σε μια μεταβλητή «μετρητή», η οποία μετρά πόσες φορές εκτελούνται οι εντολές που βρίσκονται μέσα στον επαναληπτικό βρόχο, ενώ στη δεύτερη η διαδικασία βασίζεται στον έλεγχο μίας συνθήκης. Στη συνέχεια, θα εξετάσουμε διάφορες εντολές χειρισμού, όπως οι `break` και `continue`, οι οποίες μας επιτρέπουν να παρέμβουμε στην πορεία της επαναληπτικής διαδικασίας, και να τη διακόψουμε ή να την περιορίσουμε, ανάλογα με τη δομή του κώδικα.

4.2 Η εντολή for

Η δομή επανάληψης της εντολής `for` χρησιμοποιείται όταν είναι εκ των προτέρων γνωστός ο αριθμός των επαναλήψεων που θέλουμε να εκτελεστούν. Για παράδειγμα, είδαμε στην προηγούμενη παράγραφο μία εφαρμογή όπου θέλαμε να υπολογίσουμε τον μέσο όρο δέκα βαθμών. Σε αυτή την περίπτωση, ήταν δεδομένος ο αριθμός των επαναλήψεων, αφού πρέπει να διαβαστούν και να προστεθούν μεταξύ τους οι δέκα βαθμοί. Με τη βοήθεια της εντολής `for` δημιουργήσαμε έναν επαναληπτικό βρόχο ο οποίος εκτελούσε τις απαραίτητες εντολές, χρησιμοποιώντας μία μεταβλητή «μετρητή». Για να κατανοήσουμε τη σύνταξη που χρησιμοποιήσαμε στον Κώδικα 4.2, παρουσιάζεται στον Κώδικα 4.3 η γενική μορφή της δομής επανάληψης της εντολής `for`.

```
1   for Μεταβλητή = min : βήμα : max
2
3           ... Εντολές ...
4
5   end
```

Κώδικας 4.3: Σύνταξη βρόχου `for`.

Παρατηρούμε ότι δίπλα στην εντολή `for` τοποθετούμε τη μεταβλητή «μετρητή» και ορίζουμε το εύρος τιμών μέσα στο οποίο θα κινηθεί. Για παράδειγμα, αν χρησιμοποιήσουμε την έκφραση 1:10, αυτό σημαίνει ότι η μεταβλητή θα λάβει κατά σειρά τις τιμές $1, 2, 3, \ldots$ έως και το 10 και, στη συνέχεια, θα τερματιστεί η δομή επανάληψης, οδηγώντας στο τμήμα του προγράμματος μετά το `end`.

Στην παραπάνω περίπτωση το βήμα κατά το οποίο μεταβάλαμε τον «μετρητή» ήταν 1 και για αυτό δεν χρειάστηκε να το ορίσουμε, καθώς είναι η προεπιλογή στη γλώσσα Matlab. Αν, ωστόσο, θέλουμε να ορίσουμε διαφορετικό βήμα, τότε μπορούμε να εισάγουμε το βήμα με το οποίο θέλουμε να κινηθεί ο «μετρητής» ανάμεσα στην αρχική και την τελική τιμή του. Για παράδειγμα, αν θέλουμε η μεταβλητή μας να διατρέξει μόνο τους άρτιους αριθμούς από το 2 μέχρι το 10, τότε μπορούμε να χρησιμοποιήσουμε την

έκφραση 2:2:10, το οποίο μεταφράζεται ως «από το δύο, έως το δέκα, με βήμα δύο». Με άλλα λόγια, οι τιμές που θα λάβει η μεταβλητή «μετρητής» είναι οι 2, 4, 6, 8 και 10. Στην περίπτωση που θέλαμε η μεταβλητή να διατρέξει μόνο τις περιττές τιμές, τότε θα έπρεπε να χρησιμοποιήσουμε την έκφραση 1:2:9, που μεταφράζεται ως «από το ένα, έως το εννέα, με βήμα δύο». Οι διαδοχικές τιμές που θα λάμβανε ο «μετρητής» είναι οι 1, 3, 5, 7 και 9. Υπενθυμίζεται ότι παρόμοια παρατήρηση είχαμε κάνει και στην Ενότητα 1.6 κατά τον ορισμό του διανύσματος με ορισμένη δομή.

Μπορεί κανείς να αναρωτηθεί τι θα γινόταν στην περίπτωση που χρησιμοποιούσαμε την έκφραση 1:2:10, καθώς, ξεκινώντας από τον αριθμό 1 και προχωρώντας με βήμα 2, όπως είδαμε, λαμβάνουμε μόνο περιττούς αριθμούς, ενώ ως μέγιστη τιμή έχουμε ορίσει έναν άρτιο, δηλαδή το 10. Η απάντηση είναι ότι ο «μετρητής» δεν θα έφτανε ποτέ την τιμή 10! Κατά την επαναληπτική διαδικασία θα λάμβανε τις τιμές 1, 3, 5, 7 και 9 και, στη συνέχεια, θα τερματιζόταν ο βρόχος, καθώς δεν θα μπορούσε να υπερβεί το όριο του 10 για να πάει στην επόμενη τιμή, 11. Μπορούμε, λοιπόν, να συμπεράνουμε ότι είτε χρησιμοποιήσουμε την έκφραση 1:2:10 είτε την έκφραση 1:2:9, το αποτέλεσμα είναι ακριβώς το ίδιο.

Ας δούμε μερικά παραδείγματα με τη χρήση της επαναληπτικής δομής `for`.

Παράδειγμα 4.1

Δημιουργήστε ένα αρχείο script που θα προσομοιώνει τη ρίψη ενός ζαριού 5 φορές.

Λύση Παραδείγματος 4.1

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μόνο μία μεταβλητή, τον «μετρητή» (π.χ. i). Τη μεταβλητή αυτή συχνά δεν την αρχικοποιούμε, καθώς λαμβάνει αυτόματα την πρώτη τιμή από την εντολή `for`.

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να εκτελεί έναν βρόχο χρησιμοποιώντας τη μεταβλητή «μετρητή» i, όπου σε κάθε επανάληψη θα εκτυπώνεται ένας τυχαίος αριθμός από το 1 έως το 6. Ο τυχαίος αυτός αριθμός μπορεί να ληφθεί με την εντολή randi(6).

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %RANDOM_NUMBER Display 5 random numbers
2  % Initialization
3  clear all
4
5  % Main code
6  for i=1:5
7    disp(randi(6))
8  end
```

Παρατήρηση 4.1

Για μεταβλητές «μετρητές» χρησιμοποιούμε συνήθως τους λατινικούς χαρακτήρες i, j, k, l, m και n. Η επιλογή αυτή έχει τις ρίζες της σε παλαιότερες γλώσσες προγραμματισμού όπου υπήρχε διάκριση στις μεταβλητές σε ακέραιες και πραγματικές, ανάλογα με το γράμμα από το οποίο ξεκινούσε το όνομά τους. Στις σύγχρονες γλώσσες δεν υπάρχουν τέτοιοι περιορισμοί, ωστόσο συχνά χρησιμοποιούμε τη σύμβαση ότι οι μεταβλητές με όνομα που ξεκινάν με κάποιον από τους παραπάνω χαρακτήρες παίρνουν ακέραιες τιμές.
Θα πρέπει να σημειώσουμε ότι αν ονομάσουμε κάποια μεταβλητή i και ο κώδικάς μας περιλαμβάνει πράξεις με μιγαδικούς αριθμούς, υπάρχει κίνδυνος να δημιουργηθεί σύγχυση μεταξύ του ονόματος της

μεταβλητής και της φανταστικής μονάδας i, η οποία στο Matlab συμβολίζεται με τον χαρακτήρα i. Στην περίπτωση αυτή, καλό είναι να αποφεύγεται η χρήση μεταβλητής με το όνομα αυτό.

Παράδειγμα 4.2

Ας υποθέσουμε ότι θέλουμε να σχεδιάσουμε στο χαρτί τη γραφική παράσταση της συνάρτησης $y = \log x$ στο διάστημα $x \in [1,2]$. Μπορούμε να σχεδιάσουμε, αρχικά, τους άξονες x και y και, στη συνέχεια, να τοποθετήσουμε σημεία (x,y) τα οποία, όταν τα ενώσουμε, θα μας δώσουν τη γραφική παράσταση.
Δημιουργήστε ένα αρχείο script που θα εκτυπώνει 11 σημεία (x,y) της συνάρτησης $y = \log x$ στο διάστημα $x \in [1,2]$.
Υπενθυμίζεται ότι με $\log x$ συμβολίζουμε τον λογάριθμο με βάση το e.

Λύση Παραδείγματος 4.2

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε δύο μεταβλητές στις οποίες θα αποθηκεύονται οι τιμές x και y (π.χ. x και y) και μία μεταβλητή «μετρητή» (π.χ. i). Τις μεταβλητές x και y μπορούμε να τις αρχικοποιήσουμε με την πρώτη τιμή του διαστήματος που θα σχεδιάσουμε, δηλαδή $x = 1$ και $y = \log 1 = 0$.

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να εκτελεί έναν βρόχο χρησιμοποιώντας τη μεταβλητή «μετρητή» i. Αφού το πρώτο σημείο (x,y) έχει ήδη βρεθεί κατά την αρχικοποίηση, πρέπει να βρούμε τα υπόλοιπα 10 σημεία, για να έχουμε το σύνολο 11. Άρα, η μεταβλητή i θα πρέπει να παίρνει τιμές από το 1 έως το 10 με βήμα 1 - άρα το βήμα μπορεί να παραλειφθεί.
Σε κάθε επανάληψη, το πρόγραμμα θα πρέπει να υπολογίζει και να μας εκτυπώνει τα x και y. Για τον υπολογισμό του x, πρέπει να χωρίσουμε το διάστημα $[1,2]$ σε 10 υποδιαστήματα, πράγμα που μπορεί να γίνει σύμφωνα με τη σχέση $x = 1+i*\frac{2-1}{10}$, όπου i είναι ο «μετρητής» μας που παίρνει τιμές από το 1 έως το 10. Το y, που αντιστοιχεί σε κάθε x, μπορεί να υπολογίζεται κατευθείαν από τον τύπο της συνάρτησης $y = \log x$.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %X_Y_VALUES Calculate the x-y values of y = log(x) in [1,2]
2  % Initialization
3  clear all
4  x = 1;
5  y = 0;
6  disp([x,y])
7
8  % Main code
9  for i=1:10
10   x = 1+i*(2-1)/10;
11   y = log(x);
12   disp([x,y])
13 end
```

Σημειώνεται ότι η εντολή για την κατασκευή της γραφικής παράστασης μιας συνάρτησης παρουσιάζεται στο επόμενο κεφάλαιο.

Παρατήρηση 4.2

Στο προηγούμενο παράδειγμα χρησιμοποιήσαμε σαν «μετρητή» τη μεταβλητή i, στην οποία δώσαμε ακέραιες τιμές από το 1 έως το 10, έτσι ώστε να υπολογίσουμε τις επιθυμητές τιμές για το x. Μπορεί, λοιπόν, κανείς να αναρωτηθεί αν μπορούμε να χρησιμοποιήσουμε την ίδια τη x σαν «μετρητή».
Η απάντηση είναι ότι ναι, μπορούμε να χρησιμοποιήσουμε τη x σαν «μετρητή» αρκεί να της καθορίσουμε το κατάλληλο εύρος και το σωστό βήμα. Ωστόσο, γενικά, αποφεύγουμε να χρησιμοποιούμε μη-ακέραιες μεταβλητές σαν «μετρητές» για δύο λόγους: (α) Είναι πάντα χρήσιμο να γνωρίζουμε πόσες επαναλήψεις έχουμε κάνει, καθώς εκτελείται το πρόγραμμα, και αυτό μας το πληροφορεί ο «μετρητής». Για παράδειγμα, σε μεγάλα, χρονοβόρα προγράμματα που εκτελούν χιλιάδες επαναλήψεις, είναι χρήσιμο να εκτυπώνονται σε τακτά διαστήματα (π.χ. ανά 100 επαναλήψεις) πληροφορίες για την πορεία των υπολογισμών, έτσι ώστε να είναι ενήμερος ο χρήστης για την πορεία της λύσης. (β) Σε σπάνιες περιπτώσεις, μπορεί το μη-ακέραιο βήμα να εισάγει σφάλματα στογγυλοποίησης στους υπολογισμούς, με αποτέλεσμα να μην τερματιστεί η επαναληπτική διαδικασία στη σωστή επανάληψη. Για παράδειγμα, αν εκτελέσουμε μία επαναληπτική διαδικασία από 1 έως 3 με βήμα 0.66667 (≈ 2/3), δηλαδή

```
>> for x = 1:0.66667:3
...
```

τότε η μεταβλητή x θα πάρει διαδοχικά τις τιμές 1, 1.66667 και 2.33334, αλλά δεν θα φτάσει ποτέ στο 3. Αυτό συμβαίνει επειδή η προσέγγιση του ρητού αριθμού 2/3 δεν έχει την απαιτούμενη ακρίβεια. Αν, αντί για την προσέγγιση 0.66667, χρησιμοποιήσουμε το 2/3, τότε το x θα πάρει *και* την τιμή 3 και ο κώδικας θα λειτουργήσει σωστά.
Σε κάθε περίπτωση, πάντως, η χρήση μη-ακέραιου «μετρητή» δεν είναι λάθος και συχνά μπορεί να οδηγήσει σε συντομότερο κώδικα και λιγότερες μεταβλητές, σε σχέση με τη χρησιμοποίηση ακέραιου «μετρητή».

Παρατήρηση 4.3

Είδαμε στα προηγούμενα παραδείγματα τον σημαντικό ρόλο που παίζει η μεταβλητή «μετρητής» στις επαναληπτικές δομές. Η μεταβλητή αυτή μπορεί απλώς να μετράει πόσες επαναλήψεις έχουν ολοκληρωθεί (όπως στο Παράδειγμα 4.1) ή να συμμετέχει στις εντολές και τους υπολογισμούς που περιλαμβάνονται μέσα στην επαναληπτική δομή (όπως στο παράδειγμα 4.2 όπου η τιμή του «μετρητή» i χρησιμοποιείται για τον υπολογισμό του x).
Σε κάθε περίπτωση, ένα στοιχείο, στο οποίο θα πρέπει να δώσουμε ιδιαίτερη προσοχή, είναι να μην μεταβάλλουμε την τιμή του «μετρητή» με κάποια εντολή μέσα στον επαναληπτικό βρόχο. Κάτι τέτοιο θα μπορούσε να αλλάξει τη ροή των υπολογισμών και να έχει απρόβλεπτες συνέπειες για τα αποτελέσματα του κώδικα. Συμπερασματικά, λοιπόν, μπορούμε να πούμε ότι η χρήση του «μετρητή» για υπολογισμούς μέσα στον επαναληπτικό βρόχο είναι πολύ χρήσιμη για διάφορες λειτουργίες, η μεταβολή του όμως είναι επικίνδυνη και πρέπει πάντα να αποφεύγεται.

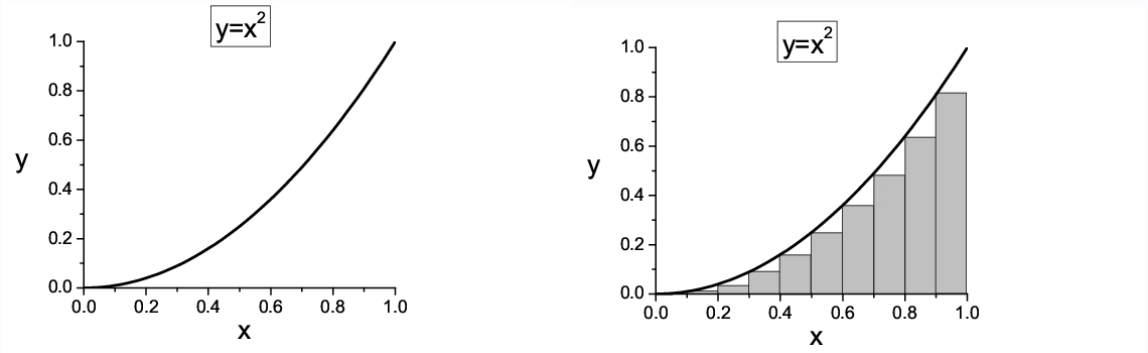
Άσκηση αυτοαξιολόγησης 4.1

Δοκιμάστε να φτιάξετε έναν εναλλακτικό αλγόριθμο επίλυσης του Παραδείγματος 4.2, όπου θα χρησιμοποιείτε σαν «μετρητή» τη μεταβλητή x.

Παράδειγμα 4.3

Ας υποθέσουμε ότι θέλουμε να υπολογίσουμε προσεγγιστικά το εμβαδόν κάτω από τη γραφική παράσταση της συνάρτησης $y = x^2$. Ένας σχετικά απλός τρόπος για να το κάνουμε αυτό στον υπολογιστή, είναι να κατασκευάσουμε πολλά παραλληλόγραμμα με κατάλληλο ύψος, όπως φαίνεται στο Σχήμα 4.1, και να αθροίσουμε τα εμβαδά τους. Η μέθοδος αυτή ονομάζεται «Κανόνας του ορθογωνίου»

(Μπράτσος, 2015) και βασίζεται στον τύπο παρεμβολής του Newton (Hoffman, 2001).



Σχήμα 4.1: Υπολογισμός του εμβαδού κάτω από τη γραφική παράσταση της συνάρτησης $y = x^2$, με τη βοήθεια εμβαδών εγγεγραμμένων παραλληλογράμμων (Παράδειγμα 4.3).

Δημιουργήστε ένα αρχείο script που θα υπολογίζει το εμβαδόν 10 παραλληλογράμμων που είναι εγγεγραμμένα στη γραφική παράσταση της συνάρτησης $y = x^2$ από το 0 έως το 1. Στη συνέχεια, συγκρίνετε το αποτέλεσμα με την ακριβή τιμή του εμβαδού.

Λύση Παραδείγματος 4.3

Για να λύσουμε αυτό το πρόβλημα, μπορούμε να βασιστούμε στη λύση του Παραδείγματος 4.2 και να προσθέσουμε τα επιπλέον στοιχεία που χρειάζονται. Ας δούμε πώς διαμορφώνεται η λύση στο Παράδειγμα αυτό:

Αρχικοποίηση μεταβλητών: Όπως και στο Παράδειγμα 4.2, θα χρειαστούμε δύο μεταβλητές στις οποίες θα αποθηκεύονται οι τιμές του x και του y (π.χ. x και y). Τις μεταβλητές αυτές θα τις αρχικοποιήσουμε με την πρώτη τιμή του διαστήματός μας, δηλαδή $x = 0$ και $y = 0$. Θα χρειαστούμε, επίσης, μία μεταβλητή, για το πλήθος των παραλληλογράμμων που θα χρησιμοποιήσουμε (π.χ. N) η οποία θα έχει την τιμή 10 και μία μεταβλητή η οποία θα περιέχει το μήκος της βάσης κάθε παραλληλογράμμου (π.χ. dx). Στη dx θα δώσουμε την τιμή $1/N$, αφού 1 είναι το συνολικό μήκος στον άξονα x και N είναι ο αριθμός των παραλληλογράμμων που θέλουμε να χωρέσουν μέσα στο μήκος αυτό. Τέλος, θα χρειαστούμε μία μεταβλητή για το άθροισμα των εμβαδών των παραλληλογράμμων (π.χ. E), την οποία θα αρχικοποιήσουμε με το ουδέτερο στοιχείο της πρόσθεσης, δηλαδή με μηδέν.

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να εκτελεί έναν βρόχο χρησιμοποιώντας μία μεταβλητή «μετρητή» (π.χ. i) που θα μεταβάλλεται από 1 έως N. Σε κάθε επανάληψη θα πρέπει να υπολογίζεται το εμβαδόν (βάση×ύψος) του αντίστοιχου παραλληλογράμμου. Το μήκος της βάσης είναι κοινό για όλα τα παραλληλόγραμμα και είναι ίσο με dx, οπότε η δυσκολία έγκειται στον υπολογισμό του ύψους. Αφού τα παραλληλόγραμμα που θα χρησιμοποιήσουμε είναι εγγεγραμμένα στην καμπύλη της $y = x^2$, το πρώτο ύψος θα το υπολογίσουμε στο $x = 0$ και θα είναι $y = f(0) = 0$, δηλαδή θα έχουμε μηδενικό ύψος. Το ύψος για το δεύτερο παραλληλόγραμμο θα το υπολογίσουμε στο $x = dx$ και θα είναι $y = f(dx) = dx^2$, για το τρίτο στο $x = 2dx$ και θα είναι $y = f(2dx) = (2dx)^2$ κ.ο.κ. Για να εντάξουμε τον υπολογισμό του ύψους μέσα στην επαναληπτική διαδικασία, θα πρέπει να εκφράσουμε με κάποιον αλγόριθμο το πώς υπολογίζεται το ύψος ανάλογα με το παραλληλόγραμμο το οποίο υπολογίζουμε. Για παράδειγμα, παρατηρούμε ότι

- όταν ο μετρητής είναι i=1 (1ο παραλληλόγραμμο), υπολογίζουμε το ύψος στο $x_1 = 0$ και είναι μηδέν,

- όταν ο μετρητής είναι i=2 (2ο παραλληλόγραμμο), υπολογίζουμε το ύψος στο $x_2 = dx$ και είναι $y_2 = dx^2$,
- όταν ο μετρητής είναι i=3 (3ο παραλληλόγραμμο), υπολογίζουμε το ύψος στο $x_3 = 2dx$ και είναι $y_3 = (2dx)^2$,
- όταν ο μετρητής είναι i=4 (4ο παραλληλόγραμμο), υπολογίζουμε το ύψος στο $x_4 = 3dx$ και είναι $y_4 = (3dx)^2$ κ.λπ.

Μπορούμε, λοιπόν, να συμπεράνουμε ότι σε κάθε επανάληψη, για να έχουμε εγγεγραμμένα παραλληλόγραμμα, θα πρέπει να υπολογίσουμε το ύψος στο

$$x_i = (i-1)dx$$

και η τιμή του θα είναι

$$y_i = x_i^2$$

Αφού μπορούμε να υπολογίσουμε το ύψος του εγγεγραμμένου παραλληλογράμμου σε κάθε επανάληψη, μπορούμε να βρούμε και το εμβαδόν του από τη σχέση βάση×ύψος, δηλαδή

$$E_i = dx \times y_i$$

Αυτό, λοιπόν, που θα κάνει ο κώδικας είναι να υπολογίζει σε κάθε επανάληψη το εμβαδόν του αντίστοιχου παραλληλογράμμου, με τον τρόπο που αναφέραμε, και να τον προσθέτει στο άθροισμα της μεταβλητής E.

Επαλήθευση: Στο συγκεκριμένο παράδειγμα, μπορούμε να κάνουμε μία εκτίμηση για την ακρίβεια των υπολογισμών μας, εκτυπώνοντας την ακριβή τιμή του εμβαδού, που υπολογίζεται εύκολα με ένα ορισμένο ολοκλήρωμα. Έτσι, για τη συνάρτηση $y = x^2$ με $x \in [0,1]$ το εμβαδόν κάτω από τη γραφική παράσταση είναι $E_{exact} = 1/3 = 0.33333...$

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %AREA_1 Calculate area below y = x^2 in [0,1]
2  % Initialization
3   clear all
4   x = 0;
5   y = 0;
6   N=10;
7   dx = 1/N;
8   E = 0;
9
10  % Main code
11  for i=1:N
12    x = (i-1)*dx;
13    y = x^2;
14
15    % Calculation of Area
16    E = E + y*dx;
17  end
18
```

```
19    disp(['Methodos parallilogrammwn: ',num2str(E)])
20    disp(['Akribhs ypologismos: ',num2str(1/3)])
```

Παρατήρηση 4.4

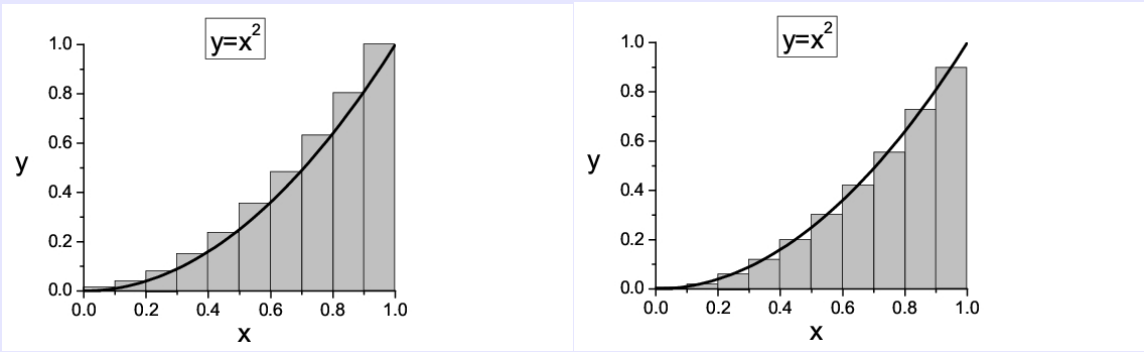
Αν εκτελέσουμε τον κώδικα του Παραδείγματος 4.3, θα δούμε ότι η τιμή που προκύπτει από τη μέθοδο των παραλληλογράμμων (0.285) δεν είναι πολύ κοντά στην ακριβή τιμή (0.3333...). Αυτό συμβαίνει γιατί χρησιμοποιήσαμε πολύ λίγα παραλληλόγραμμα, με αποτέλεσμα η προσέγγιση που παίρνουμε να είναι πολύ χονδρική. Μπορούμε όμως εύκολα να βελτιώσουμε την προσέγγισή μας, αυξάνοντας τον αριθμό των παραλληλογράμμων. Αυτό μπορεί να γίνει αν στον κώδικα του Παραδείγματος 4.3 μεταβάλλουμε το N, οπότε η λύση γίνεται

- για N=100, E=0.32835,
- για N=1000, E=0.33283,
- για N=10000, E=0.33328 κ.ο.κ.

Παρατηρούμε ότι, καθώς ο αριθμός των παραλληλογράμμων αυξάνεται, η διαφορά με την ακριβή λύση που είναι 0.3333... σταδιακά μειώνεται και για N=10000 έχει φτάσει σε αρκετά καλό επίπεδο. Επίσης, παρόλο που ο αριθμός των υπολογισμών έχει αυξηθεί αρκετά, ο χρόνος που χρειάζεται ο υπολογιστής για τις απαραίτητες πράξεις είναι πολύ μικρός. Αυτή η δυνατότητα για μεγάλο αριθμό υπολογισμών σε σύντομο χρονικό διάστημα μας βοηθάει να λύσουμε πολύπλοκα φυσικά προβλήματα, αρκεί να τα μετατρέψουμε σε αλληλουχία απλών πράξεων, όπως κάναμε στο Παράδειγμα 4.3.
Επίσης, μπορούμε να βελτιώσουμε τις τεχνικές που χρησιμοποιούμε, αν για παράδειγμα, αντί για εγγεγραμμένα παραλληλόγραμμα πάρουμε παραλληλόγραμμα που τέμνουν με κατάλληλο τρόπο την καμπύλη μας ή ακόμα και τραπέζια. Υπάρχουν πολλές ακόμα τεχνικές για βελτίωση των αποτελεσμάτων, οι οποίες μπορούν να βρεθούν σε ένα καλό βιβλίο αριθμητικής ανάλυσης.

Άσκηση αυτοαξιολόγησης 4.2

Με βάση τη λύση του Παραδείγματος 4.2, δοκιμάστε να φτιάξετε έναν αλγόριθμο που θα προσεγγίζει το εμβαδόν κάτω από τη γραφική παράσταση της συνάρτησης $y = x^2$ με 10 περιγεγραμμένα παραλληλόγραμμα, όπως φαίνεται στην αριστερή εικόνα του Σχήματος 4.2. Τι παρατηρείτε σε σχέση με το αποτέλεσμα του Παραδείγματος 4.2;



Σχήμα 4.2: Υπολογισμός του εμβαδού κάτω από τη γραφική παράσταση της συνάρτησης $y = x^2$, με τη βοήθεια εμβαδών (α) περιγεγραμμένων παραλληλογράμμων (Ασκ. Αυτοαξιολόγησης 4.2), (β) παραλληλογράμμων που τέμνουν την καμπύλη στο μέσο κάθε διαστήματος (Ασκ. Αυτοαξιολόγησης 4.3).

Άσκηση αυτοαξιολόγησης 4.3

Με βάση τη λύση του Παραδείγματος 4.2, δοκιμάστε να φτιάξετε έναν αλγόριθμο που θα προσεγγίζει το εμβαδόν κάτω από τη γραφική παράσταση της συνάρτησης $y = x^2$ με 10 παραλληλόγραμμα που θα τέμνουν την καμπύλη στο μέσο κάθε διαστήματος, όπως φαίνεται στη δεξιά εικόνα του Σχήματος 4.2. Τι παρατηρείτε σε σχέση με το αποτέλεσμα του Παραδείγματος 4.2;

4.3 Η εντολή while

Σε όλα τα παραδείγματα που είδαμε στην προηγούμενη ενότητα, ο αριθμός των επαναλήψεων που έπρεπε να εκτελεστούν ήταν δεδομένος από την εκφώνηση του προβλήματος (π.χ. 10 βαθμοί, 11 σημεία, 100 χρονικά βήματα κ.λπ.). Ωστόσο, σε πολλές εφαρμογές, ο αριθμός των επαναλήψεων δεν είναι γνωστός εκ των προτέρων, αλλά καθορίζεται από κάποια συνθήκη.

Για παράδειγμα, ας θυμηθούμε την εφαρμογή του Παραδείγματος 4.1, που προσομοίωνε ρίψεις ζαριού. Στο συγκεκριμένο παράδειγμα, η εκφώνηση μας ζητούσε να προσομοιώσουμε 5 ρίψεις, οπότε η χρήση της δομής `for` ήταν η ενδεδειγμένη. Αν, όμως, η εκφώνηση ζητούσε να προσομοιώνονται οι ρίψεις μέχρι ο χρήστης να τερματίσει τη διαδικασία, τότε το πλήθος των επαναλήψεων δεν θα ήταν προκαθορισμένο και δεν θα μπορούσε να οριστεί κατάλληλο εύρος για τη μεταβλητή «μετρητή». Στην περίπτωση αυτή, ο έλεγχος για τη συνέχιση της διαδικασίας θα έπρεπε να γίνεται με βάση την επιλογή του χρήστη μετά από την ολοκλήρωση κάθε επανάληψης. Για τη λειτουργία αυτή, υπάρχει η εντολή `while` που μας επιτρέπει να εκτελούμε επαναλήψεις όσο μια συνθήκη είναι αληθής, το συντακτικό της οποίας φαίνεται στον Κώδικα 4.4.

```
1  while Συνθήκη
2
3          ... Εντολές ...
4
5  end
```

Κώδικας 4.4: Σύνταξη βρόχου `while`.

Παρατηρούμε ότι, σε αντίθεση με τους βρόχους `for`, δεν υπάρχει προκαθορισμένη μεταβλητή «μετρητής». Οι εντολές που βρίσκονται μέσα στη δομή επαναλαμβάνονται μέχρι να σταματήσει να ισχύει η συνθήκη που συνοδεύει το `while`. Οι συνθήκες που χρησιμοποιούνται με την εντολή `while` είναι ίδιες με αυτές που είδαμε στο προηγούμενο κεφάλαιο για την εντολή `if` και μπορούν να περιλαμβάνουν είτε απλούς είτε σύνθετους ελέγχους.

Ιδιαίτερη προσοχή πρέπει να δοθεί στο γεγονός ότι, για να ξεκινήσει η επαναληπτική διαδικασία, πρέπει η συνθήκη του `while` να ικανοποιείται. Αν η συνθήκη αυτή δεν ικανοποιείται όταν, κατά την εκτέλεση του προγράμματος, φτάσουμε στην εντολή `while`, τότε η επαναληπτική διαδικασία θα αγνοηθεί και η εκτέλεση θα συνεχιστεί με τις εντολές που ακολουθούν το `end`. Μία τέτοια περίπτωση, όπου η επαναληπτική διαδικασία δεν ενεργοποιείται λόγω μη ικανοποίησης της συνθήκη του `while`, φαίνεται στον Κώδικα 4.5.

```
1  x = 0;
2  while x > 0
3
4          ... Εντολές ...
5
6  end
```

Κώδικας 4.5: Περίπτωση βρόχου `while` που δεν ενεργοποιείται, λόγω μη ικανοποίησης της συνθήκης.

Παρατηρούμε ότι η μεταβλητή x έχει αρχικά τιμή μηδέν, ενώ, για να ξεκινήσει η επαναληπτική διαδικασία, η τιμή του x πρέπει να είναι μεγαλύτερη του μηδενός. Αυτό σημαίνει ότι το πρόγραμμα δεν θα εισέλθει ποτέ μέσα στον βρόχο.

Αντίστοιχη προσοχή πρέπει να δοθεί και στην περίπτωση που δεν τερματίζεται ποτέ η επαναληπτική διαδικασία, λόγω του ότι δεν μεταβάλλεται το αποτέλεσμα του ελέγχου στη συνθήκη του `while`. Αν, δηλαδή, η μεταβλητή που χρησιμοποιεί ο έλεγχος του `while` δεν μεταβάλλεται μέσα στον βρόχο ή μεταβάλλεται αλλά με τέτοιο τρόπο ώστε να μην αλλάζει το αποτέλεσμα του ελέγχου, τότε η επαναληπτική διαδικασία θα συνεχίζεται επ' άπειρον (ή τουλάχιστον μέχρι να τη διακόψει ο χρήστης με τον συνδυασμό πλήκτρων Ctrl και c)! Μία τέτοια περίπτωση ατέρμονου βρόχου φαίνεται στον Κώδικα 4.6.

```
1  x = 0;
2  while x >= 0
3    x = x + 1;
4  end
```

Κώδικας 4.6: Περίπτωση ατέρμονου βρόχου `while`.

Παρατηρούμε ότι στον Κώδικα 4.6 η αρχική τιμή της μεταβλητής x ικανοποιεί τη συνθήκη ελέγχου του `while` και, κατά συνέπεια, η επαναληπτική διαδικασία θα ξεκινήσει. Ωστόσο, η τιμή του x συνεχώς αυξάνεται με αποτέλεσμα να ισχύει πάντα η συνθήκη x>=0. Αυτό σημαίνει ότι ο βρόχος δεν πρόκειται να τερματιστεί και το πρόγραμμα θα τρέχει επ' άπειρον, αν δεν επέμβει ο χρήστης.

Σχετικά με τις εφαρμογές της δομής επανάληψης `while`, αυτές αφορούν τόσο πρακτικές λειτουργίες στο πλαίσιο του προγραμματισμού όσο και τεχνικές υλοποίησης αριθμητικών μεθόδων. Για παράδειγμα, μία εφαρμογή της εντολής `while` είναι στον έλεγχο των δεδομένων που εισάγει ο χρήστης σε ένα πρόγραμμα. Ο έλεγχος της συνθήκης του `while` μας επιτρέπει να ζητάμε επανειλημμένα από τον χρήστη να εισάγει τα κατάλληλα δεδομένα και δεν επιτρέπει τη συνέχιση εκτέλεσης του κώδικα αν δεν ικανοποιηθεί αυτή η απαίτηση. Ας δούμε μερικά παραδείγματα:

Παράδειγμα 4.4

Δημιουργήστε ένα αρχείο script που θα ζητάει από τον χρήστη να εισάγει τη βαθμολογία ενός εξαμηνιαίου μαθήματός του. Το πρόγραμμα πρέπει να εξασφαλίζει ότι η τιμή της βαθμολογίας είναι ένας ακέραιος αριθμός από 0 έως το 10.

Λύση Παραδείγματος 4.4

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μόνο μία μεταβλητή στην οποία θα αποθηκεύεται ο βαθμός (π.χ. vathmos). Τη μεταβλητή αυτή θα την αρχικοποιήσουμε χρησιμοποιώντας την εντολή `input` με μια τιμή που θα δώσει ο χρήστης.

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να ελέγχει αν η τιμή που εισήγαγε ο χρήστης *δεν* είναι έγκυρη, δηλαδή αν είναι (α) μικρότερη του μηδενός ή (β) μεγαλύτερη του δέκα ή (γ) μη-ακέραιος αριθμός (για το πώς μπορούμε να ελέγξουμε αν ένας αριθμός είναι ακέραιος, συμβουλευτείτε το Παράδειγμα 3.4). Αν δεν ισχύει τίποτα από τα παραπάνω, τότε η τιμή είναι έγκυρη και δεν υπάρχει λόγος να εισέλθει το πρόγραμμα στον επαναληπτικό βρόχο, οπότε τον προσπερνάει και προχωράει στον τερματισμό. Στην περίπτωση που κάποια από τις συνθήκες ισχύει, τότε το πρόγραμμα εισέρχεται στον βρόχο και με νέα εντολή `input` ζητάει από τον χρήστη να δώσει έγκυρη τιμή. Η επαναληπτική διαδικασία διαρκεί έως ότου ο χρήστης εισάγει έγκυρη τιμή και μόνο τότε τερματίζεται το πρόγραμμα.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%CHECK_GRADE Check id a grade is valid
% Initialization
clear all
vathmos = input('Parakalw, dwste to vathmo: ');

% Main code
while vathmos<0 || vathmos>10 || round(vathmos)~=vathmos
  disp('Mh egkyros vatmos')
  vathmos = input('Parakalw, dwste EGKYRO vathmo: ');
end
```

Παράδειγμα 4.5

Δημιουργήστε ένα αρχείο script που θα υπολογίζει τους πρώτους δέκα πρώτους αριθμούς (first primes) που είναι μεγαλύτεροι του 100 (υπενθυμίζεται ότι πρώτος ονομάζεται κάθε φυσικός αριθμός μεγαλύτερος του 1 με την ιδιότητα οι μόνοι φυσικοί διαιρέτες του να είναι το 1 και ο εαυτός του.)

Λύση Παραδείγματος 4.5

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μία μεταβλητή στην οποία θα αποθηκεύεται ο τρέχων αριθμός (π.χ. arithmos) ο οποίος θα ελέγχεται αν είναι πρώτος. Τη μεταβλητή αυτή θα την αρχικοποιήσουμε με τον αμέσως μεγαλύτερο φυσικό αριθμό από το 100, δηλαδή το 101. Επίσης, εφόσον το πλήθος των πρώτων αριθμών που επιθυμούμε να βρούμε είναι δέκα, θα χρειαστούμε και μία μεταβλητή «μετρητή» (π.χ. count) την οποία αρχικοποιούμε με μηδέν.

Κύριος κώδικας: Το πρόγραμμα θα πρέπει εκτελέσει μία επαναληπτική διαδικασία που να ελέγχει αν η τιμή της μεταβλητής arithmos είναι πρώτος αριθμός (ο έλεγχος μπορεί να γίνει με την εντολή isprime()). Στην περίπτωση που είναι όντως πρώτος αριθμός, θα πρέπει να τον εκτυπώνει και να αυξάνει τον «μετρητή» count κατά ένα.
Μετά τον έλεγχο, θα πρέπει να αυξάνεται η τιμή της μεταβλητής arithmos, ώστε να προχωράει η διαδικασία ελέγχου. Η αύξηση του arithmos μπορεί να γίνεται κατά δύο μονάδες, καθώς γνωρίζουμε ότι κανένας άρτιος (εκτός του 2) αριθμός δεν είναι πρώτος. Ωστόσο, το πρόγραμμα θα λειτουργούσε σωστά και με αύξηση κατά μία μονάδα, απλώς θα χρειαζόταν μεγαλύτερο χρόνο για τους υπολογισμούς.
Η ολοκλήρωση της επαναληπτικής διαδικασίας θα γίνεται όταν η μεταβλητή «μετρητής» πάρει την τιμή 10, οπότε και θα έχουμε βρει τους ζητούμενους πρώτους αριθμούς.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%FIND_TWIN_PRIME Find twin prime numbers
% Initialization
clear all
clear
arithmos = 101;
count = 0;

% Main code
```

```
 9   while count < 10
10     if isprime(arithmos)
11       disp(arithmos)
12       count = count + 1;
13     end
14     arithmos = arithmos + 2;
15   end
```

Παρατήρηση 4.5

Είδαμε στη λύση του Παραδείγματος 4.5 ότι, για να μετρήσουμε δέκα αριθμούς, μπορούμε να αρχικοποιήσουμε τη μεταβλητή «μετρητή» με την τιμή 0 και να ορίσουμε ως συνθήκη ότι η τιμή της δεν πρέπει να φτάσει το 10, δηλαδή count< 10. Εναλλακτικά, θα μπορούσαμε να αρχικοποιήσουμε τη μεταβλητή «μετρητή» με την τιμή 1 και να ορίσουμε ως συνθήκη ότι η τιμή της δεν πρέπει να ξεπεράσει το 10, δηλαδή count$<= 10$. Και οι δύο αυτές εκδοχές του προγράμματος δίνουν ακριβώς το ίδιο αποτέλεσμα, ενώ το ποια από τις δύο θα χρησιμοποιηθεί έγκειται στην ευχέρεια του προγραμματιστή να το αποφασίσει, με βάση την προτίμησή του.

Σε κάθε περίπτωση όμως θα πρέπει να είμαστε πολύ προσεχτικοί, ώστε ο συνδυασμός αρχικοποίησης - συνθήκης να μας δίνει στο τέλος το επιθυμητό πλήθος αριθμών. Η επιλογή λανθασμένου συνδυασμού οδηγεί είτε σε μεγαλύτερο είτε σε μικρότερο πλήθος επαναλήψεων και αυτό επηρεάζει την ορθότητα της λύσης. Για παράδειγμα, η αρχικοποίηση του «μετρητή» με 1 σε συνδυασμό με τη συνθήκη count< 10 δίνει μόνο 9 πρώτους αριθμούς, ενώ η αρχικοποίηση με 0 σε συνδυασμό με τη συνθήκη count$<= 10$ δίνει 11.

Για να μην υπάρχει κίνδυνος σφάλματος, καλό είναι να ακολουθούμε πάντα μία από τις δύο στρατηγικές και να κάνουμε έναν έλεγχο εκτελώντας τον κώδικά μας με ένα μικρό και εύκολα ελέγξιμο πλήθος αριθμών, ώστε να διαπιστώσουμε ότι όντως δεν υπάρχει λάθος στον συνδυασμό αρχικοποίησης - συνθήκης.

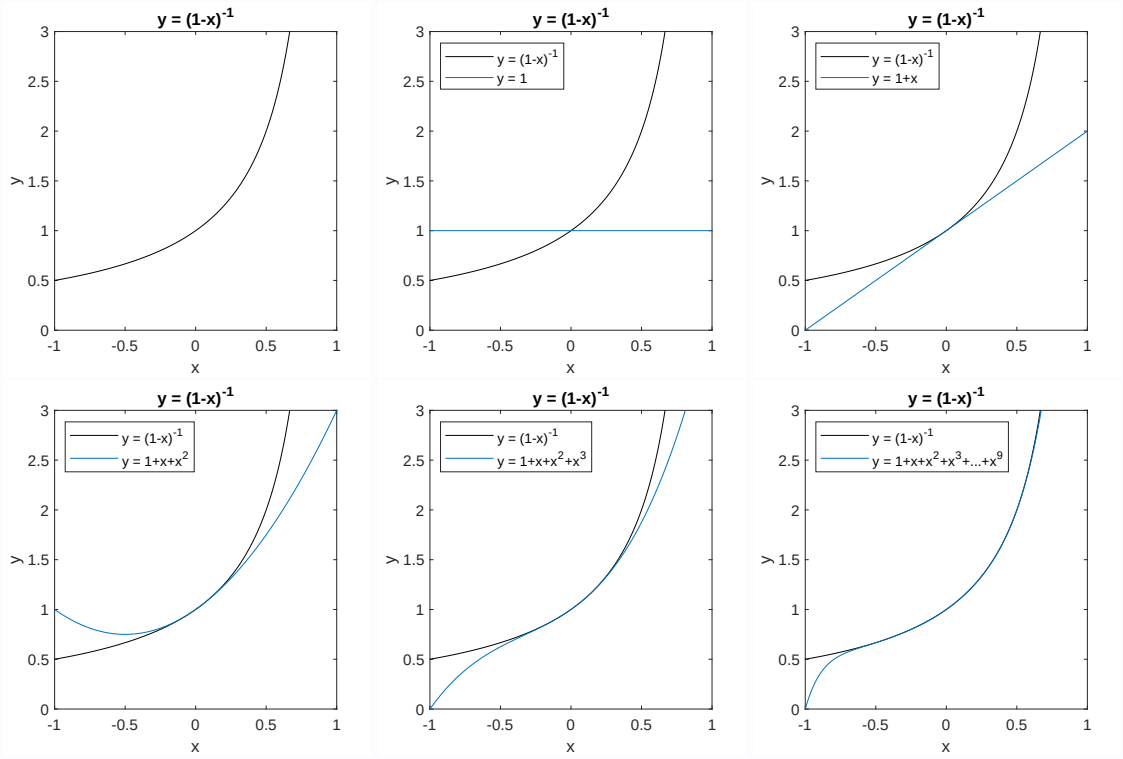
Παράδειγμα 4.6

Από την εμπειρία μας στα μαθηματικά γνωρίζουμε ότι μία πολυωνυμική συνάρτηση είναι πολύ εύκολη στην επεξεργασία (π.χ. ολοκλήρωση, παραγώγιση κ.λπ.) σε σχέση με άλλες πιο πολύπλοκες συναρτήσεις (ρητές, λογαριθμικές κ.λπ.). Για να εκμεταλλευτούμε την ευκολία αυτή, μπορούμε να προσεγγίσουμε μία συνάρτηση με ένα πολυώνυμο McLaurin. Δεν θα αναφέρουμε στο σημείο αυτό περισσότερα για το πώς βρίσκονται τα πολυώνυμα McLaurin, πότε συγκλίνουν και για τις συνθήκες που πρέπει να ικανοποιεί η συνάρτηση που προσεγγίζουμε, καθώς ξεπερνούν τις ανάγκες του συγκεκριμένου παραδείγματος (για περισσότερες πληροφορίες κάθε ενδιαφερόμενος/η μπορεί να ανατρέξει, παραδείγματος χάριν, στο σύγγραμμα των Thomas *et al.*, 2009). Ωστόσο, για να γίνει εποπτικά κατανοητή η διαδικασία προσέγγισης της συνάρτησης, παρουσιάζεται στο Σχήμα 4.3 η γραφική παράσταση της συνάρτησης $y = \frac{1}{1-x}$ και οι προσεγγίσεις της από πολυώνυμα McLaurin. Παρατηρούμε ότι όσο περισσότερους όρους έχει το πολυώνυμο, τόσο καλύτερη είναι η προσέγγιση της συνάρτησης και τόσο μικρότερη η απόκλιση της προσεγγιστικής από την ακριβή τιμή. Με βάση τα παραπάνω, στο παράδειγμα αυτό θα δημιουργήσουμε ένα πρόγραμμα που θα υπολογίζει την τιμή της συνάρτησης $y = \frac{1}{1-x}$ για $x \in (-1,1)$, χρησιμοποιώντας ένα πολυώνυμο McLaurin.

Πιο συγκεκριμένα, θα δημιουργήσουμε ένα αρχείο script που θα προσεγγίζει την τιμή της συνάρτησης $y = \frac{1}{1-x}$ στο $x = 0.5$ με το πολυώνυμο

$$y = 1 + x + x^2 + x^3 + x^4 + x^5 + ...$$

Κρατήστε τόσους όρους από το πολυώνυμο, ώστε το υπόλοιπο να είναι μικρότερο του 10^{-5}.
Υπόδειξη: Ο κώδικας θα πρέπει να εκτελεί μια επαναληπτική διαδικασία όπου θα προσθέτει νέους όρους στο πολυώνυμο μέχρις ότου επιτευχθεί η επιθυμητή ακρίβεια. Το υπόλοιπο θα ελέγχεται από την απόλυτη τιμή κάθε νέου όρου, δηλαδή την τιμή του $|x^n|$. Όταν η απόλυτη τιμή του n-οστού όρου γίνει μικρότερη από 10^{-5}, τότε το πρόγραμμα θα πρέπει να εκτυπώνει το αποτέλεσμα του αθροίσματος και να τερματίζεται.



Σχήμα 4.3: Προσέγγιση της συνάρτησης $y = \frac{1}{1-x}$ για $x \in (-1,1)$ από πολυώνυμα McLaurin. Όσο περισσότερους όρους έχει το πολυώνυμο, τόσο καλύτερη η προσέγγιση.

Λύση Παραδείγματος 4.6

Αρχικοποίηση μεταβλητών: Για το πρόβλημα αυτό, θα χρειαστούμε μία μεταβλητή στην οποία θα αποθηκεύεται η τιμή του x που θέλουμε να προσεγγίσουμε (π.χ. x), μία μεταβλητή στην οποία θα αποθηκεύεται κάθε νέος όρος $|x^n|$ (π.χ. term) και μία για το συνολικό άθροισμα (π.χ. s). Τις μεταβλητές αυτές μπορούμε να τις αρχικοποιήσουμε ως εξής:

- Τη x με την τιμή του x που μας δίνεται από την εκφώνηση, δηλαδή x=0.5.
- Την term με μια τυχαία τιμή, αφού μόλις υπολογιστεί ο πρώτος όρος του πολυωνύμου, η τιμή της θα μεταβληθεί. Πρέπει να προσέξουμε, όμως, η τυχαία αυτή τιμή σε απόλυτο να είναι μεγαλύτερη του 10^{-5}, καθώς, σε αντίθετη περίπτωση, θα είμαστε κάτω από το όριο του κριτηρίου που έχουμε αναφέρει στην εκφώνηση, οπότε η επαναληπτική διαδικασία *δεν* θα ξεκινήσει! Για τον λόγο αυτό, επιλέγουμε term=1.
- Τη μεταβλητή s την αρχικοποιούμε με το ουδέτερο στοιχείο για την πρόσθεση, δηλαδή την s=0.

Τέλος, θα χρειαστούμε μία μεταβλητή «μετρητή» η οποία θα χρησιμοποιείται ως εκθέτης στην έκφραση $|x^n|$ και θα μεγαλώνει κατά μία μονάδα σε κάθε επανάληψη (π.χ. n), την οποία αρχικοποιούμε με μηδέν.

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να εκτελέσει μία επαναληπτική διαδικασία που θα προσθέτει διαδοχικούς όρους στο άθροισμα s. Συγκεκριμένα, σε κάθε επανάληψη θα υπολογίζει την τιμή της μεταβλητής term και, στη συνέχεια, θα την προσθέτει στο τρέχον άθροισμα s. Η επαναληπτική διαδικασία θα ολοκληρώνεται όταν η απόλυτη τιμή του term γίνει μικρότερη από 10^{-5}, που είναι το όριο για το υπόλοιπο.

Επαλήθευση: Στο συγκεκριμένο παράδειγμα, μπορούμε να επιβεβαιώσουμε ότι έχουμε υπολογίσει σωστά το αποτέλεσμα, συγκρίνοντάς το με την ακριβή τιμή της συνάρτησης που θέλουμε να προσεγγίσουμε. Μπορούμε, λοιπόν, μετά το τέλος του κύριου κώδικα, να υπολογίσουμε την τιμή της $y = \frac{1}{1-x}$ στο $x = 0.5$ και να τη συγκρίνουμε με το αποτέλεσμα του πολυωνύμου McLaurin.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%MCLAURIN Calculate the McLaurin series of y = 1/(1-x)
% Initialization
clear all
x = 0.5;
term = 1;
s = 0;
n=0;

% Main code
while abs(term) >= 1.e-5
  term = x^n;
  s = s + term;
  n = n+1;
end
disp(['After ', num2str(n),' terms, the result is: '])
disp(s)

% Validation
disp('Exact value:')
disp(1/(1-x))
```

Η εκτέλεση του παραπάνω κώδικα επιστρέφει τα παρακάτω αποτελέσματα:

```
----------------------------------------
After 18 terms, the result is:
   1.999992370605469

Exact value:
     2
----------------------------------------
```

Από τα αποτελέσματα πληροφορούμαστε ότι χρησιμοποιήθηκαν 18 όροι και ότι η τιμή που υπολογίσαμε ισούται με 1.999992370605469, η οποία διαφέρει ελάχιστα από την πραγματική τιμή που είναι ίση με 2.

Άσκηση αυτοαξιολόγησης 4.4

Στη λύση του Παραδείγματος 4.6 επιλέξαμε να αρχικοποιήσουμε τις μεταβλητές term και s με μηδέν και να ξεκινήσουμε την επαναληπτική διαδικασία προσθέτοντας ως πρώτο όρο τη μονάδα, δηλαδή έναν όρο μηδενικού βαθμού $x^0 = 1$. Εναλλακτικά, θα μπορούσαμε να έχουμε αρχικοποιήσει τη μεταβλητή s με τον όρο μηδενικού βαθμού, δηλαδή να θέσουμε s=1, και να ξεκινήσουμε την επαναληπτική διαδικασία με τον όρο πρώτου βαθμού x.
Δοκιμάστε να μετατρέψετε τη λύση του Παραδείγματος 4.6, ώστε να λειτουργεί με αρχική τιμή s=1.
Υπόδειξη: Για να λειτουργήσει σωστά το πρόγραμμα, θα πρέπει να προσαρμόσετε κατάλληλα και τον «μετρητή».

Είδαμε στα προηγούμενα παραδείγματα διάφορες εφαρμογές της δομής `while`, οι οποίες είχαν κοινό χαρακτηριστικό το ότι δεν ήταν εκ των προτέρων καθορισμένος ο αριθμός των επαναλήψεων. Αυτό, άλλωστε, αναφέραμε στην αρχή της παραγράφου ως το κριτήριο που καθορίζει αν θα επιλέξουμε τη δομή `while` έναντι της δομής `for`. Ωστόσο, πρέπει να σημειώσουμε ότι η επιλογή αυτή δεν είναι υποχρεωτική. Παραδείγματος χάριν, με κατάλληλες αλλαγές στον κώδικα κάθε δομή `for` μπορεί να μετατραπεί σε βρόχο `while`. Ας δούμε ένα απλό παράδειγμα τέτοιας μετατροπής:

Παράδειγμα 4.7

Στο Παράδειγμα 4.1 δημιουργήσαμε ένα αρχείο script που προσομοίωνε τη ρίψη ενός ζαριού 5 φορές. Η επαναληπτική διαδικασία έγινε με την εντολή `for`.
Κάντε τις κατάλληλες αλλαγές στον κώδικα, ώστε να μετατραπεί σε επαναληπτική διαδικασία `while`.

Λύση Παραδείγματος 4.7

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μόνο μία μεταβλητή, τον «μετρητή» (π.χ. i), την οποία θα αρχικοποιήσουμε με την τιμή μηδέν.

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να εκτελεί έναν βρόχο, όπου σε κάθε επανάληψη θα εκτυπώνεται ένας τυχαίος αριθμός από το 1 έως το 6 (εντολή `randi(6)`) και θα αυξάνεται ο «μετρητής» κατά μία μονάδα. Η επαναληπτική διαδικασία θα τερματίζεται μόλις ο «μετρητής» πάρει την τιμή 5.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %RANDOM_NUMBER Display 5 random numbers
2  % Initialization
3   clear all
4   i = 0;
5
6  % Main code
7   while i<5
8     disp(randi(6))
9     i = i + 1;
10  end
```

4.4 Εμφωλευμένες επαναλήψεις

Είδαμε στο προηγούμενο Κεφάλαιο ότι η έννοια «εμφωλευμένες» επιλογές χρησιμοποιείται όταν μία δομή επιλογής περιέχεται μέσα σε μία άλλη. Κάτι τέτοιο συμβαίνει και με τις «εμφωλευμένες» επαναληπτικές δομές, όπου μία επαναληπτική δομή περιέχεται μέσα σε μία άλλη. Η χρησιμότητα ενός τέτοιου συνδυασμού επαναλήψεων εμφανίζεται όταν έχουμε περισσότερες από μία μεταβλητές, τις οποίες θέλουμε να υποβάλουμε σε συνδυασμένες επαναληπτικες διαδικασίες. Οι μεταβλητές αυτές μπορεί να αντιστοιχούν σε όμοια αντικείμενα (π.χ. «μετρητές», μεταβλητές που περιγράφουν χρόνο ή μήκος κ.λπ.) ή να περιέχουν εντελώς διαφορετικά μεταξύ τους χαρακτηριστικά.

Για παράδειγμα, ας υποθέσουμε ότι θέλουμε να δημιουργήσουμε μία εφαρμογή για ένα ηλεκτρονικό κατάστημα υποδημάτων, η οποία να εμφανίζει για κάθε είδος (κωδικό) παπουτσιού τα μεγέθη στα οποία υπάρχει διαθέσιμο. Για να υλοποιήσουμε ένα τέτοιο πρόγραμμα, θα πρέπει να φτιάξουμε μία επαναληπτική διαδικασία η οποία να διατρέχει όλους τους κωδικούς παπουτσιών του καταστήματος. Σε κάθε επανάληψη θα πρέπει να ελέγχει όλα τα μεγέθη του συγκεκριμένου κωδικού, για να εντοπίσει ποια από αυτά είναι διαθέσιμα. Με άλλα λόγια, για να ελέγξουμε όλα τα μεγέθη όλων των κωδικών υποδημάτων, η εξωτερική επαναληπτική διαδικασία (κωδικοί υποδημάτων), θα πρέπει να περιέχει μία εσωτερική (μεγέθη υποδημάτων), η οποία ονομάζεται εμφωλευμένη.

Οι εμφωλευμένες επαναλήψεις μπορεί να χρησιμοποιούν είτε την εντολή `for`, είτε τη `while`, είτε συνδυασμό αυτών. Για να κατανοήσουμε το πώς υλοποιούμε τις εμφωλευμένες επαναλήψεις, ας δούμε στον Κώδικα 4.7 τον γενικό τρόπο σύνταξης μίας τέτοιας διπλής δομής `for`, όπου κάθε βήμα της επαναληπτικής διαδικασίας της Μεταβλητής Α περιέχει έναν πλήρη βρόχο της Μεταβλητής Β.

```
1  for Μεταβλητή = min A : βήμα : max A
2
3      for Μεταβλητή = min B : βήμα : max B
4
5          ... Εντολές ...
6
7      end
8
9  end
```

Κώδικας 4.7: Σύνταξη εμφωλευμένου `for`.

Παρατηρείστε ότι, όπως και στην περίπτωση των εμφωλευμένων επιλογών, πρέπει να τηρείται η σειρά στον τερματισμό των επαναληπτικών δομών, δηλαδή πρέπει η εσωτερική δομή να τερματίζεται εντός της εξωτερικής και να μην διασταυρώνεται με άλλες δομές. Ο κανόνας αυτός ισχύει και για συνδυασμό δομής επιλογής (π.χ. `if`) με επαναληπτική διαδικασία (π.χ. `for`), όπως θα δούμε στα παραδείγματα που θα ακολουθήσουν.

Για να εντοπίζουμε εύκολα την αρχή και το τέλος μίας επαναληπτικής δομής, είναι χρήσιμο να δημιουργούμε «κλιμακωτή» μορφή στη γραφή του κώδικα, δηλαδή να αφήνουμε μεγαλύτερο περιθώριο από το αριστερό άκρο του κειμενογράφου για κάθε ομάδα εντολών που είναι εμφωλευμένη σε μία άλλη. Αυτό φαίνεται στο παράδειγμα σύνταξης του Κώδικα 4.7, όπου οι εντολές της επαναληπτικής διαδικασίας της Μεταβλητής Β βρίσκονται «πιο μέσα» σε σχέση με αυτές της Μεταβλητής Α.

Παράδειγμα 4.8

Δημιουργήστε ένα αρχείο script που θα υπολογίζει τον μέσο όρο των βαθμών πέντε μαθημάτων για πέντε διαφορετικούς φοιτητές. Το πρόγραμμα θα δέχεται ως είσοδο τον Αριθμό Μητρώου (Α.Μ.) και

τους βαθμούς κάθε φοιτητή και θα υπολογίζει τον μέσο όρο τους.

Λύση Παραδείγματος 4.8

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μία μεταβλητή στην οποία θα αποθηκεύεται ο Αριθμός Μητρώου (π.χ. ΑΜ), μία μεταβλητή στην οποία θα αποθηκεύεται ο κάθε βαθμός που εισάγει ο χρήστης (π.χ. vathmos) και μία μεταβλητή για το τρέχον άθροισμα (π.χ. s). Οι μεταβλητές vathmos και ΑΜ ορίζονται κατευθείαν από τον χρήστη, οπότε δεν παίζει ρόλο η τιμή με την οποία θα αρχικοποιηθούν. Για τον λόγο αυτό, μπορούμε να τις αρχικοποιήσουμε με μηδέν. Με μηδέν θα επιλέξουμε να αρχικοποιήσουμε και τη μεταβλητή s η οποία περιλαμβάνει το τρέχον άθροισμα, καθώς το μηδέν είναι το ουδέτερο στοιχείο της πρόσθεσης. Τέλος, θα χρειαστούμε δύο «μετρητές», έναν για να μετράει τον αριθμό των βαθμών και έναν για το πλήθος των Α.Μ.

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να εκτελεί μία επαναληπτική διαδικασία `for`, όπου θα μεταβάλλεται ένας «μετρητής» από το 1 έως το 5, ο οποίος αντιπροσωπεύει τους πέντε φοιτητές. Σε κάθε επανάληψη, το πρόγραμμα θα πρέπει να ζητάει από τον χρήστη να εισάγει το Α.Μ. και, στη συνέχεια, να εκτελεί μία δεύτερη (εμφωλευμένη) επαναληπτική διαδικασία `for`, όπου θα μεταβάλλεται ο δεύτερος «μετρητής» από το 1 έως το 5, ώστε να εισάγει ο χρήστης τους 5 βαθμούς. Μόλις ολοκληρωθεί η εμφωλευμένη επαναληπτική διαδικασία, θα υπολογίζεται και θα εκτυπώνεται ο μέσος όρος για τον συγκεκριμένο φοιτητή. Μετά την εκτύπωση του αποτελέσματος, η μεταβλητή s θα πρέπει να επαναρχικοποιείται με μηδέν, ώστε να είναι έτοιμη για τον υπολογισμό του νέου μέσου όρου. Το πρόγραμμα θα τερματίζεται μόλις ολοκληρωθεί η εξωτερική επαναληπτική διαδικασία για τους πέντε φοιτητές.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %MEAN_GRADE_5 Calculate the grades of 5 students
2  % Initialization
3   clear all
4  AM = 0;
5  vathmos = 0;
6  s = 0;
7
8  % Main code
9   for i = 1:5
10    AM = input('Aritmos Mhtrwou: ');
11    for j = 1:5
12      vathmos = input([num2str(j),'os Vathmos: ']);
13      s = s + vathmos;
14    end
15    disp(['Mesos Oros: ',num2str(s/5)])
16    s = 0;
17  end
```

Παρατηρήστε ότι η μεταβλητή s επαναρχικοποιείται με μηδέν στη γραμμή 16 (στο τέλος της εσωτερικής επανάληψης), έτσι ώστε να μπορεί να χρησιμοποιηθεί ξανά για τον υπολογισμό του μέσου όρου του επόμενου φοιτητή.

Όπως αναφέρθηκε και στην αρχή της ενότητας, οι εμφωλευμένες επαναληπτικές διαδικασίες μπορεί να περιλαμβάνουν βρόχους `for`, `while` ή και συνδυασμό αυτών. Επίσης, σε πολλές περιπτώσεις οι

αναγκαίες εμφωλευμένες διαδικασίες μπορεί να είναι περισσότερες από δύο. Για παράδειγμα, στα φυσικά φαινόμενα που συμβαίνουν στον τριδιάστατο χώρο, οι διαφορετικές μεταβλητές που θέλουμε να υποβάλουμε σε επαναληπτική διαδικασία είναι τρεις, όσες και οι διαστάσεις του προβλήματος. Επίσης, αν το φαινόμενο που μελετάμε δεν είναι σταθερό στον χρόνο, τότε πιθανώς να χρειάζεται και τέταρτη επαναληπτική διαδικασία κ.ο.κ. Βέβαια, για τις ανάγκες του συγγράμματος αυτού, δεν πρόκειται να ασχοληθούμε με τόσο πολύπλοκα φαινόμενα. Θα περιοριστούμε σε απλές εφαρμογές, έτσι ώστε να κατανοήσουμε τις βασικές αρχές του προγραμματισμού σε σχέση με τις εμφωλευμένες δομές.

Ένα θέμα που θα μας απασχολήσει στο πλαίσιο αυτό είναι η επεξεργασία κωδίκων που έχουμε αναπτύξει σε προηγούμενα Παραδείγματα, με σκοπό να τους κάνουμε πιο πλήρεις και λειτουργικούς. Συγκεκριμένα, όταν έχουμε να λύσουμε ένα πολύπλοκο πρόβλημα με πολλές διαδικασίες και πλήθος παραμέτρων, προσπαθούμε να το ανάξουμε σε όσο το δυνατόν περισσότερα, απλά και εύκολα διαχειρίσιμα υποπροβλήματα. Μόλις ολοκληρώσουμε την κατάστρωση και, στη συνέχεια, την υλοποίηση των επιμέρους προβλημάτων, τότε μπορούμε να τα συνθέσουμε, ώστε να αναπαράξουμε το συνολικό πρόβλημα που θέλουμε να λύσουμε. Για να γίνει κατανοητή αυτή η διαδικασία αναγωγής ενός σύνθετου προβλήματος σε πιο απλά υποπροβλήματα, θα δούμε στη συνέχεια μερικά παραδείγματα:

Παράδειγμα 4.9

Δημιουργήστε ένα αρχείο script που θα υπολογίζει τον μέσο όρο των βαθμών πέντε μαθημάτων για πέντε διαφορετικούς φοιτητές. Το πρόγραμμα θα δέχεται ως είσοδο τον Αριθμό Μητρώου (Α.Μ.) και τους βαθμούς κάθε φοιτητή και θα υπολογίζει τον μέσο όρο τους. Επίσης, το πρόγραμμα θα πρέπει να εξασφαλίζει ότι η τιμή της βαθμολογίας που εισήγαγε ο χρήστης είναι ένας ακέραιος αριθμός από το 0 έως το 10.

Λύση Παραδείγματος 4.9

Ανάλυση του προβλήματος: Παρατηρούμε ότι το συγκεκριμένο πρόβλημα είναι ένας συνδυασμός του Παραδείγματος 4.8, το οποίο αφορούσε την εύρεση του μέσου όρου πέντε βαθμών για πέντε φοιτητές, και του Παραδείγματος 4.4, που αφορούσε τον έλεγχο των βαθμών ώστε να είναι θετικοί ακέραιοι.
Αν προσπαθήσουμε να δημιουργήσουμε εξαρχής τη λύση του συνολικού προβλήματος, είναι πιθανό να δυσκολευτούμε λόγω της πολυπλοκότητας. Αν όμως συνθέσουμε τις λύσεις των Προβλημάτων 4.4 και 4.8, τότε είναι πιθανό να φτάσουμε στην επιθυμητή λύση χωρίς μεγάλη δυσκολία. Ας θυμηθούμε τη λύση του Παραδείγματος 4.8, που αφορούσε τον μέσο όρο πέντε βαθμών για πέντε φοιτητές.

```
1  %MEAN_GRADE_5 Calculate the grades of 5 students
2  % Initialization
3   clear all
4  AM = 0;
5   vathmos = 0;
6   s = 0;
7
8  % Main code
9   for i = 1:5
10    AM = input('Aritmos Mhtrwou: ');
11    for j = 1:5
12      vathmos = input([num2str(j),'os Vathmos: ']);
13      s = s + vathmos;
14    end
15    disp(['Mesos Oros: ',num2str(s/5)])
16    s = 0;
17  end
```

Κώδικας 4.8: Λύση του προβλήματος που αφορά τον μέσο όρο πέντε βαθμών για πέντε φοιτητές.

Ο κώδικας αυτός εκτελεί δύο επαναληπτικές διαδικασίες, η μία από τις οποίες είναι εμφωλευμένη μέσα στην άλλη. Η εξωτερική διαδικασία που γίνεται με τη μεταβλητή i αφορά τους φοιτητές, ενώ η εσωτερική (εμφωλευμένη) που γίνεται με τη μεταβλητή j αφορά την εισαγωγή των βαθμών και τον υπολογισμό του μέσου όρου. Αυτό που πρέπει να προστεθεί στον παραπάνω κώδικα, για να λύσουμε την άσκησή μας, είναι ο επαναλαμβανόμενος έλεγχος για το αν οι βαθμοί που εισάγει ο χρήστης είναι θετικοί ακέραιοι. Ο έλεγχος αυτός έχει υλοποιηθεί στη λύση του Παραδείγματος 4.4, η οποία παρουσιάζεται παρακάτω.

```
1  %CHECK_GRADE Check id a grade is valid
2  % Initialization
3   clear all
4   vathmos = input('Parakalw, dwste to vathmo: ');
5
6   % Main code
7   while vathmos<0 || vathmos>10 || round(vathmos)~=vathmos
8     disp('Mh egkyros vatmos')
9     vathmos = input('Parakalw, dwste EGKYRO vathmo: ');
10  end
```

Κώδικας 4.9: Λύση του προβλήματος που αφορά τον έλεγχο εγκυρότητας των βαθμών.

Η λύση είναι ένας βρόχος `while`, ο οποίος ελέγχει αν ο βαθμός που εισήγαγε ο χρήστης είναι έγκυρος. Το ερώτημα που πρέπει, λοιπόν, να απαντήσουμε είναι πώς θα συνδυάσουμε τις δύο αυτές λύσεις για να έχουμε το επιθυμητό αποτέλεσμα.

Παρατηρούμε ότι η εντολή που ζητάει από τον χρήστη να εισάγει τον βαθμό στον Κώδικα 4.8 είναι η `input` που βρίσκεται στη γραμμή 12. Αν ο χρήστης εισάγει μη-έγκυρο αριθμό στην εντολή αυτή, τότε ο υπολογισμός του μέσου όρου θα είναι λανθασμένος. Πρέπει, λοιπόν, αμέσως μετά την εισαγωγή του βαθμού να γίνεται ο έλεγχος που έχουμε υλοποιήσει στον Κώδικα 4.9. Εφόσον το όνομα της μεταβλητής που θα ελέγξουμε είναι κοινό στα δύο προγράμματα, δηλαδή vathmos, μπορούμε να εισάγουμε κατευθείαν τον βρόχο `while` στην κατάλληλη θέση. Τοποθετούμε, δηλαδή, τις γραμμές 7 έως 9 του Κώδικα 4.9 αμέσως μετά τη γραμμή 11 του Κώδικα 4.8.

Το αποτέλεσμα παρουσιάζεται στον κώδικα που ακολουθεί:

```
1  %MEAN_GRADE_CHECK Chech the A.M. and
2  % calculate the mean grades of 5 students
3
4  % Initialization
5   clear all
6   AM = 0;
7   vathmos = 0;
8   s = 0;
9
10  % Main code
11  for i = 1:5
12    AM = input('Aritmos Mhtrwou: ');
13    for j = 1:5
14      vathmos = input([num2str(j),'os Vathmos: ']);
15
```

```
16        % Elegxos vathmou
17        while vathmos<0 || vathmos>10 || round(vathmos)~=vathmos
18          disp('Mh egkyros vatmos')
19          vathmos = input('Parakalw, dwste EGKYRO vathmo: ');
20        end
21
22        s = s + vathmos;
23      end
24      disp(['Mesos Oros: ',num2str(s/5)])
25      s = 0;
26    end
```

Είδαμε στο προηγούμενο παράδειγμα πώς μπορούμε να συνδυάσουμε δύο κώδικες που εκτελούν συγκεκριμένες λειτουργίες, προκειμένου να δημιουργήσουμε ένα σύνθετο πρόγραμμα. Με τη στρατηγική αυτή πετυχαίνουμε δύο στόχους:

- Κατασκευάζουμε μικρούς κώδικες, τους οποίους μπορούμε εύκολα να ελέγξουμε τόσο για συντακτικά όσο και για λογικά σφάλματα. Με τον τρόπο αυτό εξασφαλίζουμε ότι τα επί μέρους τμήματα του προγράμματός μας λειτουργούν σωστά.
- Δημιουργούμε μία «δεξαμενή» μικρών προγραμμάτων που εκτελούν συγκεκριμένες λειτουργίες. Οι λειτουργίες αυτές μπορεί να είναι χρήσιμες για προβλήματα με εντελώς διαφορετικό αντικείμενο (π.χ. ο έλεγχος για θετικούς ακέραιους αριθμούς εμφανίζεται στο παράδειγμα με τους βαθμούς αλλά θα τον συναντήσουμε και σε ασκήσεις με πίνακες). Μπορούμε, λοιπόν, να χρησιμοποιούμε τα μικρά αυτά προγράμματα για να κατασκευάσουμε έναν γενικότερο, σύνθετο κώδικα, χωρίς να χρειάζεται να προγραμματίσουμε όλες τις λειτουργίες από την αρχή. Η ανάγκη αυτή επαναχρησιμοποίησης υπολογιστικών «εργαλείων» είναι η βάση για την ανάπτυξη των συναρτήσεων, τις οποίες θα συζητήσουμε σε επόμενο Κεφάλαιο.

4.5 Η εντολή break

Οι εντολές που μας έχουν απασχολήσει μέχρι το σημείο αυτό αφορούσαν τη δημιουργία μιας επαναληπτικής διαδικασίας. Ας δούμε τώρα μια εντολή που μας επιτρέπει να διακόψουμε μία επαναληπτική διαδικασία, κατά τη διάρκεια που αυτή βρίσκεται σε εξέλιξη.

Ας υποθέσουμε ότι έχουμε δημιουργήσει ένα πρόγραμμα το οποίο ελέγχει έναν πίνακα με πειραματικά αποτελέσματα μέχρι να βρεθεί μία συγκεκριμένη τιμή. Για να γίνει αυτό, το πρόγραμμα θα πρέπει να εκτελεί μία επαναληπτική διαδικασία που να διατρέχει όλα τα στοιχεία του πίνακα, μέχρι να εντοπιστεί η τιμή που ψάχνουμε. Μόλις εντοπιστεί η τιμή αυτή, θέλουμε η επαναληπτική διαδικασία να διακόπτεται αυτόματα και το πρόγραμμα να προχωράει στις εντολές που ακολουθούν. Για να γίνει ο τερματισμός της επαναληπτικής διαδικασίας από κάποιο εσωτερικό της σημείο, μπορούμε να χρησιμοποιήσουμε την εντολή `break`.

Η εντολή `break` χρησιμοποιείται σε μία δομή επανάληψης, για να διακόψει την επαναληπτική διαδικασία προτού αυτή ολοκληρωθεί. Συνήθως, συνδυάζεται με κάποιον έλεγχο (π.χ. `if`), ο οποίος καθορίζει τη συνθήκη υπό την οποία θα διακοπεί ο βρόχος. Ας δούμε ένα παράδειγμα:

Παράδειγμα 4.10

Δημιουργήστε ένα αρχείο script που να προσομοιώνει τη ρίψη ενός ζαριού και θα πληροφορεί τον χρήστη πόσες φορές το ζάρι ήρθε έξι. Θα πρέπει να προσομοιώνονται συνολικά πέντε ρίψεις του ζαριού, εκτός και αν το ζάρι έρθει ένα, οπότε η επαναληπτική διαδικασία θα τερματίζεται αυτόματα.

Λύση Παραδείγματος 4.10

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μία μεταβλητή που να περιέχει την τιμή του ζαριού (π.χ. zari) και μία μεταβλητή που θα μετράει πόσες φορές έχει έρθει έξι (π.χ. eksaria). Σε σχέση με την αρχικοποίηση η μεταβλητή zari θα λάβει τιμή από τη γεννήτρια τυχαίων αριθμών randi(), οπότε μπορούμε αυθαίρετα να την θέσουμε αρχικά ίση με μηδέν. Η μεταβλητή eksaria πρέπει επίσης να αρχικοποιηθεί με μηδέν, καθώς αυτό είναι το ουδέτερο στοιχείο της πρόσθεσης. Τέλος, θα χρειαστούμε και μία μεταβλητή «μετρητή» (π.χ. i), η οποία θα οριστεί κατευθείαν στον επαναληπτικό βρόχο.

Κύριος κώδικας: Για το κύριο πρόγραμμα θα χρειαστούμε μία επαναληπτική δομή, στην οποία η μεταβλητή eksaria θα αυξάνεται κατά μία μονάδα, όταν το zari γίνεται έξι. Σύμφωνα με αυτά που έχουμε αναφέρει στις προηγούμενες ενότητες, η κατάλληλη δομή επανάληψης για τη λειτουργία αυτή είναι η `for`, καθώς από την εκφώνηση έχουμε συγκεκριμένο αριθμό επαναλήψεων (5). Ωστόσο, η εκφώνηση μάς λέει ότι η δομή αυτή θα πρέπει να διακόπτεται, αν το zari λάβει την τιμή ένα. Για να γίνει αυτό, θα πρέπει να περιλάβουμε στον κώδικα έναν έλεγχο `if`, ο οποίος, όταν εντοπίσει ότι η μεταβλητή zari έχει γίνει ένα, θα διακόπτει άμεσα την επαναληπτική διαδικασία με την εντολή `break`.
Το πρόγραμμα θα ολοκληρώνεται τυπώνοντας την τιμή της μεταβλητής eksaria και το πλήθος των φορών που εκτελείται η διαδικασία της ρίψης του ζαριού.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %DICE_6 Count how many times the dice is 6
2  % Initialization
3  clear all
4  zari = 0;
5  eksaria = 0;
6
7  % Main code
8  for i=1:5
9    zari = randi(6);
10   % Count how many times zari is 6
11   if zari == 6
12     eksaria = eksaria + 1;
13   end
14
15   % Stop the program when zari is 1
16   if zari == 1
17     break
18   end
19
20 end
21
22 disp(['To 6 emfanistike ',num2str(eksaria),' fores'])
```

Παρατήρηση 4.6

Στο προηγούμενο Παράδειγμα, η επαναληπτική διαδικασία είχε δύο χαρακτηριστικά:

- Συγκεκριμένη διάρκεια (πέντε επαναλήψεις).
- Συνθήκη τερματισμού (το zari να πάρει την τιμή ένα).

Στη λύση του Παραδείγματος επιλέξαμε να χρησιμοποιήσουμε έναν συνδυασμό βρόχου `for` με την εντολή διακοπής των επαναλήψεων `break`. Εναλλακτικά, θα μπορούσαμε να αποφύγουμε τον συνδυασμό `for`-`break`, χρησιμοποιώντας μία δομή επανάληψης `while`. Στην περίπτωση αυτή, η συνθήκη της εντολής `while` θα έπρεπε να συνδυάζει τα δύο κριτήρια που προαναφέρθηκαν, αξιοποιώντας κατάλληλους ελέγχους.
Ως γενική παρατήρηση, θα μπορούσαμε να πούμε ότι το `break` μπορεί να αποφευχθεί με τη χρήση μιας δομής `while` με κατάλληλη συνθήκη. Ο λόγος που επιδιώκουμε να αποφεύγουμε τη χρήση του `break` είναι ότι επιφέρει αλλαγές στη ροή μιας επαναληπτικής διαδικασίας από το εσωτερικό της διαδικασίας αυτής. Έτσι, δεν έχουμε εποπτεία του αριθμού των επαναλήψεων κοιτάζοντας απλώς την αρχική εντολή. Με άλλα λόγια, δεν μπορούμε να αντιληφθούμε τη διάρκεια του βρόχου «με μια ματιά».
Όταν ο κώδικας είναι σύνθετος, η ύπαρξη ενός ή περισσότερων «διακοπτών» `break` σε διάφορα σημεία ενός βρόχου μπορεί να οδηγήσει σε λογικά λάθη και να αλλοιώσει τη λειτουργία του προγράμματος. Σε αυτές τις περιπτώσεις, η αποφυγή του `break` με χρήση μίας κατάλληλα διαμορφωμένης δομής `while` είναι καλή πρακτική. Ωστόσο, αν υπάρχει ανάγκη να διακοπεί αυτόματα ο βρόχος, όπως π.χ. αν θέλουμε να περιορίσουμε την επίδραση μιας μεταβλητής που κατά τη διάρκεια των πράξεων έχει απειριστεί, τότε η εντολή `break` είναι πολύ χρήσιμη.

Άσκηση αυτοαξιολόγησης 4.5

Στη λύση του Παραδείγματος 4.10 χρησιμοποιήσαμε έναν συνδυασμό `for` και `break` για να λύσουμε το πρόβλημα.
Δοκιμάστε να φτιάξετε μία εναλλακτική λύση, χωρίς χρήση της εντολής `break`. Η λύση θα πρέπει να βασίζεται σε μια επαναληπτική δομή `while` σε συνδυασμό με κατάλληλη συνθήκη.

Μία χρήσιμη πληροφορία σε σχέση με την εντολή `break` είναι ότι, αν εφαρμοστεί σε εμφωλευμένες επαναληπτικές διαδικασίες, τότε διακόπτει τον βρόχο στον οποίο αντιστοιχεί. Αν, δηλαδή, είναι μέρος του εσωτερικού βρόχου, τότε το `break` τερματίζει μόνο τον βρόχο αυτόν, ενώ ο εξωτερικός βρόχος συνεχίζεται κανονικά. Αν όμως είναι μέρος των εντολών του εξωτερικού βρόχου, τότε το `break` σταματάει τη συνολική επαναληπτική διαδικασία και οδηγεί στις εντολές που την ακολουθούν. Ας δούμε ένα σχετικό παράδειγμα:

Παράδειγμα 4.11

Δημιουργήστε ένα αρχείο script που να εκτυπώνει κατά σειρά όλα τα πολλαπλάσια του 1, του 2, του 3, του 4 και του 5, τα οποία είναι μικρότερα ή ίσα του 10.

Λύση Παραδείγματος 4.11

Αρχικοποίηση μεταβλητών: Στη λύση του Παραδείγματος αυτού, θα χρησιμοποιήσουμε δύο βρόχους `for`. Οι μεταβλητές που θα χρειαστούμε είναι αυτές που θα χρησιμοποιηθούν σαν «μετρητές» στους δύο βρόχους, π.χ. i και j. Η αρχικοποίηση των μεταβλητών αυτών δεν είναι απαραίτητη, καθώς θα οριστούν κατευθείαν στους βρόχους `for`.

Κύριος κώδικας: Για να υπολογιστούν τα πολλαπλάσια των 5 αριθμών, θα χρειαστούμε μία επαναληπτική διαδικασία που να διατρέχει τους αριθμούς αυτούς. Αφού το πλήθος των αριθμών είναι δεδομένο, μπορούμε να χρησιμοποιήσουμε την εντολή `for` με «μετρητή» τη μεταβλητή i. Σε κάθε επανάληψη

της διαδικασίας αυτής θέλουμε να εκτυπώνονται όλα τα πολλαπλάσια του δεδομένου αριθμού i. Άρα θα χρειαστούμε μία δεύτερη επαναληπτική διαδικασία `for` με «μετρητή» τη μεταβλητή j, η οποία θα είναι εμφωλευμένη στην πρώτη και θα εκτυπώνει τα γινόμενα 1*i, 2*i, 3*i και, γενικά, j*i. Η εσωτερική επαναληπτική διαδικασία θα τερματίζεται, όταν η τιμή του γινομένου γίνει μεγαλύτερη ή ίση του 10 με χρήση της εντολής `break`.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%MULTIPLES Find the multiples of 1,2,3,4 and 5
% Initialization
clear all

% Main code
% Outer loop
for i=1:5
  disp(['Pollaplasia tou ',num2str(i)])

  % Inner loop
  for j=1:10
    disp(i*j)

    % Condition for stopping the inner loop
    if i*j>=10
      break
    end

  end

end
```

4.6 Η εντολή continue

Μία ακόμα εντολή που επιτρέπει τη διαχείριση μίας επαναληπτικής διαδικασίας από το εσωτερικό της είναι η εντολή `continue`. Με τη χρήση του `continue` σε κάποιο εσωτερικό σημείο ενός βρόχου, μπορούμε να παρακάμψουμε ό,τι ακολουθεί την εντολή αυτή και να προχωρήσουμε κατευθείαν στο επόμενο βήμα. Με άλλα λόγια, η εντολή `continue` μπορεί να γίνει αντιληπτή σαν ένα «άλμα» που οδηγεί αυτόματα στο επόμενο βήμα της δομής επανάληψης.

Για να κατανοήσουμε τη λειτουργία της, ας υποθέσουμε ότι έχουμε δημιουργήσει ένα πρόγραμμα που μέσα από μία επαναληπτική διαδικασία επεξεργάζεται ένα μεγάλο πλήθος πειραματικών δεδομένων, π.χ. υπολογίζει τη μέση τιμή, την τυπική απόκλιση κ.λπ. Αν στα δεδομένα αυτά υπάρχει κάποια τιμή η οποία έχει προκύψει από εσφαλμένη μέτρηση και είναι σαφώς εκτός ορίων (ή ακόμα και άπειρη), τότε υπάρχει ο κίνδυνος να αλλοιώσει τα αποτελέσματά μας. Στην περίπτωση που εντοπιστεί μία τέτοια τιμή, θέλουμε ο κώδικας να αγνοήσει πράξεις και υπολογισμούς που γίνονται με την τιμή αυτή και να προχωρήσει κατευθείαν στη επόμενη επανάληψη. Η εντολή `continue` έχει αυτή ακριβώς τη λειτουργία, δηλαδή διακόπτει την τρέχουσα επανάληψη και προχωράει στην επόμενη.

Πρέπει να σημειώσουμε ότι, όπως και η εντολή `break`, έτσι και η εντολή `continue` επηρεάζει τον βρόχο στον οποίο αντιστοιχεί. Αν, λοιπόν, εφαρμοστεί σε εμφωλευμένη επαναληπτική διαδικασία, τότε θα προχωρήσει στην επόμενη επανάληψη μόνο τη διαδικασία αυτή, χωρίς να επηρεάσει τον εξωτερικό βρόχο. Αν, όμως, είναι μέρος των εντολών του εξωτερικού βρόχου, θα προχωρήσει στην επόμενη επανάληψη τον βρόχο αυτόν.

Ας δούμε κάποιες εφαρμογές για να κατανοήσουμε τη χρήση της εντολής `continue`.

Παράδειγμα 4.12

Δημιουργήστε ένα αρχείο script που να υπολογίζει το άθροισμα και το γινόμενο όλων των πρώτων αριθμών που υπάρχουν στο διάστημα από το 1 έως το 20.

Λύση Παραδείγματος 4.12

Αρχικοποίηση μεταβλητών: Θα χρειαστούμε μία μεταβλητή για το άθροισμα που πρέπει να υπολογίσουμε (π.χ. athroisma), την οποία θα αρχικοποιήσουμε με το ουδέτερο στοιχείο της πρόσθεσης, δηλαδή με μηδέν. Θα χρειαστούμε, επίσης, μία μεταβλητή για το γινόμενο (π.χ. ginomeno), την οποία θα αρχικοποιήσουμε με το ουδέτερο στοιχείο του πολλαπλασιασμού, δηλαδή με ένα. Θα χρησιμοποιήσουμε, τέλος, και μία μεταβλητή ως «μετρητή» για την επαναληπτική διαδικασία της εντολής `for`, την οποία δεν χρειάζεται να αρχικοποιήσουμε.

Κύριος κώδικας: Η επαναληπτική διαδικασία που θα χρησιμοποιήσουμε θα υλοποιηθεί με την εντολή `for`, αφού είναι γνωστός ο αριθμός των επαναλήψεων, και ο «μετρητής» (π.χ. i) θα διατρέχει τις τιμές από το 1 έως το 20. Σε κάθε επανάληψη θα πρέπει να προσθέτουμε τον τρέχοντα αριθμό στο athroisma και να τον πολλαπλασιάζουμε με το ginomeno, μόνο αν αυτός είναι πρώτος. Σε αντίθετη περίπτωση, θα πρέπει το πρόγραμμα να αγνοεί αυτές τις εντολές με χρήση του `continue` και να προχωράει στην επόμενη επανάληψη.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

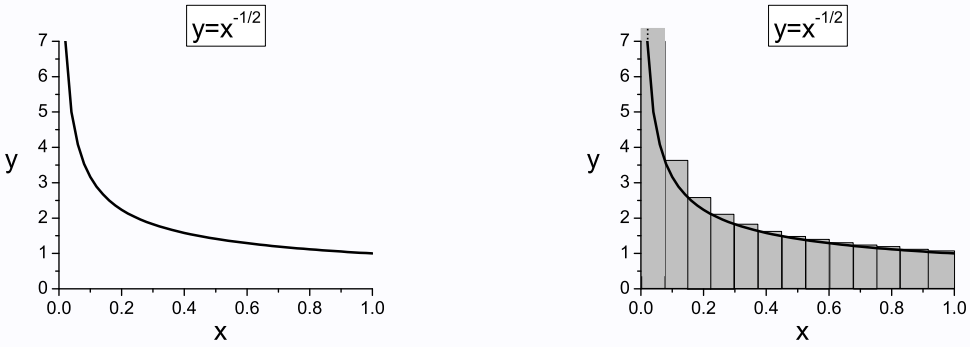
```
1  %CALC_PRIMES Calculate the sum and the product of prime numbers
2  % Initialization
3  clear all
4  athroisma = 0;
5  ginomeno = 1;
6
7  % Main code
8  for i=1:20
9
10   % Condition for non-prime numbers
11   if ~isprime(i)
12     continue
13   end
14
15   athroisma = athroisma + i;
16   ginomeno = ginomeno * i;
17
18 end
19 disp(['To athroisma einai: ',num2str(athroisma)])
20 disp(['To ginomeno einai: ',num2str(ginomeno)])
```

Άσκηση αυτοαξιολόγησης 4.6

Δοκιμάστε να φτιάξετε μία εναλλακτική λύση για το Παράδειγμα 4.12, χωρίς χρήση της εντολής `continue`.

Παράδειγμα 4.13

Δημιουργήστε ένα αρχείο script που θα υπολογίζει το εμβαδόν 100 παραλληλογράμμων που είναι περιγεγραμμένα στη γραφική παράσταση της συνάρτησης $y = 1/\sqrt{x}$ από το 0 έως το 1 (Σχήμα 4.4). Μπορείτε να συγκρίνετε το αποτέλεσμα με την ακριβή τιμή του εμβαδού, που είναι 2 (υπολογίζεται με βάση το ορισμένο ολοκλήρωμα $\int_0^1 = 1/\sqrt{x}dx$.).



Σχήμα 4.4: Υπολογισμός του εμβαδού κάτω από τη γραφική παράσταση της συνάρτησης $y = 1/\sqrt{x}$, με τη βοήθεια εμβαδών περιγεγραμμένων παραλληλογράμμων (Παράδειγμα 4.13).

Λύση Παραδείγματος 4.13

Για να λύσουμε αυτό το πρόβλημα, μπορούμε να βασιστούμε στη λύση του Παραδείγματος 4.3, που ζητάει την ίδια εργασία για άλλη συνάρτηση. Ωστόσο, έχουμε να αντιμετωπίσουμε τη δυσκολία που αφορά το πρώτο περιγεγραμμένο παραλληλόγραμμο: η γραφική παράσταση της $y = 1/\sqrt{x}$ απειρίζεται στο $x = 0$, με αποτέλεσμα να απειρίζεται και το εμβαδόν του παραλληλογράμμου αυτού (βλ. Σχήμα 4.4). Αν δεν αντιμετωπίσουμε το πρόβλημα αυτό, τότε το άθροισμα που υπολογίζουμε θα απειριστεί!
Ας δούμε πώς μπορούμε να αντιμετωπίσουμε ένα τέτοιο πρόβλημα με την εντολή `continue`.

Αρχικοποίηση μεταβλητών: Όπως και στο Παράδειγμα 4.3, θα χρειαστούμε δύο μεταβλητές στις οποίες θα αποθηκεύονται οι τιμές του x και του y (π.χ. x και y), τις οποίες αρχικοποιούμε με μηδέν. Θα χρειαστούμε, επίσης, μία μεταβλητή για το πλήθος των παραλληλογράμμων που θα χρησιμοποιήσουμε (π.χ. N), η οποία θα έχει την τιμή 100, και μια μεταβλητή, η οποία θα περιέχει το μήκος της βάσης κάθε παραλληλογράμμου (π.χ. dx) με τιμή $1/N$. Τέλος, θα χρειαστούμε μία μεταβλητή για το άθροισμα των εμβαδών των παραλληλογράμμων (π.χ. E), την οποία θα αρχικοποιήσουμε με το ουδέτερο στοιχείο της πρόσθεσης, δηλαδή με μηδέν.

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να έχει τις ίδιες εντολές με τον κώδικα του Παραδείγματος 4.3. Οι μόνες αλλαγές βρίσκονται στη συνάρτηση που θα χρησιμοποιήσουμε (εδώ θα είναι η $y = 1/\sqrt{x}$) και στον έλεγχο που θα εισάγουμε για την αποφυγή του απειρισμού. Η ποσότητα που απειρίζεται είναι το $y = 1/\sqrt{x}$, άρα και ο έλεγχός μας θα αφορά τη μεταβλητή αυτή. Συγκεκριμένα, θα πρέπει να εισαχθεί αμέσως μετά τον υπολογισμό του y και, αν εντοπίσει ότι η τιμή του έχει γίνει άπειρη (δηλαδή Inf), θα πρέπει να οδηγεί κατευθείαν στο επόμενο βήμα με την εντολή `continue`, χωρίς να προστίθεται η τιμή

Inf στο τρέχον άθροισμα E.
Επαλήθευση: Γνωρίζουμε από την εκφώνηση ότι το ακριβές εμβαδόν κάτω από τη γραφική παράσταση της $y = 1/\sqrt{x}$ είναι $E_{exact} = 2$, οπότε εκτυπώνουμε την πληροφορία αυτή.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %AREA_2 Calculate area below y = 1/sqrt(x) in [0,1]
2  % Initialization
3   clear all
4   x = 0;
5   y = 0;
6   N=1000;
7   dx = 1/N;
8   E = 0;
9
10  % Main code
11  for i=1:N
12    x = (i-1)*dx;
13    y = 1/sqrt(x);
14
15    % Check for infinity
16    if y == Inf
17      continue
18    end
19
20    % Calculation of Area
21    E = E + y*dx;
22  end
23
24  disp(['Methodos parallilogrammwn: ',num2str(E)])
25  disp(['Akribhs ypologismos: ',num2str(2)])
```

Ο παραπάνω κώδικας επιστρέφει τα αποτελέσματα

```
----------------------------------------
Methodos parallilogrammwn: 1.9533
Akribhs ypologismos: 2
----------------------------------------
```

από τα οποία διακρίνουμε ότι το σφάλμα στον υπολογισμό του εμβαδού είναι μικρότερο του 0.05.

Παρατήρηση 4.7

Για να επιβεβαιώσετε ότι, αν δεν είχε εξαιρεθεί το εμβαδόν του πρώτου παραλληλογράμμου από το άθροισμα, το αποτέλεσμα θα ήταν άπειρο, μπορείτε να απενεργοποιήσετε, δηλαδή να μετατρέψετε σε σχόλια, τις γραμμές 15-17 της λύσης του Παραδείγματος 4.13. Όταν εκτελέσετε τον κώδικα, θα παρατηρήσετε ότι το αποτέλεσμα είναι Inf!

4.7 Ασκήσεις

Άσκηση 4.1. *Δημιουργήστε ένα αρχείο script που θα εκτυπώνει τους δέκα πρώτους όρους της ακολουθίας Fibonacci. Η ακολουθία αυτή προκύπτει προσθέτοντας κάθε φορά τους δύο τελευταίους όρους, δηλαδή αν θεωρήσουμε πρώτο όρο τον $F_0 = 0$ και δεύτερο τον $F_1 = 1$, τότε προκύπτουν με τη σειρά:*

$$F_2 = F_1 + F_0 = 1 + 0 = 1$$

$$F_3 = F_2 + F_1 = 1 + 1 = 2$$

$$F_4 = F_3 + F_2 = 2 + 1 = 3$$

και, γενικά,

$$F_n = F_{n-1} + F_{n-2}$$

Άσκηση 4.2. *Δημιουργήστε ένα αρχείο script που να υπολογίζει προσεγγιστικά το e^x από τους n πρώτους όρους του τύπου του McLaurin*

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!} = \sum_{k=0}^{n} \frac{x^k}{k!}$$

Τρέξτε το πρόγραμμά σας για $x = 0.2$ και $n = 20$ και συγκρίνετε το αποτέλεσμα με την έτοιμη συνάρτηση του Matlab *exp(x).*

Υπόδειξη: *Η τιμή του παραγοντικού $n!$ υπολογίζεται στο* Matlab *με τη συνάρτηση factorial(n).*

Άσκηση 4.3. *Δημιουργήστε ένα αρχείο script που θα υπολογίζει και θα εκτυπώνει τον γεωμετρικό μέσο δέκα θετικών αριθμών. Ο γεωμετρικός μέσος G, n μη αρνητικών αριθμών ορίζεται ως $G = \sqrt[n]{x_1 \cdot x_2 \cdot ... \cdot x_n}$.*

Ο υπολογισμός θα γίνεται ως εξής: Το πρόγραμμα

- *θα διαβάζει έναν αριθμό κάθε φορά,*
- *θα ελέγχει αν είναι θετικός,*
- *θα υπολογίζει το τρέχον γινόμενο.*

Αν ο χρήστης εισάγει έναν μη θετικό αριθμό το πρόγραμμα θα πρέπει να του ζητάει να εισάγει ξανά έναν θετικό αριθμό. Όταν εισαχθούν όλοι οι αριθμοί, θα υπολογίζει τη 10η ρίζα του γινομένου και θα εκτυπώνει το αποτέλεσμα.

Υπόδειξη: *Η λύση της άσκησης μπορεί να υλοποιηθεί σε δύο στάδια, αντίστοιχα με τη λύση του Παραδείγματος 4.9: Στο πρώτο στάδιο, θα δημιουργήσετε την επαναληπτική διαδικασία για τον υπολογισμό του γεωμετρικού μέσου και, αφού βεβαιωθείτε ότι λειτουργεί σωστά, θα προσθέσετε στο κατάλληλο σημείο τον έλεγχο για τους αριθμούς που εισάγει ο χρήστης.*

Άσκηση 4.4. *Δημιουργήστε ένα αρχείο script που θα υπολογίζει τη συνισταμένη n διανυσμάτων που βρίσκονται στο $x - y$ επίπεδο. Το πρόγραμμα θα δέχεται ως είσοδο τον αριθμό n των διανυσμάτων που θα εισαχθούν και, στη συνέχεια, θα εκτελεί μια επαναληπτική διαδικασία κατά την οποία ο χρήστης θα εισάγει το μέτρο και την κατεύθυνση (γωνία) κάθε διανύσματος. Αφού γίνουν οι κατάλληλοι υπολογισμοί, το πρόγραμμα θα επιστρέφει το μέτρο και την κατεύθυνση της συνισταμένης.*

Υπόδειξη: *Στο* Matlab *οι τριγωνομετρικές συναρτήσεις* `sin()`, `cos()`, `tan()` *κ.λπ. δέχονται το όρισμα σε ακτίνια. Για να μετατρέψουμε μία γωνία από ακτίνια σε μοίρες, μπορούμε να χρησιμοποιήσουμε τη συνάρτηση* `rad2deg()`, *ενώ για την αντίστροφη μετατροπή υπάρχει η συνάρτηση* `deg2rad()`. *Τέλος, μπορούμε να χρησιμοποιήσουμε τριγωνομετρικές συναρτήσεις που δέχονται το όρισμα σε μοίρες με τις* `sind()`, `cosd()`, `tand()` *κ.λπ.*

Άσκηση 4.5. *Δημιουργήστε ένα αρχείο script που θα υλοποιεί ένα τυχερό παιχνίδι ρίψης ζαριού, το οποίο θα επαναλαμβάνεται όσες φορές θέλει ο χρήστης. Συγκεκριμένα, θα προσομοιώνεται η ρίψη ζαριού με κατάλληλη εντολή και στην περίπτωση που το ζάρι έχει την τιμή 1, θα εκτυπώνεται μήνυμα ότι ο παίχτης έχει χάσει, αν έχει τιμή 2, 3 ή 4, θα εκτυπώνεται μήνυμα ότι ο γύρος αυτός δεν μετράει, ενώ, αν έχει τιμή 5 ή 6, θα εκτυπώνεται μήνυμα ότι έχει κερδίσει. Στη συνέχεια, ο χρήστης θα ερωτάται αν επιθυμεί να συνεχίσει για έναν ακόμα γύρο και ανάλογα με την απάντησή του είτε θα επαναλαμβάνεται η διαδικασία είτε θα τερματίζεται η εκτέλεση του προγράμματος. Πριν τον τερματισμό, θα πρέπει να ενημερώνεται ο χρήστης πόσες φορές έχει κερδίσει και πόσες έχει χάσει.*

Υπόδειξη: *Σε ό,τι αφορά τη δομή του κώδικα για το τυχερό παιχνίδι, βασιστείτε στη λύση του Παραδείγματος 3.10 του προηγούμενου κεφαλαίου.*

Άσκηση 4.6. *Για την αριθμητική προσέγγιση της ρίζας μιας εξίσωσης συχνά χρησιμοποιείται η αποκαλούμενη μέθοδος της διχοτόμησης, η οποία μπορεί εν συντομία να περιγραφτεί ως εξής:*

- *Δεδομένου ενός αρχικού διαστήματος* $[a,b]$*, για το οποίο οι συναρτησιακές τιμές στα άκρα του έχουν αντίθετο πρόσημο, δηλαδή* $f(a)\cdot f(b)<0$*, η μέθοδος της διχοτόμησης διχοτομεί το διάστημα αυτό σε δύο καινούρια διαστήματα* $[a,c]$ *και* $[c,b]$*, όπου* c *είναι το μέσο του* $[a,b]$*.*
- *Στη συνέχεια, γίνεται έλεγχος σε ποιο από τα δύο διαστήματα ισχύει το κριτήριο του Bolzano, δηλαδή ελέγχεται αν* $f(a)\cdot f(c)<0$ *ή αν* $f(b)\cdot f(c)<0$*.*
- *Επιλέγεται ως νέο διάστημα αυτό στο οποίο ισχύει το κριτήριο του Bolzano και η διαδικασία συνεχίζεται με το πρώτο βήμα της μεθόδου.*

Η διαδικασία αυτή γίνεται επαναληπτικά έως ότου το τελευταίο διάστημα έχει πλάτος μικρότερο ή ίσο από την ακρίβεια με την οποία επιθυμούμε να βρούμε τη λύση.

Δημιουργήστε ένα αρχείο script που να υπολογίζει με τη μέθοδο της διχοτόμησης την ρίζα της εξίσωσης

$$f(x)=x+10e^{-x^2}\cos x$$

Εφαρμόστε τη μέθοδο στο διάστημα το $[-3,3]$ *και σταματήστε τη διαδικασία όταν το διάστημα γίνει μικρότερο ή ίσο του* 10^{-5}*.*

4.8 Απαντήσεις στις ασκήσεις αυτοαξιολόγησης

Λύση άσκησης αυτοαξιολόγησης 4.1

Τα αρχικά βήματα του εναλλακτικού αλγόριθμου είναι ίδια με αυτά του Παραδείγματος 4.2 και περιλαμβάνουν τη δημιουργία των δύο μεταβλητών στις οποίες θα αποθηκεύονται οι τιμές x και y (π.χ. x και y). Στη συνέχεια, το πρόγραμμα θα πρέπει να εκτελεί έναν βρόχο χρησιμοποιώντας τη μεταβλητή x, η οποία θα μεταβάλλεται από 1 μέχρι 2, με κατάλληλο βήμα ώστε να λάβουμε συνολικά 11 σημεία. Το βήμα αυτό είναι $\frac{2-1}{10} = 0.1$, το οποίο χωρίζει το διάστημα από 1 έως 2 σε 10 σημεία και αν συμπεριλάβουμε *και* το 1, τότε έχουμε συνολικά 11 σημεία.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %X_Y_VALUES Calculate the x-y values of y = log(x) in [1,2]
2  % Initialization
3   clear all
4   x = 0;
5   y = 0;
6
7  % Main code
8   for x=1:(2-1)/10:2
9     y = log(x);
10    disp([x,y])
11  end
```

Λύση άσκησης αυτοαξιολόγησης 4.2

Αυτό που πρέπει να αλλάξει σε σχέση με τη λύση του Παραδείγματος 4.3 είναι η θέση x στην οποία θα υπολογίσουμε το ύψος y. Συγκεκριμένα, στη λύση αυτή χρησιμοποιούσαμε εγγεγραμμένα παραλληλόγραμμα, οπότε υπολογίζαμε το ύψος στην αρχή κάθε διαστήματος της διαμέρισης (λόγω του ότι η συνάρτησή μας είναι γνησίως αύξουσα στο (0,1), βλ. την αριστερή εικόνα του Σχήματος 4.2). Στον νέο αλγόριθμο θέλουμε να αθροίσουμε περιγεγραμμένα παραλληλόγραμμα, οπότε θα υπολογίσουμε το ύψος στην τελική τιμή κάθε διαστήματος.
Η εντολή, λοιπόν, που πρέπει να αλλάξει βρίσκεται στη γραμμή 12 της λύσης του Παραδείγματος 4.3 και είναι η x = (i-1)*dx, η οποία έδινε κατά σειρά τις τιμές $0, 0.1, 0.2, \ldots$ και 0.9.
Η νέα εντολή είναι η x = i*dx, η οποία δίνει κατά σειρά τις τιμές $0.1, 0.2, \ldots, 0.9$ και 1.
Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %AREA_1 Calculate area below y = x^2 in [0,1]
2  % Initialization
3   clear all
4   x = 0;
5   y = 0;
6   N=100;
7   dx = 1/N;
8   E = 0;
9
10 % Main code
11  for i=1:N
```

```
12      x = i*dx;
13      y = x^2;
14
15      % Calculation of Area
16      E = E + y*dx;
17    end
18
19    disp(['Methodos parallilogrammwn: ',num2str(E)])
20    disp(['Akribhs ypologismos: ',num2str(1/3)])
```

Λύση άσκησης αυτοαξιολόγησης 4.3

Όπως και στη λύση της προηγούμενης άσκησης, η εντολή που πρέπει να αλλάξει βρίσκεται στη γραμμή 12 της λύσης του Παραδείγματος 4.3. Η νέα εντολή πρέπει να υπολογίζει το x στο μέσο κάθε διαστήματος (βλ. τη δεξιά εικόνα του Σχήματος 4.2). Μπορούμε, λοιπόν,

- είτε να βρούμε το x στην αρχή κάθε διαστήματος και να προσθέσουμε μισό βήμα dx (π.χ. x = (i-1)*dx + 0.5*dx),
- είτε να βρούμε το x στο τέλος κάθε διαστήματος και να αφαιρέσουμε μισό βήμα dx (π.χ. x = i*dx - 0.5*dx),

Με οποιονδήποτε από τους τρόπους αυτούς, θα λάβουμε κατά σειρά τις τιμές 0.05, 0.15, 0.025, ..., 0.85 και 0.95.

Η παραπάνω λύση, χρησιμοποιώντας τον 2ο τρόπο, υλοποιείται στον κώδικα που ακολουθεί:

```
1   %AREA_1 Calculate area below y = x^2 in [0,1]
2   % Initialization
3   clear all
4   x = 0;
5   y = 0;
6   N=10;
7   dx = 1/N;
8   E = 0;
9
10  % Main code
11  for i=1:N
12    x = (i-0.5)*dx
13    y = x^2;
14
15    % Calculation of Area
16    E = E + y*dx;
17  end
18
19  disp(['Methodos parallilogrammwn: ',num2str(E)])
20  disp(['Akribhs ypologismos: ',num2str(1/3)])
```

Λύση άσκησης αυτοαξιολόγησης 4.4

Όπως μας υποδεικνύεται και στην εκφώνηση της άσκησης, αυτό που πρέπει να αλλάξει είναι η αρχικοποίηση των μεταβλητών. Πρέπει, λοιπόν, να αλλάξουμε τη γραμμή 6 στη λύση του Παραδείγματος 4.6, θέτοντας s=1, και να ξεκινήσουμε την επαναληπτική διαδικασία με τον όρο πρώτου βαθμού x. Για να γίνει αυτό, πρέπει να αρχικοποιήσουμε τον «μετρητή» n στη γραμμή 7 με την τιμή 1.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %MCLAURIN Calculate the McLaurin series of y = 1/(1-x)
2  % Initialization
3   clear all
4   x = 0.5;
5   term = 1;
6   s = 1;
7   n=1;
8
9   % Main code
10  while abs(term) >= 1.e-5
11    term = x^n;
12    s = s + term;
13    n = n+1;
14  end
15  disp(['After ', num2str(n),' terms, the result is: '])
16  disp(s)
17
18  % Validation
19  disp('Exact value:')
20  disp(1/(1-x))
```

Λύση άσκησης αυτοαξιολόγησης 4.5

Είδαμε στη λύση του Παραδείγματος 4.10 ότι η επαναληπτική διαδικασία τερματίζεται είτε όταν συμπληρωθούν οι 5 επαναλήψεις είτε όταν το ζάρι πάρει την τιμή 1. Αν αντιστρέψουμε αυτή τη θεώρηση, η διαδικασία ρίψης του ζαριού συνεχίζεται *όσο* (`while`) ο αριθμός των επαναλήψεων είναι μικρότερος του 5 *και* όσο δεν έχει φέρει το ζάρι την τιμή 1. Άρα, οι συνθήκες που καθορίζουν τη συνέχιση των ρίψεων του ζαριού είναι

- i<5

και

- zari ~= 1

Αυτές τις δύο συνθήκες πρέπει να περιλάβουμε στον έλεγχο της επαναληπτικής δομής `while`, ώστε να πετύχουμε το αποτέλεσμα που ζητάει η εκφώνηση του Παραδείγματος 4.10.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %DICE_6 Count how many times the dice is 6
2  % Initialization
3   clear all
```

```
4   zari = 0;
5   eksaria = 0;
6   i=0;
7
8   % Main code
9   while i<5 && zari~=1
10    zari = randi(6)
11    % Count how many times zari is 6
12    if zari == 6
13      eksaria = eksaria + 1;
14    end
15    i = i + 1;
16  end
17
18  disp(['To 6 emfanistike ',num2str(eksaria),' fores'])
```

Λύση άσκησης αυτοαξιολόγησης 4.6

Είδαμε στη λύση του Παραδείγματος 4.12 ότι με τη χρήση της εντολής `continue` το πρόγραμμα αγνοεί τις εντολές για το άθροισμα και το γινόμενο, όταν ο αριθμός *δεν είναι* πρώτος. Για να αποφύγουμε τη χρήση του `continue`, πρέπει να αντιστρέψουμε αυτή τη συλλογιστική. Πρέπει, δηλαδή, να ενεργοποιούμε το άθροισμα και το γινόμενο μόνο όταν ο αριθμός *είναι* πρώτος.
Άρα, όταν το πρόγραμμα εντοπίζει έναν πρώτο αριθμό, θα εκτελεί τις απαραίτητες ενέργειες (πρόσθεση και πολλαπλασιασμό), ενώ σε αντίθετη περίπτωση δεν θα εκτελεί καμία ενέργεια και θα προχωράει στην επόμενη επανάληψη.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1   %CALC_PRIMES Calculate the sum and the product of prime numbers
2   % Initialization
3   clear all
4   athroisma = 0;
5   ginomeno = 1;
6
7   % Main code
8   for i=1:20
9
10    % Condition for prime numbers
11    if isprime(i)
12      athroisma = athroisma + i;
13      ginomeno = ginomeno * i;
14    end
15
16  end
17  disp(['To athroisma einai: ',num2str(athroisma)])
18  disp(['To ginomeno einai: ',num2str(ginomeno)])
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ελληνόγλωσση

Thomas, G. B., Finney, R. L., Weir, M. D. και Giordano, F. R. (2009). *Απειροστικός Λογισμός Τόμος ΙΙ*. ITE-Πανεπιστημιακές Εκδόσεις Κρήτης.

Μπράτσος, Α. (2015). *Μαθήματα εφαρμοσμένων μαθηματικών*. Εκδόσεις Κάλλιπος.

Ξενόγλωσση

Hoffman, J. D. (2001). *Numerical Methods for Engineers and Scientists, (2nd ed.)* CRC Press, New York.

ΚΕΦΑΛΑΙΟ 5

ΔΗΜΙΟΥΡΓΙΑ ΓΡΑΦΙΚΩΝ ΠΑΡΑΣΤΑΣΕΩΝ

Σύνοψη:

Σε αυτό το κεφάλαιο παρουσιάζονται οι βασικοί τρόποι κατασκευής και διαμόρφωσης διδιάστατων γραφικών παραστάσεων. Πιο συγκεκριμένα, αρχικά παρουσιάζονται οι διάφορες εντολές για την κατασκευή και μορφοποίηση των γραφικών παραστάσεων, ενώ, στη συνέχεια, παρατίθενται οι εντολές για τον σχεδιασμό πολλαπλών γραφικών παραστάσεων στο ίδιο γράφημα ή στο ίδιο παράθυρο.

Προαπαιτούμενη γνώση: Τα κεφάλαια 2,3 και 4 του παρόντος συγγράμματος.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του κεφαλαίου αυτού, θα γνωρίζετε:

- πώς να κατασκευάζετε διδιάστατες γραφικές παραστάσεις,
- πώς να μορφοποιείτε διδιάστατες γραφικές παραστάσεις.

Γλωσσάριο επιστημονικών όρων

- Γράφημα χρονοσειράς
- Διδιάστατες γραφικές παραστάσεις
- Άξονες

5.1 Εισαγωγή

Το Matlab παρέχει μια σειρά από εντολές και εργαλεία για την παραγωγή γραφικών παραστάσεων. Οι γραφικές παραστάσεις κατασκευάζονται σε ξεχωριστό παράθυρο, ενώ, γενικά, οι νέες γραφικές παραστάσεις κατασκευάζονται στο ίδιο παράθυρο αντικαθιστώντας την τελευταία που έχει κατασκευαστεί. Ένα κενό παράθυρο για την κατασκευή μιας νέας γραφικής παράστασης, διατηρώντας τις προηγούμενες γραφικές παραστάσεις, μπορεί να δημιουργηθεί με την εντολή `figure`. Στο καινούργιο αυτό παράθυρο μπορεί να σχεδιαστεί μια νέα γραφική παράσταση διατηρώντας το παράθυρο στο οποίο είχε αποτυπωθεί η προηγούμενη γραφική παράσταση.

Στο κεφάλαιο αυτό, αρχικά, παρουσιάζονται οι εντολές για την κατασκευή διδιάστατων γραφικών παραστάσεων και, στη συνέχεια, οι εντολές για τη μορφοποίησή τους. Τέλος, παρουσιάζονται οι διαδικασίες για τη δημιουργία πολλαπλών γραφικών παραστάσεων είτε στο ίδιο γράφημα είτε στο ίδιο παράθυρο.

5.2 Κατασκευή γραφικών παραστάσεων

Οι βασικές εντολές κατασκευής γραφικών παραστάσεων είναι οι

- `plot`
- `fplot`
- `ezplot`

οι οποίες παρουσιάζονται στη συνέχεια. Στο τέλος της ενότητας, παρουσιάζεται μια ενσωματωμένη εφαρμογή που έχει το Matlab για την απεικόνιση συναρτήσεων, η οποία εκκινεί με την εντολή `funtool`.

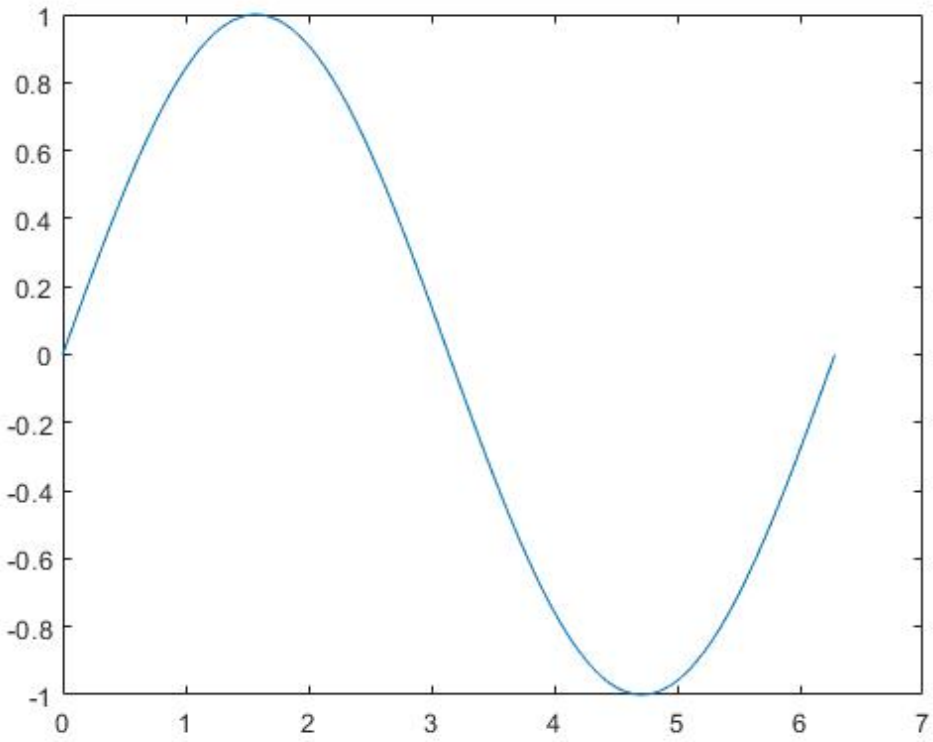
5.2.1 Η εντολή plot(x,y)

Η εντολή `plot` είναι η βασικότερη αλλά και η πιο απλή εντολή για τη δημιουργία διδιάστατων γραφικών παραστάσεων στο Matlab. Το βασικό συντακτικό της είναι το `plot(x,y)`, όπου τα x και y είναι διανύσματα με το ίδιο πλήθος στοιχείων. Παραδείγματος χάριν, οι εντολές

```
>> x=0:2*pi/100:2*pi;
>> y=sin(x);
>> plot(x,y)
```

παράγουν τη γραφική παράσταση του Σχήματος 5.1, στην οποία απεικονίζεται η συνάρτηση ημίτονο στο διάστημα $[0,2\pi]$.

Η πρώτη εντολή παράγει το διάνυσμα των σημείων πάνω από τα οποία θα υπολογιστεί η τιμή του ημιτόνου. Πιο συγκεκριμένα, το x αποτελείται από 101 σημεία ξεκινώντας από το μηδέν και με βήμα $2\pi/100$ φτάνει μέχρι το 2π. Η δεύτερη εντολή δημιουργεί τη μεταβλητή y, η οποία είναι διάνυσμα 101 στοιχείων με τις τιμές του ημιτόνου υπολογισμένες στα σημεία του x. Η τρίτη, και τελευταία εντολή, είναι αυτή που δημιουργεί τη γραφική παράσταση του Σχήματος 5.1, χρησιμοποιώντας τα σημεία $(x_i,y_i),\ i=1,2,\ldots,101$, που ορίζονται από τα x και y.



Σχήμα 5.1: Η γραφική παράσταση του ημιτόνου στο διάστημα $[0,2\pi]$.

Παρατήρηση 5.1

Σημειώνεται ότι η δεύτερη και η τρίτη εντολή στις προηγούμενες εντολές μπορούν να αντικατασταθούν από την `plot(x,sin(x))`. Η διαφορά μεταξύ αυτών των δύο διαφορετικών προσεγγίσεων είναι ότι με τον δεύτερο τρόπο δεν δημιουργείται η μεταβλητή y.

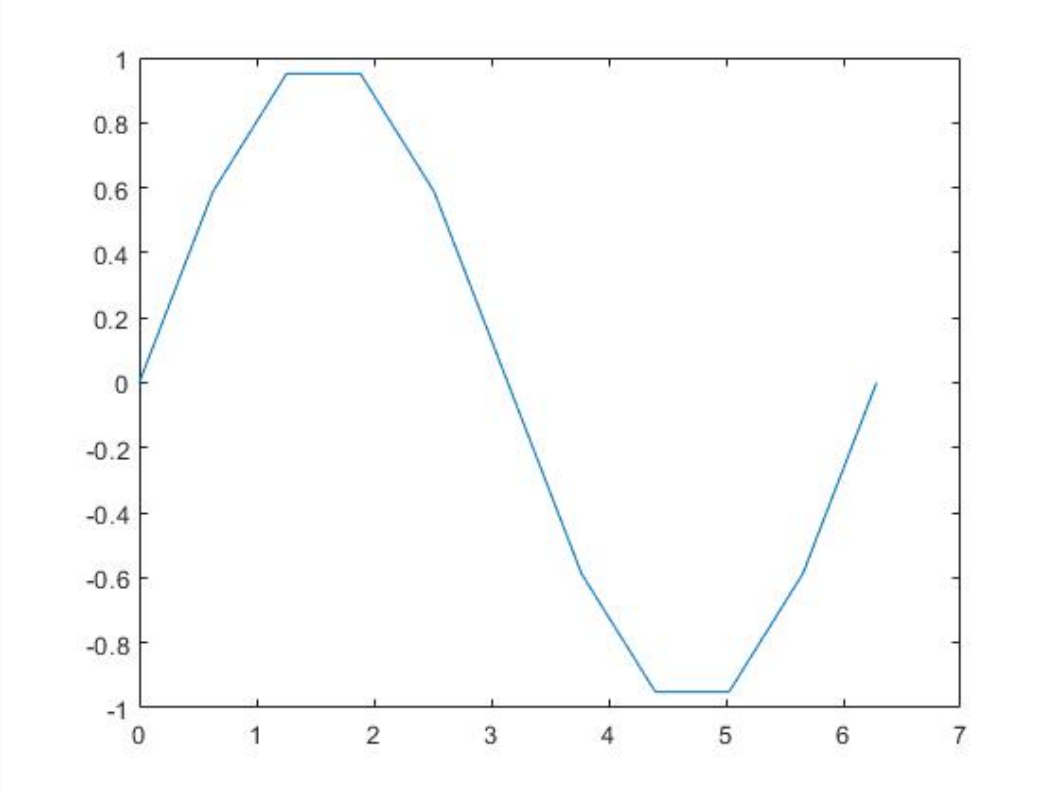
Παρατήρηση 5.2

Η εντολή `plot(x,y)`, στην ουσία, κατασκευάζει τη γραφική παράσταση ενώνοντας με ευθύγραμμα τμήματα τα σημεία (x_i,y_i), $i=1,2,\ldots,101$, που ορίζονται από τα x και y. Η καμπύλη που αποτυπώνεται στο Σχήμα 5.1 οφείλεται στο γεγονός ότι χρησιμοποιήθηκε ένα σχετικά μεγάλο πλήθος σημείων για την κατασκευή του γραφήματος. Το πλήθος αυτό δημιουργεί μια μεγάλη πυκνότητα στα σημεία με συνέπεια το γράφημα να δίνει την αίσθηση μιας ομαλής καμπύλης και όχι μιας καμπύλης που αποτελείται από ευθύγραμμα τμήματα.

Αν, αντί για τα 101 σημεία,είχαμε χρησιμοποιήσει ένα μικρότερο πλήθος σημείων, για να ορίσουμε το διάνυσμα x, όπως στις παρακάτω εντολές

```
>> x=0:2*pi/10:2*pi;
>> y=sin(x);
>> plot(x,y)
```

τότε θα λαμβάναμε την ακόλουθη γραφική παράσταση στην οποία αποτυπώνονται ευδιάκριτα τα ευθύγραμμα τμήματα, με τα οποία κατασκευάζεται η γραφική παράσταση.



Σχήμα 5.2: Η γραφική παράσταση του ημιτόνου στο διάστημα $[0, 2\pi]$ χρησιμοποιώντας μικρό πλήθος σημείων για τον σχεδιασμό της.

Επομένως, θα πρέπει κατά την κατασκευή μιας γραφικής παράστασης να χρησιμοποιείται ένας ικανοποιητικός αριθμός σημείων, έτσι ώστε να αποτυπώνεται η πραγματική φύση της συνάρτησης που απεικονίζεται.
Είναι σημαντικό, επίσης, να τονιστεί ότι πρέπει να αποφεύγονται και τα ιδιαίτερα μεγάλα πλήθη σημείων, διότι:

- όχι μόνο δεν προσφέρουν σημαντικά περισσότερη πληροφορία για τη συμπεριφορά της συνάρτησης αλλά και
- επιβαρύνουν τη μνήμη του υπολογιστή και απαιτούν περισσότερο χρόνο για την κατασκευή της γραφικής παράστασης.

Άσκηση αυτοαξιολόγησης 5.1

Κατασκευάστε το ευθύγραμμο τμήμα που ενώνει τα σημεία (2,3) και (10,5).

5.2.2 Η εντολή plot(y)

Αν καλέσουμε την εντολή `plot` με ένα όρισμα (διάνυσμα), δηλαδή ως `plot(y)`, τότε απεικονίζει τα στοιχεία του y ως προς τον αύξοντα αριθμό τους. Ένα τέτοιο γράφημα είναι ιδιαίτερα χρήσιμο όταν έχουμε επαναλαμβανόμενες, κατά τακτά χρονικά διαστήματα, μετρήσεις ενός μεγέθους και θέλουμε να αποτυπώσουμε τη χρονική εξέλιξή του. Ένα τέτοιο γράφημα αποκαλείται και «γράφημα χρονοσειράς».

Παράδειγμα 5.1

Παρακάτω, δίνεται υπό μορφή πίνακα ο μηνιαίος αριθμός (σε χιλιάδες) επιβατών αεροπορικών γραμμών στις ΗΠΑ για την περίοδο Ιανουαρίου 1949 έως Δεκέμβρη 1960. Να κατασκευάσετε το γράφημα χρονοσειράς για τα δεδομένα αυτά και να διακρίνετε ιδιαίτερα χαρακτηριστά του μηνιαίου αριθμού επιβατών.

```
>> %1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960
y = [112, 115, 145, 171, 196, 204, 242, 284, 315, 340, 360, 417;   % Jan
     118, 126, 150, 180, 196, 188, 233, 277, 301, 318, 342, 391;   % Feb
     132, 141, 178, 193, 236, 235, 267, 317, 356, 362, 406, 419;   % Mar
     129, 135, 163, 181, 235, 227, 269, 313, 348, 348, 396, 461;   % Apr
     121, 125, 172, 183, 229, 234, 270, 318, 355, 363, 420, 472;   % May
     135, 149, 178, 218, 243, 264, 315, 374, 422, 435, 472, 535;   % Jun
     148, 170, 199, 230, 264, 302, 364, 413, 465, 491, 548, 622;   % Jul
     148, 170, 199, 242, 272, 293, 347, 405, 467, 505, 559, 606;   % Aug
     136, 158, 184, 209, 237, 259, 312, 355, 404, 404, 463, 508;   % Sep
     119, 133, 162, 191, 211, 229, 274, 306, 347, 359, 407, 461;   % Oct
     104, 114, 146, 172, 180, 203, 237, 271, 305, 310, 362, 390;   % Nov
     118, 140, 166, 194, 201, 229, 278, 306, 336, 337, 405, 432];  % Dec
```

Λύση Παραδείγματος 5.1

Τα δεδομένα με τη μορφή που δίνονται σχηματίζουν έναν πίνακα 12×12 με κάθε στήλη να αντιπροσωπεύει ένα διαφορετικό έτος. Για να αναδιατάξουμε τις μετρήσεις ανά μήνα σε ένα διάνυσμα 144 στοιχείων, χρήσιμη είναι η εντολή `reshape`. Η εντολή `reshape(X, M, N)` επιστρέφει έναν $M \times N$ πίνακα, του οποίου τα στοιχεία έχουν ληφθεί από τον X κατά στήλες. Με βάση τα παραπάνω, η εντολή

```
>> y = reshape(y,1,144);
```

αναδιατάσσει τα στοιχεία του y, δημιουργώντας ένα διάνυσμα με 144 στοιχεία και ανανεώνει με αυτό την τιμή της μεταβλητής y. Η εντολή

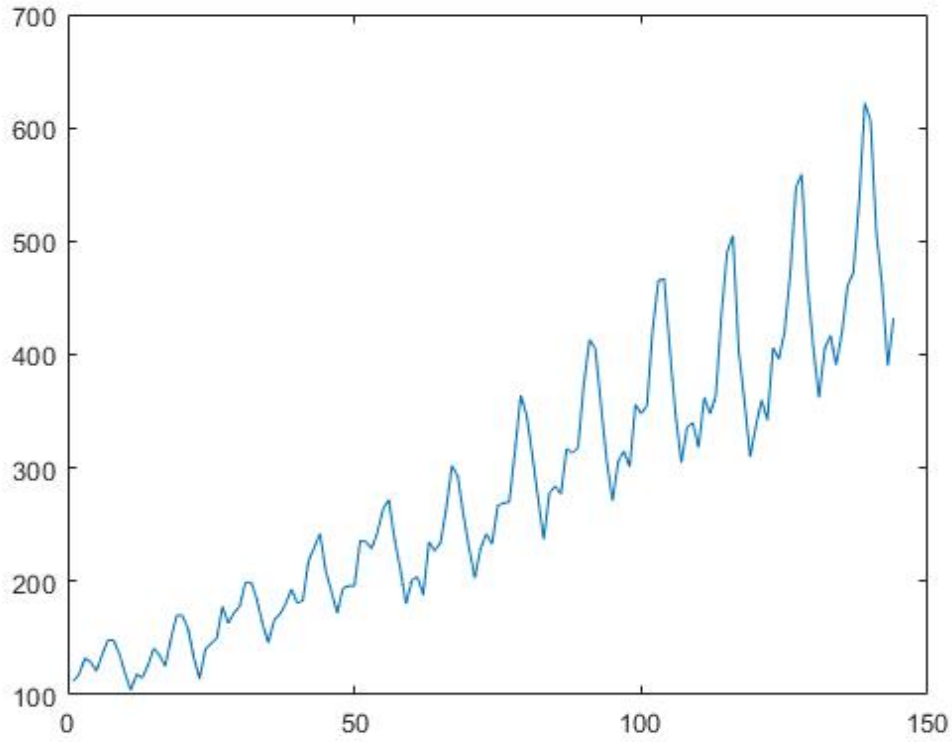
```
>> plot(y)
```

επιστρέφει το Σχήμα 5.3 με το γράφημα χρονοσειράς για τα δεδομένα του μηνιαίου αριθμού (σε χιλιάδες) επιβατών αεροπορικών γραμμών στις ΗΠΑ για την περίοδο Ιανουαρίου 1949 έως Δεκέμβρη 1960. Από το γράφημα είναι φανερή τόσο η αυξητική τάση του αριθμού επιβατών όσο και η ετήσια περιοδικότητά του.

5.2.3 Οι εντολές fplot και ezplot

Το MATLAB, πέρα από τους προαναφερθέντες τρόπους παραγωγής γραφικών παραστάσεων, έχει και τις εντολές `fplot` και `ezplot` για τη δημιουργία γραφικών παραστάσεων, χωρίς την ανάγκη εισαγωγής των συντεταγμένων πολλών σημείων από τον χρήστη. Στην πραγματικότητα, τη διαδικασία αυτήν την κάνει σιωπηλά το MATLAB και για τον λόγο αυτό η εντολή αυτή είναι συνήθως πιο αργή από την εντολή `plot`.

Παρ΄ όλα αυτά, οι δύο αυτές εντολές έχουν το πλεονέκτημα ότι μπορούν να χρησιμοποιηθούν στο πλαίσιο χρήσης συμβολικών μεταβλητών και συναρτήσεων σαν αυτές που θα αναφέρουμε στο Κεφάλαιο 10. Στην παρούσα ενότητα, παρουσιάζεται ο τρόπος εκτέλεσής τους στο Matlab με τη βοήθεια των επόμενων εντολών,



Σχήμα 5.3: Το γράφημα χρονοσειράς για τα δεδομένα του μηνιαίου αριθμού (σε χιλιάδες) επιβατών αεροπορικών γραμμών στις ΗΠΑ για την περίοδο Ιανουαρίου 1949 έως Δεκέμβρη 1960.

που σχεδιάζουν σε δύο διαφορετικά παράθυρα τη γραφική παράσταση της συνάρτησης $f(x) = x^2 sin(1/x)$ στο διάστημα $[-1,1]$.

```
>> fplot('x^2*sin(1/x)',[-1,1]
>> figure
>> ezplot('x^2*sin(1/x)',[-1,1])
```

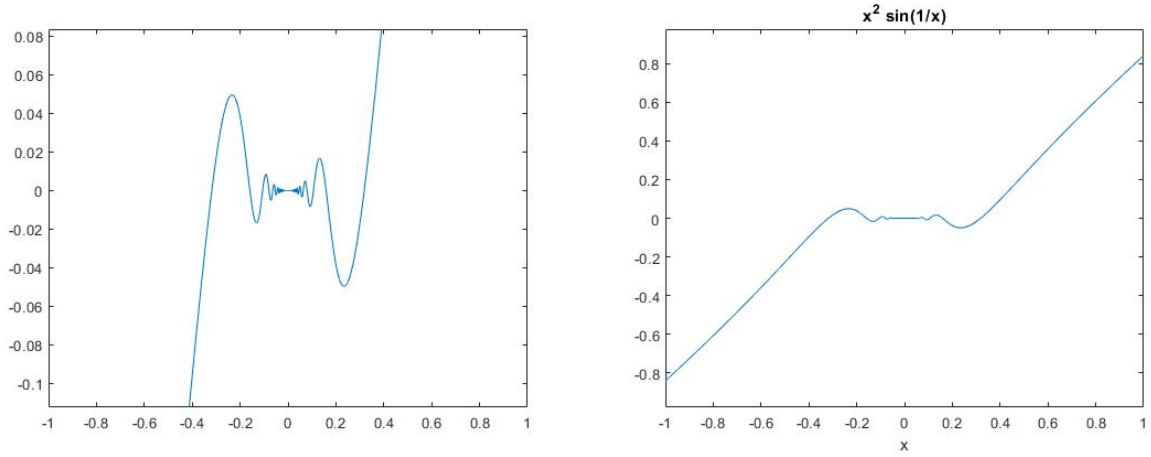
Οι γραφικές παραστάσεις που κατασκευάζονται από τις παραπάνω εντολές εμφανίζονται στο Σχήμα 5.4. Πιο συγκεκριμένα, από την εντολή `fplot` κατασκευάζεται το αριστερό γράφημα, ενώ από την `ezplot` το δεξί γράφημα. Μια αρχική παρατήρηση είναι ότι οι δύο εντολές δεν παράγουν την ίδια εικόνα. Αυτό συμβαίνει επειδή η `fplot` έχει μεγεθύνει το κεντρικό κομμάτι του γραφήματος και έχει αποκόψει τις κατά απόλυτο μεγάλες τιμές που λαμβάνει η $f(x)$ στα άκρα του διαστήματος $[-1,1]$. Από την άλλη, η `ezplot` έχει σχεδιάσει το σύνολο των τιμών της συνάρτησης για $x \in [-1,1]$.

Παρατήρηση 5.3

Κατά τη χρήση της εντολής `fplot` το Matlab τυπώνει μια προειδοποίηση ότι το συγκεκριμένο συντακτικό της εντολής δεν θα υποστηρίζεται στο μέλλον, χωρίς να αναφέρει, όμως, από ποια έκδοση του Matlab και μετά θα συμβεί αυτό. Ταυτόχρονα προτείνει το ακόλουθο συντακτικό για τον σχεδιασμό της $f(x) = x^2 sin(1/x)$

```
>> fplot(@(x) x.^2.*sin(1./x),[-1,1])
```

το οποίο βασίζεται στον ορισμό ανώνυμων συναρτήσεων (βλ. Κεφάλαιο 9).



Σχήμα 5.4: Γραφικές παραστάσεις της συνάρτησης $f(x) = x^2 sin(1/x)$ στο διάστημα $[-1,1]$ με τη βοήθεια των εντολών `fplot` (αριστερό γράφημα) και `ezplot` (δεξί γράφημα).

Άσκηση αυτοαξιολόγησης 5.2

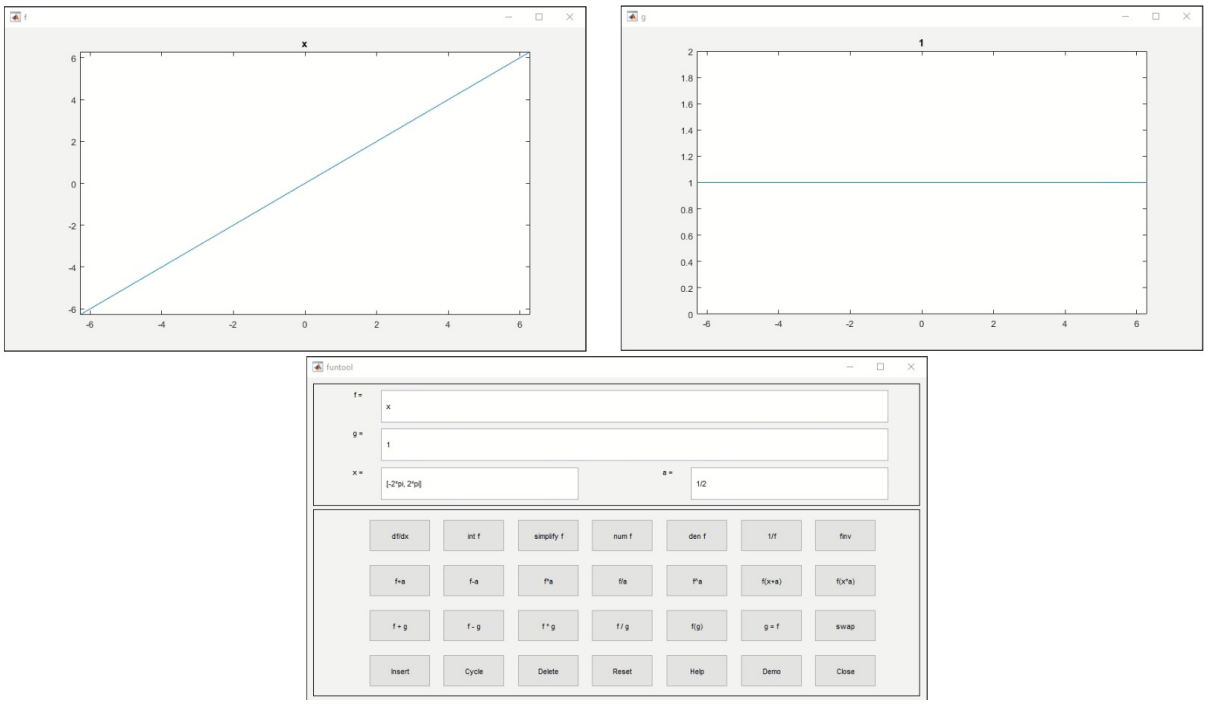
Επιλέξτε την εντολή που κατασκευάζει το γράφημα της συνάρτησης $f(x) = xcos(2x)$ στο διάστημα $[0,2\pi]$:

1. `fplot(@(x) x.*cos(2*x), [0,2*π])`
2. `fplot(@(x) x*cos(2x), [0,2*π])`
3. `fplot(@(x) x*cos(2x), [0,2*pi])`
4. `fplot(@(x) x.*cos(2*x), [0,2*pi])`

5.2.4 Η εντολή funtool

Η εντολή `funtool` ανοίγει μια διαδραστική εφαρμογή σχεδιασμού γραφικών παραστάσεων η οποία χρησιμοποιεί έναν πίνακα ελέγχου στον οποίο με απλά κλικ σε διάφορα κουμπιά μπορούμε να σχεδιάσουμε, σε δύο ξεχωριστά παράθυρα, τις γραφικές παραστάσεις δύο συναρτήσεων, $f(x)$ και $g(x)$. Ένα τρίτο παράθυρο ελέγχει τις συναρτήσεις f και g, το διάστημα πάνω από το οποίο θα σχεδιαστούν οι συναρτήσεις αυτές και την τιμή μιας σταθεράς α. Στο ίδιο παράθυρο υπάρχουν τέσσερις σειρές κουμπιών που επιτρέπουν την εφαρμογή συχνά χρησιμοποιούμενων μετασχηματισμών και συνδυασμών των συναρτήσεων $f(x)$ και $g(x)$, καθώς και κάποιες βοηθητικές επιλογές για τον ευκολότερο χειρισμό της εφαρμογής.

Στο Σχήμα 5.5 εμφανίζεται το περιβάλλον της διαδραστικής εφαρμογής γραφικών συναρτήσεων `funtool`. Στο αριστερό πάνω παράθυρο σχεδιάζεται η γραφική παράσταση της $f(x)$, ενώ στο δεξί πάνω παράθυρο σχεδιάζεται η γραφική παράσταση της $g(x)$. Στο τρίτο παράθυρο εμφανίζεται ο πίνακας ελέγχου της εφαρμογής.



Σχήμα 5.5: Το περιβάλλον της διαδραστικής εφαρμογής `funtool` για τον σχεδιασμό των γραφικών παραστάσεων δύο συναρτήσεων, $f(x)$ και $g(x)$.

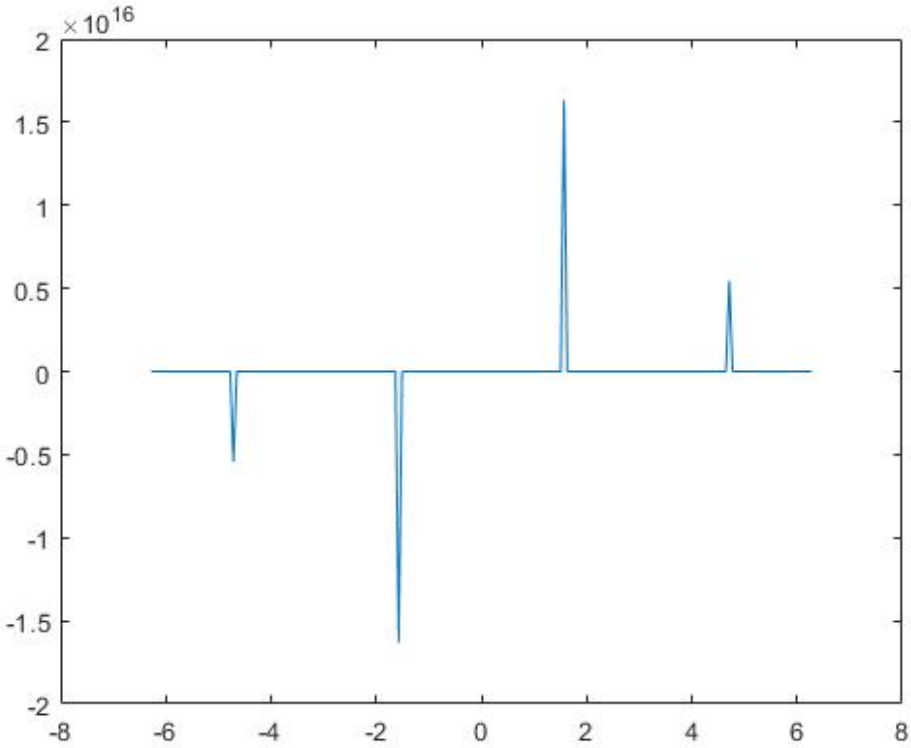
5.3 Μορφοποίηση γραφικών παραστάσεων

Στην προηγούμενη ενότητα παρουσιάστηκαν οι βασικοί τρόποι κατασκευής γραφικών παραστάσεων στο Matlab. Όλες οι γραφικές παραστάσεις, που παρουσιάστηκαν, χρησιμοποιούσαν μια μπλε συνεχόμενη γραμμή για την αποτύπωση της συνάρτησης. Επίσης, το Matlab καθόριζε αυτόματα τα όρια και τη μορφή των αξόνων. Σε πολλές περιπτώσεις, όμως, η εικόνα που λαμβάνουμε από το Matlab δεν είναι η επιθυμητή και συχνά χρειάζεται να μορφοποιήσουμε τη γραφική παράσταση έτσι ώστε να πάρουμε την επιθυμητή εικόνα.

Στην ενότητα αυτή, παρουσιάζουμε τις διαδικασίες μορφοποίησης μιας γραφικής παράστασης. Η παρουσίαση γίνεται με τη βοήθεια της συνάρτησης $f(x) = tan(x)$, την οποία επιθυμούμε να σχεδιάσουμε πάνω από το διάστημα $[-2\pi, 2\pi]$. Η επιλογή της συνάρτησης αυτής έγινε, διότι το Matlab κατασκευάζει με τις ακόλουθες εντολές

```
>> x=-2*pi:4*pi/200:2*pi;
>> y=tan(x);
>> plot(x,y)
```

τη γραφική παράσταση του Σχήματος 5.6, η οποία απέχει πολύ από την εικόνα που θα έπρεπε να έχει η γραφική παράσταση της εφαπτομένης. Ο βασικός λόγος για αυτό είναι ότι η εφαπτομένη απειρίζεται για $x = k\pi + \pi/2$, όπου k ακέραιος αριθμός. Συνέπεια αυτού είναι ότι σε σημεία κοντά σε αυτές τις τιμές η $f(x) = tan(x)$ λαμβάνει εξαιρετικά μεγάλες (κατά απόλυτο) τιμές. Αυτό αποτυπώνεται στην κλίμακα που έχει ο κάθετος άξονας που, όπως παρατηρούμε, είναι σε μονάδες της τάξης του 10^{16}. Αυτή η μεγάλη κλίμακα καθιστά αδύνατη την αποτύπωση της καμπυλότητας της συνάρτησης στα υπόλοιπα σημεία με αποτέλεσμα αυτή να αποτυπώνεται ως ευθεία γραμμή.



Σχήμα 5.6: Η γραφική παράσταση της συνάρτησης $f(x) = tan(x)$ στο διάστημα $[-2\pi, 2\pi]$, όπως αυτή κατασκευάζεται από το Matlab με την εντολή `plot(x,y)` χωρίς κάποια μορφοποίηση.

5.3.1 Μορφοποίηση αξόνων

Με τις εντολές `xlim([xmin xmax])` και `ylim([ymin ymax])` μπορούμε να ορίσουμε τα όρια του οριζόντιου και του κάθετου άξονα, αντίστοιχα. Παραδείγματος χάριν, με την εντολή

```
>> ylim([-10 10])
```

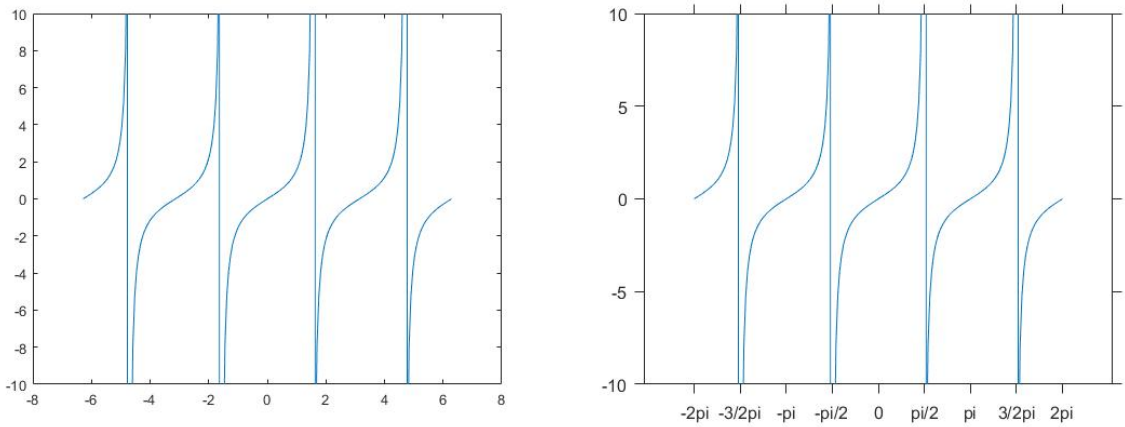
μπορούμε να περιορίσουμε τον κάθετο άξονα στο διάστημα $[-10, 10]$ και να λάβουμε το αριστερό γράφημα του Σχήματος 5.7, στο οποίο αποτυπώνεται καλύτερα η συμπεριφορά της συνάρτησης της εφαπτομένης.

Παρατήρηση 5.4

Σημειώνεται ότι με την εντολή `axis([xmin xmax ymin ymax])` μπορούν να καθοριστούν ταυτόχρονα τα όρια του οριζόντιου και του κάθετου άξονα ενός γραφήματος.

Πέρα από τα όρια των αξόνων, το Matlab επιτρέπει τη μορφοποίηση και άλλων χαρακτηριστικών των αξόνων, όπως τη γραμματοσειρά των τιμών που εμφανίζονται στους άξονες, τα σημεία που εμφανίζονται αυτές, καθώς και το κείμενο που εμφανίζεται σε αυτά τα σημεία. Παραδείγματος χάριν, με τις εντολές

```
>> ax=gca;
>> ax.FontSize = 12;
>> ax.TickDir = 'out';
>> ax.TickLength = [0.02 0.02];
>> ax.XTick = -2*pi:pi/2:2*pi;
>> ax.XTickLabel = {'-2pi','-3/2pi','-pi','-pi/2','0','pi/2','pi','3/2pi','2pi
    '};
```



Σχήμα 5.7: Αριστερό γράφημα: Η γραφική παράσταση της συνάρτησης $f(x) = tan(x)$ στο διάστημα $[-2\pi, 2\pi]$ με τα όρια στον κάθετο άξονα ίσα με $[-10, 10]$. Δεξί γράφημα: Η γραφική παράσταση της συνάρτησης $f(x) = tan(x)$ στο διάστημα $[-2\pi, 2\pi]$ με μορφοποιημένους τους άξονες.

λαμβάνουμε το δεξί γράφημα του Σχήματος 5.7. Η πρώτη εντολή δημιουργεί τη μεταβλητή ax, στην οποία αναθέτουμε τον χειρισμό των αξόνων του τρέχοντος γραφήματος. Η εντολή `gca`, στην ουσία, αποτελείται από τα αρχικά των λέξεων graph current axis, δηλαδή των λέξεων «άξονες τρέχοντος γραφήματος». Οι εντολές με τη μορφή `ax.xxxxxx` επιτρέπουν τη μορφοποίηση διάφορων χαρακτηριστικών των αξόνων. Η πρώτη από αυτές τις εντολές καθορίζει το μέγεθος της γραμματοσειράς, η δεύτερη και η τρίτη τη μορφή των σημαδιών στους άξονες, ενώ οι δύο τελευταίες τα σημεία και το κείμενο που εμφανίζονται στα σημάδια αυτά στον οριζόντιο άξονα[1].

5.3.2 Μορφοποίηση γραμμών (σύμβολα-χρώμα)

Το Matlab, όπως είδαμε, χρησιμοποιεί μια μπλε συνεχόμενη γραμμή για τον σχεδιασμό της γραφικής παράστασης μιας συνάρτησης. Αυτός ο τρόπος παρουσίασης των γραφικών παραστάσεων μπορεί να αλλάξει μεταβάλλοντας, παραδείγματος χάριν,

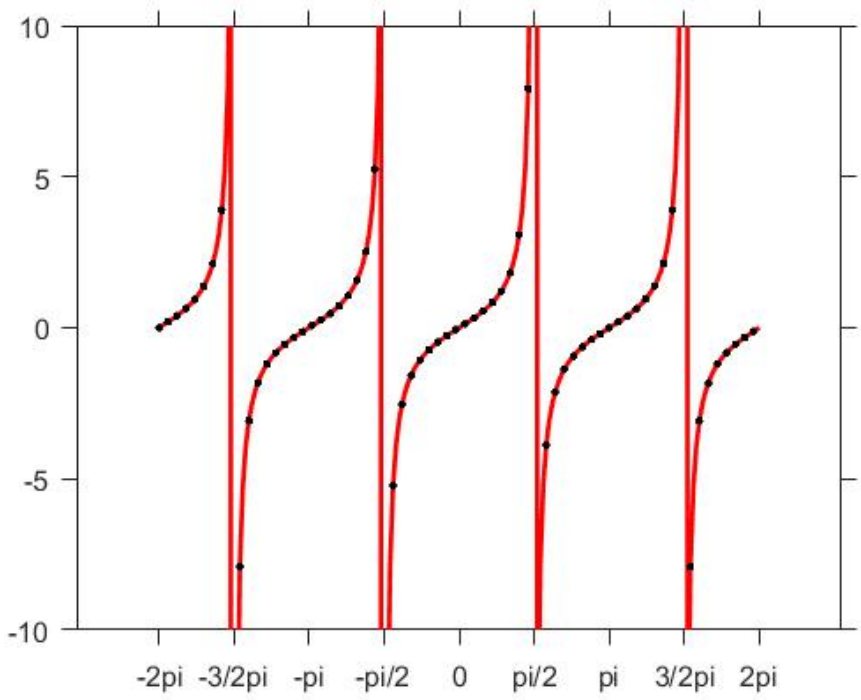
- το χρώμα και τη μορφή της γραμμής,
- την αποτύπωση των σημείων που χρησιμοποιούνται για την κατασκευή της γραφικής παράστασης.

Αυτό μπορεί να γίνει με την προσθήκη στην εντολή `plot(x,y)` επιπλέον παραμέτρων που διαμορφώνουν τη γραφική παράσταση της συνάρτησης. Παραδείγματος χάριν, η αντικατάσταση της εντολής `plot(x,y)` από την

```
>> plot(x,y,'-r.','LineWidth',2,'MarkerEdgeColor','k','MarkerSize',10,'
   MarkerIndices',1:3:length(y))
```

κατά την κατασκευή της γραφικής παράστασης της συνάρτησης της εφαπτομένης έχει ως αποτέλεσμα τη γραφική παράσταση του Σχήματος 5.8, στην οποία έχουμε επιλέξει με την εντολή `'-r.'` να σχεδιάσουμε με μια συνεχόμενη, κόκκινη γραμμή, στην οποία θα εμφανίζονται με τελείες τα σημεία (x_i, y_i). Με τα υπόλοιπα ορίσματα ορίζουμε το πάχος της γραμμής, το χρώμα και το μέγεθος των σημείων, καθώς και ότι δεν επιθυμούμε να αποτυπωθούν όλα τα σημεία, αλλά μόνο το 1ο, το 4ο, το 7ο και ούτω καθεξής.

[1] Με τις εντολές `ax.YTick` και `ax.YTickLabel` μπορούμε να αλλάξουμε και τον κάθετο άξονα.

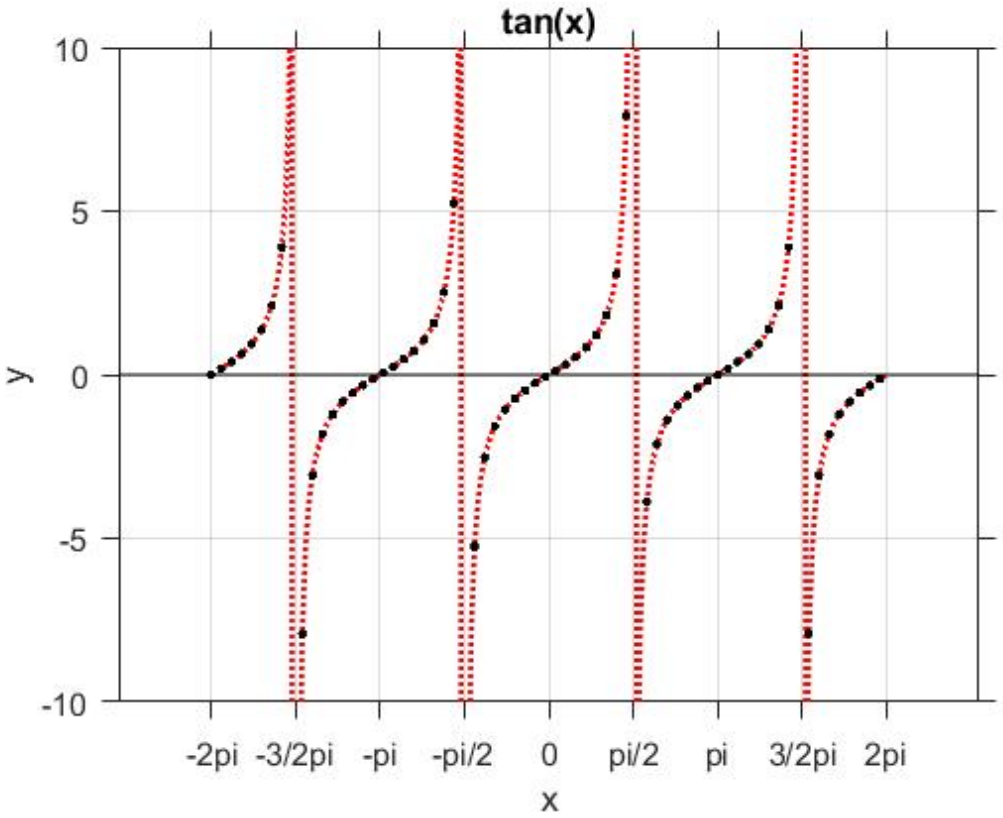


Σχήμα 5.8: Η γραφική παράσταση της συνάρτησης $f(x) = tan(x)$ στο διάστημα $[-2\pi, 2\pi]$, ύστερα από τη μορφοποίηση της καμπύλης της γραφικής παράστασης.

Πληκτρολογώντας την εντολή `help plot` λαμβάνουμε, μεταξύ άλλων, και τον ακόλουθο πίνακα με τις διαθέσιμες επιλογές για το χρώμα της γραμμής (1η στήλη), το σύμβολο για την αποτύπωση των σημείων του γραφήματος (2η στήλη) και τη μορφή της γραμμής (3η γραμμή).

```
------------------------------------------------------------
b     blue          .     point              -     solid
g     green         o     circle             :     dotted
r     red           x     x-mark             -.    dashdot
c     cyan          +     plus               -     dashed
m     magenta       *     star             (none)  no line
y     yellow        s     square
k     black         d     diamond
w     white         v     triangle (down)
                    ^     triangle (up)
                    <     triangle (left)
                    >     triangle (right)
                    p     pentagram
                    h     hexagram
------------------------------------------------------------
```

Έτσι, παραδείγματος χάριν, η εντολή `plot(x,y,'--ko')` έχει ως αποτέλεσμα την παραγωγή μιας γραφικής παράστασης, στην οποία η συνάρτηση αποτυπώνεται με μια διακεκομμένη, μαύρη γραμμή, ενώ στη γραφική παράσταση προστίθενται και τα σημεία (x_i, y_i), χρησιμοποιώντας έναν ανοικτό κύκλο.



Σχήμα 5.9: Η γραφική παράσταση της συνάρτησης $f(x) = tan(x)$ στο διάστημα $[-2\pi, 2\pi]$.

5.3.3 Εισαγωγή τίτλων, γραμμών αναφοράς και πλέγματος

Ένα σημαντικό βήμα στη μορφοποίηση των γραφικών παραστάσεων είναι η εισαγωγή τίτλων στο γράφημα και στους άξονες. Αυτό μπορεί να γίνει με τις εντολές `title`, `xlabel` και `ylabel`, όπως φαίνεται και στις παρακάτω εντολές:

```
>> title('tan(x)')
>> xlabel('x')
>> ylabel('y')
```

για τη γραφική παράσταση της συνάρτησης της εφαπτομένης.

Στις γραφικές παραστάσεις μπορούν να προστεθούν γραμμές αναφοράς ή/και ένα πλέγμα, που μπορούν να διευκολύνουν την ανάγνωση του γραφήματος. Οι γραμμές αναφοράς εισάγονται με την εντολή `refline(a,b)`, η οποία σχεδιάζει την ευθεία ax+b, ενώ το πλέγμα με την εντολή `grid`.

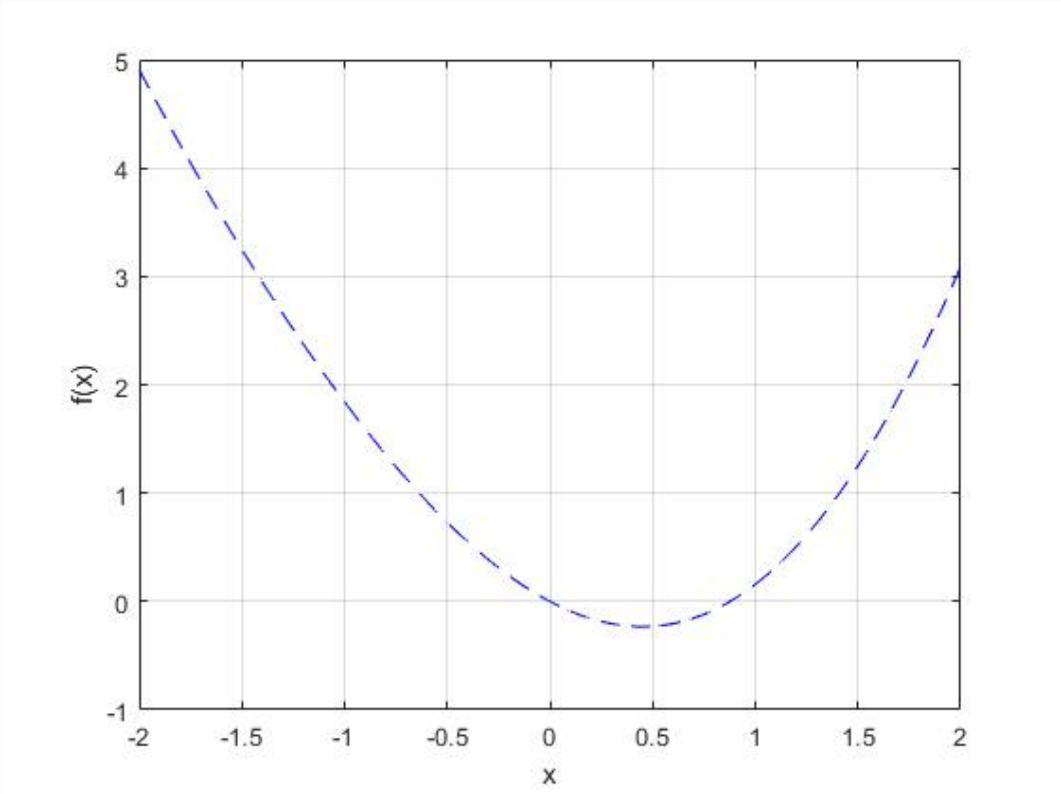
Με τις παρακάτω εντολές εισάγουμε την ευθεία $y = 0$, την οποία σχεδιάζουμε με μαύρο χρώμα (δεύτερη εντολή), καθώς και το πλέγμα, χρησιμοποιώντας τα σημεία που έχουμε ορίσει νωρίτερα στους άξονες.

```
>> hline=refline(0,0)
>> set(hline,'Color','k')
>> grid
```

Το αποτέλεσμα όλων των παραπάνω εντολών παρουσιάζεται στο Σχήμα 5.9.

Παράδειγμα 5.2

Να γράψετε τον κώδικα για την κατασκευή της γραφικής παράστασης της συνάρτησης $f(x) = x^2 - \sin(x)$ πάνω από το διάστημα $[-2, 2]$, όπως αυτή εμφανίζεται στη συνέχεια.



Σχήμα 5.10: Η γραφική παράσταση της συνάρτησης $f(x) = x^2 - sin(x)$ πάνω από το διάστημα $[-2, 2]$.

Λύση Παραδείγματος 5.2

Η γραφική παράσταση της συνάρτησης $f(x) = x^2 - sin(x)$ πάνω από το διάστημα $[-2, 2]$, όπως φαίνεται στο Σχήμα 5.10, γίνεται με την εκτέλεση των παρακάτω εντολών:

```
>> x=-2:0.01:2;
>> y=x.^2-sin(x);
>> plot(x,y,'--b')
>> xlabel('x')
>> ylabel('f(x)')
>> grid
```

στις οποίες έχουμε ορίσει τον τρόπο αποτύπωσης της καμπύλης (3η εντολή), τους τίλους των αξόνων (4η και 5η εντολή) και έχουμε εισάγει το πλέγμα με την εντολή `grid` στην τελευταία εντολή.

Άσκηση αυτοαξιολόγησης 5.3

Να γράψετε ένα αρχείο script για την κατασκευή της γραφικής παράστασης της συνάρτησης $f(x) = sin(x) \cdot cos(x)$ πάνω από ένα διάστημα που θα εισάγει ο χρήστης. Διαμορφώστε τον κώδικά σας, έτσι ώστε η καμπύλη της συνάρτησης να εμφανίζεται με μαύρο χρώμα, ενώ να εμφανίζεται και ένα πλέγμα πάνω από το γράφημα.

5.4 Πολλαπλές γραφικές παραστάσεις

Στις προηγούμενες ενότητες παρουσιάσαμε την κατασκευή και τη μορφοποίηση της γραφικής παράστασης μιας συνάρτησης. Συχνά επιθυμούμε να σχεδιάσουμε περισσότερες από μία συναρτήσεις στο ίδιο γράφημα ή να σχεδιάσουμε πολλές διαφορετικές συναρτήσεις σε διαφορετικά γραφήματα αλλά στο ίδιο παράθυρο. Στη συνέχεια, παρουσιάζουμε τις διαδικασίες που πρέπει να ακολουθήσουμε για να πετύχουμε αυτούς τους στόχους.

5.4.1 Πολλαπλές γραφικές παραστάσεις στο ίδιο γράφημα

Η βασική εντολή για τον σχεδιασμό δύο ή περισσότερων γραφικών παραστάσεων στο ίδιο γράφημα είναι η εντολή `hold`. Η εντολή `hold` διατηρεί το τρέχον γράφημα και τη μορφοποίησή του, έτσι ώστε οι επόμενες γραφικές παραστάσεις να αποτυπωθούν και αυτές στο ίδιο γράφημα.

Υπάρχουν τρεις βασικοί τρόποι εκτέλεσης της εντολής `hold`, οι οποίοι είναι:

- `hold ON` - δεσμεύει το τελευταίο γράφημα και σε αυτό αποτυπώνει όλα τα επόμενα γραφήματα,
- `hold OFF` - αποδεσμεύει το γράφημα και τα καινούργια γραφήματα αντικαθιστούν τις προηγούμενες γραφικές παραστάσεις,
- `hold` - αλλάζει την κατάσταση αναμονής του γραφήματος από `hold ON` σε `hold OFF`, αν το γράφημα είναι δεσμευμένο, και από `hold OFF` σε `hold ON`, διαφορετικά.

Η γραφική παράσταση του Σχήματος 5.11 αποτυπώνει το αποτέλεσμα των ακόλουθων εντολών

```
>> x=0:0.01:12*pi;
>> plot(x,sin(x),'r')
>> hold ON
Current plot held
>> plot(x,sqrt(x).*sin(x),'k')
>> hold OFF
```

Σημειώνεται ότι έχουμε επιλέξει την πρώτη γραφική παράσταση να την αποτυπώσουμε με κόκκινο χρώμα και τη δεύτερη με μαύρο.

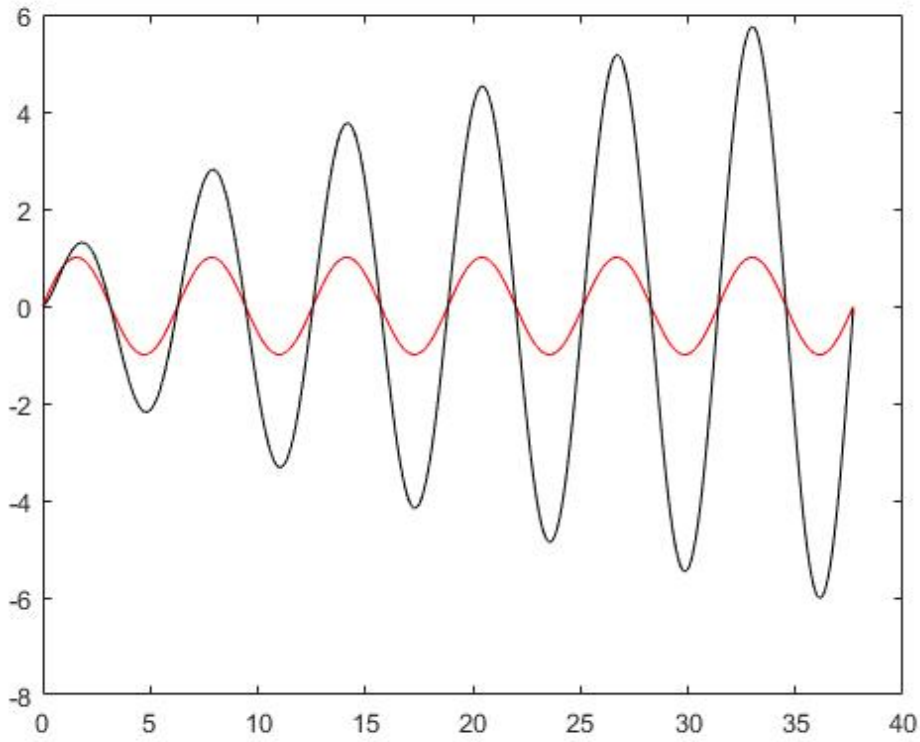
Παρατήρηση 5.5

Η εντολή `plot` παρέχει από μόνη της και έναν εναλλακτικό τρόπο σχεδιασμού πολλών γραφικών παραστάσεων στο ίδιο γράφημα. Καλώντας την με το ακόλουθο συντακτικό

plot(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...)

το Matlab συνδυάζει τα γραφήματα που ορίζονται από τις τριάδες (X_i, Y_i, S_i), όπου τα X_i και Y_i είναι διανύσματα με το ίδιο πλήθος στοιχείων που χρησιμοποιούνται για τον σχεδιασμό της καμπύλης και S_i είναι το κείμενο που καθορίζει τη μορφή της κάθε γραφικής παράστασης.
Με τις παρακάτω εντολές, παραδείγματος χάριν, μπορούμε να κατασκευάσουμε ξανά τη γραφική παράσταση του Σχήματος 5.11.

```
>> x=0:0.01:12*pi;
>> plot(x,sin(x),'r',x,sqrt(x).*sin(x),'k')
```



Σχήμα 5.11: Οι γραφικές παραστάσεις των συναρτήσεων $f(x) = sin(x)$ και $f(x) = \sqrt{x}sin(x)$ πάνω από το διάστημα $[0,12\pi]$.

Άσκηση αυτοαξιολόγησης 5.4

Να σχεδιάσετε τις γραφικές παραστάσεις των συναρτήσεων $f_1(x) = x^2$, $f_2(x) = \sqrt{x}$ και $f_3(x) = x$ στο ίδιο γράφημα για $x \in [1,3]$.

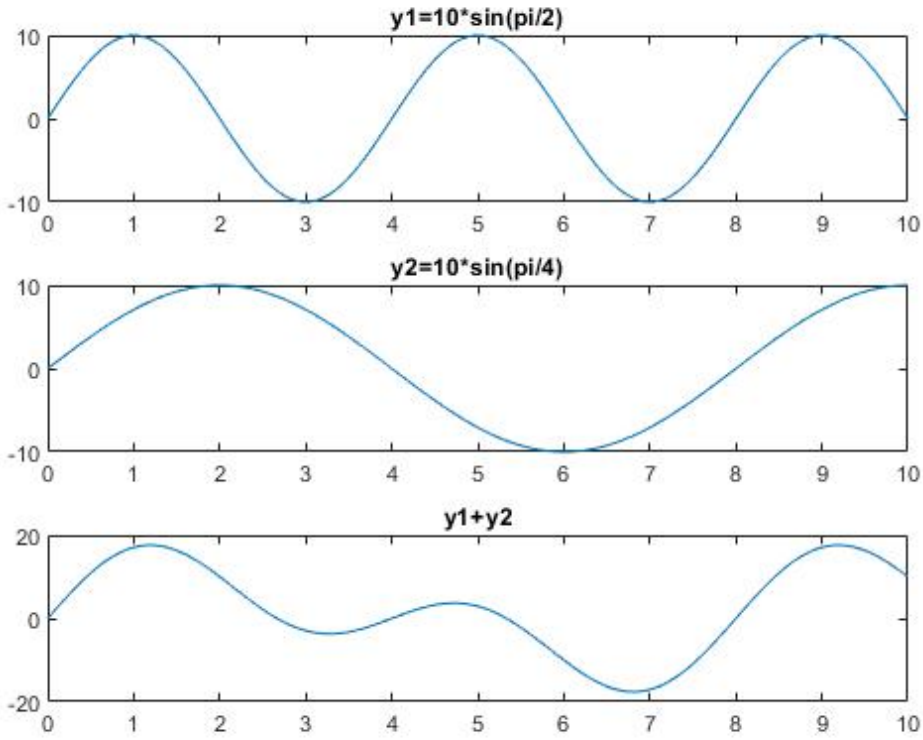
5.4.2 Πολλαπλές γραφικές παραστάσεις στο ίδιο παράθυρο

Η εντολή `subplot(m,n,p)` χωρίζει το παράθυρο του γραφήματος σε m γραμμές και n στήλες και επιλέγει τη θέση p, για να σχεδιάσει το επόμενο γράφημα. Η θέση του γραφήματος καθορίζεται μετρώντας κατά σειρά από αριστερά στα δεξιά.

Παραδείγματος χάριν, το γράφημα του Σχήματος 5.12 έχει σχεδιαστεί με τις ακόλουθες εντολές:

```
>> t=0:0.01:10;
>> y1=10*sin(pi/2*t); y2=10*sin(pi/4*t);
>> subplot(3,1,1)
>> plot(t,y1); title('y1=10*sin(pi/2)')
>> subplot(3,1,2)
>> plot(t,y2); title('y2=10*sin(pi/4)')
>> subplot(3,1,3)
>> plot(t,y1+y2); title('y1+y2')
```

Παρατηρήστε ότι μπορούμε να εκτελέσουμε οποιαδήποτε από τις εντολές μορφοποίησης, όπως την εισαγωγή τίτλου, που παρουσιάσαμε νωρίτερα, σε καθένα γράφημα.



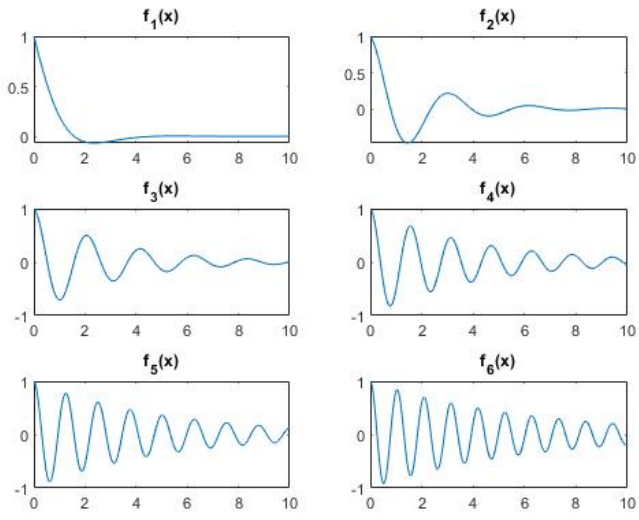
Σχήμα 5.12: Η γραφική παράσταση των συναρτήσεων $f_1(x) = 10 * sin(\pi/2)$, $f_2(x) = 10 * sin(\pi/4)$ και $f_1(x) + f_2(x)$ για $x \in [0,10]$.

Παράδειγμα 5.3

Στο παρακάτω γράφημα έχουν σχεδιαστεί στο ίδιο παράθυρο οι συναρτήσεις

$$f_k(x) = cos(kx) \cdot e^{-x/k}, \quad k = 1,2,\dots,6$$

για $0 \leq x \leq 10$



Να χρησιμοποιήσετε επαναληπτικές διαδικασίες για να κατασκευάσετε το γράφημα αυτό.

Λύση Παραδείγματος 5.3

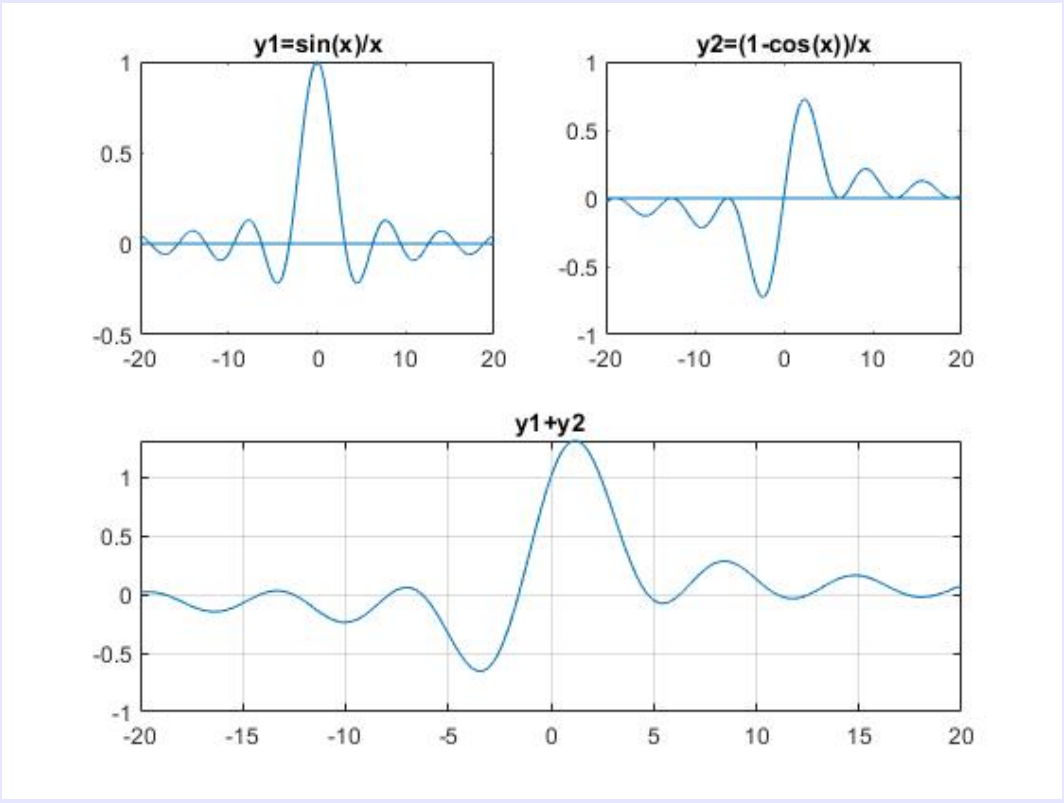
Αρχικά, παρατηρούμε ότι το παράθυρο έχει χωριστεί σε 3 γραμμές και 2 στήλες και ότι ο δείκτης k διατρέχει τα γραφήματα κατά γραμμή. Επομένως, αυτό που χρειάζεται είναι να ορίζουμε σε κάθε επανάληψη τη μεταβλητή y που θα σχεδιάζουμε και να επιλέγουμε τη θέση που θα σχεδιαστεί η κάθε γραφική παράσταση με την εντολή `subplot(3,2,k)`. Επομένως, οι ακόλουθες εντολές παράγουν το επιθυμητό αποτέλεσμα:

```
>> x=0:0.01:10;
>> for k=1:6
     subplot(3,2,k)
     y=cos(k*x).*exp(-x/k);
     plot(x,y)
     title(['f_',num2str(k),'(x)'])
   end
```

Σημειώνεται ότι το συντακτικό `'f_'`, που εμφανίζεται στην εντολή `title`, τοποθετεί το k που ακολουθεί ως δείκτη στο f.

Άσκηση αυτοαξιολόγησης 5.5

Να γράψετε τον κώδικα, για να κατασκευάσετε την παρακάτω γραφική παράσταση:



5.5 Ασκήσεις

Άσκηση 5.1. *Να σχεδιάσετε τη γραφική παράσταση της συνάρτησης*

$$f(x) = \left\lfloor \frac{x}{2} \right\rfloor - \left\lceil \frac{x}{2} \right\rceil$$

για $x \in [-5, 5]$*, όπου* $\lfloor \cdot \rfloor$ *και* $\lceil \cdot \rceil$ *η στρογγυλοποίηση ενός αριθμού προς τον πλησιέστερο ακέραιο προς το* $-\infty$ *και το* $+\infty$*, αντίστοιχα.*

Άσκηση 5.2. *Να σχεδιάσετε τη γραφική παράσταση της συνάρτησης*

$$f(x) = 4 \cdot cos(x) \cdot cos(10x)$$

για $x \in [-2\pi, 2\pi]$*. Εισάγετε στο γράφημα έναν κεντρικό τίτλο με την εξίσωση της συνάρτησης, καθώς και τους τίτλους στους άξονες.*

Στην Άσκηση Αυτοαξιολόγησης 2.4 είχε ζητηθεί να βρεθεί με ακρίβεια 0.001, αν υπάρχει, η πρώτη ρίζα της εξίσωσης $cos(x) \cdot sin(x) \cdot exp(-x/2) = 0$*, που είναι μεγαλύτερη ή ίση από το 5 και μικρότερη ή ίση από 10. Να επιβεβαιώσετε γραφικά ότι υπάρχει πράγματι μια ρίζα, η οποία ανήκει στο διάστημα* $[6.2830, 6.2840]$*. Σχεδιάστε και την ευθεία* $y = 0$ *για την καλύτερη επόπτευση της ρίζας.*

Άσκηση 5.3. *Υποθέστε ότι ο πληθυσμός (σε χιλιάδες) ενός συγκεκριμένου είδους εντόμων μετά από* t *μήνες περιγράφεται από την ακόλουθη σχέση*

$$P(t) = 3t + sin(4t) + 100.$$

Προσδιορίστε αναλυτικά με τη βοήθεια των μαθηματικών το ελάχιστο και το μέγιστο μέγεθος του πληθυσμού τους 4 πρώτους μήνες (Dawkins, 2018). Στη συνέχεια, κατασκευάστε τη γραφική παράσταση της $P(t)$ *και επιβεβαιώστε γραφικά τα αποτελέσματά σας.*

Να σχεδιάσετε στο ίδιο παράθυρο τις συναρτήσεις

$$f_k(x) = sin(kx) \cdot cos(x/k), \;\; k = 1, 2, \ldots, 9$$

για $0 \leq x \leq 10$*, χρησιμοποιώντας επαναληπτικές διαδικασίες. Χωρίστε το γράφημα σε 3 γραμμές και 3 στήλες.*

Άσκηση 5.4. *Σχεδιάστε στο ίδιο γράφημα τις συναρτήσεις*

$$f_n(x) = sin(nx)/\sqrt{n}, \;\; x \in \mathbb{R} \;\; n = 1, 2, \ldots,$$

για διάφορες τιμές του n*. Διαπιστώστε γραφικά, κατασκευάζοντας διαδοχικές γραφικές παραστάσεις (για κατάλληλες τιμές του* x*), ότι* $\lim_{n\to\infty} f_n(x) = 0 \;\forall\, x \in \mathbb{R}$*.*

Δείξτε, πάλι γραφικά, ότι δεν ισχύει το ίδιο για την παράγωγό της, $f_n'(x)$*.*

Άσκηση 5.5. *Σύμφωνα με το κριτήριο της παρεμβολής, αν* $g, f, h : A \to \mathbb{R}$ *τρεις πραγματικές συναρτήσεις τέτοιες, ώστε*

$$g(x) \leq f(x) \leq h(x), \forall\, x \in A$$

και

$$\lim_{x\to x_0} g(x) = L = \lim_{x\to x_0} h(x)$$

τότε ισχύει ότι

$$\lim_{x\to x_0} f(x) = L.$$

Χρησιμοποιώντας το παραπάνω κριτήριο να δείξετε γραφικά ότι

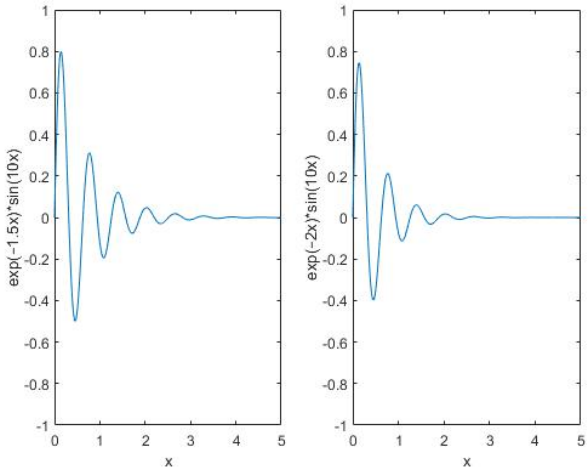
$$\lim_{x \to 0} x \cdot cos\frac{\pi}{x} = 0.$$

Υπόδειξη: *Σχεδιάστε στο ίδιο γράφημα την παραπάνω συνάρτηση και τις συναρτήσεις $|x|$ και $-|x|$ (Angenen, 2009).*

Άσκηση 5.6. *Ο παρακάτω κώδικας*

```
>> x = [0:0.01:5];
>> y = exp(-1.5*x).*sin(10*x);
>> subplot(2,1,1)
>> plot(x,y)
>> $('x')
>> $('exp(-1.5x)*sin(10x)')
>> axis([0 5 -1 1])
>> y = exp(-2*x).*sin(10*x);
>> subplot(2,1,2)
>> plot(x,y)
>> $('x')
>> $('exp-(2x)*sin(10x)')
>> axis([0 5 -1 1])
```

έχει σκοπό την κατασκευή της ακόλουθης γραφικής παράστασης.



Να εντοπίσετε και να διορθώσετε τα λάθη και να συμπληρώσετε τις εντολές ($) που λείπουν από τον κώδικα.

5.6 Απαντήσεις στις ασκήσεις αυτοαξιολόγησης

Λύση άσκησης αυτοαξιολόγησης 5.1

Για την κατασκευή του ευθύγραμμου τμήματος που ενώνει τα σημεία (2,3) και (10,5) αρκεί να ορίσουμε τις μεταβλητές $x = [2,10]$ και $y = [2,3]$ με τις τετμημένες και τεταγμένες των δύο σημείων και, στη συνέχεια, να χρησιμοποιούμε την εντολή `plot(x,y)`. Επομένως, η εκτέλεση των επόμενων εντολών θα έχει ως αποτέλεσμα την κατασκευή της ζητούμενης γραφικής παράστασης.

```
>> x=[2,10];
>> y=[3,5];
>> plot(x,y)
```

Λύση άσκησης αυτοαξιολόγησης 5.2

Σωστή επιλογή είναι η τέταρτη εντολή, αφού όχι μόνο ορίζει σωστά τη συνάρτηση και τον πολλαπλασιασμό στοιχείο προς στοιχείο των διανυσμάτων x και $cos(2x)$ αλλά ορίζει σωστά και τα όρια του διαστήματος πάνω από το οποίο θα σχεδιαστεί η γραφική παράσταση της συνάρτησης, χρησιμοποιώντας το σωστό σύμβολο για το π.

Λύση άσκησης αυτοαξιολόγησης 5.3

Για να δημιουργήσουμε ένα αρχείο script για την κατασκευή της γραφικής παράστασης της συνάρτησης $f(x) = sin(x) \cdot cos(x)$ πάνω από ένα διάστημα που θα εισάγει ο χρήστης, χρειάζεται να πληκτρολογήσουμε τις ακόλουθες εντολές και να τις αποθηκεύσουμε σε ένα αρχείο `m`.

```
1  %PLOTSINCOS plots f(x)=sin(x)cos(x) over an interval [xmin xmax]
2  % given by the user
3  % xmin should be smaller than xmax
4  interval=input('Please type the interval [xmin xmax]:');
5  if ~all(size(interval)==[1 2])
6      disp('input error: please type a 1x2 vector')
7  else
8      xmin=interval(1);
9      xmax=interval(2);
10     if xmin>=xmax
11        disp('input error: please type a vector with xmin<xmax')
12     else
13         x=xmin:(xmax-xmin)/200:xmax;
14         y=sin(x).*cos(x);
15         plot(x,y,'k')
16         grid
17     end
18 end
```

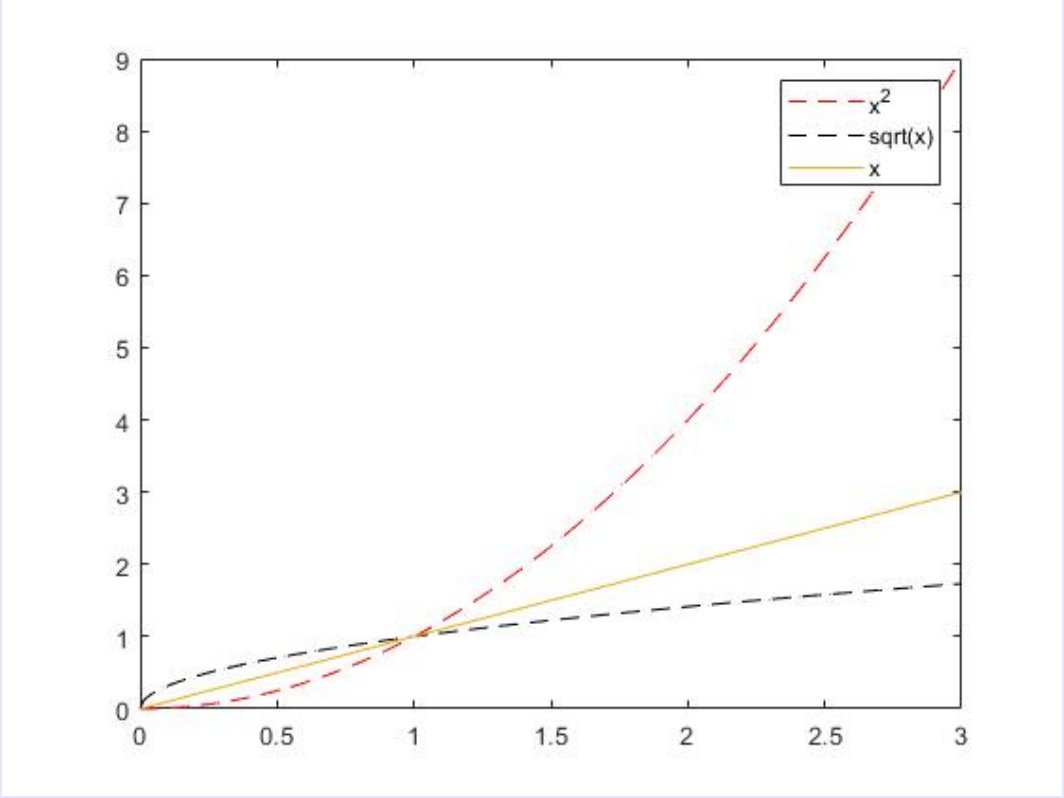
Οι έλεγχοι `if` είναι απαραίτητοι, έτσι ώστε να εξασφαλίσουμε ότι ο χρήστης θα εισάγει ένα 1×2 διάνυσμα με το πρώτο στοιχείο να είναι μικρότερο από το δεύτερο. Αν ο χρήστης δεν εισάγει ένα τέτοιο διάνυσμα, το πρόγραμμα τον πληροφορεί για το λάθος του. Αν από την άλλη εισάγει ένα σωστό διάγραμμα, τότε οι εντολές `plot(x,y,'k')` και `grid` εξασφαλίζουν ότι η γραφική παράσταση θα έχει την επιθυμητή μορφή.

Λύση άσκησης αυτοαξιολόγησης 5.4

Ο σχεδιασμός των γραφικών παραστάσεων των συναρτήσεων $f_1(x) = x^2$, $f_2(x) = \sqrt{x}$ και $f_3(x) = x$ στο ίδιο γράφημα για $x \in [1,3]$ μπορεί να γίνει με τις ακόλουθες εντολές:

```
>> x=0:0.01:3;
>> plot(x,x.^2,'--r')
>> hold ON
>> plot(x,sqrt(x),'--k')
>> plot(x,x,'--b')
>> hold OFF
>> legend('x^2','sqrt(x)','x')
```

Οι εντολές αυτές παράγουν το Σχήμα 5.13. Σημειώνεται ότι οι παραπάνω εντολές έχουν ως στόχο τον σχεδιασμό των τριών συναρτήσεων με διαφορετική μορφή έτσι ώστε αυτές να ξεχωρίζουν πιο εύκολα. Η τελευταία εντολή προσθέτει στο πάνω δεξί μέρος του παραθύρου μια επεξηγηματική λεζάντα για τις συναρτήσεις που έχουν σχεδιαστεί στο γράφημα. Σημειώνεται ότι το συντακτικό x^2, που εμφανίζεται στην τελευταία εντολή, τοποθετεί το 2 που ακολουθεί ως εκθέτη στο x.



Σχήμα 5.13: Η γραφική παράσταση των συναρτήσεων $f_1(x) = x^2$, $f_2(x) = \sqrt{x}$ και $f_3(x) = x$ στο ίδιο γράφημα για $x \in [1,3]$.

Λύση άσκησης αυτοαξιολόγησης 5.5

Η ζητούμενη γραφική παράσταση μπορεί να κατασκευαστεί με τις ακόλουθες εντολές:

```
>> x=-20:0.01:20;
>> subplot(2,2,1)
>> plot(x,sin(x)./x)
>> title('y1=sin(x)/x')
```

```
>> refline(0,0)
>> subplot(2,2,2)
>> plot(x,(1-cos(x))./x)
>> title('y2=(1-cos(x))/x')
>> refline(0,0)
>> subplot(2,2,[3,4])
>> plot(x,sin(x)./x+(1-cos(x))./x)
>> title('y1+y2')
>> grid
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ξενόγλωσση

Angenen, S. (2009). *MATH 221 – 1st Semester Calculus Lecture Notes Version 2.0*. Free Software Foundation.

Dawkins, P. (2018). *Calculus I*. Lamar University, Beaumont, Texas.

ΚΕΦΑΛΑΙΟ 6

ΔΗΜΙΟΥΡΓΙΑ ΤΡΙΔΙΑΣΤΑΤΩΝ ΓΡΑΦΙΚΩΝ ΠΑΡΑΣΤΑΣΕΩΝ

Σύνοψη:

Σε αυτό το κεφάλαιο παρουσιάζονται οι βασικοί τρόποι κατασκευής και διαμόρφωσης τριδιάστατων γραφικών παραστάσεων. Πιο συγκεκριμένα, αρχικά δίνεται έμφαση στη φιλοσοφία κατασκευής των τριδιάστατων γραφικών παραστάσεων μέσω της δημιουργίας ενός πλέγματος σημείων, με την εντολή `meshgrid` στο επίπεδο και τον υπολογισμό των τιμών της ζητούμενης συνάρτησης πάνω από τα σημεία αυτά. Στη συνέχεια, παρουσιάζονται οι διάφορες εντολές για την κατασκευή και μορφοποίηση των τριδιάστατων γραφικών παραστάσεων.

Προαπαιτούμενη γνώση: Τα κεφάλαια 2,3, 4 και 5 του παρόντος συγγράμματος.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του κεφαλαίου αυτού, θα γνωρίζετε τρόπους:

- κατασκευής τριδιάστατων γραφικών παραστάσεων,
- μορφοποίησης τριδιάστατων γραφικών παραστάσεων.

Γλωσσάριο επιστημονικών όρων

- Γράφημα έντασης
- Γράφημα ισοϋψών
- Πλέγμα
- Οριζόντιο επίπεδο αναφοράς

6.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο παρουσιάστηκαν οι διαδικασίες κατασκευής και μορφοποίησης διδιάστατων γραφικών παραστάσεων. Στο παρόν κεφάλαιο, παρατίθενται οι αντίστοιχες διαδικασίες για την κατασκευή και μορφοποίηση τριδιάστατων γραφικών παραστάσεων. Προτού όμως προβούμε στην αναλυτική παρουσίαση των διαδικασιών αυτών, θα παρουσιαστεί η φιλοσοφία κατασκευής των γραφικών παραστάσεων σε ένα υπολογιστικό πρόγραμμα, όπως το Matlab, και η απαραίτητη εντολή `meshgrid` για την επίτευξη αυτού του στόχου.

6.1.1 Η εντολή meshgrid

Τα βασικά βήματα για την κατασκευή της διδιάστατης γραφικής παράστασης μιας συνάρτησης $y = f(x)$ είναι:

1. ο ορισμός ενός συνόλου τιμών $x_i, i = 1,2,\dots n$, τα οποία ανήκουν στο διάστημα πάνω από το οποίο θέλουμε να σχεδιάσουμε την $y = f(x)$,
2. ο υπολογισμός της τιμής της $y = f(x)$ για $x = x_i$ και
3. η ένωση με ευθύγραμμα τμήματα των σημείων $(x_i, y_i),\ i = 1,2,\dots,n$.

Παρόμοια φιλοσοφία ακολουθείται και για τον σχεδιασμό της τριδιάστατης γραφικής παράστασης μιας συνάρτησης $z = f(x,y)$. Πιο συγκεκριμένα,

1. αρχικά, ορίζεται ένα πλέγμα σημείων $(x_i, y_j), i = 1,2,\dots n_x, j = 1,2,\dots n_y$ στο επίπεδο xy, τα οποία ανήκουν στο χωρίο πάνω από το οποίο θέλουμε να σχεδιάσουμε την $z = f(x,y)$,
2. στη συνέχεια, πραγματοποιείται ο υπολογισμός της τιμής της $z = f(x,y)$ για κάθε ζευγάρι τιμών $(x,y) = (x_i, y_j)$ και
3. τέλος, τα σημεία $(x_i, y_j, z_{ij}), i = 1,2,\dots n_x, j = 1,2,\dots n_y$ ενώνονται με τετράπλευρες επιφάνειες, δημιουργώντας μια προσέγγιση της επιφάνειας της συνάρτησης $z = f(x,y)$.

Όπως και στη διδιάστατη περίπτωση, το πλήθος n_x και n_y σημείων, τα οποία καθορίζουν και το πλήθος των σημείων του πλέγματος, είναι αυτό που ευθύνεται για το αν η τριδιάστατη γραφική παράσταση έχει ικανοποιητική μορφή ή όχι. Παραδείγματος χάριν, ένας μικρός αριθμός για τα n_x και n_y δεν θα μπορέσει πιθανά να αποδώσει την καμπυλότητα της συνάρτησης, αποτυπώνοντας έτσι μια ελλιπή εικόνα για τη συμπεριφορά της συνάρτησης.

Για την κατασκευή του πλέγματος το Matlab διαθέτει μια πολύ βοηθητική εντολή, η οποία επιτρέπει εύκολα τον ορισμό του πλέγματος σημείων $(x_i, y_j), i = 1,2,\dots n_x, j = 1,2,\dots n_y$ στο επίπεδο xy. Η εντολή αυτή είναι η `meshgrid`, η οποία επιτρέπει, με βάση τον ορισμό ενός συνόλου τιμών $x_i, i = 1,2,\dots n_x$ στον άξονα x και ενός συνόλου τιμών $y_j, j = 1,2,\dots n_y$, να λάβουμε τις συντεταγμένες του πλέγματος που ορίζουν τα σημεία αυτά σε κατάλληλη μορφή, έτσι ώστε να υπολογίσουμε στη συνέχεια τις τιμές της $z = f(x,y)$ για όλα τα $(x,y) = (x_i, y_j)$.

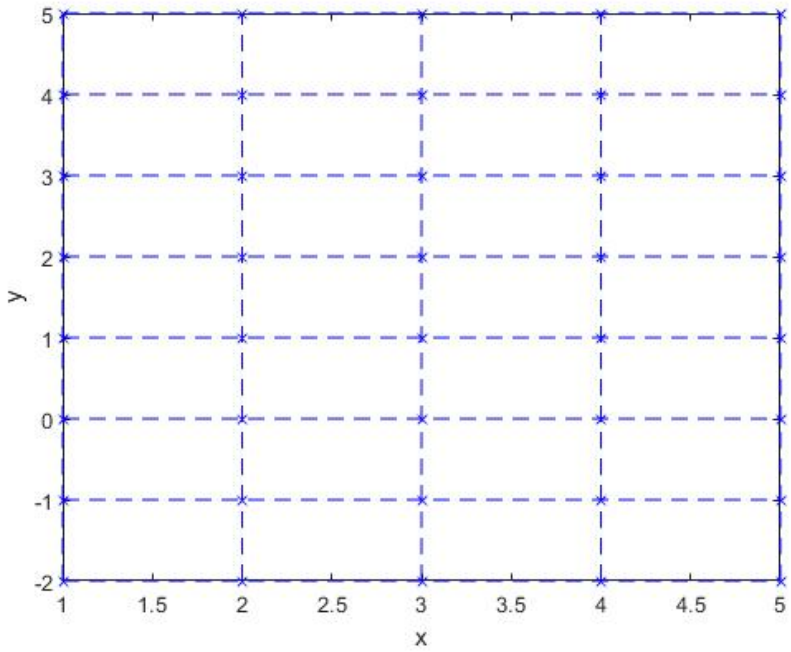
Για να γίνει πιο κατανοητή η λειτουργία της εντολής `meshgrid` παρατίθεται το παρακάτω παράδειγμα κατασκευής του πλέγματος σημείων στο επίπεδο xy που εμφανίζεται στο Σχήμα 6.1. Το πλέγμα αυτό βασίζεται αρχικά στον ορισμό των σημείων $1,2,3,4,5$ στον άξονα x και των σημείων $-2,-1,0,1,2,3,4,5$ στον άξονα y, δηλαδή σε 5 και σε 8 σημεία, αντίστοιχα, και δημιουργείται με τον ακόλουθο κώδικα.

```
>> x=1:5;
>> y=-2:5;
>> [X, Y] = meshgrid(1:5, -2:5)
```

```
X =
     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5

Y =
    -2    -2    -2    -2    -2
    -1    -1    -1    -1    -1
     0     0     0     0     0
     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
     4     4     4     4     4
     5     5     5     5     5
```

Από το παραπάνω παράδειγμα αντιλαμβανόμαστε ότι η εντολή `meshgrid` ορίζεται πάνω στα διανύσματα που ορίζουν τα σημεία στους δύο άξονες και επιστρέφει δύο 5×8 πίνακες. Ο πίνακας X είναι ένας πίνακας, του οποίου κάθε σειρά είναι ένα αντίγραφο του διανύσματος $[1,2,3,4,5]$, δηλαδή του διανύσματος που ορίζει τα σημεία στον άξονα x. Από την άλλη, ο πίνακας Y αποτελείται από αντίγραφα (σε στήλες) του διανύσματος $[-2,-1,0,1,2,3,4,5]$, δηλαδή του διανύσματος που ορίζει τα σημεία στον άξονα y.

Οι δύο αυτοί πίνακες αντιπροσωπεύουν τις συντεταγμένες X και Y των σημείων του πλέγματος, με τον



Σχήμα 6.1: Γραφική απεικόνιση του πλέγματος σημείων στο επίπεδο xy που δημιουργείται με την εντολή `meshgrid`.

τρόπο που αυτά παρουσιάζονται στο Σχήμα 6.1. Οι πίνακες αυτοί μπορούν να χρησιμοποιηθούν, όπως θα συζητήσουμε στην ενότητα που ακολουθεί, για τον υπολογισμό των τιμών της $z = f(x,y)$ για όλα τα $(x,y) = (x_i, y_j)$.

6.2 Κατασκευή τριδιάστατων γραφικών παραστάσεων

Η βασική εντολή κατασκευής τριδιάστατων γραφικών παραστάσεων στο Matlab είναι η εντολή `surf`. Το βασικό συντακτικό της εντολής `surf` είναι το `surf(X,Y,Z)`, όπου τα X, Y και Z είναι $n_x \times n_y$ πίνακες με

- τον X να περιέχει τις συντεταγμένες ως προς τον άξονα x,
- τον Y να περιέχει τις συντεταγμένες ως προς τον άξονα y

των σημείων του πλέγματος στο επίπεδο xy. Ο πίνακας Z περιέχει τις τιμές της συνάρτησης $z = f(x,y)$ για όλα τα $(x,y) = (x_i, y_j), i = 1,2 \ldots, n_x, j = 1,2 \ldots, n_y$. Στην ουσία, οι τρεις αυτοί πίνακες ορίζουν τα σημεία $(x_i, y_j, z_{ij}), i = 1,2,\ldots n_x, j = 1,2,\ldots, n_y$, με τη βοήθεια των οποίων σχεδιάζεται η γραφική παράσταση της συνάρτησης $z = f(x,y)$.

Παρατήρηση 6.1

Σημειώνεται ότι οι πίνακες X και Y μπορούν να κατασκευαστούν εύκολα με την εντολή `meshgrid`, την οποία αναφέραμε νωρίτερα. Ο πίνακας Z μπορεί να οριστεί χρησιμοποιώντας τις πράξεις στοιχείο προς στοιχείο μεταξύ πινάκων, όπως αυτές παρουσιάστηκαν στο Κεφάλαιο 1.

Παράδειγμα 6.1

Να κατασκευαστεί η γραφική παράσταση της συνάρτησης

$$f(x,y) = \cos(x + y^2)\exp(-x)$$

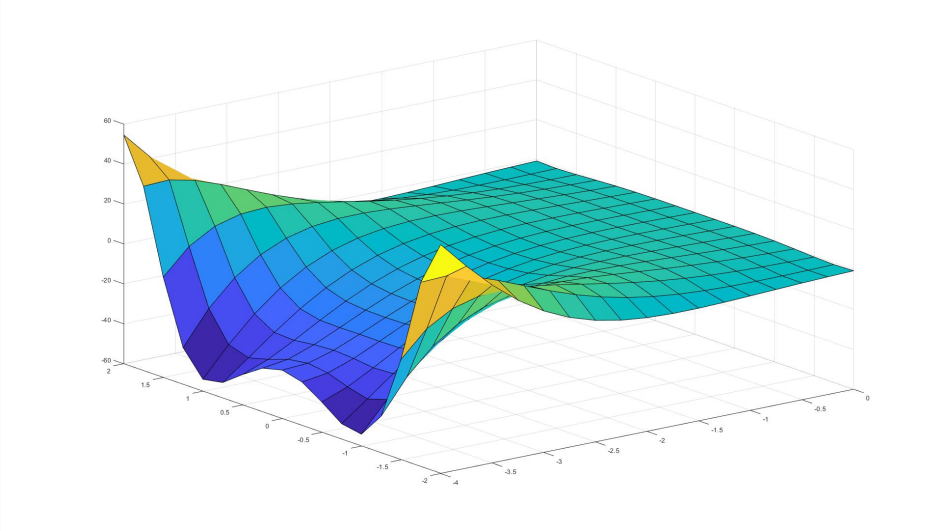
για $x \in [-5,0]$ και $y \in [-2,2]$.

Λύση Παραδείγματος 6.1

Για την κατασκευή της γραφικής παράστασης πρέπει, αρχικά, να κατασκευάσουμε τους πίνακες X και Y που περιέχουν τις συντεταγμένες των σημείων (x_i, y_j) ενός πλέγματος στο επίπεδο xy. Για την κατασκευή αυτών των πινάκων θα χρησιμοποιήσουμε την εντολή `meshgrid`. Επειδή ζητείται η γραφική παράσταση να αποτυπωθεί για $x \in [-4,0]$ και $y \in [-2,2]$, αρχικά θα ορίσουμε διανύσματα με σημεία από τα διαστήματα αυτά και, στη συνέχεια, θα εκτελέσουμε την εντολή `meshgrid`. Οι εντολές που πρέπει να εκτελεστούν εμφανίζονται παρακάτω.

```
>> x=-4:0.25:0;
>> y=-2:0.25:2;
>> [X, Y] = meshgrid(x,y);
>> Z=cos(X+Y.^2).*exp(-X);
>> surf(X,Y,Z)
```

Οι δύο τελευταίες εντολές υπολογίζουν τις τιμές της συνάρτησης $f(x,y)$ για τα σημεία του πλέγματος και κατασκευάζουν τη γραφική παράσταση του Σχήματος 6.2, αντίστοιχα.



Σχήμα 6.2: Η γραφική παράσταση της συνάρτησης $f(x,y) = cos(x + y^2)exp(-x)$ για $x \in [-4,0]$ και $y \in [-2,2]$.

Άσκηση αυτοαξιολόγησης 6.1

Χρησιμοποιώντας τις εντολές hold on και holf off, να κατασκευάσετε τις γραφικές παραστάσεις των συναρτήσεων $f_1(x,y) = y \cdot e^{x^2-5}$ και $f_2(x,y) = \frac{1}{2}x \cdot \cos(y)$ για $x \in [-3,3]$ και $y \in [-3,3]$ σε κοινό γράφημα.

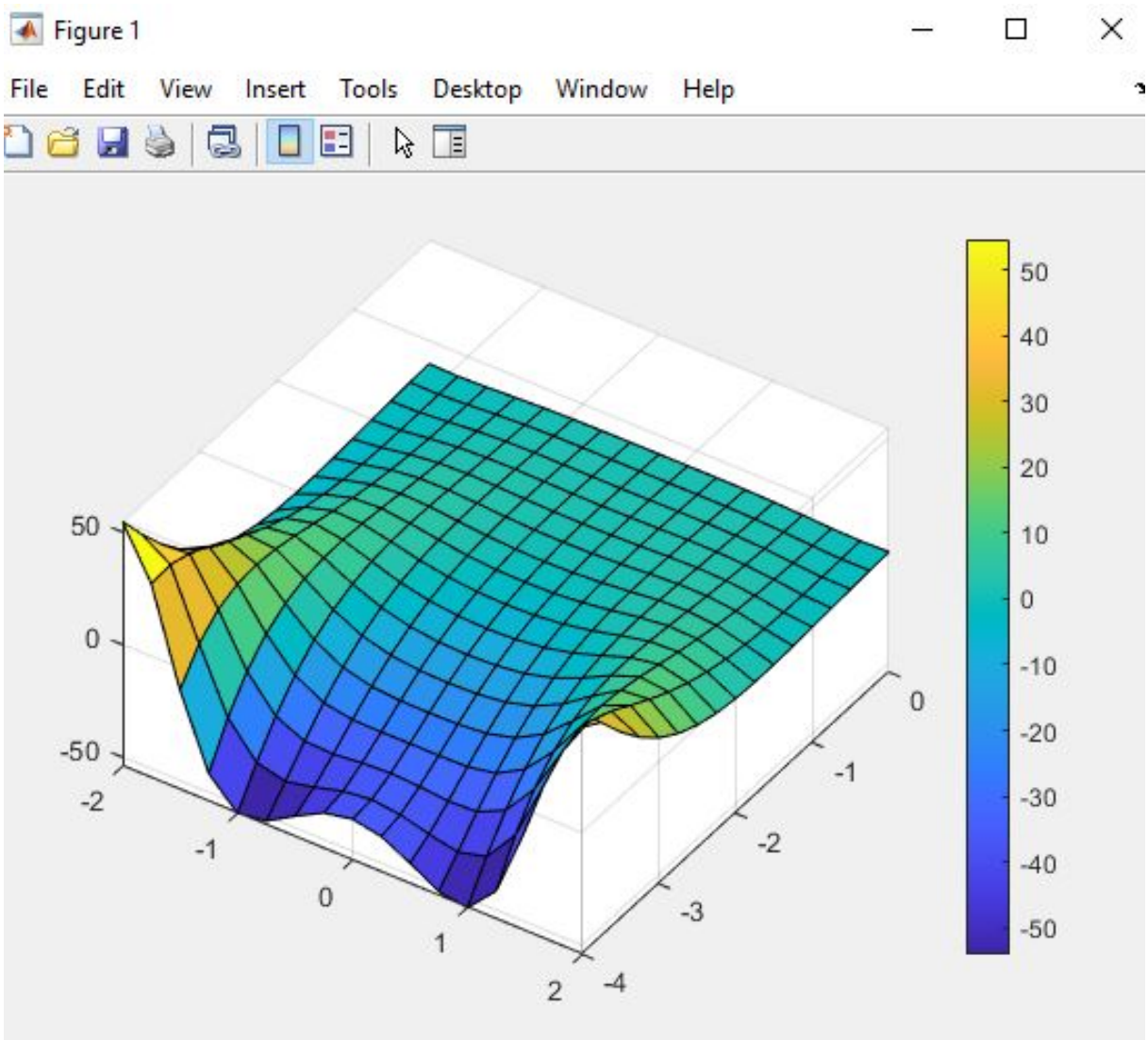
6.2.1 Επισκόπηση τριδιάστατων γραφικών παραστάσεων

Το Matlab χρωματίζει τις τριδιάστατες γραφικές παραστάσεις με βάση την τιμή της συνάρτησης. Μεγάλες τιμές απεικονίζονται με ανοιχτόχρωμο κίτρινο, ενώ μικρές τιμές απεικονίζονται με σκούρο μπλε χρώμα (βλ. Σχήμα 6.2). Ενδιάμεσες τιμές αποτυπώνονται χρησιμοποιώντας ενδιάμεσους χρωματικούς τόνους. Αυτή η χρωματική αποτύπωση των τιμών μπορεί να ερμηνευτεί ακόμα πιο εύκολα προσθέτοντας στο γράφημα μια χρωματική μπάρα, όπως φαίνεται στο Σχήμα 6.3.

Η χρωματική μπάρα μπορεί να προστεθεί εκτελώντας την εντολή `colorbar` στο Command window. Εναλλακτικά, η χρωματική μπάρα μπορεί να προστεθεί κάνοντας κλικ στο εικονίδιο με τη χρωματική μπάρα στο παράθυρο αποτύπωσης στο Matlab της γραφικής παράστασης, όπως φαίνεται στο Σχήμα 6.3.

Το Matlab επιτρέπει και την περιστροφή του γραφήματος και την επόπτευσή του από διαφορετική γωνία. Η περιστροφή του γραφήματος μπορεί να γίνει πατώντας συνεχώς το αριστερό κλικ του ποντικιού και μετακινώντας τον κέρσορα δεξιά, αριστερά, πάνω και κάτω. Εναλλακτικά, μπορούμε να πληκτρολογήσουμε την εντολή `view`.

Η εντολή `view` έχει δύο διαφορετικά συντακτικά. Το συντακτικό `view([AZ, EL])` ορίζει τη γωνία θέασης του τρέχοντος τριδιάστατου γραφήματος. Το AZ ορίζει το αζιμούθιο ή, αλλιώς, την οριζόντια περιστροφή και το EL την κατακόρυφη περιστροφή του γραφήματος (και τα δύο σε μοίρες). Η εντολή `view` μπορεί να εκτελεστεί και ως `view([X, Y, Z])` ορίζοντας το σημείο θέασης σε καρτεσιανές συντεταγμένες. Το σημείο θέασης του Σχήματος 6.3 έχει οριστεί πληκτρολογώντας την εντολή view([3,-2,-5]), η οποία ορίζει το σημείο $[3,-2,-5]$ από το οποίο εποπτεύεται η γραφική παράσταση.



Σχήμα 6.3: Το παράθυρο αποτύπωσης στο Matlab της γραφικής παράστασης της συνάρτησης $f(x,y) = cos(x + y^2)exp(-x)$ για $x \in [-4,0]$ και $y \in [-2,2]$. Στη γραφική παράσταση έχει προστεθεί η χρωματική μπάρα αποτύπωσης των τιμών της συνάρτησης, ενώ η ίδια η γραφική παράσταση έχει στραφεί.

6.2.2 Άλλες εντολές κατασκευής τριδιάστατων γραφικών παραστάσεων

Το Matlab παρέχει και άλλες εντολές πέρα από την `surf` για την αποτύπωση της συμπεριφοράς διμεταβλητών συναρτήσεων. Μερικές από τις εντολές αυτές είναι οι:

- `mesh`
- `contour`
- `surfc`
- `meshc`
- `polar`

6.2.2.1 Η εντολή mesh

Η εντολή `mesh` είναι παρόμοια και συντάσσεται με τον ίδιο ακριβώς τρόπο με την εντολή `surf`. Στο Σχήμα 6.4 παρουσιάζεται η γραφική παράσταση της συνάρτησης

$$f(x,y) = sin(\sqrt{x^2 + y^2}) \text{ για } x,y \in [-6,6]$$

με την εντολή `surf` (αριστερό γράφημα) και με την εντολή `mesh` (δεξί γράφημα). Η βασική διαφορά των δύο γραφικών παραστάσεων είναι ότι η εντολή `mesh` απενεργοποιεί τον χρωματισμό των τετράπλευρων που ορίζουν το γράφημα και χρωματίζει μόνο τις πλευρές τους. Αυτό, σε κάποιες περιπτώσεις, επιτρέπει την καλύτερη επόπτευση της συμπεριφοράς μιας συνάρτησης σε σχέση με τη γραφική παράσταση που δημιουργεί η εντολή `surf`. Αυτό συμβαίνει γιατί ο χρωματισμός των τετράπλευρων με τον ταυτόχρονο χρωματισμό με μαύρες γραμμές των πλευρών τους, που χρησιμοποιεί η εντολή `surf`, μπορεί να μην αποτυπώνει καθαρά την εικόνα. Μια τέτοια χαρακτηριστική περίπτωση είναι όταν τα σημεία του πλέγματος είναι πολύ πυκνά, δηλαδή κοντά το ένα στο άλλο. Μπορείτε να δοκιμάσετε να αποτυπώσετε τη γραφική παράσταση της $f(x,y)$, χρησιμοποιώντας ένα πυκνό πλέγμα χωρίζοντας τα διαστήματα $[-6,6]$ στους άξονες x και y σε σημεία που η απόσταση μεταξύ τους ισούται με 0.1.

6.2.2.2 Οι εντολές contour, surfc, meshc και pcolor

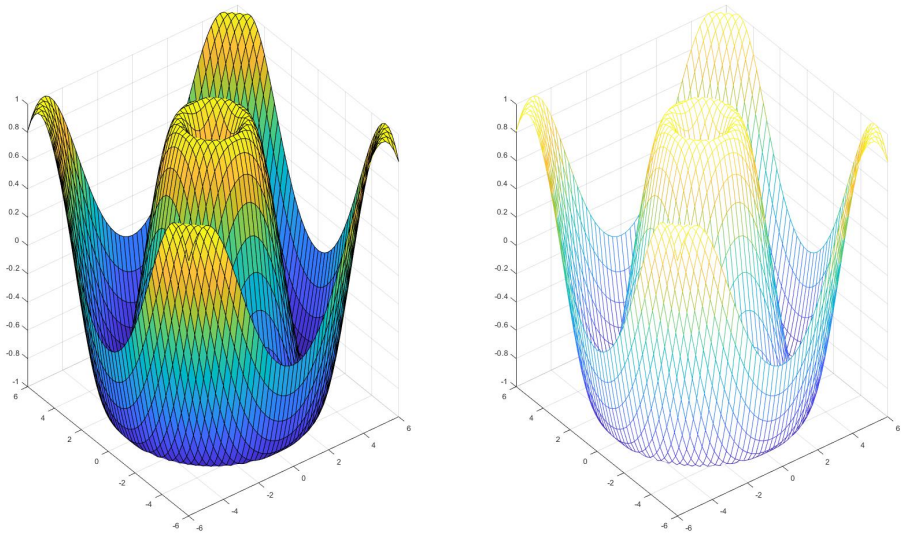
Μια ενδιαφέρουσα επιλογή για την αποτύπωση της συμπεριφοράς μιας διμεταβλητής είναι το γράφημα ισοϋψών, όπως αυτό που φαίνεται στο Σχήμα 6.5 για τη συνάρτηση $f(x,y) = sin(\sqrt{x^2 + y^2})$ για $x,y \in [-6,6]$. Το γράφημα αυτό αποτυπώνει στο επίπεδο xy τη συμπεριφορά της συνάρτησης σχεδιάζοντας γραμμές, οι οποίες συνδέουν σημεία με την ίδια τιμή. Οι γραμμές αυτές χρωματίζονται με χρώμα αναλόγως με την τιμή τους. Με το γράφημα ισοϋψών μπορούν εύκολα να εντοπιστούν συμμετρίες και τοπικά ακρότατα.

Στο Matlab το γράφημα ισοϋψών κατασκευάζεται με την εντολή `contour`, η οποία συντάσσεται με τον ίδιο τρόπο με τις `surf` και `mesh`, δηλαδή ως `contour(X,Y,Z)`. Με τις εντολές `surfcr(X,Y,Z)` και `meshcr(X,Y,Z)` κατασκευάζονται οι γραφικές παραστάσεις του Σχήματος 6.6, στις οποίες έχουν αποτυπωθεί οι τριδιάστατες γραφικές παραστάσεις της $f(x,y) = sin(\sqrt{x^2 + y^2})$ για $x,y \in [-6,6]$ με τις εντολές `surfc` (αριστερό γράφημα) και `meshc` (δεξί γράφημα) έχοντας τοποθετήσει σε αυτές και το γράφημα ισοϋψών.

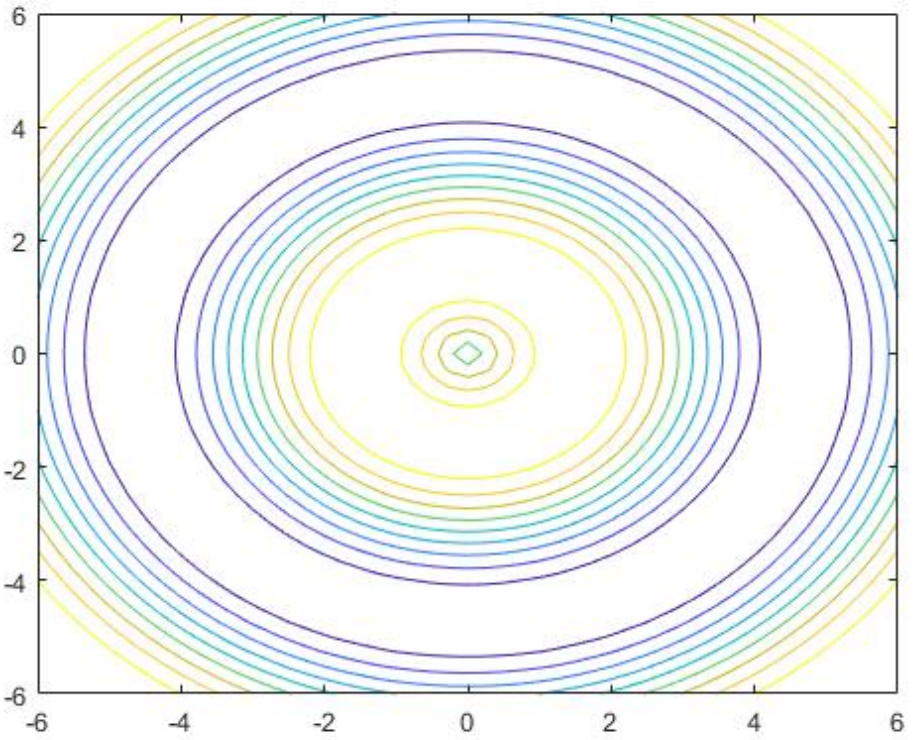
Μια παρόμοια εντολή με την εντολή `contour` είναι η εντολή pcolor. Η εντολή `pcolor(X,Y,Z)` σχεδιάζει γραφικές παραστάσεις της μορφής του Σχήματος 6.7. Αυτές οι γραφικές παραστάσεις αναφέρονται και ως γραφήματα έντασης και χρωματίζουν ολόκληρο το επίπεδο xy σύμφωνα με την τιμή της συνάρτησης.

6.3 Μορφοποίηση τριδιάστατων γραφικών παραστάσεων

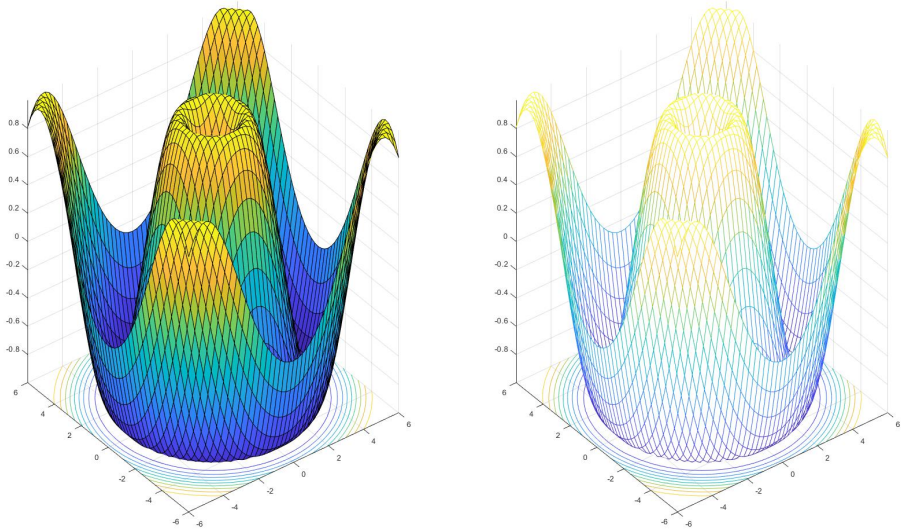
Οι τριδιάστατες γραφικές παραστάσεις στο Matlab μπορούν να μορφοποιηθούν έτσι ώστε να πληρούν τις απαιτήσεις του χρήστη. Περαιτέρω, παρουσιάζονται οι βασικοί τρόποι επεξεργασίας των τριδιάστατων γραφικών παραστάσεων.



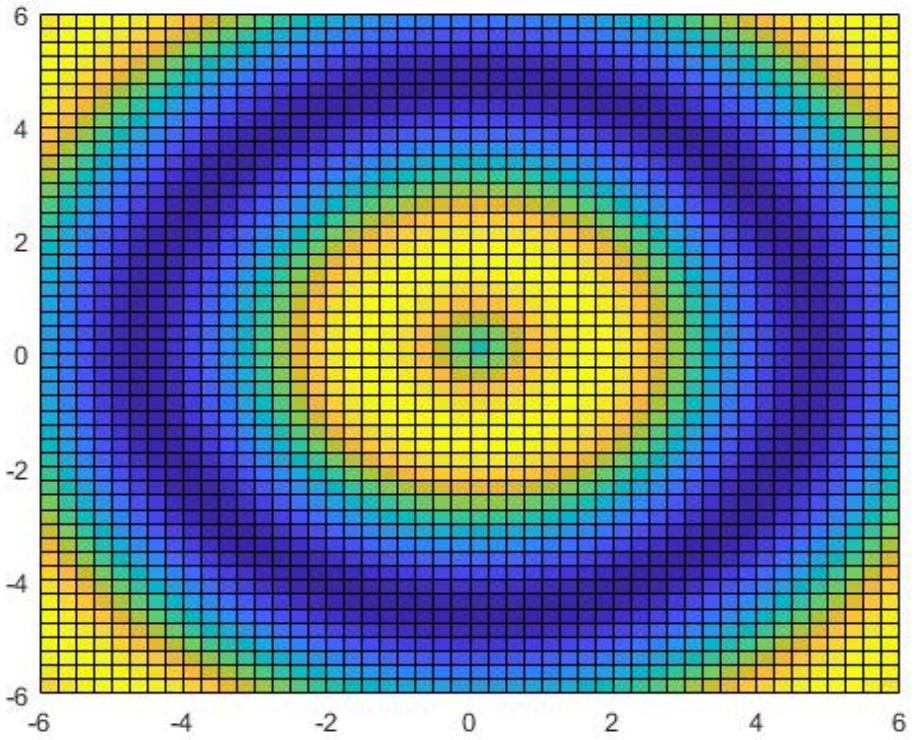
Σχήμα 6.4: Η γραφική παράσταση της συνάρτησης $f(x,y) = sin(\sqrt{x^2+y^2})$ για $x,y \in [-6,6]$ με την εντολή `surf` (αριστερό γράφημα) και με την εντολή `mesh` (δεξί γράφημα).



Σχήμα 6.5: Γράφημα ισοϋψών της συνάρτησης $f(x,y) = sin(\sqrt{x^2+y^2})$ για $x,y \in [-6,6]$.



Σχήμα 6.6: Η γραφική παράσταση της συνάρτησης $f(x,y) = sin(\sqrt{x^2 + y^2})$ για $x,y \in [-6,6]$ με την εντολή `surfc` (αριστερό γράφημα) και με την εντολή `meshc` (δεξί γράφημα).



Σχήμα 6.7: Γράφημα έντασης της συνάρτησης $f(x,y) = sin(\sqrt{x^2 + y^2})$ για $x,y \in [-6,6]$.

6.3.1 Εισαγωγή τίτλων και μορφοποίηση αξόνων

Η εισαγωγή τίτλων και η μορφοποίηση των αξόνων γίνεται με παρόμοιο τρόπο με τις διδιάστατες γραφικές παραστάσεις. Έτσι, παραδείγματος χάριν, οι παρακάτω εντολές

```
>> title('plot 1')
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
```

εισάγουν τον τίτλο plot 1 στο γράφημα και τα ονόματα x, y και z στους άξονες.

Τα όρια των αξόνων μπορούν να μεταβληθούν με τις εντολές `xlim`, `ylim` και `zlim`. Τα σημάδια στους άξονες μπορούν και αυτά να μεταβληθούν με τις ίδιες εντολές, όπως αυτές που παρουσιάστηκαν στις διδιάστατες γραφικές παραστάσεις.

Παράδειγμα 6.2

Η θερμοκρασία που αναπτύσσεται περιμετρικά από μια πηγή θερμότητας, τοποθετημένη στο σημείο (0,0), περιγράφεται από την εξίσωση

$$f_1(x,y) = Tmax - 4\sqrt{x^2+y^2}$$

για $x,y \in [-50,50]$ (μέτρα), όπου $Tmax$ η θερμοκρασία που υπάρχει στο σημείο που βρίσκεται η πηγή θερμότητας. Σημειώνεται ότι η θερμοκρασία σε οποιοδήποτε σημείο (προφανώς μακριά από την πηγή θερμότητας) δεν μπορεί να πέσει κάτω από τη θερμοκρασία περιβάλλοντος $Temp$. Επομένως, η πραγματική εξίσωση της θερμοκρασίας που αναπτύσσεται σε κάθε σημείο περιμετρικά από την πηγή θερμοκρασίας περιγράφεται από τη σχέση

$$f_2(x,y) = max\left(Temp, Tmax - 4\sqrt{x^2+y^2}\right).$$

Να γράψετε ένα *script* αρχείο για την κατασκευή της γραφικής παράστασης της μεταβολής της θερμοκρασίας περιμετρικά από την πηγή θερμότητας. Πιο συγκεκριμένα, να κατασκευάσετε δύο γραφικές παραστάσεις σε ένα παράθυρο, σχεδιάζοντας στο αριστερό γράφημα την $f_1(x,y)$ και στο δεξί γράφημα την $f_2(x,y)$. Στα γραφήματα θα πρέπει να προσθέσετε και κατάλληλους τίτλους. Το αρχείο θα πρέπει να ζητά από τον χρήστη τη θερμοκρασία που αναπτύσσεται από την πηγή θερμότητας, καθώς και τη θερμοκρασία περιβάλλοντος.

Λύση Παραδείγματος 6.2

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε μία μεταβλητή που να περιέχει την τιμή της θερμοκρασίας που αναπτύσσεται από την πηγή θερμότητας (Tmax) και μία μεταβλητή που θα περιέχει την τιμή της θερμοκρασίας περιβάλλοντος (Temp). Οι τιμές αυτών των δύο μεταβλητών θα πρέπει να εισάγονται από τον χρήστη. Για τον λόγο αυτό θα χρειαστούμε δύο input.

Κύριος κώδικας: Για το κύριο πρόγραμμα χρειαζόμαστε αρχικά να ορίσουμε τα διανύσματα των σημείων στους άξονες x και y με τη βοήθεια των οποίων θα ορίσουμε το πλέγμα (`meshgrid`). Στη συνέχεια, χρησιμοποιώντας το πλέγμα, θα ορίσουμε τις τιμές της συνάρτησης $Z1 = f_1(x,y)$ και θα σχεδιάσουμε τη γραφική της παράσταση με την εντολή `surf(X,Y,Z1)`. Σημειώνεται ότι, επειδή μας ζητείται

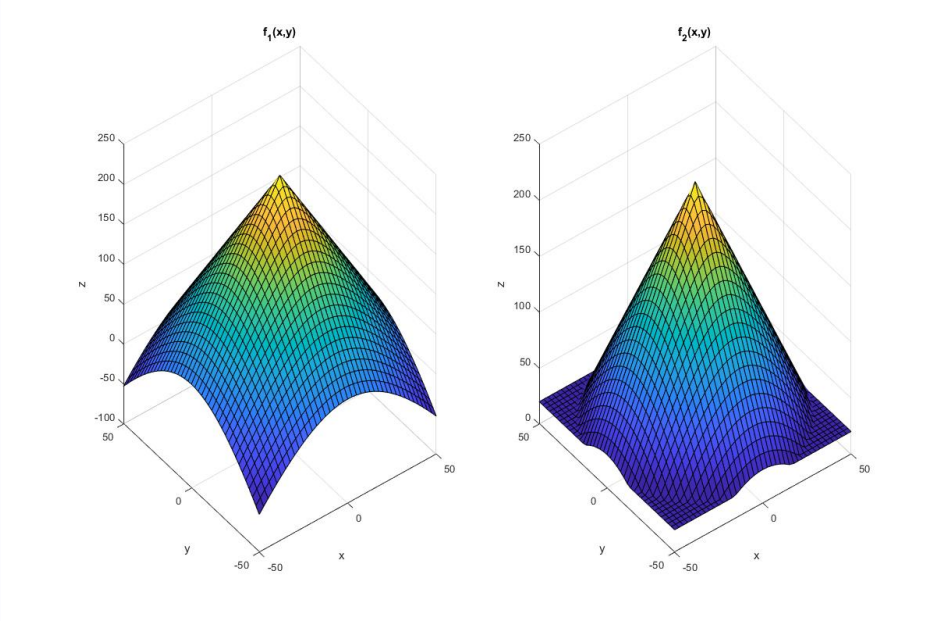
η κατασκευή της γραφικής παράστασης στο αριστερό γράφημα ενός παραθύρου με δύο γραφικές παραστάσεις, θα πρέπει να χρησιμοποιήσουμε και την εντολή `subplot(1,2,1)`. Τέλος, επειδή πρέπει να προσθέσουμε και κατάλληλους τίτλους θα πρέπει να πληκτρολογήσουμε και τις εντολές

```
xlabel('x')
ylabel('y')
zlabel('z')
title('f_1(x,y)')
```

Για την κατασκευή της γραφικής παράστασης της $f_2(x,y)$ ακολουθούμε παρόμοια διαδικασία. Σημαντικό βήμα για την επίτευξη αυτού του σκοπού είναι ο υπολογισμός των τιμών της $f_2(x,y)$. Οι τιμές της $f_2(x,y)$ μπορούν να υπολογιστούν αντικαθιστώντας τις τιμές του $Z1$, που είναι μικρότερες από το Temp, με την τιμή Temp.
Ο παρακάτω κώδικας υλοποιεί τη διαδικασία που περιγράφτηκε παραπάνω.

```
%HEATSOURCE plots f1(x,y)=Tmax-4\sqrt(x^2+y^2) and
% f2(x,y)=max(Temp,Tmax-4\sqrt(x^2+y^2)) for
% Tmax and Temp given by the user

Tmax=input('type the temperarature of the heat source:');
Temp=input('type ambient temperature:');

x=-50:2.5:50;
y=x;
[X,Y]=meshgrid(x,y);
Z1=Tmax-4*sqrt(X.^2+Y.^2);

subplot(1,2,1)
surf(X,Y,Z1)
xlabel('x')
ylabel('y')
zlabel('z')
title('f_1(x,y)')
Z2=Z1;
Z2(Z2<=Temp)=Temp;

subplot(1,2,2)
surf(X,Y,Z2)
xlabel('x')
ylabel('y')
zlabel('z')
title('f_2(x,y)')
```

Εκτελώντας το παραπάνω script αρχείο και πληκτρολογώντας τις τιμές 230 και 20 για θερμοκρασία που αναπτύσσεται από την πηγή θερμότητας και τη θερμοκρασία περιβάλλοντος, αντίστοιχα, λαμβάνουμε τις γραφικές παραστάσεις των $f_1(x,y)$ και $f_2(x,y)$ του Σχήματος 6.8.



Σχήμα 6.8: Οι γραφικές παραστάσεις των $f_1(x,y)$ (αριστερό γράφημα) και $f_2(x,y)$ (δεξί γράφημα), για θερμοκρασία που αναπτύσσεται από την πηγή θερμότητας και θερμοκρασία περιβάλλοντος, ίσες με 230 και 20, αντίστοιχα.

Άσκηση αυτοαξιολόγησης 6.2

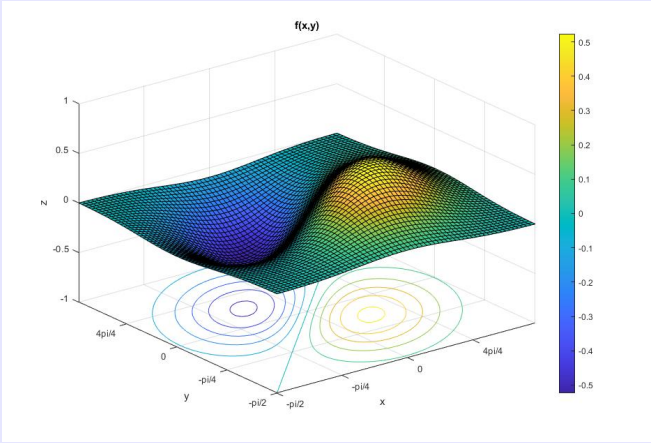
Ξαναγράψτε τον κώδικα της προηγούμενης άσκησης χρησιμοποιώντας επαναληπτικές διαδικασίες και ελέγχους για να ορίσετε τις τιμές της $f_2(x,y)$.

Άσκηση αυτοαξιολόγησης 6.3

Να γράψετε τον κώδικα για να κατασκευάσετε την παρακάτω γραφική παράσταση της συνάρτησης

$$f(x,y) = e^{-(x^2+y^2)} sin(x-y)$$

για $x,y \in [-\pi/2, \pi/2]$.



Σχήμα 6.9: Η γραφική παράσταση της $f(x,y) = e^{-(x^2+y^2)} sin(x-y)$ για $x,y \in [-\pi/2, \pi/2]$.

6.3.2 Αλλαγή χρωματισμού

Το Matlab επιτρέπει την αλλαγή της χρωματικής παλέτας που χρησιμοποιεί για την αποτύπωση των τριδιάστατων γραφικών παραστάσεων. Η εντολή που πρέπει να εκτελεστεί, για να αλλάξει το χρώμα απεικόνισης, είναι η `colormap`. Στο Σχήμα 6.10 αποτυπώνεται, στο ίδιο παράθυρο, η γραφική παράσταση της $f(x,y) = sin(x)cos(y)$ για $x,y \in [-\pi,\pi]$, σχεδιασμένη με δύο διαφορετικές χρωματικές παλέτες.

Για την κατασκευή των γραφικών παραστάσεων του Σχήματος 6.10 είναι απαραίτητη η χρήση δύο ακόμα εντολών. Η πρώτη εντολή είναι η εντολή `tiledlayout`, η οποία είναι παρόμοια με την `subplot` και την οποία έχουμε παρουσιάσει νωρίτερα. Η `tiledlayout` χωρίζει το παράθυρο σε $m \times n$ τμήματα, έτσι ώστε να μπορούμε να σχεδιάσουμε μια διαφορετική παράσταση σε καθένα από αυτά.

Η πρώτη διαφορά `tiledlayout` με την `subplot` είναι ότι η `tiledlayout` συντάσσεται ως `tiledlayout(m,n)`, ενώ η ενεργοποίηση της επόμενης θέσης γίνεται με την εντολή `nexttile`. Η δεύτερη είναι ότι επιτρέπει τη χρησιμοποίηση διαφορετικής χρωματικής παλέτας σε κάθε θέση, κάτι που με την `subplot` δεν είναι επιτρεπτό.

Στη συνέχεια, εμφανίζεται ο κώδικας για την κατασκευή των γραφικών παραστάσεων του Σχήματος 6.10.

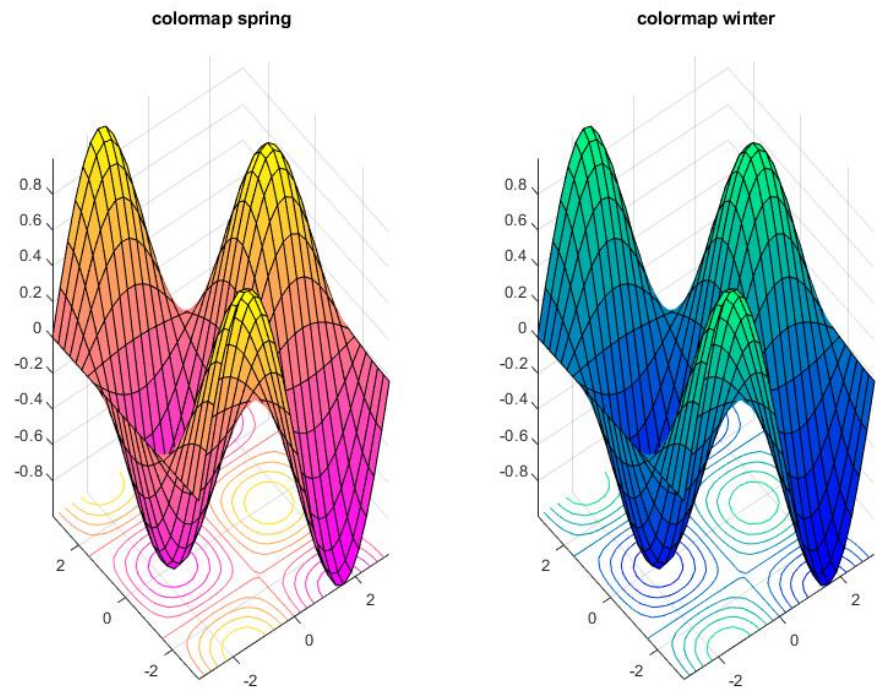
```
x=-pi:0.25:pi;
y=x;
[X,Y]=meshgrid(x,y);
Z=sin(X).*cos(Y);

tiledlayout(1,2)
ax1 = nexttile;
surfc(X,Y,Z)
colormap(ax1,spring)
title('colormap spring')
ax2 = nexttile;
surfc(X,Y,Z)
colormap(ax2,winter)
title('colormap winter')
```

Οι εντολές colormap(ax1,spring) και colormap(ax1,winter) είναι αυτές που είναι υπεύθυνες για τον διαφορετικό χρωματισμό των παραστάσεων. Για περισσότερα παραδείγματα και πληροφορίες για την εντολή colormap ο/η αναγνώστης/στρια παραπέμπεται στη σελίδα https://www.mathworks.com/help/matlab/ref/colormap.html.

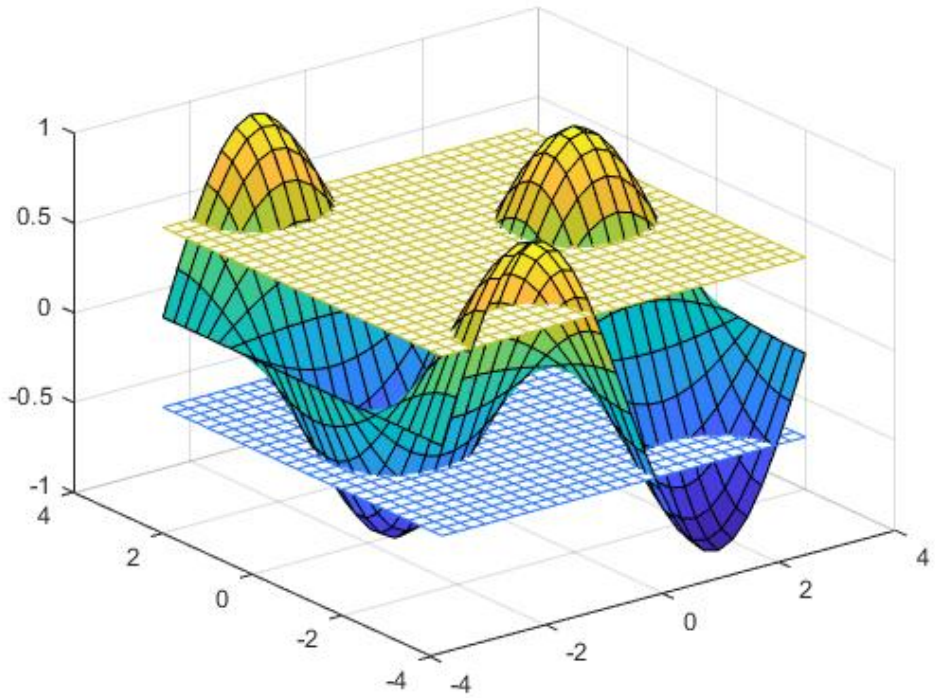
6.3.3 Εισαγωγή οριζόντιου επιπέδου αναφοράς

Συχνά, για την καλύτερη επόπτευση των τριδιάστατων γραφικών παραστάσεων είναι απαραίτητη η εισαγωγή ενός ή περισσότερων επιπέδων αναφοράς παράλληλων με το επίπεδο xy. Τέτοια επίπεδα μπορούν να προστεθούν εύκολα σε μια τριδιάστατη παράσταση ορίζοντας μια σταθερά συνάρτηση $g(x,y) = c$, όπου c μια σταθερά και σχεδιάζοντάς τη σταθερά συνάρτηση στο ίδιο γράφημα με τη συνάρτηση που έχουμε σχεδιάσει. Για να το πετύχουμε αυτό, η $g(x,y)$ πρέπει να οριστεί με τέτοιο τρόπο, ώστε να μπορούμε να την χρησιμοποιήσουμε στις εντολές κατασκευής τριδιάστατων γραφικών παραστάσεων, όπως, παραδείγματος χάριν, η εντολή `surf`.

Ακολούθως, εμφανίζεται ο κώδικας για την κατασκευή της γραφικής παράστασης του Σχήματος 6.11. Στη γραφική παράσταση εμφανίζεται η συνάρτηση $f(x,y) = sin(x)cos(y)$ για $x,y \in [-\pi,\pi]$ μαζί με τα οριζόντια



Σχήμα 6.10: Η γραφική παράσταση της $f(x,y) = sin(x)cos(y)$ για $x,y \in [-\pi,\pi]$, σχεδιασμένη με δύο διαφορετικές χρωματικές παλέτες.



Σχήμα 6.11: Η γραφική παράσταση της $f(x,y) = sin(x)cos(y)$ για $x,y \in [-\pi,\pi]$ μαζί με τα οριζόντια επίπεδα για $z = -0.5$ και $z = 0.5$.

επίπεδα για $z = -0.5$ και $z = 0.5$.

```
x=-pi:0.25:pi;
y=x;
[X,Y]=meshgrid(x,y);
Z=sin(X).*cos(Y);
surf(X,Y,Z)
hold on
Z1=0*X-0.5;
mesh(X,Y,Z1)
Z2=0*X+0.5;
mesh(X,Y,Z2)
hold off
```

Οι εντολές `Z1=0*X-0.5` και `Z2=0*X+0.5` εξασφαλίζουν ότι οι μεταβλητές $Z1$ και $Z2$ θα έχουν την ίδια διάσταση με την X και θα μπορούν, επομένως, να χρησιμοποιηθούν με την εντολή `surf`.

6.4 Ασκήσεις

Άσκηση 6.1. *Ποιες εντολές πρέπει να εισάγουμε στο* Matlab, *έτσι ώστε να πάρουμε τη γραφική παράσταση της*

$$f(x,y) = -\left(\frac{x}{2}\right)^2 - \left(\frac{y}{5}\right)^2 - 9$$

για $-3 \leq x,y \leq 3$*;*

Άσκηση 6.2. *Ποιες εντολές πρέπει να εισάγουμε στο* Matlab, *έτσι ώστε να πάρουμε τη γραφική παράσταση της*

$$f(x,y) = -\left(\frac{x}{2}\right)^2 - \left(\frac{y}{5}\right)^2 - 9$$

για $-3 \leq x,y \leq 3$*;*

Άσκηση 6.3. *Στο βιβλίο Ιωακειμίδης (2008) παρατίθεται το ακόλουθο παράδειγμα μετάδοσης θερμοκρασίας: Σε ένα μόνιμο (όχι μεταβαλλόμενο με τον χρόνο* t*) θερμοκρασιακό πρόβλημα σε περιοχή* D *του επιπέδου* xy*, η θερμοκρασία* $\theta = \theta(x,y)$ *διαπιστώθηκε ότι δίνεται από τη συνάρτηση*

$$\theta(x,y) = b(x^2 - y^2)$$

με το b *γνωστή σταθερά. Σημειώνεται ότι οι καμπύλες του επιπέδου* xy*, που ορίζονται από την εξίσωση*

$$b(x^2 - y^2) = \theta_0$$

αποτελούν τις ισόθερμες καμπύλες, δηλαδή τις ισοϋψείς καμπύλες.

Να κατασκευάσετε (σε ένα γράφημα 2×2*) τα διαγράμματα των ισόθερμων καμπυλών για* $b = 0.1, 0.5, 1, 2$*.*

Άσκηση 6.4. *Ο δείκτης ψυχρότητας αέρα (Wind Chill Factor - WCF) αποτελεί ένα μέτρο της αισθητής από τον ανθρώπινο οργανισμό θερμοκρασίας σε χαμηλές θερμοκρασίες. Ο δείκτης υπολογίζεται για θερμοκρασίες μικρότερες των* $7^\circ C$ *ως συνάρτηση της πραγματικής θερμοκρασίας* T *και της ταχύτητας ανέμου (μεγαλύτερη από 6.5* Km/h*) από τη σχέση:*

$$WCF = 13.12 + 0.6215 \cdot T - 11.37 \cdot v^{0.16} + 0.3965 \cdot T \cdot v^{0.16}.$$

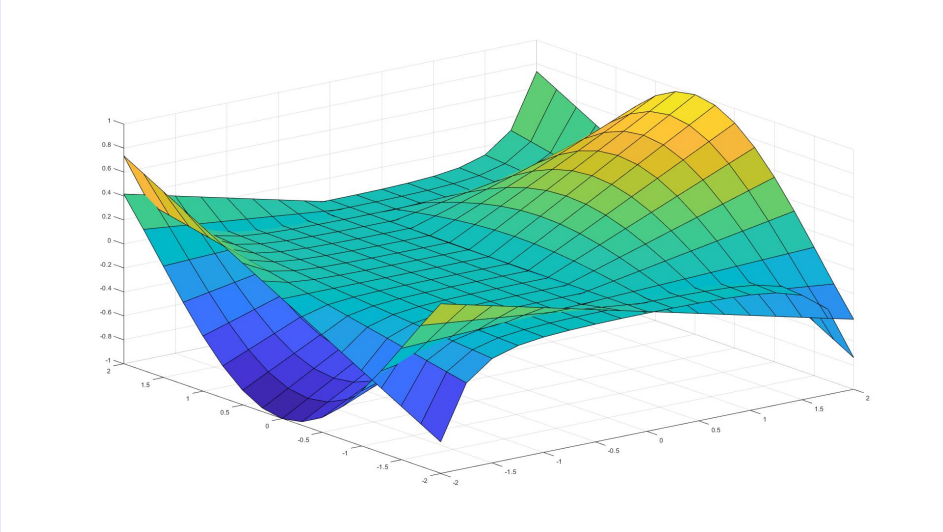
Δοκιμάστε διαφορετικούς τύπους διαγραμμάτων για να αποτυπώσετε γραφικά τον δείκτη WCF *για διάφορες τιμές της θερμοκρασίας* T *και της ταχύτητας,* v*, του ανέμου (Attaway, 2012).*

6.5 Απαντήσεις στις ασκήσεις αυτοαξιολόγησης

Λύση άσκησης αυτοαξιολόγησης 6.1

Η ζητούμενη γραφική παράσταση (Σχήμα 6.12) μπορεί να κατασκευαστεί με τις ακόλουθες εντολές

```
>> x=-2:0.25:2;
>> y=x;
>> [X,Y]=meshgrid(x,y);
>> Z1=Y.*exp(X.^2-5);
>> surf(X,Y,Z1)
>> hold on
>> Z2=1/2*X.*cos(Y);
>> surf(X,Y,Z2)
>> hold off
```



Σχήμα 6.12: Η γραφική παράσταση των συναρτήσεων $f_1(x,y) = y \cdot e^{x^2-5}$ και $f_2(x,y) = \frac{1}{2}x \cdot cos(y)$ για $x \in [-3,3]$ και $y \in [-3,3]$.

Η εντολή hold on επιτρέπει τον σχεδιασμό περισσότερων της μίας γραφικής παράστασης στο ίδιο παράθυρο, δεσμεύοντας το ενεργό παράθυρο της γραφικής παράστασης, όπως ακριβώς και στις διδιάστατες γραφικές παραστάσεις. Από την άλλη, η εντολή hold off αποδεσμεύει το παράθυρο και οποιαδήποτε καινούργια εντολή κατασκευής μίας γραφικής παράστασης θα αντικαταστήσει τις τρέχουσες.

Λύση άσκησης αυτοαξιολόγησης 6.2

Ο κώδικας θα είναι ίδιος, αλλά θα πρέπει να αντικαταστήσουμε την εντολή στη γραμμή 17, έτσι ώστε να υπολογίζουμε τις τιμές της μεταβλητής $Z2 = f(x,y)$ με τη βοήθεια επαναληπτικών διαδικασιών και ελέγχων. Για να το πετύχουμε αυτό, θα πρέπει:

- να διατρέξουμε πάνω από όλα τα στοιχεία του $Z2$ - υπενθυμίζεται ότι ο $Z2$ είναι ένας πίνακας, όποτε θα χρειαστούμε δύο «μετρητές», έναν για τις γραμμές και έναν για τις στήλες,
- να ελέγξουμε ένα ένα τα στοιχεία του $Z2$, αν είναι μικρότερα από το Temp. Αν ένα στοιχείο είναι

μικρότερο από το Temp, θα πρέπει να το αντικαταστήσουμε με αυτή την τιμή.

Χρήσιμη εντολή για την υλοποίηση του παραπάνω κώδικά είναι η εντολή size, η οποία επιστρέφει το πλήθος των γραμμών και το πλήθος των στηλών ενός πίνακα.
Τα παραπάνω μπορούν να υλοποιηθούν με τη βοήθεια του παρακάτω κώδικα.

```
1  Z2=Z1;
2  sizeZ2=size(Z2);
3  for n=1:sizeZ2(1)
4      for m=1:sizeZ2(2)
5          if Z2(n,m)<Temp
6              Z2(n,m)=Temp;
7          end
8      end
9  end
```

Λύση άσκησης αυτοαξιολόγησης 6.3

Για την κατασκευή της γραφικής παράστασης του Σχήματος 6.7 κατ΄ αρχάς παρατηρούμε ότι έχει χρησιμοποιηθεί η εντολή `surfc`. Οπότε, με βάση τις ακόλουθες εντολές, μπορούμε να πάρουμε μια πρώτη εκδοχή του γραφήματος.

```
>> x=-pi/2:0.05:pi/2;
>> y=x;
>> [X,Y]=meshgrid(x,y);
>> Z=exp(-(X.^2+Y.^2)).*sin(X-Y);
>> surfc(X,Y,Z)
```

Οι τίτλοι στο γράφημα και στους άξονες ορίζονται με τις ακόλουθες εντολές:

```
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
>> title('f(x,y)')
```

Τα όρια του κάθετου z άξονα, έτσι ώστε αυτά να ταυτίζονται με αυτά του Σχήματος 6.7, καθορίζονται με την εντολή:

```
>> zlim([-1,1])
```

Η μορφοποίηση των αξόνων x και y γίνεται με τις εντολές:

```
>> ax=gca;
>> ax.XTick = -pi/2:pi/4:pi/2;
>> ax.XTickLabel = {'-pi/2','-pi/4','0','4pi/4','pi/2','pi'};
>> ax.YTick = -pi/2:pi/4:pi/2;
>> ax.YTickLabel = {'-pi/2','-pi/4','0','4pi/4','pi/2','pi'};
```

Τέλος, η χρωματική μπάρα μπορεί να εμφανιστεί εκτελώντας την εντολή:

```
>> colorbar
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ελληνόγλωσση

Ιωακειμίδης, Ν. Ι. (2008). *Εφαρμοσμένα μαθηματικά ΙΙ για πολιτικούς μηχανικούς*. Πάτρα: Εκδόσεις Gotsis.

Ξενόγλωσση

Attaway, S. (2012). *Matlab: A Practical Introduction to Programming and Problem Solving*. Butterworth-Heinemann, MA, USA.

ΚΕΦΑΛΑΙΟ 7

ΠΟΛΥΩΝΥΜΑ ΚΑΙ ΜΙΓΑΔΙΚΟΙ ΑΡΙΘΜΟΙ

Σύνοψη:

Σε αυτό το κεφάλαιο παρουσιάζεται ο χειρισμός των πολυωνύμων και των μιγαδικών αριθμών από το Matlab. Ιδιαίτερη έμφαση δίνεται στις εντολές για την εκτέλεση πράξεων μεταξύ πολυωνύμων στον υπολογισμό των τιμών τους, την παραγώγιση και την ολοκλήρωσή τους, ενώ παρουσιάζεται και η διαδικασία εύρεσης των ριζών των πολυωνύμων. Στα πλαίσια εύρεσης των ριζών των πολυωνύμων γίνεται αναφορά στους μιγαδικούς αριθμούς και στον τρόπο χειρισμού τους από το Matlab. Τέλος, παρουσιάζεται η διαδικασία προσαρμογής πολυωνύμων για την περιγραφή της σχέσης μεταξύ δύο μεταβλητών.

Προαπαιτούμενη γνώση: Τα κεφάλαια 3, 4 και 5 του παρόντος συγγράμματος.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του κεφαλαίου αυτού, θα γνωρίζετε:

- για την τέλεση πράξεων μεταξύ πολυωνύμων,
- για την παραγώγιση και ολοκλήρωση πολυωνύμων,
- για την εύρεση των ριζών πολυωνύμων,
- για την τέλεση πράξεων μεταξύ μιγαδικών αριθμών,
- για την προσαρμογή πολυωνύμων για την περιγραφή της σχέσης δύο μεταβλητών.

Γλωσσάριο επιστημονικών όρων

- Θεμελιώδες θεώρημα άλγεβρας
- Μέτρο μιγαδικού
- Μιγαδικός αριθμός
- Όρισμα μιγαδικού
- Πολυώνυμο
- Πραγματικό και φανταστικό μέρος μιγαδικού
- Ρίζα πολυωνύμου
- Συζυγής μιγαδικός

7.1 Εισαγωγή

Τα πολυώνυμα είναι από τις πιο χρήσιμες συναρτήσεις στα μαθηματικά και τις φυσικές επιστήμες και τις συναντάμε συχνά σε πολλά επιστημονικά πεδία συμπεριλαμβανομένων:

- της μηχανικής,
- της χημείας,
- της φυσικής,
- των επιστημών υγείας,
- των οικονομικών επιστημών,
- τις κοινωνικές επιστήμες.

Η σπουδαιότητα των πολυωνύμων δεν προκύπτει μόνο από την άμεση χρήση τους για την περιγραφή διάφορων φαινομένων στα παραπάνω πεδία αλλά και από το γεγονός ότι για οποιαδήποτε συνεχή συνάρτηση σε ένα διάστημα υπάρχει ένα πολυώνυμο το οποίο προσεγγίζει ικανοποιητικά τη συνάρτηση αυτή. Το παραπάνω διατυπώνεται αυστηρά στο επόμενο θεώρημα.

Θεώρημα 7.1: Θεώρημα Weierstraß (Jeffreys *et al.*, 1999; Siegmund-Schultze, 2016)

Αν $f[\alpha,\beta] \to \mathbb{R}$ μια συνεχής συνάρτηση στο $[\alpha,\beta]$ και $\epsilon > 0$, τότε υπάρχει ένα πολυώνυμο $p(x)$ στο $[\alpha,\beta]$ τέτοιο, ώστε

$$|f(x) - p(x)| < \epsilon$$

για όλα τα $x \in [\alpha,\beta]$. Αυτό σημαίνει ότι οποιαδήποτε συνεχής συνάρτηση σε ένα κλειστό και φραγμένο διάστημα μπορεί να προσεγγιστεί (με οποιονδήποτε βαθμό ακρίβειας) στο διάστημα αυτό από ένα πολυώνυμο.

Από το παραπάνω θεώρημα γίνεται φανερό ότι τα πολυώνυμα αποτελούν σημαντικά εργαλεία για τη μελέτη απλών αλλά και σύνθετων προβλημάτων και φαινομένων. Το Matlab αναγνωρίζει το γεγονός αυτό, που αντικατοπτρίζει τη σπουδαιότητα των πολυωνύμων και για τον λόγο αυτό καθιστά, όπως θα δούμε στο κεφάλαιο αυτό, τη διαχείριση των πολυωνύμων μια εξαιρετικά εύκολη διαδικασία.

7.2 Δήλωση και βασικός χειρισμός πολυωνύμων

Η δήλωση ενός πολυωνύμου στο Matlab, όπως προαναφέρθηκε, είναι εξαιρετικά απλή και εύκολη. Στην πραγματικότητα το Matlab ερμηνεύει κάθε διάνυσμα σαν εν δυνάμει πολυώνυμο αναγνωρίζοντας το γεγονός ότι κάθε πολυώνυμο καθορίζεται από τους συντελεστές του. Για να γίνει αυτό πιο κατανοητό, αρκεί να παρατηρήσουμε ότι τα ακόλουθα δύο πολυώνυμα

$$\pi(x) = 2x^3 - x + 1$$
$$\kappa(y) = 2y^3 - y + 1$$

εκφράζουν το ίδιο πολυώνυμο $3^{ου}$ βαθμού με συντελεστές 2, 0, -1 και 1. Ούτε το όνομα (π ή κ) ούτε η μεταβλητή (x ή y) καθορίζουν τις τιμές και τη συμπεριφορά του πολυωνύμου. Η συμπεριφορά, επομένως, του πολυωνύμου καθορίζεται αποκλειστικά από τον βαθμό του (δηλαδή τον μεγιστοβάθμιο όρο) και τις τιμές των συντελεστών του. Το γεγονός αυτό επιτρέπει στο Matlab να ερμηνεύει, αν ζητηθεί, κάθε διάνυσμα (στο παραπάνω παράδειγμα το $[2, 0, -1, 1]$) ως ένα διάνυσμα που καταγράφει τους συντελεστές ενός πολυωνύμου.

Για παράδειγμα, αν θέλουμε να δηλώσουμε στο Matlab το πολυώνυμο

$$x^5 - 12x^3 + x^2 + 2x + 11$$

τότε εισάγουμε στο Matlab την εντολή

```
>> p1=[1,0,-12,1,2,11];
```

η οποία αποθηκεύει στο διάνυσμα $p1$ τους συντελεστές του παραπάνω πολυωνύμου. Αντίστοιχα, το διάνυσμα

```
>> p2=[-12,8,0,0,0];
```

αναπαριστά το πολυώνυμο $-12x^4 + 8x^3$.

Παρατήρηση 7.1

Παρατηρήστε ότι είναι απαραίτητη η δήλωση των συντελεστών όλων των όρων ενός πολυωνύμου, ακόμα και αν αυτοί είναι ίσοι με μηδέν. Παραδείγματος χάριν, το πολυώνυμο

$$x^5 - 12x^3 + x^2 + 2x + 11$$

δηλώθηκε στο Matlab με την εντολή

```
>> p1=[1,0,-12,1,2,11];
```

Αν το είχαμε δηλώσει ως

```
>> p1=[1,-12,1,2,11];
```

δηλαδή παραβλέποντας το γεγονός ότι ο συντελεστής του x^4 είναι μηδέν, τότε το Matlab θα ερμήνευε το $p1$ ως το πολυώνυμο

$$x^4 - 12x^3 + x^2 + 2x + 11$$

το οποίο είναι προφανές ότι δεν ταυτίζεται με το αρχικό.

Παράδειγμα 7.1

Να δηλώσετε στο Matlab τα πολυώνυμα:

1. $\pi_1(x) = x^5 - 1,$
2. $\pi_2(x) = x^5 - x^3 + x^2 + 10,$
3. $\pi_3(x) = x^{100} - x^{99} + \ldots + x^2 - x + 1.$

Λύση Παραδείγματος 7.1

Για την εισαγωγή των δύο πρώτων πολυωνύμων στο Matlab πληκτρολογούμε τις εντολές

```
>> p1=[1,0,0,0,0,1];
>> p2=[1,0,-1,1,0,1];
```

Για το τρίτο πολυώνυμο μπορούμε να βασιστούμε σε μια διαδικασία επανάληψης για να ορίσουμε το διάνυσμα των συντελεστών του. Πιο συγκεκριμένα, μπορούμε αρχικά να ορίσουμε ένα διάνυσμα μονάδων με 101 στοιχεία. Παρατηρήστε ότι είναι ένα πολυώνυμο $100^{ου}$ βαθμού, οπότε χρειαζόμαστε 100+1 (για την σταθερά) όρους και, στη συνέχεια, να αλλάζουμε το πρόσημο μόνο στις άρτιες θέσεις, δηλαδή στις θέσεις που αντιστοιχούν στους συντελεστές των όρων $x^{99}, x^{97}, \ldots, x^3, x$. Αυτό μπορεί να

γίνει με τις παρακάτω εντολές

```
>> p3=ones(1,101);
>> for n = 2:2:101
       p3(n)=-1;
   end
```

όπου η εντολή `ones(n,k)` κατασκευάζει έναν $n \times k$ πίνακα με όλα τα στοιχεία του ίσα με ένα.

Άσκηση αυτοαξιολόγησης 7.1

Να δηλώσετε στο Matlab το πολυώνυμο

$$k(x) = x^{200} + x^{198} + \ldots + x^4 + x^2 + 1.$$

7.2.1 Υπολογισμός των τιμών ενός πολυωνύμου

Για να βρούμε την τιμή ενός πολυωνύμου σε ένα σημείο, χρησιμοποιούμε την εντολή `polyval`. Πιο συγκεκριμένα, αν έχουμε αποθηκεύσει τους συντελεστές ενός πολυωνύμου $\pi(x)$ σε ένα διάνυσμα, έστω p, τότε η τιμή του πολυωνύμου στο σημείο $x = x_0$ μπορεί να υπολογιστεί με τη βοήθεια του Matlab με την ακόλουθη εντολή

```
>> polyval(p,x0)
```

Παράδειγμα 7.2

Υπολογίστε την τιμή του πολυωνύμου

$$\alpha(x) = 8x^4 - 2x + 10$$

για $x = 0.9$ και σχεδιάστε τη γραφική παράστασή του στο $[-1.5,1.5]$.

Λύση Παραδείγματος 7.2

Η δήλωση του πολυωνύμου στο Matlab γίνεται με τη βοήθεια της εντολής

```
>> a=[-8,0,0,-2,10];
```

η οποία εισάγει το διάνυσμα των συντελεστών του πολυωνύμου και το αποθηκεύει στη μεταβλητή a.
Η τιμή του πολυωνύμου στο σημείο $x_0 = 0.9$ λαμβάνεται με την εντολή

```
>> y=polyval(a,0.9)
```

η οποία επιστρέφει το ακόλουθο αποτέλεσμα

```
y =
    2.9512
```

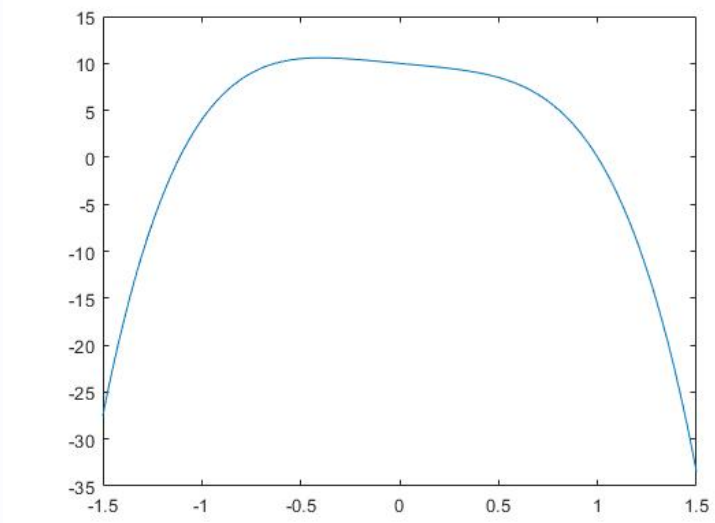
στο οποίο δηλώνεται ότι

$$\alpha(0.9) = 2.9512.$$

Πράγματι, αν θέσουμε $x = 0.9$ στη σχέση $8x^4 - 2x + 10$, τότε λαμβάνουμε ως τιμή το 2.9512.
Για τον σχεδιασμό της γραφικής παράστασης του πολυωνύμου πάνω από το διάστημα $[-1.5,1.5]$ είναι απαραίτητο να υπολογίσουμε την τιμή του πολυωνύμου για διάφορες τιμές του x και όχι μόνο για ένα μεμονωμένο x. Αυτό επιτυγχάνεται με τις ακόλουθες εντολές

```
>> x=-1.5:0.1:1.5;
>> y=polyval(a,x);
>> plot(x,y)
```

Η πρώτη εντολή ορίζει ένα σύνολο τιμών για το x από το -1.5 ως το 1.5 με βήμα 0.01, η δεύτερη υπολογίζει την τιμή του πολυωνύμου $\alpha(x)$ πάνω από αυτά τα x και αποθηκεύει τις τιμές του στο διάνυσμα y. Η τελευταία εντολή παράγει την παρακάτω γραφική παράσταση που αποτυπώνει τις τιμές του πολυωνύμου $\alpha(x)$ στο $[-1.5,1.5]$.



7.2.2 Πρόσθεση και αφαίρεση πολυωνύμων

Το Matlab δεν έχει κάποια έτοιμη, ειδική συνάρτηση, για την πρόσθεση και την αφαίρεση πολυωνύμων, παρ΄ όλα αυτά αυτές οι πράξεις μπορούν να γίνουν εύκολα προσθέτοντας ή αφαιρώντας τα διανύσματα των συντελεστών τους. Πράγματι, αν

$$\pi_1(x) = \alpha_n x^n + \alpha_{n-1}x^{n-1} + \ldots + \alpha_1 x + \alpha_0$$

και

$$\pi_2(x) = \beta_n x^n + \beta_{n-1}x^{n-1} + \ldots + \beta_1 x + \beta_0$$

δύο πολυώνυμα, τότε το άθροισμά τους ισούται με

$$(\alpha_n + \beta_n)x^n + (\alpha_{n-1} + \beta_{n-1})x^{n-1} + \ldots + (\alpha_1 + \beta_1))x + (\alpha_0 + \beta_0)$$

και η διαφορά τους με

$$(\alpha_n - \beta_n)x^n + (\alpha_{n-1} - \beta_{n-1})x^{n-1} + \ldots + (\alpha_1 - \beta_1))x + (\alpha_0 - \beta_0).$$

Αυτό σημαίνει ότι, αν στο Matlab προστεθούν (ή αφαιρεθούν) τα διανύσματα των συντελεστών των δύο πολυωνύμων, τότε λαμβάνουμε το διάνυσμα των συντελεστών του αθροίσματος (ή της διαφοράς) των δύο πολυωνύμων.

Παράδειγμα 7.3

Να βρεθεί το άθροισμα και η διαφορά των πολυωνύμων $\pi_1(x) = 3x^3 + x^2 - 2x + 5$ και $\pi_2(x) = 2x^3 + x^2 - x - 3$.

Λύση Παραδείγματος 7.3

Αρχικά, δηλώνουμε στο Matlab τα δύο πολυώνυμα εισάγοντας τα διανύσματα των συντελεστών των πολυωνύμων με τις ακόλουθες εντολές:

```
>> p1=[3,1,-2,5];
>> p2=[2,1,-1,-3];
```

Το άθροισμα και η διαφορά των δύο πολυωνύμων προκύπτουν αθροίζοντας και αφαιρώντας αντίστοιχα τα δύο διανύσματα $p1$ και $p2$ με τις ακόλουθες εντολές:

```
>> p1+p2
ans =
     5     2    -3     2
>> p1-p2
ans =
     1     0    -1     8
```

Επομένως, το άθροισμα των δύο πολυωνύμων ισούται με

$$5x^3 + 2x^2 - 3x + 2$$

ενώ η διαφορά με

$$x^3 - x + 8.$$

Αξίζει να σημειωθεί ότι, αν τα πολυώνυμα δεν είναι του ίδιου βαθμού, τότε, για να μπορέσουμε να τα προσθέσουμε ή να τα αφαιρέσουμε, θα πρέπει να επεκτείνουμε το διάνυσμα του πολυωνύμου μικρότερης τάξης με μηδενικά στην αρχή του, έτσι ώστε να αποκτήσει την ίδια διάσταση με το διάνυσμα του πολυωνύμου μεγαλύτερης τάξης. Αυτό φαίνεται πιο καθαρά στο ακόλουθο παράδειγμα.

Παράδειγμα 7.4

Να βρεθεί το άθροισμα των πολυωνύμων $a(x) = -x^3 + x$ και $b(x) = x^6 + x^5 - 2x^3 + 5$.

Λύση Παραδείγματος 7.4

Αρχικά, παρατηρούμε ότι το πρώτο πολυώνυμο είναι $3^{ου}$ βαθμού, ενώ το δεύτερο $6^{ου}$. Αυτό σημαίνει ότι στο Matlab το πρώτο πολυώνυμο αναπαρίσταται με ένα διάνυσμα 4 στοιχείων, ενώ το δεύτερο με ένα διάνυσμα 7 στοιχείων. Συνέπεια αυτού είναι να μην ορίζεται το άθροισμα των δύο αυτών διανυσμάτων. Η πρόσθεση, όμως, των δύο πολυωνύμων ορίζεται και πρέπει να μπορούμε να την υπολογίσουμε και στο Matlab.

Για να το πετύχουμε αυτό, εκφράζουμε το πολυώνυμο μικρότερης τάξης, εδώ το $a(x)$, ως ακολούθως

$$a(x) = 0x^6 + 0x^5 + 0x^4 - x^3 + x,$$

δηλαδή προσθέτοντας όρους με μηδενικούς συντελεστές, έτσι ώστε να μπορεί να ερμηνευτεί από το Matlab ως ομόβαθμο με το $b(x)$ και να μπορεί να εκτελεστεί η πρόσθεση. Επομένως, τα δύο πολυώνυμα μπορούν να δηλωθούν στο Matlab εκτελώντας τις εντολές

```
>>a=[0,0,0,-1,0,1,0];
```

```
>>b=[1,1,0,-2,0,0,5];
```

όπου στην πρώτη έχουμε προσθέσει στο διάνυσμα για το $a(x)$ 3 μηδενικά, όσο και η διαφορά των βαθμών των πολυωνύμων. Με αυτόν τον τρόπο επιτρέπουμε στο Matlab να είναι σε θέση να υπολογίσει το άθροισμα με την παρακάτω εντολή

```
>>c=a+b
```

και να επιστρέψει το αποτέλεσμα

```
c =
     1     1     0    -3     0     1     5
```

το οποίο δηλώνει ότι το άθροισμα των δύο πολυωνύμων ισούται με

$$x^6 + x^5 + 0x^4 - 3x^3 + 0x^2 + x + 5 = x^6 + x^5 - 3x^3 + x + 5.$$

Άσκηση αυτοαξιολόγησης 7.2

Να γράψετε ένα αρχείο script το οποίο θα ζητάει από τον χρήστη τα διανύσματα των συντελεστών δύο πολυωνύμων. Στη συνέχεια, αν χρειάζεται, το πρόγραμμα πρέπει να επεκτείνει μόνο του το διάνυσμα του πολυωνύμου μικρότερης τάξης με μηδενικά στην αρχή του, έτσι ώστε να αποκτήσει την ίδια διάσταση με το διάνυσμα του πολυωνύμου μεγαλύτερης τάξης και να επιστρέψει στον χρήστη το διάνυσμα των συντελεστών του αθροίσματος των δύο πολυωνύμων.

7.2.3 Πολλαπλασιασμός και διαίρεση πολυωνύμων

Για τον πολλαπλασιασμό και τη διαίρεση δύο πολυωνύμων, το Matlab έχει ειδικές συναρτήσεις που βασίζονται στα διανύσματα των συντελεστών των πολυωνύμων. Πιο συγκεκριμένα, η εντολή

```
>> conv(a,b)
```

επιστρέφει το διάνυσμα των συντελεστών του γινομένου δύο πολυωνύμων τα οποία έχουν δηλωθεί με τα διανύσματα a και b. Από την άλλη, η διαίρεση[1] δύο πολυωνύμων στο Matlab γίνεται με την εντολή `deconv`, η οποία επιστρέφει το πηλίκο και το υπόλοιπο και για αυτό πρέπει να καλείται ως εξής

```
>> [p,u]=deconv(a,b)
```

όπου και πάλι τα a και b είναι τα διανύσματα των συντελεστών των δύο πολυωνύμων.

Παράδειγμα 7.5

Να υπολογιστεί το γινόμενο των πολυωνύμων

$$a(x) = x^5 - x^3 + x \text{ και } b(x) = x^6 + x^5 - 11.$$

[1] Με τη διαίρεση δύο πολυωνύμων ($\Delta(x) \rightarrow$ διαιρετέος και $\delta(x) \rightarrow$ διαιρέτης) εννοούμε την εύρεση δύο άλλων πολυωνύμων, τέτοιων ώστε $\Delta(x) = \delta(x) \cdot \pi(x) + \upsilon(x)$, όπου το $\pi(x)$ είναι το πηλίκο και $\upsilon(x)$ το υπόλοιπο της διαίρεσης. Σημειώνεται ότι το υπόλοιπο της διαίρεσης είναι μηδέν ή έχει πάντα βαθμό μικρότερο από τον βαθμό του $\delta(x)$.

Λύση Παραδείγματος 7.5

Το γινόμενο των δύο πολυωνύμων λαμβάνεται με την εκτέλεση των παρακάτω εντολών.

```
>> a=[1,0,-1,0,1,0];
>> b=[1,1,0,0,0,0,-11]
>> conv(a,b)
ans =
     1     1    -1    -1     1     1   -11     0    11     0   -11     0
```

Με τις δύο πρώτες εντολές δηλώνονται τα δύο πολυώνυμα, ενώ με την τρίτη υπολογίζεται το γινόμενό τους, το οποίο ισούται με

$$x^{11} + x^{10} - x^9 - x^8 + x^7 + x^6 - 11x^5 + 11x^3 - x.$$

Παράδειγμα 7.6

Να γίνει η γραφική παράσταση (στο ίδιο γράφημα) του πηλίκου και του υπολοίπου της διαίρεσης των πολυωνύμων $a(x) = 2x^6 + 4x^5 + 6x^4 + 8x^3 + 6x^2 + 4x + 2$ και $b(x) = x^3 + 2x^2 + 3x + 4$ για $-2 \leq x \leq 2$.

Λύση Παραδείγματος 7.6

Για την κατασκευή της γραφικής παράστασης (στο ίδιο γράφημα) του πηλίκου και του υπολοίπου της διαίρεσης των πολυωνύμων, αρχικά, πρέπει να δηλώσουμε στο Matlab τα πολυώνυμα με τις εντολές

```
>> a=[2,4,6,8,6,4,2];
>> b=[1,2,3,4];
```

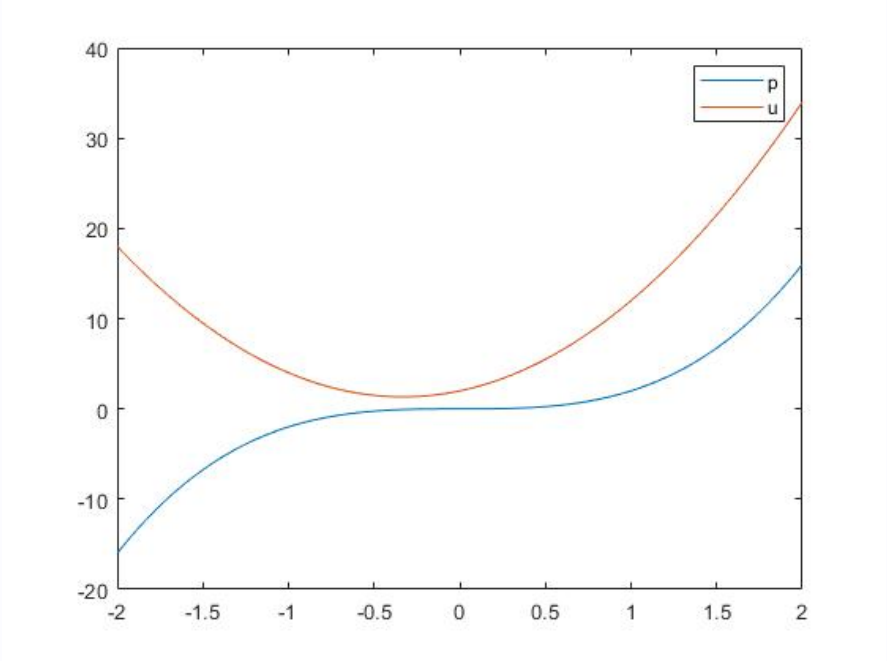
Στη συνέχεια, με την εντολή

```
>> [p,u]=deconv(a,b)
p =
     2     0     0     0
u =
     0     0     0     0     6     4     2
```

λαμβάνουμε το πηλίκο, $p(x)$, και το υπόλοιπο, $u(x)$, της διαίρεσης των δύο πολυωνύμων, τα οποία ισούνται με $p(x) = x^3$ και $u(x) = 6x^2 + 4x + 2$, αντίστοιχα.
Τέλος, η γραφική παράσταση των δύο παραπάνω πολυωνύμων για $-2 \leq x \leq 2$ μπορεί να γίνει με τις εντολές

```
>> x=-2:0.01:2;
>> plot(x,polyval(p,x))
>> hold ON
>> plot(x,polyval(u,x))
>> hold OFF
>> legend('p','u')
```

οι οποίες κατασκευάζουν το ακόλουθο γράφημα.



Παρατήρηση 7.2

Σημειώνεται ότι αν καλέσουμε τη συνάρτηση `deconv` χωρίς να κάνουμε ανάθεση των αποτελεσμάτων της σε δύο τιμές, τότε αυτή επιστρέφει μόνο το πηλίκο.

7.3 Παραγώγιση και ολοκλήρωση πολυωνύμων

Η παραγώγιση ενός πολυωνύμου και ο υπολογισμός του αόριστου ολοκληρώματός του γίνεται στο Matlab με τις εντολές `polyder` και `polyint`, αντίστοιχα. Πιο συγκεκριμένα, οι εντολές

```
>> polyder(a)
>> polyint(a)
```

επιστρέφουν τα διανύσματα των πολυωνύμων που προκύπτουν από την παραγώγιση και την ολοκλήρωση ενός πολυωνύμου του οποίου οι συντελεστές έχουν αποθηκευτεί στο διάνυσμα a. Σημειώνεται ότι η εντολή για τον υπολογισμό του αόριστου ολοκληρώματος μπορεί να συνταχθεί και ως ακολούθως

```
>> polyint(a,k)
```

Το k που εμφανίζεται μέσα στην εντολή καθορίζει τη σταθερά του αποτελέσματος. Αν αυτή παραλείπεται, τότε το Matlab θέτει $k = 0$.

Παράδειγμα 7.7

Να υπολογιστεί η παράγωγος και το ολοκλήρωμα του πολυωνύμου

$$a(x) = -5x^5 - x^3 + x + 1.$$

Λύση Παραδείγματος 7.7

Πληκτρολογώντας τις ακόλουθες εντολές στο Matlab δηλώνουμε το πολυώνυμο $a(x)$ και υπολογίζουμε την παράγωγο και το ολοκλήρωμά του.

```
>> a=[-5,0,-1,0,1,1];
>> da=polyder(a)
da =
   -25     0    -3     0     1
>> inta=polyint(a)
inta =
   -0.8333         0   -0.2500         0    0.5000    1.0000         0
```

Από τα αποτελέσματα παρατηρούμε ότι το Matlab έχει πράγματι υπολογίσει την παράγωγο και το ολοκλήρωμα του πολυωνύμου, αφού η παράγωγός του $a(x)$ ισούται με

$$a'(x) = -25x^4 - 3x^3 + 1$$

και το ολοκλήρωμά του με

$$\int a(x)dx = -\frac{5}{6}x^6 - \frac{1}{4}x^4 + \frac{1}{2}x^2 + x + c,$$

όπου c μια σταθερά. Παρατηρήστε ότι το Matlab θέτει τη σταθερά c ίση με το μηδέν, αφού δεν δηλώθηκε κάποια άλλη τιμή κατά την εκτέλεση της εντολής.

7.4 Εύρεση ριζών

Ένα από τα σημαντικότερα θεωρήματα στα μαθηματικά είναι το αποκαλούμενο **θεμελιώδες θεώρημα της άλγεβρας**, το οποίο ασχολείται με τις ρίζες ενός πολυωνύμου, δηλαδή την εύρεση των σημείων στα οποία το πολυώνυμο μηδενίζεται. Στη συνέχεια, παρουσιάζεται μια απλοποιημένη εκδοχή του θεωρήματος. Μια εκτενής μελέτη και παρουσίαση του θεμελιώδους θεωρήματος της άλγεβρας μπορεί να αναζητηθεί από τον/την αναγνώστη/στρια στο βιβλίο των Fine and Rosenberger (2012).

Θεώρημα 7.2

Κάθε πολυώνυμο n-οστού βαθμού

$$\pi(x) = \alpha_n x^n + \alpha_{n-1}x^{n-1} + \ldots + \alpha_1 x + \alpha_0, \alpha_n \neq 0$$

έχει, συμπεριλαμβανομένων των πολλαπλοτήτων, ακριβώς n ρίζες.

Με τον όρο πολλαπλότητα μιας ρίζας, r_i, αναφερόμαστε στον μεγαλύτερο φυσικό αριθμό, k, που είναι τέτοιος ώστε το πολυώνυμο $(x - r_i)^k$ να διαιρεί το πολυώνυμο $\pi(x)$. Σημειώνεται ότι κάθε πολυώνυμο μπορεί να γραφτεί στη μορφή

$$\pi(x) = (x - r_1)^{k_1}(x - r_2)^{k_2} \cdots (x - r_m)^{k_m}$$

όπου $r_i, i = 1, 2, \ldots, m$ είναι οι m διακριτές ρίζες του πολυωνύμου και $k_i, i = 1, 2, \ldots, m$ οι αντίστοιχες πολλαπλότητές τους με $k_i \geq 1$.

Συνέπεια των παραπάνω παρατηρήσεων είναι η ισοδύναμη έκφραση του παραπάνω θεμελιώδους θεωρήματος της άλγεβρας, η οποία διατυπώνει ότι κάθε πολυώνυμο n-οστού βαθμού μπορεί να έχει το πολύ n διαφορετικές ρίζες, αφού η πολλαπλότητα κάθε ρίζας είναι υποχρεωτικά μεγαλύτερη ή ίση από το ένα.

Το 1826 ο Νορβηγός Μαθηματικός Niels Henrik Abel απέδειξε ότι η εύρεση γενικών, αλγεβρικών τύπων για τον προσδιορισμό των ριζών πολυωνύμου $5^{ου}$ ή μεγαλύτερου βαθμού είναι αδύνατη (Abel, 1826). Το γεγονός αυτό καθιστά τη χρήση των υπολογιστών για την εύρεση των ριζών ενός πολυωνύμου ακόμα πιο επιτακτική, αφού ο προσδιορισμός τους (ειδικά για μεγάλου βαθμού πολυώνυμα) απαιτεί τη χρήση είτε ιδιαίτερων τεχνικών και συναρτήσεων είτε αριθμητικών μεθόδων.

Το Matlab έχει ενσωματωμένες διαδικασίες προσδιορισμού των ριζών ενός πολυωνύμου. Οι ρίζες ενός πολυωνύμου μπορούν να ανακτηθούν με την εντολή

```
>> roots(p)
```

η οποία επιστρέφει (υπό τη μορφή ενός διανύσματος) όλες τις ρίζες ενός πολυωνύμου, του οποίου οι συντελεστές έχουν δηλωθεί στη μεταβλητή p.

Παράδειγμα 7.8

Να προσδιοριστούν οι ρίζες των πολυωνύμων

$$\pi_1(x) = -8x^3 + 16x^2 - 32x + 62$$

και

$$\pi_2(x) = x^7 - 7x^6 + 21x^5 - 37x^4 + 44x^3 - 38x^2 + 24x - 8.$$

Λύση Παραδείγματος 7.8

Για να προσδιορίσουμε τις ρίζες των πολυωνύμων θα πρέπει αρχικά να δηλώσουμε στο Matlab τα πολυώνυμα ορίζοντας τα διανύσματα των συντελεστών τους. Στη συνέχεια, με την εντολή `roots` μπορούμε να προσδιορίσουμε όλες τις ρίζες των πολυωνύμων αυτών. Αυτό μπορεί να γίνει με την εκτέλεση των παρακάτω εντολών.

```
>> p1=[-8,16,-32, 62];
>> roots(p1)
ans =
   1.9682 + 0.0000i
   0.0159 + 1.9843i
   0.0159 - 1.9843i
>> p2=[1,-7,21,-37,44,-38,24,-8];
>> roots(p2)
ans =
   2.0000 + 0.0000i
   2.0000 - 0.0000i
   0.0000 + 1.0000i
   0.0000 - 1.0000i
   1.0000 + 1.0000i
   1.0000 - 1.0000i
   1.0000 + 0.0000i
```

Παρατηρούμε ότι το πολυώνυμο

$$\pi_1(x) = -8x^3 + 16x^2 - 32x + 62$$

ως πολυώνυμο τρίτου βαθμού έχει ακριβώς τρεις ρίζες:

$$1.9682,\ 0.0159 + 1.9843i,\ 0.0159 - 1.9843i.$$

Παρατηρήστε ότι μόνο μία από αυτές είναι πραγματικός αριθμός, ενώ οι άλλοι δύο είναι μιγαδικοί.
Για το πολυώνυμο

$$\pi_2(x) = x^7 - 7x^6 + 21x^5 - 37x^4 + 44x^3 - 38x^2 + 24x - 8.$$

οι ρίζες είναι οι

$$2(\text{διπλή}),\ i, -i,\ 1+i,\ 1-i,\ 1$$

Παρατηρούμε πάλι ότι το πλήθος των ριζών, λαμβάνοντας υπόψη την πολλαπλότητα των ριζών, ισούται με 7, όσο δηλαδή και ο βαθμός του πολυωνύμου.

Παρατήρηση 7.3

Όπως είδαμε στο προηγούμενο παράδειγμα, η εύρεση των ριζών ενός πολυωνύμου εμπλέκει ουσιαστικά τους μιγαδικούς αριθμούς. Για τον λόγο αυτό, στην επόμενη ενότητα, γίνεται μια σύντομη παρουσίαση του συνόλου των μιγαδικών αριθμών και τον τρόπο που χειρίζεται τους αριθμούς αυτούς το Matlab.

Άσκηση αυτοαξιολόγησης 7.3

Να προσδιορίσετε τις ρίζες του πολυωνύμου

$$x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1.$$

7.4.1 Ανάκτηση του πολυωνύμου από τις ρίζες του

Αν γνωρίζουμε τις ρίζες ενός πολυωνύμου, μπορούμε να ανακτήσουμε το πολυώνυμο που έχει αυτές τις ρίζες με την εντολή `poly`, όπως φαίνεται και στο ακόλουθο παράδειγμα.

Παράδειγμα 7.9

Να βρεθεί πολυώνυμο, το οποίο έχει τις ακόλουθες ρίζες

$$1.9682, 0.0159 + 1.9843i, 0.0159 - 1.9843i.$$

Λύση Παραδείγματος 7.9

Ο προσδιορισμός του πολυωνύμου που έχει ρίζες τους αριθμούς

$$1.9682, 0.0159 + 1.9843i, 0.0159 - 1.9843i$$

γίνεται στο Matlab πληκτρολογώντας τις ακόλουθες εντολές

```
>> r=[1.9682, 0.0159 + 1.9843i, 0.0159 - 1.9843i];
>> p=poly(r)
```

Η τελευταία εντολή επιστρέφει το ακόλουθο αποτέλεσμα

```
p =
    1.0000   -2.0000    4.0003   -7.7502
```

που εκφράζει τους συντελεστές του ζητούμενου πολυωνύμου, δηλαδή του

$$x^3 - 2x^2 + 4.0003x - 7.7502.$$

Παρατήρηση 7.4

Παρατηρήστε ότι το πολυώνυμο που λάβαμε με την εντολή `poly(r)` στο προηγούμενο παράδειγμα δεν ταυτίζεται με το πολυώνυμο που είχαμε στο Παράδειγμα 7.8, παρόλο που έχουν τις ίδιες ρίζες. Στην πραγματικότητα, οι συγκεκριμένοι αριθμοί δεν είναι ρίζες μόνο για αυτά τα δύο πολυώνυμα αλλά για μια ολόκληρη οικογένεια πολυώνυμων, η οποία ορίζεται από τη σχέση

$$c(x^3 - 2x^2 + 4.0003x - 7.7502)$$

όπου $c \in \mathbb{R} \setminus \{0\}$. Τα δύο προαναφερθέντα πολυώνυμα, δηλαδή το

$$\pi_1(x) = -8x^3 + 16x^2 - 32x + 62$$

και

$$x^3 - 2x^2 + 4.0003x - 7.7502.$$

είναι στην ουσία πολλαπλάσιο το ένα του άλλου. Οι διαφορές που παρατηρούνται μεταξύ των συντελεστών των δύο πολυωνύμων, όταν το δεύτερο πολλαπλασιαστεί με το -8, οφείλεται στο γεγονός ότι οι αριθμοί

$$1.9682,\ 0.0159 + 1.9843i,\ 0.0159 - 1.9843i$$

δεν είναι ακριβώς οι ρίζες του πολυωνύμου

$$\pi_1(x) = -8x^3 + 16x^2 - 32x + 62$$

άλλα η στρογγυλοποίηση στο τέταρτο δεκαδικό ψηφίο.

Παρατήρηση 7.5

Το Matlab με την εντολή `poly(r)` επιστρέφει από το άπειρο πλήθος των πολυωνύμων, που έχουν ρίζες τους αριθμούς του διανύσματος r, το πολυώνυμο εκείνο για το οποίο ο συντελεστής του μεγιστοβάθμιου όρου ισούται με ένα.

Άσκηση αυτοαξιολόγησης 7.4

Να προσδιορίσετε ένα πολυώνυμο που έχει ρίζες τους πραγματικούς αριθμούς $1, 2, 3$ και 4.

7.5 Μιγαδικοί αριθμοί

Η δημιουργία των μιγαδικών αριθμών οφείλεται στην προσπάθεια επίλυσης απλών εξισώσεων, όπως παραδείγματος χάριν, της εξίσωσης

$$x^2 + 1 = 0.$$

Η παραπάνω εξίσωση μπορεί να εκφραστεί ως $x^2 = -1$, η οποία προφανώς και δεν έχει κάποια λύση στο σύνολο των πραγματικών αριθμών. Στην προσπάθεια επίλυσης της παραπάνω εξίσωσης επινοήθηκε μια επέκταση των πραγματικών αριθμών με την προσθήκη ενός αριθμού, ο οποίος ονομάστηκε φανταστική μονάδα και συμβολίστηκε με i, που το τετράγωνό του ισούται με -1, δηλαδή ενός αριθμού με την ιδιότητα

$$i^2 = -1.$$

Με την προσθήκη αυτής της φανταστικής μονάδας το σύνολο των πραγματικών αριθμών μπορεί να επεκταθεί στο αποκαλούμενο σύνολο $\mathbb{C}$ των μιγαδικών αριθμών, το οποίο αποτελείται από αριθμούς της μορφής

$$\alpha + \beta i$$

όπου $\alpha, \beta \in \mathbb{R}$. Για παράδειγμα, ο μιγαδικός αριθμός $3 + 2i$ είναι ένας μιγαδικός με πραγματικό μέρος 3 και φανταστικό μέρος 2.

Οι μιγαδικοί αριθμοί στο Matlab, όπως είδαμε και στα Παραδείγματα της προηγούμενης ενότητας, δηλώνονται και ορίζονται με τη βοήθεια της φανταστικής μονάδας, που στο Matlab συμβολίζεται με το i. Παραδείγματος χάριν, με τις ακόλουθες εντολές

```
>> z1=3+i
>> z2=-1-3i
>> z3=-8i
```

ορίζουμε τρεις μεταβλητές $z1, z2$ και $z3$, στις οποίες αναθέτουμε τους μιγαδικούς αριθμούς $3 + i$, $-1 - 3i$ και $-8i$, αντίστοιχα.

Παρατήρηση 7.6

Για το σύμβολο της φανταστικής μονάδας το Matlab επιτρέπει τη χρήση και του συμβόλου j. Έτσι ο μιγαδικός αριθμός $z4 = 2 - i$ μπορεί να οριστεί και με τον ακόλουθο τρόπο.

```
>> z4=2-j
```

7.5.1 Πράξεις με μιγαδικούς αριθμούς

Όλες οι πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμός,…) μεταξύ μιγαδικών αριθμών στο Matlab εκτελούνται κανονικά με τα συνήθη σύμβολα. Παραδείγματος χάριν, χρησιμοποιώντας τις μεταβλητές $z1$, $z2$, $z3$ και $z4$, που ορίστηκαν στην αρχή της ενότητας, μπορούμε να εκτελέσουμε τις ακόλουθες εντολές:

```
>> z1+z2
ans =
   2.0000 - 2.0000i
>> z1-z3
ans =
   3.0000 + 9.0000i
>> z1/z4
ans =
   1.0000 + 1.0000i
>> z1*z4
ans =
   7.0000 - 1.0000i
>> z1^2
ans =
   8.0000 + 6.0000i
```

οι οποίες επιστρέφουν το άθροισμα, τη διαφορά, το πηλίκο, το γινόμενο και τη δύναμη των μιγαδικών αριθμών, αντίστοιχα.

7.5.2 Συζυγείς μιγαδικοί

Στη θεωρία των μιγαδικών αριθμών δύο αριθμοί ονομάζονται συζυγείς μιγαδικοί αριθμοί, όταν αυτοί έχουν ίσα πραγματικά και αντίθετα φανταστικά μέρη. Για να πάρουμε στο Matlab τον συζυγή ενός μιγαδικού, πρέπει

να ακολουθήσουμε έναν από τους δύο ακόλουθους τρόπους:

```
>> z2=-1-3i;

>> conj(z2)
ans =
  -1.0000 + 3.0000i

>> z2'
ans =
  -1.0000 + 3.0000i
```

οι οποίοι επιστρέφουν τον συζυγή αριθμό του μιγαδικού αριθμού $z2 = -1 - 3i$.

Παρατήρηση 7.7

Σημειώνεται ότι αν ένας μιγαδικός αριθμός είναι ρίζα ενός πολυωνύμου (με πραγματικούς συντελεστές), τότε ο συζυγής του είναι κι αυτός ρίζα του πολυωνύμου. Αυτό μπορούμε να το δούμε και στην επίλυση των παραδειγμάτων της προηγούμενης ενότητας.

7.5.3 Πραγματικό και φανταστικό μέρος μιγαδικού - Εντολές real και imag

Όπως προαναφέρθηκε, κάθε μιγαδικός αριθμός εκφράζεται ως

$$\alpha + \beta i$$

όπου $\alpha, \beta \in \mathbb{R}$. Το α είναι το αποκαλούμενο πραγματικό μέρος, ενώ το β το φανταστικό μέρος του μιγαδικού. Στο Matlab μπορούμε να απομονώσουμε το πραγματικό και το φανταστικό μέρος ενός μιγαδικού αριθμού χρησιμοποιώντας τις εντολές `real` και `imag`, όπως φαίνεται στο παρακάτω παράδειγμα.

```
>> >> z5=(-1-3i)'
z5 =
  -1.0000 + 3.0000i
>> re=real(z5)
re =
    -1
>> im=imag(z5)
im =
     3
```

Οι παραπάνω εντολές είναι ιδιαίτερα χρήσιμες στην περίπτωση που ζητάμε να εξετάσουμε αν ένας αριθμός είναι πραγματικός, δηλαδή έχει φανταστικό μέρος ίσο με το μηδέν, ή αν είναι φανταστικός, δηλαδή το πραγματικό μέρος ισούται με μηδέν. Παραδείγματος χάριν, με τις παρακάτω εντολές μπορούμε εύκολα να διαπιστώσουμε αν ένα πολυώνυμο, π.χ. το

$$\pi_1(x) = x^6 + x^5 - 3x^3 + x + 5$$

ή το

$$\pi_2(x) = x^6 - 3x^5 + 2.25x^4 - 0.25x^3 - 1.25x^2 + 2.75x - 2.5$$

έχει τουλάχιστον μία πραγματική ρίζα ή όχι.

```
>> p1=[1,1,0,-3,0,1,5];
>> any(imag(roots(p1))==0)
ans =
```

```
    0
>> p2=[1,-3,2.25,-0.25,-1.25,2.75,-2.5];
>> any(imag(roots(p2))==0)
ans =
    1
```

Από τα παραπάνω αποτελέσματα διαπιστώνουμε ότι το πολυώνυμο $p_1(x)$, δεν έχει καμιά πραγματική ρίζα ενώ το πολυώνυμο $p_2(x)$ έχει τουλάχιστον μία.

Παρατήρηση 7.8

Για τον έλεγχο αν ένας αριθμός είναι πραγματικός ή όχι, δηλαδή αν το φανταστικό του μέρος ισούται με μηδέν, το Matlab διαθέτει και την εντολή `isreal`, η οποία επιστρέφει την τιμή 1, αν είναι πραγματικός αριθμός, και, σε διαφορετική περίπτωση, την τιμή μηδέν, όπως φαίνεται στο παρακάτω παράδειγμα.

```
>> v=i^i;
>> isreal(v)
ans =
   1
```

Σημειώνεται ότι $i^i = 0.2079$.

7.5.4 Γραφική απεικόνιση μιγαδικού αριθμού

Με την εντολή `compass` μπορούμε να παραστήσουμε γραφικά με ένα διδιάστατο διάνυσμα έναν μιγαδικό αριθμό αντιστοιχώντας το πραγματικό μέρος του με τον οριζόντιο άξονα και το φανταστικό με τον κάθετο άξονα. Στην περίπτωση που χρησιμοποιηθεί ένα διάνυσμα ως όρισμα της εντολής `compass`, τότε στη γραφική παράσταση εμφανίζονται όλοι οι μιγαδικοί αριθμοί. Για παράδειγμα, το Σχήμα 7.1 έχει δημιουργηθεί με τις εντολές

```
>> p1=[1,1,0,-3,0,1,5];
>> compass(roots(p1))
```

και αναπαριστά γραφικά, υπό τη μορφή διανυσμάτων, όλες τις ρίζες του πολυωνύμου $\pi_1(x) = x^6 + x^5 - 3x^3 + x + 5$.

7.5.5 Τριγωνομετρική αναπαράσταση μιγαδικού - Μέτρο και όρισμα

Το Σχήμα 7.1 επιτρέπει την αναπαράσταση του μιγαδικού αριθμού

$$z = \alpha + \beta i$$

με τη λεγόμενη τριγωνομετρική αναπαράστασή του, δηλαδή την

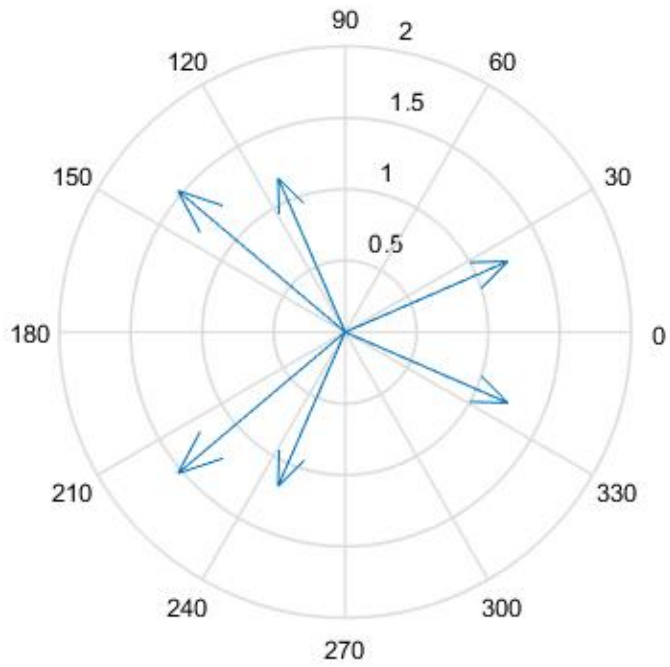
$$z = r(\cos\theta + \sin\theta),$$

όπου

- r είναι το μέτρο και
- θ το όρισμα

του μιγαδικού αριθμού z. Το μέτρο και το όρισμα του μιγαδικού αριθμού

$$z = \alpha + \beta i$$



Σχήμα 7.1: Γράφημα των ριζών του πολυωνύμου $\pi_1(x) = x^6 + x^5 - 3x^3 + x + 5$.

δίνονται από τις σχέσεις

$$|z| = \sqrt{\alpha^2 + \beta^2} \qquad Argz = tan^{-1}(\beta/\alpha)$$

και εκφράζουν το μήκος και τη γωνία που σχηματίζει με τον άξονα των πραγματικών αριθμών το διάνυσμα του μιγαδικού αριθμού.

Παρατήρηση 7.9

Οι ομόκεντροι κύκλοι αλλά και οι αριθμοί (μοίρες) που εμφανίζονται στο Σχήμα 7.1 στην ουσία μας βοηθάνε να προσδιορίσουμε (προσεγγιστικά) το μέτρο και το όρισμα κάθε μιγαδικού αριθμού. Παραδείγματος χάριν, η μοναδική ρίζα που βρίσκεται στο πρώτο τεταρτημόριο έχει μέτρο περίπου 0.75, ενώ το όρισμά της είναι λίγο μικρότερο από 30°.

Παρατήρηση 7.10

Από τον ορισμό του ορίσματος ενός μιγαδικού είναι φανερό ότι αυτό δεν είναι μοναδικό. Πράγματι, κάθε γωνιά της μορφής

$$Argz \pm 2k\pi, k \in \mathbb{Z}$$

μπορεί να θεωρηθεί ως όρισμα του z. Από όλα τα ορίσματα του z ακριβώς ένα βρίσκεται στο διάστημα $(-\pi, \pi]$. Το όρισμα αυτό ονομάζεται πρωτεύον όρισμα και είναι και αυτό που επιστρέφει το Matlab, όπως φαίνεται στη συνέχεια.

Στο Matlab το μέτρο και το (πρωτεύον) όρισμα ενός μιγαδικού αριθμού υπολογίζονται με τις εντολές `abs` και `angle` αντίστοιχα . Η εντολή `angle` επιστρέφει το όρισμα το μιγαδικού σε ακτίνια. Αν επιθυμούμε να εκφράσουμε τα ακτίνια σε μοίρες, τότε μπορούμε να χρησιμοποιήσουμε την εντολή `radtodeg`. Η χρήση αυτών των εντολών παρουσιάζεται στο παρακάτω παράδειγμα.

Παράδειγμα 7.10

Να υπολογιστεί το μέτρο και το όρισμα του μιγαδικού αριθμού $z = 2.5 - 1.2i$.

Λύση Παραδείγματος 7.10

Αρχικά, μπορούμε να δηλώσουμε τον μιγαδικό αριθμό $z = 2.5 - 1.2i$, πληκτρολογώντας την εντολή

```
>> z=2.5-1.2i;
```

Αφού έχουμε δημιουργήσει τη μεταβλητή z, μπορούμε να εκτελέσουμε την ακόλουθη εντολή

```
>> abs(z)
ans =
    2.7731
```

η οποία επιστρέφει το μέτρο του μιγαδικού, το οποίο στη συγκεκριμένη περίπτωση ισούται με 2.7731. Το (πρωτεύον) όρισμα του z προσδιορίζεται με την εντολή

```
>> angle(z)
ans =
   -0.4475
```

η οποία επιστρέφει τη γωνία (σε ακτίνια) που σχηματίζει το διάνυσμα του μιγαδικού με την ευθεία των πραγματικών. Η γωνία αυτή ανήκει στο διάστημα $(-\pi, \pi]$ και, επομένως, αποτελεί το πρωτεύον όρισμα του z. Η μετατροπή των ακτινίων σε μοίρες γίνεται με την εντολή

```
>> radtodeg(angle(z)) % Convert angles
                      % from radians to degrees
ans =
  -25.6410
```

η οποία μας πληροφορεί ότι το πρωτεύον όρισμα ισούται με -25.614°.

Άσκηση αυτοαξιολόγησης 7.5

Να προσδιοριστούν τα μέτρα και τα ορίσματα (σε μοίρες) όλων των ριζών του πολυωνύμου

$$\pi_1(x) = x^6 + x^5 - 3x^3 + x + 5.$$

7.6 Προσαρμογή πολυωνύμου

Στην αρχή του κεφαλαίου αναφέρθηκε ότι για οποιαδήποτε συνεχή συνάρτηση σε ένα διάστημα υπάρχει ένα πολυώνυμο το οποίο προσεγγίζει ικανοποιητικά τη συνάρτηση αυτή. Στην ενότητα αυτή θα δούμε πώς μπορούμε να χρησιμοποιήσουμε το Matlab, για να βρούμε το πολυώνυμο που προσεγγίζει ικανοποιητικά μια συνεχή συνάρτηση σε ένα συγκεκριμένο διάστημα. Χρήσιμη εντολή για την επίτευξη του σκοπού αυτού είναι η εντολή `polyfit`.

Η εντολή `polyfit`, στην πραγματικότητα, δεν έχει αναπτυχθεί ακριβώς για την προσέγγιση μιας συνεχούς συνάρτησης, αλλά για την προσαρμογή μιας πολυωνυμικής εξίσωσης ή, καλύτερα, ενός πολυωνυμικού μοντέλου στις παρατηρήσεις ενός φαινομένου. Για να γίνει αυτό πιο κατανοητό, ας υποθέσουμε ότι έχουμε στη διάθεσή μας τις τιμές που εμφανίζονται στον Πίνακα 7.1 και αποτυπώνονται στο Σχήμα 7.2. Οι τιμές αυτές αφορούν 30 διαφορετικές εταιρείες, για τις οποίες καταγράψαμε το μηνιαίο κέρδος (y) σε χιλιάδες ευρώ που είχαν σε σχέση με τους τόνους (x) των πρώτων υλικών που εξόρυξαν από ορυχεία.

Η γραφική παράσταση του Σχήματος 7.2 κατασκευάζεται με τις ακόλουθες εντολές με τη βοήθεια των οποίων

δηλώνουμε και τις μεταβλητές x και y που περιέχουν τις τιμές των παρατηρήσεων.

```
>> x=[0.41,0.56,0.91,1.09,1.22,1.4,1.53,1.55,1.57,1.74,2,2.29,2.86,2.94,3.54,
3.91,4.34,4.82,6.19,6.98,7.16,7.3,7.64,7.73,7.81,8.14,8.23,8.68,9.41,9.99];
>> y=[4.89,2.94,6.3,10.48,7.35,12.91,14.01,12.63,10.74,13.37,17.95,21.29,27.4,
26.91,32.03,33.21,33.58,40.62,36.47,29.02,27.38,24.17,19.55,13.97,13.66,5.79,
4.82,-12.51,-40.38,-67.63];
>> plot(x,y,'k.','MarkerSize',10)
>> xlabel('x')
>> ylabel('y')
```

Από το γράφημα παρατηρούμε ότι το κέρδος ακολουθεί μια αυξητική πορεία, παρουσιάζει ένα μέγιστο ανάμεσα στους 5 με 6 τόνους και στη συνέχεια φθίνει. Μάλιστα, φαίνεται ότι για μεγάλες ποσότητες εξόρυξης (πάνω από 8) οι εταιρείες παρουσιάζουν ζημίες (αρνητικό κέρδος). Το γεγονός αυτό μπορεί να οφείλεται στο υψηλό κόστος εξόρυξης και στην αδυναμία απορρόφησης των υλικών από την αγορά.

Παρατήρηση 7.11

Κατά την κατασκευή του γραφήματος του Σχήματος 7.2 χρησιμοποιήθηκε η επιλογή `'MarkerSize'` στην εντολή `plot`. Η επιλογή αυτή καθορίζει το μέγεθος (εδώ το 10) των σημείων στο γράφημα.

Αυτό που επιθυμούμε σε ένα τέτοιο πρόβλημα είναι η εκτίμηση της σχέσης μεταξύ του μηνιαίου κέρδους (y) - κατά μέσο όρο - και των τόνων (x) των πρώτων υλικών που εξορύχθηκαν. Στη προσπάθειά μας αυτή θα βασιστούμε στα πολυώνυμα και στην ιδιότητά τους να προσεγγίζουν ικανοποιητικά οποιαδήποτε συνεχή συνάρτηση. Πιο συγκεκριμένα, θα χρησιμοποιήσουμε την εντολή `polyfit` για την προσαρμογή, δηλαδή την εκτίμηση/προσδιορισμό, του βέλτιστου πολυωνύμου στα δεδομένα[2].

Η εντολή `polyfit` συντάσσεται με τον ακόλουθο τρόπο

```
polyfit(x,y,k)
```

όπου k ο βαθμός του πολυωνύμου που επιθυμούμε να προσαρμόσουμε στα σημεία $(x_i, y_i), i = 1, 2, \ldots, n$ και επιστρέφει το διάνυσμα των συντελεστών του πολυώνυμου $k^{ου}$ που προσαρμόζεται καλύτερα στα δεδομένα.

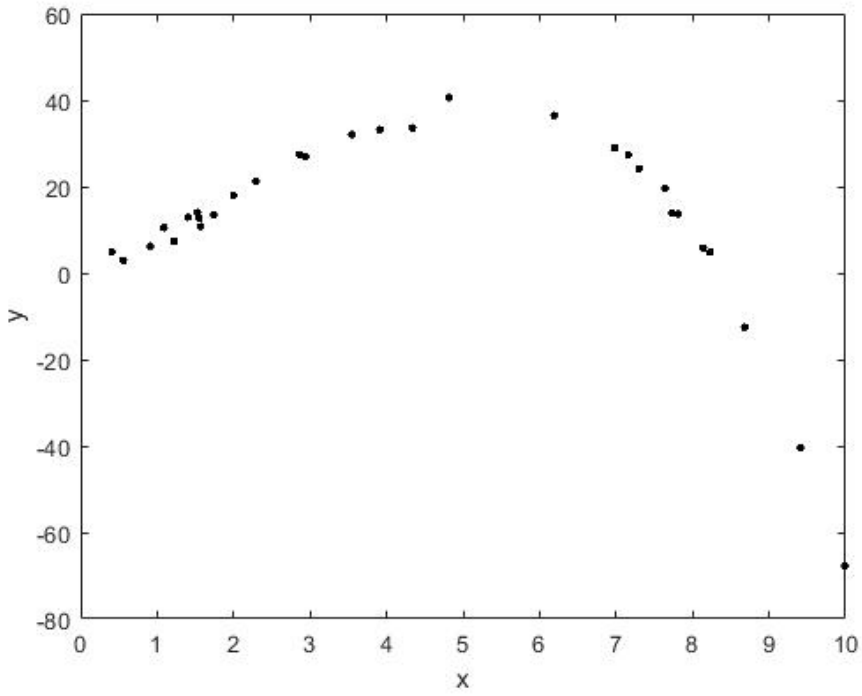
Εκτελώντας τις ακόλουθες εντολές μπορούμε να προσαρμόσουμε πολυώνυμα $1^{ου}$, $2^{ου}$ και $3^{ου}$ βαθμού στα δεδομένα και να προσθέσουμε τις γραφικές τους παραστάσεις στο γράφημα του Σχήματος 7.2.

```
>> hold ON
>> coef1=polyfit(x,y,1);
>> xpoints=0:0.01:10;
>> newy1=polyval(coef1,xpoints);
>> plot(xpoints,newy1,'-b','LineWidth',2)
>> coef2=polyfit(x,y,2);
>> newy2=polyval(coef2,xpoints);
>> plot(xpoints,newy2,'-r','LineWidth',2)
>> coef3=polyfit(x,y,3);
>> newy3=polyval(coef3,xpoints);
>> plot(xpoints,newy3,'-k','LineWidth',2)
>> legend('(xi,yi)','1ou','2ou','3ou')
>> hold OFF
```

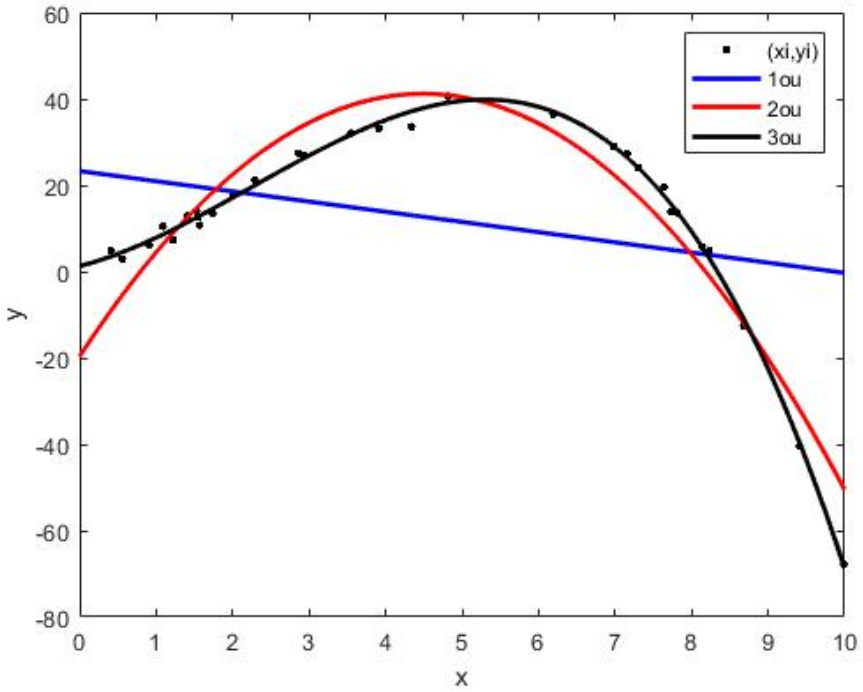
[2] Η διαδικασία προσαρμογής του πολυωνύμου βασίζεται στη μέθοδο ελαχίστων τετραγώνων. Η μέθοδος ελαχίστων τετραγώνων αποτελεί μία από τις βασικότερες τεχνικές προσαρμογής μοντέλων παλινδρόμησης - όπως εδώ ενός πολυωνυμικού μοντέλου - στη στατιστική. Για περισσότερες και αναλυτικότερες πληροφορίες για τη μέθοδο ελαχίστων τετραγώνων ο αναγνώστης παραπέμπεται, μεταξύ άλλων, στο βιβλίο των Καρώνη και Οικονόμου (2017).

x	y	x	y
0.41	4.89	3.91	33.21
0.56	2.94	4.34	33.58
0.91	6.3	4.82	40.62
1.09	10.48	6.19	36.47
1.22	7.35	6.98	29.02
1.4	12.91	7.16	27.38
1.53	14.01	7.3	24.17
1.55	12.63	7.64	19.55
1.57	10.74	7.73	13.97
1.74	13.37	7.81	13.66
2.00	17.95	8.14	5.79
2.29	21.29	8.23	4.82
2.86	27.4	8.68	-12.51
2.94	26.91	9.41	-40.38
3.54	32.03	9.99	-67.63

Πίνακας 7.1: Πίνακας δεδομένων του μηνιαίου κέρδους (y) σε χιλιάδες ευρώ και των τόνων (x) των πρώτων υλικών που εξόρυξαν από ορυχεία 30 διαφορετικές εταιρείες.



Σχήμα 7.2: Διάγραμμα διασκόρπισης του μηνιαίου κέρδους (y) σε χιλιάδες ευρώ και των τόνων (x) των πρώτων υλικών που εξόρυξαν από ορυχεία 30 διαφορετικές εταιρείες.



Σχήμα 7.3: Διάγραμμα διασκόρπισης του μηνιαίου κέρδους (y) σε χιλιάδες ευρώ και των τόνων (x) των πρώτων υλικών που εξόρυξαν από ορυχεία 30 διαφορετικές εταιρείες και τα εκτιμώμενα πολυώνυμα $1^{ου}$, $2^{ου}$ και $3^{ου}$ βαθμού.

Πιο συγκεκριμένα, στην πρώτη εντολή ζητάμε από το Matlab να κρατήσει τη γραφική παράσταση των σημείων των παρατηρήσεων που είχαμε κατασκευάσει νωρίτερα και να προσθέσει σε αυτήν οποιαδήποτε γραφική παράσταση δημιουργήσουμε στη συνέχεια. Ύστερα, ζητάμε με την εντολή `coef1=polyfit(x,y,1)` την προσαρμογή ενός πολυωνύμου $1^{ου}$ βαθμού στα δεδομένα x και y και την αποθήκευση των συντελεστών του στη μεταβλητή $coef1$. Οι τρεις επόμενες εντολές αφορούν την κατασκευή της γραφικής παράστασης του πολυωνύμου $1^{ου}$ βαθμού, η οποία αποτυπώνεται με μπλε χρώμα στο γράφημα του Σχήματος 7.3.

Στη συνέχεια, το δεύτερο και το τρίτο μπλοκ εντολών προσαρμόζουν και σχεδιάζουν τις καμπύλες (με κόκκινο και μαύρο χρώμα) του $2^{ου}$ και $3^{ου}$ βαθμού πολυωνύμων, αντίστοιχα. Από τη γραφική παράσταση του Σχήματος 7.3 είναι φανερό ότι το πολυώνυμο $3^{ου}$ βαθμού παρουσιάζει μια ικανοποιητική προσαρμογή στα δεδομένα και καταφέρνει να περιγράψει τη συνάρτηση που συνδέει τα δύο υπό μελέτη μεγέθη στο διάστημα $[0,10]$. Τέλος, η προτελευταία εντολή προσθέτει τη λεζάντα στο πάνω δεξί μέρος του γραφήματος που μας πληροφορεί για τον χρωματικό προσδιορισμό των πολυωνύμων, ενώ η τελευταία αποδεσμεύει τη γραφική παράσταση.

Παρατήρηση 7.12

Κατά την κατασκευή των γραφημάτων των πολυωνύμων του Σχήματος 7.3 χρησιμοποιήθηκε η επιλογή `'LineWidth'` στην εντολή `plot`. Η επιλογή αυτή καθορίζει το πάχος (εδώ 2) της καμπύλης του γραφήματος.

7.7 Ασκήσεις

Άσκηση 7.1. *Ποια από τις παρακάτω εντολές επιστρέφει το μέτρο των ριζών του πολυωνύμου* $3x^4 + x^2 + x$*;*

```
1) abs(roots([3 1  1]))
2) roots(abs([3 1  1]))
3) abs(roots([3 0 1  1]))
4) roots(abs([3 0 1  1]))
5) other. . . . . .
```

Άσκηση 7.2. *Έστω τα πολυώνυμα*

$$\pi_1(x) = 2x^5 + 3x^3 + 4x$$
$$\pi_2(x) = -4x^3 + 2x - 1$$

- *Να υπολογίσετε το άθροισμα, τη διαφορά, το γινόμενο και το αποτέλεσμα της διαίρεσης των δύο πολυωνύμων.*
- *Να γίνει η γραφική παράσταση του πολυωνύμου* $\mu(x) = \pi_1(x) * \cdot \pi_2(x)$ *για* $x \in (-1,1)$ *με τη βοήθεια της εντολής* `polyval`*.*
- *Βρείτε όλες τις ρίζες του πολυωνύμου* $\mu(x)$*.*
- *Βρείτε το μέτρο και το όρισμα των ριζών του πολυωνύμου* $\mu(x)$*.*
- *Παραστήστε γραφικά τις ρίζες του πολυωνύμου* $\mu(x)$*.*

Άσκηση 7.3. *Να υπολογίσετε την τιμή του πολυωνύμου*

$$\alpha(x) = \sum_{n=0} 100(-1)^n x^n$$

για $x = -1, x = 0$ *και για* $x = 1$*. Στη συνέχεια, να υπολογίσετε την παράγωγο και το ολοκλήρωμα του* $\alpha(x)$*.*

Άσκηση 7.4. *Να βρείτε το μέτρο και το όρισμα όλων των ριζών του πολυωνύμου με συντελεστές*

$$1,\ 1,\ 0,\ 0,\ 0,\ 0,\ -11$$

Να γραφτεί ένα script αρχείο, το οποίο:

1. *Θα ζητάει από τον χρήστη να εισάγει έναν* $nx4$ *πίνακα, ο οποίος περιέχει τους συντελεστές* n *πολυωνύμων* $3^{ου}$ *βαθμού (δηλαδή θεωρούμε ότι κάθε σειρά αποτελείται από τους συντελεστές ενός πολυωνύμου* $3^{ου}$ *βαθμού).*
2. *Στην περίπτωση που ο χρήστης δεν εισάγει έναν* $nx4$ *πίνακα, το πρόγραμμα θα πρέπει να ζητάει επανειλημμένως από τον χρήστη να εισάγει έναν* $nx4$ *πίνακα μέχρις ότου αυτός το πράξει.*
3. *Στη συνέχεια, το πρόγραμμα θα πρέπει να επιστρέφει τις ρίζες κάθε πολυωνύμου υπό μορφή ενός* $nx3$ *πίνακα.*

Άσκηση 7.5. *Δημιουργήστε ένα script αρχείο το οποίο:*

1. *Θα υπολογίζει τον χρόνο που χρειάζεται ένα αντικείμενο, το όποιο ρίχνεται ευθεία πάνω, για να χτυπήσει το έδαφος.*
2. *Το μέγιστο ύψος στο οποίο θα φτάσει το αντικείμενο.*

Σημειώνεται ότι η εξίσωση

$$S(t) = -1/2gt^2 + v_0t + h_0$$

παριστάνει το ύψος στο οποίο βρίσκεται το αντικείμενο τη χρονική στιγμή t (μηδενικό ύψος σημαίνει ότι το αντικείμενο χτύπησε στο έδαφος), όταν εκτοξεύεται από αρχικό ύψος h_0 με αρχική ταχύτητα v_0. Με g συμβολίζεται η επιτάχυνση της βαρύτητας, η οποία θεωρούμε ότι είναι ίση με $9.8m/s^2$.

Το πρόγραμμα θα πρέπει να ζητάει από τον χρήστη την αρχική ταχύτητα v_0 και το αρχικό ύψος h_0.

Υπόδειξη: *Ο ζητούμενος χρόνος είναι η θετική ρίζα της εξίσωσης $S(t) = 0$.*

7.8 Απαντήσεις στις ασκήσεις αυτοαξιολόγησης

Λύση άσκησης αυτοαξιολόγησης 7.1

Ακολουθώντας παρόμοιο σκεπτικό με την Άσκηση 7.1 πρέπει να πληκτρολογήσουμε τις εντολές

```
>> k=ones(1,201);
>> for n = 2:2:length(k)
       k(n)=0;
   end
```

για να δηλώσουμε στο Matlab το πολυώνυμο $k(x) = x^{200} + x^{198} + ... + x^4 + x^2 + 1$.

Λύση άσκησης αυτοαξιολόγησης 7.2

Για να δημιουργήσουμε ένα αρχείο script το οποίο:

- θα ζητάει από τον χρήστη τα διανύσματα των συντελεστών δύο πολυωνύμων,
- θα επεκτείνει, αν χρειάζεται, το διάνυσμα του πολυωνύμου μικρότερης τάξης με μηδενικά στην αρχή του, έτσι ώστε να αποκτήσει την ίδια διάσταση με το διάνυσμα του πολυωνύμου μεγαλύτερης τάξης και
- θα επιστρέφει στον χρήστη το διάνυσμα των συντελεστών του αθροίσματος των δύο πολυωνύμων

χρειάζεται να πληκτρολογήσουμε τις ακόλουθες εντολές και να τις αποθηκεύσουμε σε ένα αρχείο `m`.

```
1  %POLYADD adds two polynomials given by the user
2  p1=input('Please type the coefficients of the first polynomial:');
3  p2=input('Please type the coefficients of the second polynomial:');
4
5  sp1=length(p1);
6  sp2=length(p2);
7  if sp1<sp2
8      p1=[zeros(1,sp2-sp1),p1];
9  elseif sp2<sp1
10     p2=[zeros(1,sp1-sp2),p2];
11 end
12 disp('The coefficients of sum of the two  polynomials are:')
13 p1+p2
```

Στον παραπάνω κώδικα:

- στις γραμμές 2 και 3 ζητάμε από τον χρήστη τη δήλωση των δύο πολυωνύμων,
- στις γραμμές 8 έως 12 επεκτείνουμε, αν χρειάζεται, το διάνυσμα του πολυωνύμου μικρότερης τάξης με μηδενικά στην αρχή του, έτσι ώστε να αποκτήσει την ίδια διάσταση με το διάνυσμα του πολυωνύμου μεγαλύτερης τάξης. Αυτό γίνεται με τη σύγκριση των διαστάσεων των δύο διανυσμάτων με τους συντελεστές των δύο πολυωνύμων. Στην περίπτωση που δεν έχουν το ίδιο πλήθος στοιχείων, δηλαδή που το μήκος του ενός είναι μικρότερο από το μήκος του άλλου, τότε το διάνυσμα που αναπαριστά το μικρότερο πολυώνυμο επεκτείνεται με μηδενικά στην αρχή του. Παραδείγματος χάριν, στη γραμμή 9 επεκτείνουμε το πολυώνυμο $p1$ με μηδενικά στην αρχή του. Το πλήθος των μηδενικών αυτών καθορίζεται από τη διαφορά $sp1 - sp2$, η οποία εκφράζει τη διαφορά των βαθμών των δύο πολυωνύμων,
- στις γραμμές 13 και 14 επιστρέφουμε το διάνυσμα των συντελεστών του αθροίσματος των δύο πολυωνύμων.

Λύση άσκησης αυτοαξιολόγησης 7.3

Ο προσδιορισμός των ριζών του πολυωνύμου

$$x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$$

μπορεί να γίνει εκτελώντας την εντολή

```
>> roots(ones(1,10))
```

η οποία επιστρέφει τα ακόλουθα αποτελέσματα

```
   0.8090 + 0.5878i
   0.8090 - 0.5878i
   0.3090 + 0.9511i
   0.3090 - 0.9511i
  -1.0000 + 0.0000i
  -0.8090 + 0.5878i
  -0.8090 - 0.5878i
  -0.3090 + 0.9511i
  -0.3090 - 0.9511i
```

Στο παραπάνω διάνυσμα εμφανίζονται οι 9 ρίζες του πολυωνύμου. Παρατηρείστε ότι μόνο μία είναι πραγματική, η -1, και όλες οι υπόλοιπες είναι μη πραγματικές.

Λύση άσκησης αυτοαξιολόγησης 7.4

Ένα πολυώνυμο που έχει ρίζες τους πραγματικούς αριθμούς $1, 2, 3$ και 4 μπορεί να προσδιοριστεί με τη βοήθεια του Matlab πληκτρολογώντας την εντολή

```
>> poly([1,2,3,4])
```

η οποία επιστρέφει το αποτέλεσμα

```
ans =
     1    -10     35    -50     24
```

το οποίο αντιπροσωπεύει το πολυώνυμο $x^4 - 10x^3 + 35x^2 - 50x + 24$.

Λύση άσκησης αυτοαξιολόγησης 7.5

Με τη βοήθεια των εντολών

```
>> p1=[1,1,0,-3,0,1,5];
>> r=roots(p1);
>> metro=abs(r);
>> orisma=radtodeg(angle(r));
>> [metro,orisma]
```

πραγματοποιούμε τις ακόλουθες διαδικασίες:

- δηλώνουμε το πολυώνυμο $\pi_1(x) = x^6 + x^5 - 3x^3 + x + 5$ (γραμμή 1),
- προσδιορίζουμε τις ρίζες του πολυωνύμου και τις αποθηκεύουμε στη μεταβλητή r (γραμμή 2),
- υπολογίζουμε το μέτρο όλων των ριζών και αποθηκεύουμε το αποτέλεσμα στη μεταβλητή metro (γραμμή 3),
- υπολογίζουμε το πρωτεύον όρισμα (σε μοίρες) όλων των ριζών και αποθηκεύουμε το αποτέλεσμα

στη μεταβλητή orisma (γραμμή 4).

Η τελευταία εντολή επιστρέφει τα αποτελέσματα στην ακόλουθη μορφή

```
ans =
    1.2420   23.5173
    1.2420  -23.5173
    1.5313  139.7912
    1.5313 -139.7912
    1.1757  113.5298
    1.1757 -113.5298
```

Στην πρώτη στήλη του πίνακα δηλώνονται τα μέτρα των ριζών του πολυωνύμου, ενώ στη δεύτερη τα πρωτεύοντα ορίσματά τους σε μοίρες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ελληνόγλωσση

Καρώνη, Χ. και Οικονόμου, Π. (2017). *Στατιστικά Μοντέλα Παλινδρόμησης*. (2η έκδ.), Αθήνα: Συμεών.

Ξενόγλωσση

Abel, N. (1826). Beweis der Unmöglichkeit, algebraische Gleichungen von höheren Graden als dem vierten allgemein aufzulösen. ger. *Journal für die reine und angewandte Mathematik*, *1*, pp. 65–84.

Fine, B. and Rosenberger, G. (2012). *The Fundamental Theorem of Algebra*. Undergraduate Texts in Mathematics. Springer New York.

Jeffreys, H., Jeffreys, B. and Swirles, B. (1999). *Methods of Mathematical Physics*. Cambridge Mathematical Library. Cambridge University Press.

Siegmund-Schultze, R. (2016). Weierstraß's Approximation Theorem (1885) and his 1886 lecture course revisited. In: *Karl Weierstraß (1815-1897): Aspekte seines Lebens und Werkes - Aspects of his Life and Work*. Ed. by W. König and J. Sprekels. Springer Spektrum, Wiesbaden.

ΚΕΦΑΛΑΙΟ 8

ΠΙΝΑΚΕΣ

Σύνοψη:

Σε αυτό το κεφάλαιο παρουσιάζεται ο χειρισμός των διανυσμάτων και των πινάκων από το Matlab. Συγκεκριμένα, παρουσιάζονται οι τρόποι δημιουργίας τέτοιων δομών, οι εντολές που χρησιμοποιούνται, η επεξεργασία τους και οι αριθμητικές πράξεις. Ιδιαίτερη έμφαση δίνεται στις τεχνικές που αφορούν τη χρήση υποπινάκων, καθώς και στην αξιοποίησή τους για τη βελτίωση του κώδικα. Επιπρόσθετα, αναφέρονται μια σειρά από εντολές, οι οποίες επιτρέπουν τη χρήση των πινάκων για σύνθετα προβλήματα, όπως η επίλυση συστημάτων, ενώ, τέλος, γίνεται μια αναφορά και στους λεγόμενους πολυδιάστατους πίνακες.

Προαπαιτούμενη γνώση: Τα κεφάλαια 3, 4 και 5 του παρόντος συγγράμματος.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του κεφαλαίου αυτού, θα γνωρίζετε:

- για τη δημιουργία διανυσμάτων και πινάκων (διδιάστατων ή πολυδιάστατων),
- για την επεξεργασία των στοιχείων ενός πίνακα,
- για τον προγραμματισμό με χρήση υποπινάκων,
- για την τέλεση πράξεων μεταξύ διανυσμάτων ή/και πινάκων,
- για τον υπολογισμό των βασικών μεγεθών που χαρακτηρίζουν έναν πίνακα,
- για την επίλυση γραμμικών αλγεβρικών συστημάτων.

Γλωσσάριο επιστημονικών όρων

- Ανάστροφος πίνακας
- Αντίστροφος πίνακας
- Γραμμικό αλγεβρικό σύστημα
- Εσωτερικό γινόμενο διανυσμάτων
- Ιδιοτιμές πίνακα
- Ορίζουσα πίνακα
- Πολλαπλασιασμός πινάκων
- Ταυτοτικός πίνακας

8.1 Εισαγωγή

Οι πίνακες αποτελούν ένα πολύ χρήσιμο εργαλείο για την επίλυση σύνθετων προβλημάτων σε όλο το εύρος των θετικών επιστημών και των επιστημών του μηχανικού. Το βασικό χαρακτηριστικό τους είναι ότι μας επιτρέπουν να αποθηκεύουμε και να μεταχειριζόμαστε πλήθος τιμών, μέσω μίας μόνο δομής. Έτσι, μπορούμε να συγκεντρώσουμε σε έναν πίνακα πολλά και, πιθανώς, ανόμοια χαρακτηριστικά ενός φυσικού ή μηχανικού συστήματος, να διερευνήσουμε τις ιδιότητές του και να κατανοήσουμε πώς συμπεριφέρεται.

Σε πρακτικό επίπεδο, μπορούμε να χρησιμοποιήσουμε τους πίνακες για να εκτελέσουμε αριθμητικές πράξεις σε ένα μεγάλο πλήθος αριθμών, με συγκεκριμένο τρόπο. Είδαμε, για παράδειγμα, κάποιες βασικές λειτουργίες στο Κεφάλαιο 1, όπου περιγράψαμε τις πράξεις στοιχείο προς στοιχείο μεταξύ δύο πινάκων, καθώς και στο Κεφάλαιο 6, όπου υπολογίσαμε τις τιμές μίας συνάρτησης δύο μεταβλητών και κατασκευάσαμε τη γραφική της παράσταση.

Στο κεφάλαιο αυτό, θα επικεντρωθούμε στην έννοια του πίνακα στον προγραμματισμό και θα αναφερθούμε εκτενώς στις λειτουργίες του. Ειδικότερα, στο πλαίσιο του Matlab η έννοια του πίνακα – όπως μπορούμε να αντιληφθούμε και μόνο από το όνομα του λογισμικού – είναι από τις πιο κεντρικές. Ωστόσο, ξεκινώντας, πρέπει να ρωτήσουμε τι είναι «ένας πίνακας».

Μέχρι στιγμής έχουμε αναφερθεί στους πίνακες δύο διαστάσεων, οι οποίοι είναι διατάξεις στοιχείων σε γραμμές και στήλες. Για παράδειγμα, ο

$$\begin{bmatrix} 1 & 2 & 3 \\ -1 & 2 & 0 \end{bmatrix}$$

είναι ένας πίνακας που αποτελείται από 6 στοιχεία διατεταγμένα σε 2 σειρές και 3 στήλες. Μία τέτοια διάταξη ονομάζεται πίνακας 2×3. Ειδική περίπτωση αποτελούν οι πίνακες οι οποίοι έχουν μία μόνο γραμμή ή μία μόνο στήλη, οι οποίοι ονομάζονται *διάνυσμα-γραμμή* ή *διάνυσμα-στήλη*, αντίστοιχα. Για παράδειγμα, ένα διάνυσμα με μία γραμμή και τρεις στήλες – διάνυσμα 1×3 – είναι το

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

ενώ ένα διάνυσμα-στήλη 3×1 είναι το

$$\begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

Τα διανύσματα μπορούν να θεωρηθούν ως μονοδιάστατοι πίνακες, ενώ οι πίνακες $m \times n$ ονομάζονται διδιάστατοι. Στις επόμενες ενότητες θα μας απασχολήσουν τα παραπάνω είδη πινάκων, ενώ θα επεκταθούμε και σε πίνακες περισσότερων διαστάσεων. Θα αναφερθούμε σε ειδικούς τύπους πινάκων, όπως ο μοναδιαίος και ο μηδενικός πίνακας, και σε εντολές που μας βοηθούν να ελέγξουμε εύκολα τα στοιχεία ενός πίνακα (`find`). Τέλος, θα παρουσιάσουμε τον τρόπο με τον οποίο μπορούμε να χρησιμοποιήσουμε τμήματα ενός πίνακα (υποπίνακες) στους υπολογισμούς και, γενικότερα, θα δοθεί έμφαση στον *προγραμματισμό με διανύσματα* (array programming).

8.2 Μονοδιάστατοι πίνακες – Διανύσματα

Στην ενότητα αυτή θα εξετάσουμε τους τρόπους κατασκευής διανυσμάτων, καθώς και τις βασικές πράξεις μεταξύ αυτών. Κάποια βασικά στοιχεία έχουν ήδη αναφερθεί στο Κεφάλαιο 1, ενώ πληροφορίες σε σχέση με μαθηματικούς ορισμούς και πράξεις μπορούν να βρεθούν στο Παράρτημα Α΄.

8.2.1 Κατασκευή διανυσμάτων

Η κατασκευή ενός διανύσματος μπορεί να γίνει με πολλούς τρόπους, ανάλογα με την πολυπλοκότητα των στοιχείων, τη διάταξή τους μέσα στο διάνυσμα και το μέγεθος του διανύσματος. Στις επόμενες παραγράφους θα αναφερθούμε στους βασικούς τρόπους κατασκευής ενός διανύσματος.

8.2.1.1 Κατασκευή διανυσμάτων με εισαγωγή στοιχείων

Για να κατασκευάσουμε ένα διάνυσμα-γραμμή, εισάγουμε τα στοιχεία του διαχωρισμένα με κόμματα ή κενά, όπως φαίνεται παρακάτω

```
>> arrayRow = [1,2,3]

arrayRow =

     1     2     3

 >> arrayRow2 = [1 2 3]

arrayRow2 =

     1     2     3
```

Μία ακόμα επιλογή που υπάρχει στο Matlab είναι να συνενώσουμε διανύσματα, τα οποία έχουν δημιουργηθεί σε προηγούμενο χρόνο, και να κατασκευάσουμε ένα ενιαίο διάνυσμα. Για παράδειγμα, συνδυάζοντας τα διανύσματα-γραμμή arrayRow και arrayRow2 που δημιουργήσαμε προηγουμένως, μπορούμε να κατασκευάσουμε το arrayAll

```
>> arrayAll = [arrayRow , 2*arrayRow2]

arrayAll =

     1     2     3     2     4     6
```

το οποίο περιέχει σε σειρά το arrayRow και το arrayRow2 πολλαπλασιασμένο επί δύο.

Τέλος, μπορούμε να προσθέσουμε τιμές σε οποιαδήποτε θέση ενός διανύσματος και να δημιουργήσουμε νέες θέσεις. Για παράδειγμα, αν στο διάνυσμα arrayRow, το οποίο όπως είδαμε προηγουμένως είχε τρία στοιχεία, εισάγουμε ένα στοιχείο στη θέση 5, τότε αυτόματα το μήκος του arrayRow γίνεται 5. Στη θέση 4, για την οποία δεν έχουμε δώσει κάποιον αριθμό, τοποθετείται η τιμή 0.

```
>> arrayRow = [1,2,3]

arrayRow =

     1     2     3

>> arrayRow(5) = 12

arrayRow =
```

```
     1     2     3     0    12

>> arrayRow(8) = 2

arrayRow =

     1     2     3     0    12     0     0     2
```

Εκτός από τα διανύσματα-γραμμή, υπάρχουν και τα διανύσματα-στήλη. Για να ορίσουμε ένα διάνυσμα-στήλη, ακολουθούμε αντίστοιχη διαδικασία με το διάνυσμα-γραμμή, διαχωρίζοντας τα στοιχεία με το σύμβολο “;”:

```
>> arrayColumn = [1;2;3]

arrayColumn =

     1
     2
     3
```

Μπορούμε, επίσης, να συνδυάσουμε δύο διανύσματα-στήλη για να κατασκευάσουμε ένα ενιαίο διάνυσμα-στήλη, όπως φαίνεται παρακάτω:

```
>> arrayColumnAll = [arrayColumn ; 5*arrayColumn]

arrayColumnAll =

     1
     2
     3
     5
    10
    15
```

8.2.1.2 Κατασκευή διανύσματος-γραμμής με χρήση του τελεστή “:”

Η κατασκευή διανυσμάτων με εισαγωγή στοιχείων, αν και απλή στην εφαρμογή, δεν είναι πρακτική όταν έχουμε μεγάλο πλήθος στοιχείων, όπως για παράδειγμα στα διανύσματα που χρησιμοποιήθηκαν στην κατασκευή γραφικών παραστάσεων στο Κεφάλαιο 5. Ένας εναλλακτικός τρόπος να ορίσουμε διανύσματα-γραμμή, τα οποία περιέχουν διατεταγμένες και ισαπέχουσες τιμές, είναι κάνοντας χρήση του τελεστή “:”, που έχουμε ήδη δει από τις επαναληπτικές διαδικασίες. Αν, λοιπόν, θέλουμε να κατασκευάσουμε ένα διάνυσμα-γραμμή, το οποίο να περιέχει τους ακέραιους από το 1 έως το 5, μπορούμε να γράψουμε

```
>> arrayRow = 1:5

arrayRow =

     1     2     3     4     5
```

Παρατηρούμε ότι με τον τρόπο αυτό ορισμού διανυσμάτων, δεν είναι απαραίτητη η χρήση αγκύλων. Αν τώρα θέλαμε να ορίσουμε ένα διάνυσμα-γραμμή το οποίο θα περιείχε μόνο τους περιττούς αριθμούς στο διάστημα $[1,10]$, θα μπορούσαμε να χρησιμοποιήσουμε την τιμή 2 για το βήμα στον ορισμό του διανύσματος, όπως φαίνεται παρακάτω

```
>> arrayRow = 1:2:10

arrayRow =

     1     3     5     7     9
```

Όπως μπορούμε να δούμε και από την έξοδο, οι αριθμοί που περιέχονται στο διάνυσμά μας είναι περιττοί και φράσσονται από τα δύο ακραία ορίσματα της εντολής μας. Με αντίστοιχο τρόπο θα μπορούσαμε να ορίσουμε ένα διάνυσμα με τις τιμές αυτές, ταξινομημένες σε φθίνουσα σειρά, χρησιμοποιώντας την τιμή -2 για το βήμα.

```
>> arrayRow = 9:-2:1

arrayRow =

     9     7     5     3     1
```

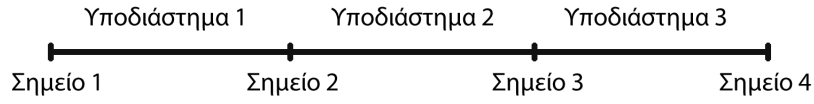
Η γενίκευση του παραπάνω είναι η εξής: η εντολή $a : step : b$, όταν εφαρμόζεται στο πλαίσιο δημιουργίας διανυσμάτων, επιστρέφει όλες τις τιμές που περιέχονται στο κλειστό διάστημα $[a, b]$ και προκύπτουν ξεκινώντας από την τιμή a με βήμα $step$ έως και την τιμή b, όπου αυτό είναι δυνατόν.

8.2.1.3 Κατασκευή διανύσματος-γραμμής με την εντολή linspace

Σε ορισμένες εφαρμογές είναι πιο χρήσιμο να ορίσουμε τον αριθμό των σημείων που θέλουμε να έχουμε από το a μέχρι το b, από το να ορίσουμε την τιμή του βήματος $step$. Για παράδειγμα, αν θέλουμε να χωρίσουμε το διάστημα από το 0 μέχρι το 1 σε τρία ίσα υποδιαστήματα, τότε πρέπει να χρησιμοποιήσουμε βήμα $1/3 = 0.33333...$, αριθμός που έχει άπειρα δεκαδικά ψηφία.

Για να αποφύγουμε σφάλματα στους υπολογισμούς μας λόγω προσεγγίσεων στο βήμα, μπορούμε να χρησιμοποιήσουμε την εντολή linspace(a,b,num). Η εντολή αυτή δημιουργεί ένα πλήθος num γραμμικά ισαπέχοντων τιμών μεταξύ των a και b, ώστε το διάνυσμα-γραμμή που θα προκύψει να χωριστεί σε $num - 1$ ίσα υποδιαστήματα, όπως φαίνεται στο Σχήμα 8.1.

```
>> arrayRowLin = linspace(0,1,4)

arrayRowLin =

         0    0.3333    0.6667    1.0000
```



Σχήμα 8.1: Για να χωρίσουμε ένα διάστημα σε τρία υποδιαστήματα, χρειαζόμαστε τέσσερα σημεία.

Παρατήρηση 8.1

Είδαμε στο παραπάνω παράδειγμα ότι, για να χωρίσουμε το διάστημα από το 0 μέχρι το 1 σε τρία ίσα υποδιαστήματα, χρειαζόμαστε τέσσερα σημεία: δύο για τα όρια και δύο για τα ενδιάμεσα χωρίσματα. Γενικά, για να χωρίσουμε ένα διάστημα σε N ίσα υποδιαστήματα, χρειαζόμαστε $N + 1$ σημεία, εκ των οποίων τα δύο αντιστοιχούν στα όρια του διαστήματος και τα υπόλοιπα στα εσωτερικά σημεία.

8.2.2 Δείκτες και προσπέλαση των στοιχείων ενός διανύσματος

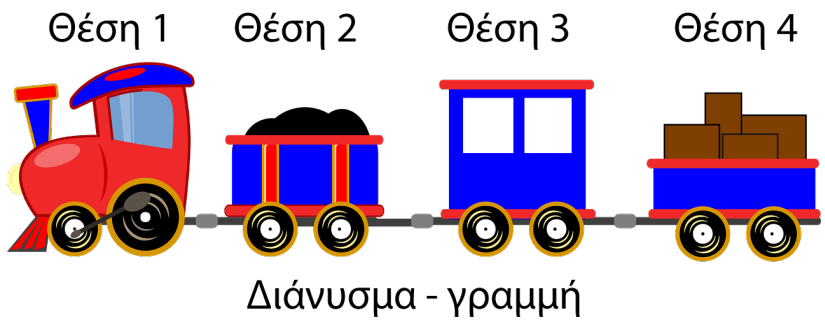
Είδαμε στις προηγούμενες παραγράφους ότι τα διανύσματα είναι δομές που περιέχουν ένα πλήθος τιμών με μία συγκεκριμένη διάταξη. Σε κάθε τιμή ενός διανύσματος αντιστοιχεί ένας δείκτης, που προσδιορίζει τη θέση της συγκεκριμένης τιμής μέσα στο διάνυσμα. Αυτό μπορούμε να το αντιληφθούμε ως εξής: αν παρομοιάσουμε το διάνυσμα με ένα τρένο, τότε οι θέσεις του διανύσματος είναι τα βαγόνια. Η αρίθμηση των θέσεων-βαγονιών ξεκινάει από το 1 και σε κάθε θέση-βαγόνι μπορεί να τοποθετηθεί μία μόνο τιμή (Σχήμα 8.2). Αν θεωρήσουμε ένα διάνυσμα με 4 θέσεις, αυτό σημαίνει ότι υπάρχουν 4 τιμές στις οποίες μπορώ να αναφερθώ με τον αντίστοιχο δείκτη. Όπως έχουμε δει, για να ορίσουμε ή να προσπελάσουμε μία συγκεκριμένη θέση ενός διανύσματος, πρέπει να τοποθετήσουμε τον δείκτη της μέσα σε παρενθέσεις αμέσως μετά το όνομα του διανύσματος. Για παράδειγμα, ας ορίσουμε το εξής διάνυσμα-γραμμή

```
>> array = 10:15

array =

    10    11    12    13    14    15
```

Το διάνυσμά μας περιλαμβάνει έξι τιμές, οι οποίες καταλαμβάνουν τις θέσεις 1 έως 6. Μπορούμε, λοιπόν, να ανακαλέσουμε/προσπελάσουμε καθεμία από τις έξι τιμές χρησιμοποιώντας τον δείκτη της θέσης της.

```
>> array(1)
ans =
    10
>> array(2)
ans =
    11
>> array(3)
ans =
    12
>> array(4)
ans =
    13
>> array(5)
ans =
    14
>> array(6)
ans =
    15
```

Επιπλέον, μπορούμε να ανακαλέσουμε με μία εντολή περισσότερες από μία τιμές του διανύσματος, χρησιμοποιώντας τους αριθμούς των θέσεων στις οποίες βρίσκονται. Για παράδειγμα, αν θέλουμε να ανακαλέσουμε τις τιμές που βρίσκονται στις θέσεις 2 και 5, μπορούμε να γράψουμε



Σχήμα 8.2: Ένα διάνυσμα μπορεί να παρομοιαστεί με ένα τρένο. Κάθε θέση του διανύσματος αντιστοιχεί σε ένα βαγόνι του τρένου.

```
>> array([2,5])

ans =

    11     14
```

Παρατηρούμε ότι τις θέσεις 2 και 5 δεν τις εισαγάγαμε στο όρισμα του array ως ξεχωριστούς αριθμούς, αλλά ως τα στοιχεία ενός διανύσματος, δηλαδή μέσα σε αγκύλες! Αυτό σημαίνει ότι, αν επιθυμούμε να χρησιμοποιούμε ένα εύρος θέσεων, πρέπει να τις ορίσουμε ως στοιχεία ενός δεύτερου διανύσματος.

Μπορούμε, επίσης, να αφαιρέσουμε μία ή περισσότερες θέσεις ενός διανύσματος, θέτοντάς τις ίσες με το κενό διάνυσμα, δηλαδή με "[ ]". Για παράδειγμα, μπορούμε να διαγράψουμε τις θέσεις 2 έως 4 του διανύσματος array, που είδαμε παραπάνω, ως εξής

```
>> array(2:4)=[ ]

array =

    10     14     15
```

Παράδειγμα 8.1

Ορίστε ένα διάνυσμα-γραμμή που να περιέχει 41 σημεία από το 0 μέχρι και το 1, σε ίσες αποστάσεις. Στη συνέχεια, δημιουργήστε ένα δεύτερο διάνυσμα που να περιέχει μόνο την αρχική, τη μεσαία και την τελική τιμή του πρώτου διανύσματος.

Λύση Παραδείγματος 8.1

Για να κατασκευάσουμε το πρώτο διάνυσμα, θα επιλέξουμε την εντολή `linspace`, καθώς έχουμε ως δεδομένα την αρχική τιμή, την τελική τιμή και τον αριθμό των στοιχείων. Συγκεκριμένα, αν ονομάσουμε το διάνυσμα αυτό A, θα πληκτρολογήσουμε

```
>> A = linspace(0,1,41);
```

Στη συνέχεια, θα δημιουργήσουμε ένα νέο διάνυσμα B ως εξής: γνωρίζουμε ότι η πρώτη τιμή βρίσκεται στη θέση 1 και η τελευταία στη θέση 41. Η μεσαία τιμή θα βρίσκεται στο μέσο μεταξύ του 1 και του 41, δηλαδή στη θέση 21 (δηλαδή 20 θέσεις μετά την πρώτη τιμή). Για να κατασκευάσουμε, λοιπόν, το

διάνυσμα B, θα χρησιμοποιήσουμε στο όρισμα του A τις τιμές 1, 21 και 41. Όπως έχουμε αναφέρει, οι τιμές αυτές πρέπει να βρίσκονται μέσα σε αγκύλες για να μπορέσουμε να τις συμπεριλάβουμε στο όρισμα του A.

```
>> B = A([1,21,41])

B =

         0    0.5000    1.0000
```

Άσκηση αυτοαξιολόγησης 8.1

Δοκιμάστε να επαναλάβετε το Παράδειγμα 8.1, χωρίς τη χρήση της εντολής `linspace`. Θα πρέπει να χρησιμοποιήσετε τον τελεστή ":" σε συνδυασμό με κατάλληλο βήμα.

Παρατήρηση 8.2

Στο Παράδειγμα 8.1, γνωρίζαμε από την εκφώνηση ότι η τελευταία τιμή του διανύσματος A βρίσκεται στη θέση 41. Ωστόσο, πολλές φορές δεν είναι άμεσα διαθέσιμη η πληροφορία αυτή, καθώς το διάνυσμα μπορεί να είναι αποτέλεσμα κάποιας πράξης ή να έχει εισαχθεί από τον χρήστη. Στην περίπτωση αυτή, μπορούμε να προσπελάσουμε το τελευταίο στοιχείο του διανύσματος χρησιμοποιώντας ως δείκτη την εντολή `end`. Συνεπώς, μία εναλλακτική λύση του 2ου ερωτήματος του Παραδείγματος 8.1 θα ήταν η εξής

```
>> B = A([1,21,end])

B =

         0    0.5000    1.0000
```

Σε κάθε περίπτωση, η θέση του τελευταίου στοιχείου ενός διανύσματος μπορεί να βρεθεί εύκολα με την εντολή `length`, η οποία επιστρέφει το μήκος του διανύσματος και, συνεπώς, μας δείχνει ποια είναι η τελευταία θέση.

Παράδειγμα 8.2

Ορίστε ένα διάνυσμα-γραμμή με όνομα Z που να περιέχει όλους τους ακέραιους αριθμούς από το -10 έως το 10. Στη συνέχεια, χρησιμοποιώντας το Z, δημιουργήστε ένα νέο διάνυσμα-γραμμή Zpos, το οποίο να περιέχει μόνο τις θετικές τιμές του Z.

Λύση Παραδείγματος 8.2

Θα κατασκευάσουμε το πρώτο διάνυσμα χρησιμοποιώντας τον τελεστή ":" , καθώς γνωρίζουμε ότι το βήμα είναι μονάδα. Συγκεκριμένα, θα πληκτρολογήσουμε

```
>> Z = -10:10;
```

Στη συνέχεια, θα δημιουργήσουμε ένα νέο διάνυσμα Zpos ως εξής: γνωρίζουμε ότι το διάνυσμα Z έχει 21 συνολικά στοιχεία (10 αρνητικά, το 0 και 10 θετικά) τοποθετημένα σε αύξουσα σειρά. Για να αποθηκεύσουμε στο Zpos μόνο τα θετικά, πρέπει να επιλέξουμε τα στοιχεία του Z που βρίσκονται μετά την 11η θέση, στην οποία βρίσκεται το μηδέν. Για να επιβεβαιώσουμε τον συλλογισμό αυτόν, ελέγχουμε την τιμή του στοιχείου στη θέση 11 ως εξής

```
>> disp( Z(11) )
     0
```

Αφού επιβεβαιώσουμε ότι το μηδέν βρίσκεται στη θέση 11, τοποθετούμε όλα τα επόμενα στοιχεία, δηλαδή αυτά που βρίσκονται στις θέσεις 12 έως το τέλος του Z, στο νέο διάνυσμα Zpos.

```
>> Zpos = Z(12:end)

Zpos =

     1     2     3     4     5     6     7     8     9    10
```

Παράδειγμα 8.3

Ο όρος «κινητός μέσος όρος» ή «κινούμενος μέσος» (moving average) μιας ακολουθίας δεδομένων αφορά τον προσδιορισμό της ακολουθίας των μέσων όρων των τιμών διαδοχικών διαστημάτων, μήκους m παρατηρήσεων, της ακολουθίας αυτής. Σημειώνεται ότι ο κινητός μέσος όρος χρησιμοποιείται συνήθως με δεδομένα χρονοσειρών για την εξομάλυνση των βραχυπρόθεσμων διακυμάνσεων και την επισήμανση μακροπρόθεσμων τάσεων ή κύκλων.

Για παράδειγμα, ας εξετάσουμε τις τιμές κλεισίματος μιας χρηματιστηριακής μετοχής κατά τη διάρκεια δέκα ημερών. Αν οι τιμές για τις ημέρες αυτές είναι

$$T_1 = 51, \quad T_2 = 52.2, \quad T_3 = 53.4, \quad T_4 = 51.1, \quad T_5 = 52.9,$$

$$T_6 = 53.2, \quad T_7 = 54, \quad T_8 = 54, \quad T_9 = 53.2, \quad T_{10} = 54$$

τότε η ακολουθία $y_t, t = 1, 2 \ldots, 10$ των κινητών μέσων όρων 5 ημερών, υπολογίζεται με βάση τις ακόλουθες σχέσεις

$$\begin{aligned}
y_1 &= T_1 \\
y_2 &= \frac{T_1 + T_2}{2} \\
y_3 &= \frac{T_1 + T_2 + T_3}{3} \\
y_4 &= \frac{T_1 + T_2 + T_3 + T_4}{4} \\
y_5 &= \frac{T_1 + T_2 + T_3 + T_4 + T_5}{5} \\
y_t &= \frac{T_{t-4} + T_{t-3} + T_{t-2} + T_{t-1} + T_t}{5}, t > 5
\end{aligned}$$

Παρατηρείστε ότι οι κινητοί μέσοι όροι κάθε χρονική στιγμή υπολογίζονται από τη μέση τιμή των 5 προηγούμενων παρατηρήσεων, αν υπάρχουν αυτές, ή από τον μέσο όρο των διαθέσιμων παρατηρήσεων. Για τις παραπάνω τιμές έχουμε, παραδείγματος χάριν, ότι $´y_2 = 51.6$ και $y_6 = 52.56$.

Στο παράδειγμα αυτό ζητείται η υλοποίηση μιας παρόμοιας με την παραπάνω διαδικασία, μέσω της κατασκευής ενός script αρχείου για τον προσδιορισμό του κινούμενου μέσου όρου 5 ημερών μιας υποθετικής μετοχής, σε διάστημα 30 ημερών. Για τον έλεγχο του αρχείου δημιουργείστε ένα διάνυσμα με 30 τυχαίες τιμές από το 50 μέχρι και το 54, θεωρώντας ότι οι τιμές αυτές αντιστοιχούν στις τιμές της μετοχής σε ένα διάστημα 30 ημερών. Τέλος, αποτυπώστε στο ίδιο διάγραμμα τις παρατηρούμενες τιμές και τον μέσο όρο 5 ημερών της μετοχής.

Λύση Παραδείγματος 8.3

Αρχικοποίηση μεταβλητών: Για να αποθηκεύσουμε την κίνηση της μετοχής, θα χρειαστούμε ένα διάνυσμα με 30 θέσεις (π.χ. T). Το διάνυσμα αυτό θα πρέπει να περιέχει τυχαίες τιμές από το 50 μέχρι και το 54, τις οποίες θα δημιουργήσουμε με την εντολή `rand` (η `rand(m,n)` κατασκευάζει έναν πίνακα με μέγεθος $m \times n$, ο οποίος περιέχει τιμές από το μηδέν έως το ένα). Για να επιτύχουμε το επιθυμητό εύρος, θα χρησιμοποιήσουμε την εξής τεχνική:

- θα δημιουργήσουμε, αρχικά, ένα διάνυσμα-γραμμή 30 θέσεων (1×30), που θα έχει σε όλες τις θέσεις την τιμή 50 (δηλαδή `50*ones(1,30)`),
- στη συνέχεια, θα προσθέσουμε τυχαίες τιμές από το 0 έως το 4 (δηλαδή `4*rand(1,30)`).

Έπειτα, θα ορίσουμε το διάνυσμα του κινούμενου μέσου όρου (π.χ. ΚΜΟ). Οι τιμές του ΚΜΟ προσδιορίζονται ως ακολούθως:

- η πρώτη τιμή του ΚΜΟ θα είναι η πρώτη τιμή του T,
- η δεύτερη τιμή του ΚΜΟ θα είναι ο μέσος όρος των δύο πρώτων τιμών του T,
- η τρίτη τιμή του ΚΜΟ θα είναι ο μέσος όρος των τριών πρώτων τιμών του T,
- η τέταρτη τιμή του ΚΜΟ θα είναι ο μέσος όρος των τεσσάρων πρώτων τιμών του T,
- η καθεμία από τις επόμενες τιμές του T θα είναι ο μέσος όρος των προηγουμένων 5 παρατηρήσεων του T.

Αυτό σημαίνει ότι το διάνυσμα ΚΜΟ θα έχει συνολικά 30 θέσεις, δηλαδή μέγεθος 1×30. Μπορούμε να το αρχικοποιήσουμε με μηδενικά (`zeros`), καθώς οι τιμές του θα προκύψουν από τον υπολογισμό των μέσων όρων.
Τέλος, είναι σαφές ότι για την υλοποίηση της λύσης θα χρειαστεί μια επαναληπτική διαδικασία, οπότε θα χρειαστούμε και έναν μετρητή (π.χ. n).

Κύριος κώδικας: Για το κύριο πρόγραμμα θα χρειαστούμε μία επαναληπτική δομή, στην οποία ο μετρητής n θα πάρει τις τιμές από 1 έως 30, έτσι ώστε να υπολογιστούν όλοι οι κινούμενοι μέσοι όροι. Για τις πέντε πρώτες επαναλήψεις ο κινούμενος μέσος θα υπολογίζεται από τις διαθέσιμες μέχρι εκείνη τη στιγμή μετρήσεις. Στη συνέχεια, για τις υπόλοιπες επαναλήψεις θα υπολογίζεται το άθροισμα του πιο πρόσφατου πενθημέρου και θα διαιρείται διά 5, ώστε να βρεθεί ο κινητός μέσος όρος. Για να καταλάβουμε πώς γίνεται αυτό, έχουμε τα εξής:

- Για το πρώτο (πλήρες) πενθήμερο η τιμή y_5 του κινητού μέσου όρου βρίσκεται από την εντολή `sum(T(1:5))/5`.

  Δηλαδή, ανακαλούμε τα πέντε πρώτα στοιχεία του T μέσω της εντολής T(1:5) και τα αθροίζουμε χρησιμοποιώντας την εντολή `sum`.

- Για το δεύτερο πενθήμερο η τιμή y_6 του κινητού μέσου όρου βρίσκεται από την εντολή `sum(T(2:6))/5`.

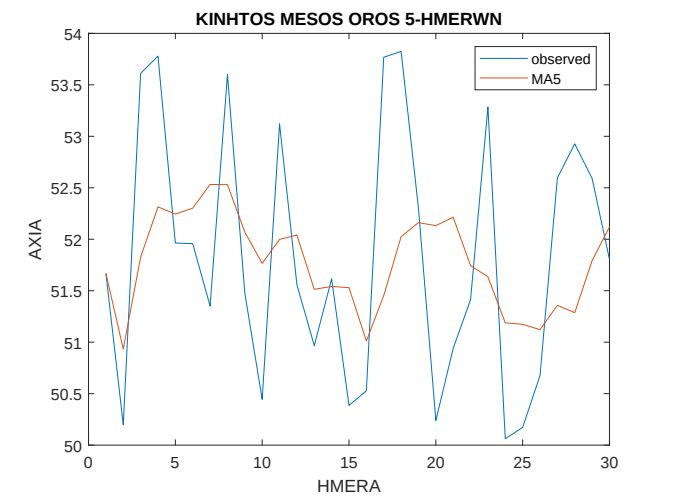
- Για το τρίτο πενθήμερο η τιμή y_7 του κινητού μέσου όρου βρίσκεται από την εντολή `sum(T(3:7))/5`.

- Γενικά, για το n-οστό πενθήμερο η τιμή y_n του κινητού μέσου όρου βρίσκεται από την εντολή `sum(T((n-5+1):n))/5`.

Έπειτα από 30 επαναλήψεις, θα έχει υπολογιστεί ο κινητός μέσος όρος των τιμών της μετοχής, οπότε μπορούμε να κατασκευάσουμε το αντίστοιχο διάγραμμα με την εντολή `plot`. Στο διάγραμμα αυτό μπορούμε με τη χρήση της εντολής `hold ON` να εισάγουμε και τις αρχικές τιμές του T, λαμβάνοντας το γράφημα του Σχήματος 8.3.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%KINHTOS MESOS OROS 5-HMERWN
% Initialization
clear all
T = 50*ones(1,30) + 4*rand(1,30);
KMO = zeros(1,30);

% Main code
for n = 1:30
    if n<5
        KMO(n) = sum( T(1:n) ) / n;
    else
        KMO(n) = sum( T((n-4):n) ) / 5;
    end
end

plot(T)
hold ON
plot(KMO)
legend('observed','MA5')
title('KINHTOS MESOS OROS 5-HMERWN')
xlabel('HMERA')
ylabel('AXIA')
hold OFF
```



Σχήμα 8.3: Το διάγραμμα που θα προκύψει από τη λύση του Παραδείγματος 8.3 θα έχει τη μορφή που φαίνεται στο σχήμα. Προφανώς, λόγω των τυχαίων τιμών, κάθε φορά που θα εκτελείται ο κώδικας θα προκύπτει διαφορετικό διάγραμμα.

Από το Σχήμα 8.3 είναι φανερή η εξομάλυνση των δεδομένων της χρονοσειράς που επιτυγχάνεται με τη χρήση του κινητού μέσου. Ο κινητός μέσος, επομένως, πετυχαίνει την εξομάλυνση των βραχυπρόθεσμων διακυμάνσεων των τιμών και φανερώνει, για αυτά τα δεδομένα, ότι δεν παρουσιάζουν κάποια μακροπρόθεσμη τάση.

Παρατήρηση 8.3

Στο Παράδειγμα 8.3 θα μπορούσαμε να παραλείψουμε την αρχικοποίηση του διανύσματος ΚΜΟ χωρίς να παρουσιαστεί πρόβλημα στην εκτέλεση του κώδικα. Αυτό συμβαίνει καθώς το μέγεθος του ΚΜΟ θα διαμορφωνόταν *μέσα* στην επαναληπτική διαδικασία. Με άλλα λόγια, σε κάθε επανάληψη θα προστίθετο ένα ακόμα στοιχείο στο ΚΜΟ, μέχρι να τερματιστεί ο βρόχος.
Αν και η τεχνική αυτή μας επιτρέπει να δημιουργούμε διανύσματα που διαμορφώνονται κατά την εκτέλεση μιας επαναληπτικής διαδικασίας, η χρήση της μπορεί να επιβραδύνει τον κώδικα και δεν συνιστάται.

Παράδειγμα 8.4

Σε πολλές εφαρμογές είναι χρήσιμη η υιοθέτηση λογαριθμικής κλίμακας στα διαγράμματα για την αποτύπωση των φαινομένων προκειμένου να είναι ευδιάκριτες οι μεταβολές στα διάφορα μεγέθη. Για παράδειγμα, η κλίμακα Richter για τη μέτρηση της έντασης των σεισμών, τα Decibel στη μέτρηση της έντασης του ήχου και η εντροπία στη θερμοδυναμική είναι μερικά παραδείγματα μεγεθών στα οποία χρησιμοποιείται η λογαριθμική κλίμακα. Το χαρακτηριστικό της λογαριθμικής κλίμακας είναι ότι οι τιμές αυξάνονται εκθετικά και όχι γραμμικά. Για παράδειγμα, οι τιμές 20 και 30 έχουν διαφορετική απόσταση πάνω στη λογαριθμική κλίμακα από τις τιμές 80 και 90. Αντίθετα, οι αποστάσεις μεταξύ των τιμών 1, 10, 100 και 1000 είναι ίδιες.
Κατασκευάστε ένα διάνυσμα το οποίο να έχει σχεδόν ισαπέχουσες τιμές από το 0.01 έως το 10^5, σε λογαριθμική κλίμακα, και φτιάξτε τη γραφική παράσταση της $y = \log_{10}(x + 3)$. Για να είναι σχεδόν ίσες οι αποστάσεις μεταξύ των τιμών, κατασκευάστε το διάνυσμα ώστε να περιέχει τις τιμές $0.01,\ 0.02,\ 0.04,\ 0.07, 0.1, 0.2, 0.4, 0.7, 1, 2, 4, 7, 10, 20, \ldots, 10^5$. Για την κατασκευή της γραφικής παράστασης χρησιμοποιείστε διάγραμμα με λογαριθμικούς άξονες με την εντολή `loglog`. (Η εντολή `loglog(x,y)` κατασκευάζει το διάγραμμα (x-y) σε λογαριθμικούς άξονες x και y).

Λύση Παραδείγματος 8.4

Αρχικοποίηση μεταβλητών: Στο παράδειγμα αυτό, η αρχικοποίηση του διανύσματος με τη λογαριθμική κλίμακα μπορεί να γίνει με εισαγωγή στοιχείων. Ας ονομάσουμε το διάνυσμα xlog και ας το αρχικοποιήσουμε με μηδενικά. Το διάνυσμα αυτό θα έχει $4 \times 7 + 1 = 29$ θέσεις, καθώς θα ξεκινάει από το 10^{-2} και θα φθάνει μέχρι και το 10^5. Μπορούμε να εισάγουμε αρχικά τις πρώτες τέσσερις τιμές, δηλαδή τις 0.01, 0.02, 0.04 και 0.07, με απευθείας ορισμό `xlog(1:4) = [0.01, 0.02, 0.04, 0.07]`.
Οι επόμενες τέσσερις τιμές, δηλαδή αυτές που θα αποθηκευτούν στις θέσεις 5 έως 8, θα είναι αυτές που είδαμε προηγουμένως, πολλαπλασιασμένες επί 10. Άρα, μπορούμε να τις εισάγουμε στο διάνυσμα xlog χρησιμοποιώντας τις τέσσερις προηγούμενες τιμές, δηλαδή με την εντολή `xlog(5:8)=10*xlog(1:4)`.
Αυτό που γίνεται, λοιπόν, στην παραπάνω εντολή, είναι ότι οι τιμές που υπάρχουν στις θέσεις 1 έως 4 του διανύσματος xlog πολλαπλασιάζονται επί δέκα και τοποθετούνται στις επόμενες τέσσερις θέσεις, δηλαδή στις θέσεις 5 έως 8. Η διαδικασία συνεχίζεται με αντίστοιχο τρόπο, μέχρι να φτάσουμε στην τετράδα 10^4, 2×10^4, 4×10^4 και 7×10^4. Το τελευταίο στοιχείο είναι το 10^5 και πρέπει να εισαχθεί στην 29η θέση του xlog με απευθείας ορισμό, δηλαδή ως `xlog(29)=1e5`.

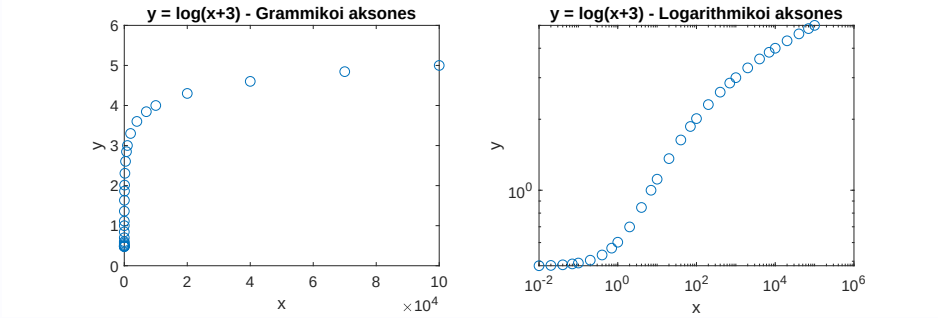
Αφού ολοκληρώσουμε το διάνυσμα xlog με τη λογαριθμική κλίμακα, μπορούμε να υπολογίσουμε το y, το οποίο θα έχει τις τιμές $y = \log_{10}(x + 3)$.

Κύριος κώδικας: Για το κύριο πρόγραμμα θα χρησιμοποιήσουμε την εντολή `loglog(x,y)`, σύμφωνα με την εκφώνηση της άσκησης, ώστε να κατασκευάσουμε τη γραφική παράσταση σε λογαριθμικούς άξονες. Η χρήση λογαριθμικών αξόνων μας επιτρέπει να παραστήσουμε ευδιάκριτα τα σημεία, στην περίπτωση που έχουμε λογαριθμικές συναρτήσεις. Αυτό μπορεί να γίνει κατανοητό από τα διαγράμματα του Σχήματος 8.4, τα οποία απεικονίζουν τη γραφική παράσταση της $y = \log_{10}(x + 3)$ σε γραμμικούς και σε λογαριθμικούς άξονες. Παρατηρούμε ότι το δεύτερο διάγραμμα είναι πολύ πιο ευανάγνωστο από το πρώτο.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%GRAFIKI PARASTASI THS y=log(x+3)
% Initialization
clear all
format short

xlog = zeros(1,29);

xlog(1:4) = [0.01 0.02 0.04 0.07];
xlog(5:8) = 10*xlog(1:4);
xlog(9:12) = 10*xlog(5:8);
xlog(13:16) = 10*xlog(9:12);
xlog(17:20) = 10*xlog(13:16);
xlog(21:24) = 10*xlog(17:20);
xlog(25:28) = 10*xlog(21:24);
xlog(29) = 10^5;

y = log10(xlog+3);

% Main code
subplot(1,2,1)
plot(xlog,y,'o')
title('y = log(x+3) - Grammikoi aksones')
xlabel('x')
ylabel('y')

subplot(1,2,2)
loglog(xlog,y,'o')
title('y = log(x+3) - Logarithmikoi aksones')
xlabel('x')
ylabel('y')
```

Από τις γραφικές παραστάσεις του Σχήματος 8.4 είναι εμφανές ότι το δεύτερο διάγραμμα (διάγραμμα σε λογαριθμικούς άξονες) είναι πολύ πιο ευανάγνωστο από το πρώτο (διάγραμμα σε γραμμικούς άξονες), ειδικά για μικρές τιμές της μεταβλητής x.



Σχήμα 8.4: Τα δύο διαγράμματα απεικονίζουν τη γραφική παράσταση της $y = \log_{10}(x + 3)$ σε γραμμικούς (αριστερό γράφημα) και σε λογαριθμικούς άξονες (δεξί γράφημα).

Άσκηση αυτοαξιολόγησης 8.2

Στη λύση του Παραδείγματος 8.4 ορίσαμε τις τιμές του διανύσματος xlog με έναν «άκομψο» τρόπο, καθώς επαναλάβαμε την ίδια εντολή πολλές φορές, με διαφορετικά νούμερα. Δοκιμάστε να βελτιώσετε τον κώδικα εισάγοντας μια επαναληπτική διαδικασία αντί για τις εντολές των γραμμών 7 έως 12. Είναι σαφές ότι υπάρχουν πολλοί διαφορετικοί τρόποι να λύσετε το πρόβλημα αυτό!

Παρατήρηση 8.4

Στο Παράδειγμα 8.4 εφαρμόσαμε μία διαδικασία προκειμένου να έχουμε σχεδόν ισαπέχοντα διαστήματα σε λογαριθμική κλίμακα. Στο Matlab υπάρχει ειδική εντολή που εκτελεί αυτή τη λειτουργία και είναι η `logspace(a,b,num)`. Λειτουργεί με αντίστοιχα ορίσματα με τη `linspace` και κατασκευάζει ένα διάνυσμα με num ισαπέχοντα σημεία από το 10^a έως το 10^b, σε λογαριθμική κλίμακα. Για να επιτύχουμε παρόμοιο αποτέλεσμα με αυτό του Παραδείγματος 8.4, θα μπορούσαμε να γράψουμε

```
>> xlog = logspace(-2,5,29);
```

Παρατήρηση 8.5

Είδαμε στα Παραδείγματα 8.2 και 8.4 ότι μπορούμε να αναθέσουμε ένα τμήμα ενός διανύσματος σε τμήμα του ίδιου ή άλλου διανύσματος, είτε χρησιμοποιώντας τον τελεστή ":" είτε ορίζοντας συγκεκριμένες θέσεις. Πρέπει, όμως, να είμαστε πολύ προσεκτικοί ώστε τα δύο τμήματα να έχουν το ίδιο μέγεθος. Δεν μπορούμε, δηλαδή, να αναθέσουμε ένα διάνυσμα (ή τμήμα διανύσματος) με πέντε στοιχεία σε ένα τμήμα διανύσματος που έχει είτε λιγότερα είτε περισσότερα στοιχεία. Ο ελεγκτής του Matlab θα εμφανίσει σφάλμα, όπως φαίνεται στο παρακάτω απόσπασμα κώδικα.

```
>> A(1:4)=[5,3,3,4,5]
Unable to perform assignment because the left and right sides have a
   different number of elements.
```

Επίσης, ένα στοιχείο που πρέπει να προσέχουμε, όταν ανακαλούμε τα στοιχεία ενός διανύσματος, είναι να μην υπερβαίνουμε το μέγεθος του διανύσματος αυτού. Για παράδειγμα, αν ορίσουμε ένα διάνυσμα με 10 στοιχεία και, στη συνέχεια, προσπαθήσουμε να ανακαλέσουμε το στοιχείο 11 (το οποίο δεν υπάρχει), ο ελεγκτής του Matlab θα εμφανίσει σφάλμα. Ένα χαρακτηριστικό παράδειγμα μιας τέτοιας περίπτωσης παρουσιάζεται στον παρακάτω κώδικα.

```
>> A = zeros(1,10);
>> A(11)
Index exceeds array bounds.
```

Το μήνυμα αυτό μας ειδοποιεί ότι κάποιος δείκτης, στην προκειμένη περίπτωση ο δείκτης 11, έχει υπερβεί τη διάσταση, δηλαδή το πλήθος των στοιχείων του διανύσματος.

8.3 Πληροφορίες για το μέγεθος διανύσματος και τα στοιχεία του - Εντολές length, sum, max/min και find

Σε πολλές εφαρμογές είναι εξαιρετικά χρήσιμο να γνωρίζουμε το μήκος ενός διανύσματος, δηλαδή το πλήθος των στοιχείων του. Επίσης, πολύ συχνά χρειάζεται να έχουμε πληροφορίες για τα στοιχεία ενός διανύσματος, όπως τη μέγιστη/ελάχιστη τιμή, τις θέσεις στις οποίες εμφανίζονται, πού υπάρχουν μηδενικά, μιγαδικοί αριθμοί, τιμές NaN κ.ο.κ. Στις επόμενες παραγράφους, θα μιλήσουμε για εντολές που εκτελούν αυτές τις λειτουργίες.

8.3.1 Μήκος διανύσματος - Εντολή length

Η εντολή `length` επιστρέφει το πλήθος των στοιχείων ενός διανύσματος, ανεξάρτητα από το αν είναι γραμμή ή στήλη. Για παράδειγμα,

```
>> A = zeros(1,10);
>> length(A)

ans =

    10
```

8.3.2 Άθροισμα στοιχείων διανύσματος - Εντολή sum

Η εντολή `sum` επιστρέφει το άθροισμα των στοιχείων ενός διανύσματος. Για παράδειγμα,

```
>> A = 1:5;
>> sum(A)

ans =

    15
```

8.3.3 Μέγιστη/ελάχιστη τιμή διανύσματος - Εντολές max και min

Η εντολές `max` και `min` έχουν διαφορετικές λειτουργίες, ανάλογα με το όρισμα που εισάγουμε. Όταν το όρισμα είναι ένα διάνυσμα, τότε η εντολές `max` και `min` επιστρέφουν τη μέγιστη και ελάχιστη τιμή, αντίστοιχα, των στοιχείων του διανύσματος.

Για παράδειγμα,

```
>> A = [6,5,12,2,-3,4];
>> max(A)
```

```
ans =

    12

>> min(A)

ans =

    -3
```

Πολλές φορές μας ενδιαφέρει να μάθουμε, εκτός από τη μέγιστη τιμή, *και* τη θέση μέσα στο διάνυσμα, στην οποία εμφανίζεται. Αν, λοιπόν, στο παραπάνω παράδειγμά μας θέλουμε να βρούμε και το *πού* εμφανίζονται η μέγιστη/ελάχιστη τιμή, τότε πρέπει να εκχωρήσουμε το αποτέλεσμα της εντολής `max/min` σε ένα διάνυσμα με δύο θέσεις. Στην πρώτη αποθηκεύεται η τιμή και στη δεύτερη η θέση της.

```
>> A = [6,5,12,2,-3,4 ];
>> [xMax,iMax] = max(A)

xMax =

    12


iMax =

     3
```

Η παραπάνω εντολή μας πληροφορεί ότι η μέγιστη τιμή είναι 12 και βρίσκεται στην τρίτη θέση του διανύσματος. Με αντίστοιχο τρόπο, μπορούμε να βρούμε την ελάχιστη τιμή και τη θέση της στο διάνυσμα.

8.3.4 Εντολή find

Είδαμε στην προηγούμενη παράγραφο ότι μπορούμε να βρούμε τη θέση μέσα στο διάνυσμα, στην οποία εμφανίζεται μία χαρακτηριστική τιμή (π.χ. μέγιστη ή ελάχιστη). Με την εντολή `find` μπορούμε να βρούμε όλες τις θέσεις ενός διανύσματος, στις οποίες ισχύει μία συγκεκριμένη συνθήκη. Για δεδομένο διάνυσμα A η εντολή find(A) επιστρέφει τις θέσεις που έχουν μη-μηδενικές τιμές.

```
>> A = [0,5,0,0,-3,4];
>> find(A)

ans =

     2     5     6
```

Στη γενικότερη περίπτωση, όμως, μπορούμε να εισάγουμε οποιονδήποτε έλεγχο επιθυμούμε. Για παράδειγμα, αν θέλουμε να βρούμε σε ποιες θέσεις του A υπάρχουν μηδενικές τιμές, μπορούμε να γράψουμε

```
>> z = find(A==0)

z =

     1     3     4
```

Το διάνυσμα z έχει αποθηκευμένες τις θέσεις του A που έχουν την τιμή μηδέν.

Μπορούμε, επίσης, να μεταβάλλουμε τα στοιχεία στις θέσεις αυτές, θέτοντάς τα ίσα με μία επιθυμητή τιμή. Αν, για παράδειγμα, θέλουμε να αποφύγουμε τα μηδενικά, τα οποία σε μία πιθανή διαίρεση θα μας δώσουν αποτέλεσμα Inf ή NaN, μπορούμε να επέμβουμε στις συγκεκριμένες θέσεις και να θέσουμε μία μη-μηδενική τιμή, π.χ. 0.0001. Αυτό γίνεται ως εξής

```
>> A(z) = 0.0001

A =

    0.0001     5     0.0001     0.0001    -3     4
```

Η εντολή A(z) συμβολίζει τις θέσεις A(1), A(3) και A(4), αφού 1,3 και 4 είναι οι τιμές του z. Θα μπορούσαμε να επιτύχουμε το ίδιο αποτέλεσμα, χωρίς να χρησιμοποιήσουμε το «αποθηκευτικό» διάνυσμα z, χρησιμοποιώντας την εντολή `find` ως όρισμα του διανύσματος, δηλαδή

```
>> A = [0,5,0,0,-3,4];
>> A( find(A==0) ) = 0.0001

A =

    0.0001     5     0.0001     0.0001    -3     4
```

Η παραπάνω εντολή μπορεί να περιγραφτεί με λόγια ως εξής: στις θέσεις του διανύσματος A, στις οποίες θα βρεις μηδενικές τιμές, τοποθέτησε την τιμή 0.0001. Αν και είναι σύνθετη στη σύνταξή της, εκτελεί μία σύνθετη λειτουργία με μία μόνο γραμμή κώδικα.

Τέλος, μία χρήσιμη εφαρμογή της εντολής `find` προκύπτει όταν συνδυάζεται με τον έλεγχο isempty, για να διερευνήσει την ύπαρξη ενός στοιχείου σε ένα διάνυσμα. Για παράδειγμα, μπορούμε να επιβεβαιώσουμε ότι ο παραπάνω πίνακας A δεν έχει στοιχεία μεγαλύτερα του 10 με την εντολή

```
>> isempty( find(A>10) )

ans =

  logical

   1
```

Το αποτέλεσμα μάς πληροφορεί ότι η πρότασή μας ήταν αληθής, δηλαδή ότι το αποτέλεσμα της εντολής `find` είναι το κενό σύνολο.

Παράδειγμα 8.5

Ένα θερμόμετρο μπορεί να λαμβάνει μετρήσεις με ικανοποιητική ακρίβεια, μόνο όταν αυτές βρίσκονται μεταξύ των -10 και 40 βαθμών Κελσίου. Αν υποθέσουμε ότι έχουμε μία δειγματοληψία με 15 μετρήσεις, οι οποίες είναι αποθηκευμένες στο διάνυσμα Temp, βρείτε πόσες τιμές βρίσκονται εκτός των προαναφερθέντων ορίων και, στη συνέχεια, γράψτε μία εντολή η οποία να τις διαγράφει από το διάνυσμα. (Για τις ανάγκες της άσκησης, κατασκευάστε το διάνυσμα Temp, τοποθετώντας ακέραιες ψευδοτυχαίες τιμές από -20 έως 50, χρησιμοποιώντας την εντολή `randi`).

Λύση Παραδείγματος 8.5

Θα ξεκινήσουμε με το διάνυσμα Temp, το οποίο, σύμφωνα με την εκφώνηση της άσκησης, θα έχει 15 τιμές μεταξύ του -20 και του 50. Θα το δημιουργήσουμε χρησιμοποιώντας την εντολή `randi`

```
>> Temp = randi([-20,50], 1 , 15)

Temp =

    42    22    -2     5    43    43    11    -1    19    -8    41    -4
    -9   -19     2
```

Στη συνέχεια, θα εντοπίσουμε τις θέσεις των τιμών που είναι μικρότερες από -10 και τις θέσεις των τιμών που είναι μεγαλύτερες από 40 και θα τις τοποθετήσουμε όλες μαζί σε ένα διάνυσμα (π.χ. positions). Για να το επιτύχουμε αυτό, θα χρησιμοποιήσουμε την εντολή `find`

```
>> positions = [find(Temp < -10) , find(Temp > 40)]

positions =

    14     1     5     6    11
```

Η παραπάνω εντολή τοποθέτησε στο διάνυσμα positions τις θέσεις στις οποίες εμφανίζονται τιμές εκτός των ορίων [-10,40]. Μπορούμε να ελέγξουμε το αποτέλεσμα, παρατηρώντας τις τιμές του Temp, οι οποίες φαίνονται παραπάνω. Συγκεκριμένα, έχουμε τιμή εκτός ορίων στη θέση 14 (η τιμή είναι -19), στη θέση 1 (η τιμή είναι 42) κ.ο.κ. Για να βρούμε πόσες τιμές είναι εκτός ορίων, θα χρησιμοποιήσουμε την εντολή `length`.

```
>> length(positions)

ans =

     5
```

Τις πέντε αυτές τιμές μπορούμε να τις αφαιρέσουμε από το διάγραμμα των μετρήσεων Temp, χρησιμοποιώντας το κενό σύνολο “[ ]”.

```
>> Temp(positions) = [ ]

Temp =

    22    -2     5    11    -1    19    -8    -4    -9     2
```

Παρατηρούμε ότι το αποτέλεσμα έχει μόνο τις 10 τιμές του αρχικού διανύσματος, που ήταν εντός των ορίων.

Παρατήρηση 8.6

Θα πρέπει να τονίσουμε ότι η εντολή `find`, όταν εφαρμόζεται σε ένα διάνυσμα, επιστρέφει τις θέσεις που ικανοποιούν τη συνθήκη και όχι τις τιμές αυτές καθαυτές. Έτσι, στο Παράδειγμα 8.5 το διάνυσμα positions περιείχε τις θέσεις στις οποίες οι τιμές ήταν εκτός ορίων και όχι τις ίδιες τις τιμές. Αν θέλαμε να εμφανίσουμε τις τιμές, θα έπρεπε να εισαγάγουμε τις θέσεις (positions) ως όρισμα στο διάνυσμα Temp (προφανώς προτού τις αφαιρέσουμε) ως εξής:

```
>> Temp(positions)

ans =

   -19    42    43    43    41
```

8.3.5 Πράξεις μεταξύ διανυσμάτων

Οι πράξεις που θα μας απασχολήσουν στην παράγραφο αυτή είναι η πρόσθεση/αφαίρεση διανυσμάτων, ο πολλαπλασιασμός στοιχείο προς στοιχείο (γινόμενο Hadamard), η αναστροφή διανύσματος και το εσωτερικό γινόμενο. Μερικές από αυτές τις πράξεις έχουν ήδη αναφερθεί στο Κεφάλαιο 1, ωστόσο θα τις επαναλάβουμε εν συντομία, χάριν πληρότητας.

8.3.5.1 Πρόσθεση/αφαίρεση διανυσμάτων και πολλαπλασιασμός στοιχείο προς στοιχείο

Για να γίνουν οι πράξεις της πρόσθεσης, αφαίρεσης ή πολλαπλασιασμού στοιχείο προς στοιχείο μεταξύ δύο διανυσμάτων, θα πρέπει τα διανύσματα αυτά να έχουν το ίδιο μέγεθος. Ιδιαίτερη προσοχή πρέπει να δοθεί στο σύμβολο του πολλαπλασιασμού στοιχείο προς στοιχείο, το οποίο είναι ".*", και όχι ο απλός αστερίσκος ο οποίος στα διανύσματα χρησιμοποιείται για το εσωτερικό γινόμενο, όπως θα δούμε σε επόμενη παράγραφο. Μερικά παραδείγματα πράξεων παρουσιάζονται παρακάτω

- Πρόσθεση διανυσμάτων

```
>> arrayA = [1,2,3,4,5]
>> arrayB = arrayA + 2*arrayA

arrayB =

     3     6     9    12    15
```

Παρατηρούμε ότι αν πολλαπλασιάσουμε ένα διάνυσμα με έναν αριθμό, τότε πολλαπλασιάζεται κάθε στοιχείο του διανύσματος με τον αριθμό αυτό.

- Πολλαπλασιασμός στοιχείο προς στοιχείο

```
>> arrayCol = [2;4;6]
>> arrayCol2 = arrayCol .* arrayCol

arrayCol2 =
```

```
     4
    16
    36
```

Η αντίστροφη πράξη του πολλαπλασιασμού στοιχείο προς στοιχείο είναι η διαίρεση στοιχείο προς στοιχείο, η οποία εκτελείται με το σύμβολο "./". Τέλος, η ύψωση όλων των στοιχείων ενός διανύσματος σε μία δύναμη μπορεί να εκτελεστεί και με το σύμβολο ".^".

8.3.5.2 Ανάστροφο διάνυσμα

Έχουμε αναφέρει ότι ένα διάνυσμα μπορεί να θεωρηθεί ως ένας μονοδιάστατος πίνακας. Είναι, επίσης, γνωστό από τη Γραμμική Άλγεβρα πινάκων ότι ανάστροφος ενός $m \times n$ πίνακα Α είναι ένας $n \times m$ πίνακας Β, του οποίου κάθε στοιχείο b_{ij} είναι το a_{ji} στοιχείο του Α. Με άλλα λόγια, ο ανάστροφος ενός πίνακα είναι αυτός στον οποίον οι γραμμές έχουν γίνει στήλες και οι στήλες έχουν γίνει γραμμές.

Με βάση τα παραπάνω, το ανάστροφο διάνυσμα ενός διανύσματος-γραμμής με μέγεθος $1 \times n$ είναι ένα διάνυσμα-στήλη με μέγεθος $n \times 1$. Αντίστοιχα, το ανάστροφο διάνυσμα ενός διανύσματος-στήλης $m \times 1$ είναι ένα διάνυσμα-γραμμή $1 \times m$. Αυτό στο Matlab γίνεται με τη χρήση του συμβόλου της αποστρόφου " ' ". Για παράδειγμα,

```
>> arrayRow = [1,2,3,4,5];
>> arrayCol = arrayRow'

arrayCol =

     1
     2
     3
     4
     5
```

Παρατήρηση 8.7

Στο σημείο αυτό, πρέπει να παρατηρήσουμε ότι η χρήση του ανάστροφου διανύσματος μας δίνει τη δυνατότητα να κατασκευάσουμε ένα διάνυσμα-στήλη με συγκεκριμένο μοτίβο είτε με χρήση του τελεστή ":" είτε με την εντολή `linspace`. Συγκεκριμένα, όπως είδαμε στις προηγούμενες παραγράφους, οι παραπάνω τρόποι κατασκευής διανυσμάτων δίνουν μόνο διανύσματα-γραμμή. Για να κατασκευάσουμε, λοιπόν, το επιθυμητό διάνυσμα-στήλη, κατασκευάζουμε το αντίστοιχο διάνυσμα-γραμμή και, στη συνέχεια, βρίσκουμε το ανάστροφό του.

8.3.5.3 Εσωτερικό γινόμενο διανυσμάτων

Η πράξη του εσωτερικού γινομένου διανυσμάτων είναι βασικό στοιχείο της διανυσματικής ανάλυσης. Για να την περιγράψουμε εν συντομία, θα θεωρήσουμε δύο διανύσματα $\boldsymbol{a}$ και $\boldsymbol{b}$ στον τριδιάστατο χώρο, τα οποία είναι για παράδειγμα τα

$$\boldsymbol{a} = a_1\hat{\boldsymbol{i}} + a_2\hat{\boldsymbol{j}} + a_3\hat{\boldsymbol{k}}$$

και

$$\boldsymbol{b} = b_1\hat{\boldsymbol{i}} + b_2\hat{\boldsymbol{j}} + b_3\hat{\boldsymbol{k}},$$

όπου $\hat{\boldsymbol{i}}, \hat{\boldsymbol{j}}$ και $\hat{\boldsymbol{k}}$ είναι τα μοναδιαία διανύσματα στο καρτεσιανό σύστημα συντεταγμένων. Το εσωτερικό γινόμενο $\boldsymbol{a} \cdot \boldsymbol{b}$ είναι η τιμή

$$\boldsymbol{a} \cdot \boldsymbol{b} = a_1b_1 + a_2b_2 + a_3b_3.$$

Στο Matlab η πράξη του εσωτερικού γινομένου μπορεί να γίνει μεταξύ ενός διανύσματος-γραμμής κι ενός διανύσματος-στήλης με ίσο αριθμό στοιχείων, χρησιμοποιώντας το σύμβολο του πολλαπλασιασμού "*". Ας δούμε ένα παράδειγμα:

```
>> a=[2,4,6];
>> b=[3;1;2];
>> a * b

ans =

    22
```

Γενικότερα, η πράξη του εσωτερικού γινομένου μπορεί να γίνει μεταξύ ενός διανύσματος-γραμμής με μέγεθος $1 \times m$ κι ενός διανύσματος-στήλης με μέγεθος $m \times 1$, ανεξάρτητα από την τιμή του m. Για να είναι η πράξη εσωτερικό γινόμενο και να προκύψει μία τιμή ως αποτέλεσμα, πρέπει απαραίτητα ο πολλαπλασιασμός να γίνει με τη συγκεκριμένη σειρά, δηλαδή γραμμή επί στήλη. Σε αντίθετη περίπτωση, το αποτέλεσμα είναι πίνακας και όχι τιμή, όπως θα δούμε στην επόμενη παράγραφο.

Για την εκτέλεση του εσωτερικού γινομένου, είναι χρήσιμη η λειτουργία της αναστροφής διανύσματος, καθώς μας επιτρέπει να βρούμε το εσωτερικό γινόμενο χρησιμοποιώντας δύο διανύσματα-γραμμή, όπως φαίνεται στο παρακάτω παράδειγμα:

```
>> a = ones(1,30);
>> dotProduct = a * a'

dotProduct =

    30
```

Παράδειγμα 8.6

Ορίστε ένα διάνυσμα-γραμμή A που να περιέχει τους ακέραιους από το -100 έως το 100. Στη συνέχεια, ορίστε ένα νέο διάνυσμα B με τις τιμές του A σε απόλυτη τιμή. Τέλος, υπολογίστε το εσωτερικό γινόμενο μεταξύ του A και του ανάστροφου διανύσματος του B. Πώς ερμηνεύετε το αποτέλεσμα;

Λύση Παραδείγματος 8.6

Για να κατασκευάσουμε το διάνυσμα A, θα χρησιμοποιήσουμε τον τελεστή ":" ως εξής:

```
>> A = -100:100;
```

Στη συνέχεια, θα δημιουργήσουμε το διάνυσμα B με τις απόλυτες τιμές του διανύσματος A:

```
>> B = abs(A);
```

Τέλος θα εφαρμόσουμε την εντολή του εσωτερικού γινομένου:

```
>> dotProduct = A*B'
```

```
dotProduct =

     0
```

Παρατηρούμε ότι το αποτέλεσμα είναι μηδέν, παρόλο που ούτε το A ούτε το B είναι μηδενικά διανύσματα. Αυτό συμβαίνει διότι τα δύο διανύσματα είναι «ορθογώνια» μεταξύ τους. Η έννοια των ορθογώνιων διανυσμάτων μπορεί να γίνει αντιληπτή στις δύο και στις τρεις διαστάσεις, ωστόσο επεκτείνεται και σε περισσότερες διαστάσεις, παρόλο που δεν μπορούμε να αντιληφθούμε γεωμετρικά τα διανύσματα αυτά.

8.4 Διδιάστατοι πίνακες

Οι διδιάστατοι πίνακες αποτελούν ένα πολύ χρήσιμο εργαλείο των μαθηματικών και βρίσκουν εφαρμογές σε πολλά επιστημονικά πεδία. Μερικές από τις χρήσεις τους είναι στη μηχανική για τη μελέτη της κίνησης ρευστών και στερεών σωμάτων, στη θεωρία πιθανοτήτων και τη στατιστική, όπου οι στοχαστικοί πίνακες χρησιμοποιούνται για να περιγράψουν σύνολα πιθανοτήτων, σε γραφικά ηλεκτρονικών υπολογιστών, όπου χρησιμοποιούνται για το σχέδιο εικόνας τριών διαστάσεων κ.λπ. Ιδιαίτερα για το Matlab, οι πίνακες αποτελούν κομβική έννοια, κάτι που αντικατοπτρίζεται και στο ίδιο το όνομα της γλώσσας - το όνομα Matlab προέρχεται από τις λέξεις MATrix και LABoratory. Όλες οι μεταβλητές στο Matlab ορίζονται ως διδιάστατοι πίνακες με κατάλληλο μέγεθος. Όπως έχουμε δει, ένα διάνυσμα-γραμμή έχει μέγεθος $1 \times n$, ένα διάνυσμα-στήλη είναι $n \times 1$, ενώ ακόμα και οι απλοί αριθμοί ορίζονται στο Matlab ως πίνακες με μέγεθος 1×1.

```
>> a=1;
>> whos
  Name        Size              Bytes  Class     Attributes

  a           1x1                   8  double
```

Στις επόμενες παραγράφους θα παρουσιάσουμε τρόπους κατασκευής ενός διδιάστατου πίνακα, θα εξετάσουμε τις τεχνικές με τις οποίες μπορούμε να προσπελάσουμε τα στοιχεία του και θα αναφερθούμε σε πράξεις μεταξύ πινάκων και χρήσιμες εντολές.

8.4.1 Κατασκευή πινάκων

Η κατασκευή ενός πίνακα μπορεί να γίνει με διάφορους τρόπους, όπως με απευθείας εισαγωγή των στοιχείων, με χρήση του τελεστή ":", με πράξεις και συνενώσεις μεταξύ άλλων πινάκων ή/και διανυσμάτων και με εντολές του Matlab, οι οποίες κατασκευάζουν ειδικούς τύπους πινάκων. Ας δούμε τους βασικούς τρόπους κατασκευής ενός πίνακα.

8.4.1.1 Κατασκευή πίνακα με εισαγωγή στοιχείων

Είδαμε στις προηγούμενες ενότητες ότι μπορούμε να κατασκευάσουμε δύο ειδών διανύσματα: τα διανύσματα-γραμμή και τα διανύσματα-στήλη. Τώρα, με τον συνδυασμό αυτών μπορούμε εύκολα να κατασκευάσουμε έναν πίνακα δύο διαστάσεων, χωρίζοντας τα στοιχεία κάθε γραμμής με κενά ή "," και εισάγοντας τον τελεστή ";" κάθε φορά που θέλουμε να δημιουργήσουμε νέα γραμμή. Ας δούμε ένα απλό παράδειγμα:

```
>> mtrx1 = [1 2 3 4 5; 6 7 8 9 10]

mtrx1 =

     1     2     3     4     5
     6     7     8     9    10
```

Μπορούμε επίσης να χρησιμοποιήσουμε τη δομή «αρχή, βήμα, τέλος» για να ορίσουμε στοιχεία πινάκων. Για τον πίνακα του προηγούμενου παραδείγματος θα γράφαμε:

```
>> mtrx1 = [1:5 ; 6:10]

mtrx1 =

     1     2     3     4     5
     6     7     8     9    10
```

8.4.1.2 Οι εντολές zeros και ones

Αν θέλουμε να κατασκευάσουμε έναν οποιονδήποτε πίνακα, οι τιμές του οποίου να είναι αρχικοποιημένες με μηδενικά, τότε χρησιμοποιούμε την εντολή `zeros`. Η εντολή συντάσσεται δεχόμενη ένα ή δύο ορίσματα, αναλόγως του αν θέλουμε ο πίνακας να είναι τετραγωνικός. Με άλλα λόγια, η εντολή `zeros(n)` επιστρέφει έναν μηδενικό τετραγωνικό $n \times n$ πίνακα, ενώ η εντολή `zeros(m,n)` επιστρέφει έναν μηδενικό $m \times n$ πίνακα. Επίσης, χρησιμοποιώντας την εντολή `zeros(1,n)` μπορούμε να κατασκευάσουμε ένα μηδενικό διάνυσμα-γραμμή, ενώ με την `zeros(n,1)` ένα μηδενικό διάνυσμα-στήλη, όπως είδαμε και στην προηγούμενη ενότητα. Γενικά, μπορούμε να πούμε ότι υπάρχουν δύο διακριτές περιπτώσεις:

- η εντολή `zeros(n)`: παράγει έναν $n \times n$ πίνακα με όλα του τα στοιχεία μηδενικά,
- η εντολή `zeros(m,n)`: παράγει έναν $m \times n$ πίνακα με όλα του τα στοιχεία μηδενικά.

```
>> zeros(3)

ans =

     0     0     0
     0     0     0
     0     0     0
```

Με την ίδια ακριβώς λογική, και χρησιμοποιώντας την εντολή `ones`, μπορούμε να παράξουμε έναν πίνακα με όλα του τα στοιχεία να είναι 1.

- η εντολή `ones(n)` παράγει έναν $n \times n$ πίνακα με όλα του τα στοιχεία να είναι 1,
- η εντολή `ones(m,n)` παράγει έναν $m \times n$ πίνακα με όλα του τα στοιχεία να είναι 1.

8.4.1.3 Η εντολή eye - Ταυτοτικός πίνακας

Ταυτοτικός είναι ο πίνακας ο οποίος έχει μονάδες στην κύρια διαγώνιο και μηδενικά ως στοιχεία σε κάθε άλλη θέση του. Στο Matlab μπορούμε να δημιουργήσουμε ταυτοτικούς πίνακες με την εντολή `eye`. Όπως και στις προηγούμενες εντολές, η `eye` συντάσσεται δεχόμενη ένα ή δύο ορίσματα, αναλόγως του αν ο πίνακας θέλουμε να είναι τετραγωνικός. Για παράδειγμα:

```
>> eye(5)

ans =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

Ο ταυτοτικός πίνακας έχει ιδιαίτερη σημασία στη γραμμική άλγεβρα, καθώς, όταν πολλαπλασιαστεί με έναν άλλο τετραγωνικό πίνακα, τον αφήνει αμετάβλητο.Με άλλα λόγια, ο ταυτοτικός πίνακας είναι το αντίστοιχο του αριθμού 1 στον πολλαπλασιασμό αριθμών. Αν και η ιδιότητα αυτή ισχύει μόνο για τους τετραγωνικούς πίνακες, η εντολή `eye` μπορεί να κατασκευάσει και μη τετραγωνικούς πίνακες, όπως φαίνεται στο παρακάτω παράδειγμα:

```
>> eye(5,7)

ans =

     1     0     0     0     0     0     0
     0     1     0     0     0     0     0
     0     0     1     0     0     0     0
     0     0     0     1     0     0     0
     0     0     0     0     1     0     0
```

8.4.1.4 Γεννήτριες ψευδοτυχαίων αριθμών - Οι εντολές rand και randi

Ένας ακόμα τρόπος κατασκευής πινάκων, και συγκεκριμένα ψευδοτυχαίων πινάκων, είναι, όπως έχουμε ήδη δει, μέσω των εντολών `rand` και `randi`. Η πρώτη κατασκευάζει πίνακα με το μέγεθος που ορίζει ο χρήστης, και τοποθετεί στις θέσεις του ψευδοτυχαίους πραγματικούς αριθμούς μεταξύ του 0 και του 1. Η `randi` έχει παρόμοια λειτουργία, καθώς τοποθετεί στον πίνακα ψευδοτυχαίους, ακέραιους αριθμούς μεταξύ δύο τιμών που εισάγει ο χρήστης. Η σύνταξή των εντολών αυτών ακολουθεί τη λογική των προηγούμενων εντολών κατασκευής πινάκων, δηλαδή

- η `rand(n)` παράγει έναν $n \times n$ πίνακα ψευδοτυχαίων πραγματικών αριθμών μεταξύ του 0 και του 1 ,
- η `rand(m,n)` παράγει έναν $m \times n$ πίνακα ψευδοτυχαίων πραγματικών αριθμών μεταξύ του 0 και του 1.

Επίσης,

- η `randi([a,b],n)`: παράγει έναν $n \times n$ πίνακα ψευδοτυχαίων ακεραίων από το διάστημα [a,b],
- Η εντολή `randi([a,b],m,n)`: παράγει έναν $m \times n$ ψευδοτυχαίων ακεραίων από το διάστημα [a,b].

Παρατήρηση 8.8

Σημειώνεται ότι οι εντολές `rand` και `randi` παράγουν τυχαίους αριθμούς από τη (συνεχή) ομοιόμορφη και τη διακριτή ομοιόμορφη κατανομή αντίστοιχα. Οι κατανομές αυτές χρησιμοποιούνται για την περιγραφή τυχαίων φαινομένων, κατά τα οποία διαστήματα ίσου μήκους (για τη συνεχή περίπτωση) ή μεμονωμένες τιμές έχουν την ίδια πιθανότητα να παρατηρηθούν. Για περισσότερες πληροφορίες σχετικά με τις κατανομές αυτές, ο/η αναγνώστης/στρια παραπέμπεται, μεταξύ άλλων, στα βιβλία των Johnson *et al.* (1994, 2005) και Κουτρουβέλης (2011).

8.4.1.5 Συνδυασμός πινάκων και η εντολή repmat

Ένας ακόμα τρόπος για να δημιουργήσουμε πίνακες είναι να χρησιμοποιήσουμε συνδυασμούς από ήδη υπάρχοντες πίνακες ή/και τις εντολές που είδαμε στις προηγούμενες παραγράφους. Για παράδειγμα, μπορούμε να κατασκευάσουμε έναν πίνακα 8×8 που στο άνω αριστερό τεταρτημόριο να έχει μηδενικά, στο άνω δεξιά μονάδες, στο κάτω αριστερά έναν ταυτοτικό υποπίνακα και στο κάτω δεξιά ψευδοτυχαίους ακέραιους αριθμούς μεταξύ του 1 και του 10. Ο πίνακας αυτός κατασκευάζεται με την παρακάτω εντολή:

```
>> A = [zeros(4) , ones(4) ; eye(4) , randi([1,10],4)]

A =

     0     0     0     0     1     1     1     1
     0     0     0     0     1     1     1     1
     0     0     0     0     1     1     1     1
     0     0     0     0     1     1     1     1
     1     0     0     0     9     7    10    10
     0     1     0     0    10     1    10     5
     0     0     1     0     2     3     2     9
     0     0     0     1    10     6    10     2
```

Επίσης, μπορούμε να επαναλάβουμε ένα διάνυσμα ή έναν πίνακα που έχουμε κατασκευάσει χρησιμοποιώντας την εντολή `repmat(A,m,n)`, όπου A είναι ο πίνακας που θέλουμε να αντιγράψουμε, m είναι ο αριθμός των αντιγράφων που θα υπάρχουν στην κάθετη διεύθυνση και n ο αριθμός των αντιγράφων στην οριζόντια διεύθυνση. Για παράδειγμα, αν θέλουμε να επαναλάβουμε έναν 2×2 ταυτοτικό πίνακα, ώστε να δημιουργήσουμε έναν συνδυασμό με μέγεθος 4×6, πρέπει να εισάγουμε την εντολή:

```
>> A = repmat(eye(2) , 2 , 3)

A =

     1     0     1     0     1     0
     0     1     0     1     0     1
     1     0     1     0     1     0
     0     1     0     1     0     1
```

8.4.2 Δείκτες και προσπέλαση των στοιχείων ενός πίνακα

Ο τρόπος με τον οποίο μπορούμε να προσπελάσουμε στα στοιχεία ενός πίνακα δύο διαστάσεων είναι παρόμοιος με αυτόν των διανυσμάτων. Η διαφορά έγκειται στη χρήση δύο ξεχωριστών δεικτών, έναν για την αναφορά σε γραμμές και έναν για την αναφορά σε στήλες. Ας θεωρήσουμε έναν πίνακα με μέγεθος 2×5

```
>> mtrx1 = [1 2 3 4 5; 6 7 8 9 10]

mtrx1 =

     1     2     3     4     5
     6     7     8     9    10
```

Έστω ότι θέλουμε να ανακαλέσουμε από τον πίνακα την τιμή 8. Θα πρέπει να σκεφτούμε τη θέση στην οποία βρίσκεται αυτή η τιμή και να χρησιμοποιήσουμε τους κατάλληλους δείκτες. Εφόσον η εν λόγω τιμή είναι αποθηκευμένη στη δεύτερη γραμμή και στην τρίτη στήλη του πίνακα του παραδείγματος, το στοιχείο που θα ανακαλέσουμε είναι το $(2,3)$, δηλαδή

```
>> mtrx1(2,3)

ans =

     8
```

Αν επιθυμούμε να εξάγουμε ένα υποσύνολο του αρχικού μας πίνακα, μπορούμε να χρησιμοποιήσουμε τη γνωστή μας δομή «αρχή:βήμα:τέλος». Για παράδειγμα, για να ανακαλέσουμε τα δύο πρώτα στοιχεία που βρίσκονται στην πρώτη γραμμή του πίνακα mtrx1, μπορούμε να γράψουμε

```
>> mtrx1(1 , 1:2)

ans =

     1     2
```

όπου το 1 στην πρώτη θέση της παρένθεσης αντιστοιχεί στην πρώτη γραμμή, ενώ το 1:2 στη δεύτερη θέση αντιστοιχεί στις στήλες 1 και 2. Επίσης, αν θέλουμε να κατασκευάσουμε έναν νέο πίνακα mtrx2 με τις «ακριανές» τιμές του mtrx1, δηλαδή αυτές που βρίσκονται πάνω αριστερά, πάνω δεξιά, κάτω αριστερά και κάτω δεξιά, θα μπορούσαμε να εκτελέσουμε την εντολή

```
>> mtrx2 = mtrx1(1:2 , 1:4:5)

mtrx2 =

     1     5
     6    10
```

όπου το 1:2 στην πρώτη θέση της παρένθεσης αντιστοιχεί στην πρώτη και τη δεύτερη γραμμή, ενώ το 1:4:5 στη δεύτερη θέση αντιστοιχεί στις στήλες 1 και 5 (από το 1, με βήμα 4, έως το 5). Ο πίνακας που προκύπτει έχει μέγεθος 2×2.

Επίσης, αν θέλουμε να προσπελάσουμε *όλα* τα στοιχεία μιας συγκεκριμένης γραμμής ή στήλης του πίνακα μπορούμε να χρησιμοποιήσουμε αντί για τη δομή «αρχή:τέλος» απλώς τον τελεστή ":". Για παράδειγμα, ο πίνακας mtrx2 που είδαμε παραπάνω περιείχε και τις δύο γραμμές του mtrx1. Θα μπορούσαμε, λοιπόν, να τον δημιουργήσουμε γράφοντας

```
>> mtrx2 = mtrx1(: , 1:4:5)

mtrx2 =

     1     5
     6    10
```

Ο τελεστής ":" συμβολίζει τις δύο γραμμές του πίνακα mtrx1.

Ομοίως, αν θέλαμε να εξάγουμε ολόκληρη την πρώτη γραμμή του mtrx1, θα γράφαμε

```
>> array = mtrx1(1,:)

array =

     1     2     3     4     5
```

Παρατηρούμε ότι το αποτέλεσμα έχει τη μορφή διανύσματος-γραμμή, αφού εξάγουμε μόνο μία γραμμή από τον αρχικό μας πίνακα.

Όπως και στην περίπτωση των διανυσμάτων, μπορούμε και εδώ να αναθέσουμε μια συγκεκριμένη τιμή αντί μιας άλλης σε οποιαδήποτε θέση σε έναν πίνακα. Ένα παράδειγμα αυτού είναι το παρακάτω:

```
>> mtrx1(1 , 1) = 0

mtrx1 =

     0     2     3     4     5
     6     7     8     9    10
```

Μπορούμε, επίσης, να αλλάξουμε τις τιμές σε ένα τμήμα του πίνακα ή και σε μια ολόκληρη γραμμή ή στήλη, υπό την προϋπόθεση οι διαστάσεις του νέου τμήματος και αυτού που αλλάζουμε να συμπίπτουν. Για παράδειγμα, μπορούμε να προσπελάσουμε και να αλλάξουμε τις τιμές στις θέσεις (1,2), (1,3), (2,2) και (2,3) του mtrx1, οι οποίες σχηματίζουν ένα 2×2 υποπίνακα, εισάγοντας έναν νέο 2×2 πίνακα.

```
>> mtrx1(: , 2:3) = [102 103 ; 107 108]

mtrx1 =

     0   102   103     4     5
     6   107   108     9    10
```

Αν για κάποιον λόγο, το μέγεθος του τμήματος που επιθυμούμε να αλλάξουμε δεν συμπίπτει με τον νέο πίνακα που έχουμε ορίσει, τότε ο ελεγκτής του Matlab τυπώνει κατάλληλο μήνυμα σφάλματος. Ας δούμε ένα τέτοιο παράδειγμα

```
>> mtrx1(: , 2:3) = [102 103]
```

```
Unable to perform assignment because the size of the left side is 2-by-2 and
    the size of the right side is 1-by-2.
```

Το μήνυμα μας ενημερώνει ότι το τμήμα του πίνακα το οποίο θέλουμε να μεταβάλλουμε (αριστερό μέλος) έχει μέγεθος 2×2, ενώ ο πίνακας που θέλουμε να αναθέσουμε (δεξί μέλος) έχει μέγεθος 1×2. Η ασυμβατότητα αυτή δεν επιτρέπει την εκτέλεση της εντολής.

Τέλος, μπορούμε να διαγράψουμε ολόκληρες γραμμές ή στήλες ενός πίνακα, θέτοντάς τες ίσες με τον κενό πίνακα "[ ]". Για παράδειγμα, μπορούμε να αφαιρέσουμε την κεντρική στήλη του πίνακα mtrx1, που είδαμε παραπάνω, με την εντολή

```
>> mtrx1(:, 3) = [ ]

mtrx1 =

     0   102     4     5
     6   107     9    10
```

Γενικά, η χρήση υποπινάκων στο Matlab αποτελεί ένα πολύ χρήσιμο εργαλείο. Μας επιτρέπει να διαχειριζόμαστε τους πίνακες με ευκολία, αποφεύγοντας περιττές δομές επαναλήψεων που περιπλέκουν τον κώδικα και, σε κάποιες περιπτώσεις, τον καθιστούν πιο αργό. Για τον λόγο αυτό, παρόλο που η χρήση υποπινάκων απαιτεί αρκετή εξοικείωση, συστήνεται όπου αυτό είναι εφικτό.

Παράδειγμα 8.7

Κατασκευάστε έναν πίνακα με διαστάσεις 6×3, του οποίου το άνω μισό να περιέχει μηδενικά και το κάτω μισό να περιέχει μονάδες.

Λύση Παραδείγματος 8.7

Υπάρχουν πολύ τρόποι να κατασκευάσουμε τον ζητούμενο πίνακα. Ένας από αυτούς είναι να δημιουργήσουμε με την εντολή `zeros` έναν πίνακα με μηδενικά στοιχεία, ο οποίος να έχει την επιθυμητή διάσταση και, στη συνέχεια, να μετατρέψουμε το κάτω μισό, δηλαδή τις γραμμές 4 έως 6, σε μονάδες χρησιμοποιώντας την εντολή `ones`, ως εξής

```
>> A = zeros(6,3);
>> A(4:6,:) = ones(3);
>> disp(A)
     0     0     0
     0     0     0
     0     0     0
     1     1     1
     1     1     1
     1     1     1
```

Ένας εναλλακτικός τρόπος θα ήταν να ακολουθήσουμε την αντίστροφη διαδικασία, δηλαδή να κατασκευάσουμε πρώτα τον πίνακα με τις μονάδες και, στη συνέχεια, να αντικαταστήσουμε τα στοιχεία στο άνω μισό με μηδενικά. Η λύση αυτή φαίνεται παρακάτω

```
>> A = ones(6,3);
>> A(1:3,:) = zeros(3);
```

Ένας τρίτος τρόπος θα ήταν να κατασκευάσουμε δύο τετραγωνικούς πίνακες διαστάσεων 3×3, εκ των οποίων ο ένας να έχει μηδενικά και ο δεύτερος μονάδες, και να τους ενώσουμε σε μία στήλη, δηλαδή

```
>> A0 = zeros(3);
>> A1 = ones(3);
>> A = [A0 ; A1];
```

Όλες οι παραπάνω λύσεις επιτυγχάνουν το ίδιο αποτέλεσμα και υπάρχουν πολλοί ακόμα τρόποι για να υλοποιήσουμε τη συγκεκριμένη άσκηση, όπως π.χ με χρήση δομών επαναλήψεων, με εισαγωγή ελέγχων κ.λπ.

Παράδειγμα 8.8

Σε πολλές εφαρμογές στην αριθμητική ανάλυση χρειάζεται η κατασκευή ενός τριδιαγώνιου πίνακα της μορφής

```
A =

    -2     1     0     0     0
     1    -2     1     0     0
     0     1    -2     1     0
     0     0     1    -2     1
     0     0     0     1    -2
```

δηλαδή με την τιμή -2 στην κύρια διαγώνιο και μονάδες εκατέρωθεν αυτής. Ένας τέτοιος πίνακας μπορεί να προκύψει από τη διακριτοποίηση της εξίσωσης Laplace. Δημιουργήστε ένα αρχείο script, που να κατασκευάζει έναν τέτοιο πίνακα με το μέγεθος που ορίζει ο χρήστης.

Λύση Παραδείγματος 8.8

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε το μέγεθος του (τετραγωνικού) πίνακα, το οποίο θα εισάγει ο χρήστης μέσω της εντολής `input`, σε μία μεταβλητή (π.χ. N). Στη συνέχεια, θα χρειαστεί να ορίσουμε τον πίνακα (π.χ. A), ο οποίος μπορεί να περιέχει, αρχικά, μόνο μηδενικά στοιχεία.

Κύριος κώδικας: Παρατηρούμε ότι ο πίνακας έχει τρεις περιοχές στοιχείων, τα οποία είναι μη-μηδενικά: την κύρια διαγώνιο όπου η τιμή είναι -2, τα στοιχεία ακριβώς πάνω από την κύρια διαγώνιο που έχουν τιμή 1, και τα στοιχεία κάτω από την κύρια διαγώνιο, τα οποία επίσης έχουν τιμή 1. Με βάση την παραπάνω ανάλυση, η λύση του προβλήματος θα γίνει σε τρία στάδια.

- Αρχικά, θα εισάγουμε τα στοιχεία της κύριας διαγωνίου. Αυτό μπορεί να γίνει με έναν επαναληπτικό βρόχο, όπου ο μετρητής (π.χ. i) θα διατρέχει τις τιμές από 1 έως N και θα θέτει τα στοιχεία της κύριας διαγωνίου, δηλαδή τα A(i,i), ίσα με -2.

- Στη συνέχεια, θα εισάγουμε τα στοιχεία που βρίσκονται πάνω από την κύρια διαγώνιο. Ο συνολικός αριθμός των στοιχείων αυτών είναι N-1. Για τον λόγο αυτό, ο μετρητής του βρόχου θα διατρέχει τις τιμές από 1 έως N-1. Παρατηρούμε από τον πίνακα της εκφώνησης ότι το πρώτο στοιχείο πάνω από τη διαγώνιο που έχει τιμή 1 είναι το A(1,2), το δεύτερο είναι το A(2,3), το τρίτο είναι το A(3,4) κ.ο.κ. Προκύπτει, λοιπόν, ότι η γενική σχέση που εκφράζει τα στοιχεία πάνω από την κύρια διαγώνιο είναι A(i,i+1). Τα στοιχεία αυτά τα θέτουμε ίσα με 1.

- Στο τελευταίο βήμα, θα εισάγουμε τα στοιχεία που βρίσκονται κάτω από την κύρια διαγώνιο. Ο συνολικός αριθμός των στοιχείων αυτών είναι N-1, οπότε ο μετρητής του βρόχου θα πάρει τιμές από 1 έως N-1. Επίσης, παρατηρούμε ότι το πρώτο στοιχείο κάτω από τη διαγώνιο που έχει

τιμή 1 είναι το A(2,1), το δεύτερο είναι το A(3,2) κ.ο.κ. Άρα, για τα στοιχεία κάτω από την κύρια διαγώνιο μπορούμε να γράψουμε A(i+1,i)=1.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%KATASKEYH TRISDIAGWNIOU PINAKA
% Initialization
clear all
N = input('Give matrix size N: ');
A = zeros(N);

% Main code

% Main diagonal elements
for i=1:N
  A(i,i) = -2;
end
% Elements above the main diagonal
for i=1:N-1
  A(i,i+1) = 1;
end
% Elements below the main diagonal
for i=1:N-1
  A(i+1,i) = 1;
end
disp(A)
```

Άσκηση αυτοαξιολόγησης 8.3

Όπως έχουμε αναφέρει, η χρήση υποπινάκων μας επιτρέπει, σε ορισμένες περιπτώσεις, να αποφύγουμε τις επαναληπτικές διαδικασίες. Δοκιμάστε να δημιουργήσετε έναν τριδιαγώνιο πίνακα όπως αυτόν του Παραδείγματος 8.11, χωρίς τη χρήση επαναληπτικών βρόχων. Για να το πετύχετε αυτό, θα πρέπει να χρησιμοποιήσετε την εντολή `eye` σε συνδυασμό με τους υποπίνακες.

Παρατήρηση 8.9

Η χρήση δύο δεικτών κατά την προσπέλαση κάποιου στοιχείου ενός πίνακα είναι πολύ εύχρηστη, καθώς μας δείχνει τη θέση του σημείου στον πίνακα. Ωστόσο, μπορούμε να προσπελάσουμε το ίδιο στοιχείο, χρησιμοποιώντας έναν μόνο δείκτη, αν θεωρήσουμε ότι ο διδιάστατος πίνακας είναι μια δομή από συνεχόμενα διανύσματα-γραμμή. Με άλλα λόγια, σε έναν πίνακα A με μέγεθος 3×3, το στοιχείο A(3) αντιστοιχεί στο A(1,3), το στοιχείο A(6) αντιστοιχεί στο A(2,3), το στοιχείο A(8) αντιστοιχεί στο A(3,2) κ.ο.κ. Ωστόσο, ο τρόπος αυτός προσπέλασης στοιχείων δεν είναι εύχρηστος, καθώς δεν γίνεται εύκολα αντιληπτή η θέση του στοιχείου.

8.5 Πληροφορίες για το μέγεθος πίνακα και τα στοιχεία του - Εντολές size, length, numel, sum, max/min και find

Σε πολλές εφαρμογές είναι εξαιρετικά χρήσιμο να γνωρίζουμε το μέγεθος ενός πίνακα, δηλαδή το πλήθος των στοιχείων που περιέχονται σε καθεμία εκ των διαστάσεων. Επίσης, πολύ συχνά χρειάζεται να έχουμε πληροφορίες για τα στοιχεία ενός πίνακα, όπως το άθροισμά τους, τις μέγιστες και ελάχιστες τιμές, τις θέσεις στις οποίες υπάρχουν μηδενικά, μιγαδικοί αριθμοί, τιμές NaN κ.ο.κ. Στις επόμενες παραγράφους θα μιλήσουμε για εντολές που εκτελούν αυτές τις λειτουργίες.

8.5.1 Η συνάρτηση size

Ξεκινάμε με την εντολή `size`, η οποία έχει παρουσιαστεί και στο Κεφάλαιο 1. Η εντολή αυτή επιστρέφει το μήκος κάθε διάστασης ενός πίνακα. Για παράδειγμα, αν έχουμε έναν 3×5 πίνακα A, η εντολή size(A) θα μας επιστρέψει ένα διάνυσμα με τις τιμές 3 και 5, που είναι το πλήθος των γραμμών και των στηλών του A, αντίστοιχα.

```
>> A = zeros(3,5);
>> size(A)

ans =

     3     5
```

Το αποτέλεσμα αυτό, μπορούμε να το αποθηκεύσουμε είτε ως διάνυσμα είτε ως ξεχωριστές τιμές, ώστε να το αξιοποιήσουμε στον κώδικά μας.

Αν επιθυμούμε να λάβουμε πληροφορίες για συγκεκριμένη διάσταση, δηλαδή γραμμές ή στήλες ενός πίνακα, τότε μπορούμε να χρησιμοποιήσουμε την εντολή `size` σε συνδυασμό με κατάλληλο δείκτη (1 για γραμμές, 2 για στήλες). Για παράδειγμα, αν θέλαμε μόνο τον αριθμό των γραμμών του πίνακα A, που είδαμε προηγουμένως, θα μπορούσαμε να γράψουμε

```
>> size(A,1)

ans =

     3
```

8.5.2 Η εντολή length

Την εντολή `length` την είδαμε στην ενότητα που αφορούσε τα διανύσματα. Ωστόσο, εφαρμόζεται και στους πίνακες, όπου επιστρέφει το μήκος της διάστασης με τα περισσότερα στοιχεία. Για παράδειγμα, για όλους τους πίνακες A, που ακολουθούν, το αποτέλεσμα της εντολής `length(A)` είναι 5.

```
>> A = zeros(2,5)

ans =

     5
>> A = zeros(5,2)
```

```
ans =

     5
>> A = zeros(5,5)

ans =

     5
```

8.5.3 Η εντολή numel

Με την εντολή `numel` μπορούμε να εξάγουμε το γινόμενο του αριθμού των γραμμών επί τον αριθμό των στηλών. Το γινόμενο αυτό δίνει το συνολικό πλήθος στοιχείων του πίνακα. Έτσι γράφοντας

```
>> A = zeros(3,5);
>> numel(A)

ans =

    15
```

λαμβάνουμε το αποτέλεσμα $3 \times 5 = 15$

Παράδειγμα 8.9

Κατασκευάστε έναν μηδενικό πίνακα A με μέγεθος 5×6. Στη συνέχεια, αποσπάστε τις πληροφορίες που θα μας έδιναν οι εντολές `length` και `numel`, χρησιμοποιώντας, όμως, αποκλειστικά την εντολή `size`.

Λύση Παραδείγματος 8.9

Για να κατασκευάσουμε το διάνυσμα A, θα χρησιμοποιήσουμε την εντολή `zeros` ως εξής

```
>> A = zeros(5,6);
```

Στη συνέχεια, θα υπολογίσουμε το μήκος της διάστασης με τα περισσότερα στοιχεία, δηλαδή τον αριθμό των στηλών, χρησιμοποιώντας την εντολή `size` σε συνδυασμό με την εντολή `max`

```
>> max(size(A))

ans =

     6
```

Η παραπάνω γραμμή κώδικα υλοποιεί δύο βήματα:

- η εντολή `size(A)` δημιουργεί ένα διάνυσμα δύο θέσεων, όπου στην πρώτη υπάρχει ο αριθμός των γραμμών, δηλαδή 5, και στη δεύτερη θέση ο αριθμός των στηλών, δηλαδή 6,
- η εντολή `max(...)`, με όρισμα το διάνυσμα που προαναφέραμε, επιστρέφει τη μέγιστη μεταξύ των δύο τιμών, δηλαδή το 6.

Θα μπορούσαμε να έχουμε εκτελέσει τα δύο παραπάνω βήματα σε δύο ξεχωριστές εντολές, χρησιμοποιώντας μία βοηθητική μεταβλητή. Ο τρόπος που επιλέξαμε, όμως, είναι πιο «οικονομικός» υπολογιστικά, καθώς χρειάζεται μόνο μία γραμμή κώδικα.
Για να υπολογίσουμε το γινόμενο του αριθμού των σειρών επί τον αριθμό των στηλών, χωρίς την εντολή `numel`, θα αξιοποιήσουμε τη δυνατότητα της εντολής `size` να μας δίνει ξεχωριστά αριθμό γραμμών και στηλών, ως εξής

```
>> size(A,1) * size(A,2)

ans =

    30
```

8.5.4 Η εντολή sum

Όπως αναφέραμε στην ενότητα που αφορούσε τα διανύσματα, η εντολή `sum` υπολογίζει το άθροισμα των στοιχείων ενός διανύσματος. Η εντολή αυτή εφαρμόζεται και στους πίνακες, αλλά δεν επιστρέφει το άθροισμα όλων των στοιχείων του πίνακα. Αυτό που κάνει η εντολή `sum` είναι να υπολογίζει το άθροισμα κάθε στήλης του πίνακα και να επιστρέφει ένα διάνυσμα-γραμμή με τα αθροίσματα αυτά. Ας δούμε ένα παράδειγμα:

```
>> A = randi([1,10],4)

A =

     9     7    10    10
    10     1    10     5
     2     3     2     9
    10     6    10     2

>> sum(A)

ans =

    31    17    32    26
```

Στο παραπάνω παράδειγμα, δημιουργήσαμε έναν πίνακα με ψευδοτυχαίους ακέραιους αριθμούς μεταξύ του 1 και του 10. Με την εντολή `sum(A)` εξαγουμε ένα διάνυσμα-γραμμή με το άθροισμα κάθε στήλης του A.

Αν θέλαμε να βρούμε το άθροισμα όλων των στοιχείων του A, τότε θα έπρεπε να προσθέσουμε μεταξύ τους τις τιμές του διανύσματος που προκύπτει από την εντολή `sum(A)`. Με άλλα λόγια, θα έπρεπε να εφαρμόσουμε δύο φορές την εντολή `sum`, όπως φαίνεται παρακάτω

```
>> sum( sum(A) )

ans =

   106
```

Η σύνθετη αυτή εντολή υπολογίζει πρώτα το επιμέρους άθροισμα κάθε στήλης και, στη συνέχεια, προσθέτει όλα αυτά τα αθροίσματα μεταξύ τους.

8.5.5 Οι εντολές max/min

Οι εντολές `max/min`, όταν αφορούν πίνακες, λειτουργούν όπως και η `sum`. Αυτό που κάνουν είναι να επιστρέφουν τη μέγιστη/ελάχιστη τιμή κάθε στήλης του πίνακα, τοποθετώντας τις αυτές σε ένα διάνυσμα-γραμμή. Ας δούμε ένα παράδειγμα:

```
>> A = randi([1,10],4)

A =

     9     7    10    10
    10     1    10     5
     2     3     2     9
    10     6    10     2

>> max(A)

ans =

    10     7    10    10

>> min(A)

ans =

     2     1     2     2
```

Για να βρούμε το μέγιστο/ελάχιστο όλων των στοιχείων του A, θα πρέπει να εφαρμόσουμε δύο φορές την αντίστοιχη εντολή, όπως φαίνεται παρακάτω:

```
>> max( max(A) )

ans =

    10

>> min( min(A) )

ans =

     1
```

8.5.6 Εντολή find

Η εντολή `find` χρησιμοποιείται για να εντοπίσει τις θέσεις σε έναν πίνακα, στις οποίες υπάρχουν μη-μηδενικές τιμές ή, γενικότερα, στις οποίες ισχύει μία δεδομένη συνθήκη. Το αποτέλεσμα της εντολής `find` επιστρέφει τις θέσεις είτε υπό τη μορφή γραμμή-στήλη είτε μετρώντας γραμμικά όλα τα στοιχεία του πίνακα, σαν να ήταν μία συνέχεια από διανύσματα-στήλη. Ας δούμε ένα παράδειγμα. Θεωρούμε τυχαίο πίνακα 3×3 με ακέραιες τιμές στο διάστημα [0,2].

```
>> A = randi([0,2],3)

A =

     2     0     0
     0     2     0
     2     1     1
```

Η εντολή `find(A)` θα επιστρέψει τις θέσεις στις οποίες υπάρχουν μη-μηδενικές τιμές, θεωρώντας τον πίνακα μία σειρά από συνεχόμενα διανύσματα-στήλες.

```
>> find(A)

ans =

     1
     3
     5
     6
     9
```

Το αποτέλεσμα αυτό δεν γίνεται εύκολα κατανοητό, καθώς χρειάζεται επιπλέον πράξεις για να το εκφράσουμε στη μορφή γραμμή-στήλη, η οποία είναι η τυπική για διδιάστατους πίνακες. Είναι, λοιπόν, πιο εύχρηστο να αναθέσουμε το αποτέλεσμα της εντολής `find` σε έναν συνδυασμό από δύο μεταβλητές, όπως φαίνεται παρακάτω.

```
>> [iArray,jArray]=find(A)

iArray =

     1
     3
     2
     3
     3

jArray =

     1
     1
     2
     2
     3
```

Στην περίπτωση αυτή, το πρώτο διάνυσμα (iArray στο παράδειγμα αυτό) περιέχει τη γραμμή και το δεύτερο (jArray) τη στήλη, στην οποία εμφανίζεται το μη-μηδενικό στοιχείο. Για το παραπάνω παράδειγμα, μπορούμε να αντιληφθούμε ότι μη-μηδενικές τιμές υπάρχουν στις θέσεις (1,1), (3,1), (2,3), (2,2) και (3,3) του πίνακα A.

Στη γενικότερη μορφή της, η εντολή `find` μπορεί να συνδυαστεί με οποιαδήποτε συνθήκη.

Παράδειγμα 8.10

Ορίστε έναν πίνακα A με μέγεθος 5×5 που να περιέχει τυχαίους ακέραιους από το 0 έως το 100. Στη συνέχεια, βρείτε τη μέγιστη και την ελάχιστη τιμή του A, καθώς και τις θέσεις στις οποίες βρίσκονται.

Λύση Παραδείγματος 8.10

Για να κατασκευάσουμε τον πίνακα A, θα χρησιμοποιήσουμε την εντολή `randi`

```
>> A = randi([0,100],5)

A =

    80    14    14    40    24
    43    13    86     7    42
    91    87    62    24     5
    18    58    35    12    91
    26    55    51    18    95
```

Στη συνέχεια, θα βρούμε τη μέγιστη τιμή (π.χ. maxA) και την ελάχιστη (π.χ. minA), χρησιμοποιώντας τις εντολές `max` και `min`, αντίστοιχα.

```
>> maxA = max( max(A) )

maxA =

    95

>> minA = min( min(A) )

minA =

     5
```

Τέλος, θα βρούμε τις θέσεις των maxA και minA μέσα στον πίνακα A με την εντολή `find`

```
>> [iMax,jMax] = find( A==maxA )

iMax =

     5


jMax =

     5

>> [iMin,jMin] = find( A==minA )
```

```
iMin =

     3


jMin =

     5
```

Το αποτέλεσμα των εντολών μας πληροφορεί ότι η τιμή maxA βρίσκεται στη θέση (5,5) και η τιμή minA στη θέση (3,5). Τις πληροφορίες αυτές μπορούμε να τις επιβεβαιώσουμε από τον πίνακα A, ο οποίος παρουσιάζεται στην αρχή της άσκησης.

8.6 Πράξεις μεταξύ πινάκων

Οι πράξεις μεταξύ πινάκων είναι παρόμοιες με αυτές των διανυσμάτων και παρουσιάζονται συνοπτικά στο Παράρτημα Α΄. Αναλυτικές πληροφορίες για τον πολλαπλασιασμό πινάκων αλλά και γενικότερα για τους πίνακες και τις εφαρμογές τους, μπορεί να αναζητηθούν σε οποιοδήποτε εισαγωγικό βιβλίο γραμμικής άλγεβρας, όπως, παραδείγματος χάριν, στα βιβλία των Anton και Rorres (2000) και Strang και Πάμφιλος (1996 [ανατύπωση 2002]). Σε αυτήν την παράγραφο θα περιγράψουμε την υλοποίησή τους στη γλώσσα Matlab και θα παρουσιάσουμε χαρακτηριστικά παραδείγματα.

Η πρώτη μεγάλη κατηγορία πράξεων περιλαμβάνει τις πράξεις που, για να εκτελεστούν, πρέπει οι δύο πίνακες να έχουν το ίδιο ακριβώς μέγεθος. Στην κατηγορία αυτή ανήκουν η πρόσθεση και η αφαίρεση, καθώς και οι πράξεις στοιχείο προς στοιχείο (γινόμενο, διαίρεση, ύψωση σε δύναμη). Στις πράξεις στοιχείο προς στοιχείο δεν θα πρέπει να ξεχνάμε να τοποθετήσουμε μία τελεία “.” πριν από τον τελεστή της πράξης.

```
>> A = randi([1,5],3)

A =

     5     5     2
     5     4     3
     1     1     5

>> B = A + A

B =

    10    10     4
    10     8     6
     2     2    10
```

```
>> C = A .* B

C =

    50    50     8
    50    32    18
     2     2    50

>> D = A.^2

D =

    25    25     4
    25    16     9
     1     1    25
```

Η δεύτερη κατηγορία περιλαμβάνει τον πολλαπλασιασμό πινάκων. Για να μπορέσει να γίνει αυτή η πράξη, θα πρέπει τα μεγέθη των δύο πινάκων να έχουν μία συγκεκριμένη σχέση μεταξύ τους: θα πρέπει ο αριθμός των στηλών του 1ου να είναι ίσος με τον αριθμό των γραμμών του 2ου. Για παράδειγμα, ένας $m \times k$ πίνακας μπορεί να πολλαπλασιαστεί μόνο με έναν $k \times n$ πίνακα. Ας δούμε ένα παράδειγμα:

```
>> A=randi([1,5],2,3)

A =

     1     5     5
     3     4     4

>> B = randi([1,5],3,3)

B =

     1     4     2
     5     4     4
     5     4     1

>> A * B

ans =

    51    44    27
    43    44    26
```

Αν προσπαθήσουμε να αλλάξουμε τη σειρά πολλαπλασιασμού, ο ελεγκτής του Matlab μας ειδοποιεί με κατάλληλο μήνυμα σφάλματος.

```
>> B * A
Error using  *

Incorrect dimensions for matrix multiplication. Check that the number of
   columns in the first matrix matches the number of rows in the second matrix
   . To perform elementwise multiplication, use "".*.
```

Παρατηρούμε ότι το μήνυμα σφάλματος μας ειδοποιεί για το πρόβλημα που εντόπισε ο ελεγκτής και προτείνει μία πιθανή αιτία: πολλές φορές, ενώ θέλουμε να κάνουμε πολλαπλασιασμό στοιχείο προς στοιχείο, εκ παραδρομής δεν τοποθετούμε την τελεία "." πριν το σύμβολο του γινομένου, δηλαδή τον αστερίσκο "∗". Το μήνυμα μάς εφιστά την προσοχή στο συχνό αυτό λάθος, χωρίς όμως απαραίτητα να είναι αυτή η λύση στο πρόβλημά μας, αφού οι δύο πίνακες δεν έχουν ίδιο μέγεθος, οπότε ούτε ο πολλαπλασιασμός στοιχείο προς στοιχείο είναι δυνατός.

Γενικά, τα μηνύματα του ελεγκτή του Matlab είναι χρήσιμα, καθώς μας ενημερώνουν για το είδος του σφάλματος που έχει προκύψει, ωστόσο πολλές φορές απαιτείται προσεκτική ανάλυση από τον προγραμματιστή, προκειμένου να εντοπιστεί η ακριβής αιτία του προβλήματος.

Για να καταλάβουμε καλύτερα τον πολλαπλασιασμό πινάκων, ακολουθεί το επόμενο παράδειγμα στο οποίο περιγράφεται και κατασκευάζεται ένας αναλυτικός κώδικας για την αναπαραγωγή της πράξης του πολλαπλασιασμού πινάκων.

Παράδειγμα 8.11

Ο πολλαπλασιασμός δύο πινάκων είναι μία σύνθετη πράξη κατά την οποία κάθε γραμμή του πρώτου πίνακα πολλαπλασιάζεται με εσωτερικό γινόμενο με κάθε στήλη του δεύτερου πίνακα. Το εσωτερικό γινόμενο το είδαμε όταν εξετάσαμε τις πράξεις μεταξύ διανυσμάτων, ενώ συνοπτικές πληροφορίες για την πράξη του πολλαπλασιασμού πινάκων υπάρχουν στο εγχειρίδιο με τα βασικά στοιχεία των Πινάκων, στο Παράρτημα Α΄.
Δημιουργήστε ένα αρχείο script το οποίο θα ζητά από τον χρήστη να εισάγει δύο πίνακες A και B και θα υπολογίζει το γινόμενό τους, C. Οι πίνακες μπορεί να έχουν οποιοδήποτε μέγεθος, θα πρέπει όμως απαραίτητα ο αριθμός των στηλών του A να είναι ίσος με τον αριθμό των γραμμών του B. Αν, δηλαδή, ο πρώτος έχει μέγεθος $m \times k$, ο δεύτερος θα πρέπει να είναι $k \times n$.
Ο πίνακας του γινομένου C υπολογίζεται ως εξής:

- το στοιχείο C(1,1) είναι το αποτέλεσμα του εσωτερικού γινομένου της 1ης γραμμής του A με την 1η στήλη του B,
- το στοιχείο C(1,2) είναι το αποτέλεσμα του εσωτερικού γινομένου της 1ης γραμμής του A με τη 2η στήλη του B,
- το στοιχείο C(2,1) είναι το αποτέλεσμα του εσωτερικού γινομένου της 2ης γραμμής του A με την 1η στήλη του B,
- γενικά, το στοιχείο C(i,j) είναι το αποτέλεσμα του εσωτερικού γινομένου της i γραμμής του A με την j στήλη του B.

Σύμφωνα με τα παραπάνω, το αποτέλεσμα του πολλαπλασιασμού ενός $m \times k$ πίνακα με έναν $k \times n$ πίνακα θα έχει μέγεθος $m \times n$.

Λύση Παραδείγματος 8.11

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε τους δύο πίνακες που θα εισάγει ο χρήστης σε δύο μεταβλητές (π.χ. A και B) μέσω της εντολής `input`. Στη συνέχεια, θα χρειαστεί να βρούμε τα μεγέθη τους με χρήση της εντολής `size` (π.χ. m και k1 ο αριθμός γραμμών και στηλών του A και k2 και n ο αριθμός γραμμών και στηλών του B). Σημειώνουμε ότι για να μπορεί να γίνει η πράξη του πολλαπλασιασμού, θα πρέπει η τιμή των k1 και k2 να είναι κοινή στους δύο πίνακες. Τέλος, θα ορίσουμε τον πίνακα του γινομένου (π.χ. C), ο οποίος θα έχει μέγεθος $m \times n$, σύμφωνα με την εκφώνηση.

Κύριος κώδικας: Όπως αναλύεται στην εκφώνηση, κάθε στοιχείο (i,j) του πίνακα C προκύπτει από το εσωτερικό γινόμενο της i γραμμής του A με την j στήλη του B. Άρα, για τον υπολογισμό όλων των στοιχείων του C θα χρειαστούμε δύο επαναληπτικές δομές, η μία εκ των οποίων θα είναι εμφωλευμένη μέσα στην άλλη. Η μία θα εκτελεί m επαναλήψεις και η άλλη n και το αποτέλεσμα είναι το ίδιο, ανεξάρτητα από το ποια από τις δύο είναι εσωτερική της άλλης. Μέσω της διπλής επαναληπτικής διαδικασίας θα υπολογιστούν $m \times n$ εσωτερικά γινόμενα, τα οποία θα αποτελέσουν τα στοιχεία του πίνακα C.

Επαλήθευση: Στο συγκεκριμένο παράδειγμα, μπορούμε να επιβεβαιώσουμε ότι έχουμε υπολογίσει σωστά το αποτέλεσμα, συγκρίνοντάς το με την πράξη του πολλαπλασιασμού πινάκων, η οποία εκτελείται με τον τελεστή "*".

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %POLLAPLASIASMOS PINAKWN
2  % Initialization
3   clear all
4   A = input('Give matrix A: ');
5   B = input('Give matrix B: ');
6   [m,k1] = size(A);
7   [k2,n] = size(B);
8   C = zeros(m,n);
9
10
11  % Main code
12    for i=1:m
13      for j=1:n
14        C(i,j) = A(i,:) * B(:,j);
15      end
16    end
17
18    disp("To apotelesma einai:")
19    disp(C)
20
21   % Verification
22    disp("A * B =")
23    disp(A*B)
```

Παρατήρηση 8.10

Όπως είδαμε στα παραπάνω παραδείγματα κώδικα ένας $m \times k$ πίνακας A, μπορεί να πολλαπλασιαστεί μόνο με έναν $k \times n$ πίνακα B, ενώ το γινόμενο $B * A$ δεν ορίζεται. Μοναδική περίπτωση που ορίζονται και τα δύο γινόμενα $A * B$ και $B * A$ είναι όταν οι πίνακες A και B είναι τετραγωνικοί (βλ. την επόμενη ενότητα), δηλαδή έχουν το ίδιο πλήθος γραμμών και στηλών. Σημειώνεται, όμως, ότι και σε αυτήν την περίπτωση δεν ισχύει η αντιμεταθετική ιδιότητα, αφού γενικά $A * B \neq B * A$.

8.7 Τετραγωνικοί πίνακες - Επίλυση γραμμικών συστημάτων

Οι τετραγωνικοί πίνακες έχουν ιδιαίτερη θέση ανάμεσα στους διδιάστατους πίνακες, καθώς μπορούμε να υπολογίσουμε μεγέθη, όπως η ορίζουσα, το ίχνος και οι ιδιοτιμές. Επίσης, εκτός από τις πράξεις που έχουμε αναφέρει ήδη (πρόσθεση, αφαίρεση, πράξεις στοιχείο προς στοιχείο και πολλαπλασιασμός), μπορούμε να εκτελέσουμε την αντίστοιχη πράξη της διαίρεσης, δηλαδή πολλαπλασιασμό με τον αντίστροφο πίνακα και να τους υψώσουμε σε δυνάμεις. Επίσης, οι τετραγωνικοί πίνακες είναι πολύ χρήσιμοι στην επίλυση γραμμικών αλγεβρικών συστημάτων. Στις επόμενες παραγράφους, παρουσιάζονται οι βασικές εντολές του Matlab που χρησιμοποιούνται για τετραγωνικούς πίνακες, καθώς και ο τρόπος επίλυσης γραμμικών συστημάτων.

8.7.1 Ορίζουσα πίνακα - Εντολή det

Η ορίζουσα ενός πίνακα προκύπτει μετά από πράξεις μεταξύ των στοιχείων του, όπως περιγράφεται στο Παράρτημα Α΄. Είναι μία μοναδική τιμή για κάθε πίνακα και υπολογίζεται με την εντολή `det`, όπως φαίνεται στο παρακάτω παράδειγμα.

```
>> A = randi([1,5],3)

A =

     5     5     2
     5     4     3
     1     1     5

>> det(A)

ans =

   -23
```

Πρέπει να σημειωθεί ότι, όταν η ορίζουσα είναι διάφορη του μηδενός, ο πίνακας μπορεί να αντιστραφεί. Επίσης, στα γραμμικά αλγεβρικά συστήματα η μη-μηδενική ορίζουσα μας δείχνει ότι υπάρχει μοναδική λύση.

8.7.2 Ίχνος πίνακα - Εντολή trace

Το ίχνος ενός τετραγωνικού πίνακα είναι το άθροισμα των στοιχείων της διαγωνίου (βλ. Παράρτημα Α΄) και υπολογίζεται με την εντολή trace. Αν θεωρήσουμε τον πίνακα A της προηγούμενης παραγράφου, το ίχνος υπολογίζεται ως εξής:

```
>> trace(A)
```

```
ans =

    14
```

8.7.3 Ιδιοτιμές πίνακα - Εντολή eig

Οι ιδιοτιμές ενός πίνακα είναι χαρακτηριστικές τιμές οι οποίες μπορούν να μας δώσουν πολλές πληροφορίες για τις ιδιότητές του και είναι ιδιαίτερα χρήσιμες σε πολλές εφαρμογές. Παραδείγματος χάριν, οι ιδιοτιμές χρησιμοποιούνται για τον προσδιορισμό των φυσικών συχνοτήτων (ή ιδιοσυχνοτήτων) μιας δόνησης (βλ., μεταξύ άλλων, Σταυρουλάκης κ.ά., 2015, Κεφάλαιο 4) ή για τη μείωση διαστάσεων σε ένα πολυδιάστατο πρόβλημα μέσω της μεθόδου της ανάλυσης κυρίων συνιστωσών (βλ., μεταξύ άλλων, Jolliffe, 2013). Οι εφαρμογές αυτές ξεφεύγουν από τους σκοπούς του παρόντος συγγράμματος και για αυτό τον λόγο περιοριζόμαστε μόνο στην παρουσίαση της εντολής του Matlab για τον υπολογισμό των ιδιοτιμών ενός πίνακα. Οι ιδιοτιμές ενός πίνακα υπολογίζονται μέσω της εντολής `eig`, όπως φαίνεται παρακάτω.

```
>> eig(A)

ans =

   10.4345
   -0.5373
    4.1028
```

8.7.4 Δυνάμεις τετραγωνικών πινάκων

Το γινόμενο δύο τετραγωνικών πινάκων με μέγεθος $n \times n$, είναι κι αυτό ένας τετραγωνικός πίνακας με το ίδιο μέγεθος. Το γεγονός αυτό επιτρέπει τον πολλαπλασιασμό ενός πίνακα με τον εαυτό του, μία ή περισσότερες φορές, δηλαδή την ύψωσή του σε κάποια δύναμη. Η πράξη αυτή γίνεται με το σύμβολο " ^ ", όπως και στους αριθμούς.

```
>> A=randi([1,5],2)

A =

     4     2
     1     1

>> A^2

ans =

    18    10
     5     3
```

8.7.5 Αντιστροφή πίνακα

Στην άλγεβρα πινάκων το αντίστοιχο της πράξης της διαίρεσης είναι ο πολλαπλασιασμός με τον αντίστροφο. Ο αντίστροφος ενός τετραγωνικού πίνακα A συμβολίζεται με A^{-1} και το γινόμενο των δύο αυτών πινάκων δίνει ως αποτέλεσμα τον ταυτοτικό πίνακα. Ο αντίστροφος, A^{-1}, ενός τετραγωνικού πίνακα A, αν υπάρχει, είναι ένας πολύ χρήσιμος πίνακας, ειδικά για την επίλυση γραμμικών συστημάτων, όπως θα δούμε στην επόμενη ενότητα. Ο αντίστροφος πίνακας μπορεί να προσδιοριστεί με διάφορες μεθόδους (βλ., μεταξύ άλλων, τα βιβλία των Lipschutz και Lipson, 2018; Φιλιππάκης, 2018), αλλά η αναλυτική παρουσίαση των μεθόδων αυτών ξεφεύγει από τους σκοπούς του παρόντος συγγράμματος και για αυτό τον λόγο περιοριζόμαστε μόνο στην παρουσίαση της εντολής `inv` του Matlab που επιστρέφει τον αντίστροφο ενός τετραγωνικού πίνακα.

```
>> A=randi([1,5],3)

A =

     1     2     3
     5     5     2
     4     1     4

>> inv(A)

ans =

   -0.3529    0.0980    0.2157
    0.2353    0.1569   -0.2549
    0.2941   -0.1373    0.0980

>> A * inv(A)

ans =

    1     0     0
    0     1     0
    0     0     1
```

Παρατήρηση 8.11

Σημειώνεται ότι ο αντίστροφος ενός τετραγωνικού πίνακα υπάρχει αν και μόνο αν η ορίζουσά του είναι διάφορη του μηδενός.

Παρατήρηση 8.12

Ενώ στη γραμμική άλγεβρα πινάκων δεν ορίζεται η πράξη της διαίρεσης, αλλά μόνο ο πολλαπλασιασμός με τον αντίστροφο πίνακα, το Matlab επιτρέπει τη χρήση του συμβόλου της διαίρεσης "/", για να εκτελεστεί η πράξη αυτή. Με άλλα λόγια, αν έχουμε δύο $n \times n$ πίνακες A και B, η πράξη

```
>> A * inv(B);
```

δίνει το ίδιο αποτέλεσμα με την εντολή

```
>> A / B;
```

8.7.6 Επίλυση γραμμικών συστημάτων

Πολλές εφαρμογές στην αριθμητική ανάλυση περιλαμβάνουν την επίλυση γραμμικών συστημάτων εξισώσεων. Τα συστήματα αυτά γράφονται στη μορφή

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \cdots & \alpha_{nn} \end{pmatrix} * \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix}$$

ή συμβολικά

$$A * X = B$$

όπου A είναι ο πίνακας των συντελεστών των αγνώστων, X είναι το διάνυσμα των αγνώστων και B το διάνυσμα των σταθερών όρων. Η επίλυση τέτοιων συστημάτων μπορεί να γίνει με πολλούς τρόπους, όπως η απαλοιφή Gauss και οι επαναληπτικές μέθοδοι Jacobi, Gauss-Seidel και των συζυγών υπολοίπων ή και με τη χρήση του αντίστροφου πίνακα, αν υπάρχει. Η τελευταία προσέγγιση, με τον αντίστροφο πίνακα, βασίζεται στο γεγονός ότι, αν υπάρχει ο A^{-1}, τότε, αν πολλαπλασιάσουμε από αριστερά και τα δύο μέρη της σχέσης $A * X = B$ με τον A^{-1}, προκύπτει ότι

$$X = A^{-1}B,$$

η οποία σχέση μας δίνει τη μοναδική λύση του συστήματος.

Παρατήρηση 8.13

Το σύστημα $A * X = B$ έχει μοναδική λύση, την $X = A^{-1}B$, αν και μόνο αν η ορίζουσά του είναι διάφορη του μηδενός.

Στο Matlab η επίλυση ενός τέτοιου συστήματος γίνεται με χρήση του τελεστή “\” (πλάγια γραμμή με αντίθετη κλίση από το σύμβολο της διαίρεσης) ή, όπως προαναφέρθηκε, με τη χρήση του αντίστροφου πίνακα. Για παράδειγμα, ας υποθέσουμε ότι έχουμε το σύστημα

$$x_1 - 3x_2 + 2x_3 = 1$$

$$4x_1 - 2x_2 + 5x_3 = 15$$

$$x_1 - x_2 - x_3 = -4$$

Για να λύσουμε το παραπάνω σύστημα πρέπει να εισάγουμε στο Matlab τον πίνακα των συντελεστών των αγνώστων A και το διάνυσμα των σταθερών όρων B ως εξής:

```
>> A = [1,-3,2;4,-2,5;1,-1,-1]

A =

     1    -3     2
     4    -2     5
     1    -1    -1

>> B =[1;15;-4]
```

```
B =

     1
    15
    -4
```

Η λύση δίνεται με την εντολή

```
>> X = A \ B

X =

     1
     2
     3
```

ή με την

```
>> X = inv(A) * B
```

η οποία επιστρέφει το ίδιο αποτέλεσμα.

Μπορούμε να επιβεβαιώσουμε τη λύση που βρήκαμε αν πολλαπλασιάσουμε τον πίνακα A με το διάνυσμα X και λάβουμε το αποτέλεσμα B.

```
>> A * X

ans =

     1
    15
    -4
```

Παρατήρηση 8.14

Η μέθοδος επίλυσης με τον τελεστή "\" δεν είναι η μοναδική στη γλώσσα Matlab, ωστόσο είναι πολύ «ασφαλής» στη χρήση της, καθώς πριν την επίλυση ελέγχει αν ο πίνακας των αγνώστων έχει ειδικές ιδιότητες. Λόγω των ελέγχων αυτών, είναι πιο αργή σε σχέση με άλλες εναλλακτικές εντολές, όπως, παραδείγματος χάριν, η `linsolve`. Ας δούμε πώς εφαρμόζεται η `linsolve` για την επίλυση του παραπάνω συστήματος.

```
>> X = linsolve(A,B)

X =

     1
     2
     3
```

8.8 Πολυδιάστατοι πίνακες

Στο κεφάλαιο αυτό συζητήσαμε για την έννοια του πίνακα σε μία και δύο διαστάσεις. Είδαμε ότι υπάρχουν πολλές ομοιότητες μεταξύ διανυσμάτων και πινάκων σε ό,τι αφορά τον τρόπο προσπέλασης των στοιχείων, τις πράξεις αλλά και την εκτέλεση των σχετικών εντολών. Αυτό μπορούμε να το κατανοήσουμε, αν αντιληφθούμε τα στοιχεία, τα διανύσματα και τους διδιάστατους πίνακες σαν μία διαδοχική επέκταση της έννοιας του πίνακα σε περισσότερες διαστάσεις (Σχήμα 8.5): μία μεταβλητή στο Matlab μπορεί να θεωρηθεί σαν ένας πίνακας μηδενικής διάστασης, δηλαδή πίνακας-στοιχείο. Αν συνδυάσουμε πολλά τέτοια στοιχεία σε μία διάσταση, προκύπτει η δομή που καλούμε διάνυσμα, δηλαδή μονοδιάστατος πίνακας. Αν τοποθετήσουμε πολλά διανύσματα το ένα δίπλα στο άλλο, προκύπτει ένας διδιάστατος πίνακας. Με τη λογική αυτή, αν κατασκευάσουμε μία ακολουθία από διδιάστατους πίνακες, η δομή που προκύπτει είναι ένας τριδιάστατος πίνακας. Η διαδικασία μπορεί να επεκταθεί και σε περισσότερες διαστάσεις, παρόλο που δεν μπορούμε να απεικονίσουμε έναν τέτοιο πίνακα με κάποιο γεωμετρικό ανάλογο του τριδιάστατου χώρου.

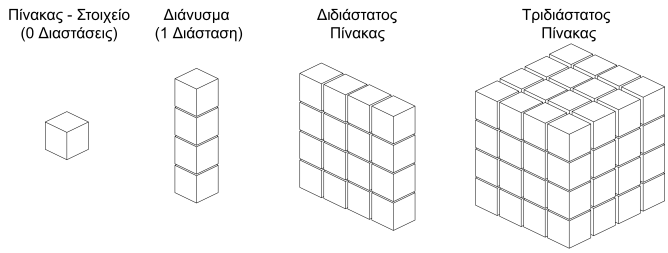
Πέρα, όμως, από τη γεωμετρική θεώρηση του πίνακα, αυτό το οποίο μας έχει απασχολήσει στα παραδείγματα του κεφαλαίου αυτού, είναι οι εφαρμογές των διανυσμάτων και των διδιάστατων πινάκων. Για παράδειγμα, οι θέσεις ενός διανύσματος μπορεί να αντιστοιχούν σε διαδοχικές μετρήσεις (π.χ. θερμοκρασίας) που ελήφθησαν σε ένα χρονικό διάστημα. Επίσης, τα στοιχεία ενός πίνακα μπορεί να είναι οι τιμές μιας μεταβλητής σε διαφορετικά σημεία μιας περιοχής. Γενικά, οι πίνακες δίνουν τη δυνατότητα να προσεγγίσουμε τις συνεχείς κατανομές στον χώρο και στον χρόνο με ένα σύνολο διακριτών στοιχείων. Έτσι, παρόλο που η θερμοκρασιακή μεταβολή στη διάρκεια ενός χρονικού διαστήματος είναι συνεχής, εμείς μπορούμε να την αντιμετωπίσουμε σαν μία ακολουθία από διακριτές τιμές, τις οποίες έχουμε λάβει από τις συνεχόμενες μετρήσεις ενός θερμομέτρου.

8.8.1 Κατασκευή πινάκων

Με βάση τα παραπάνω, γίνεται κατανοητό ότι οι πίνακες μπορούν να χρησιμοποιηθούν για να περιγράψουν φυσικά μεγέθη τα οποία μεταβάλλονται στον χώρο και στον χρόνο. Είναι, λοιπόν, αναμενόμενο ότι οι δύο διαστάσεις δεν είναι αρκετές για να περιγράψουν μια μεταβλητή που αντιστοιχεί σε ένα μέγεθος του τριδιάστατου χώρου. Για να μπορέσουμε να το περιγράψουμε, χρειαζόμαστε μία τριδιάστατη δομή πίνακα. Αυτό μπορούμε να το επιτύχουμε αν συνδυάσουμε πολλούς διδιάστατους πίνακες, οι οποίοι, αν θεωρηθούν σαν κάτι ενιαίο, δημιουργούν μία τριδιάστατη δομή. Για παράδειγμα, οι μετρήσεις της θερμοκρασίας σε διάφορα - ομοιόμορφα κατανεμημένα - σημεία ενός δωματίου μπορούν να εισαχθούν σε έναν τριδιάστατο πίνακα. Για να γίνει αυτό, πρέπει είτε να εισάγουμε τα δεδομένα σαν μια ακολουθία από διδιάστατους πίνακες είτε να δημιουργήσουμε μία τριδιάστατη δομή με μία από τις γνωστές εντολές δημιουργίας πινάκων (`zeros`, `ones`, `rand` κ.λπ.). Ένα παράδειγμα εισαγωγής δεδομένων φαίνεται παρακάτω: Αρχικά, δημιουργούμε έναν διδιάστατο πίνακα A με μέγεθος 2×3.

```
>> A = [1,2,3;4,5,6]

A =

     1     2     3
     4     5     6
```

Αυτό είναι το πρώτο επίπεδο του τριδιάστατου πίνακα. Στη συνέχεια, εισάγουμε με αντίστοιχο τρόπο το δεύτερο επίπεδο. Για να δηλώσουμε στο Matlab ότι πρόκειται για τριδιάστατο πίνακα, χρησιμοποιούμε τρεις δείκτες στο όρισμα του A. Αφού θέλουμε να εισάγουμε τα στοιχεία του 2ου επιπέδου, χρησιμοποιούμε τον



Σχήμα 8.5: Μπορούμε να αντιληφθούμε τα στοιχεία, τα διανύσματα, τους διδιάστατους και τους τριδιάστατους πίνακες σαν μία διαδοχική επέκταση της έννοιας του πίνακα σε περισσότερες διαστάσεις.

δείκτη 2, όπως φαίνεται παρακάτω.

```
>> A(:,:,2) = [11,12,13;14,15,16]

A(:,:,1) =

     1     2     3
     4     5     6


A(:,:,2) =

    11    12    13
    14    15    16
```

Με την παραπάνω εντολή, τοποθετήσαμε τον διδιάστατο πίνακα του δεξιού μέλους στο δεύτερο επίπεδο του τριδιάστατου πίνακα, ο οποίος πλέον έχει μέγεθος $2 \times 3 \times 2$. Καθώς δεν υπάρχει τρόπος να παρασταθεί ο πίνακας αυτός στις δύο διαστάσεις της οθόνης, το Matlab εμφανίζει τα διαδοχικά επίπεδα του πίνακα, το ένα κάτω από το άλλο. Με αντίστοιχο τρόπο μπορούμε να εισάγουμε όσα επίπεδα θέλουμε στην τρίτη διάσταση.

Εναλλακτικά, θα μπορούσαμε να χρησιμοποιήσουμε μία από τις εντολές του Matlab για να δημιουργήσουμε τριδιάστατο πίνακα με συγκεκριμένα στοιχεία. Για παράδειγμα, για να δημιουργήσουμε έναν πίνακα με μέγεθος $2 \times 3 \times 2$ με μηδενικά πληκτρολογούμε:

```
>> A = zeros(2,3,2)

A(:,:,1) =

     0     0     0
     0     0     0


A(:,:,2) =

     0     0     0
     0     0     0
```

Οι τριδιάστατοι πίνακες, όπως οι παραπάνω, μπορούμε να θεωρήσουμε ότι περιέχουν τις τιμές μίας τριδιάστατης μεταβλητής. Για παράδειγμα, οι τιμές ενός τέτοιου πίνακα μπορεί να αντιστοιχούν στην κατανομή της θερμοκρασίας σε ένα δωμάτιο. Αν θεωρήσουμε ότι η θερμοκρασία του δωματίου

μεταβάλλεται καθώς περνάει ο χρόνος, τότε για την περιγραφή της θα χρειαστούμε κατανομές τις θερμοκρασίας σε διαδοχικές χρονικές στιγμές. Για να συμπεριλάβουμε όλες τις κατανομές σε μία μόνο δομή, θα χρειαστούμε έναν πίνακα τεσσάρων διαστάσεων (τρεις διαστάσεις για τον χώρο και μία για τον χρόνο). Ο πίνακας αυτός κατασκευάζεται με αντίστοιχο τρόπο με τον τριδιάστατο: είτε προσθέτουμε δείκτες στο όρισμα του πίνακα είτε χρησιμοποιούμε μία από τις γνωστές εντολές δημιουργίας πινάκων (zeros, ones, rand κ.λπ.). Για παράδειγμα, η εντολή

```
>> A = rand(2,3,2,4);
```

κατασκευάζει έναν πίνακα με διαστάσεις $2 \times 3 \times 2 \times 4$.

Με την παραπάνω τεχνική μπορούμε να κατασκευάσουμε έναν πίνακα με τον επιθυμητό αριθμό διαστάσεων, έτσι ώστε να περιγράψουμε μία κατανομή που εξαρτάται από πολλές μεταβλητές. Ωστόσο, από ένα σημείο και μετά οι πίνακες αρχίζουν να γίνονται δύσχρηστοι, λόγω των πολλών διαστάσεων, οπότε είναι καλύτερο να χρησιμοποιούμε ξεχωριστούς πίνακες για τις εφαρμογές αυτές.

8.8.2 Πληροφορίες για το μέγεθος πίνακα και τα στοιχεία του

Οι βασικές εντολές που έχουμε δει μέχρι στιγμής για τα διανύσματα και τους διδιάστατους πίνακες εφαρμόζονται και για τους πίνακες περισσότερων διαστάσεων. Για να δούμε μερικά παραδείγματα, ας κατασκευάσουμε έναν πίνακα A μεγέθους $2 \times 3 \times 2$ με τυχαίους ακέραιους αριθμούς μεταξύ του 1 και του 5.

```
>> A = randi([1,5],2,3,2)

A(:,:,1) =

     3     2     5
     1     2     1


A(:,:,2) =

     1     4     4
     1     4     3
```

Μπορούμε να εξάγουμε το μέγεθος του πίνακα με την εντολή `size`.

```
>> size(A)

ans =

     2     3     2
```

Το αποτέλεσμα της εντολής `size` είναι ένα διάνυσμα τριών θέσεων με το μήκος κάθε διάστασης του πίνακα A.

Για να βρούμε το άθροισμα όλων των στοιχείων του πίνακα, θα πρέπει να χρησιμοποιήσουμε την τριπλή εντολή `sum` ως εξής:

```
>> sum( sum( sum(A) ) )

ans =

    31
```

Άλλες εντολές, όπως η `find`, είναι πιο σύνθετες, όταν εφαρμόζονται σε πολυδιάστατους πίνακες, και η περιγραφή τους είναι πέρα από τον σκοπό του συγγράμματος αυτού.

Παράδειγμα 8.12

Δημιουργήστε έναν τριδιάστατο πίνακα με μέγεθος $4 \times 4 \times 4$, του οποίου

- τα ακριανά στοιχεία κάθε διάστασης θα έχουν τιμή 1,
- όλα τα εσωτερικά σημεία θα έχουν τιμή μηδέν.

Λύση Παραδείγματος 8.12

Για να λύσουμε το πρόβλημα, θα κατασκευάσουμε τον πίνακα A με μέγεθος $4 \times 4 \times 4$, τον οποίο θα αρχικοποιήσουμε με μηδενικά.

```
>>  A = zeros(4,4,4);
```

Στη συνέχεια, θα πρέπει να θέσουμε τα στοιχεία που βρίσκονται στην πρώτη και τελευταία (δηλαδή 4η) θέση κάθε διάστασης ίσα με τη μονάδα.

```
>> A(1,:,:) = 1;
>> A(4,:,:) = 1;
>> A(:,1,:) = 1;
>> A(:,4,:) = 1;
>> A(:,:,1) = 1;
>> A(:,:,4) = 1;
```

Με τον τρόπο αυτό, αν φανταστούμε τον τριδιάστατο πίνακα σαν έναν κύβο αποτελούμενο από σημεία, όλα τα στοιχεία που βρίσκονται σε ακριανές θέσεις έχουν τιμή μηδέν, ενώ όλα τα εσωτερικά έχουν τιμή ένα. Το αποτέλεσμα αυτό επιβεβαιώνεται αν εκτυπώσουμε τον πίνακα A.

```
>> disp(A)
(:,:,1) =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
(:,:,2) =
     1     1     1     1
     1     0     0     1
     1     0     0     1
     1     1     1     1
(:,:,3) =
     1     1     1     1
```

```
     1     0     0     1
     1     0     0     1
     1     1     1     1
(:,:,4) =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

Παρατηρούμε ότι το πρώτο και το τελευταίο (4ο) επίπεδο του A έχει μονάδες, όπως και τα ακριανά σημεία των εσωτερικών επιπέδων. Τα ενδιάμεσα έχουν τιμή μηδεν.

Άσκηση αυτοαξιολόγησης 8.4

Δοκιμάστε να λύσετε το Παράδειγμα 8.12, ορίζοντας πρώτα έναν πίνακα με μονάδες και, στη συνέχεια, μηδενίζοντας τα εσωτερικά του σημεία.

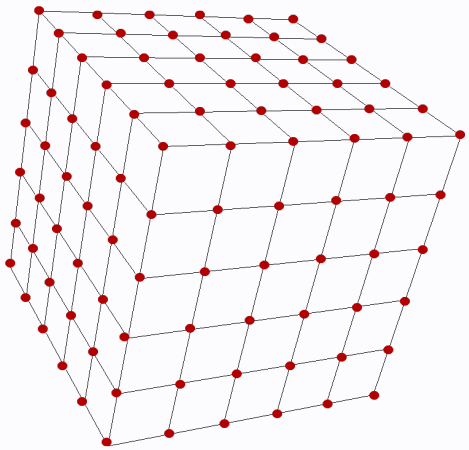
Παράδειγμα 8.13

Σε πολλές μηχανολογικές εφαρμογές μας ενδιαφέρει να υπολογίσουμε τον χρόνο που χρειάζεται ένα σώμα για να θερμανθεί μέχρι μία επιθυμητή θερμοκρασία ή, αντίστροφα, ο χρόνος που χρειάζεται ένα σώμα για να ψυχθεί. Για να μπορέσουμε να λύσουμε ένα τέτοιο πρόβλημα στον υπολογιστή, πρέπει να θεωρήσουμε ότι το σώμα αποτελείται από διακριτά σημεία στα οποία υπολογίζουμε τη θερμοκρασία. Η διαδικασία αυτή ονομάζεται «διακριτοποίηση».

Για παράδειγμα, αν το σώμα που εξετάζουμε είναι ένας μεταλλικός κύβος, τότε μπορούμε να θεωρήσουμε ένα πλέγμα από $20 \times 20 \times 20$ σημεία, το οποίο προσεγγίζει τη γεωμετρία του κύβου (Σχήμα 8.6). Η θερμοκρασία T σε κάθε σημείο του πλέγματος μία δεδομένη χρονική στιγμή, εξαρτάται από τη θερμοκρασία Told που είχε το σημείο αυτό, καθώς και τα γειτονικά του σημεία την προηγούμενη χρονική στιγμή. Ο τύπος που εκφράζει αυτή τη συσχέτιση για ένα τυχαίο σημείο (i,j,k) του πλέγματος είναι

$$\begin{aligned} T(i,j,k) = (1 - 6\,s)\,Told(i,j,k) + \; & s\,(\,Told(i+1,j,k) + Told(i-1,j,k) + \\ & + Told(i,j+1,k) + Told(i,j-1,k) + \\ & + Told(i,j,k+1) + Told(i,j,k-1)\,) \end{aligned}$$

όπου s είναι μια σταθερά που εξαρτάται από το πλέγμα, το χρονικό βήμα και το υλικό του σώματος.



Σχήμα 8.6: Ο κύβος μπορεί να θεωρηθεί σαν ένα πλέγμα από διακριτά σημεία. Οι θερμοκρασίες στα σημεία αυτά μπορούν να παρασταθούν από έναν πίνακα με τρεις διαστάσεις.

Για το παράδειγμα αυτό, θεωρήστε έναν θερμό μεταλλικό κύβο που βρίσκεται στους 500^oC. Για να τον ψύξουμε, τον τοποθετούμε σε δοχείο με νερό. Αν θεωρήσουμε ότι η θερμοκρασία στα σημεία επαφής του κύβου με το νερό παραμένει σταθερή στους 100^oC, λόγω εξάτμισης, να υπολογίσετε σε πόση ώρα θα έχει ψυχθεί ο κύβος, δηλαδή θα έχει φτάσει η θερμοκρασία σε όλα τα σημεία του κάτω από τους 110^oC. Δίνεται η σταθερά s = 0.01, και επίσης θεωρήστε ότι δύο διαδοχικές χρονικές στιγμές διαφέρουν κατά 1 sec.

Λύση Παραδείγματος 8.13

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστούμε τον πίνακα T, στον οποίον θα αποθηκευτούν οι θερμοκρασίες των κόμβων του κύβου. Οι τιμές των στοιχείων του πίνακα είναι 100 στην πρώτη και τελευταία θέση κάθε διάστασης, καθώς αντιστοιχούν στα σημεία επαφής του κύβου με το νερό, ενώ όλα τα ενδιάμεσα (εσωτερικά) στοιχεία έχουν την τιμή 500. Για να κατασκευάσουμε έναν τέτοιο τριδιάστατο πίνακα, μπορούμε είτε να δημιουργήσουμε έναν πίνακα όπου όλα τα στοιχεία έχουν την τιμή 500 και, στη συνέχεια, να μεταβάλλουμε τα αρχικά και τελικά σημεία κάθε διάστασης, ώστε να έχουν την τιμή 100, είτε να ακολουθήσουμε την αντίστροφη διαδικασία. Στη συγκεκριμένη περίπτωση, θα επιλέξουμε τον δεύτερο τρόπο, καθώς είναι πιο σύντομος, προγραμματιστικά. Θα δημιουργήσουμε, δηλαδή έναν πίνακα $20 \times 20 \times 20$ με την εντολή `ones` και θα τον πολλαπλασιάσουμε με 100, ώστε όλα τα στοιχεία του να έχουν αυτή την τιμή.

```
T = 100*ones(20,20,20);
```

Έπειτα, θα επιλέξουμε τα εσωτερικά στοιχεία, δηλαδή από το 2 έως το 19, σε όλες τις διαστάσεις, και θα τα θέσουμε ίσα με 500.

```
T(2:19,2:19,2:19) = 500;
```

Στη συνέχεια, θα δημιουργήσουμε τον πίνακα Told, τον οποίο θέτουμε ίσο με τον T. Τέλος, χρειαζόμαστε τις μεταβλητές s, η οποία είναι σταθερά με τιμή 0.01, και τη μεταβλητή για τον χρόνο t, την οποία αρχικοποιούμε με μηδέν.

Κύριος κώδικας: Στον κύριο κώδικα θα πρέπει να υπολογίσουμε πώς μεταβάλλεται η θερμοκρασία στα εσωτερικά σημεία του κύβου, καθώς περνάει ο χρόνος. Για τον λόγο αυτό, θα χρειαστούμε, αρχικά, έναν επαναληπτικό βρόχο για τον χρόνο, ο οποίος θα επαναλαμβάνεται όσο υπάρχει κάποιο στοιχείο του πίνακα T, το οποίο στοιχείο να έχει τιμή μεγαλύτερη από 110, σύμφωνα με την εκφώνηση. Εφόσον δεν γνωρίζουμε τον αριθμό των επαναλήψεων, θα προτιμήσουμε την εντολή while για την επαναληπτική διαδικασία και η συνθήκη που θα χρησιμοποιήσουμε θα πρέπει να ελέγχει αυτόματα όλα τα στοιχεία του T. Για να γίνει αυτό, χρειαζόμαστε την εντολή `find`, η οποία μπορεί να χρησιμοποιηθεί ως εξής:

```
while ~isempty( find( T>110 ))
```

Για να αναλύσουμε την παραπάνω γραμμή κώδικα, μπορούμε να πούμε ότι

- η εντολή find(T>110) επιστρέφει ένα διάνυσμα με όλες τις θέσεις του T, στις οποίες υπάρχει τιμή μεγαλύτερη του 110,
- η εντολή isempty(find(T>110)) ελέγχει αν το παραπάνω διάνυσμα είναι κενό ή όχι,
- η εντολή while isempty(find(T>110)) συνεχίζει την επαναληπτική διαδικασία, όσο το παραπάνω διάνυσμα *δεν* είναι κενό. Με άλλα λόγια, η διαδικασία θα σταματήσει όταν *όλα* τα στοιχεία του T έχουν τιμή κάτω από 110.

Είδαμε, λοιπόν, ότι ο βρόχος για τον χρόνο εκφράζει την εξέλιξη του φαινομένου, καθώς περνάνε τα δευτερόλεπτα. Σύμφωνα με τη λογική αυτή, σε κάθε επανάληψη θα πρέπει να αυξάνουμε την τιμή

του χρόνου που έχει περάσει κατά ένα δευτερόλεπτο και να ανανεώνουμε την τιμή της θερμοκρασίας θέτοντας τον πίνακα Told ίσο με T.
Ο βρόχος για τον χρόνο θα πρέπει, επίσης, να υπολογίζει τη νέα θερμοκρασία σε όλα τα εσωτερικά σημεία του κύβου. Θα πρέπει, λοιπόν, να περιλαμβάνει εμφωλευμένους βρόχους, που να διατρέχουν όλα τα εσωτερικά στοιχεία του πίνακα T, δηλαδή τα στοιχεία από 2 έως 19. Τα στοιχεία που βρίσκονται στις αρχικές και τελικές θέσεις του T δεν επηρεάζονται, καθώς έχουν ήδη τιμή 100 από την αρχικοποίηση (χαμηλή θερμοκρασία λόγω επαφής με το νερό). Άρα, χρειαζόμαστε τρεις επαναληπτικούς βρόχους for, οι οποίοι θα διατρέχουν τα στοιχεία από 2 έως 19 σε όλες τις διαστάσεις, ώστε να καλύψουν το σύνολο των εσωτερικών σημείων του κύβου. Σε καθένα από τα στοιχεία αυτά θα υπολογίζεται η τιμή της θερμοκρασίας T με βάση τις θερμοκρασίες Told στο ίδιο και στα γειτονικά στοιχεία, σύμφωνα με τη σχέση που δίνεται στην εκφώνηση.
Μόλις οι θερμοκρασίες σε όλα τα σημεία του κύβου γίνουν μικρότερες του 110, ο εξωτερικός βρόχος θα τερματιστεί και θα εκτυπωθεί το αποτέλεσμα με κατάλληλο μήνυμα.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%PSYKSI METALLIKOY KYVOU
% Initialization
clear all
T = 100*ones(20,20,20);
T(2:19,2:19,2:19) = 500;
Told = T;
s = 0.01;
t = 0;

% Main code

while ~isempty( find( T>110 ))
  % AYKSISI XRONOU
   t = t + 1;
   Told = T;

  % YPOLOGISMOS PSYKSIS
   for i = 2:19
     for j = 2:19
       for k = 2:19
         T(i,j,k) = (1-6*s)*Told(i,j,k) + ...
             s*(Told(i+1,j,k) + Told(i-1,j,k) + ...
                 Told(i,j+1,k) + Told(i,j-1,k) + ...
                   Told(i,j,k+1) + Told(i,j,k-1));
       end
     end
   end

end

disp(['Gia tin psyksi xreiazontai ',num2str(t),' sec'])
```

Η εκτέλεση του κώδικα επιστρέφει το μήνυμα

```
Gia tin psyksi xreiazontai 5371 sec
```

το οποίο μας πληροφορεί ότι χρειάζονται 5371 δευτερόλεπτα για να ψυχθεί ο κύβος, δηλαδή να φτάσει η θερμοκρασία σε όλα τα σημεία του κάτω από τους 110^oC.

Παρατήρηση 8.15

Στη λύση του Παραδείγματος 8.13 χρησιμοποιήθηκαν τρεις εμφωλευμένοι επαναληπτικοί βρόχοι for, με σκοπό να υπολογιστούν οι απαιτούμενες τιμές για κάθε στοιχείο του τριδιάστατου πίνακα T. Ένας εναλλακτικός τρόπος, θα ήταν να χρησιμοποιήσουμε πράξεις υποπινάκων στοιχείο προς στοιχείο, προκειμένου να αποφύγουμε τον τριπλό βρόχο for.

Γενικότερα, η πρακτική αντικατάστασης βρόχων με εκφράσεις διανυσμάτων ή πινάκων αναφέρεται συνήθως ως «διανυσματοποίηση» (vectorization) και είναι βασικό στοιχείο του προγραμματισμού με διανύσματα (Harris *et al.*, 2020). Η ικανότητα του Matlab να εκτελεί λειτουργίες σε ολόκληρους πίνακες δίνει τη δυνατότητα για χρησιμοποίηση του προγραμματισμού με διανύσματα, δημιουργώντας καθαρότερο και (συνήθως) ταχύτερο κώδικα. Αν και στο Matlab υπάρχει πολύ καλή διαχείριση για τους βρόχους (π.χ. ο επιταχυντής just-in-time (JIT)), ο διανυσματικός προγραμματισμός προσφέρει καλύτερη απόδοση στη χρήση πίνακα μέσα σε συνάρτηση, καθώς και τη δυνατότητα εκτέλεσης πράξεων στην κάρτα γραφικών (GPU)) - εφόσον η κάρτα έχει αυτή τη δυνατότητα - μέσω της δημιουργίας διανυσμάτων στην κάρτα γραφικών (gpuArray).

Άσκηση αυτοαξιολόγησης 8.5

Δοκιμάστε να αντικαταστήσετε τις εντολές των γραμμών 18-27 της λύσης του Παραδείγματος 8.13 με μία γραμμή κώδικα που θα υλοποιεί πράξεις μεταξύ υποπινάκων.

8.9 Ασκήσεις

Άσκηση 8.1. *Μια προμηθεύτρια εταιρεία προμηθεύει με υλικά 3 κατασκευαστικές εταιρείες. Το δελτίο παραγγελίας του τρέχοντος μηνός των 3 αυτών εταιρειών παρουσιάζεται στη συνέχεια.*

	Τσιμέντο (κ.μ.)	*Πλακάκια (τ.μ.)*	*Καλώδια (μ.)*
Εταιρεία Α	*1010*	*200*	*200*
Εταιρεία Β	*500*	*100*	*50*
Εταιρεία Γ	*750*	*210*	*20*

Υπολογίστε τον συνολικό αριθμό κ.μ. τσιμέντου, τ.μ. πλακακιών και μ. καλωδίου που πούλησε η εταιρεία με χρήση (α) επαναληπτικών μεθόδων (for) και (β) της εντολής (sum).

Με βάση την τρέχουσα τιμή των υλικών η εταιρεία κερδίζει 5€ για κάθε κ.μ. τσιμέντου. Τα αντίστοιχα ποσά για κάθε τ.μ. πλακακίου και κάθε μέτρο καλωδίου είναι 10€ και 3€, αντίστοιχα. Υπολογίστε το κέρδος που βγάζει η εταιρεία από κάθε κατασκευαστική εταιρεία κάνοντας χρήση (α) επαναληπτικών μεθόδων (for) και (β) του πολλαπλασιασμού πινάκων.

Άσκηση 8.2. *Δημιουργήστε ένα αρχείο script, το οποίο αρχικά θα ορίζει και θα αναθέτει στη μεταβλητή NMAX την τιμή 6. Ακολούθως, θα ζητάει από τον χρήστη να ορίσει το πλήθος των γραμμών N και των στηλών K του πίνακα τυχαίων φυσικών αριθμών που θέλει να παράγει με χρήση της εντολής randi(NMAX,N,K). Στη συνέχεια, θέλουμε το πρόγραμμα:*

(α) να παράγει δύο τέτοιους τυχαίους πίνακες,

(β) να υπολογίζει το άθροισμά τους και

(γ) να εξετάζει αν ο αριθμός 3 ή 12 έχει εμφανιστεί τουλάχιστον μία φορά. Στη συνέχεια, να τυπώνει αυτήν την πληροφορία, καθώς και το πλήθος των φορών που εμφανίστηκε ο κάθε αριθμός.

Άσκηση 8.3. *Δημιουργήστε ένα αρχείο script, το οποίο θα ζητάει από τον χρήστη να εισάγει δύο πίνακες. Στη συνέχεια, θα εξετάζει αν το πλήθος των στηλών του πρώτου πίνακα ισούται με το πλήθος των γραμμών του δεύτερου πίνακα. Αν αυτό αληθεύει, τότε θα υπολογίζει το γινόμενο των δύο πινάκων. Σε διαφορετική περίπτωση θα ενημερώνει τον χρήστη ότι δεν ορίζεται ο πολλαπλασιασμός μεταξύ των πινάκων που εισήγαγε.*

Άσκηση 8.4. *Δημιουργήστε ένα αρχείο script, το οποίο θα ζητάει έναν φυσικό αριθμό N. Στη συνέχεια, θα ελέγχει αν αυτός ο αριθμός είναι πράγματι φυσικός (αν N>0 και η στρογγυλοποίηση του N στον πλησιέστερο ακέραιο ισούται με το N, τότε το N είναι φυσικός) και, αν είναι, θα δημιουργεί έναν τετραγωνικό πίνακα μονάδων A. Στην περίπτωση που το N δεν είναι φυσικός αριθμός, το πρόγραμμα θα πρέπει να ζητάει επανειλημμένως από τον χρήστη έναν φυσικό αριθμό μέχρις ότου αυτός το πράξει. Στη συνέχεια, το πρόγραμμα θα πρέπει να κατασκευάζει έναν ταυτοτικό πίνακα I με ίδιο μέγεθος με τον A και να τον πολλαπλασιάζει με τον A.*

Άσκηση 8.5. *Δημιουργήστε ένα αρχείο script, που θα δημιουργεί έναν πίνακα A με μέγεθος 5×5, με ψευδοτυχαίους, ακέραιους αριθμούς μεταξύ του -20 και του 20. Στη συνέχεια, χωρίστε τον A σε δύο πίνακες 5×5, εκ των οποίων ο ένας θα έχει μόνο τα θετικά στοιχεία του A, στις ίδιες θέσεις που υπάρχουν και στον A, και ο δεύτερος μόνο τα αρνητικά. Εκτυπώστε το αποτέλεσμα για να επιβεβαιώσετε ότι έχετε χωρίσει σωστά τους πίνακες.*

8.10 Απαντήσεις στις ασκήσεις αυτοαξιολόγησης

Λύση άσκησης αυτοαξιολόγησης 8.1

Θα πρέπει να υπολογίσουμε το κατάλληλο βήμα, ώστε να καταλήξουμε στο ίδιο αποτέλεσμα με τη λύση του Παραδείγματος 8.1. Σύμφωνα με την Παρατήρηση 8.1, αφού τα σημεία που ζητάει η άσκηση είναι 41, τα υποδιαστήματα θα είναι 40. Άρα το βήμα θα είναι 1/40. Η εντολή που θα χρησιμοποιήσουμε είναι

```
>> A = 0:1/40:1;
```

Στη συνέχεια, μπορούμε να κατασκευάσουμε το διάνυσμα B χρησιμοποιώντας τις τιμές 1, 21 και 41, όπως και στο Παράδειγμα 8.1.

```
>> B = A([1,21,41])

B =

         0    0.5000    1.0000
```

Παρατηρούμε ότι οι τρεις τιμές του B είναι ίδιες με αυτές που βρήκαμε στη λύση του Παραδείγματος 8.1, οπότε μπορούμε να συμπεράνουμε ότι κατασκευάσαμε σωστά *και* το διάνυσμα A.

Λύση άσκησης αυτοαξιολόγησης 8.2

Το διάνυσμα που θέλουμε να κατασκευάσουμε έχει επτά σετ τιμών, εκ των οποίων καθένα έχει 4 τιμές. Το πρώτο σετ, δηλαδή το 0.01, 0.02, 0.04 και 0.07, έχει εισαχθεί στο διάνυσμα με απευθείας ορισμό. Για την εισαγωγή των υπόλοιπων σετ αριθμών μπορούμε να μετατρέψουμε τις επαναλαμβανόμενες εντολές σε επαναληπτική διαδικασία χρησιμοποιώντας κατάλληλο μετρητή. Ο μετρητής θα ξεκινάει από το δεύτερο σετ τιμών, δηλαδή από τη θέση 5, και θα μεταβάλλεται με βήμα 4, όσες είναι, δηλαδή, και οι τιμές που έχει κάθε σετ.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %GRAFIKI PARASTASI THS y=log(x+3)
2  % Initialization
3  clear all
4  format short
5  xlog = zeros(1,29);
6  xlog(1:4) = [0.01 0.02 0.04 0.07];
7  for n=5:4:25
8    xlog(n:n+3) = 10*xlog(n-4:n-1);
9  end
10 xlog(29) = 10^5;
11
12 y = log10(xlog+3);
13
14 % Main code
15 subplot(1,2,1)
16 plot(xlog,y,'o')
17 title('y = log(x+3) - Grammikoi aksones')
18 xlabel('x')
```

```
19    ylabel('y')
20
21    subplot(1,2,2)
22    loglog(xlog,y,'o')
23    title('y = log(x+3) - Logarithmikoi aksones')
24    xlabel('x')
25    ylabel('y')
```

Λύση άσκησης αυτοαξιολόγησης 8.3

Ας υποθέσουμε, για απλότητα, ότι ο τριδιαγώνιος πίνακας που θέλουμε να δημιουργήσουμε έχει μέγεθος 5×5. Το πρώτο βήμα για τη λύση της άσκησης είναι να ορίσουμε τα στοιχεία της κύριας διαγωνίου ίσα με -2. Αυτό μπορεί να γίνει με την εντολή `eye` ως εξής

```
>> A = -2*eye(5)

A =

    -2     0     0     0     0
     0    -2     0     0     0
     0     0    -2     0     0
     0     0     0    -2     0
     0     0     0     0    -2
```

Στη συνέχεια, θα πρέπει να εισάγουμε τις τιμές που βρίσκονται πάνω από την κύρια διαγώνιο. Για να το επιτύχουμε αυτό χωρίς τη χρήση επαναληπτικού βρόχου, θα πρέπει να χρησιμοποιήσουμε υποπίνακες. Συγκεκριμένα, παρατηρούμε στο Σχ. 8.7(α) ότι τα στοιχεία που βρίσκονται πάνω από την κύρια διαγώνιο περιλαμβάνονται όλα μέσα σε ένα τμήμα πίνακα με μέγεθος 4×4 και, μάλιστα, καταλαμβάνουν την κύρια διαγώνιο αυτού του υποπίνακα. Μπορούμε, λοιπόν, να τα εισάγουμε προσθέτοντας στο συγκεκριμένο 4×4 τμήμα του πίνακα A έναν νέο ταυτοτικό πίνακα. Η εντολή που θα χρησιμοποιήσουμε είναι

```
>> A(1:4,2:5) = A(1:4,2:5) + eye(4)

A =

    -2     1     0     0     0
     0    -2     1     0     0
     0     0    -2     1     0
     0     0     0    -2     1
     0     0     0     0    -2
```

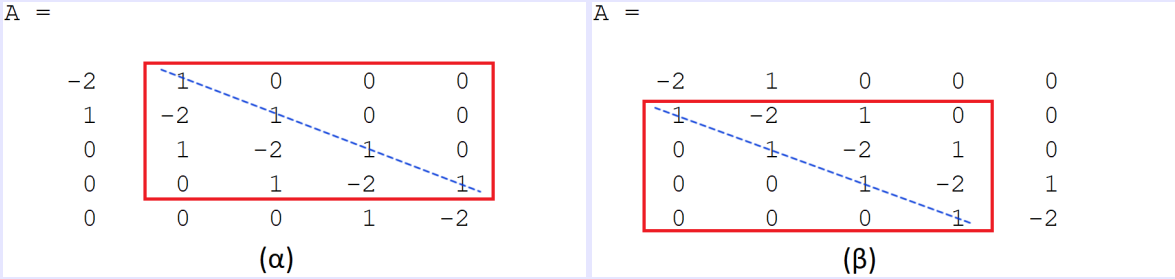
Για να ολοκληρώσουμε τη λύση, θα κάνουμε την αντίστοιχη διαδικασία για τα στοιχεία που βρίσκονται κάτω από την κύρια διαγώνιο. Όπως και προηγουμένως, τα στοιχεία αυτά καταλαμβάνουν την κύρια διαγώνιο ενός 4×4 υποπίνακα (Σχήμα 8.7(β)). Για να εισάγουμε την τιμή 1, θα πρέπει να προσθέσουμε στον συγκεκριμένο υποπίνακα του A έναν νέο ταυτοτικό πίνακα. Η εντολή που θα χρησιμοποιήσουμε είναι

```
>> A(2:5,1:4) = A(2:5,1:4) + eye(4)

A =

    -2     1     0     0     0
     1    -2     1     0     0
     0     1    -2     1     0
     0     0     1    -2     1
     0     0     0     1    -2
```

Με την εντολή αυτή ολοκληρώνεται η κατασκευή του τριδιαγώνιου πίνακα, χωρίς επαναληπτική διαδικασία.

Θα μπορούσαμε να γενικεύσουμε τη συγκεκριμένη λύση για οποιοδήποτε μέγεθος πίνακα, όπως φαίνεται στον κώδικα που ακολουθεί:

```
1  %KATASKEYH TRISDIAGWNIOU PINAKA
2  % Initialization
3   clear all
4   N = input('Give matrix size N: ');
5   A = -2*eye(N);
6   A(1:N-1,2:N) = A(1:N-1,2:N) + eye(N-1);
7   A(2:N,1:N-1) = A(2:N,1:N-1) + eye(N-1);
8
9
10  % Main code
11    disp(A)
```



Σχήμα 8.7: Τριδιαγώνιος πίνακας του Παραδείγματος 8.11. (α) Τα στοιχεία που βρίσκονται πάνω από την κύρια διαγώνιο περιλαμβάνονται μέσα σε έναν υποπίνακα 4×4 και καταλαμβάνουν την κύρια διαγώνιό του. (β) Τα στοιχεία που βρίσκονται κάτω από την κύρια διαγώνιο περιλαμβάνονται μέσα σε έναν υποπίνακα 4×4 και καταλαμβάνουν την κύρια διαγώνιό του.

Λύση άσκησης αυτοαξιολόγησης 8.4

Με τον διανυσματικό προγραμματισμό μπορούμε να αποφύγουμε την τριπλή επανάληψη, που χρησιμοποιούμε για να υπολογίσουμε ένα-ένα τα στοιχεία του πίνακα Τ. Για να γίνει αυτό, θα πρέπει να χρησιμοποιήσουμε υποπίνακες, αφού τα στοιχεία του Τ που θα αλλάζουν καθώς περνάει ο χρόνος, είναι μόνο τα εσωτερικά. Για να αποφύγουμε τις πολύπλοκες παραστάσεις μέσα στο όρισμα του πίνακα, μπορούμε να ορίσουμε και να χρησιμοποιήσουμε κατάλληλα διανύσματα. Έτσι, αντί να γράψουμε

```
T( 2:19 , 2:19 , 2:19 ) = ...
```

μπορούμε να ορίσουμε το διάνυσμα

```
P = 2:19;
```

και να το χρησιμοποιήσουμε ως όρισμα στον T, ως εξής

```
T( P , P , P ) = ...
```

Με αντίστοιχο τρόπο μπορούμε να ορίσουμε τα γειτονικά σημεία. Για παράδειγμα, αντί να γράψουμε

```
... Told( 1:18 , 2:19 , 2:19 ) = ...
```

μπορούμε να ορίσουμε το διάνυσμα

```
B = 1:18;
```

και να το χρησιμοποιήσουμε ως εξής

```
... Told( B , P , P ) = ...
```

Με την τεχνική αυτή ο κώδικας είναι ευανάγνωστος κι εύκολα ελέγξιμος.
Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %PSYKSI METALLIKOY KYVOU
2  % Initialization
3   clear all
4   T = 100*ones(20,20,20);
5   T(2:19,2:19,2:19) = 500;
6   Told = T;
7   s = 0.01;
8   t = 0;
9
10  % Main code
11
12  while ~isempty( find( T>110 ))
13    % AYKSISI XRONOU
14     t = t + 1;
15     Told = T;
16
17    % YPOLOGISMOS PSYKSIS
18     B = 1:18;
19     P = 2:19;
20     F = 3:20;
21
22     T(P,P,P) = (1-6*s)*Told(P,P,P) + ...
23               s*(Told(F,P,P) + Told(B,P,P) + ...
24                   Told(P,F,P) + Told(P,B,P) + ...
25                     Told(P,P,F) + Told(P,P,B));
26  end
27  disp(['Gia tin psyksi xreiazontai ',num2str(t),' sec'])
```

Λύση άσκησης αυτοαξιολόγησης 8.5

Για να κατασκευάσουμε τον τριδιάστατο πίνακα με μονάδες, θα χρησιμοποιήσουμε την εντολή `ones`

```
>> A=ones(4,4,4);
```

Στη συνέχεια, θα μηδενίσουμε τα εσωτερικά στοιχεία, δηλαδή αυτά που βρίσκονται στις θέσεις 2 και 3, σε κάθε διάσταση του `ones`

```
>> A(2:3,2:3,2:3)=0;
```

Ο πίνακας που προκύπτει μας δίνει το ζητούμενο αποτέλεσμα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ελληνόγλωσση

Lipschutz, S. και Lipson, M. L. (2018). *Θεωρία και προβλήματα στη γραμμική άλγεβρα*. (5η έκδ.) Θεσσαλονίκη: Τζιόλας.

Strang, G. και Πάμφιλος, Π. (1996 [ανατύπωση 2002]). *Γραμμική άλγεβρα και εφαρμογές*. Πανεπιστημιακή Βιβλιοθήκη Θετικών Επιστημών / Μαθηματικά. Τίτλος πρωτοτύπου: Linear algebra and its applications, 1998. Ηράκλειο : Πανεπιστημιακές Εκδόσεις Κρήτης.

Κουτρουβέλης, Ι. Α. (2011). *Εφαρμοσμένες πιθανότητες και στατιστική*. Συμμετρία.

Σταυρουλάκης, Γ., Μουράντοβα Κονταδάκη, Α. και Σταυρουλάκη, Μ. (2015). *Υπολογιστική Μηχανική*. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις.

Φιλιππάκης, Μ. Ε. (2018). *Εφαρμοσμένη ανάλυση & στοιχεία γραμμικής άλγεβρας*. (2η έκδ.) Αθήνα: Τσότρας.

Ξενόγλωσση

Anton, H. and Rorres, C. (2000). *Elementary linear algebra :* (8th ed.) New York : Wiley.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), pp. 357–362.

Johnson, N., Kemp, A. W. and Kotz, S. (2005). *Univariate Discrete Distributions, (3nd ed.)* New York: Wiley and Sons, Inc.

Johnson, N., Kotz, S. and Balakrishnan, N. (1994). *Continuous Univariate Distributions, Vol. 1, (2nd ed.)* New York: Wiley and Sons, Inc.

Jolliffe, I. (2013). *Principal Component Analysis*. Springer Series in Statistics. Springer New York.

ΚΕΦΑΛΑΙΟ 9

ΔΗΜΙΟΥΡΓΙΑ ΣΥΝΑΡΤΗΣΕΩΝ

Σύνοψη:

Σε αυτό το κεφάλαιο παρουσιάζεται η διαδικασία κατασκευής συναρτήσεων από τον ίδιο τον χρήστη. Έμφαση δίνεται στη δομή των συναρτήσεων και τον χειρισμό των λαθών. Επιπρόσθετα, παρουσιάζονται βασικά χαρακτηριστικά των συναρτήσεων, όπως οι υποσυναρτήσεις και η αναδρομή, ενώ παρουσιάζεται και ο τρόπος δημιουργίας συναρτήσεων με μεταβλητό μήκος εισόδου ή/και εξόδου. Τέλος, γίνεται μια σύντομη αναφορά στις ανώνυμες συναρτήσεις που μπορούμε να δημιουργήσουμε στο Matlab.

Προαπαιτούμενη γνώση: Τα κεφάλαια 3-8 του παρόντος συγγράμματος.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του κεφαλαίου αυτού, θα γνωρίζετε:

- να κατασκευάζετε συναρτήσεις στο Matlab,
- να χειρίζεστε λάθη κατά την κλήση των συναρτήσεων,
- να έχετε μεταβλητό μήκος εισόδου ή/και εξόδου στις συναρτήσεις,
- να κατασκευάζετε ανώνυμες συναρτήσεις στο Matlab.

Γλωσσάριο επιστημονικών όρων

- Αναδρομή
- Ανώνυμη συνάρτηση
- Μεταβλητή εισόδου
- Μεταβλητή εξόδου
- Μεταβλητό μήκος εισόδου/εξόδου
- Υποσυνάρτηση
- Χειρισμός λαθών

9.1 Εισαγωγή

Το Matlab, όπως έχει αναφερθεί, έχει ενσωματωμένες πολλές συναρτήσεις. Ενδεικτικά αναφέρουμε ως παραδείγματα τις συναρτήσεις που εκτελούνται με τις εντολές `roots` και `abs`. Στην πραγματικότητα, οι εντολές αυτές καλούν κάποια m-αρχεία, τα οποία δέχονται κάποιες μεταβλητές εισόδου και επιστρέφουν (μεταβλητές εξόδου) κάποια αποτελέσματα μιας συγκεκριμένης διαδικασίας. Στο Matlab αυτά τα m-αρχεία ονομάζονται m-αρχεία συναρτήσεων ή απλώς αρχεία συναρτήσεων (m function files). Τα αρχεία αυτά μπορούν να εκτελούν απλές ή περίπλοκες διαδικασίες καλώντας τες με τα κατάλληλα ορίσματα. Αυτό όχι μόνο επιτρέπει στον χρήστη να μην γράφει εκτενή κώδικα αλλά, επιπλέον, και να έχει στη διάθεσή του ισχυρά εργαλεία, τα οποία μπορεί να χρησιμοποιήσει πολλές φορές, χωρίς να παρεμβαίνει στον κώδικά τους.

Το Matlab πέρα από τις ενσωματωμένες συναρτήσεις επιτρέπει τη συγγραφή συναρτήσεων και από τον χρήστη. Στη συνέχεια του κεφαλαίου θα δούμε πώς μπορούμε να δημιουργήσουμε αρχεία συναρτήσεων στο Matlab, πώς μπορούμε να εκμεταλλευτούμε κάποιες επιπλέον δυνατότητές τους και πώς μπορούμε να τα εκτελέσουμε έτσι ώστε να λάβουμε τα ζητούμενα αποτελέσματα.

9.1.1 Βασική δομή συναρτήσεων

Τα αρχεία συναρτήσεων αποτελούν μια ομάδα εντολών που εκτελούν μια συγκεκριμένη διαδικασία. Παρουσιάζουν σημαντικές ομοιότητες με τα script αρχεία που έχουμε δει στα προηγούμενα κεφάλαια αλλά και ουσιώδεις διαφορές. Τα αρχεία συναρτήσεων, όπως θα δούμε, είναι αρχεία τα οποία κάνουν χρήση:

- των ελέγχων ροής,
- των επαναληπτικών διαδικασιών και,
- γενικά, των ενσωματωμένων εντολών και συναρτήσεων του Matlab,

όπως ακριβώς κάνουν και τα script αρχεία. Από την άλλη, όμως, τα αρχεία συναρτήσεων κατασκευάζουν ένα τοπικό περιβάλλον, στο οποίο εκτελούνται όλες οι πράξεις και οι διαδικασίες και το οποίο επικοινωνεί με το υπόλοιπο Matlab μόνο μέσω των μεταβλητών εισόδου και εξόδου. Οι μεταβλητές εισόδου είναι τα ορίσματα των συναρτήσεων, δηλαδή οι τιμές εκείνες που εισάγει ο χρήστης κατά την κλήση της συνάρτησης, ενώ οι μεταβλητές εξόδου είναι τα μόνα στοιχεία που επιστρέφονται από τη διαδικασία εκτέλεσης της συνάρτησης.

Για τη δημιουργία μιας νέας συνάρτησης χρειαζόμαστε ένα m-αρχείο, στο οποίο στην πρώτη γραμμή του να πληκτρολογήσουμε

```
1 function [output_parameter_list]=function_name(input_parameter_list)
2 ...
```

- Το λεκτικό **function** πρέπει να εμφανίζεται στην αρχή της πρώτης γραμμής.
- Αν η συνάρτηση επιστρέφει περισσότερα από ένα στοιχεία, τότε αυτά πρέπει υποχρεωτικά να δηλώνονται μέσα σε αγκύλες σαν διάνυσμα **([output_parameter_list])**. Τα στοιχεία αυτά αποτελούν τις μεταβλητές εξόδου της συνάρτησης.
- Το όνομα της συνάρτησης **(function_name)** με το οποίο θα καλείται η συνάρτηση από το Matlab. Το όνομα της συνάρτησης ακολουθεί τους ίδιους κανόνες με την ονοματολογία μεταβλητών που παρουσιάστηκε στο Κεφάλαιο 1 του παρόντος βιβλίου. Σημειώνεται ότι το όνομα του m-αρχείου, στο οποίο θα αποθηκευτεί η συνάρτηση, θα πρέπει να είναι το ίδιο με το όνομα της συνάρτησης.

- Οι παράμετροι ή, αλλιώς, οι μεταβλητές εισόδου της συνάρτησης, δηλώνονται στο **([input_parameter_list])**. Οι μεταβλητές αυτές πρέπει να δηλώνονται κατά την εκτέλεση της συνάρτησης.

Για να γίνει καλύτερα κατανοητή η βασική δομή των συναρτήσεων στο Matlab, στη συνέχεια παρουσιάζεται μια πρώτη απλή συνάρτηση, σκοπός της οποίας είναι να υπολογίζει το άθροισμα και το γινόμενο δύο αριθμών που δίνει ο χρήστης κατά την κλήση της.

```
1  function [s,p] = sum_prod(a,b)
2
3  s = a+b;
4  p = a*b;
5
6  end
```

Η παραπάνω συνάρτηση:

- δέχεται δύο μεταβλητές εισόδου, τις a και b,
- το όνομα της είναι sum_prod,
- ενώ επιστρέφει δύο μεταβλητές με τα ονόματα s και p, οι οποίες αντιστοιχούν στο άθροισμα και το γινόμενο των a και b.

Ο κώδικας της συνάρτησης ολοκληρώνεται με το `end` στην τελευταία γραμμή.

Αν αποθηκεύσουμε την παραπάνω συνάρτηση σε ένα αρχείο με όνομα sum_prod.m, τότε μπορούμε να την καλέσουμε από το Command Window του Matlab, αρκεί να είναι αποθηκευμένο μέσα στον ενεργό φάκελο του Matlab. Ο τρόπος εκτέλεσης και τα αποτελέσματα που επιστρέφει η συνάρτηση για a=3 και b=5 εμφανίζονται στη συνέχεια.

```
>>[s,p] = sum_prod(3,5)
s =
     8
p =
    15
```

Η παραπάνω συνάρτηση μπορεί να εμπλουτιστεί και με ένα κείμενο βοήθειας, όπως οι ενσωματωμένες συναρτήσεις του Matlab. Η βοήθεια αυτή γράφεται αμέσως μετά την πρώτη σειρά υπό μορφή σχολίων. Έτσι, η παραπάνω συνάρτηση μπορεί να εμπλουτιστεί ως ακολούθως:

```
1  function [s,p] = sum_prod(a,b)
2  %SUM_PROD computes the sum and the product of two numbers
3  % - - - - - - - - - - - - - - -
4  %EXAMPLE
5  %>> [s,p] = sum_prod(3,5)
6  %s =
7  %     8
8  %p =
9  %    15
10 % - - - - - - - - - - - - - - -
11
```

```
12  s = a+b;
13  p = a*b;
14
15  end
```

Στην παραπάνω συνάρτηση έχουμε προσθέσει (γραμμές 2-10) τη βοήθεια που θέλουμε να εμφανίζεται για τη συνάρτηση sum_prod, όταν πληκτρολογούμε την εντολή help sum_prod στο Command Window του Matlab. Πράγματι, πληκτρολογώντας help sum_prod εμφανίζονται τα ακόλουθα στοιχεία:

```
>> help sum_prod
 sum_prod computes the sum and the product of two numbers
 - - - - - - - - - - - - - - -
 EXAMPLE
 >> [s,p] = sum_prod(3,5)
 s =
      8
 p =
     15
 - - - - - - - - - - - - - - -
```

Σημειώνεται ότι στο κείμενο της βοήθειας για τη συνάρτηση sum_prod έχουμε προσθέσει, χωρίς αυτό να είναι προαπαιτούμενο, και ένα παράδειγμα, έτσι ώστε ο χρήστης να διευκολυνθεί σχετικά με τον τρόπο με τον οποίο μπορούμε να καλέσουμε τη συνάρτηση.

Παρατήρηση 9.1

Σημειώνεται ότι η κλήση της συνάρτησης γίνεται με τον ακόλουθο τρόπο

```
>>[s,p] = sum_prod(3,5)
```

δεν είναι υποχρεωτική όσον αφορά στα ονόματα των μεταβλητών εξόδου. Παραδείγματος χάριν, ο ακόλουθος τρόπος

```
>>[x,y] = sum_prod(3,5)
```

είναι αποδεκτός. Σε αυτήν την περίπτωση, στη μεταβλητή x ανατίθεται το άθροισμα του 3 και του 5, ενώ στη μεταβλητή y ανατίθεται το γινόμενο των παραπάνω αριθμών.

Παρατήρηση 9.2

Στην περίπτωση που η συνάρτηση επιστρέφει περισσότερες από μία μεταβλητές εξόδου, όπως η συνάρτηση sum_prod, τότε, αν την καλέσουμε χωρίς να αναθέσουμε στις μεταβλητές εξόδου κάποια ονόματα, δηλαδή ως

```
>> sum_prod(3,5)
```

τότε επιστρέφει μόνο την πρώτη μεταβλητή εξόδου, δηλαδή το άθροισμα στη συγκεκριμένη περίπτωση, η οποία αποτελεί τη βασική μεταβλητή εξόδου. Έτσι η παραπάνω εντολή επιστρέφει το παρακάτω αποτέλεσμα:

```
ans =
     8
```

Παρατήρηση 9.3

Το Matlab επιλέγοντας από τη γραμμή εργαλείων New⟶ Function δημιουργεί τον ακόλουθο κώδικα:

```
function [outputArg1,outputArg2] = untitled1(inputArg1,inputArg2)
%UNTITLED1 Summary of this function goes here
%   Detailed explanation goes here
outputArg1 = inputArg1;
outputArg2 = inputArg2;
end
```

Ο κώδικας αυτός αποτελεί έναν βασικό οδηγό για τη δημιουργία συναρτήσεων, αφού περιέχει τη βασική δομή των συναρτήσεων στο Matlab.

Παράδειγμα 9.1

Να κατασκευάσετε μια συνάρτηση η οποία να επιστρέφει το ελάχιστο και το μέγιστο στοιχείο ενός διανύσματος.

Παρουσιάστε τον τρόπο εκτέλεσης αλλά και το αποτέλεσμα της συνάρτησης για το διάνυσμα $[1,-2,3,4,9,-8]$.

Λύση Παραδείγματος 9.1

Η συνάρτηση που θέλουμε να κατασκευάσουμε θα πρέπει να επιστρέφει δύο μεταβλητές, το ελάχιστο, m, και το μέγιστο, M, στοιχείο ενός διανύσματος x. Οπότε οι μεταβλητές εξόδου θα είναι οι μεταβλητές m και M, ενώ η μεταβλητή εισόδου θα είναι ένα διάνυσμα, το x. Επειδή η άσκηση δεν μας ζητάει συγκεκριμένο όνομα για τη συνάρτηση, μπορούμε να χρησιμοποιήσουμε όποιο όνομα θέλουμε, όπως το min_max. Επομένως, η ζητούμενη συνάρτηση μπορεί να έχει την ακόλουθη δομή:

```
function [m,M] = min_max(x)
%MIN_MAX returns the min and the max of elements
%   MIN_MAX returns the minimum and the maximum
%   value of the elements of x

m = min(x);
M = max(x);
end
```

Στον παραπάνω κώδικα

1. έχουμε ακολουθήσει τη βασική δομή των συναρτήσεων (δομή πρώτης γραμμής, βοήθεια και το `end` στο τέλος),
2. έχουμε χρησιμοποιήσει τις ενσωματωμένες εντολές του Matlab, min και max, για να προσδιορίσουμε το ελάχιστο και το μέγιστο στοιχείο του διανύσματος x.

   Αν αποθηκεύσουμε το παραπάνω αρχείο σε ένα αρχείο με το όνομα min_max.m, τότε, εκτελώντας την εντολή

   ```
   >> [m,M] = min_max([1,-2,3,4,9,-8])
   ```

   για το διάνυσμα $[1,-2,3,4,9,-8]$, λαμβάνουμε τα ακόλουθα αποτελέσματα:

```
m =
    -8
M =
     9
```

που αποτελούν πράγματι την ελάχιστη και τη μέγιστη τιμή του διανύσματος $[1,-2,3,4,9,-8]$.

Παράδειγμα 9.2

Να κατασκευάσετε μια συνάρτηση, η οποία να επιστρέφει το μέγιστο πέντε αριθμών, χωρίς να γίνεται χρήση της ενσωματωμένης συνάρτησης max του Matlab, που θα εισάγει ο χρήστης κατά την κλήση της συνάρτησης.

Λύση Παραδείγματος 9.2

Για να κατασκευάσουμε μια συνάρτηση που θα επιστρέφει το μέγιστο, M, πέντε αριθμών, $n1, n2, n3, n4, n5$, που θα εισάγει ο χρήστης κατά την κλήση της συνάρτησης μπορούμε να χρησιμοποιήσουμε τον ακόλουθο κώδικα:

```
1  function M = mymax(n1, n2, n3, n4, n5)
2  %MYMAX returns the max
3  %      MYMAX calculates the maximum of the
4  %       five numbers given as input
5  M =  n1;
6   if n2 > M
7      M = n2;
8  end
9   if n3 > M
10     M = n3;
11 end
12  if n4 > M
13     M = n4;
14 end
15  if n5 > M
16     M = n5;
17 end
```

Στον κώδικα αυτό έχουμε κάνει χρήση διαδοχικών ελέγχων if, οι οποίοι εξετάζουν διαδοχικά αν οι τιμές $n2, n3, n4, n5$ είναι μεγαλύτερες από την τρέχουσα τιμή του μεγίστου, M. Αν είναι, τότε ανανεώνουν την τιμή του M με το αντίστοιχο n_i και το πρόγραμμα συνεχίζει τους ελέγχους που ακολουθούν.
Με την αποθήκευση του παραπάνω κώδικα σε ένα αρχείο με το όνομα mymax.m μπορούμε να καλέσουμε τη συνάρτηση και να λάβουμε το μέγιστο των 5 αριθμών που εισάγει ο χρήστης. Παραδείγματος χάριν, εκτελώντας την εντολή

```
>> mymax(20,15,24,9,16)
```

το Matlab επιστρέφει την τιμή 24.

Παρατήρηση 9.4

Στο παραπάνω παράδειγμα η συνάρτηση επέστρεφε μόνο μία μεταβλητή εξόδου. Στις περιπτώσεις αυτές μπορούν να παραληφθούν οι αγκύλες [] κατά τη δήλωση της μεταβλητής εξόδου.

Άσκηση αυτοαξιολόγησης 9.1

Η μετατροπή της θερμοκρασίας από βαθμούς Fahrenheit, T_f, σε βαθμούς Κελσίου, T_c, γίνεται μέσω της σχέσης

$$T_c = \frac{5}{9}(T_f - 32)$$

Κατασκευάστε μια συνάρτηση που να μετατρέπει σε βαθμούς Κελσίου ένα διάνυσμα μετρήσεων θερμοκρασίας σε βαθμούς Fahrenheit.
Παραθέστε τον τρόπο εκτέλεσης της συνάρτησης, καθώς και το αποτέλεσμά της για τους παρακάτω βαθμούς Fahrenheit για $T = [T1, T2, T3] = [100, 150, 200]$.

9.1.2 Ολοκληρωμένη δομή συναρτήσεων - Χειρισμός λαθών

Οι ενσωματωμένες συναρτήσεις του Matlab ελέγχουν κατά την εκτέλεσή τους αν οι μεταβλητές εισόδου είναι σύμφωνες με αυτές που δέχεται η συνάρτηση. Παραδείγματος χάριν, η εντολή

```
>> det([1,2,3;2,4,5])
```

επιστρέφει το ακόλουθο μήνυμα σφάλματος

```
Error using det
Matrix must be square.
```

αφού ο πίνακας

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \end{pmatrix}$$

δεν είναι τετραγωνικός και, επομένως, δεν ορίζεται η ορίζουσά του.

Μια συνάρτηση που ορίζει ένας χρήστης, για να θεωρείται ολοκληρωμένη, θα πρέπει να ενσωματώνει ανάλογους ελέγχους και μηνύματα. Στο πλαίσιο αυτής της προσέγγισης προτείνουμε δύο εναλλακτικές προσεγγίσεις.

Η πρώτη είναι αντίστοιχη με αυτή που υιοθετεί το Matlab, δηλαδή την εκτύπωση ενός μηνύματος σφάλματος και τη διακοπή της συνάρτησης χωρίς να επιστρέφει κάποιες τιμές για τις μεταβλητές εξόδου. Η προσέγγιση αυτή επιτυγχάνεται κάνοντας χρήση της εντολής `error` μέσα σε κατάλληλους ελέγχους.

Παραδείγματος χάριν, η συνάρτηση του Παραδείγματος 9.1 μπορεί να εμπλουτιστεί ως ακολούθως, έτσι ώστε να εξασφαλίσουμε ότι κατά την κλήση της συνάρτησης ο χρήστης εισάγει ένα διάνυσμα και όχι έναν πίνακα.

```
1  function [m,M] = min_max(x)
2  %MIN_MAX returns the min and the max of elements
3  %   MIN_MAX returns the minimum and the maximum
4  %   value of the elements of x
5
6  if min(size(x))~=1
7      error('input error')
8  end
9
```

```
10 m = min(x);
11 M = max(x);
12 end
```

Στον παραπάνω κώδικα οι εντολές που βρίσκονται στις γραμμές 6-8 ελέγχουν αν η μεταβλητή εισόδου x είναι διάνυσμα, δηλαδή αν η ελάχιστη διάστασή της είναι διαφορετική του 1. Στην περίπτωση που αυτό ισχύει, τότε τυπώνεται το κείμενο input error και η συνάρτηση διακόπτεται χωρίς να ανατεθεί κάποια τιμή στις μεταβλητές εξόδου. Μια τέτοια περίπτωση αφορά και η ακόλουθη κλήση της συνάρτησης.

```
>> [m,M] = min_max([1,2,3;2,4,5])
Error using min_max (line 7)
input error
```

Παρατηρήστε ότι το Matlab, πέρα από το μήνυμα λάθους που έχουμε ζητήσει εμείς να μας επιστρέψει σε περίπτωση λάθους, επιστρέφει και το κείμενο Error using min_max (line 7), που δηλώνει το όνομα του αρχείου και τη γραμμή που προκάλεσε τον τερματισμό της διαδικασίας.

Η δεύτερη προσέγγιση χειρισμών των σφαλμάτων διαφοροποιείται από την πρώτη στο ότι η συνάρτηση επιστρέφει κάποιες προκαθορισμένες τιμές (συνήθως `NaN`) στις μεταβλητές εξόδου. Η διαδικασία αυτή μπορεί να γίνει πιο κατανοητή με τη βοήθεια της παρακάτω συνάρτησης.

```
1  function [m,M] = min_max(x)
2  %MIN_MAX returns the min and the max of elements
3  %   MIN_MAX returns the minimum and the maximum
4  %   value of the elements of x
5
6  if min(size(x))~=1
7      m=NaN;
8      M=NaN;
9      warning('input error')
10     return
11 end
12
13 m = min(x);
14 M = max(x);
15 end
```

Η συνάρτηση αυτή αποτελεί μια διαφορετική εκδοχή της συνάρτησης του Παραδείγματος 9.1, στην οποία, σε περίπτωση λάθους μεταβλητής εισόδου, ζητάμε:

- αντί για μήνυμα σφάλματος να τυπωθεί μια προειδοποίηση (γραμμή 9 - `warning` - η προειδοποίηση αυτή δεν διακόπτει το πρόγραμμα),
- να ανατεθεί και στις δύο μεταβλητές εξόδου η τιμή `NaN` (γραμμές 7 και 8).

Η συνάρτηση διακόπτεται από την εντολή `return` (γραμμή 10) έτσι ώστε να μην συνεχιστεί η εκτέλεση των εντολών που ακολουθούν.

Το αποτέλεσμα της κλήσης αυτής της μορφής της συνάρτησης για $x = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \end{pmatrix}$ είναι το ακόλουθο

```
>> [m,M] = min_max([1,2,3;2,4,5])
Warning: input error
> In min_max (line 9)
m =
    NaN
M =
    NaN
```

στο οποίο μπορούμε να παρατηρήσουμε τη μορφή της προειδοποίησης που τυπώνει το Matlab, καθώς και τις τιμές που έχουν ανατεθεί στις μεταβλητές εξόδου.

Παράδειγμα 9.3

Ο τυχαίος περίπατος αποτελεί μια στοχαστική διαδικασία με πλήθος εφαρμογών σε πολλά επιστημονικά πεδία, όπως στη φυσική, τη μηχανική και τα οικονομικά. Για παράδειγμα, ο τυχαίος περίπατος συχνά χρησιμοποιείται για την περιγραφή της κίνησης ενός σωματιδίου σε ρευστό μέσο (π.χ. αέρας ή νερό) ή τη διακύμανση της τιμής μιας μετοχής.
Μια από τις πιο απλές εκδοχές του τυχαίου περιπάτου είναι αυτή του μονοδιάστατου απλού τυχαίου περιπάτου με δυνατότητα πραγματοποίησης τριών διαφορετικών «βημάτων» κάθε χρονική στιγμή. Μια τέτοια διαδικασία μπορεί να περιγραφτεί από τις ακόλουθες σχέσεις

$$x(n+1) = x(n) + step$$
$$x(1) = x_1$$

όπου $x(n)$ η τιμή (θέση) της στοχαστικής διαδικασίας που περιγράφει τον τυχαίο περίπατο τη χρονική στιγμή n και x_1 η αρχική θέση. Το $step$ λαμβάνει τυχαία μια από τις τιμές -1, 0 και 1 και αποτυπώνει το γεγονός της μείωσης κατά μία μονάδα, της διατήρησης ή της αύξησης κατά μία μονάδα της τιμής της διαδικασίας κάθε χρονική στιγμή.
Η παραπάνω σχέση ορισμού του απλού τυχαίου περιπάτου αποτελεί μια αναδρομική σχέση. Λαμβάνοντας υπόψη την παραπάνω παρατήρηση, κατασκευάστε μια συνάρτηση που να προσομοιώνει μια διαδικασία τυχαίου περιπάτου με απορροφητικά φράγματα. Τα απορροφητικά φράγματα είναι κάποιες τιμές (κάτω φράγμα, άνω φράγμα), στις οποίες, αν φτάσει η διαδικασία, σταματάει.
Χρήσιμη εντολή για την επίτευξη του σκοπού αυτού θα σας φανεί η εντολή `datasample([-1,0,1],1)`, η οποία επιλέγει τυχαία έναν αριθμό μεταξύ των -1, 0 και 1.
Η συνάρτηση θα πρέπει να δέχεται ως ορίσματα:

- την αρχική τιμή, $start$,
- το κάτω φράγμα, low,
- το άνω φράγμα, hi, και
- τη μεταβλητή εισόδου fig, η οποία, αν έχει την τιμή 1, θα πρέπει να κατασκευάζει το γράφημα του τυχαίου περιπάτου ως προς το n, δηλαδή τον αύξοντα αριθμό της παρατήρησης.

Σημειώνεται ότι πρέπει να ισχύει $low < start < hi$. Τέλος, η συνάρτηση θα πρέπει να επιστρέφει το διάνυσμα των τιμών του τυχαίου περιπάτου.

Λύση Παραδείγματος 9.3

Η υλοποίηση του απλού τυχαίου περιπάτου που περιγράφεται από την αναδρομική σχέση του Παραδείγματος μπορεί να γίνει με τη βοήθεια της παρακάτω συνάρτησης.

```
1  function x=randomwalk(start,low,hi,fig)
2  %RANDOM WALK Simulate a random walk with absorbing barriers
3  %            x(n+1)=x(n)+step, where step equals
4  %            to -1, 0 or 1 with equal probability
5  %            x(1)=start, the starting point
6  %            low and hi the absorbing barriers (low<start<hi)
7  %            set fig=1 to display the figure
8
9  if low>=start || hi<=start
10     error('inout error!  low<start<hi should be satisfied')
11 end
12
13 n=1;
14 x(n)=start;
15 while x(n)>low && x(n)<hi
16     step=datasample([-1,0,1],1);
17     x(n+1)=x(n)+step;
18     n=n+1;
19 end
20
21 if fig==1
22     plot(x)
23     ylim([low hi])
24 end
25
26 end
```

Στην παραπάνω συνάρτηση στις γραμμές:

- 2-7 έχουμε πληκτρολογήσει τη σχετική βοήθεια της συνάρτησης,
- 9-11 εξασφαλίζουμε την ισχύ της ανίσωσης $low < start < hi$. Στην περίπτωση που η ανίσωση αυτή δεν ισχύει, τότε η συνάρτηση διακόπτεται τυπώνοντας ένα σχετικό μήνυμα.
- 13-19 υλοποιούνται οι σχέσεις

$$x(n+1) = x(n) + step$$
$$x(1) = x_1$$

  μέχρις ότου η διαδικασία χτυπήσει σε ένα από τα δύο απορροφητικά φράγματα.
- 21-24 παρατίθενται οι εντολές για την κατασκευή της γραφικής παράστασης των τιμών της διαδικασίας. Η γραφική παράσταση κατασκευάζεται μόνο αν καλέσουμε τη συνάρτηση με $fig = 1$.

Άσκηση αυτοαξιολόγησης 9.2

Τροποποιήστε τη συνάρτηση που κατασκευάσατε στην Άσκηση αυτοαξιολόγησης 9.1, έτσι ώστε να εξασφαλίσετε:

- ότι ο χρήστης καλεί τη συνάρτηση χρησιμοποιώντας ένα διάνυσμα και όχι έναν πίνακα τιμών και
- ότι κανένα στοιχείο του παραπάνω διανύσματος δεν είναι μικρότερο από -459.67, τιμή που αντιστοιχεί στο απόλυτο μηδέν στην κλίμακα Fahrenheit.

Άσκηση αυτοαξιολόγησης 9.3: (Kalechman, 2018)

Η ισοδύναμη αντίσταση (ή ολική αντίσταση) R των αντιστατών σε ένα ηλεκτρικό κύκλωμα που αποτελείται από n αντιστάτες R_i δίνεται από τη σχέση

$$R = \sum_{i=1}^{n} R_i$$

όταν οι αντιστάτες είναι συνδεδεμένοι σε σειρά και από τη σχέση

$$R = \frac{1}{\sum_{i=1}^{n} 1/R_i}$$

όταν οι αντιστάτες είναι συνδεδεμένοι παράλληλα.
Κατασκευάστε μια συνάρτηση που θα υπολογίζει και θα επιστρέφει την ισοδύναμη αντίσταση. Η συνάρτηση θα πρέπει να δέχεται ως όρισμα

- το διάνυσμα των αντιστατών $[R_1, R_2, \ldots, R_n]$ και
- το αν αυτοί είναι συνδεδεμένοι σε σειρά ή παράλληλα (η πληροφορία αυτή θα πρέπει να εισάγεται στο πρόγραμμα με τη βοήθεια μιας μεταβλητής που θα παίρνει την τιμή 1, αν είναι σε σειρά, και 2, αν είναι παράλληλα συνδεδεμένη).

Η συνάρτηση θα πρέπει να εξασφαλίζει ότι η μεταβλητή εισόδου είναι πράγματι ένα διάνυσμα και ότι σε αυτό δεν περιέχεται καμία αρνητική τιμή.
Παραθέστε τον τρόπο εκτέλεσης της συνάρτησης, καθώς και το αποτέλεσμά της για τα παρακάτω κυκλώματα:

- Σύνδεση σε σειρά: $R_1 = 1, R_2 = 3, R_3 = 0.5$.
- Παράλληλη σύνδεση: $R_1 = 0.5, R_2 = 1.3, R_3 = 0.2$.

Παραθέστε και παραδείγματα στα οποία η συνάρτηση επιστρέφει μηνύματα σφάλματος.

9.2 Υποσυναρτήσεις

Ένα βασικό χαρακτηριστικό των συναρτήσεων στο Matlab είναι ότι σε ένα m-αρχείο μπορούμε να ορίσουμε παραπάνω από μία συνάρτηση, οι οποίες μπορούν να επικοινωνούν μεταξύ τους δημιουργώντας ένα τοπικό περιβάλλον λειτουργίας. Η συνάρτηση που ορίζεται στην πρώτη γραμμή του αρχείου, η οποία είναι και αυτή που δίνει και το όνομα του m-αρχείου, είναι η κύρια. Οι επιπλέον συναρτήσεις μπορούν να γραφτούν μετά το τέλος της κύριας συνάρτησης και ονομάζονται υποσυναρτήσεις. Οι υποσυναρτήσεις είναι προσβάσιμες μόνο από την κύρια συνάρτηση και από τις άλλες υποσυναρτήσεις του ίδιου αρχείου, αλλά δεν μπορούμε να τις

καλέσουμε από το command window του Matlab. Συνήθως, οι υποσυναρτήσεις αυτές εκτελούν λειτουργίες της κύριας συνάρτησης, οι οποίες επαναλαμβάνονται πολλές φορές.

Για να γίνει κατανοητή η παραπάνω λειτουργία των συναρτήσεων, στη συνέχεια παρουσιάζουμε ένα παράδειγμα όπου γίνεται χρήση των υποσυναρτήσεων.

Παράδειγμα 9.4

Να κατασκευάσετε μια συνάρτηση (με τη βοήθεια υποσυναρτήσεων) που θα υπολογίζει το εμβαδόν ενός τριγώνου με βάση τις συντεταγμένες των κορυφών του σύμφωνα με τον τύπο του Ήρωνα (Id and Kennedy, 1969).
Σημειώνεται ότι ο τύπος του Ήρωνα υπολογίζει το εμβαδόν ενός τριγώνου με βάση το μήκος, a,b,c, των πλευρών του τριγώνου, μέσω της σχέσης

$$E = \sqrt{S \cdot (S-a) \cdot (S-b) \cdot (S-c)}$$

όπου $S = \frac{a+b+c}{2}$.

Λύση Παραδείγματος 9.4

Η συνάρτηση με βάση την εκφώνηση θα πρέπει να δέχεται ως μεταβλητή εισόδου τις συντεταγμένες των κορυφών του τριγώνου. Επομένως, το πρώτο πράγμα που πρέπει να αποφασίσουμε είναι ο τρόπος που θα εισάγονται οι συντεταγμένες των κορυφών. Ένας εύκολος τρόπος είναι ο χρήστης να εισάγει τις συντεταγμένες με τη βοήθεια ενός πίνακα της μορφής

$$\begin{pmatrix} xA & xB & xC \\ yA & yB & yC \end{pmatrix}$$

δηλαδή στην πρώτη γραμμή να πληκτρολογούνται οι τετμημένες, ενώ στη δεύτερη οι τεταγμένες των κορυφών A, B και C.
Το δεύτερο που πρέπει να σκεφτούμε είναι ότι ο τύπος του Ήρωνα βασίζεται στο μήκος των πλευρών και όχι στις κορυφές. Επομένως, θα πρέπει να υπολογίσουμε τις αποστάσεις μεταξύ των τριών κορυφών υπολογίζοντας για κάθε ζευγάρι σημείων την Ευκλείδεια απόστασή τους.
Υπενθυμίζεται ότι η Ευκλείδεια απόσταση, d, μεταξύ δύο σημείων με συντεταγμένες$(x1,y1)$ και $(x2,y2)$, δίνεται από τη σχέση

$$d = \sqrt{(x1-x2)^2 + (y1-y2)^2}.$$

Έχοντας υπόψη τα παραπάνω μπορούμε να κατασκευάσουμε την ακόλουθη πρώτη έκδοση της συνάρτησης:

```
1  function [TrArea]=triangleArea(points)
2  %TRIANGLEAREA calculates the area of a triangle
3  %             given the coordinates of the three vertices
4  %
5  %             type the coordinates in the form
6  %             [xA,xB,xC;yA,yB,yC]
7
8
9      A=[points(1,1),points(2,1)];
10     B=[points(1,2),points(2,2)];
11     C=[points(1,3),points(2,3)];
12     a=side(B,C);
13     b=side(A,C);
```

```
14      c=side(A,B);
15      S=(a+b+c)/2;
16
17      TrArea=sqrt(S*(S-a)*(S-b)*(S-c));
18
19  end
20
21  function [x]=side(v1,v2)
22  %calculates the distance between two points
23
24      x=sqrt((v1(1,1)-v2(1,1))^2+(v1(1,2)-v2(1,2))^2);
25
26  end
```

Στην παραπάνω συνάρτηση, την οποία ονομάσαμε `triangleArea`, υπολογίζεται το εμβαδόν του τριγώνου με βάση τον τύπο του Ήρωνα, χρησιμοποιώντας την υποσυνάρτηση `side`. Η υποσυνάρτηση `side` εμφανίζεται μετά το `end` της κύριας συνάρτησης και υπολογίζει την Ευκλείδεια απόσταση μεταξύ δύο οποιωνδήποτε σημείων.

Η υποσυνάρτηση `side` καλείται από την κύρια συνάρτηση τρεις φορές (στις γραμμές 12, 13 και 14), υπολογίζοντας τα μήκη των πλευρών που είναι απέναντι από τις κορυφές A, B και C, αντίστοιχα. Παραδείγματος χάριν, στη γραμμή 12 υπολογίζεται το μήκος της πλευράς απέναντι από την κορυφή A, προσδιορίζοντας την απόσταση μεταξύ των κορυφών B και C. Στη γραμμή αυτή καλείται η υποσυνάρτηση `side` με ορίσματα τις συντεταγμένες των δύο κορυφών B και C και επιστρέφει την Ευκλείδεια απόστασή τους, η οποία υπολογίζεται στη γραμμή 24.

Σημειώνεται ότι οι μεταβλητές A, B και C, που περιέχουν τις συντεταγμένες των κορυφών, έχουν εξαχθεί από τον αρχικό πίνακα points (μεταβλητή εισόδου) με τη βοήθεια των εντολών των γραμμών 9-11. Στις γραμμές αυτές επιλέγονται κάθε φορά τα κατάλληλα στοιχεία του πίνακα και ανατίθενται στις μεταβλητές σύμφωνα με τις A, B και C και με τη μορφή του πίνακα που πρέπει να δώσει ο χρήστης κατά την κλήση της συνάρτησης. Η μορφή αυτή αναφέρεται και στη βοήθεια της κύριας συνάρτησης (γραμμές 2-6).

Αξίζει να σημειωθεί ότι η βοήθεια που εμφανίζεται στην υποσυνάρτηση (γραμμή 23), δεν εμφανίζεται, αν πληκτρολογήσουμε κάποια από τις εντολές `help triangleArea` και `help side` στο Command window του Matlab. Η πρώτη εντολή θα επιστρέψει τη βοήθεια της κύριας συνάρτησης, δηλαδή το κείμενο που εμφανίζεται στις γραμμές 2-6, ενώ η δεύτερη θα επιστρέψει ένα μήνυμα σφάλματος ότι δεν υπάρχει μεταβλητή ή συνάρτηση με το όνομα `side`. Ο λόγος που συμβαίνει αυτό είναι επειδή η υποσυνάρτηση `side` είναι προσβάσιμη μόνο από την κύρια συνάρτηση, δηλαδή την `triangleArea` και όχι από το υπόλοιπο Matlab. Συνέπεια αυτού είναι η βοήθεια που έχουμε πληκτρολογήσει για την υποσυνάρτηση να είναι χρήσιμη μόνο για κάποιον που θα διαβάσει τον κώδικα και όχι για κάποιον που απλώς θα χρησιμοποιήσει την κύρια συνάρτηση.

Η παραπάνω έκδοση του κώδικα όμως δεν είναι ολοκληρωμένη, αφού δεν έχουμε ελέγξει ότι ο χρήστης εισάγει τον πίνακα των συντεταγμένων με τη μορφή που θέλουμε. Φυσικά, δεν μπορούμε να εξετάσουμε αν έχει βάλει με τη σωστή σειρά τις συντεταγμένες των κορυφών A, B και C. Αυτό που μπορούμε και πρέπει να ελέγξουμε είναι ότι εισάγει έναν $2x3$ πίνακα. Στην περίπτωση που ο χρήστης δεν εισάγει τη σωστή μορφή μεταβλητών μπορούμε, παραδείγματος χάριν, να τον ενημερώνουμε για το σφάλμα και να επιστρέφουμε την τιμή `NaN` για τη μεταβλητή εξόδου, διακόπτοντας την εκτέλεση της συνάρτησης. Μια τέτοια προσέγγιση υλοποιείται προσθέτοντας στον προηγούμενο κώδικα τις εντολές των γραμμών 7-12, όπως εμφανίζεται στη συνέχεια.

```
function [TrArea]=triangleArea(points)
%TRIANGLEAREA calculates the area of a triangle
%            given the coordinates of the three vertices
%
%            type the coordinates in the form
%            [xA,xB,xC;yA,yB,yC]
s=size(points);
if s(1)~=2 || s(2)~=3
   TrArea=NaN;
   warning('input error')
   return
end

    A=[points(1,1),points(2,1)];
    B=[points(1,2),points(2,2)];
    C=[points(1,3),points(2,3)];
    a=side(B,C);
    b=side(A,C);
    c=side(A,B);
    S=(a+b+c)/2;
    TrArea=sqrt(S*(S-a)*(S-b)*(S-c));

end

function [x]=side(v1,v2)
%calculates the distance between two points

x=sqrt((v1(1,1)-v2(1,1))^2+(v1(1,2)-v2(1,2))^2);

end
```

Ολοκληρώνοντας το παράδειγμα μπορούμε να δούμε και τον τρόπο εκτέλεσης της συνάρτησης με τη βοήθεια ενός παραδείγματος. Αφού έχουμε αποθηκεύσει τη συνάρτηση με το όνομα `triangleArea.m`, μπορούμε να πληκτρολογήσουμε τις ακόλουθες εντολές, λαμβάνοντας τα αντίστοιχα αποτελέσματα:

```
>> x=[-1,2,2;3,3,-6];
>> ar=triangleArea(x);
>> ar
ar =
   13.5000
```

Από τα παραπάνω αποτελέσματα διαπιστώνουμε ότι το εμβαδόν του τριγώνου με κορυφές τα σημεία (-1,3), (2,3) και (2,-6) ισούται με 13.5, η οποία τιμή έχει ανατεθεί και αποθηκευτεί στη μεταβλητή με το όνομα ar.

Άσκηση αυτοαξιολόγησης 9.4

Τροποποιήστε τη συνάρτηση του Παραδείγματος 9.2, έτσι ώστε να επιστρέφει το μέγιστο n αριθμών, κάνοντας χρήση μιας επαναληπτικής διαδικασίας και κατάλληλης υποσυνάρτησης για τη σύγκριση δύο αριθμών. Η συνάρτηση θα πρέπει να εξασφαλίζει ότι έχουν εισαχθεί τουλάχιστον 2 πραγματικοί αριθμοί.

9.3 Αναδρομή

Ένα ιδιαίτερο χαρακτηριστικό των συναρτήσεων στο Matlab είναι ότι μπορούν να καλούν όχι μόνο άλλες συναρτήσεις αλλά και τον ίδιο τους τον εαυτό. Το χαρακτηριστικό αυτό είναι ιδιαίτερα χρήσιμο όταν είναι απαραίτητος κάποιος υπολογισμός μέσω κάποιας αναδρομικής σχέσης. Κλασικό παράδειγμα μιας τέτοιας περίπτωσης είναι ο υπολογισμός του $n!$, ο οποίος μπορεί να εκφραστεί αναδρομικά ως εξής:

$$n! = n \cdot (n-1)!$$
$$1! = 1$$

Σημειώνεται ότι το $0!$ ορίζεται να ισούται με 1.

Ο υπολογισμός του $n!$ στο Matlab μπορεί να γίνει χρησιμοποιώντας την παρακάτω συνάρτηση:

```
1  function [result]=nfactorial(n)
2  %NFACTRORIAL returns the n!
3  if round(n)~=n || n<0
4      warning([num2str(n),' is not a natural number'])
5      result=NaN;
6      return
7  end
8  if n==0
9      result=1;
10 else
11     result=n*nfactorial(n-1);
12 end
13 end
```

Η παραπάνω συνάρτηση, αν το n είναι μεγαλύτερο του 1, καλεί τον εαυτό της (γραμμή 11), υλοποιώντας την αναδρομική σχέση ορισμού του $n!$. Έτσι, πληκτρολογώντας την εντολή

```
>> nfactorial(3)
```

η συνάρτηση επιστρέφει την τιμή 6, ακολουθώντας τα παρακάτω βήματα:

1. Ελέγχει και διαπιστώνει ότι το 3 είναι φυσικός αριθμός (γραμμή 3).
2. Έλέγχει αν το 3 είναι ίσο με 0 (γραμμή 8). Αφού δεν είναι ίσο με μηδέν, συνεχίζει με το `else` (γραμμή 10) και εκτελεί την εντολή 3*nfactorial(2).
3. Το `nfactorial(2)` στο προηγούμενο βήμα καλεί εκ νέου τη συνάρτηση `nfactorial`, αλλά αυτή τη φορά με $n = 2$, δημιουργώντας ένα καινούργιο περιβάλλον λειτουργίας.
4. Επαναλαμβάνονται εκ νέου τα βήματα 1 και 2, αλλά με $n = 2$. Στο τέλος της επανάληψης του δεύτερου βήματος εκτελείται η εντολή `2*nfactorial(1)`.

5. Το `nfactorial(1)` στο προηγούμενο βήμα καλεί εκ νέου τη συνάρτηση `nfactorial`, αλλά αυτή τη φορά με $n = 1$.
6. Επαναλαμβάνονται εκ νέου τα βήματα 1 και 2, αλλά με $n = 1$, δημιουργώντας ένα καινούργιο περιβάλλον λειτουργίας. Στο τέλος της επανάληψης του δεύτερου βήματος εκτελείται η εντολή `1*nfactorial(0)`.
7. Το `nfactorial(0)` στο προηγούμενο βήμα καλεί εκ νέου τη συνάρτηση `nfactorial`, αλλά αυτή τη φορά με $n = 0$.
8. Επαναλαμβάνονται εκ νέου τα βήματα 1 και 2, αλλά με $n = 0$, δημιουργώντας ένα καινούργιο περιβάλλον λειτουργίας, αλλά αυτή τη φορά ικανοποιείται η συνθήκη $n == 0$, οπότε επιστρέφεται η τιμή 1 και κλείνει το περιβάλλον λειτουργίας που δημιούργησε το `nfactorial(0)`.
9. Η τιμή 1 του `nfactorial(0)` επιστρέφεται και πολλαπλασιάζεται με το 1 (βήμα 6). Στη συνέχεια, επιστρέφεται η τιμή του `nfactorial(1)`, δηλαδή το αποτέλεσμα του γινομένου, δηλαδή πάλι το 1. Η παραπάνω διαδικασία κλείνει το περιβάλλον λειτουργίας που δημιούργησε το `nfactorial(1)` στο βήμα 3.
10. Η τιμή 1 του `nfactorial(1)` επιστρέφεται και πολλαπλασιάζεται με το 2 (βήμα 4). Στη συνέχεια εκτελείται η πράξη του γινομένου τους, με αποτέλεσμα το 2, το οποίο είναι και η τιμή του `nfactorial(2)`.
11. Η τιμή 2 του `nfactorial(2)` επιστρέφεται και πολλαπλασιάζεται με το 3 (βήμα 2). Στη συνέχεια εκτελείται η πράξη του γινομένου τους, με αποτέλεσμα το 6, το οποίο είναι και η τιμή του `nfactorial(3)`. Η τιμή του `nfactorial(3)` επιστρέφεται και ολοκληρώνεται η διαδικασία.

Παράδειγμα 9.5

Κατασκευάστε μια συνάρτηση που να επιστρέφει

- το διάνυσμα με τα ψηφία και
- το πρόσημο

ενός ακέραιου αριθμού.
Για παράδειγμα, για τον αριθμό 23531 θα πρέπει να επιστρέφει το διάνυσμα $[2,3,5,3,1]$ και το $+1$ για το πρόσημό του.

Λύση Παραδείγματος 9.5

Η ζητούμενη συνάρτηση μπορεί να κατασκευαστεί παρατηρώντας αρχικά ότι αρκεί να απομονώσουμε τα ψηφία της απόλυτης τιμής του ακέραιου και να προσδιορίσουμε το πρόσημό του. Επομένως, η συνάρτηση θα υλοποιηθεί για την απόλυτη τιμή, έστω N, του ακέραιου αριθμού K.
Το τελευταίο ψηφίο ενός ακέραιου, N, ισούται με το υπόλοιπο της διαίρεσης του N με το 10. Από την άλλη, το ακέραιο μέρος του αποτελέσματος της διαίρεσης του N με το 10 περιέχει όλα τα υπόλοιπα ψηφία του αριθμού.
Έτσι, από τη διαίρεση του N με το 10 μπορούμε να:

- εξάγουμε το τελικό ψηφίο, κρατώντας το υπόλοιπο της διαίρεσης (εντολή `rem`),
- να κρατήσουμε τα υπόλοιπα ψηφία, στρογγυλοποιώντας προς τα κάτω (δηλαδή προς τον πλησιέστερο ακέραιο) το αποτέλεσμα της διαίρεσης (εντολή `floor`).

Η παραπάνω διαδικασία θα πρέπει να επαναληφθεί για τα υπόλοιπα ψηφία του αριθμού μέχρις ότου έχουμε έναν μονοψήφιο αριθμό. Η επανάληψη της διαδικασίας μπορεί να γίνει καλώντας σε κάθε βήμα την ίδια την αρχική συνάρτηση.

Η παραπάνω διαδικασία μπορεί να υλοποιηθεί με τη βοήθεια της παρακάτω συνάρτησης:

```
1   function [Digits, s] = Number2Digits(K)
2   %NUMBER2DIGITS returns the separate digits of an integer
3
4   if round(K)~=K
5       warning([num2str(K),' is not an integer number'])
6       Digits=NaN;
7       s=NaN;
8       return
9   end
10
11  N=abs(K);
12  if N<10
13      Digits = N;
14  else
15      M = floor(N/10);
16      Digits = [Number2Digits(M),rem(N,10)];
17  end
18
19  s=sign(N);
20
21  end
```

Στην παραπάνω συνάρτηση:

- Στη γραμμή 2 έχουμε πληκτρολογήσει τη σχετική βοήθεια της συνάρτησης.
- Στις γραμμές 4-9 ελέγχεται αν η μεταβλητή εισόδου K είναι ακέραιος και, αν δεν είναι, τυπώνεται σχετική προειδοποίηση, ανατίθεται η τιμή `NaN` στις μεταβλητές εξόδου και διακόπτεται η συνάρτηση.
- Στη γραμμή 11 ορίζεται η μεταβλητή N στην οποία ανατίθεται η απόλυτη τιμή του K.
- Στις γραμμές 11-17 υλοποιείται η περιγραφθείσα διαδικασία για την απομόνωση των ψηφίων του N. Πιο συγκεκριμένα, αν το N είναι μικρότερο του 10, τότε είναι μονοψήφιο και επιστρέφεται η τιμή του, ενώ, αν είναι μεγαλύτερο του 10:

  1. υπολογίζεται το ακέραιο μέρος της διαίρεσης του N με το 10 στρογγυλοποιώντας προς τα κάτω (δηλαδή προς τον πλησιέστερο ακέραιο) (γραμμή 15) και ανατίθεται στη μεταβλητή M,
  2. καλείται εκ νέου η συνάρτηση `Number2Digits`, αλλά αυτή τη φορά με την τιμή M και απομονώνεται το τελικό ψηφίο του N με την εντολή `rem(N,10)`.

     Η διαδικασία αυτή επαναλαμβάνεται μέχρι να απομονωθούν όλα τα ψηφία του K.

- Στη γραμμή 18 ανατίθεται στη μεταβλητή εξόδου, s, το πρόσημο του K με τη βοήθεια της ενσωματωμένης εντολής του Matlab `sign`.

Άσκηση αυτοαξιολόγησης 9.5

Η ακολουθία αριθμών Fibonacci ορίζεται από τις σχέσεις

$$F_1 = 0, \quad F_2 = 1$$

και

$$F_n = F_{n-1} + F_{n-2}$$

για $n > 2$. Κατασκευάστε μια συνάρτηση, η οποία θα επιστρέφει τον αριθμό F_n, δηλαδή τον n-οστο όρο της ακολουθίας Fibonacci.

9.4 Σύγκριση script αρχείων και συναρτήσεων

Όπως έχει αναφερθεί, τα αρχεία συναρτήσεων παρουσιάζουν σημαντικές ομοιότητες με τα script αρχεία αλλά και ουσιώδεις διαφορές. Οι πιο σημαντικές διαφορές είναι οι ακόλουθες:

- Στα αρχεία συναρτήσεων οι παράμετροι της συνάρτησης δηλώνονται κατά την κλήση της συνάρτησης, ενώ στα script αρχεία μπορούμε να εισάγουμε τις τιμές που θέλουμε με κατάλληλες εντολές, όπως η εντολή `input`.
- Οι μεταβλητές που ορίζονται στα αρχεία συναρτήσεων είναι τοπικές, ενώ στα script αρχεία όχι.
- Στα αρχεία συναρτήσεων μπορούν να οριστούν και επιπρόσθετες συναρτήσεις (υπό-συναρτήσεις), ενώ στα script αρχεία όχι.

Λαμβάνοντας υπόψη τις παραπάνω διαφορές, είναι δυνατόν να μετατρέψουμε κάθε script σε μια ισοδύναμη συνάρτηση, όπως παρουσιάζεται στο επόμενο παράδειγμα.

Παράδειγμα 9.6

Στη συνέχεια, παρουσιάζεται ένα script αρχείο για τον προσδιορισμό του:

1. μέγιστου μέτρου όλων των ριζών,
2. μέγιστου μέτρου όλων των πραγματικών ριζών

ενός πολυωνύμου.

```
1  A=input('dwse tous syntelestes tou polyonumou toulaxiston 2ou bathou:'
       );
2
3  sA=size(A);
4  while sA(1)~=1 ||sA(2)<3||A(1)==0
5      disp('input error, try again');
6      A=input('dwse tous syntelestes tou polyonumou toulaxiston 2ou
           bathou');
7      sA=size(A);
8  end
9
10 rA=roots(A);
11 max_metro_all=max(abs(rA));
12 real_rootsA=find(imag(rA)==0);
```

```
13
14  if ~isempty(real_rootsA)
15      max_metro_real=max(abs(rA(real_rootsA)));
16  else
17      max_metro_real=NaN;
18  end
19
20  [max_metro_all,max_metro_real]
```

Μετατρέψτε το παραπάνω script αρχείο σε μια συνάρτηση.

Λύση Παραδείγματος 9.6

Για τη μετατροπή ενός script αρχείου σε συνάρτηση, το πρώτο που πρέπει να γίνει είναι να εντοπιστούν οι μεταβλητές που εισάγει ο χρήστης με κατάλληλες εντολές, όπως η εντολή `input`. Στο συγκεκριμένο παράδειγμα υπάρχει μία μόνο τέτοια μεταβλητή, το A. Η μεταβλητή αυτή θα είναι η μεταβλητή εισόδου για τη συνάρτηση.
Εν συνεχεία, πρέπει να αποφασιστούν οι μεταβλητές και να δηλωθούν στο κατάλληλο πεδίο κατά τη δημιουργία της συνάρτησης. Οι μεταβλητές αυτές είναι οι μεταβλητές ενδιαφέροντος, οι οποίες για το παράδειγμα είναι οι μεταβλητές που προσδιορίζουν:

1. το μέγιστο μέτρο όλων των ριζών, `max_metro_all`, και το
2. μέγιστο μέτρο όλων των πραγματικών ριζών, `max_metro_real`,

του πολυωνύμου A.
Το τρίτο βήμα είναι να εντοπίσουμε τα στοιχεία εκείνα του κώδικα που αφορούν τον χειρισμό λαθών και να τροποποιήσουμε, έτσι ώστε να εντάσσονται στη φιλοσοφία λειτουργίας των συναρτήσεων. Στο script αρχείο του παραδείγματος το μέρος αυτό του κώδικα είναι το ακόλουθο:

```
1  sA=size(A);
2  while sA(1)~=1 ||sA(2)<3||A(1)==0
3      disp('input error, try again');
4      A=input('dwse tous syntelestes tou polyonumou toulaxiston 2ou
           bathou');
5      sA=size(A);
6  end
```

Στο τμήμα αυτό του κώδικα ελέγχεται αν το A έχει την κατάλληλη μορφή και, αν δεν έχει, τότε τυπώνεται ένα μήνυμα σφάλματος και ζητείται εκ νέου η εισαγωγή μιας νέας τιμής για το A. Στις συναρτήσεις πρέπει να γίνει πάλι ο ίδιος έλεγχος και να τυπώνεται το αντίστοιχο μήνυμα σφάλματος, αλλά θα πρέπει ο κώδικας να διακόπτεται.
Λαμβάνοντας υπόψη όλα τα παραπάνω, μπορούμε να κατασκευάσουμε την ακόλουθη συνάρτηση, η οποία είναι ισοδύναμη του script αρχείου του παραδείγματος.

```
1  function [ max_metro_all,  max_metro_real] = absofroots(A)
2  %ABSOFROOTS returns the max modulus of all the roots
3  %           and the max modulus of all the real roots
4  %           of a polynomial
5      sA=size(A);
6      if sA(1)~=1 ||sA(2)<3||A(1)==0
7          error('input error, try again');
```

```
 8       end
 9
10       rA=roots(A);
11       max_metro_all=max(abs(rA));
12       real_rootsA=find(imag(rA)==0);
13
14       if ~isempty(real_rootsA)
15           max_metro_real=max(abs(rA(real_rootsA)));
16       else
17           max_metro_real=NaN;
18       end
19   end
```

Στην παραπάνω συνάρτηση έχουμε κρατήσει τον κύριο κώδικα του script αρχείου, αφού τα script αρχεία και οι συναρτήσεις κάνουν χρήση των ίδιων:

- ελέγχων ροής,
- επαναληπτικών διαδικασιών και
- ενσωματωμένων εντολών και συναρτήσεων του Matlab.

9.5 Συναρτήσεις με μεταβλητό μήκος εισόδου ή/και εξόδου

Οι συναρτήσεις που έχουν παρουσιαστεί μέχρις αυτό το σημείο έχουν ένα σταθερό και προκαθορισμένο πλήθος μεταβλητών εισόδου και εξόδου. Σε αρκετές, όμως, περιπτώσεις κάτι τέτοιο δεν είναι πάντα επιθυμητό, αφού μπορεί κάποιες από τις μεταβλητές να μην χρειάζονται σε κάποιες περιπτώσεις και, πιθανά, κάποιες από τις διαδικασίες που εκτελούνται μέσα στη συνάρτηση να επιστρέφουν διαφορετικό πλήθος μεταβλητών, αναλόγως την περίπτωση.

Το Matlab επιτρέπει τη δημιουργία συναρτήσεων με μεταβλητό μήκος εισόδου ή/και εξόδου, δηλαδή με μεταβλητό, μη αυστηρά προκαθορισμένο πλήθος μεταβλητών εισόδου ή/και εξόδου. Αυτό πετυχαίνεται με τη χρήση των εντολών varargin και varargout στα πεδία δήλωσης των μεταβλητών εισόδου και εξόδου, αντίστοιχα. Οι δύο προαναφερθείσες εντολές χρησιμοποιούνται από το Matlab για την αποθήκευση επιπρόσθετων, συχνά μη απαραίτητων μεταβλητών, διαφόρων τύπων (αριθμών, διανυσμάτων, πινάκων, ...) κατά την κλήση της συνάρτησης. Η πρόσβαση στο πρώτο, δεύτερο και ούτω καθεξής στοιχείο των varargin και varargout γίνεται με τις εντολές

$$varargin\{1\}, varargin\{2\}, \ldots$$

και

$$varargout\{1\}, varargout\{2\}, \ldots$$

αντίστοιχα.

9.5.1 Μεταβλητό μήκος εισόδου

Η μεταβλητή varargin, όπως προαναφέρθηκε, χρησιμοποιείται για την αποθήκευση κατά την κλήση της συνάρτησης ενός μεταβλητού αριθμού ορισμάτων εισόδου. Η μεταβλητή varargin μπορεί να

χρησιμοποιηθεί για το σύνολο των μεταβλητών εισόδου ή μόνο για μερικές από αυτές. Στη δεύτερη περίπτωση, οι μεταβλητές εισόδου που δηλώνονται εκτός της varargin είναι απαραίτητο να υπάρχουν κατά την κλήση της συνάρτησης.

Συνέπεια των παραπάνω είναι να υπάρχουν δύο διαφορετικοί τρόποι εισαγωγής δεδομένων.

Ο πρώτος τρόπος μπορεί να περιγραφθεί από την ακόλουθη βασική δομή

```
1 function [outputArg] = untitled(varargin)
2 ...
3 end
```

στην οποία όλες οι μεταβλητές εισόδου είναι προαιρετικές και συμπεριλαμβάνονται στη μεταβλητή varargin.

Ο δεύτερος τρόπος έχει την ακόλουθη βασική δομή

```
1 function [outputArg] = untitled(inputArg1,inputArg2,varargin)
2 ...
3 end
```

και είναι υποχρεωτική η παρουσία των μεταβλητών εισόδου inputArg1 και inputArg2 κατά την κλήση της συνάρτησης.

Βοηθητική εντολή, ειδικά στις συναρτήσεις με μεταβλητό μήκος εισόδου, είναι η εντολή `nargin`. Η εντολή `nargin` επιστρέφει το πλήθος των μεταβλητών εισόδου κατά την κλήση της συνάρτησης. Ιδιαίτερη προσοχή πρέπει να δοθεί στο γεγονός ότι η εντολή `nargin` επιστρέφει το πλήθος του συνόλου μεταβλητών εισόδου κατά την κλήση της συνάρτησης και όχι το πλήθος των μεταβλητών που περιέχονται στην varargin.

9.5.2 Μεταβλητό μήκος εξόδου

Όπως με τις μεταβλητές εισόδου, έτσι μπορεί και το πλήθος των μεταβλητών εξόδου να μην είναι προκαθορισμένο αλλά μεταβλητό. Για να επιτευχθεί αυτό, απαραίτητο είναι να δηλωθεί η ύπαρξη προαιρετικών μεταβλητών με τη βοήθεια της εντολής varargout στο πεδίο των μεταβλητών εξόδων της συνάρτησης. Σημειώνεται ότι υπάρχουν πάλι δύο διαφορετικές εκδοχές. Η πρώτη αφορά συναρτήσεις που όλες οι μεταβλητές εξόδου χαρακτηρίζονται ως προαιρετικές και η βασική της μορφή είναι η ακόλουθη:

```
1 function [varargout] = untitled(inputArg1,inputArg2)
2 ...
3 end
```

και μια δεύτερη, όπου κάποιες μεταβλητές εξόδου δηλώνονται ότι πρέπει να επιστραφούν υποχρεωτικά, ενώ κάποιες άλλες όχι. Για τη δεύτερη περίπτωση, η βασική μορφή είναι η ακόλουθη:

```
1 function [outputArg, varargout] = untitled(inputArg1,inputArg2)
2 ...
3 end
```

Βοηθητική εντολή για τις συναρτήσεις με μεταβλητό μήκος εξόδου είναι η εντολή `nargout`. Η εντολή `nargout` επιστρέφει το πλήθος των μεταβλητών εξόδου που αναμένεται από την κλήση της συνάρτησης. Παραδείγματος χάριν, αν η μια συνάρτηση κληθεί με τον ακόλουθο τρόπο

```
[x, y, z, w] = untitled(...
```

τότε η εντολή `nargout` επιστρέφει τον αριθμό 4.

Παράδειγμα 9.7

Να κατασκευαστεί μια συνάρτηση, η οποία θα μετατρέπει τα μίλια σε μέτρα. Η συνάρτηση θα πρέπει να δέχεται ως όρισμα την τιμή των μιλίων. Επειδή συχνά η απόσταση δίνεται ως μίλια ή πόδια ή ίντσες και όχι ως δεκαδικός αριθμός (π.χ. το 1.24 μίλι μπορεί να εκφραστεί ισοδύναμα ως 1 μίλι, 1267 πόδια και 2.4 ίντσες), η συνάρτηση θα πρέπει να δέχεται ως προαιρετικές μεταβλητές εισόδου τον αριθμό ποδιών και ιντσών της απόστασης.
Τέλος, η συνάρτηση θα πρέπει, αν ζητείται, να επιστρέφει την απόσταση και σε Km.

Λύση Παραδείγματος 9.7

Εδώ, ζητείται να κατασκευάσουμε μια συνάρτηση με μεταβλητό μήκος εισόδου και εξόδου. Επομένως, η βασική της μορφή θα είναι η ακόλουθη:

```
1  function [meters, varargout] = untitled(miles,varargin)
2  ...
3  end
```

αφού θέλουμε υποχρεωτικά να δέχεται την τιμή των μιλίων και να επιστρέφει την απόσταση σε μέτρα. Η μεταβλητή varargin θα χρησιμοποιηθεί, για να αποθηκεύσει τις τιμές των ποδιών και των ιντσών, αν αυτές δηλωθούν κατά την κλήση της συνάρτησης. Από την άλλη, η μεταβλητή varargout θα χρησιμοποιηθεί, για να αποθηκεύσει την τιμή της απόστασης σε Km, αν αυτό ζητηθεί κατά την κλήση της συνάρτησης.
Στη συνέχεια, βασιζόμενοι στα προαναφερθέντα, παρουσιάζεται ο κώδικας για την υλοποίηση της ζητούμενης συνάρτησης.

```
1  function [meters,varargout] = miles2meters(miles, varargin)
2  % MILES2METERS Converts MILES (plus optional FEET and INCHES input)
3  %              values to METERS.
4      if nargin > 3
5          error('1 to 3 input arguments are required');
6      end
7
8      if nargin == 1
9          inches = miles * 63360;
10     else
11         feet=varargin{1};
12         inches = feet * 12;
13         inches = inches + miles * 63360;
14
15         if nargin==3
16             inches = inches + varargin{2};
17         end
18     end
19     meters = inches * 0.0254;
20
21     if nargout==2
22         varargout{1}=meters/1000;
23     end
24 end
```

Στον παραπάνω κώδικα:

- Στις γραμμές 2 και 3 έχουμε πληκτρολογήσει τη σχετική βοήθεια της συνάρτησης.
- Στις γραμμές 4-6 ελέγχεται με τη βοήθεια της εντολής `nargin` αν το πλήθος των μεταβλητών εισόδου είναι μεγαλύτερο από 3. Στην περίπτωση αυτή η συνάρτηση τυπώνει σχετικό μήνυμα λάθους και διακόπτεται.
- Στη γραμμή 9 μετατρέπεται η τιμή της μεταβλητής εισόδου miles σε μέτρα, αν δεν έχει εισαχθεί καμία προαιρετική μεταβλητή εισόδου (`nargin == 1` - γραμμή 8).
- Αν έχει εισαχθεί τουλάχιστον μια προαιρετική μεταβλητή εισόδου, τότε
  - μετατρέπονται τα πόδια (`varargin{1}`) σε ίντσες (γραμμές 11 και 12),
  - και προστίθενται στις ίντσες που αντιστοιχούν στα μίλια (γραμμή 13),
  - στη συνέχεια εξετάζεται αν έχει εισαχθεί και δεύτερη προαιρετική μεταβλητή εισόδου (γραμμή 15), η οποία αντιστοιχεί στις επιπρόσθετες ίντσες. Αν έχει συμβεί αυτό, τότε ανανεώνεται η τιμή της μεταβλητής inches.

  Αφού έχουν μετατραπεί όλες οι ποσότητες σε ίντσες, τότε η τιμή αυτή μετατρέπεται σε μέτρα (γραμμή 19).
- Στις γραμμές 21-23 μετατρέπονται τα μέτρα σε km. Η μετατροπή αυτή γίνεται μόνο αν κατά την κλήση της συνάρτησης δηλωθεί μια επιπρόσθετη, της υποχρεωτικής, μεταβλητή εξόδου.

Στη συνέχεια, παρουσιάζονται διάφοροι τρόποι κλήσης της συνάρτησης για τη μετατροπή των 1.24 μιλίων σε μέτρα (ή και km).

```
>> format shortg
>> meters = miles2meters(1.24)
meters =
       1995.6

>> meters = miles2meters(1,1267,2.4)
meters =
       1995.6

>> [meters, Km] = miles2meters(1,1267,2.4)
meters =
       1995.6
Km =
       1.9956
```

Στις παραπάνω εντολές και στα αντίστοιχα αποτελέσματά τους μπορούμε να δούμε εύκολα τη λειτουργία των προαιρετικών μεταβλητών εισόδων και εξόδου.

Άσκηση αυτοαξιολόγησης 9.6

Τροποποιήστε τη συνάρτηση του Παραδείγματος 9.3, έτσι ώστε ο χρήστης να έχει τη δυνατότητα, αν το επιθυμεί, να περάσει τις εντολές μορφοποίησης του γραφήματος.

9.6 Ανώνυμες συναρτήσεις

Κλείνοντας το κεφάλαιο, πρέπει να αναφέρουμε ότι το Matlab πέρα από τα αρχεία συναρτήσεων επιτρέπει τη δημιουργία προσωρινών και σύντομων συναρτήσεων, οι οποίες αποκαλούνται ανώνυμες συναρτήσεις. Οι συναρτήσεις αυτές μπορούν να οριστούν στο Command window, σε script αρχεία αλλά και σε αρχεία συναρτήσεων. Οι ανώνυμες συναρτήσεις αποτελούνται από μία μόνο έκφραση του Matlab και μπορεί να έχει οποιοδήποτε πλήθος μεταβλητών εισόδου και εξόδου.

Το συντακτικό για τη δημιουργία μιας ανώνυμης συνάρτησης είναι το ακόλουθο:

```
f = @(arglist)expression
```

Ένα παράδειγμα μιας τέτοιας συνάρτησης είναι το ακόλουθο

```
power = @(x, y) x.^y;
```

η οποία υπολογίζει τον αριθμό x^y για x και y που θα δοθούν κατά την κλήση της.

Έτσι, παραδείγματος χάριν, οι εντολές

```
result1 = power(2, 4);
result2 = power(10, 3);
```

δημιουργούν τις μεταβλητές result1 και result2 με τιμές 16 και 1000, αντίστοιχα.

9.7 Ασκήσεις

Άσκηση 9.1. *Δημιουργήστε μια συνάρτηση η οποία θα υπολογίζει και θα επιστρέφει (υπό μορφή διανύσματος) τους συντελεστές του πολυωνύμου με συγκεκριμένες ρίζες. Η συνάρτηση πρέπει να δέχεται ως όρισμα το διάνυσμα των ριζών του πολυωνύμου.*

Παραθέστε τον τρόπο εκτέλεσης της συνάρτησης, καθώς και το αποτέλεσμά της για το πολυώνυμο με ρίζες το 2, το 3 και το 4.

Άσκηση 9.2. *Συμπληρώστε τη συνάρτηση της προηγούμενης άσκησης, έτσι ώστε να επιστρέφει και την τιμή του πολυωνύμου στο σημείο* $x = -1$.

Παραθέστε τον τρόπο εκτέλεσης της συνάρτησης, καθώς και το αποτέλεσμά της για το πολυώνυμο με ρίζες το 2, το 3 και το 4.

Άσκηση 9.3. *Κατασκευάστε μια συνάρτηση που να μας δίνει τους τυχερούς αριθμούς και τον νικητή στο ακόλουθο παιχνίδι.*

> *Κ παίκτες επιλέγουν τυχαία* s *διαφορετικούς αριθμούς ανάμεσα στους* $1, 2, \ldots, n$.
>
> *Ο υπολογιστής επιλέγει και αυτός* s *αριθμούς ανάμεσα στους* $1, 2, \ldots, n$. *Αν ένας παίκτης έχει βρει και τους* s *τυχερούς αριθμούς, τότε ανακηρύσσεται νικητής.*

Για την κατασκευή της συνάρτησης, χρήσιμη μπορεί να σας φανεί η εντολή `randperm`.

Άσκηση 9.4. *Κατασκευάστε μια συνάρτηση που να μας δίνει το κέντρο βάρους μιας δοκού (χωρίς βάρος), όταν τοποθετούμε σε αυτήν βάρη σε διάφορα σημεία της. Δοκιμάστε τη συνάρτηση τοποθετώντας τα βάρη*

$$weights = [9,2,3,1,1]$$

στις αποστάσεις

$$locations = [1,14,15,18,19]$$

από την αρχή της δοκού, η οποία αρχή υποθέτουμε ότι βρίσκεται στο μηδέν.

Άσκηση 9.5. *Ένας μηχανικός είχε φτιάξει παλαιότερα μια συνάρτηση στο* Matlab *με το όνομα* $converttemp$, *με τη βοήθεια της οποίας μετέτρεπε τη θερμοκρασία από βαθμούς* $Celsius$ *σε βαθμούς* $Fahrenheit$ *και βαθμούς* $Kelvin$.

Μάλιστα, είχε κρατήσει και την ακόλουθη εκτύπωση του αποτελέσματος της συνάρτησης για θερμοκρασια 10 βαθμών $Celsius$.

```
[F,K]=converttemp(-10)
F =
    14
K =
  263.1500
```

Βοηθήστε τον μηχανικό να ξαναφτιάξει τη συνάρτηση αυτή. Δίνεται ότι

$$Fahrenheit = \frac{9}{5} Celsius + 32$$

και

$$Kelvin = Celsius + 273.15.$$

9.8 Απαντήσεις στις ασκήσεις αυτοαξιολόγησης

Λύση άσκησης αυτοαξιολόγησης 9.1

Η μετατροπή της θερμοκρασίας από βαθμούς *Fahrenheit* σε βαθμούς *Celsius* γίνεται μέσω της σχέσης

$$T_c = \frac{5}{9}(T_f - 32).$$

Κατασκευάστε μια συνάρτηση που να μετατρέπει σε βαθμούς *Celsius* ένα διάνυσμα μετρήσεων θερμοκρασίας σε βαθμούς Fahrenheit.
Παραθέστε τον τρόπο εκτέλεσης της συνάρτησης, καθώς και το αποτέλεσμά της για τους παρακάτω βαθμούς Fahrenheit $T1 = 100, T2 = 150, T3 = 200$.

```
1  function TC = Fahrenheit2Celsium(TF)
2  %Fahrenheit2Celsium Convert Fahrenheit2 to Celsium
3  TC=5/9 *(TF-32);
4
5  end
```

Λύση άσκησης αυτοαξιολόγησης 9.2

Για να τροποποιήσουμε τη συνάρτηση που κατασκευάσαμε στην Άσκηση αυτοαξιολόγησης 9.1, έτσι ώστε να εξασφαλίσουμε:

- ότι ο χρήστης καλεί τη συνάρτηση χρησιμοποιώντας ένα διάνυσμα και όχι έναν πίνακα τιμών και
- ότι κανένα στοιχείο του παραπάνω διανύσματος δεν είναι μικρότερο από -459.67, τιμή που αντιστοιχεί στο απόλυτο μηδέν στην κλίμακα Fahrenheit,

πρέπει να εισάγουμε δύο κατάλληλους ελέγχους. Αφού η εκφώνηση δεν διευκρινίζει ποιους από τους δύο τρόπους πρέπει να υιοθετήσουμε για τον χειρισμό των σφαλμάτων, μπορούμε αυθαίρετα να επιλέξουμε όποιον επιθυμούμε. Εδώ, επιλέγουμε την προσέγγιση με την εντολή `error`, όπως φαίνεται στον παρακάτω κώδικα.

```
1  function TC = Fahrenheit2Celsium(TF)
2  %Fahrenheit2Celsium Convert Fahrenheit2 to Celsium
3
4  if min(size(TF))~=1
5      error('input error. Please use an array of temperatures')
6  end
7
8  if any(TF<-459.67)
9      error('input error. You can not use a temperature smaller than
           -459.67F')
10 end
11
12 TC=5/9 *(TF-32);
13
14 end
```

Οι εντολές στις γραμμές 4-6 ελέγχουν αν ο το x είναι ένα διάνυσμα και, αν δεν είναι, τότε επιστρέφουν το ανάλογο μήνυμα διακόπτοντας τη συνάρτηση. Ένα παράδειγμα αυτής της περίπτωσης εμφανίζεται στη συνέχεια.

```
 >> TC = Fahrenheit2Celsium([1,2;3,4])
Error using Fahrenheit2Celsium (line 5)
input error. Please use an array of temperatures
```

Από την άλλη, οι εντολές στις γραμμές 8-10 ελέγχουν αν υπάρχει κάποιο στοιχείο στο x που είναι μικρότερο από -459.67. Αν υπάρχει, τότε τυπώνεται το κατάλληλο μήνυμα σφάλματος και διακόπτεται η εκτέλεση της συνάρτησης, όπως φαίνεται και στο ακόλουθο παράδειγμα.

```
>> TC = Fahrenheit2Celsium([1,2,-500])
Error using Fahrenheit2Celsium (line 9)
input error. You can not use a temperature smaller than -459.67F
```

Λύση άσκησης αυτοαξιολόγησης 9.3

Η ισοδύναμη αντίσταση R των αντιστατών σε ένα ηλεκτρικό κύκλωμα που αποτελείται από n αντιστάτες R_i είτε είναι συνδεδεμένοι σε σειρά είτε παράλληλα, μπορεί να υπολογιστεί με τη βοήθεια της παρακάτω συνάρτησης.

```
1  function [TotalR] = TotalResistance(R,type)
2  %TOTALRESISTANCE calcualtes the total resistance of an electric
       circuit
3  %   R contains the n resistors
4  %   type=1 if the resistors are conected in series
5  %   type=2 if the resistors are conected in parallel
6
7   if min(size(R))~=1
8       warning('input error. Please use an array of  the resistors')
9       TotalR=NaN;
10      return
11  end
12
13  if any(R<0)
14      warning('input error. Resistors should have non negative values')
15      TotalR=NaN;
16      return
17  end
18
19  if type~=1 && type~=2
20      warning('input error. Variable type can take only two values: 1 or
            2')
21      TotalR=NaN;
22      return
23  end
24
25  if type==1
```

```
26      TotalR=sum(R);
27  else
28      TotalR=1/sum(1./R);
29  end
30
31  end
```

Στην παραπάνω συνάρτηση στις γραμμές:

- 2-5 έχουμε πληκτρολογήσει τη σχετική βοήθεια της συνάρτησης,
- 7-23 εξασφαλίζουμε ότι η συνάρτηση καλείται με τα σωστά ορίσματα. Στην περίπτωση που δεν συμβαίνει αυτό η συνάρτηση διακόπτεται αφού έχει τυπώσει ένα σχετικό μήνυμα (υπό τη μορφή προειδοποίησης) και αναθέσει στη μεταβλητή εξόδου την τιμή `NaN`,
- 25-29 υπολογίζουμε την ισοδύναμη αντίσταση των αντιστατών λαμβάνοντας υπόψη αν αυτοί είναι συνδεδεμένοι σε σειρά ή παράλληλα.

Στη συνέχεια, παρουσιάζονται τα αποτελέσματα της συνάρτησης για τις περιπτώσεις που ζητάει η εκφώνηση. Οι τρεις τελευταίες περιπτώσεις παράγουν σφάλματα, αφού τα ορίσματα δεν είναι σωστά ορισμένα.

```
>> [TotalR] = TotalResistance([1,3,0.5],1)
TotalR =
    4.5000

>> [TotalR] = TotalResistance([0.5,1.3,0.2],2)
TotalR =
    0.1287

>> [TotalR] = TotalResistance([0.5,1.3;0.2,0.1],1)
Warning: input error. Please use an array of  the resistors
> In TotalResistance (line 8)
TotalR =
   NaN

>> [TotalR] = TotalResistance([0.5,1.3,-0.2],2)
Warning: input error. esistors should have non negative values
> In TotalResistance (line 14)
TotalR =
   NaN

>> [TotalR] = TotalResistance([0.5,1.3,0.2],3)
Warning: input error. Variable type can take only two values: 1
or 2
> In TotalResistance (line 20)
TotalR =
   NaN
```

Λύση άσκησης αυτοαξιολόγησης 9.4

Ένας τρόπος με τον οποίο μπορεί να τροποποιηθεί η συνάρτηση του Παραδείγματος 9.2, έτσι ώστε να επιστρέφει το μέγιστο n αριθμών κάνοντας χρήση μιας επαναληπτικής διαδικασίας και κατάλληλης υποσυνάρτησης για τη σύγκριση δύο αριθμών είναι ο ακόλουθος:

```
1  function M = mymaxshort(x)
2  %MYMAXSHORT returns the max
3  %      MYMAXSHORT calculates the maximum of the
4  %      elements of x (array)
5  %      all elements of x must be real
6      if min(size(x))~=1
7          error('x must be an array')
8      end
9
10     if max(size(x))==1
11         error('x must have at least two elements')
12     end
13
14     if any(imag(x))~=0
15         error('all elements of x must be real')
16     end
17
18     M =  x(1);
19     for k=2:length(x)
20         M=findmaximum(M,x(k));
21     end
22
23 end
24
25 function y = findmaximum(A,B)
26     if A>B
27         y=A;
28     else
29         y=B;
30     end
31 end
```

Στην παραπάνω συνάρτηση στις γραμμές:

- 2-5 έχουμε πληκτρολογήσει τη σχετική βοήθεια της συνάρτησης,
- 6-16 εξασφαλίζουμε ότι η συνάρτηση καλείται με τα σωστά ορίσματα. Στην περίπτωση που δεν συμβαίνει αυτό η συνάρτηση διακόπτεται αφού έχει τυπώσει ένα σχετικό μήνυμα (υπό τη μορφή προειδοποίησης) και αναθέσει στη μεταβλητή εξόδου την τιμή `NaN`,
- 18-21 υπολογίζουμε το μέγιστο των στοιχείων του x, δηλαδή της μεταβλητής εισόδου που περιέχει το διάνυσμα των n πραγματικών αριθμών. Ο υπολογισμός του μεγίστου γίνεται με χρήση της υποσυνάρτησης findmaximum(γραμμές 25-31), η οποία συγκρίνει τους δύο αριθμούς και επιστρέφει το μέγιστό τους. Πιο συγκεκριμένα, η υποσυνάρτηση findmaximum καλείται από την κύρια συνάρτηση στη γραμμή 20 και συγκρίνει την τρέχουσα τιμή του μεγίστου με το k-οστό στοιχείο του x.

Λύση άσκησης αυτοαξιολόγησης 9.5

Για τον υπολογισμό του αριθμού F_n, δηλαδή του n-οστού όρου της ακολουθίας Fibonacci, μπορούμε να βασιστούμε στην ακόλουθη συνάρτηση.

```
1  function Fn = FibonacciSequence(n)
2  %FIBONACCISEQUENCE Calculates the nth term of the Fibonacci Sequence
3  %The Fibonacci numbers are the sequence 1, 1, 2, 3, 5, 8, 13, 21,...
4   if round(n)~=n || n<0
5       warning([num2str(n),' is not a natural number'])
6       Fn=NaN;
7       return
8  end
9
10  if n==1||n==2
11     Fn= 1;
12  else
13     Fn= FibonacciSequence(n-2)+FibonacciSequence(n-1);
14  end
```

Η συνάρτηση αυτή οποτεδήποτε η μεταβλητή εισόδου n είναι ένας φυσικός αριθμός μεγαλύτερος του 2 καλεί εκ νέου τον εαυτό της, δύο φορές μάλιστα, με $(n-1)$ και $(n-2)$ (γραμμή 13), υλοποιώντας με αυτόν τον τρόπο την αναδρομική σχέση ορισμού της ακολουθίας Fibonacci.

Λύση άσκησης αυτοαξιολόγησης 9.6

Για να τροποποιήσουμε τη συνάρτηση του Παραδείγματος 9.3, έτσι ώστε ο χρήστης να έχει τη δυνατότητα, αν το επιθυμεί, να περάσει τις εντολές μορφοποίησης του γραφήματος, το μόνο που χρειάζεται είναι να χρησιμοποιήσουμε την εντολή varargin στις μεταβλητές εισόδου. Η μεταβλητή αυτή θα μπορεί να περιέχει τη μορφοποίηση του γραφήματος, δηλαδή θα μπορεί να περιέχει εντολές της μορφής 'k-*' ή 'c+:', οι οποίες θα χρησιμοποιούνται μαζί με την εντολή plot κατά την κατασκευή του γραφήματος. Ο παρακάτω κώδικας υλοποιεί την παραπάνω διαδικασία.

```
1  function x=randomwalk(x0,low,hi,fig, varargin)
2  %RANDOM WALK Simulate a random walk with absorbing barriers
3  %            x(n+1)=x(n)+step, where step equals
4  %            to -1, 0 or 1 with equal probability
5  %            x(1)=x0, the starting point
6  %            low and hi the absorbing barriers (low<x0<hi)
7  %            set fig=1 to display the figure
8  %            varargin: a character string to format the figure (see
       help
9  %            plot)
10  if low>=x0 || hi<=x0
11      error('inout error!  low<x0<hi should be satisfied')
12  end
13
14  n=1;
15  x(n)=x0;
16  while x(n)>low && x(n)<hi
17      step=datasample([-1,0,1],1);
```

```
18      x(n+1)=x(n)+step;
19      n=n+1;
20  end
21
22  if fig==1
23      if nargin==4
24          plot(x)
25      else
26          plot(x,varargin{1})
27      end
28      ylim([low hi])
29  end
30
31  end
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ξενόγλωσση

Id, Y. and Kennedy, E. S. (1969). A medieval proof of Heron's formula. *The Mathematics Teacher*, *62*(7), pp. 585–587.

Kalechman, M. (2018). *Practical MATLAB Basics for Engineers*. CRC Press, Boca Raton, Florida, USA.

ΚΕΦΑΛΑΙΟ 10

ΣΥΜΒΟΛΙΚΑ ΜΑΘΗΜΑΤΙΚΑ

Σύνοψη:

Σε αυτό το κεφάλαιο παρουσιάζονται οι βασικές εντολές που αφορούν τα συμβολικά μαθηματικά στο Matlab. Συγκεκριμένα, παρουσιάζονται οι τρόποι ορισμού συμβολικών μεταβλητών και συμβολικών συναρτήσεων και οι εντολές για την εύρεση ορίων, την παραγώγιση, την ολοκλήρωση και την επίλυση γραμμικών και μη-γραμμικών αλγεβρικών συστημάτων. Ιδιαίτερη έμφαση δίνεται στην κατανόηση των διαφορών μεταξύ συμβολικών μεταβλητών και μεταβλητών διπλής ακρίβειας, καθώς και στον τρόπο που συνδυάζονται οι δύο αυτές κλάσεις μεταβλητών.

Προαπαιτούμενη γνώση: Τα κεφάλαια 3, 4 και 5 του παρόντος συγγράμματος, καθώς και βασικές γνώσεις μαθηματικής ανάλυσης.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του κεφαλαίου αυτού, θα γνωρίζετε:

- για τη δημιουργία συμβολικών μεταβλητών και συμβολικών συναρτήσεων,
- για την εύρεση ορίων μαθηματικής συνάρτησης,
- για την παραγώγιση μαθηματικής συνάρτησης,
- για την ολοκλήρωση μαθηματικής συνάρτησης,
- για την επίλυση αλγεβρικών συστημάτων.

Γλωσσάριο επιστημονικών όρων

- Ανάπτυγμα μαθηματικής παράστασης
- Ολοκλήρωμα μαθηματικής συνάρτησης
- Όριο μαθηματικής συνάρτησης
- Παραγοντοποίηση πολυωνύμου
- Παράγωγος μαθηματικής
- Σειρά Fourier

10.1 Εισαγωγή

Το Matlab αποτελεί ένα πλήρες πακέτο λογισμικού με το οποίο μπορεί κανείς να έχει ένα ευρύ φάσμα προγραμματιστικών εργαλείων, από τα βασικά (δομές ελέγχου, επαναληπτικές διαδικασίες) μέχρι αρκετά εξειδικευμένα (δημιουργία γραφικών παραστάσεων, δημιουργία γραφικού περιβάλλοντος χρήστη). Στο πλαίσιο αυτό, το Matlab παρέχει τη δυνατότητα πράξεων και υπολογισμών με συμβολικά μαθηματικά, μέσα από την εργαλειοθήκη Symbolic Math Toolbox^{TM}. Η λειτουργία αυτή μπορεί να είναι χρήσιμη για απλές μαθηματικές λειτουργίες εκπαιδευτικού χαρακτήρα ή πιο σύνθετες εφαρμογές, όπως η επίλυση αλγεβρικών συστημάτων και ο υπολογισμός σειρών Fourier. Στις επόμενες παραγράφους θα παρουσιάσουμε τους βασικούς τρόπους ορισμού συμβολικών μεταβλητών και μερικές χρήσιμες εντολές για τις βασικές μαθηματικές λειτουργίες, χωρίς ωστόσο να εξαντλήσουμε το αντικείμενο των συμβολικών μαθηματικών, καθώς αυτό είναι πέρα από τους στόχους του παρόντος συγγράμματος. Σε αυτό το πλαίσιο, θα γίνουν αναφορές σε στοιχεία της μαθηματικής ανάλυσης, για τα οποία μπορεί κανείς να απευθυνθεί σε ένα σχετικό σύγγραμμα, όπως στο σύγγραμμα των Thomas *et al.* (2009).

10.2 Συμβολικές μεταβλητές και συμβολικές συναρτήσεις

Οι συμβολικές μεταβλητές είναι διαφορετικό είδος μεταβλητών σε σχέση με αυτές που έχουμε συναντήσει μέχρι τώρα. Για να το κατανοήσουμε αυτό, ας δούμε το εξής παράδειγμα:

Στο Matlab, αν προσπαθήσουμε να χρησιμοποιήσουμε μία μεταβλητή χωρίς να την έχουμε αρχικοποιήσει, η εκτέλεση του προγράμματος θα διακοπεί και θα εκτυπωθεί κατάλληλο μήνυμα σφάλματος.

```
>> clear
>> x = 2*a+1
Undefined function or variable 'a'.
```

Αυτό συμβαίνει λόγω του ότι κάθε μεταβλητή είναι στην πραγματικότητα ένα μέσο αποθήκευσης μίας αριθμητικής τιμής ή μίας διάταξης τιμών, στην περίπτωση των πινάκων. Αντίθετα, μία συμβολική μεταβλητή αποτελεί ένα προγραμματιστικό αντικείμενο, ανεξάρτητα από το αν της έχουμε δώσει τιμή. Μπορούμε, λοιπόν, να τη χρησιμοποιήσουμε για θεωρητικούς υπολογισμούς, όπως για τον ορισμό μαθηματικών συμβολικών συναρτήσεων, τον υπολογισμό παραγώγων και ολοκληρωμάτων και τη δημιουργία γραφικών παραστάσεων.

10.2.1 Δήλωση συμβολικών μεταβλητών

Μπορούμε να ορίσουμε μία ή περισσότερες συμβολικές μεταβλητές, χρησιμοποιώντας την εντολή `syms`, όπως φαίνεται στη εντολή που ακολουθεί

```
>> syms x y
```

Αν εκτελέσουμε την εντολή `whos`, η οποία εκτυπώνει πληροφορίες για τις μεταβλητές που έχουν οριστεί, θα λάβουμε το αποτέλεσμα

```
>> whos
  Name        Size                Bytes  Class     Attributes

  x           1x1                     8  sym
  y           1x1                     8  sym
```

Παρατηρούμε ότι το Matlab αντιλαμβάνεται τα αντικείμενα x και y που ορίστηκαν ως πίνακες με μέγεθος 1×1 που ανήκουν στην κλάση `sym` (δηλαδή συμβολική μεταβλητή) και όχι στην κλάση `double` (δηλαδή μεταβλητή διπλής ακρίβειας), στην οποία ανήκουν όλες οι μεταβλητές που έχουμε εξετάσει μέχρι το σημείο αυτό. Αυτό σημαίνει ότι οι μεταβλητές x και y αντιμετωπίζονται με ιδιαίτερο τρόπο όταν συμμετέχουν σε μαθηματικές πράξεις και όταν χρησιμοποιούνται σε συνδυασμό με βασικές εντολές του Matlab. Τις διαφορές αυτές θα τις αναλύσουμε στις επόμενες παραγράφους.

Ένας εναλλακτικός τρόπος για να ορίσουμε μία συμβολική μεταβλητή είναι μέσω της εντολής `syms`, ως εξής

```
>> x = sym('x')

x =

x
```

Μπορούμε, επίσης, να ορίσουμε μια συμβολική παράσταση με την εντολή `str2sym`

```
>> y = str2sym('3*x+z')

y =

3*x + z
```

Τέλος, μπορούμε να χρησιμοποιήσουμε μία συμβολική μεταβλητή για να αναπαραστήσουμε μία σταθερά, όπως για παράδειγμα τη χρυσή τομή $\rho = \frac{1+\sqrt{5}}{2}$

```
>> rho = str2sym('(1 + sqrt(5))/2')

rho =

5^(1/2)/2 + 1/2
```

Παρατηρείστε ότι στη μεταβλητή rho αποθηκεύεται η παράσταση που εισαγάγαμε και όχι το αποτέλεσμα της πράξης, όπως θα γινόταν σε μία μεταβλητή διπλής ακρίβειας.

10.2.2 Δήλωση συμβολικών συναρτήσεων

Εκτός από τον ορισμό συμβολικών μεταβλητών, η εντολή `syms` μπορεί να ορίσει συμβολικές συναρτήσεις. Για παράδειγμα, μπορούμε να ορίσουμε την $f(x)$ ως εξής

```
>> syms f(x)
>> whos
  Name        Size                Bytes  Class     Attributes

  f           1x1                     8  symfun
  x           1x1                     8  sym
```

Παρατηρούμε ότι με την παραπάνω εντολή ορίσαμε ταυτόχρονα τη συμβολική μεταβλητή x (κλάση `syms`), καθώς και τη συμβολική συνάρτηση f (κλάση symfun). Ακολούθως, μπορούμε με νέα εντολή να εισάγουμε και τον τύπο της συμβολικής συνάρτησης, ως εξής

```
>> f(x) = x^2 + 2*x + 3

f(x) =

x^2 + 2*x + 3
```

Οι συμβολικές συναρτήσεις μοιάζουν με αυτό που αναφέραμε στην προηγούμενη παράγραφο ως συμβολική παράσταση, ωστόσο υποστηρίζουν περισσότερες λειτουργίες από τις συμβολικές παραστάσεις. Για παράδειγμα, μπορούμε εύκολα να υπολογίσουμε την τιμή μίας συνάρτησης σε ένα σημείο, αντικαθιστώντας στη θέση του ορίσματος την τιμή της μεταβλητής. Συγκεκριμένα, για να βρούμε την τιμή της $f(x)$ που ορίσαμε παραπάνω στο $x = 1$, μπορούμε να γράψουμε

```
>> f(1)

ans =

6
```

Για να κάνουμε κάτι αντίστοιχο σε μία συμβολική παράσταση, πρέπει να χρησιμοποιήσουμε την εντολή `subs`, που θα παρουσιάσουμε σε επόμενη παράγραφο.

Οι συμβολικές συναρτήσεις μπορεί να έχουν παραπάνω από μία συμβολικές μεταβλητές, όπως φαίνεται στο παρακάτω παράδειγμα

```
>> syms f(x,y)
>> f(x,y) = x^2 + y^2;
>> f(1,2)

ans =

5
```

10.2.3 Δήλωση περιορισμών για τις συμβολικές μεταβλητές

Σε πολλές μαθηματικές εφαρμογές χρειάζεται να εισάγουμε περιορισμούς για τις μεταβλητές και τις παραμέτρους που χρησιμοποιούμε. Οι περιορισμοί αυτοί μπορούν να εισαχθούν είτε κατά τον ορισμό της μεταβλητής είτε μεταγενέστερα με χρήση της εντολής `assume`.

Ας υποθέσουμε ότι θέλουμε να ορίσουμε τη συμβολική μιγαδική μεταβλητή $z = x + iy$, όπου οι αριθμοί x και y είναι πραγματικοί, και i είναι η φανταστική μονάδα. Μπορούμε να δημιουργήσουμε αρχικά τις συμβολικές μεταβλητές x και y, τις οποίες θα ορίσουμε να είναι πραγματικές και, στη συνέχεια, να κατασκευάσουμε την z

```
>> syms x y real
>> z=x+i*y

z =

x + y*1i
```

Αν θέλουμε να επιβάλλουμε περιορισμό σε κάποια μεταβλητή αφού την έχουμε δημιουργήσει, μπορούμε να χρησιμοποιήσουμε την εντολή `assume` σε συνδυασμό με την κατάλληλη συνθήκη ή χαρακτηριστική έκφραση

```
>> syms x y
>> assume(x>10)
>> assume(y,'real')
```

Μπορούμε, επίσης, να επιβάλλουμε περισσότερους από έναν περιορισμούς σε μια μεταβλητή, χρησιμοποιώντας την εντολή `assumeAlso`. Για παράδειγμα, αν θέλουμε η μεταβλητή n να συμβολίζει έναν ακέραιο αριθμό μεταξύ του 1 και του 9, μπορούμε να χρησιμοποιήσουμε τις παρακάτω εντολές

```
>> syms n positive
>> assumeAlso(n<10)
>> assumeAlso(n,'integer')
```

Η σειρά με την οποία εισήγαμε τους περιορισμούς είναι τυχαία, θα μπορούσαμε δηλαδή να έχουμε ορίσει την n ως ακέραιο (integer) και, στη συνέχεια, να δηλώσουμε ότι είναι μεγαλύτερη του 0 και μικρότερη του 10.

Για να βεβαιωθούμε ότι έχουν εισαχθεί σωστά όλοι οι περιορισμοί και, γενικότερα, για να ελέγξουμε ποιοι περιορισμοί έχουν δηλωθεί για μία μεταβλητή, μπορούμε να χρησιμοποιήσουμε την εντολή `assumptions`, η οποία εκτυπώνει τις πληροφορίες αυτές

```
>> assumptions(n)

ans =

[ 0 < n, n < 10, in(n, 'integer')]
```

10.2.4 Γραφικές παραστάσεις συμβολικών συναρτήσεων - Εντολές fplot, ezplot, fsurf και ezsurf

Για να δημιουργήσουμε τη γραφική παράσταση μίας συμβολικής συνάρτησης, χρησιμοποιούμε είτε την εντολή `fplot` είτε την `ezplot`, τις οποίες έχουμε αναφέρει και στο Κεφάλαιο 5. Οι εντολές μπορούν να χρησιμοποιηθούν απλώς εισάγοντας στο όρισμα τη συμβολική συνάρτηση ή συμβολική παράσταση, την οποία θέλουμε να απεικονίσουμε, και δημιουργούν το γράφημά της σε προκαθορισμένο διάστημα (η `fplot` στο $[-5,5]$, ενώ η `ezplot` στο $[-\pi,\pi]$). Για παράδειγμα, για να φτιάξουμε τη γραφική παράσταση της $y = \sin|x|^3$ γράφουμε

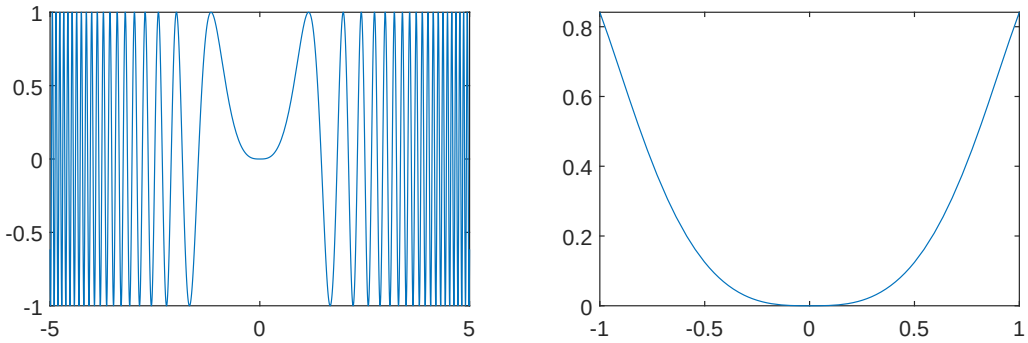
```
>> syms x
>> fplot(sin(abs(x)^3))
```

Για να κατασκευάσουμε τη γραφική παράσταση σε διαφορετικό διάστημα από το προκαθορισμένο, εισάγουμε στο όρισμα το διάστημα υπό τη μορφή διανύσματος, όπως φαίνεται παρακάτω

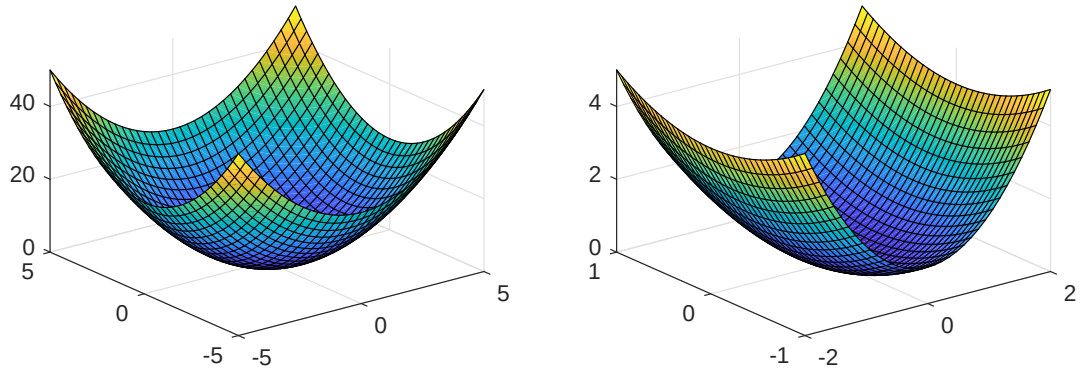
```
>> syms x
>> fplot(sin(abs(x)^3),[-1,1])
```

Οι γραφικές παραστάσεις που δημιουργούν οι παραπάνω γραμμές κώδικα φαίνονται στο Σχήμα 10.1.

Αντίστοιχες εντολές για τη δημιουργία τριδιάστατων γραφικών παραστάσεων είναι οι `fsurf` και `ezsurf`.



Σχήμα 10.1: Γραφική παράσταση της συνάρτησης $y = \sin|x|^3$ στο προκαθορισμένο διάστημα της εντολής `fplot` (αριστερό γράφημα) και στο διάστημα $[-1,1]$ (δεξί γράφημα).



Σχήμα 10.2: Γραφική παράσταση της συνάρτησης $y = x^2 + y^2$ στο προκαθορισμένο διάστημα της εντολής `fsurf` (αριστερό γράφημα) και στο διάστημα $x \in [-2,2]$ και $y \in [-1,1]$ (δεξί γράφημα).

Η σύνταξή τους είναι παρόμοια με αυτή των αντίστοιχων εντολών για τις διδιάστατες γραφικές παραστάσεις, όπως φαίνεται παρακάτω:

```
>> syms x y
>> fsurf(x^2+y^2)
>> fsurf(x^2+y^2,[-2,2,-1,1])
```

Όταν εισάγουμε συγκεκριμένο διάστημα για τη γραφική παράσταση, οι δύο πρώτοι αριθμοί του διανύσματος αντιστοιχούν στην ελάχιστη και στη μέγιστη τιμή του x, ενώ οι δύο τελευταίοι στην ελάχιστη και στη μέγιστη τιμή του y. Τα διαγράμματα που δημιουργούν οι παραπάνω γραμμές κώδικα φαίνονται στο Σχήμα 10.2.

10.3 Ανάλυση με συμβολικά μαθηματικά

Οι συμβολικές μεταβλητές μας δίνουν τη δυνατότητα να εφαρμόζουμε τα εργαλεία της μαθηματικής ανάλυσης, στο Matlab. Μπορούμε να βρούμε όρια συναρτήσεων, να κάνουμε παραγωγίσεις και να υπολογίσουμε ορισμένα και αόριστα ολοκληρώματα χρησιμοποιώντας κατάλληλες εντολές. Τις βασικές εντολές για τις λειτουργίες αυτές θα παρουσιάσουμε στις επόμενες παραγράφους.

10.3.1 Αντικατάσταση συμβολικής μεταβλητής με αριθμητική τιμή - Εντολή subs

Όπως έχει ήδη αναφερθεί, οι συμβολικές μεταβλητές δεν έχουν κάποια συγκεκριμένη αριθμητική τιμή, όπως οι μεταβλητές διπλής ακρίβειας. Ακόμα κι όταν θέλουμε να τις αντικαταστήσουμε με έναν αριθμό, για να υπολογίσουμε για παράδειγμα την τιμή μίας συνάρτησης, δεν μπορούμε να χρησιμοποιήσουμε τον τελεστή "=". Αυτό συμβαίνει λόγω του ότι ο τελεστής "=" μετατρέπει αυτόματα τη συμβολική μεταβλητή σε μεταβλητή διπλής ακρίβειας αλλάζοντας την κλάση της. Ας δούμε ένα παράδειγμα:

```
>> syms x
>> whos
  Name        Size              Bytes  Class     Attributes

  x           1x1                   8  sym
```

Παρατηρούμε ότι με την εντολή `syms` έχουμε ορίσει τη συμβολική μεταβλητή x. Αν τώρα προσπαθήσουμε να δώσουμε τιμή στην x χρησιμοποιώντας τον τελεστή "=", την μετατρέπουμε σε μεταβλητή διπλής ακρίβειας (double).

```
>> x = 5;
>> whos
  Name        Size              Bytes  Class     Attributes

  x           1x1                   8  double
```

Πώς μπορούμε, λοιπόν, να αντικαταστήσουμε την x με μία αριθμητική τιμή; Η απάντηση είναι χρησιμοποιώντας την εντολή `subs`. Η εντολή αυτή συντάσσεται με τη μορφή `subs(f,x,`αριθμός`)`, όπου f είναι η συνάρτηση ή η μαθηματική παράσταση την οποία θέλουμε να υπολογίσουμε και x είναι η μεταβλητή που θέλουμε να αντικαταστήσουμε με τον αριθμό που υπάρχει στο όρισμα. Ας δούμε ένα παράδειγμα:

```
>> syms f x
>> f = log(2*x) + exp(-x);
>> subs(f,x,0.5)

ans =

exp(-1/2)
```

Με τις παραπάνω εντολές ορίσαμε τη συμβολική παράσταση f, η οποία περιέχει τη συμβολική μεταβλητή x και, στη συνέχεια, δώσαμε τον τύπο της. Τέλος, αντικαταστήσαμε τη μεταβλητή x με την τιμή 0.5, για να πάρουμε το αποτέλεσμα `exp(-1/2)` (θυμίζουμε ότι $\log(1) = 0$).

Σημειώστε ότι στο συγκεκριμένο παράδειγμα η f έχει οριστεί σαν συμβολική παράσταση και όχι σαν συμβολική συνάρτηση. Για τον λόγο αυτό δεν μπορούμε να εισάγουμε απλώς την τιμή της x μέσα σε παρενθέσεις.

Στην περίπτωση που έχουμε συνάρτηση ή παράσταση με παραπάνω από μία μεταβλητή, τότε μπορούμε να κάνουμε πολλαπλή αντικατάσταση στην ίδια εντολή `subs`, χρησιμοποιώντας διανύσματα στο όρισμα

```
>> syms a b
>> syms n integer
>> subs(cos(n*a) + sin(n*b), [a, b], [1, 2])

ans =
```

```
sin(2*n) + cos(n)
```

10.3.2 Υπολογισμός αριθμητικής τιμής συμβολικής παράστασης - Εντολή double

Είδαμε στην προηγούμενη παράγραφο πώς μπορούμε να αντικαταστήσουμε μία ή περισσότερες μεταβλητές σε μία συμβολική παράσταση χρησιμοποιώντας την εντολή `subs`. Ωστόσο, σε πολλές περιπτώσεις, το Matlab επιστρέφει την αριθμητική παράσταση και όχι το τελικό αποτέλεσμα που προκύπτει μετά τις πράξεις. Για παράδειγμα, όταν αντικαταστήσαμε την τιμή 0.5 στην παράσταση `f = log(2*x) + exp(-x)`, το αποτέλεσμα ήταν `exp(-1/2)`. Αν θέλουμε να μετατρέψουμε τον τύπο αυτό σε αριθμητική τιμή, μπορούμε να τον εισάγουμε στην εντολή `double`. Η εντολή αυτή μετατρέπει το αποτέλεσμα από παράσταση συμβολικών μαθηματικών σε μεταβλητή διπλής ακρίβειας, όπως φαίνεται παρακάτω:

```
>> syms f x
>> f = log(2*x) + exp(-x);
>> double(subs(f,x,0.5))

ans =

    0.6065
```

10.3.3 Όρια συναρτήσεων - Εντολή limit

Ο υπολογισμός ορίων συναρτήσεων στο Matlab γίνεται με την εντολή `limit`. Το όρισμα της `limit` πρέπει να περιλαμβάνει μια συμβολική συνάρτηση ή μια παράσταση συμβολικών μεταβλητών τις οποίες έχουμε ήδη ορίσει, καθώς και πρόσθετες πληροφορίες για το όριο, όπως το σημείο στο οποίο θα το υπολογίσουμε, αν θα είναι πλευρικό κ.λπ. Ας εξετάσουμε μερικά παραδείγματα εύρεσης ορίου. Για να χρησιμοποιήσουμε την εντολή `limit`, πρέπει πρώτα να έχουμε ορίσει τις κατάλληλες συμβολικές μεταβλητές. Ως όρισμα μπορούμε είτε να χρησιμοποιήσουμε μια συμβολική συνάρτηση ή συμβολική παράσταση που έχουμε ήδη ορίσει είτε να εισάγουμε απευθείας τον τύπο της συνάρτησης, όπως φαίνεται στις παρακάτω εντολές:

```
>> syms x
>> limit(log(x)/(x-1),x,1)

ans =

1
```

με τη βοήθεια των οποίων υπολογίζουμε το $\lim_{x\to 1} \frac{\log x}{x-1}$.

Παρατηρούμε ότι στη δεύτερη θέση του ορίσματος της εντολής `limit` υπάρχει η ανεξάρτητη μεταβλητή και στην τρίτη θέση η τιμή στην οποία τείνει.

Για να υπολογίσουμε την τιμή ενός ορίου στο άπειρο χρησιμοποιούμε την έκφραση Inf. Για παράδειγμα, το όριο της $\tan^{-1} x$ στο $-\infty$ βρίσκεται με την εντολή:

```
>> syms f(x)
```

```
>> f(x) = atan(x);
>> limit(f,x,-Inf)

ans =

-pi/2
```

Το Matlab μπορεί, επίσης, να επεξεργαστεί απροσδιόριστες μορφές, όπως $\frac{0}{0}, \frac{\infty}{\infty}, \infty - \infty, 1^{\infty}, 0^{0}$ κ.λπ.

```
>> limit((1+1/x)^x,x,Inf)

ans =

exp(1)
```

Αν η απροσδιόριστη μορφή δεν λύνεται, τότε το αποτέλεσμα είναι NaN,

```
>> limit((x-x)/(x^2-x^2),x,Inf)

ans =

NaN
```

Αν, όμως, το όριο απειρίζεται, τότε το αποτέλεσμα είναι Inf.

Επίσης, μπορούμε να υπολογίσουμε πλευρικά όρια χρησιμοποιώντας τις δεσμευμένες λέξεις `'left'` και `'right'`.

```
>> limit(1/x,x,0,'right')

ans =

Inf

>> limit(1/x,x,0,'left')

ans =

-Inf
```

Παράδειγμα 10.1

Σύμφωνα με τον ορισμό της παραγώγου, η παράγωγος μίας συνάρτησης $f(x)$ στο σημείο x_0 είναι το όριο

$$\lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Δημιουργήστε μία function με όνομα defDer, η οποία θα δέχεται ως ορίσματα τον τύπο της συνάρτησης $f(x)$ υπό τη μορφή σειράς χαρακτήρων, και την τιμή του x_0 και θα υπολογίζει την τιμή της παραγώγου χρησιμοποιώντας το παραπάνω όριο. (Θα χρησιμοποιούμε τον όρο "function" στην άσκηση αυτή για να αποφύγουμε πιθανή σύγχυση με τη μαθηματική συνάρτηση του προβλήματος, καθώς και με τον όρο

συμβολική συνάρτηση).

Λύση Παραδείγματος 10.1

Αρχικοποίηση μεταβλητών: Σύμφωνα με την εκφώνηση, οι είσοδοι που θα δέχεται η function είναι ο τύπος της συνάρτησης $f(x)$ και η τιμή του x_0. Για τις εισόδους αυτές θα χρειαστούμε δύο μεταβλητές (π.χ. fun και x0), τις οποίες θα εισάγουμε στο όρισμα της function. Για έξοδο χρειαζόμαστε μία μεταβλητή με το αποτέλεσμα που θα προκύψει από τον υπολογισμό του ορίου (π.χ. result). Εξετάζοντας τον ορισμό της παραγώγου, παρατηρούμε ότι θα πρέπει να ορίσουμε τη συμβολική συνάρτηση f(x) (άρα εμμέσως και τη συμβολική μεταβλητή x), στην οποία θα εισάγουμε τον τύπο της συνάρτησης. Επίσης, θα πρέπει να ορίσουμε τη συμβολική μεταβλητή h, η οποία εμφανίζεται στον τύπο της παραγώγου. Οι παραπάνω συμβολικές μεταβλητές και συναρτήσεις θα οριστούν με την εντολή `syms`. Στη συνέχεια, θα πρέπει να εισάγουμε τον τύπο της συνάρτησης, ο οποίος είναι αποθηκευμένος σαν σειρά χαρακτήρων στη μεταβλητή fun στη συμβολική συνάρτηση f(x), χρησιμοποιώντας την εντολή `str2sym`.

Κύριος κώδικας: Για το κύριο πρόγραμμα θα χρειαστούμε μόνο την εκτέλεση μίας εντολής `limit`, η οποία θα υπολογίζει το όριο της παράστασης της παραγώγου όταν το h τείνει στο μηδέν. Μπορούμε, προαιρετικά, να χρησιμοποιήσουμε μία βοηθητική μεταβλητή (π.χ. df), στην οποία θα αποθηκεύσουμε την παράσταση που υπάρχει μέσα στο όριο, για να είναι πιο ευανάγνωστος ο κώδικας. Αν χρησιμοποιήσουμε τη βοηθητική αυτή μεταβλητή, στο όρισμα της `limit` θα εισάγουμε την df, το h και την τιμή μηδέν. Το αποτέλεσμα θα αποθηκεύεται στη μεταβλητή result, την οποία θα επιστρέφει η function όταν αυτή ολοκληρωθεί.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  function [result] = defDer(fun, x0)
2  %Calculated the derivative using the definition
3
4  % Initialization
5  syms f(x) h;
6  f(x) = str2sym(fun);
7
8  % Main code
9  df = ( f(x0+h) - f(x0) ) / h;
10 result = limit(df, h, 0);
11 end
```

Για να δούμε πώς μπορούμε να καλέσουμε την παραπάνω συνάρτηση, θα υπολογίσουμε την παράγωγο της $f(x) = x^2$ στο σημείο $x_0 = 4$.

```
>> defDer('x^2', 4)

ans =

8
```

Μπορούμε, επίσης, να υπολογίσουμε την παράγωγο σε περισσότερα από ένα σημεία, εισάγοντας στο όρισμα της συνάρτησης ένα διάνυσμα με τιμές, όπως φαίνεται παρακάτω

```
>> defDer('x^2', [0:4])

ans =

[ 0, 2, 4, 6, 8]
```

10.3.4 Παράγωγος συνάρτησης - Εντολή diff

Είδαμε στο παράδειγμα 10.2 πώς μπορούμε να υπολογίσουμε την παράγωγο μίας συνάρτησης σε ένα ή περισσότερα σημεία. Ωστόσο, για πολλές μαθηματικές εφαρμογές χρειαζόμαστε τον τύπο της συνάρτησης της παραγώγου, και όχι της τιμής της. Για τη λειτουργία αυτή μπορούμε να χρησιμοποιήσουμε την εντολή `diff`. Στο όρισμα της `diff` μπορούμε να εισάγουμε μία συμβολική συνάρτηση ή παράσταση που έχουμε ορίσει προηγουμένως ή να εισάγουμε απευθείας τον τύπο της συνάρτησης που θέλουμε να παραγωγίσουμε, όπως φαίνεται παρακάτω:

```
>> syms x
>> diff(sin(1/x))

ans =

-cos(1/x)/x^2
```

Μπορούμε, επίσης, να βρούμε παραγώγους ανώτερης τάξης, εισάγοντας την τάξη της παραγώγου στο όρισμα της `diff`, ως εξής:

```
>> diff(log(x),2)

ans =

-1/x^2
```

Τέλος, υπάρχει η δυνατότητα υπολογισμού μερικών παραγώγων σε συναρτήσεις πολλών μεταβλητών, αν περιλάβουμε στο όρισμα της `diff` τη μεταβλητή ως προς την οποία θέλουμε να παραγωγίσουμε. Για παράδειγμα, ας θεωρήσουμε τη συνάρτηση δύο μεταβλητών $f(x,y) = e^{xy}$

```
>> syms f(x,y)
>> f(x) = exp(x*y)

f(x) =

exp(x*y)
```

Η μερική παράγωγος ως προς x υπολογίζεται με την εντολή

```
>> diff(f,x)

ans(x) =

y*exp(x*y)
```

ενώ η μερική παράγωγος ως προς y υπολογίζεται με την εντολή

```
>> diff(f,y)

ans(x) =

x*exp(x*y)
```

Τέλος, αν θέλουμε να υπολογίσουμε τη μερική παράγωγο δεύτερης τάξης ως προς y, γράφουμε

```
>> diff(f,y,2)

ans(x) =

x^2*exp(x*y)
```

10.3.5 Ολοκλήρωμα συνάρτησης - Εντολή int

Η αντίστροφη πράξη της παραγώγισης είναι η ολοκλήρωση. Στα συμβολικά μαθηματικά του Matlab η πράξη της ολοκλήρωσης γίνεται με την εντολή `int`. Μπορούμε να υπολογίσουμε το αόριστο ολοκλήρωμα μίας συνάρτησης εισάγοντας στο όρισμα της εντολής `int` μία συμβολική συνάρτηση ή παράσταση που έχουμε ορίσει προηγουμένως ή απευθείας τον τύπο της συνάρτησης που θέλουμε να ολοκληρώσουμε, όπως φαίνεται παρακάτω:

```
>> syms x
>> int(1/x)

ans =

log(x)

>> int(exp(-2*x))

ans =

-exp(-2*x)/2
```

Για να υπολογίσουμε το ορισμένο ολοκλήρωμα συνάρτησης, πρέπει να εισάγουμε στο όρισμα της `int` τα όρια ολοκλήρωσης αμέσως μετά τη συνάρτηση, όπως φαίνεται παρακάτω:

```
>> int(exp(x),0,1)

ans =

exp(1) - 1

>> int(x*log(x),1,2)

ans =

log(4) - 3/4
```

Αν θέλουμε να υπολογίσουμε το αριθμητικό αποτέλεσμα του ολοκληρώματος, μπορούμε να μετατρέψουμε την έκφραση συμβολικών μαθηματικών σε μεταβλητή διπλής ακρίβειας, χρησιμοποιώντας την εντολή `double`.

```
>> double(int(x*log(x),1,2))

ans =

    0.6363
```

Με τα συμβολικά μαθηματικά του Matlab μπορούμε, επίσης, να υπολογίσουμε γενικευμένα ολοκληρώματα, δηλαδή ορισμένα ολοκληρώματα όπου ένα ή και τα δύο όρια ολοκλήρωσης τείνουν στο άπειρο, χρησιμοποιώντας την έκφραση Inf.

```
>> int(exp(-x),0,Inf)

ans =

1
```

Στην περίπτωση που το γενικευμένο ολοκλήρωμα δεν δίνει πεπερασμένο αποτέλεσμα, τότε η απάντηση είναι Inf (ή -Inf).

```
>> int(1/x,1,Inf)

ans =

Inf
```

Τέλος, αν έχουμε συνάρτηση πολλών μεταβλητών, μπορούμε να επιλέξουμε τη μεταβλητή ως προς την οποία θα ολοκληρώσουμε, τοποθετώντας την στο όρισμα της `int` ανάμεσα στον τύπο της συνάρτησης και τα όρια ολοκλήρωσης. Για παράδειγμα, για να ολοκληρώσουμε την $f(x,y) = 3x^2 + 2y$ από το $x = 0$ έως το $x = 1$, μπορούμε να γράψουμε

```
>> int(3*x^2+2*y,x,0,1)

ans =

2*y + 1
```

Παρατηρούμε ότι η μεταβλητή y αντιμετωπίζεται σαν σταθερά κατά την ολοκλήρωση. Για να υπολογίσουμε το διπλό ολοκλήρωμα της f, για παράδειγμα το $\int_2^3 \int_0^1 3x^2 + 2y\,dx\,dy$, θα πρέπει να χρησιμοποιήσουμε δύο φορές την εντολή `int`, όπως φαίνεται παρακάτω:

```
>> int(int(3*x^2+2*y,x,0,1),2,3)

ans =

6
```

Παράδειγμα 10.2

Οι σειρές Fourier είναι ένα πολύ χρήσιμο μαθηματικό εργαλείο, καθώς μας επιτρέπουν να προσεγγίσουμε μία συνάρτηση με ένα άθροισμα τριγωνομετρικών συναρτήσεων. Το ανάπτυγμα σε σειρά Fourier μίας συνάρτησης $f(x)$, που είναι ορισμένη στο διάστημα $[-1,1]$, έχει τη μορφή

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos n\pi x + b_n \sin n\pi x)$$

Οι συντελεστές a_0, a_n και b_n μπορούν να υπολογιστούν από τα ορισμένα ολοκληρώματα

$$a_0 = \int_{-1}^{1} f(x)\,dx$$

$$a_n = \int_{-1}^{1} f(x) \cos n\pi x\,dx$$

και

$$b_n = \int_{-1}^{1} f(x) \sin n\pi x\,dx$$

τα οποία ονομάζονται τύποι του Euler.

Δεν θα αναφέρουμε στο σημείο αυτό περισσότερα για τις σειρές Fourier (πότε συγκλίνουν και για τις συνθήκες που πρέπει να ικανοποιεί η συνάρτηση που προσεγγίζουμε κ.λπ.), καθώς ξεπερνούν τις ανάγκες του συγκεκριμένου Παραδείγματος (για περισσότερες πληροφορίες κάθε ενδιαφερόμενος/μενη μπορεί να απευθυνθεί, παραδείγματος χάριν, στο σύγγραμμα των Jeffreys *et al.*, 1999).

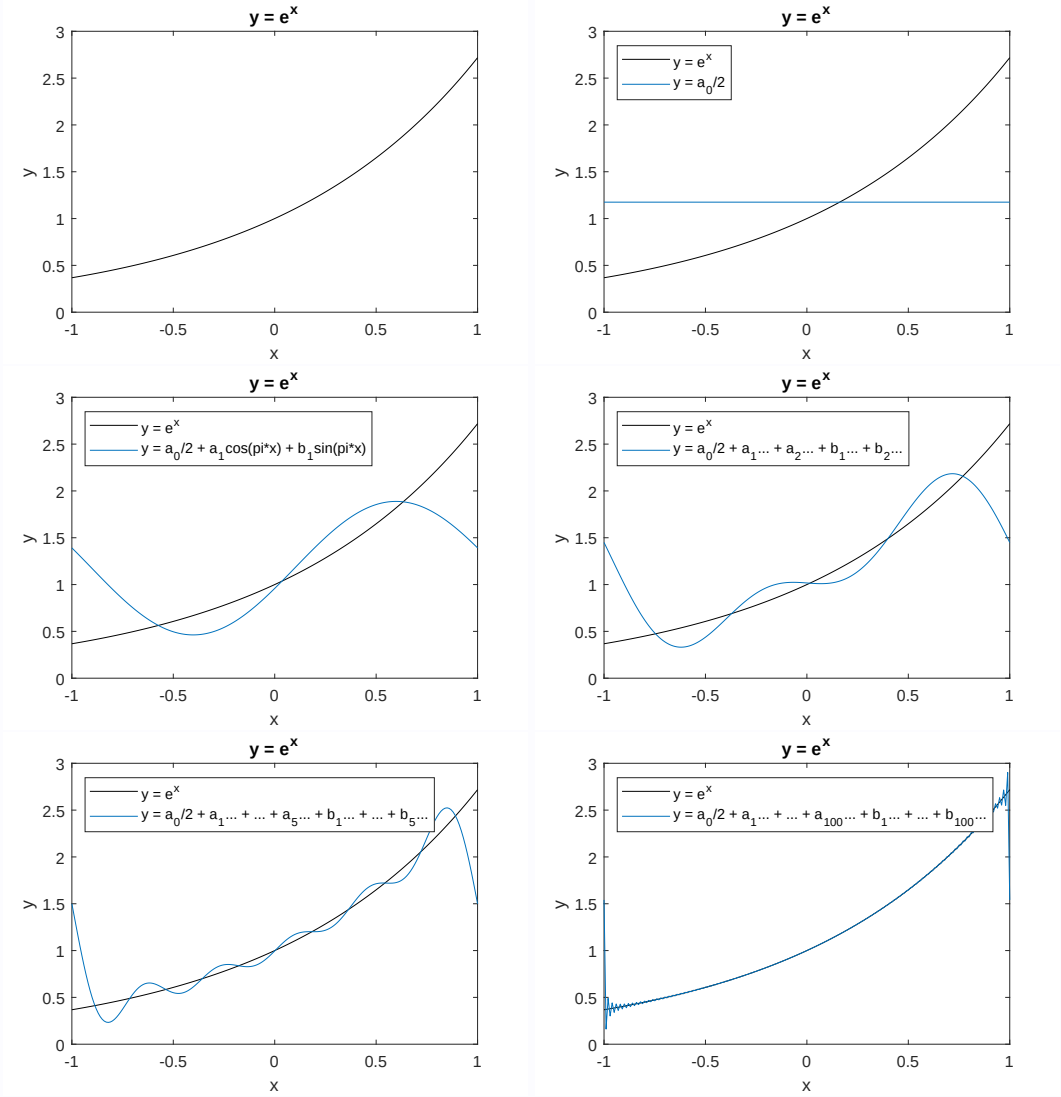
Ωστόσο, για να γίνει εποπτικά κατανοητή η διαδικασία προσέγγισης της συνάρτησης, παρουσιάζεται στο Σχήμα 10.3 η γραφική παράσταση της συνάρτησης

$$y = e^x$$

και οι προσεγγίσεις της με τους όρους της σειράς Fourier. Παρατηρούμε ότι όσο περισσότερους όρους υπολογίσουμε, τόσο καλύτερη είναι η προσέγγιση της συνάρτησης και τόσο μικρότερη η απόκλιση της προσεγγιστικής από την ακριβή τιμή. Με βάση τα παραπάνω, στο παράδειγμα αυτό θα δημιουργήσουμε ένα πρόγραμμα που θα υπολογίζει τους τύπους του Euler, δηλαδή τα ολοκληρώματα που δίνουν τους συντελεστές της σειράς Fourier.

Ζητείται, επομένως, η δημιουργία ενός αρχείου script που θα προσεγγίζει την τιμή της συνάρτησης $y = e^x$ στο διάστημα $[-1,1]$ με σειρά Fourier.

Συγκεκριμένα, ο κώδικας θα πρέπει να υπολογίζει τους N πρώτους όρους της σειράς και θα δημιουργεί τη γραφική τους παράσταση. Τον αριθμό N θα τον δίνει ο χρήστης. Για επαλήθευση στο διάγραμμα θα περιλαμβάνεται και η γραφική παράσταση της συνάρτησης $y = e^x$.



Σχήμα 10.3: Προσέγγιση της συνάρτησης $y = e^x$ από σειρά Fourier. Όσο περισσότερους όρους της σειράς υπολογίσουμε, τόσο καλύτερη η προσέγγιση.

Λύση Παραδείγματος 10.2

Αρχικοποίηση μεταβλητών: Για το πρόβλημα αυτό θα χρειαστούμε συμβολικές μεταβλητές αλλά και μεταβλητές διπλής ακρίβειας. Για να μην υπάρξει σύγχυση μεταξύ των δύο κλάσεων μεταβλητών, τα ονόματα των μεταβλητών διπλής ακρίβειας θα έχουν το γράμμα D στο τέλος.

Ξεκινώντας από τις συμβολικές μεταβλητές, παρατηρούμε ότι τα ολοκληρώματα περιέχουν συναρτήσεις του x, καθώς και την παράμετρο n, η οποία είναι θετικός ακέραιος. Θα ορίσουμε, λοιπόν, αυτές τις δύο συμβολικές μεταβλητές και θα επιβάλουμε τους περιορισμούς για το n, χρησιμοποιώντας την εντολή `assume`. Επίσης, μπορούμε προαιρετικά να ορίσουμε τις a0, an και bn ή, αλλιώς, να τις αφήσουμε να οριστούν αυτόματα κατά τον υπολογισμό των ολοκληρωμάτων. Θα επιλέξουμε το πρώτο, ώστε να μπορεί κανείς να δει όλες τις χρησιμοποιούμενες μεταβλητές του προγράμματος στην αρχή του κώδικα. Σε σχέση με τις μεταβλητές διπλής ακρίβειας, θα χρειαστούμε την ND, που θα περιέχει τον αριθμό των όρων της σειράς Fourier και θα εισάγεται από τον χρήστη με την εντολή input, καθώς και το διάνυσμα xD που θα περιέχει της τιμές του x που θα χρειαστούμε για τη δημιουργία των γραφικών παραστάσεων. Τέλος, θα ορίσουμε μία μεταβλητή, η οποία θα περιέχει το άθροισμα της σειράς Fourier (π.χ. η FourierD).

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να υπολογίσει τα τρία ολοκληρώματα για τα a_0, a_n και b_n, τα οποία θα αποθηκεύσουμε στις συμβολικές μεταβλητές a0, an και bn. Το αποτέλεσμα της ολοκλήρωσης για το a_0 θα είναι μία τιμή, ενώ για τα a_n και b_n θα είναι παραστάσεις της παραμέτρου n.
Ο υπολογισμός της σειράς Fourier θα ξεκινήσει από τον όρο a_0. Εφόσον η μεταβλητή a0 είναι συμβολική, για να μετατρέψουμε την τιμή της σε διπλής ακρίβειας και να την αναθέσουμε στη μεταβλητή FourierD, θα πρέπει να χρησιμοποιήσουμε την εντολή `double`.
Αφού δώσουμε στην FourierD την πρώτη της τιμή, θα πρέπει να εκτελέσουμε έναν επαναληπτικό βρόχο για να υπολογίσουμε τους ND όρους της σειράς. Σε κάθε επανάληψη του βρόχου, θα πρέπει να γίνονται τα εξής:

- να αντικαθιστούμε μέσα στα an και bn, στη θέση της τιμής του n, τον αριθμό της τρέχουσας επανάληψης με την εντολή `subs`,
- να μετατρέπουμε τις παραπάνω συμβολικές παραστάσεις σε διπλής ακρίβειας με την εντολή `double`,
- να πολλαπλασιάζουμε τα παραπάνω αποτελέσματα με $\cos n\pi x$ και $\sin n\pi x$, αντίστοιχα, όπου x θα είναι το διάνυσμα xD,
- να προσθέτουμε το συνολικό αποτέλεσμα στην τρέχουσα τιμή του αθροίσματος FourierD.

Τα τέσσερα παραπάνω βήματα μπορούν να συμπτυχθούν σε μία μόνο εντολή, η οποία θα εκτελεί όλες τις λειτουργίες. Μόλις ολοκληρωθεί ο επαναληπτικός βρόχος, θα έχει αποθηκευτεί στη μεταβλητή FourierD το αποτέλεσμα της σειράς Fourier για κάθε τιμή x του διανύσματος xD. Συνεπώς, μπορούμε να δημιουργήσουμε τη γραφική παράσταση της σειράς Fourier, χρησιμοποιώντας μία εντολή plot με τις μεταβλητές xD και FourierD.

Επαλήθευση: Στο συγκεκριμένο παράδειγμα μπορούμε να συγκρίνουμε το αποτέλεσμα με τη γραφική παράσταση της συνάρτησης που θέλουμε να προσεγγίσουμε. Μπορούμε, λοιπόν, μετά το τέλος του κύριου κώδικα να υπολογίσουμε τη γραφική παράσταση της $y = e^x$ στο $[-1,1]$ και να την τοποθετήσουμε στο ίδιο διάγραμμα με τη σειρά Fourier.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %FOURIER SERIES OF y = exp(x) in [-1,1]
2  % Initialization
3     clear all
4     % Symbolic
5     syms x n a0 an bn
6     assume(n,'integer')
7     assume(n,'positive')
8     % Double
9     ND = input('Give number of Fourier terms: ');
10    xD = -1:0.01:1;
11    FourierD = 0;
12
13 % Main code
14  a0 = int(exp(x),-1,1);
15  an = int(exp(x)*cos(n*pi*x),x,-1,1);
16  bn = int(exp(x)*sin(n*pi*x),x,-1,1);
```

```
17
18    FourierD = double(a0/2);
19    for i=1:ND
20      FourierD = FourierD + double(subs(an,i)) * cos(i*pi*xD) +...
21                            double(subs(bn,i)) * sin(i*pi*xD);
22    end
23
24    plot(xD, FourierD)
25
26    % Validation
27    hold
28    fplot('exp(x)',[-1,1])
```

Άσκηση αυτοαξιολόγησης 10.1

Στη λύση του Παραδείγματος 10.2 επιλέξαμε να εισάγουμε τη συνάρτηση e^x που αναπτύσσουμε σε σειρά Fourier, υπό τη μορφή συμβολικής παράστασης exp(x) σε κάθε εντολή που τη χρειαζόταν (`int`, `fplot` κ.λπ.). Ο τρόπος αυτός δυσχεραίνει τη γενίκευση του κώδικα σε άλλες συναρτήσεις, καθώς θα πρέπει να γίνουν οι κατάλληλες αλλαγές σε πολλές γραμμές του προγράμματος.
Δοκιμάστε να μετατρέψετε τη λύση του Παραδείγματος 10.2, ώστε η συνάρτηση να αρχικοποιείται στην αρχή του προγράμματος σαν μια συμβολική συνάρτηση f(x), τον τύπο της οποίας θα δίνει ο χρήστης υπό τη μορφή σειράς χαρακτήρων. Δοκιμάστε αν λειτουργεί σωστά ο κώδικας, εισάγοντας τη συνάρτηση f(x) = abs(x).

10.3.6 Απλοποίηση παράστασης συμβολικών μαθηματικών - Εντολή simplify

Είδαμε στις προηγούμενες παραγράφους ότι το Matlab έχει τη δυνατότητα να κάνει πολύπλοκους υπολογισμούς ολοκληρωμάτων και παραγώγων, με συναρτήσεις μίας ή πολλών μεταβλητών οι οποίες μπορεί να περιέχουν και παραμέτρους. Οι υπολογισμοί αυτοί δίνουν πολλές φορές πολύπλοκες μαθηματικές παραστάσεις οι οποίες επιδέχονται απλοποίηση. Η απλοποίηση σε παραστάσεις συμβολικών μαθηματικών γίνεται με την εντολή simplify. Για παράδειγμα, η παραγώγιση της συνάρτησης $\frac{x^3-1}{x^2}$ θα μας δώσει το αποτέλεσμα:

```
>> syms x
>> a=diff((x^3-1)/x^2,x)

a =

3 - (2*(x^3 - 1))/x^3
```

Η παραπάνω έκφραση μπορεί να γραφτεί με απλούστερο τρόπο ως εξής:

```
>> simplify(a)

ans =

(x^3 + 2)/x^3
```

10.4 Εργαλεία συμβολικών μαθηματικών

Η χρήση συμβολικών μαθηματικών στο Matlab, εκτός από τη μαθηματική ανάλυση, έχει εφαρμογές και στην άλγεβρα, σε προβλήματα εύρεσης ριζών και επίλυσης συστημάτων. Στις επόμενες παραγράφους θα περιγράψουμε τις εντολές που αφορούν την παραγοντοποίηση πολυωνύμων, στην ανάπτυξη γινομένων και στην επίλυση εξισώσεων ή συστημάτων εξισώσεων συμβολικών μεταβλητών.

10.4.1 Ανάπτυγμα γινομένου/μαθηματικής παράστασης - Εντολή expand

Η εντολή `expand` μπορεί να αναπτύξει μαθηματική παράσταση εκτελώντας τις πράξεις που μπορούν να γίνουν και εφαρμόζοντας κατάλληλες μαθηματικές ταυτότητες. Για παράδειγμα, η έκφραση $(x+1)^4$ μπορεί να αναπτυχθεί μέσω του διωνυμικού αναπτύγματος (βλ. Thomas *et al.*, 2009), ως εξής:

```
>> expand((x+1)^4)

ans =

x^4 + 4*x^3 + 6*x^2 + 4*x + 1
```

Επίσης, η τριγωνομετρική συνάρτηση $cos(x+y)$ αναπτύσσεται με χρήση γνωστής ταυτότητας, ως εξής

```
>> expand(cos(x+y))

ans =

cos(x)*cos(y) - sin(x)*sin(y)
```

Τέλος, η εκθετική συνάρτηση $exp(x+y)$ μπορεί να γραφτεί ως γινόμενο εκθετικών συναρτήσεων, ως εξής:

```
>> expand(exp(x+y))

ans =

exp(x)*exp(y)
```

10.4.2 Παραγοντοποίηση πολυωνύμου - Εντολή factor

Η εντολή `factor` χρησιμοποιείται για να μετατρέψει ένα πολυώνυμο σε γινόμενο παραγόντων. Μερικά παραδείγματα παρουσιάζονται παρακάτω:

```
>> factor(x^3 + 3*x^2 + 3*x + 1)

ans =

[ x + 1, x + 1, x + 1]
```

```
>> factor(x^7-1)

ans =

(x - 1)*(x^6 + x^5 + x^4 + x^3 + x^2 + x + 1)
```

10.4.3 Επίλυση εξισώσεων και συστημάτων - Εντολή solve

Με την εντολή `solve` μπορούμε να βρούμε τις ρίζες μιας εξίσωσης. Αν μέσα στο όρισμα της εντολής τοποθετήσουμε μια παράσταση συμβολικών μαθηματικών, το Matlab την θέτει αυτόματα ίση με το μηδέν και υπολογίζει τις ρίζες. Για παράδειγμα, για να βρούμε τις ρίζες της εξίσωσης $3 * x^2 - 5 = 0$ γράφουμε

```
>> solve(3*x^2-5)

ans =

 -15^(1/2)/3
  15^(1/2)/3
```

Οι δύο ρίζες δίνονται υπό τη μορφή συμβολικών μεταβλητών και μπορούν να μετατραπούν σε μεταβλητές διπλής ακρίβειας με την εντολή `double`.

Η εντολή `solve` μπορεί να υπολογίσει και μιγαδικές ρίζες. Αν, για παράδειγμα, έχουμε την εξίσωση $3 * x^2 + 5 = 0$, η οποία έχει μιγαδικές ρίζες, το αποτέλεσμα της `solve` είναι

```
>> solve(3*x^2+5)

ans =

 -(15^(1/2)*1i)/3
  (15^(1/2)*1i)/3
```

Στην περίπτωση που επιθυμούμε το δεξί μέλος της εξίσωσης να μην είναι μηδέν μπορούμε να χρησιμοποιήσουμε τον τελεστή "==" (διπλό ίσον, όπως και στους ελέγχους) και να εισάγουμε την επιθυμητή τιμή ή τη μεταβλητή.

```
>> syms r
>> solve(r^2==-1)

ans =

 -1i
  1i
```

Αν θέλουμε να επιβάλλουμε περιορισμούς στη λύση μας, τότε αυτοί εισάγονται κατά τον ορισμό των συμβολικών μεταβλητών (εντολή `syms`) είτε μεταγενέστερα, μέσω της εντολής `assume`. Για παράδειγμα, αν θέλουμε μόνο τις θετικές ρίζες της εξίσωσης $t^2 - 1 = 0$, μπορούμε να γράψουμε

```
>> syms t positive
>> solve(t^2-1)

ans =

1
```

Αν θέλουμε την αντίθετη λειτουργία από αυτή που αναφέραμε, δηλαδή να αγνοήσουμε του περιορισμούς που έχουν οριστεί για μια μεταβλητή, τότε μπορούμε να χρησιμοποιήσουμε την έκφραση 'IgnoreProperties', true μέσα στο όρισμα της εντολής `solve`. Για παράδειγμα, αν θέλαμε να αγνοήσουμε τον περιορισμό $t > 0$ που επιβάλαμε στο παραπάνω παράδειγμα, ώστε να πάρουμε *και* την αρνητική ρίζα, θα γράφαμε

```
>> solve(t^2-1, 'IgnoreProperties', true)

ans =

 -1
  1
```

Η εντολή `solve` μπορεί να επιλύσει και συστήματα εξισώσεων. Για παράδειγμα, το γραμμικό σύστημα τριών μεταβλητών

$$x + y = 1$$

$$x - 11y = 5$$

$$x + z = 3$$

Η λύση του συστήματος γίνεται με τις εντολές

```
>> syms x y z
>> [Sx,Sy,Sz] = solve(x + y == 1, x - 11*y == 5, x+z==3)

Sx =

4/3

Sy =

-1/3

Sz =

5/3
```

Παρατηρούμε ότι, για να εισαχθεί το σύστημα στην εντολή `solve`, πρέπει πρώτα να έχουν οριστεί οι συμβολικές μεταβλητές που θα χρησιμοποιήσουμε. Επίσης, το αποτέλεσμα της `solve` πρέπει να ανατεθεί σε διάνυσμα με κατάλληλο αριθμό μεταβλητών (όσες δηλαδή και οι άγνωστοι του συστήματος), ώστε να αποθηκευτεί η λύση.

Αν και η χρήση συμβολικών μαθηματικών για την επίλυση γραμμικού συστήματος είναι πιο πολύπλοκη από τη χρήση πινάκων (βλ. παράγραφο «Επίλυση γραμμικών συστημάτων», Κεφάλαιο 8), η συγκεκριμένη τεχνική

έχει πολύ περισσότερες δυνατότητες, καθώς μπορεί να λύσει και μη-γραμμικά συστήματα. Για παράδειγμα, οι πραγματικές ρίζες του μη-γραμμικού συστήματος τριών μεταβλητών

$$x^2 + y^2 + z = 1$$

$$x^2 - 2 * y = 2$$

$$x^2 + z^2 == 3$$

υπολογίζονται με τις εντολές

```
>> syms x y z real
>> [Sx,Sy,Sz] = solve(x^2 + y^2 +z == 1, x^2 - 2*y == 2, x^2+z^2==3)

Sx =

 -2^(1/2)
  2^(1/2)


Sy =

 0
 0


Sz =

 -1
 -1
```

Παρατηρούμε ότι υπάρχουν δύο σετ πραγματικών ριζών, οι $x = -\sqrt{2}, y = 0, z = -1$ και $x = \sqrt{2}, y = 0, z = -1$.

10.5 Ασκήσεις

Άσκηση 10.1. *Να υπολογίσετε τις παραγώγους των συναρτήσεων:*

- $f(x) = \alpha\sqrt{x\sqrt{x}}, \alpha \in \mathbb{R}$
- $f(x) = \alpha(x + \log x + 1), \alpha \in \mathbb{R}$
- $f(x) = \frac{x}{tanx}$
- $f(x) = \frac{e^x}{x^3}$
- $f(x) = \log(\log(\log(x)))$

Άσκηση 10.2. *Να υπολογίσετε τα ακόλουθα αόριστα ολοκληρώματα:*

- $\int(\sqrt{x} + x - sinx)dx$
- $\int \frac{1}{x\log x}dx$
- $\int x^3 e^{-x^4}dx$

Άσκηση 10.3. *Να υπολογίσετε τα ακόλουθα ορισμένα ή γενικευμένα ολοκληρώματα:*

- $\int_0^{\pi/2}(\sqrt{x} + x - sinx)dx$
- $\int_1^2 \frac{1}{x\log x}dx$
- $\int_0^{\infty} x^3 e^{-x^4}dx$

Άσκηση 10.4. *Να βρεθούν οι πραγματικές ρίζες των συναρτήσεων:*

- $f(x) = \frac{10}{x^2+1} + 2 + x$
- $f(x) = e^x - 5 + x^2$

Άσκηση 10.5. *Το ανάπτυγμα μίας συνάρτησης σε πολυώνυμο McLaurin το εξετάσαμε στο Παράδειγμα 4.6 του Κεφαλαίου 4. Ωστόσο, δεν αναφέραμε τον τρόπο υπολογισμού των συντελεστών του πολυωνύμου, ο οποίος περιλαμβάνει τον υπολογισμό παραγώγων της συνάρτησης. Συγκεκριμένα, ο τύπος για την κατασκευή του πολυωνύμου McLaurin της συνάρτησης $f(x)$ είναι*

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \cdots + \frac{f^{(n)}(0)}{n!}x^n$$

Δημιουργήστε ένα αρχείο script που θα προσεγγίζει την τιμή μίας συνάρτησης που θα δίνει ο χρήστης υπό τη μορφή σειράς χαρακτήρων στο διάστημα $[-1,1]$, με ένα πολυώνυμο McLaurin. Τον αριθμό των όρων του πολυωνύμου θα τον δίνει ο χρήστης. Συγκρίνετε γραφικά την προσέγγιση του πολυωνύμου McLaurin με τη συνάρτηση $f(x)$, κατασκευάζοντας τις γραφικές του παραστάσεις στο ίδιο διάγραμμα.

Άσκηση 10.6. *Ένας τρούλος (σε οριζόντια θέση) μπορεί να περιγραφτεί από την περιστροφή της συνάρτησης $y = c\sqrt{x}$ γύρω από τον άξονα των x.*
(α) Υπολογίστε το εμβαδόν του τρούλου συναρτήσει του ύψους του και του c.
(β) Ένας ζωγράφος θέλει να σχεδιάσει (κατακόρυφα) πάνω στην επιφάνεια του τρούλου μια φιγούρα μήκους

10 μέτρων. Υπολογίστε τη σχέση που πρέπει να ικανοποιεί το ύψος (συναρτήσει του c), έτσι ώστε να χωρέσει η φιγούρα.

(γ) Κατασκευάστε έναν πίνακα που θα επιστρέφει το ελάχιστο ύψος h του τρούλου για το ερώτημα (β) και το αντίστοιχο εμβαδόν, όταν το c = [0.8 , 0.9 , 1.0 , 1.1 , 1.2]. Αν το h είναι μεγαλύτερο των 9.3 μέτρων να επιστρέφει την τιμή μηδέν, αφού δεν υπάρχει, με βάση τον πολεοδομικό κανονισμό της περιοχής, η δυνατότητα κατασκευής τρούλου με ύψος μεγαλύτερο των 9.3 μέτρων.

10.6 Απαντήσεις στις ασκήσεις αυτοαξιολόγησης

Λύση άσκησης αυτοαξιολόγησης 10.1

Για να γενικευθεί ο κώδικας του Παραδείγματος 10.2, θα πρέπει να ορίσουμε τη συμβολική συνάρτηση $f(x)$ με χρήση της εντολής `syms`. Στη συνέχεια, θα πρέπει ο χρήστης να εισάγει τον τύπο της υπό τη μορφή σειράς χαρακτήρων σε μία μεταβλητή (π.χ. fun), τους οποίους χαρακτήρες στη συνέχεια θα μετατρέψουμε στη συνάρτηση f, χρησιμοποιώντας την εντολή `str2sym`. Τέλος, στα ορίσματα των εντολών `int` και `fplot` θα πρέπει να εισάγουμε τη συνάρτηση $f(x)$ που έχουμε ορίσει.
Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%FOURIER SERIES OF y = exp(x) in [-1,1]
% Initialization
   clear all
   % Symbolic
   syms f(x) n a0 an bn
   assume(n,'integer')
   assume(n,'positive')
   fun = input('Give function str: ');
   f(x) = str2sym(fun);
   % Double
   ND = input('Give number of Fourier terms: ');
   xD = -1:0.01:1;
   FourierD = 0;

% Main code
a0 = int(f(x),-1,1);
an = int(f(x)*cos(n*pi*x),x,-1,1);
bn = int(f(x)*sin(n*pi*x),x,-1,1);

FourierD = double(a0/2);
for i=1:ND
   FourierD = FourierD + double(subs(an,i)) * cos(i*pi*xD) +...
                         double(subs(bn,i)) * sin(i*pi*xD);
end

plot(xD, FourierD)

% Validation
hold
fplot(f(x),[-1,1])
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ελληνόγλωσση

Thomas, G. B., Finney, R. L., Weir, M. D. και Giordano, F. R. (2009). *Απειροστικός Λογισμός Τόμος ΙΙ*. ITE-Πανεπιστημιακές Εκδόσεις Κρήτης.

Ξενόγλωσση

Jeffreys, H., Jeffreys, B. and Swirles, B. (1999). *Methods of Mathematical Physics*. Cambridge Mathematical Library. Cambridge University Press.

ΚΕΦΑΛΑΙΟ 11

ΕΦΑΡΜΟΓΕΣ

Σύνοψη:

Σε αυτό το κεφάλαιο παρουσιάζονται ολοκληρωμένα προγράμματα με στόχο την επίλυση σύνθετων και απαιτητικών προβλημάτων. Τα προγράμματα αυτά συνδυάζουν και συνθέτουν τη φιλοσοφία ανάπτυξης κώδικα και τις εντολές που παρουσιάστηκαν στα προηγούμενα κεφάλαια του παρόντος συγγράμματος, παρουσιάζοντας τη μεθοδολογία επίλυσης πιο σύνθετων προβλημάτων.

Προαπαιτούμενη γνώση: Τα προηγούμενα κεφάλαια του παρόντος συγγράμματος.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του κεφαλαίου αυτού, θα γνωρίζετε:

- να μελετάτε ολοκληρωμένα προβλήματα,
- να συνθέτετε τις αποκτηθείσες γνώσεις στον προγραμματισμό με στόχο την επίλυση σύνθετων προβλημάτων.

Γλωσσάριο επιστημονικών όρων

- γωνιακή ταχύτητα
- εξομάλυνση χρονοσειρών
- κίνηση εκκρεμούς
- κινητός μέσος όρος
- ρυθμός ανάπτυξης πληθυσμού
- σιγμοειδής συνάρτηση

11.1 Εισαγωγή

Στα προηγούμενα κεφάλαια παρουσιάστηκε η φιλοσοφία αλλά και οι εντολές για την επίλυση βασικών προβλημάτων μέσω κατάλληλου κώδικα. Στο παρόν κεφάλαιο οι προαναφερθείσες γνώσεις συνδυάζονται για την ανάπτυξη πιο ολοκληρωμένων προγραμμάτων με στόχο την επίλυση σύνθετων και πιο απαιτητικών προβλημάτων.

Πιο συγκεκριμένα, παρουσιάζεται ο κώδικας:

- μοντελοποίησης της κίνησης ενός εκκρεμούς,
- μοντελοποίησης του ρυθμού ανάπτυξης ενός πληθυσμού σε ένα οικοσύστημα και
- εξομάλυνσης μιας χρονοσειράς.

11.2 Κίνηση εκκρεμούς

Η κίνηση ενός απλού εκκρεμούς (Σχήμα 11.1) μπορεί να περιγραφτεί με τη βοήθεια ενός συστήματος διαφορικών εξισώσεων. Αν θεωρήσουμε ότι η γωνία που σχηματίζει το εκκρεμές με τον κάθετο άξονα είναι θ, η γωνιακή ταχύτητα είναι ω, το μήκος του νήματος είναι L και η επιτάχυνση της βαρύτητας είναι g, τότε οι δύο διαφορικές εξισώσεις που περιγράφουν το πρόβλημα είναι

$$\frac{d\theta}{dt} = \omega \quad \text{και} \quad \frac{d\omega}{dt} = -\frac{g}{L} sin\theta.$$

Αν και είναι δύσκολο να λυθεί αναλυτικά το σύστημα των διαφορικών εξισώσεων, μπορούμε πολύ εύκολα να το μετατρέψουμε σε ένα αλγεβρικό σύστημα, χρησιμοποιώντας τον ορισμό της παραγώγου

$$\frac{d\theta}{dt} = \lim_{\Delta t \to 0} \frac{\theta(t + \Delta t) - \theta(t)}{\Delta t}, \text{ (ομοίως για το } \omega).$$

Για να απλοποιήσουμε τα σύμβολα, μπορούμε να θέσουμε $\theta(t) \equiv \theta_0$ και $\theta(t + \Delta t) \equiv \theta$. Αν θεωρήσουμε ένα αρκετά μικρό πεπερασμένο Δt, τότε η παράγωγος εκφράζεται προσεγγιστικά ως

$$\frac{d\theta}{dt} \approx \frac{\theta - \theta_0}{\Delta t}$$

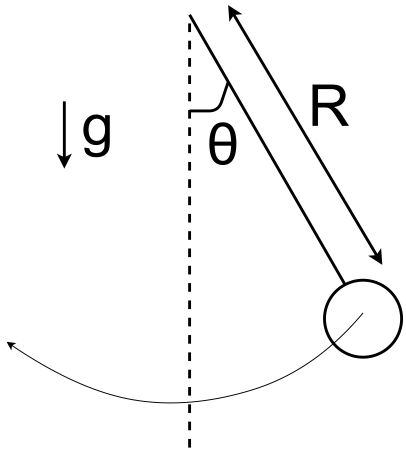
Με παρόμοιο σκεπτικό η γωνιακή ταχύτητα μπορεί να εκφραστεί και αυτή προσεγγιστικά ως

$$\frac{d\omega}{dt} \approx \frac{\omega - \omega_0}{\Delta t}.$$

Μπορούμε, λοιπόν, να αντικαταστήσουμε τις εκφράσεις αυτές στις δύο αρχικές διαφορικές εξισώσεις, ώστε για δεδομένη αρχική γωνία θ_0 και γωνιακή ταχύτητα ω_0 του εκκρεμούς να μπορούμε να εκφράσουμε προσεγγιστικά τη νέα θέση θ και τη νέα γωνιακή ταχύτητα ω μέσα από τις αλγεβρικές εξισώσεις

$$\theta = \theta_0 + \omega \Delta t$$
$$\omega = \omega_0 - \frac{g}{L} \sin\theta \Delta t.$$

Σημειώνουμε ότι η περιγραφή της διαδικασίας επίλυσης που προηγήθηκε, περιορίζεται στις απολύτως απαραίτητες πληροφορίες που καλύπτουν τις ανάγκες του παρόντος συγγράμματος, ενώ για περαιτέρω ανάλυση ο/η αναγνώστης/στρια παραπέμπεται στο σύγγραμμα των Giordano and Nakanishi (2005).



Σχήμα 11.1: Κίνηση απλού εκκρεμούς.

Παράδειγμα 11.1

Δημιουργήστε ένα αρχείο script που θα περιγράφει την κίνηση ενός εκκρεμούς για 100 χρονικά βήματα, σύμφωνα με τις προαναφερθείσες αλγεβρικές σχέσεις. Το πρόγραμμα θα πρέπει να εκτυπώνει σε κάθε χρονική στιγμή τη γωνία και τη γωνιακή ταχύτητα του εκκρεμούς. Επιπρόσθετα, θα πρέπει να αποτυπώνει με χρήση κατάλληλων γραφικών παραστάσεων την κίνηση αυτή. Για την επίτευξη αυτού του σκοπού χρήσιμη είναι η εντολή `drawnow`, η οποία επιτρέπει τη συνεχή ανανέωση μιας γραφικής παράστασης μέσα σε έναν βρόχο, δίνοντας τη δυνατότητα δημιουργίας μιας κινούμενης εικόνας.
Θεωρήστε ότι αρχικά το εκκρεμές το έχουμε εκτρέψει από τη θέση ισορροπίας κατά $\theta_0 = \frac{\pi}{6}$ και ότι η αρχική του γωνιακή ταχύτητα είναι $\omega_0 = 0$. Για τους υπολογισμούς μπορείτε να θεωρήσετε ότι η επιτάχυνση της βαρύτητας είναι $g = 9.81 m/s^2$, το μήκος του νήματος είναι $L = 0.1 m$ και το βήμα του χρόνου είναι $\Delta t = 0.01 s$.

Λύση Παραδείγματος 11.1

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα, θα χρειαστεί να ορίσουμε τις σταθερές:

- για το βήμα του χρόνου dt=0.01,
- για το μήκος του νήματος L=0.1 και
- για την επιτάχυνση της βαρύτητας g=9.81.

Αξίζει να σημειωθεί ότι οι μονάδες μέτρησης των παραπάνω μεγεθών δεν λαμβάνονται υπόψη με κάποιον τρόπο στο συγκεκριμένο πρόγραμμα.
Επίσης, πρέπει να ορίσουμε τις αρχικές τιμές για τη γωνία και τη γωνιακή επιτάχυνση, που θα είναι theta_0=pi/6 και omega_0=0, αντίστοιχα. Τέλος, θα πρέπει να ορίσουμε τις μεταβλητές:

- για τη νέα γωνία (π.χ. theta) και
- τη νέα γωνιακή ταχύτητα (π.χ. omega),

τις οποίες για απλότητα θέτουμε αρχικά ίσες με μηδέν.

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να εκτελεί έναν βρόχο 100 επαναλήψεων, χρησιμοποιώντας μία μεταβλητή «μετρητή», όπου θα υπολογίζονται επαναλαμβανόμενα πρώτα η γωνιακή ταχύτητα omega και, στη συνέχεια, η γωνία theta από τις δύο αλγεβρικές εξισώσεις που χρησιμοποιούνται για την προσεγγιστική περιγραφή της κίνησης του εκκρεμούς. Συγκεκριμένα:

- Υπολογίζουμε το omega χρησιμοποιώντας το omega_0 που είναι γνωστό από την αρχικοποίηση, την τρέχουσα τιμή του theta και τις σταθερές L, g και dt.
- Στη συνέχεια, υπολογίζουμε το theta χρησιμοποιώντας τα theta_0 και την τρέχουσα τιμή του omega, καθώς και το σταθερό dt.
- Εκτυπώνουμε στην οθόνη τα δύο αποτελέσματα theta και omega.
- Αφού έχουμε υπολογίσει τα δύο αποτελέσματα theta και omega για το τρέχον χρονικό βήμα, μπορούμε να προχωρήσουμε στο επόμενο. Αυτό σημαίνει ότι ως αρχικές τιμές theta_0 και omega_0 θα πρέπει να χρησιμοποιήσουμε τα δύο αποτελέσματα που μόλις υπολογίσαμε, καθώς αυτά είναι η βάση για να προχωρήσουμε στο επόμενο βήμα. Αυτό το κάνουμε θέτοντας ότι το theta_0 είναι πλέον το theta που υπολογίσαμε και το omega_0 είναι το omega που έχουμε βρει. Έτσι σε κάθε επανάληψη «βαδίζουμε» μπροστά κατά ένα χρονικό βήμα, υπολογίζοντας με τη σειρά όλες τις τιμές theta και omega που θα προκύψουν κατά την κίνηση του εκκρεμούς

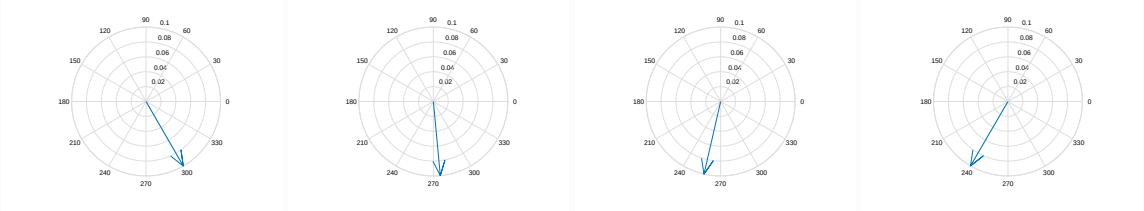
Σημειώνεται ότι λόγω της προσεγγιστικής λύσης είναι πιθανόν να λάβουμε μια γωνιά μεγαλύτερη κατά απόλυτη τιμή από την αρχική γωνιά. Στην περίπτωση αυτή, θα πρέπει να «διορθώσουμε» τον κώδικα θέτοντας τη γωνία ίση με τη μέγιστη (σε απόλυτη τιμή) δυνατή γωνία και μηδενίζοντας την ταχύτητα. Αυτό μπορεί να επιτευχθεί με την εισαγωγή μιας διαδικασίας ελέγχου `if`. Για την πραγματοποίηση του έλεγχου αυτού είναι απαραίτητο να αποθηκεύσουμε και την αρχική τιμή της γωνίας, έτσι ώστε να μπορούμε να κάνουμε σύγκριση της τρέχουσας τιμής του theta με αυτήν. Αυτό μπορεί να επιτευχθεί με τον ορισμό μιας μεταβλητής με το όνομα, παραδείγματος χάριν, theta_start, στην οποία θα ανατεθεί η τιμή $\frac{pi}{6}$.

Αποτελέσματα - οπτικοποίηση: Για να μπορέσουμε να δούμε σε μορφή γραφήματος τη λύση που βγάλαμε, χρησιμοποιούμε τις εντολές `compass` και `drawnow`, οι οποίες δημιουργούν κι εμφανίζουν στην οθόνη ένα πολικό διάγραμμα, με το όποιο μπορούμε να αντιληφθούμε την κίνηση του εκκρεμούς. Δεν θα αναλύσουμε, όμως, περαιτέρω τις εντολές αυτές, καθώς ξεφεύγουν από το αντικείμενο του συγκεκριμένου Παραδείγματος.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %PENDULUM Calculate the movement of a pendulum
2  % Initialization
3   clear all
4   theta_start = pi/6;
5   theta_0 = theta_start;
6   omega_0 = 0;
7   theta = 0;
8   omega = 0;
9   dt = 0.01;
10  g = 9.81;
11  L=0.1;
12
13  % Main code
14  for j=1:100
15    omega = omega_0 - g/L*sin(theta)*dt;
16    theta = theta_0 + omega*dt;
17    if abs(theta)>abs(theta_start)
```

```
18          theta=sign(theta)*theta_start;
19          omega=0;
20      end
21      disp([theta ,omega])
22      theta_0 = theta;
23      omega_0 = omega;
24
25      % Visualization
26      compass(L*exp(i*(theta-pi/2)))
27      drawnow
28  end
```



Σχήμα 11.2: Θέσεις που παίρνει το εκκρεμές σε διάφορες χρονικές στιγμές μίας ταλάντωσης, σύμφωνα με τα στοιχεία του Παραδείγματος 11.1.

Παρατήρηση 11.1

Η λογική της λύσης του προηγούμενου Παραδείγματος είναι ότι αντιμετωπίζουμε την κίνηση του εκκρεμούς σαν μια αλληλουχία διαδοχικών στιγμιοτύπων. Στην ουσία «φωτογραφίζουμε» το εκκρεμές κάθε dt και με αυτή την διαδικασία μετατρέπουμε την κίνηση του σώματος, η οποία είναι συνεχής στο φυσικό περιβάλλον, σε ένα σύνολο διακριτών σημείων (γωνία και γωνιακή ταχύτητα) στον υπολογιστικό χώρο. Πρέπει να σημειώσουμε ότι ακολουθώντας ανάλογη λογική, μπορούμε να μετατρέψουμε ακόμα και τις πιο δύσκολες διαφορικές εξισώσεις σε αλγεβρικά συστήματα και, εκμεταλλευόμενοι τις δυνατότητες των υπολογιστών, να καταφέρουμε να παράξουμε λύσεις που θα μας βοηθήσουν να προσομοιώσουμε πολύπλοκα φυσικά προβλήματα.

Άσκηση αυτοαξιολόγησης 11.1

Τροποποιήστε το αρχείο script του προηγούμενου Παραδείγματος, έτσι ώστε να επιστρέφει τον συνολικό χρόνο που χρειάζεται το εκκρεμές για να ολοκληρώσει n πλήρεις ταλαντώσεις για n που θα ορίζει ο χρήστης.

Παράδειγμα 11.2

Η κίνηση ενός απλού εκκρεμούς, όπως είδαμε, περιγράφεται προσεγγιστικά από δύο αλγεβρικές εξισώσεις. Στις περισσότερες φορές, όμως, στο εκκρεμές ασκούνται δυνάμεις τριβής ή αντίστασης με αποτέλεσμα η ταλάντωση να αποσβένεται, δηλαδή να φθίνει. Σε μια τέτοια περίπτωση, η γωνιακή ταχύτητα, $omega_t$, τη χρονική στιγμή t, περιγράφεται προσεγγιστικά από τη σχέση

$$\omega = \omega_0 - \frac{g}{L}\sin\theta\Delta t - q\omega_0\Delta t$$

όπου ω_0 η ταχύτητα την αμέσως προηγούμενη χρονική στιγμή και q ο λεγόμενος συντελεστής απόσβεσης.

Στην παρούσα άσκηση ζητείται να τροποποιηθεί ο κώδικας του Παραδείγματος 11.1, έτσι ώστε

- να συμπεριλαμβάνεται ο συντελεστής επιβράδυνσης στην περιγραφή της κίνησης του εκκρεμούς και
- να πληροφορείται ο χρήστης για τον χρόνο που χρειάζεται το εκκρεμές μέχρι να φτάσει σε ισορροπία, καθώς και για το πλήθος των ταλαντώσεων που θα έχει ολοκληρώσει μέχρι να συμβεί αυτό.

Θεωρήστε ότι το εκκρεμές έχει φτάσει σε ισορροπία όταν το πλάτος της ταλάντωσης είναι μικρότερο από 0.01 και ότι ο συντελεστής επιβράδυνσης ισούται με 1.

Λύση Παραδείγματος 11.2

Αρχικοποίηση μεταβλητών: Για να λύσουμε το πρόβλημα θα χρειαστεί να ορίσουμε επιπρόσθετα τέσσερις μεταβλητές. Πιο συγκεκριμένα, θα πρέπει να ορίσουμε:

- τη μεταβλητή q, που θα αρχικοποιήσουμε με την τιμή 1,
- τη μεταβλητή $theta\_min$, που θα αρχικοποιήσουμε με την τιμή $-theta\_start$ και θα εκφράζει το τρέχον μέγιστο της γωνίας ταλάντωσης αριστερά από την κατακόρυφο (ή καλύτερα από την αντίθετη μεριά που ξεκινάει η ταλάντωση),
- την μεταβλητή $theta\_max$, που θα αρχικοποιήσουμε με την τιμή $theta\_start$ και θα εκφράζει το τρέχον μέγιστο της γωνίας ταλάντωσης δεξιά από την κατακόρυφο (ή καλύτερα από τη μεριά που ξεκινάει η ταλάντωση),
- τη μεταβλητή $oscillations$, που θα μετράει το πλήθος των ταλαντώσεων και την οποία θα αρχικοποιήσουμε με την τιμή 0.

Κύριος κώδικας: Το πρόγραμμα θα πρέπει να επαναλαμβάνει τη διαδικασία της περιγραφής της κίνησης του εκκρεμούς μέχρι το πλάτος της ταλάντωσης να γίνει μικρότερο από 0.01. Αυτό μπορεί να επιτευχθεί με τη χρήση ενός βρόχου `while`. Η συνθήκη στον βρόχο `while` θα πρέπει να είναι η `theta_max-theta_min>0.01` και οι μεταβλητές $theta\_min$ και $theta\_max$ θα πρέπει να ανανεώνονται σε κάθε ταλάντωση.
Ο προσδιορισμός της χρονικής στιγμής που πρέπει να γίνει αυτή η ανανέωση γίνεται παρατηρώντας ότι η γωνιακή ταχύτητα στη μέγιστη γωνία αλλάζει πρόσημο. Το γεγονός αυτό μας επιτρέπει με έναν έλεγχο `if` να προσδιορίσουμε αυτές τις χρονικές στιγμές και να ανανεώσουμε κατάλληλα τις τιμές των μεταβλητών αυτών. Μάλιστα, σε αυτές τις χρονικές στιγμές ολοκληρώνεται και μισή ταλάντωση, οπότε μπορούμε να ανανεώσουμε και την τιμή της μεταβλητής $oscillations$, αυξάνοντας την τιμή κατά 0.5.
Τέλος, θα πρέπει να τροποποιήσουμε τη σχέση υπολογισμού της γωνιακής ταχύτητας με την εντολή `omega = omega_0 - g/L*sin(theta)*dt - q*omega_0*dt`, έτσι ώστε να συμπεριλάβουμε και τον συντελεστή επιβράδυνσης στη διαδικασία περιγραφής της κίνησης του εκκρεμούς.

Αποτελέσματα - οπτικοποίηση: Ο κώδικας για την οπτικοποίηση της κίνησης του εκκρεμούς δεν χρειάζεται να τροποποιηθεί. Αυτό που χρειάζεται να προστεθεί στο τέλος του προγράμματος είναι οι κατάλληλες εντολές (`disp` και `num2str`) που θα ενημερώνουν τον χρήστη για τον συνολικό χρόνο που χρειάστηκε το εκκρεμές μέχρι να φτάσει σε κατάσταση ισορροπίας, καθώς και για το πλήθος των ταλαντώσεων που έχουν πραγματοποιηθεί.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1  %PENDULUM_DUMP Calculate the movement of a pendulum with damping
```

```
2  % Initialization
3  clear all
4  theta_start = pi/6;
5  theta_0 = theta_start
6  omega_0 = 0;
7  theta = 0;
8  omega = 0;
9  dt = 0.01;
10 g = 9.81;
11 L=0.1;
12 q = 1;
13 theta_min=-theta_start;
14 theta_max=theta_start;
15 oscillations=0;
16
17 j=1;
18 % Main code
19 while theta_max-theta_min>0.01
20   omega = omega_0 - g/L*sin(theta)*dt - q*omega*dt;
21   theta = theta_0 + omega*dt;
22
23   if sign(omega)~=sign(omega_0)
24       if theta>0
25           theta_max=theta;
26       else
27           theta_min=theta;
28       end
29       oscillations=oscillations+0.5;
30   end
31
32   disp([theta,omega])
33   theta_0 = theta;
34   omega_0 = omega;
35
36   % Visualization
37   compass(L*exp(i*(theta-pi/2)))
38   drawnow
39
40   j=j+1;
41 end
42
43 disp(['the pendulum will stop (approximately) after ', num2str((j-1)
     *dt),'sec'])
44 disp(['and will have completed ', num2str(oscillations),'
     oscillations'])
```

Εκτελώντας τον παραπάνω κώδικα μπορούμε να παρακολουθήσουμε την ταλάντωση του εκκρεμούς και να πληροφορηθούμε ότι το εκκρεμές θα σταματήσει περίπου 9.53 δευτερόλεπτα μετά την εκκίνηση της

ταλάντωσής του και έχοντας πραγματοποιήσει 15.5 ταλαντώσεις.

11.3 Ρυθμός ανάπτυξης πληθυσμού

Η σιγμοειδής συνάρτηση έχει πολλές εφαρμογές σε διάφορους επιστημονικούς τομείς, όπως τη φυσική (Finck and Schwartz, 2013; Hamed *et al.*, 2016), τη χημεία (Bakshi *et al.*, 2013; Lu *et al.*, 2020), τη μηχανική (Xujiang *et al.*, 2019; Szabó, 2020), την ιατρική (Park *et al.*, 2020), τη βιολογία (Simpson *et al.*, 2022) και τη στατιστική (Kleinbaum *et al.*, 2002; Sharma *et al.*, 2017) μεταξύ άλλων. Μία από τις εφαρμογές της είναι στη μελέτη οικοσυστημάτων, όπου εκφράζει τη μέση μεταβολή του πληθυσμού ενός είδους συναρτήσει χρόνου. Ο τύπος της, για την εφαρμογή αυτή, είναι

$$P(t) = \frac{K}{1 + \frac{K-P_0}{P_0}e^{-rt}}$$

όπου P_0 είναι ο αρχικός πληθυσμός, K είναι ο μέγιστος πληθυσμός που μπορεί να συντηρηθεί από το οικοσύστημα και r είναι ο ρυθμός ανάπτυξης του πληθυσμού. Στο Σχήμα 11.3 παρουσιάζεται η γραφική παράσταση της συνάρτησης $P(t) = \frac{K}{1+\frac{K-P_0}{P_0}e^{-rt}}$ για $K = 120$, $P_0 = 10$ και $r = 0.6$, η οποία έχει κατασκευαστεί με τις ακόλουθες εντολές.

```
>> t=0:0.01:15;
>> P0=10;
>> K=120;
>> r=0.6;
>> P=K ./ ( 1 + (K-P0)/P0.*exp(-r*t));
>> plot(t,P)
>> xlabel('t')
>> ylabel('P(t)')
```

Από τη γραφική παράσταση είναι φανερό ότι η συνάρτηση αυτή:

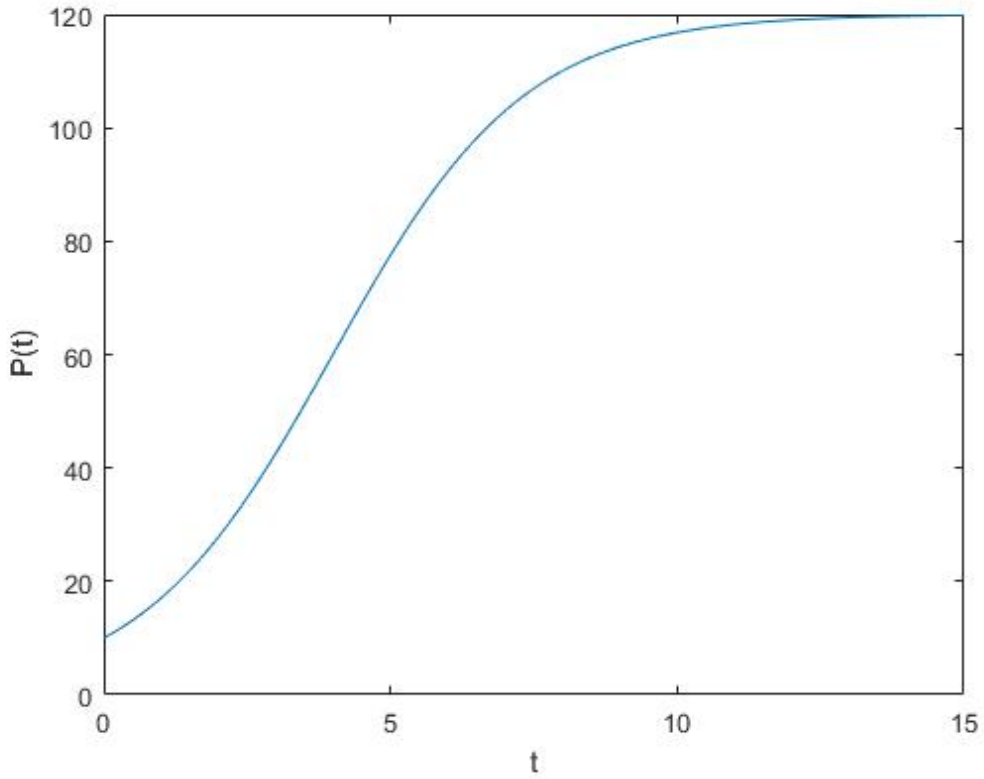
- είναι μια γνησίως αύξουσα συνάρτηση ως προς t,
- έχει σχήμα παρόμοιο με το τελικό σίγμα - εξού και το όνομά της,
- συγκλίνει ασυμπτωτικά στην τιμή K.

Αξίζει να παρατηρήσουμε το γεγονός ότι η συνάρτηση $P(t)$ είναι φραγμένη, εκφράζει το ότι ο πληθυσμός σε μια περιοχή δεν μπορεί να αυξάνεται ανεξέλεγκτα λόγω των περιορισμένων διαθέσιμων πόρων. Επιπρόσθετα, επειδή σε έναν πραγματικό πληθυσμό αναμένονται μικρές μεταβολές σε μικρά χρονικά διαστήματα (θάνατοι/γεννήσεις), η συνάρτηση $P(t)$ είναι προτιμότερο να ερμηνεύεται ως η αναμενόμενη τιμή του μεγέθους του υπό μελέτη πληθυσμού.

Στο ακόλουθο παράδειγμα εφαρμόζεται η παραπάνω σχέση για την περιγραφή της μέσης ετήσιας ανάπτυξης ενός πληθυσμού σε μια νησιωτική περιοχή.

Παράδειγμα 11.3

Θεωρήστε μία νησιωτική περιοχή που αποτελείται από έξι διαφορετικούς τομείς (2×3), ανάλογα με τη μορφολογία του εδάφους, τη βλάστηση κ.λπ. (Σχήμα 11.1). Οι διαφορές αυτές αποτυπώνονται στον ρυθμό ανάπτυξης του πληθυσμού, ο οποίος εκφράζεται για τους έξι τομείς από τον παρακάτω πίνακα



Σχήμα 11.3: Η γραφική παράσταση της συνάρτησης $P(t) = \frac{K}{1+\frac{K-P_0}{P_0}e^{-rt}}$ για $K = 120$, $P_0 = 10$ και $r = 0.6$.

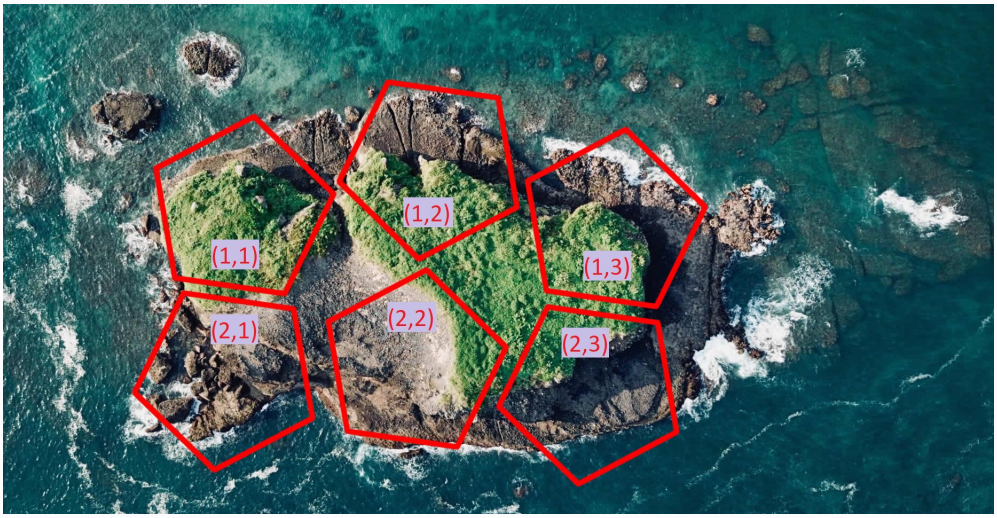
```
r =
    0.5000    0.6000    0.9000
    0.4000    0.2000    0.6000
```

σε άτομα/έτος.

Ας υποθέσουμε ότι σε κάθε τομέα του νησιού εισάγεται ένας αρχικός πληθυσμός $P_0 = 10$ ατόμων ενός είδους και ότι ο μέγιστος αριθμός ατόμων που μπορεί να φτάσει ανά τομέα είναι

```
K =
   100   130   200
    40    10   120
```

Βρείτε τον χρόνο που θα χρειαστεί το νέο είδος για να φτάσει τον μέγιστο πληθυσμό σε όλο το νησί. Στη συνέχεια, εντοπίστε τον τομέα στον οποίο χρειάστηκε τον περισσότερο χρόνο για να φτάσει στον μέγιστο πληθυσμό.



Εικόνα 11.1: Το νησί του Παραδείγματος 11.3 αποτελείται από έξι συνολικά τομείς με διαφορετική μορφολογία εδάφους.

Λύση Παραδείγματος 11.3

Αρχικοποίηση μεταβλητών: Για το παράδειγμα αυτό, θα χρειαστεί να ορίσουμε:

- Τις μεταβλητές P0, K και r, οι οποίες εμφανίζονται στην συνάρτηση. Η P0 έχει την τιμή 10, ενώ οι K και r θα αρχικοποιηθούν σύμφωνα με τους πίνακες που δίνονται στην εκφώνηση.
- Τη μεταβλητή του πληθυσμού P, την οποία θα αρχικοποιήσουμε με την τιμή P0, καθώς αυτός είναι ο αρχικός πληθυσμός σε κάθε περιοχή του νησιού.
- Τη μεταβλητή t για τα έτη, την οποία θα αρχικοποιήσουμε με μηδέν.
- Τις μεταβλητές flag και year, τις οποίες θα αρχικοποιήσουμε με έναν 2×3 μηδενικό πίνακα. Ο πίνακας flag θα χρησιμοποιηθεί για τον έλεγχο της επίτευξης ενός μεγέθους πληθυσμού πολύ κοντά στο μέγιστο σε κάθε τομέα και ο πίνακας year, για να αποθηκευτεί το έτος στο οποίο συμβαίνει αυτό.

Κύριος κώδικας: Για το κύριο πρόγραμμα θα χρησιμοποιήσουμε μία επαναληπτική διαδικασία, η οποία θα αυξάνει τον χρόνο κατά ένα έτος σε κάθε επανάληψη. Δεν γνωρίζουμε το πλήθος των επαναλήψεων που θα χρειαστούν, οπότε θα προτιμήσουμε τη δομή while. Η διαδικασία θα συνεχίζεται μέχρι ο πληθυσμός P να φτάσει τη μέγιστη τιμή K σε όλους τους τομείς του νησιού. Με άλλα λόγια, η συνθήκη του while θα ελέγχει τη διαφορά K - P, η οποία είναι διαφορετική σε κάθε θέση, θα βρίσκει τη μέγιστη τιμή και θα συνεχίζεται όσο η μέγιστη διαφορά είναι μεγαλύτερη από 0.5 (θεωρούμε ότι όταν η διαφορά γίνει μικρότερη του 0.5 σε όλες τις θέσεις, έχουμε φτάσει τον μέγιστο πληθυσμό παντού). Για να εντοπίσουμε τη μέγιστη διαφορά του πίνακα K - P, θα χρησιμοποιήσουμε διπλή εντολή `max`.
Μέσα στην επαναληπτική δομή, θα τοποθετήσουμε τον υπολογισμό του πληθυσμού P. Οι τιμές του είναι διαφορετικές σε κάθε τομέα (i,j) του νησιού, οπότε, για να καλύψουμε όλους τους τομείς, θα χρειαστούμε δύο εσωτερικούς βρόχους: έναν για τις γραμμές του πίνακα και έναν για τις στήλες. Οι βρόχοι αυτοί θα εκτελεστούν με την εντολή for, καθώς γνωρίζουμε ότι οι γραμμές είναι 2 και οι στήλες 3.
Συνολικά, λοιπόν, έχουμε δύο εμφωλευμένους επαναληπτικούς βρόχους for, οι οποίοι βρίσκονται εντός του εξωτερικού βρόχου while. Μέσα στον τριπλό αυτό βρόχο θα τοποθετήσουμε τον υπολογισμό του P, μέσω της συνάρτησης που δίνεται στην εκφώνηση.

Επιπρόσθετα, μέσα στην εσωτερική επαναληπτική δομή θα τοποθετηθεί και μια δομή `if` για την πραγματοποίηση του ελέγχου της επίτευξης του μέγιστου πληθυσμού σε κάθε τομέα. Όπως και πριν, θεωρούμε ότι ο μέγιστος πληθυσμός επιτυγχάνεται μόλις η διαφορά της τρέχουσας τιμής του μεγέθους του πληθυσμού από τη μέγιστη δυνατή τιμή του γίνει μικρότερη από 0.5. Αν η συνθήκη του ελέγχου `if` ικανοποιείται για κάποια περιοχή, τότε:

- αποθηκεύεται στην αντίστοιχη θέση του πίνακα year η τιμή του t και
- μεταβάλλουμε την τιμή στην αντίστοιχη θέση του πίνακα flag έτσι ώστε να μην μην εκτελούνται οι εντολές του ελέγχου `if` στις επόμενες επαναλήψεις για την περιοχή αυτή.

Στο τελευταίο μέρος του κώδικα, θα εκτυπώσουμε το αποτέλεσμα σε συνδυασμό με κατάλληλα μηνύματα μέσα σε δύο εμφωλευμένους επαναληπτικούς βρόχους for.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%SYNARTISI PLITHISMOU
% Initialization
clear all
P0 = 10;
K = [100 , 130 , 200 ; 40 , 10 , 120];
r = [0.5 , 0.6 , 0.9 ; 0.4 , 0.2 , 0.6];
P = P0;
t = 0;
flag=zeros(2,3);
year=zeros(2,3);

% Main code
while max(max(K - P)) > 0.5
  % AYKSISI ETOUS
  t = t + 1;

  % YPOLOGISMOS NEOU PLITHISMOU
  for i = 1:2
    for j = 1:3
      P(i,j) = K(i,j) / ( 1 + (K(i,j)-P0)/P0.*exp(-r(i,j)*t) );

      if K(i,j)-P(i,j)<0.5 && flag(i,j)==0
          year(i,j)=t;
          flag(i,j)=1;
      end

    end
  end

end

for i = 1:2
    for j = 1:3
```

```
34          disp(['Stin perioxi (', num2str(i),',',num2str(j),')
                xreiazontai ',num2str(year(i,j)),' xronia gia megisto
                plithismo'])
35      end
36  end
37  disp(['Gia megisto plithismo se kathe perioxi xreiazontai ',num2str(
        t),' xronia'])
```

Με την ολοκλήρωση του κώδικα εκτυπώνονται τα μηνύματα

```
Stin perioxi (1,1) xreiazontai 15 xronia gia megisto plithismo
Stin perioxi (1,2) xreiazontai 14 xronia gia megisto plithismo
Stin perioxi (1,3) xreiazontai 10 xronia gia megisto plithismo
Stin perioxi (2,1) xreiazontai 14 xronia gia megisto plithismo
Stin perioxi (2,2) xreiazontai 1 xronia gia megisto plithismo
Stin perioxi (2,3) xreiazontai 14 xronia gia megisto plithismo
Gia megisto plithismo se kathe perioxi xreiazontai 15 xronia
```

τα οποία μας πληροφορούν για τον χρόνο που χρειάζεται για να επιτευχθεί το μέγιστο του πληθυσμού σε κάθε τομέα του νησιού, καθώς και τον χρόνο που θα χρειαστεί, για να γίνει αυτό σε όλους τους τομείς του νησιού. Από τα αποτελέσματα αυτά διαπιστώνουμε εύκολα ότι η περιοχή (1,1) είναι αυτή στην οποία θα χρειαστεί να παρέλθει μεγαλύτερος χρόνος (15 έτη), έτσι ώστε να επιτευχθεί ο μέγιστος πληθυσμός.

Άσκηση αυτοαξιολόγησης 11.2

Δοκιμάστε να αντικαταστήσετε τις εντολές των γραμμών 17-21 της λύσης του Παραδείγματος 11.3 με μία γραμμή κώδικα που θα υλοποιεί πράξεις μεταξύ πινάκων.

11.4 Εξομάλυνση χρονοσειρών

Στο Κεφάλαιο 5, στο Παράδειγμα 5.1, χρησιμοποιήθηκαν τα δεδομένα του μηνιαίου αριθμού (σε χιλιάδες) επιβατών αεροπορικών γραμμών στις ΗΠΑ για την περίοδο Ιανουαρίου 1949 έως Δεκέμβρη 1960, για την κατασκευή του γραφήματος χρονοσειράς για τα δεδομένα αυτά. Από το γράφημα αυτό (Σχήμα 5.3) μπορέσαμε εύκολα να διαπιστώσουμε τόσο την αυξητική τάση του αριθμού επιβατών όσο και την ετήσια περιοδικότητα του μεγέθους αυτού. Στην παρούσα ενότητα, θα χρησιμοποιήσουμε πάλι τα δεδομένα αυτά για να εφαρμόσουμε τεχνικές εξομάλυνσης και, πιο συγκεκριμένα, τη μέθοδο του «κινητού μέσου όρου» ή «κινούμενου μέσου» (moving average), που παρουσιάστηκε στο Κεφάλαιο 8 στο Παράδειγμα 8.3.

Ο κινητός μέσος όρος, y_t, των παρατηρήσεων x_t, $t = 1,2,\ldots,n$, μπορεί να οριστεί με βάση τις σχέσεις

$$y_1 = x_1$$
$$y_2 = \frac{x_1 + x_2}{2}$$
$$\ldots\ldots$$
$$y_m = \frac{x_1 + x_2 + \ldots + x_m}{m}$$
$$\ldots\ldots$$
$$y_t = \frac{T_{t-m+1} + T_{t-m+2} + \ldots + T_{t-1} + T_t}{m}, t > m$$

όπου m είναι το πλήθος των παρατηρήσεων στο παράθυρο που χρησιμοποιούνται για να εκτιμήσουν την τιμή του κινητού μέσου όρου. Η παράμετρος m παίζει ιδιαίτερα σημαντικό ρόλο, αφού μικρές τιμές του m έχουν ως συνέπεια τον μικρό βαθμό εξομάλυνσης της χρονοσειράς, ενώ μεγάλες τιμές τον μεγαλύτερο βαθμό εξομάλυνσης.

Στο παρακάτω Παράδειγμα θα χρησιμοποιηθούν, όπως προαναφέρθηκε, τα δεδομένα του Παραδείγματος 5.1, για να εξομαλύνουν τη χρονοσειρά αλλά και για να διαπιστωθεί η επίδραση της τιμής της παραμέτρου m στον βαθμό εξομάλυνσής τους.

Λύση Παραδείγματος 11.4

Αρχικοποίηση μεταβλητών: Για το παράδειγμα αυτό θα χρειαστεί:

- να αναθέσουμε τα δεδομένα σε μία μεταβλητή (π.χ. την y),
- να αναδιατάξουμε τα δεδομένα σε ένα διάνυσμα-γραμμή με χρήση της εντολής `reshape` και
- να αναθέσουμε, με τη χρήση της εντολής `input`, σε μία μεταβλητή (π.χ. την $mall$) τις τιμές του m που θα πληκτρολογήσει ο χρήστης.

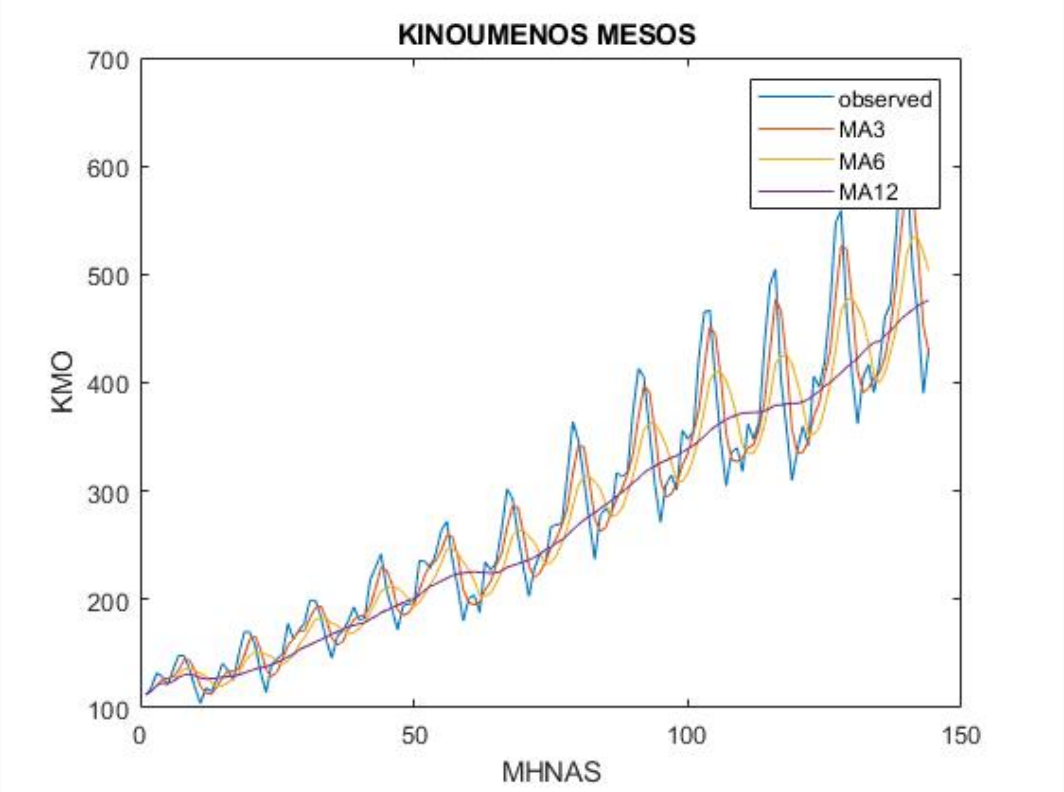
Κύριος κώδικας: Για το κύριο πρόγραμμα θα χρησιμοποιήσουμε ως κορμό τον κώδικα που παρουσιάστηκε στο Παράδειγμα 8.3. Πιο συγκεκριμένα, αυτή η επαναληπτική διαδικασία θα είναι εμφωλευμένη σε μια εξωτερική επαναληπτική διαδικασία, η οποία θα διατρέχει τις τιμές της μεταβλητής $mall$. Σε κάθε επανάληψη θα υπολογίζεται ο κινητός μέσος για την καινούργια τιμή του m και, στη συνέχεια, θα προστίθεται η γραφική παράστασή του στη γραφική παράσταση των τιμών της χρονοσειράς. Η γραφική παράσταση της χρονοσειράς θα πρέπει να έχει δημιουργηθεί πριν από την εκτέλεση των επαναληπτικών διαδικασιών και θα πρέπει να έχει χρησιμοποιηθεί η εντολή `hold ON`. Τέλος, για την καλύτερη επόπτευση του γραφήματος θα πρέπει να προστεθεί, πέρα από τους τίτλους στο γράφημα και στους άξονες, και μια επεξηγηματική λεζάντα για τις ποσότητες που έχουν σχεδιαστεί στο γράφημα.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1 % MOVING AVERAGE smoothing
2 % Initialization
3   clear all
4 %1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960
5 y = [112, 115, 145, 171, 196, 204, 242, 284, 315, 340, 360, 417;   % Jan
6  118, 126, 150, 180, 196, 188, 233, 277, 301, 318, 342, 391;   % Feb
7  132, 141, 178, 193, 236, 235, 267, 317, 356, 362, 406, 419;   % Mar
```

```
 129, 135, 163, 181, 235, 227, 269, 313, 348, 348, 396, 461;   % Apr
 121, 125, 172, 183, 229, 234, 270, 318, 355, 363, 420, 472;   % May
 135, 149, 178, 218, 243, 264, 315, 374, 422, 435, 472, 535;   % Jun
 148, 170, 199, 230, 264, 302, 364, 413, 465, 491, 548, 622;   % Jul
 148, 170, 199, 242, 272, 293, 347, 405, 467, 505, 559, 606;   % Aug
 136, 158, 184, 209, 237, 259, 312, 355, 404, 404, 463, 508;   % Sep
 119, 133, 162, 191, 211, 229, 274, 306, 347, 359, 407, 461;   % Oct
 104, 114, 146, 172, 180, 203, 237, 271, 305, 310, 362, 390;   % Nov
 118, 140, 166, 194, 201, 229, 278, 306, 336, 337, 405, 432]; % Dec

nall=numel(y);
y = reshape(y,1,nall);

mall=input('type m:');

% Main code
plot(y)
hold ON
for m = mall
    KMO = repmat(NaN,1,nall);
    for t = 1:nall
        if t<m
            KMO(t)=mean(y(1:t));
        else
            KMO(t) = sum(y((t-m+1):t))/m;
        end
    end
    plot(KMO)
end
legend('observed','MA3','MA6','MA12')
title(['KINOUMENOS MESOS ']))
xlabel('MHNAS')
ylabel('KMO')
hold OFF
```

Εκτελώντας τον παραπάνω κώδικα λαμβάνουμε τη γραφική παράσταση του Σχήματος 11.4.



Σχήμα 11.4: Το γράφημα χρονοσειράς για τα δεδομένα του μηνιαίου αριθμού (σε χιλιάδες) επιβατών αεροπορικών γραμμών στις ΗΠΑ για την περίοδο Ιανουαρίου 1949 έως Δεκέμβρη 1960, μαζί με τους κινητούς μέσους όρους για $m = 3, 6$ και 12.

Παρατήρηση 11.2

Εξετάζοντας τη γραφική παράσταση του Σχήματος 11.4, μπορούμε να διαπιστώσουμε κάποια ιδιαίτερα χαρακτηριστικά των κινητών μέσων όρων. Πιο συγκεκριμένα, οι τιμές του κινητού μέσου όρου φαίνεται να υστερούν κατά μία παρατήρηση από τις αρχικές (η υστέρηση αυτή είναι ιδιαίτερα φανερή στα μέγιστα κάθε έτους). Επιπρόσθετα, από την εικόνα των καμπυλών των κινητών μέσων όρων είναι φανερό ότι, για μικρές τιμές του m, η μέθοδος ακολουθεί τις παρατηρήσεις χωρίς να προσφέρει πρόσθετες πληροφορίες για τη μακροχρόνια τάση των παρατηρήσεων, αφού δεν έχουμε κάποια ιδιαίτερη εξομάλυνση, ενώ για μεγάλες (π.χ. $m = 12$) δεν αποτυπώνεται καθόλου η περιοδικότητα των παρατηρήσεων. Ένα πρόσθετο πρόβλημα της μεθόδου του κινητού μέσου είναι ότι σταθμίζει όλες τις m προηγούμενες παρατηρήσεις με το ίδιο βάρος ($1/m$), με συνέπεια πιο πρόσφατες παρατηρήσεις να έχουν την ίδια επίδραση με παλαιότερες.

Μια διέξοδο στο τελευταίο πρόβλημα προσφέρει η μέθοδος της εκθετικής ομαλοποίησης, η οποία δίνει μεγαλύτερο βάρος στις πιο πρόσφατες παρατηρήσεις (Brown, 1956). Λύση στο πρόβλημα της εξομάλυνσης της τάσης και της περιοδικότητας προσφέρουν μέθοδοι εξομάλυνσης που έχουν αναπτυχθεί ειδικά για δεδομένα που παρουσιάζουν τάση ή/και εποχικότητα. Τέτοιες μέθοδοι εξομάλυνσης είναι οι μέθοδοι των Winters (1960) και Holt (1957, reprint 2004), οι οποίες αναφέρονται ως

- Διπλής εκθετικής ομαλοποίησης,
- Holt-Winters Seasonal Method .

Η αναλυτική παρουσίαση των μεθόδων αυτών ξεφεύγει από τους σκοπούς αυτού του βιβλίου. Ο/Η

ενδιαφερόμενος/η αναγνώστης/στρια παραπέμπεται, πέρα από τα προαναφερθέντα άρθρα, και στο βιβλίο των Montgomery *et al.* (2011).

11.5 Απαντήσεις στις ασκήσεις αυτοαξιολόγησης

Λύση άσκησης αυτοαξιολόγησης 11.1

Στη συγκεκριμένη άσκηση δεν έχουμε ένα προκαθορισμένο πλήθος επαναλήψεων και για αυτόν τον λόγο θα πρέπει να αντικαταστήσουμε τον βρόχο `for` με έναν βρόχο `while`. Η συνθήκη στο `while` θα πρέπει να εξασφαλίζει ότι η διαδικασία θα συνεχίζεται μέχρι την ολοκλήρωση n πλήρων ταλαντώσεων για n που θα ορίζει ο χρήστης. Επομένως, με την αρχικοποίηση των μεταβλητών θα πρέπει:

- Να εισάγουμε μια εντολή `input` για να αναθέσουμε σε μια μεταβλητή (π.χ. n) το πλήθος των πλήρων ταλαντώσεων που θα ζητήσει ο χρήστης.
- Να δημιουργήσουμε έναν «μετρητή» (π.χ. με το όνομα oscillations) και να τον αρχικοποιήσουμε με την τιμή μηδέν.

Κύριος κώδικας: Ο κύριος κώδικας θα πρέπει να συμπληρωθεί με έναν έλεγχο `if`, ο οποίος θα ελέγχει αν η γωνία theta είναι ίση με το theta_start. Αν είναι, αυτό σημαίνει ότι έχει ολοκληρωθεί μια πλήρης ταλάντωση. Για να εξασφαλίσουμε ότι ο «μετρητής» oscillations δεν θα αυξηθεί κατά το πρώτο βήμα που το εκκρεμές ξεκινάει την ταλάντωση, θα πρέπει να συμπληρώσουμε τη συνθήκη στο `if` με το `j>1`.

Αποτελέσματα - οπτικοποίηση: Στο τέλος του προγράμματος θα πρέπει να προσθέσουμε κατάλληλες εντολές (`disp` και `num2str`) για να ενημερώνεται ο χρήστης για τον συνολικό χρόνο που χρειάστηκε το εκκρεμές να ολοκληρώσει τον αριθμό των πλήρων ταλαντώσεων που όρισε ο χρήστης, εκτελώντας την πράξη $(j-1)*dt$.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
%PENDULUM Calculate the movement of a pendulum
% Initialization
clear all
theta_start = pi/6;
theta_0 = theta_start
omega_0 = 0;
theta = 0;
omega = 0;
dt = 0.01;
g = 9.81;
L=0.1;

n=input('type the total number of completed oscillation required:');
oscillations=0;
j=1;
% Main code
while oscillations<n
  omega = omega_0 - g/L*sin(theta)*dt;
  theta = theta_0 + omega*dt;
  if abs(theta)>abs(theta_start)
      theta=sign(theta)*theta_start;
      omega=0;
  end
```

```
24      disp([theta,omega])
25      theta_0 = theta;
26      omega_0 = omega;
27
28      % Visualization
29      compass(L*exp(i*(theta-pi/2)))
30      drawnow
31
32      if theta==theta_start && j>1
33          oscillations=oscillations+1;
34      end
35
36      j=j+1;
37    end
38
39    disp(['the time required to complete ', num2str(n),' oscillations is
          approximately ', num2str((j-1)*dt),'sec'])
```

Λύση άσκησης αυτοαξιολόγησης 11.2

Με τις πράξεις στοιχείο προς στοιχείο μπορούμε να αποφύγουμε τη διπλή επανάληψη, που χρησιμοποιείται για να υπολογίσουμε ένα-ένα τα στοιχεία του πίνακα P. Πρέπει να δοθεί ιδιαίτερη προσοχή στη χρήση των σωστών τελεστών για τις πράξεις στοιχείο προς στοιχείο, δηλαδή να προηγείται η τελεία των συμβόλων πολλαπλασιασμού και διαίρεσης.

Η παραπάνω λύση υλοποιείται στον κώδικα που ακολουθεί:

```
1   %SYNARTISI PLITHISMOU
2   % Initialization
3   clear all
4   P0 = 10;
5   K = [100 , 130 , 200 ; 40 , 10 , 120];
6   r = [0.5 , 0.6 , 0.9 ; 0.4 , 0.2 , 0.6];
7   P = P0;
8   t = 0;
9
10  % Main code
11  while max(max(K - P)) > 0.5
12    % AYKSISI ETOUS
13    t = t + 1;
14
15    % YPOLOGISMOS NEOU PLITHISMOU
16    P = K ./ ( 1 + (K-P0)./P0.*exp(-r*t) );
17
18  end
19  disp(['Gia megisto plithismo xreiazontai ',num2str(t),' xronia'])
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ξενόγλωσση

Bakshi, S., Zavalov, O., Halámek, J., Privman, V. and Katz, E. (2013). Modularity of biochemical filtering for inducing sigmoid response in both inputs in an enzymatic AND gate. *The Journal of Physical Chemistry B*, *117*(34), pp. 9857–9865.

Brown, R. G. (1956). Exponential Smoothing for Predicting Demand. *Cambridge, Massachusetts: Arthur D. Little Inc.*, pp. 1–15.

Finck, B. Y. and Schwartz, B. J. (2013). Understanding the origin of the S-curve in conjugated polymer/fullerene photovoltaics from drift-diffusion simulations. *Applied Physics Letters*, *103*(5), p. 053306.

Giordano, N. and Nakanishi, H. (2005). *Computational Physics*. (2nd ed.) Pearson.

Hamed, M., Eltaher, M., Sadoun, A. and Almitani, K. (2016). Free vibration of symmetric and sigmoid functionally graded nanobeams. *Applied Physics A*, *122*(9), pp. 1–11.

Holt, C. (1957, reprint 2004). Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, *20*(1). Originally published in Office of Naval Research Memorandum, 52, pp. 5–10.

Kleinbaum, D. G., Dietz, K., Gail, M., Klein, M. and Klein, M. (2002). *Logistic regression*. Springer.

Lu, H., Xu, Z., Gao, K., Zhang, Z., Li, Z. and Xie, J. (2020). A new invertible model of magnetorheological damper based on sigmoid function. *Smart Materials and Structures*, *29*(11), p. 115026.

Montgomery, D., Jennings, C. and Kulahci, M. (2011). *Introduction to Time Series Analysis and Forecasting*. Wiley Series in Probability and Statistics. Wiley.

Park, J. E., Jeong, S. J., Park, S. H., Shin, S. E., Park, S. H., Moon, T.-Y. and Lee, J. W. (2020). A comparison of growth curve models for forensically important flies using sigmoid functions. *Korean Journal of Legal Medicine*, *44*(2), pp. 84–91.

Sharma, S., Sharma, S. and Athaiya, A. (2017). Activation functions in neural networks. *Towards data science*, *6*(12), pp. 310–316.

Simpson, M. J., Browning, A. P., Warne, D. J., Maclaren, O. J. and Baker, R. E. (2022). Parameter identifiability and model selection for sigmoid population growth models. *Journal of theoretical biology*, *535*, p. 110998.

Szabó, F. J. (2020). Analysis of Wear Curves as Sigmoid Functions. In: *Vehicle and Automotive Engineering*. Springer, pp. 273–281.

Winters, P. (1960). Forecasting Sales by Exponentially Weighted Moving Averages. *Management Science*, *6*(3), pp. 324–342.

Xujiang, H., Wenzhe, Z. and Pu, L. (2019). A Path Planning Method for Vehicle Overtaking Maneuver Using Sigmoid Functions. *IFAC-PapersOnLine*, *52*(8). 10th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2019, pp. 422–427.

ΠΑΡΑΡΤΗΜΑΤΑ

ΠΑΡΑΡΤΗΜΑ Α΄

ΔΙΑΝΥΣΜΑΤΑ ΚΑΙ ΠΙΝΑΚΕΣ

Σύνοψη:

Σε αυτό το παράρτημα παρουσιάζονται οι ορισμοί των πινάκων και των διανυσμάτων, καθώς και κάποιες βασικές ιδιότητές τους.

Προαπαιτούμενη γνώση: Δεν απαιτείται κάποια εξειδικευμένη γνώση.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του Παραρτήματος αυτού, θα έχετε γνωρίσει τις έννοιες του πίνακα και των διανυσμάτων και των ιδιοτήτων τους. Το Παράρτημα αυτό δεν αποσκοπεί στην πλήρη κατανόηση των εννοιών και των εργαλείων χειρισμού των πινάκων, αλλά στο να αποτελέσει ένα σύντομο εγχειρίδιο για τις βασικές γνώσεις που πρέπει να έχετε για να μπορείτε να χειριστείτε τους πίνακες και τα διανύσματα στο Matlab.

Γλωσσάριο επιστημονικών όρων

- Ανάστροφος πίνακας
- Αντίστροφος πίνακας
- Διαγώνιος πίνακας
- Διάνυσμα
- Δύναμη πίνακα
- Εσωτερικό γινόμενο διανυσμάτων
- Ίχνος τετραγωνικού πίνακα
- Μοναδιαίος πίνακας
- Ορίζουσα πίνακα
- Πίνακας
- Πίνακας μονάδων
- Πολλαπλασιασμός πινάκων
- Πρόσθεση και αφαίρεση πινάκων
- Συμμετρικός πίνακας
- Τάξη πίνακα
- Ταυτοδύναμος πίνακας

- Τετραγωνικός πίνακας

Α΄.1 Ορισμοί

Ορισμός Α΄.1

Πίνακας ονομάζεται κάθε διάταξη στοιχείων σε n γραμμές και m στήλες

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1m} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \cdots & \alpha_{nm} \end{pmatrix} = (\alpha_{ij})_{n\times m},$$

Ορισμός Α΄.2

Διάνυσμα ονομάζεται ένας $1 \times m$ πίνακας, δηλαδή ένας πίνακας με μία μόνο γραμμή.
Ένας $n \times 1$ πίνακας, δηλαδή ένας πίνακας με μία μόνο στήλη, ονομάζεται *διάνυσμα - στήλη* (ή απλώς διάνυσμα).

Σημειώνεται ότι δύο πίνακες A και B λέγονται ίσοι ($\mathrm{A} = \mathrm{B}$) αν και μόνο αν

- έχουν τις ίδιες διαστάσεις και
- $\alpha_{ij} = \beta_{ij}\ \forall\,(i,j)$.

Α΄.2 Πράξεις πινάκων

Στη συνέχεια, παρουσιάζονται οι πράξεις μεταξύ πινάκων (και διανυσμάτων).

- **Πρόσθεση και αφαίρεση**

$$C = A + B \Leftrightarrow c_{ij} = \alpha_{ij} + \beta_{ij}$$

$$D = A - B \Leftrightarrow d_{ij} = \alpha_{ij} - \beta_{ij}$$

Σχόλιο: Η πρόσθεση και η αφαίρεση πινάκων ορίζεται μόνο αν οι πίνακες έχουν τις ίδιες διαστάσεις.

- **Πολλαπλασιασμός**
  - *Γινόμενο αριθμού με πίνακα*

$$\lambda A = (\lambda \alpha_{ij})_{n\times m}$$

  - *Πολλαπλασιασμός διανυσμάτων*

    Ο πολλαπλασιασμός διανυσμάτων ορίζεται μόνο μεταξύ $1 \times m$ και $m \times 1$ διανυσμάτων. Στην περίπτωση αυτή, το αποτέλεσμα είναι ένας αριθμός, το ονομαζόμενο *εσωτερικό γινόμενο*, ο οποίος ορίζεται ως

$$\begin{pmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_m \end{pmatrix} \cdot \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{pmatrix} = \alpha_1\beta_1 + \ldots + \alpha_m\beta_m$$

– *Πολλαπλασιασμός πίνακα με πίνακα*

Ο πολλαπλασιασμός $\mathrm{A} \cdot B$ του πίνακα A με τον πίνακα B ορίζεται μόνο όταν το πλήθος των στηλών του A ισούται με το πλήθος των γραμμών του B.

Στην περίπτωση αυτή, δηλαδή αν A είναι ένας $n \times m$ πίνακας και B είναι ένας $m \times k$ πίνακας, το γινόμενο $C = A \cdot B$ είναι ένας $n \times k$ πίνακας, του οποίου το (i,j) στοιχείο ορίζεται ως

$$c_{ij} = \begin{pmatrix} \alpha_{i1} & \alpha_{i2} & \cdots & \alpha_{im} \end{pmatrix} \cdot \begin{pmatrix} \beta_{1j} \\ \beta_{2j} \\ \vdots \\ \beta_{mj} \end{pmatrix} = \alpha_{i1}\beta_{1j} + \ldots + \alpha_{im}\beta_{mj}$$

Σχόλιο: Στον πολλαπλασιασμό πινάκων δεν ισχύει η αντιμεταθετική ιδιότητα ($\mathrm{A} \cdot B \neq \mathrm{B} \cdot \mathrm{A}$).

Σχόλιο: Στον πολλαπλασιασμό πινάκων μπορεί να ισχύει $\mathrm{A} \cdot B = 0$ με $\mathrm{A} \neq 0$ και $\mathrm{B} \neq 0$.

Ένα τέτοιο παράδειγμα είναι το ακόλουθο:

$$\begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix} \cdot \begin{pmatrix} 2 & -3 \\ -2 & 3 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

↪ Ένας πίνακας με όλα τα στοιχεία του ίσα με το μηδέν ονομάζεται *μηδενικός πίνακας*.

Α΄.3 Ειδικοί πίνακες

1. **Τετραγωνικός πίνακας (Square matrix)**

   Ένας πίνακας με τον ίδιο αριθμό γραμμών και στηλών ονομάζεται τετραγωνικός.

Παρατήρηση Α΄.1

Οι δυνάμεις πινάκων ορίζονται μόνο για τετραγωνικούς πίνακες ως ακολούθως

$$\mathrm{A}^2 = \mathrm{A} \cdot A \quad \mathrm{A}^3 = \mathrm{A}^2 \cdot A, \cdots$$

Ένας τετραγωνικός πίνακας ονομάζεται ταυτοδύναμος (Idempotent matrix), αν $\mathbf{A}^2 = \mathbf{A}.\mathbf{A} = \mathbf{A}$.

2. **Διαγώνιος πίνακας (Diagonal matrix)**

   Ένας τετραγωνικός πίνακας, του οποίου όλα τα εκτός της κύριας διαγωνίου στοιχεία του ισούνται με το μηδέν, ονομάζεται διαγώνιος. Οι διαγώνιοι πίνακες έχουν επομένως την ακόλουθη μορφή

$$\begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix},$$

   και συμβολίζονται και ως $diag(d_i)$.

3. **Μοναδιαίος πίνακας (Identity matrix)**

   Ένας $n \times n$ διαγώνιος πίνακας, του οποίου όλα τα διαγώνια στοιχεία είναι ίσα με τη μονάδα, ονομάζεται μοναδιαίος ή ταυτοτικός και συμβολίζεται με $\mathbf{I}_n$ ή πιο απλά με $\mathbf{I}$.

   Για έναν οποιοδήποτε $n \times m$ πίνακα $\mathbf{A}$ ισχύει $\mathbf{IA} = \mathbf{A}$ και για έναν οποιοδήποτε $m \times n$ πίνακα $\mathbf{B}$ ισχύει $\mathbf{BI} = \mathbf{B}$.

4. **Πίνακας μονάδων (Matrix of ones)**

   Ένας πίνακας με m γραμμές και n στήλες, του οποίου όλα τα στοιχεία είναι ίσα με τη μονάδα, ονομάζεται πίνακας μονάδων και συμβολίζεται με $\mathbf{J}_{mn}$. Στην περίπτωση τετραγωνικών πινάκων χρησιμοποιείται συχνά ο συμβολισμός $\mathbf{J}_n$ αντί του $\mathbf{J}_{nn}$ ή ακόμα και ο $\mathbf{J}$.

5. **Ανάστροφος πίνακας (Transposition)**

   Ο ανάστροφος ενός $m \times n$ πίνακα **A** είναι ένας $n \times m$ πίνακας, του οποίου το ij στοιχείο είναι το ji στοιχείο του **A**. Ο ανάστροφος του **A** συμβολίζεται με $\mathbf{A}'$. Για τους ανάστροφους ισχύουν οι παρακάτω ιδιότητες:

   - $(\mathbf{A}')' = \mathbf{A}$
   - $(\mathbf{A} + \mathbf{B})' = \mathbf{A}' + \mathbf{B}'$
   - $(\mathbf{AB})' = \mathbf{B}'\mathbf{A}'$

6. **Συμμετρικός πίνακας (Symmetric matrix)**

   Ένας τετραγωνικός πίνακας ονομάζεται συμμετρικός, αν ισούται με τον ανάστροφό του, δηλαδή αν $\mathbf{A}' = \mathbf{A}$. Τότε ισχύει $a_{ij} = a_{ji}$.

Α΄.4 Επιπλέον χαρακτηριστικά πινάκων

1. **Ίχνος τετραγωνικού πίνακα (Trace)**

   Το ίχνος ενός τετραγωνικού πίνακα **A** με στοιχεία α_{ij} ορίζεται ως το άθροισμα των στοιχείων της κύριας διαγωνίου του, δηλαδή ως

   $$tr(\mathbf{A}) = \alpha_{11} + \alpha_{22} + \cdots + \alpha_{nn}\,.$$

   - Αν **A** είναι ένας $n \times n$ πίνακας και **B** ένας $n \times n$ πίνακας, τότε

     $$tr(\mathbf{AB}) = tr(\mathbf{BA})\,.$$

   - Έστω **A** και **B** δύο $k \times k$ πίνακες και σταθερές a και b, τότε

     $$tr(a\mathbf{A} + b\mathbf{B}) = a\ tr(\mathbf{A}) + b\ tr(\mathbf{B})\,.$$

2. **Ορίζουσα πίνακα (Determinant)**

   - Η ορίζουσα $|\mathbf{A}|$ ή $\det(\mathbf{A})$ ενός 2×2 πίνακα ορίζεται ως εξής

     $$\det\begin{pmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{pmatrix} = \alpha_{11}\alpha_{22} - \alpha_{12}\alpha_{21}\,.$$

   - Η ορίζουσα ενός 3×3 πίνακα μπορεί να οριστεί ως ακολούθως

     $$\begin{aligned}\det\begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{pmatrix} &= \alpha_{11}\det\begin{pmatrix} \alpha_{22} & \alpha_{23} \\ \alpha_{32} & \alpha_{33} \end{pmatrix} - \alpha_{12}\det\begin{pmatrix} \alpha_{21} & \alpha_{23} \\ \alpha_{31} & \alpha_{33} \end{pmatrix} + \\ &\quad + \alpha_{13}\det\begin{pmatrix} \alpha_{21} & \alpha_{22} \\ \alpha_{31} & \alpha_{32} \end{pmatrix} \\ &= \alpha_{11}\det(\mathbf{A}_{11}) - \alpha_{12}\det(\mathbf{A}_{12}) + \alpha_{13}\det(\mathbf{A}_{13})\,,\end{aligned}$$

     όπου $\mathbf{A}_{ij}$ είναι ο 2×2 πίνακας που προκύπτει από τον **A**, αν διαγράψουμε την i γραμμή και την j στήλη του.

- Με παρόμοιο τρόπο μπορεί να οριστεί και η ορίζουσα ενός $n \times n$ πίνακα **A**, δηλαδή

$$\det(\mathbf{A}) = \sum_{j=1}^{n} (-1)^{1+j} \alpha_{1j} \det\left(\mathbf{A}_{1j}\right).$$

- Αν $|\mathbf{A}| \neq 0$, τότε ο πίνακας **A** ονομάζεται μη ιδιάζων (nonsingular).
- Αν **A** είναι ένας $n \times n$ πίνακας και a μια σταθερά, τότε $|\mathbf{A}'| = |\mathbf{A}|$ και $|a\mathbf{A}| = a^n|\mathbf{A}|$.

3. **Τάξη ενός πίνακα (Rank)**

   Τάξη ενός πίνακα ονομάζεται το πλήθος των γραμμικών ανεξάρτητων στηλών (ή γραμμών) του.

   - Για έναν $m \times n$ πίνακα **A** ισχύει

$$rank(\mathbf{A}) \leq \min(m,n).$$

   - Για έναν ταυτοδύναμο πίνακα **A** ισχύει $rank(\mathbf{A}) = tr(\mathbf{A})$.

4. **Αντίστροφος πίνακας (Inverse)**

   - Ένας $n \times n$ πίνακας **A** λέγεται αντιστρέψιμος, αν υπάρχει ένας $n \times n$ πίνακας **B**, τέτοιος ώστε

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}.$$

     Ο πίνακας **B** ονομάζεται αντίστροφος πίνακας του **A**, είναι μοναδικός και συμβολίζεται ως $\mathbf{A}^{-1}$.

     Ένας $n \times n$ πίνακας **A** είναι αντιστρέψιμος, αν και μόνο αν $\det(\mathbf{A}) \neq 0$.

   - Αν ο πίνακας **A** είναι αντιστρέψιμος, τότε ισχύει $(\mathbf{A}')^{-1} = (\mathbf{A}^{-1})'$ και $|\mathbf{A}^{-1}| = |\mathbf{A}|^{-1}$. Αν είναι και συμμετρικός, τότε είναι και ο $\mathbf{A}^{-1}$, αφού $(\mathbf{A}^{-1})' = (\mathbf{A}')^{-1} = \mathbf{A}^{-1}$.

ΠΑΡΑΡΤΗΜΑ Β΄

ΠΟΛΥΩΝΥΜΑ ΚΑΙ ΜΙΓΑΔΙΚΟΙ ΑΡΙΘΜΟΙ

Σύνοψη:

Σε αυτό το παράρτημα παρουσιάζονται τα βασικά στοιχεία των ριζών ενός πολυωνύμου και οι βασικές ιδιότητες των μιγαδικών αριθμών.

Προαπαιτούμενη γνώση: Δεν απαιτείται κάποια εξειδικευμένη γνώση.

Προσδοκώμενα μαθησιακά αποτελέσματα: Όταν θα έχετε ολοκληρώσει τη μελέτη του Παραρτήματος αυτού, θα έχετε γνωρίσει τα βασικά στοιχεία των ριζών ενός πολυωνύμου και τις βασικές ιδιότητες των μιγαδικών αριθμών. Το Παράρτημα αυτό δεν αποσκοπεί στην πλήρη κατανόηση των παραπάνω μαθηματικών εννοιών και των εργαλείων χειρισμού των πινάκων αλλά στο να αποτελέσει ένα σύντομο εγχειρίδιο για τις βασικές γνώσεις που πρέπει να έχετε για να μπορείτε να χειριστείτε τα πολυώνυμα και τους μιγαδικούς στο Matlab.

Γλωσσάριο επιστημονικών όρων

- Εκθετική μορφή μιγαδικού αριθμού
- Μιγαδικός αριθμός
- Μιγαδικό επίπεδο
- Πολλαπλότητα ρίζας
- Ρίζα πολυωνύμου
- Τριγωνομετρική μορφή μιγαδικού αριθμού

Β΄.1 Πολυώνυμα

Κάθε πολυώνυμο αποτελεί ένα άθροισμα μη αρνητικών ακέραιων δυνάμεων μιας μεταβλητής πολλαπλασιασμένων με κάποιες σταθερές, οι οποίες στο πλαίσιο του βιβλίου αυτού υποθέτουμε ότι είναι πραγματικοί αριθμοί. Επομένως, ένα πολυώνυμο δίνεται από την ακόλουθη έκφραση

$$\pi(x) = \alpha_n x^n + \alpha_{n-1} x^{n-1} + \ldots + \alpha_1 x + \alpha_0, \alpha_n \neq 0$$

όπου α_i, $i = 0,1,\ldots,n$ πραγματικές σταθερές. Το παραπάνω πολυώνυμο αναφέρεται ως πολυώνυμο n-οστού βαθμού με πραγματικούς συντελεστές.

Παρατήρηση Β΄.1

Οποιαδήποτε σταθερά μπορεί να θεωρηθεί ως ένα σταθερό πολυώνυμο μηδενικού βαθμού. Αν η σταθερά ισούται με το μηδέν αναφέρεται ως το μηδενικό πολυώνυμο.

Ένα από τα σημαντικότερα θεωρήματα στα μαθηματικά είναι το αποκαλούμενο θεμελιώδες θεώρημα της άλγεβρας, το οποίο ασχολείται με τις ρίζες ενός πολυωνύμου, δηλαδή την εύρεση των σημείων στα οποία το πολυώνυμο μηδενίζεται. Μια απλοποιημένη εκδοχή του θεωρήματος παρουσιάζεται στη συνέχεια.

Θεώρημα Β΄.1

Κάθε πολυώνυμο n-οστού βαθμού έχει, συμπεριλαμβανομένων των πολλαπλοτήτων, ακριβώς n ρίζες.

Με τον όρο πολλαπλότητα μιας ρίζας, r_i, αναφερόμαστε στον μεγαλύτερο φυσικό αριθμό k που είναι τέτοιος ώστε το πολυώνυμο $(x - r_i)^k$ να διαιρεί το πολυώνυμο $\pi(x)$.

Πόρισμα Β΄.1

Συνέπεια του θεμελιώδους θεωρήματος της άλγεβρας είναι κάθε πολυώνυμο να μπορεί να γραφτεί στη μορφή

$$\pi(x) = (x - r_1)^{k_1}(x - r_2)^{k_2} \cdots (x - r_m)^{k_m}$$

όπου $r_i, i = 1,2,\ldots,m$ είναι οι m διακριτές ρίζες του πολυωνύμου και $k_i, i = 1,2,\ldots,m$ οι αντίστοιχες πολλαπλότητές τους με $k_i \geq 1$.

Παρατήρηση Β΄.2

Οι ρίζες r_i ενός πολυωνύμου με πραγματικούς συντελεστές μπορεί να είναι πραγματικοί ή μιγαδικοί αριθμοί. Χαρακτηριστικό παράδειγμα πολυωνύμου με μιγαδικές ρίζες είναι το

$$\pi(x) = x^2 + 1$$

που έχει ρίζες τους αριθμούς i και $-i$, όπου i η αποκαλούμενη φανταστική μονάδα, η οποία ορίζεται ως ο αριθμός που ικανοποιεί τη σχέση $i^2 = -1$.

Β΄.2 Μιγαδικοί αριθμοί

Οι μιγαδικοί αριθμοί αποτελούν επέκταση των πραγματικών αριθμών και ορίζονται από τη σχέση

$$z = \alpha + \beta i$$

όπου τα α και β είναι πραγματικοί αριθμοί και το i είναι η αποκαλούμενη φανταστική μονάδα, η οποία ορίζεται ως ο αριθμός που ικανοποιεί τη σχέση $i^2 = -1$. Το α ονομάζεται πραγματικό και το β φανταστικό μέρος του

μιγαδικού αριθμού z.

Στους μιγαδικούς αριθμούς μπορούμε να ορίσουμε τις πράξεις της πρόσθεσης, της αφαίρεσης, του πολλαπλασιασμού και της διαίρεσης. Πιο συγκεκριμένα, έστω $z_1 = \alpha_1 + \beta_1 i$ και $z_2 = \alpha_2 + \beta_2 i$ δύο μιγαδικοί αριθμοί, τότε

- η πρόσθεση των z_1 και z_2 δίνεται από τη σχέση

$$z_1 + z_2 = (\alpha_1 + \alpha_2) + (\beta_1 + \beta_2)i$$

- η αφαίρεση του z_2 από το z_1 δίνεται από τη σχέση

$$z_1 - z_2 = (\alpha_1 - \alpha_2) + (\beta_1 - \beta_2)i$$

- ο πολλαπλασιασμός των z_1 και z_2 δίνεται από τη σχέση

$$z_1 \cdot z_2 = (\alpha_1\alpha_2 - \beta_2\beta_2) + (\alpha_1\beta_2 + \alpha_2\beta_1)i$$

  αφού

$$\begin{aligned} z_1 \cdot z_2 &= (\alpha_1 + \beta_1 i) \cdot (\alpha_2 + \beta_2 i) \\ &= \alpha_1\alpha_2 + \alpha_1\beta_2 i + \alpha_2\beta_1 i + \beta_2\beta_2 i^2 \\ &= (\alpha_1\alpha_2 - \beta_2\beta_2) + (\alpha_1\beta_2 + \alpha_2\beta_1)i \end{aligned}$$

- η διαίρεση του z_1 με τον z_2 δίνεται από τη σχέση

$$\frac{z_1}{z_2} = \frac{(\alpha_1\alpha_2 + \beta_1\beta_2)}{\alpha_2^2 + \beta_2^2} + \frac{(\alpha_2\beta_1 - \alpha_1\beta_2)}{\alpha_2^2 + \beta_2^2} i$$

  η οποία προκύπτει ως ακολούθως:

$$\begin{aligned} \frac{z_1}{z_2} &= \frac{\alpha_1 + \beta_1 i}{\alpha_2 + \beta_2 i} \\ &= \frac{(\alpha_1 + \beta_1 i)(\alpha_2 - \beta_2 i)}{(\alpha_2 + \beta_2 i)(\alpha_2 - \beta_2 i)} \\ &= \frac{(\alpha_1\alpha_2 - \beta_1\beta_2 i^2) + (\alpha_2\beta_1 - \alpha_1\beta_2)i}{\alpha_2^2 + \beta_2^2} \\ &= \frac{(\alpha_1\alpha_2 + \beta_1\beta_2)}{\alpha_2^2 + \beta_2^2} + \frac{(\alpha_2\beta_1 - \alpha_1\beta_2)}{\alpha_2^2 + \beta_2^2} i \end{aligned}$$

Κατά τον υπολογισμό της διαίρεσης πολλαπλασιάσαμε και διαιρέσαμε με τον λεγόμενο συζυγή μιγαδικό του z_2. Συζυγής ενός μιγαδικού αριθμού ονομάζεται εκείνος ο μιγαδικός ο οποίος έχει το ίδιο πραγματικό μέρος, αλλά αντίθετο φανταστικό.

Β΄.2.1 Τριγωνομετρική μορφή

Ένας μιγαδικός αριθμός, εκτός από τη μορφή $z = \alpha + \beta i$, μπορεί να γραφτεί και με τη λεγόμενη τριγωνομετρική μορφή. Η τριγωνομετρική μορφή του μιγαδικού βασίζεται στην αναπαράσταση του μιγαδικού z ως ένα διάνυσμα με αρχή το κέντρο των αξόνων και πέρας το σημείο (α, β) σε ένα επίπεδο, το

οποίο ορίζεται από τον άξονα των πραγματικών αριθμών (οριζόντιος άξονας) και τον άξονα των φανταστικών[1] αριθμών (κάθετος άξονας). Το καρτεσιανό αυτό επίπεδο ονομάζεται μιγαδικό επίπεδο.

Η τριγωνομετρική μορφή χρησιμοποιεί τις πολικές συντεταγμένες του μιγαδικού z στο μιγαδικό επίπεδο, δηλαδή το μέτρο $r = |z|$ και το όρισμά του, ϕ, και δίνεται από τη σχέση

$$z = r(\cos\phi + i\sin\phi).$$

Το μέτρο του μιγαδικού z ορίζεται ως

$$r = |z| = \alpha^2 + \beta^2$$

και εκφράζει την απόσταση του σημείου (α, β) από την αρχή των αξόνων. Το όρισμα του z είναι η γωνία που σχηματίζει το διάνυσμα του μιγαδικού με τον θετικό ημιάξονα των πραγματικών αριθμών. Ως πρωτεύον όρισμα ορίζεται η γωνία εκείνη που βρίσκεται στο διάστημα $(-\pi, \pi)$ και συμβολίζεται με $Arg(z)$.

Β΄.2.2 Εκθετική μορφή

Χρησιμοποιώντας τη σχέση του Euler[2], η τριγωνομετρική μορφή ενός μιγαδικού μπορεί να εκφραστεί ως

$$z = |z|e^{i\phi}$$

η οποία αποτελεί τη λεγόμενη εκθετική μορφή.

Σημειώνεται ότι το αποτέλεσμα του πολλαπλασιασμού και της διαίρεσης δύο μιγαδικών αριθμών μπορούν να προσδιοριστούν πολύ εύκολα με τη βοήθεια της εκθετικής μορφής ως ακολούθως:

$$z_1 \cdot z_2 = r_1 e^{i\phi_1} \cdot r_2 e^{i\phi_2} = r_1 r_2 e^{i(\phi_1+\phi_2)}$$

και

$$\frac{z_1}{z_2} = \frac{r_1 e^{i\phi_1}}{r_2 e^{i\phi_2}} = \frac{r_1}{r_2} e^{i(\phi_1-\phi_2)}$$

αντίστοιχα.

Παρατήρηση Β΄.3

Με βάση τα παραπάνω, ο πολλαπλασιασμός και η διαίρεση δύο μιγαδικών αριθμών μπορεί να ιδωθεί ως μια στροφή και μια επιμήκυνση ή σμίκρυνση ενός διανύσματος.
Χαρακτηριστικό παράδειγμα αποτελεί ο πολλαπλασιασμός ενός μιγαδικού αριθμού με το i. Παραδείγματος χάριν, ο πολλαπλασιασμός του $z = \alpha + \beta i$ με το i έχει ως αποτέλεσμα τον μιγαδικό $w = -\beta + \alpha i$, ο οποίος έχει το ίδιο μέτρο με το z, αλλά σχηματίζει με αυτόν γωνία 90 μοιρών.

[1] Φανταστικός αριθμός ονομάζεται κάθε μιγαδικός με μηδενικό πραγματικό μέρος.
[2] Σύμφωνα με τη σχέση του Euler κάθε πραγματικός αριθμός x ικανοποιεί τη σχέση $e^{ix} = (\cos x + i\sin x)$.

EYPETHPIO

Κατά τις τελευταίες δεκαετίες, η διείσδυση του προγραμματισμού σε διάφορες επιστημονικές και τεχνολογικές δραστηριότητες του σύγχρονου τρόπου ζωής αυξάνεται ραγδαία. Η διαδικασία επεξεργασίας και ανάλυσης μεγάλου όγκου δεδομένων και η ανάγκη για τη μελέτη περίπλοκων φυσικών συστημάτων και σύνθετων τεχνολογικών εφαρμογών απαιτούν την επέκταση των ήδη υπαρχόντων προγραμμάτων ή και την ανάπτυξη νέων. Η δημιουργία ευέλικτων και αξιόπιστων προγραμμάτων με χρήση κώδικα αποτελεί πλέον απαραίτητο εφόδιο για τους μηχανικούς και γενικότερα για όλους τους επιστήμονες.

Στη λογική αυτή, στόχος του παρόντος συγγράμματος είναι η εξοικείωση των αναγνωστών με βασικά στοιχεία ανάπτυξης προγραμματιστικών δεξιοτήτων. Το βιβλίο μπορεί να χρησιμοποιηθεί ως ένα εισαγωγικό σύγγραμμα στη λογική του προγραμματισμού. Το μέσο για την επίτευξη αυτού του σκοπού είναι το MATLAB.

Στο παρόν σύγγραμμα, στα δύο πρώτα κεφάλαια μελετώνται βασικές αρχές του προγραμματισμού, το περιβάλλον του MATLAB και οι βασικές μαθηματικές και λογικές συναρτήσεις. Στη συνέχεια, στα Κεφάλαια 3 και 4 παρουσιάζονται οι εντολές με τις οποίες μπορούν να υλοποιηθούν δομές ελέγχου της ροής ή επαναληπτικές δομές σε ένα πρόγραμμα.

Στα Κεφάλαια 5 και 6 παρουσιάζονται οι βασικοί τρόποι κατασκευής και διαμόρφωσης διδιάστατων και τριδιάστατων γραφικών παραστάσεων, αντίστοιχα. Στο Κεφάλαιο 7 η προσοχή στρέφεται στα πολυώνυμα και στους μιγαδικούς αριθμούς, ενώ στο Κεφάλαιο 8 στους πίνακες και στον χειρισμό τους από το MATLAB.

Στο Κεφάλαιο 9 παρουσιάζεται η διαδικασία κατασκευής συναρτήσεων από τον ίδιο τον χρήστη, ενώ στο Κεφάλαιο 10 παρουσιάζονται οι βασικές εντολές που αφορούν τα συμβολικά μαθηματικά στο MATLAB. Τέλος, στο Κεφάλαιο 11 αναπτύσσονται ολοκληρωμένα προγράμματα με στόχο την επίλυση σύνθετων και απαιτητικών προβλημάτων, χρησιμοποιώντας συνδυαστικά τις γνώσεις όλων των προηγούμενων κεφαλαίων.

Το παρόν σύγγραμμα δημιουργήθηκε στο πλαίσιο του Έργου ΚΑΛΛΙΠΟΣ+	
Χρηματοδότης	Υπουργείο Παιδείας και Θρησκευμάτων, Προγράμματα ΠΔΕ, ΕΠΑ 2020-2025
Φορέας υλοποίησης	ΕΛΚΕ ΕΜΠ
Φορέας λειτουργίας	ΣΕΑΒ/Παράρτημα ΕΜΠ/Μονάδα Εκδόσεων
Διάρκεια 2ης Φάσης	2020-2023
Σκοπός	Η δημιουργία ακαδημαϊκών ψηφιακών συγγραμμάτων ανοικτής πρόσβασης (περισσότερων από 700) • Προπτυχιακών και μεταπτυχιακών εγχειριδίων • Μονογραφιών • Μεταφράσεων ανοικτών textbooks • Βιβλιογραφικών Οδηγών
Επιστημονικά Υπεύθυνος	Νικόλαος Μήτρου, Καθηγητής ΣΗΜΜΥ ΕΜΠ

ISBN: 978-618-5667-86-3 **DOI:** http://dx.doi.org/10.57713/kallipos-145

Το παρόν σύγγραμμα χρηματοδοτήθηκε από το Πρόγραμμα Δημοσίων Επενδύσεων του Υπουργείου Παιδείας