ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Επίλυση Αριθμητικών Προβλημάτων με τη Χρήση του Λογισμικού MATLAB

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΑΝΑΣΤΑΣΙΟΥ ΕΥΣΤΑΘΙΟΣ (ΑΕΜ: 1236)

ΠΑΣΧΟΣ ΜΙΛΤΙΑΔΗΣ (ΑΕΜ:1318)

Επιβλέπων : ΒΕΡΓΑΔΟΣ ΔΗΜΗΤΡΙΟΣ

Αναπληρωτής Καθηγητής

Καστοριά 8/2/2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Επίλυση Αριθμητικών Προβλημάτων με τη Χρήση του Λογισμικού MATLAB

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΑΝΑΣΤΑΣΙΟΥ ΕΥΣΤΑΘΙΟΣ (ΑΕΜ: 1236)

ΠΑΣΧΟΣ ΜΙΛΤΙΑΔΗΣ (ΑΕΜ:1318)

Επιβλέπων : ΒΕΡΓΑΔΟΣ ΔΗΜΗΤΡΙΟΣ

Αναπληρωτής Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 8/2/2024

Βέργαδος Δημήτριος	Νίκος Δημόκας	Ιωάννης Τουλόπουλος
Αναπληρωτής Καθηγητής	Επίκουρος Καθηγητής	Επίκουρος Καθηγητής

Καστοριά 8/2/2024

Ευχαριστίες

Έχοντας φτάσει στο τελικό σημείο του ταξιδιού, θα θέλαμε να ευχαριστήσουμε θερμά τον κ. Βέργαδο Δημήτριο για την εμπιστοσύνη που μας έδειξε όπως και για την υπομονή που έκανε κατά τη διάρκεια υλοποίησης της πτυχιακής εργασίας.

Επίσης θα θέλαμε να ευχαριστήσουμε και όσους είχαν άμεσο και έμμεσο ρόλο για τη δημιουργία της εργασίας και τέλος να απευθύνω τις ευχαριστίες μας στους γονείς μας οι οποίοι στήριξαν τις σπουδές μας με διαφόρους τρόπους.

i

Περίληψη

Η παρούσα πτυχιακή εργασία εκπονήθηκε στα πλαίσια των σπουδών μας στο τμήμα Μηχανικών Πληροφορικής Τ.Ε του ΤΕΙ Δυτικής Μακεδονίας. Το θέμα της είναι η Ανάπτυξη λογισμικού για αριθμητική επίλυση προβλημάτων με χρήση Matlab. Στην σημερινή εποχή όλο και περισσότερο ο ηλεκτρονικός υπολογιστής χρησιμοποιείται ως εργαλείο για την Μαθηματική διερεύνηση και επίλυση προβλημάτων. Το σύγχρονο μαθηματικό λογισμικό παρέχει ολοκληρωμένα περιβάλλοντα που αποτελούν ισχυρά εργαλεία για την επίλυση προβλημάτων αλλά και προγραμματισμού. Ένα από τα σύγχρονα εργαλεία είναι και το Matlab το οποίο σήμερα αποτελεί ένα δυναμικό περιβάλλον για επιστημονικούς και αριθμητικούς υπολογισμούς. Το Matlab μπορεί να εγκατασταθεί σε διάφορα υπολογιστικά συστήματα όπως Windows, Macintosh OS και Unix. Το όνομα του προέρχεται από την συντομογραφία των λέξεων Matrix Laboratory και η λειτουργιά του βασίζεται κυρίως στη χρήση πινάκων τα στοιχειά των οποίων μπορεί να είναι πραγματικοί η μιγαδικοί αριθμοί. Το Matlab χρησιμοποιεί αλγορίθμους υψηλής αξιοπιστίας και ακριβείας επομένως μπορούμε να έχουμε μεγάλη εμπιστοσύνη στα αποτελέσματα που προκύπτουν με την χρήση του. Μπορούν να εκτελεστούν σημαντικές εργασίες εφαρμόζοντας μόνο μια η δύο εντολές η συναρτήσεις. Είναι φανερό ότι πίσω από αυτές τις εντολές βρίσκονται συγκεκριμένες προσεγγιστικές μέθοδοι. Επίσης ο χρήστης μπορεί να δημιουργήσει την δική του συνάρτησης η αρχείο με απλές και εύκολα κατανοητές εντολές για τη επίλυση ιδιαίτερων εφαρμογών η για να εφαρμόσει συγκεκριμένες μεθόδους για την επίλυση κάποιων προβλημάτων. Τέλος το Matlab παρέχει την δυνατότητα της δημιουργίας πολύ καλών γραφικών, και την εισαγωγή των εικόνων μέσα σε κείμενα. Η παρούσα πτυχιακή εργασία αποσκοπεί στην εφαρμογή του Matlab στην αριθμητική ανάλυση, στην αριθμητική ολοκλήρωση, στις διαφορικές εξισώσεις, καθώς και στις διαφορικές εξισώσεις με την μέθοδο Runge-Kutta.

Abstract

The current thesis was completed during our studies on the department of computer science and information technology of the Technological Educational Institute of West Macedonia. The subject of the thesis illustrates software development for mathematical problems solving with the use of Matlab. In our present days, personal computers are used even more as a tool for mathematical software research and problem solving. The modern scientific mathematical software provides complete integrated development environments for problem solving and development. One of the most popular tools is Matlab, which provides a dynamic environment for scientific calculations. Matlab can be installed and used on different operation systems such as Windows, Macintosh OS and Unix. Its name derives from the words Matrix and Labarotary and its main function is based on matrices of both real and imaginary numbers. Matlab uses algorithms of high reliability and precision so we can safely trust the results that it produces. Major tasks can be performed by applying only one or two commands or functions. It is obvious that behind these commands are certain approximate methods. Also, the user has the ability to create his own function or script with easily understand basic functions to solve particular applications or to apply specific methods and algorithms for problem solving. Lastly, Matlab provides the ability to create very good graphs and to insert images on texts. The current thesis aims to implement Matlab in numerical analysis, numerical integration, differential equations, and differential equations using the Runge-Kutta method.

Πίνακας Περιεχομένων

Λίστα Πινάκων

Λίστα Εικόνων

Εισαγωγή

Στην παρούσα εργασία θα ασχοληθούμε με την επίλυση βασικών μαθηματικών προβλημάτων με τη χρήση του προγραμματιστικού λογισμικού MATLAB. Πιο συγκεκριμένα, θα εξετάσουμε το μαθηματικό πεδίο της Αριθμητικής Ανάλυσης, το οποίο ασχολείται με την αλγοριθμική επίλυση προβλημάτων, έτσι ώστε να μπορούμε να τα επιλύσουμε με τη χρήση ηλεκτρονικού υπολογιστή.

Για λόγους ποιοτικού αλλά καθώς και για την κάλυψη όσο μεγαλύτερου μέρους με την αλγοριθμική καθώς και την υλοποίηση αυτών των αλγορίθμων με το λογισμικό MATLAB, δεν θα καλύψουμε στην αρχή το κεφάλαιο των σφαλμάτων στους αριθμητικούς υπολογισμούς λόγω των περιορισμών των αριθμών μηχανής και των αριθμών κινητής υποδιαστολής, ωστόσο θα παραθέτουμε κάποια σημεία αναφοράς κατά την ανάπτυξη των αλγορίθμων μας.

Τα προβλήματα τα οποία θα καλύψουμε είναι τα εξής:

- Αριθμητική Επίλυση μη Γραμμικών Εξισώσεων
- Αριθμητική Επίλυση Γραμμικών Συστημάτων
- Παραγοντοποίηση Πινάκων
- Παρεμβολή Συνάρτησης

1. Αριθμητική Επίλυση μη Γραμμικών Εξισώσεων

Ένα από τα πιο βασικά προβλήματα που αντιμετωπίζουμε στις εφαρμογές που καλούμαστε να επιλύσουμε είναι η εύρεση μιας εξίσωσης της μορφής

$$f(x) = 0$$

όπου η $f(x)$ είναι μια συνάρτηση πραγματικής ή και ακόμα μιγαδικής μεταβλητής x.

Για την συγκεκριμένη εργασία, θα υποθέσουμε πως οι συναρτήσεις που θα μελετήσουμε και θα αναπτύξουμε τρόπους για να βρούμε τις λύσεις τους, θα είναι συναρτήσεις συνεχείς με μια πραγματική μεταβλητή.

Προφανώς, όπως ήδη γνωρίζουμε, για συγκεκριμένους τύπους συναρτήσεων, τις πολυωνυμικές συναρτήσεις, υπάρχουν ήδη τύποι οι οποίοι μας δίνουν τις λύσεις τους, αν και δεν καλύπτουν την πληθώρα των συναρτήσεων που αντιμετωπίζουμε, μιας και έχουν βρεθεί τέτοιοι "κλειστοί τύποι" μόνο για πολυώνυμα από πρώτου μέχρι και τέταρτου βαθμού.

Επομένως, με τις αριθμητικές μεθόδους, θα προσπαθήσουμε να προσεγγίσουμε την ρίζα της συνάρτησης που έχουμε, δηλαδή θα προσπαθήσουμε να βρούμε ένα ξ τέτοιο ώστε $f(\xi) = 0$. Ωστόσο, ένα από τα πιο σημαντικά ζητήματα που έχουμε στην ανάπτυξη αριθμητικών μεθόδων για την εύρεση των ριζών μιας συνάρτησης, είναι αρχικά η εύρεση μιας πιθανής περιοχής $[a, b]$στην οποία θα ανήκει η υποψήφια τιμή της ρίζας.

Ωστόσο το παραπάνω πρόβλημα λύνεται εύκολα με το εξής Θεώρημα:

όπου με $C[a, b]$ συμβολίζουμε το σύνολο των συνεχών πραγματικών συναρτήσεων. Από την στιγμή που εξασφαλίζουμε την αλήθεια του θεωρήματος για την συνάρτηση που μελετάμε, τότε μπορούμε να υλοποιήσουμε αριθμητικούς αλγόριθμους.

1.1 Μέθοδος της διχοτόμησης – Μέθοδος Bolzano

Η μέθοδος της διχοτόμησης, ή αλλιώς η μέθοδος Bolzano, έχει ως κεντρική ιδέα την εξής: από το αρχικό διάστημα που έχουμε, το $[a, b]$, θα προσπαθούμε να βρούμε ένα μικρότερο διάστημα μέσα σε αυτό, που να περιέχει την πιθανή ρίζα ξ. Επαναληπτικά, θα

προσπαθήσουμε να βρούμε ένα πιο μικρό, μέχρις ότου να καταλήξουμε σε ένα σύνολο τόσο μικρό, που θα προσεγγίζει ικανοποιητικά την τιμή της ρίζα. Ας αναπτύξουμε πιο φορμαλιστικά την παραπάνω ιδέα.

Καταρχήν, υποθέτουμε ότι για την συνάρτηση $f(x)$ που μελετάμε, έχουμε ένα διάστημα $[a, b]$ για το οποίο ισχύει ότι $f(x) \in C[a, b]$ και πως ισχύει ότι $f(a) \cdot f(b) < 0$, δηλαδή ότι εντός του διαστήματος, υπάρχει μια ρίζα της συνάρτησης. Το επόμενο βήμα είναι να μελετήσουμε τον μέσο του διαστήματος, έστω $c_0 = (a_0 + b_0)/2$, όπου $a_0 = a, b_0 = b$και στην συνέχεια υπολογίζουμε την τιμή $f(c_0)$. Τότε διακρίνουμε τις εξής περιπτώσεις:

- Αν$f(c_0) = 0$, τότε βρήκαμε επιτυχώς την ρίζα της συνάρτησης μας, και άρα ισχύει ότι $c_0 = \xi$.
- Αν $f(c_0) \neq 0$, τότε υπολογίζουμε τα γινόμενα $f(a_0) \cdot f(c_0)$και $f(c_0) \cdot f(b_0)$.

Με βάση τα παραπάνω γινόμενα προχωράμε ως εξής:

- Αν $f(a_0) \cdot f(c_0) < 0$, τότε η ρίζα της συνάρτησης βρίσκεται μέσα στο διάστημα (a_0, c_0)και για να επαναλάβουμε την ίδια διαδικασία με πριν, θέτουμε $a_1 = a_0$και $b_1 = c_0$, οπότε προκύπτει το νέο διάστημα $[a_1, b_1]$, το οποίο έχει το μισό πλάτος του αρχικού μας διαστήματος.
- Αν $f(c_0) \cdot f(b_0) < 0$, τότε η ρίζα της συνάρτησης βρίσκεται μέσα στο διάστημα (c_0, b_0)και για να επαναλάβουμε την ίδια διαδικασία με πριν, θέτουμε $a_1 = c_0$και $b_1 = b_0$, οπότε προκύπτει το νέο διάστημα $[a_1, b_1]$, το οποίο έχει το μισό πλάτος του αρχικού μας διαστήματος.

Το νέο διάστημα (a_1, b_1) που προκύπτει είτε από την πρώτη μας περίπτωση, είτε από την δεύτερη περίπτωση, εμπεριέχει σίγουρα την ρίζα της συνάρτησης, με αποτέλεσμα η όλη παραπάνω διαδικασία να επαναληφθεί. Επομένως, με τη συνεχή επανάληψη της παραπάνω διαδικασίας, δημιουργείται μια ακολουθία διαστημάτων $[a_0, b_0], [a_1, b_1], \ldots, [a_n, b_n]$ για τα οποία ισχύει ότι

$$a_0 \leqslant a_1 \leqslant a_2 \leqslant \ldots \leqslant b_0$$

$$b_0 \geq b_1 \geq b_2 \geq \ldots \geq a_0$$

με

$$f(a_n) \cdot f(b_n) \leqslant 0, n = 0,1,2,\ldots$$

και

$$b_n - a_n = \frac{1}{2}\left(b_{n-1} - a_{n-1_1}\right)$$

Παρατηρούμε πως οι ακολουθίες a_n, b_n είναι μονότονες και φραγμένες, επομένως συγκλίνουν. Επομένως, από την τελευταία μας σχέση έχουμε ότι

$$b_n - a_n = \frac{b_0 - a_0}{2^n}$$

Επομένως, παίρνοντας το όριο έχουμε

$$\lim_{n\to\infty} b_n - a_n = \lim_{n\to\infty} \frac{b_0 - a_0}{2^n} = 0$$

Άρα, αν θέσουμε

$$\xi = \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n$$

και λάβουμε υπόψιν ότι ισχύει $f(a_n) \cdot f(b_n) \leqslant 0, n = 0,1,2,\ldots$ και πως η συνάρτηση μας είναι συνεχείς, τότε προκύπτει ότι

$$[f(\xi)]^2 \leq 0 \Rightarrow f(\xi) = 0$$

δηλαδή τα όρια των διαστημάτων τελικά συγκλίνουν προς τη ρίζα ξτης συνάρτησης μας.

Όταν σταματήσουμε αυτή την επαναληπτική διαδικασία στο διάστημα $[a, b]$, τότε η καλύτερη προσέγγιση της ρίζα δίνεται από το μέσο αυτού του διαστήματος, δηλαδή το σημείο

$$c_n = \frac{a_n + b_n}{2}$$

και το σφάλμα που προκύπτει σε αυτή την προσέγγιση της ρίζα μας είναι

$$|\xi - c_n| = \left|\xi - \frac{a_n + b_n}{2}\right| \leq \frac{b_n - a_n}{2}$$

Ωστόσο, από την σχέση $b_n - a_n = \frac{b_0 - a_0}{2^n}$, η προηγούμενη μας σχέση γίνεται

$$|\xi - c_n| \leq \frac{b_0 - a_0}{2^{n+1}}$$

Αν μας δίνεται κάποιο $\varepsilon > 0$, το οποίο θα καθορίζει το μέγεθος του σφάλματος του οποίου αποδεχόμαστε για την προσέγγιση της ρίζας της συνάρτησης, τότε προκύπτει ότι θέλουμε να ισχύει ότι

$$\frac{b_0 - a_0}{2^{n+1}} \leq \varepsilon$$

από το οποίο, αν λύσουμε ως προς n, έχουμε ότι

$$n \geq \left\lceil \frac{log(b_0 - a_0) - log2\varepsilon}{log2} \right\rceil$$

Αυτό που προκύπτει ουσιαστικά, όταν ολοκληρωθούν n επαναλήψεις είναι ένα διάστημα $[a, b]$ το οποίο θα περιέχει μέσα του μια ρίζα της εξίσωσης $f(x) = 0$ και η προσέγγιση της ρίζας αυτής από το μέσο του συγκεκριμένου διαστήματος δίνεται από τον τύπο

$$|\xi - c_n| \leq \frac{b_0 - a_0}{2^{n+1}}$$

1.1.1 Ο αλγόριθμος της διχοτόμησης

Στο παρόν εδάφιο, θα συνοψίσουμε την διαδικασία της Διχοτόμησης σε έναν αλγόριθμο, έτσι ώστε στην συνέχεια να μπορέσουμε να τον υλοποιήσουμε στο προγραμματιστικό περιβάλλον της MATLAB.

Έστω πως δίνεται ως είσοδος η συνάρτηση $f(x) \in C[a_0, b_0]$ τέτοια ώστε $f(a_0) \cdot f(b_0) < 0$. Για $n = 0,1,2,\ldots$, μέχρις ότου να επιτευχθεί η απαιτούμενη σύγκλιση με κατάλληλο κριτήριο, να εκτελούνται τα ακόλουθα

1. $c_n = \frac{a_n + b_n}{2}$
2. Αν $f(c_n) = 0$ τότε $\xi = c_n$, διαφορετικά

   αν $f(a_n) \cdot f(c_n) < 0$ τότε

   $$a_{n+1} = a_n, b_{n+1} = c_n$$

   αλλιώς

   $$a_{n+1} = c_n, b_{n+1} = b_n$$

Στον παραπάνω αλγόριθμο, αναφερόμαστε στην αρχή σε ένα “κατάλληλο κριτήριο”. Αυτό το κριτήριο είναι το κριτήριο διακοπής $|f(c)| < \delta$ όπου ως δ θεωρούμε

μια ανεκτικότητας της μορφής $\frac{1}{2}10^{-k}$για δεδομένο k, με αυτό να ναι συνήθως ίσο με k=*6*.

Ουσιαστικά με αυτό το κριτήριο, ελέγχουμε αν το c είναι ρίζα της $f(x)$. Ο λόγος που χρησιμοποιούμε αυτό το κριτήριο και όχι αυτό που αναπτύξαμε στην διαδικασία παραπάνω, το

$$|\xi - c_n| \leq \frac{b_0 - a_0}{2^{n+1}} \leq \varepsilon$$

δηλαδή να έχουμε ένα δεδομένο ε είναι για τον ευνόητο λόγο του ότι δεν γνωρίζουμε την ρίζα ξ που ψάχνουμε.

Αν ωστόσο θέλουμε να έχουμε αναγκαστικά ως κριτήριο τερματισμού κάτι το οποίο πλησιάζει περισσότερο στην ιδέα της διαδικασίας που κατασκευάσαμε παραπάνω, μπορούμε να χρησιμοποιήσουμε το κριτήριο τερματισμού

$$|c_n - c_{n-1}| < \varepsilon$$

και για να το κάνουμε πιο αποτελεσματικό, το κριτήριο

$$\frac{|c_n - c_{n-1}|}{|c_n|} < \varepsilon, |c_n| \neq 0$$

Και τα 2 παραπάνω κριτήρια, με το δ και το ε είναι ικανά να σταματήσουν στο σωστό σημείο που θέλουμε τον αλγόριθμο. Ωστόσο πρέπει να είμαστε προσεκτικοί για άλλη μια φορά στις τιμές που θέλουμε να ικανοποιούν. Οι τιμές αυτές δεν θα πρέπει να είναι κοντά στην Μονάδα Μηχανής, δηλαδή αν k είναι το μέγιστο πλήθος των δεκαδικών ψηφίων που μπορούν να αποθηκευτούν, τότε πρέπει να ισχύει

$$\delta, \varepsilon \geq \frac{1}{2}10^{-k+2}$$

Αν προκύψει και έχουμε τελικά τιμές για τις σταθερές δ και ε πολύ κοντά στην τιμή της Μονάδας Μηχανής, τότε θα προκύψουν λεπτά σφάλματα τα οποία μπορούν να γίνουν καταστροφικά. Το πιο κλασσικό σφάλμα που γίνεται σε τέτοιες περιπτώσεις είναι η Μηχανή μας να εκτελέσει πράξεις των οποίων η συνεχής σύγκλιση να επέκταση των σφαλμάτων να αλλάξουν δραστικά τα τελικά αποτελέσματα.

1.1.2 Ο αλγόριθμος της διχοτόμησης στο MATLAB

Παρακάτω, θα παρουσιάσουμε τις εντολές με τις οποίες μπορούμε να υλοποιήσουμε τον αλγόριθμο της Διχοτόμησης με την χρήση του προγραμματιστικού περιβάλλοντος MATLAB. Έχουμε:

```
function [root, steps]=Dixotomisi(xL, xR, tol, f)
        % INPUTS:
        %       xL = The left point, a, of the interval [a,b]
        %       xR = The right point, b, of the interval [a,b]
        %       tol = The value that we have as critera of termination
        %       f = The function that we are looking for her root
        %  OUTPUTS:
        %       root = The estimation of the root
        %       steps = The number of the steps that where made to achive the root.
        steps = 0;
        xM = xL + (xR - xL) /2 ;
        while ( abs(xL-xR) > tol) && (f(xM) ~= 0)
                xM = xL + (xR-xL) /2;
                if sign( f(xL) ) ~= sign( f(xM) )
                        xR = xM;
                else
                        xL = xM;
                endif
                steps = steps+1;
        endwhile
        root = xM;
```

Εικόνα 1 Ο αλγόριθμος διχοτόμησης στο MATLAB.

Όπως βλέπουμε παραπάνω, ο αλγόριθμος δέχεται σαν είσοδο τα άκρα του διαστήματος με τα ορίσματα *xL* και *xR* να αντιστοιχούν στις τιμές a_n και b_n, το μέγεθος του σφάλματος που δεχόμαστε με την τιμή *tol*, όπου πιο συγκεκριμένα ως κριτήριο θεωρούμε το μήκος του διαστήματος να είναι αρκετά μικρό, δηλαδή ο μέσος να συμπίπτει με τα άκρα του διαστήματος με τα αρχικά ψηφία τους να είναι αρκετά ίσα, τουλάχιστον σε 3 δεκαδικά ψηφία, και το τελευταίο όρισμα είναι το *f*, όπου με αυτό εισάγουμε την συνάρτηση με την οποία θέλουμε να δουλέψουμε και να βρούμε μια ρίζα της. Σαν έξοδο, λαμβάνουμε δύο τιμές, την τιμή *root* η οποία είναι η προσέγγιση της πραγματικής ρίζας και την τιμή *steps* η οποία μας δίνει τον αριθμό των επαναλήψεων που έκανε ο αλγόριθμος, ώστε να μας δώσει τα αποτελέσματα που αναζητάμε.

Ένα παράδειγμα εκτέλεσης του παραπάνω αλγορίθμου είναι το εξής:

```
>> [r,n] = Dixotomisi(1,2, 1e-5, @(x) x.^3 + 2*x.^2-3*x-1)
r = 1.1987
n = 17
```

Εικόνα 2 Παράδειγμα εκτέλεσης του αλγορίθμου διχοτόμησης στο MATLAB.

Αρχικά, διαλέγουμε να αποθηκεύσουμε τις τιμές που επιστρέφει ο αλγόριθμος μας στο διάνυσμα $[r, n]$, όπου στην μεταβλητή r θα αποθηκεύσουμε την τιμή της ρίζα που βρήκαμε και στη μεταβλητή n θα αποθηκεύσουμε τον αριθμό των βημάτων που εκτελέστηκαν.

Στη συνέχεια βλέπουμε πως καλούμε τον αλγόριθμο με είσοδο το διάστημα [1,2], με τιμή για το σφάλμα ε το *1e-5* , το οποίο ισούται με $1 \times 10^{-5} = 0{,}00001$ και ως συνάρτηση της οποίας αναζητούμε την ρίζα της, την συνάρτηση $f(x) = x^3 + 2x^2 - 3x - 1$.

Η έξοδος του αλγόριθμου βλέπουμε πως είναι ίση με r = *1.1987* και n = *17*, πράγμα το οποίο σημαίνει πως μέσα σε 17 επαναλήψεις, ο αλγόριθμος υπολόγισε πως η τιμή της ρίζα της εξίσωσης είναι ίση με 1.1987. Αναφορικά θα πούμε πως η πραγματική ρίζα είναι ίση με ξ =1.1986912435. Επομένως η προσέγγιση r είναι αρκετά ακριβής σε σχέση με την πραγματική ρίζα ξ .

1.1.3 Σχόλια - Παρατηρήσεις

Με βάση την παραπάνω απόδειξη της αποτελεσματικότητας της ιδέας της διχοτόμησης και την απόδειξη της υλοποίησης του σκοπού της, παρατηρούμε πρώτα απ' όλα πως ένα αρνητικό της συγκεκριμένης μεθόδου είναι το μεγάλο σφάλμα στην προσέγγιση της ρίζας σε κάθε βήμα. Πιο συγκεκριμένα, βλέπουμε πως σε κάθε επανάληψη n που κάνουμε, το σφάλμα της προσέγγισης δίνεται από τον τύπο:

$$|\xi - c_n| \leq \frac{b_0 - a_0}{2^{n+1}}$$

Λόγω του παρονομαστή, παρατηρούμε πως για να γίνει αρκετά μικρό αυτό το σφάλμα, ο αριθμός επαναλήψεων πρέπει να είναι αρκετά μεγάλος για να καταφέρουμε να το επιτύχουμε αυτό. Επιπροσθέτως παρατηρούμε πως όσο πιο μεγάλο είναι το αρχικό διάστημα $[a_0, b_0]$, τόσες ανάλογα περισσότερες θα είναι οι επαναλήψεις ώστε να

πετύχουμε ικανοποιητικά μικρό σφάλμα. Με λίγα λόγια η ταχύτητα της όλης διαδικασίας είναι σχετικά αργή, και αυτό είναι και το βασικό της μειονέκτημα.

Άλλο ένα ζήτημα το οποίο προκύπτει από τον αλγόριθμο της διχοτόμησης είναι η αδυναμία εύρεσης παραπάνω από μιας ρίζας. Πιο συγκεκριμένα, λόγω της συνθήκης $f(a_n) \cdot f(b_n) \leqslant 0, n = 0,1,2,\ldots$ που θέλουμε να ικανοποιείται σε κάθε βήμα, ουσιαστικά επικεντρωνόμαστε το να αναζητούμε μια ρίζα και έτσι οι υπόλοιπες ρίζες που μπορεί να υπάρχουν μέσα στο ίδιο διάστημα ουσιαστικά δεν λαμβάνονται υπόψιν.

Τέλος, πάλι από το γεγονός πως η διαδικασία της Διχοτόμησης απαιτεί η συνάρτηση να αλλάζει πρόσημο γύρω από την ρίζα, προκύπτει το πρόβλημα πως δεν γίνεται να εντοπίσουμε ρίζες άρτιας πολλαπλότητας.

1.2 Η μέθοδος της εσφαλμένης θέσης

Η μέθοδος της Εσφαλμένης Θέσης έχει περίπου την ίδια φιλοσοφία με την μέθοδο της Διχοτόμησης, με την βασική τους διαφορά να βρίσκεται στον τρόπο υπολογισμού της προσέγγισης της ρίζας. Η μέθοδος αυτή αναπτύχθηκε διότι όπως προαναφέραμε και διαπιστώσαμε, η μέθοδος της Διχοτόμησης είναι αρκετά αργή.

Επειδή το μόνο που αλλάζει σε αυτή την μέθοδο είναι ο τρόπος προσέγγισης της ρίζας, οι αρχικές υποθέσεις μας συνεχίζουν να είναι οι ίδιες. Πιο συγκεκριμένα, υποθέτουμε πως για την συνάρτηση μας ισχύει πως $f(x) \in C[a,b]$ και πως ισχύει ότι $f(a) \cdot f(b) < 0$, δηλαδή ότι εντός του διαστήματος, υπάρχει μια ρίζα της συνάρτησης, έστω ξ στο διάστημα $[a,b]$ τέτοιο ώστε $f(\xi) = 0$.

Ο τρόπος με τον οποίο θα προσεγγίσουμε την ρίζα ξ τώρα δεν θα είναι με το μέσο του διαστήματος, αλλά με το σημείο x_0 όπου η ευθεία που διέρχεται από τα σημεία $(a, f(a))$ και $(b, f(b))$ τέμνει τον άξονα των x. Θέτουμε $a_0 = a$ και $b_0 = b$, εξίσωση της ευθείας που διέρχεται από τα σημεία $(a_0, f(a_0))$ και $(b_0, f(b_0))$ δίνεται από τον τύπο

$$\frac{y - f(b_0)}{x - b_0} = \frac{f(a_0) - f(b_0)}{a_0 - b_0}$$

Από τη στιγμή που θέλουμε να βρούμε το σημείο τομής της με τον άξονα x, θα θέσουμε στην παραπάνω εξίσωση $y = 0$ και $x = x_0$ και θα λύσουμε ως προς x_0 και έχουμε ότι

$$x_0 = b_0 - f(b_0)\frac{b_0 - a_0}{f(b_0) - f(a_0)}$$

Ωστόσο, από τη συνθήκη $f(a_0) \cdot f(b_0) < 0$, ο παρανομαστής είναι σίγουρα διάφορος του μηδενός, άρα το x_0 θα είναι πάντα ορισμένο και θα λαμβάνει μια τιμή. Προφανώς αν έχουμε ότι $f(x_0) = 0$, τότε ισχύει ότι $\xi = x_0$. Αν δεν ισχύει, τότε ελέγχουμε αν ισχύει η συνθήκη $f(a_0) \cdot f(x_0) < 0$ και αν ισχύει, τότε η ρίζα βρίσκεται μέσα στο διάστημα (a_0, x_0), όποτε ορίζουμε το νέο διάστημα $[a_1, b_1]$ όπου έχουμε $a_1 = \alpha_0$ και $b_1 = x_0$.

Επαναλαμβάνουμε την διαδικασία για το νέο μας διάστημα και υπολογίζουμε την προσέγγιση x_1 από τον τύπο

$$x_1 = b_1 - f(b_1)\frac{b_1 - a_1}{f(b_1) - f(a_1)}$$

και στην συνέχεια κάνουμε πάλι τους απαραίτητους ελέγχους και συγκρίσεις.

Γενικά, η ακολουθία των προσεγγίσεων της ρίζα $\{x_n\}$, *n = 0,1,2, …* δίνεται από τον γενικό τύπο

$$x_n = b_n - f(b_n)\frac{b_n - a_n}{f(b_n) - f(a_n)}, n = 0,1,2, \ldots$$

Προφανώς η παραπάνω ακολουθία συγκλίνει για τους εξής λόγους:

1. Η ακολουθία $\{a_n\}$, *n = 0,1,2, …*, είναι γνησίως αύξουσα και ταυτόχρονα φραγμένη, διότι

   $$a_0 \leqslant a_1 \leqslant a_2 \leqslant \ldots \leqslant b_0$$

   Επομένως συγκλίνει σε κάποιον αριθμό μέσα στο διάστημα $[a_0, b_0]$.

2. Η ακολουθία $\{b_n\}$, *n = 0,1,2, …*, είναι γνησίως φθίνουσα και ταυτόχρονα φραγμένη, διότι

   $$b_0 \geq b_1 \geq b_2 \geq \ldots \geq a_0$$

   Επομένως συγκλίνει σε κάποιον αριθμό μέσα στο διάστημα $[a_0, b_0]$.

3. Για την συνάρτηση *f(x)* ισχύει ότι $f(x) \in C[a, b]$, επομένως οι ακολουθίες $\{f(a_n)\}, \{f(b_n)\}$, *, n = 0,1,2, …* συγκλίνουν και αυτές.

Άρα η ακολουθία $\{x_n\}$, *n = 0,1,2, …* συγκλίνει μέσα στο διάστημα $[a_0, b_0]$.

1.2.1 Ο αλγόριθμος της εσφαλμένης θέσης

Στο παρόν εδάφιο, θα συνοψίσουμε την διαδικασία της Εσφαλμένης Θέσης σε έναν αλγόριθμο, έτσι ώστε στην συνέχεια να μπορέσουμε να τον υλοποιήσουμε στο προγραμματιστικό περιβάλλον της MATLAB.

Έστω πως δίνεται ως είσοδος η συνάρτηση $f(x) \in C[a_0, b_0]$ τέτοια ώστε $f(a_0) \cdot f(b_0) < 0$. Για $n = 0,1,2,\ldots$, μέχρις ότου να επιτευχθεί η απαιτούμενη σύγκλιση με κατάλληλο κριτήριο, να εκτελούνται τα ακόλουθα:

1. Υπολογίζουμε το

$$x_n = b_n - f(b_n)\frac{b_n - a_n}{f(b_n) - f(a_n)}$$

2. Αν $f(x_n) = 0$ τότε $\xi = x_n$, διαφορετικά

αν $f(a_n) \cdot f(x_n) < 0$ τότε

$$a_{n+1} = a_n, b_{n+1} = x_n$$

αλλιώς

$$a_{n+1} = x_n, b_{n+1} = b_n$$

Στον παραπάνω αλγόριθμο, αναφερόμαστε στην αρχή σε ένα "κατάλληλο κριτήριο". Αυτό το κριτήριο είναι το κριτήριο διακοπής $|f(x_n)| < \delta$ όπου ως δ θεωρούμε μια ανεκτικότητας της μορφής $\frac{1}{2}10^{-k}$ για δεδομένο k, με αυτό να ναι συνήθως ίσο με k=6. Ουσιαστικά με αυτό το κριτήριο, ελέγχουμε αν το x_nείναι ρίζα της $f(x)$.

Ο λόγος που χρησιμοποιούμε αυτό το κριτήριο , και όχι αυτό που αναπτύξαμε στην διαδικασία της Διχοτόμησης παραπάνω, το

$$|\xi - x_n| \leq \varepsilon$$

δηλαδή να έχουμε ένα δεδομένο ε, είναι για τον ευνόητο λόγο του ότι δεν γνωρίζουμε την ρίζα ξ που ψάχνουμε. Αν ωστόσο θέλουμε να έχουμε αναγκαστικά ως κριτήριο τερματισμού κάτι το οποίο πλησιάζει περισσότερο στην ιδέα της διαδικασίας της Διχοτόμησης, μπορούμε να χρησιμοποιήσουμε το κριτήριο τερματισμού

$$|x_n - x_{n-1}| < \varepsilon$$

και για να το κάνουμε πιο αποτελεσματικό, το κριτήριο

$$\frac{|x_n - x_{n-1}|}{|x_n|} < \varepsilon, |x_n| \neq 0$$

Και τα 2 παραπάνω κριτήρια, με το δ και το ε είναι ικανά να σταματήσουν στο σωστό σημείο που θέλουμε τον αλγόριθμο.

Ωστόσο πρέπει να είμαστε προσεκτικοί για άλλη μια φορά στις τιμές που θέλουμε να ικανοποιούν. Οι τιμές αυτές δεν θα πρέπει να είναι κοντά στην Μονάδα Μηχανής, δηλαδή ανk είναι το μέγιστο πλήθος των δεκαδικών ψηφίων που μπορούν να αποθηκευτούν, τότε πρέπει να ισχύει

$$\delta, \varepsilon \geq \frac{1}{2} 10^{-k+2}$$

1.2.2 Ο αλγόριθμος της εσφαλμένης θέσης στο MATLAB

Παρακάτω, θα παρουσιάσουμε τις εντολές με τις οποίες μπορούμε να υλοποιήσουμε τον αλγόριθμο της Εσφαλμένης Θέσης με την χρήση του προγραμματιστικού περιβάλλοντος MATLAB.

Έχουμε:

```
function [root, steps]=ET(xL, xR, tol, f)
% INPUTS:
%       xL = The left point, a, of the interval [a,b]
%       xR = The right point, b, of the interval [a,b]
%       tol = The value that we have as critera of termination
%       f = The function that we are looking for her root
% OUTPUTS:
%       root = The estimation of the root
%       steps = The number of the steps that where made to achive the root.
steps = 0;
xN = xR - f(xR)* (xR - xL) /(f(xR) - f(xL) ) ;
while ( abs(xL-xR) > tol) && (f(xM) ~= 0)
  xN = xR - f(xR)* (xR - xL) /(f(xR) - f(xL) ) ;
  if sign( f(xL) ) ~= sign( f(xM) )
    xR = xN;
  else
    xL = xN;
  end
  steps = steps+1;
end
root = xM;
```

Εικόνα 3 Ο αλγόριθμος της εσφαλμένης θέσης στο MATLAB.

Όπως βλέπουμε παραπάνω, ο αλγόριθμος δέχεται σαν είσοδο τα άκρα του διαστήματος με τα ορίσματα xL και xR να αντιστοιχούν στις τιμές a_n και b_n, το

μέγεθος του σφάλματος που δεχόμαστε με την τιμή *tol*, όπου πιο συγκεκριμένα ως κριτήριο θεωρούμε το μήκος του διαστήματος να είναι αρκετά μικρό, δηλαδή ο μέσος να συμπίπτει με τα άκρα του διαστήματος με τα αρχικά ψηφία τους να είναι αρκετά ίσα, τουλάχιστον σε 3 δεκαδικά ψηφία, και το τελευταίο όρισμα είναι το*f*, όπου με αυτό εισάγουμε την συνάρτηση με την οποία θέλουμε να δουλέψουμε και να βρούμε μια ρίζα της. Σαν έξοδο, λαμβάνουμε δύο τιμές, την τιμή *root* η οποία είναι η προσέγγιση της πραγματικής ρίζας και την τιμή *steps* η οποία μας δίνει τον αριθμό των επαναλήψεων που έκανε ο αλγόριθμος, ώστε να μας δώσει τα αποτελέσματα που αναζητάμε.

Ένα παράδειγμα εκτέλεσης του παραπάνω αλγορίθμου είναι το εξής:

```
>> [r,n] = ET(1,2, 1e-5, @(x) x.^3 + 2*x.^2-3*x-1)

r = 1.1987

n = 11
```

Εικόνα 4 Παράδειγμα εκτέλεσης του αλγορίθμου εσφαλμένης θέσης στο MATLAB.

Αρχικά, διαλέγουμε να αποθηκεύσουμε τις τιμές που επιστρέφει ο αλγόριθμος μας στο διάνυσμα $[r, n]$, όπου στην μεταβλητή r θα αποθηκεύσουμε την τιμή της ρίζα που βρήκαμε και στην μεταβλητή n θα αποθηκεύσουμε τον αριθμό των βημάτων που εκτελέστηκαν.

Στη συνέχεια βλέπουμε πως καλούμε τον αλγόριθμο με είσοδο το διάστημα $[1,2]$, με τιμή για το σφάλμα ε το *1e-5* , το οποίο ισούται με $1 \times 10^{-5} = 0{,}00001$ και ως συνάρτηση της οποίας αναζητούμε την ρίζα της, την συνάρτηση $f(x) = x^3 + 2x^2 - 3x - 1$.

Η έξοδος του αλγόριθμου βλέπουμε πως είναι ίση με $r = 1.1987$ και $n = 17$, πράγμα το οποίο σημαίνει πως μέσα σε 11 επαναλήψεις, ο αλγόριθμος υπολόγισε πως η τιμή της ρίζα της εξίσωσης είναι ίση με 1.1987. Αναφορικά θα πούμε πως η πραγματική ρίζα είναι ίση με ξ =1.1986912435. Επομένως η προσέγγιση r είναι αρκετά ακριβής σε σχέση με την πραγματική ρίζα ξ .

1.2.3 Σχόλια - Παρατηρήσεις

Το πιο βασικό σχόλιο που μπορούμε να κάνουμε , είναι πως με βάση τα παραδείγματα, είναι φανερό πως ο αλγόριθμος της Εσφαλμένης Θέσης συγκλίνει ταχύτερα στην ζητούμενη ρίζα της συνάρτησης.

Καθώς υλοποιήσαμε το παράδειγμα στον αλγόριθμο της Διχοτόμησης, είδαμε πως χρειάστηκαν 17 επαναλήψεις, ενώ όταν υλοποιήσαμε το παράδειγμα στον αλγόριθμο της

Εσφαλμένης Θέσης, είδαμε πως χρειάστηκαν μόνο 11 επαναλήψεις. Είχαμε μια μείωση των επαναλήψεων δηλαδή περίπου στο 36% από τις αρχικές, κάτι που σε πιο περίπλοκες συναρτήσεις θα επιφέρει καλύτερα αποτελέσματα.

1.3 Η μέθοδος Newton – Raphson

Η μέθοδος Newton – Raphson είναι μια από τις καλύτερες και αποτελεσματικότερες μεθόδους για την εύρεση μιας ρίζα της εξίσωσης $f(x) = 0$. Η βασική συνθήκη που πρέπει να ισχύει αρχικά, είναι ότι η $f(x)$ θέλουμε να είναι μια διπλά συνεχής και παραγωγίσιμη συνάρτηση στο διάστημα $[a, b]$ που αναζητούμε την ρίζα, δηλαδή θέλουμε $f \in C^2[a, b]$. Θεωρούμε τώρα το $x_n \in [a, b]$ μια προσέγγιση της ρίζα ξ τέτοια ώστε $f'(x_n) \neq 0$ και η ποσότητα $|x_n - \xi|$ να είναι αρκετά «μικρή».

Θεωρούμε το πολυώνυμο δευτέρου βαθμό της σειράς Taylor για την συνάρτηση $f(x)$ γύρω από το x_n , δηλαδή :

$$f(x) = f(x_n) + (x - x_n)f'(x_n) + \frac{(x - x_n)^2}{2!}f''(\eta(x))$$

όπου $\eta(x)$ βρίσκεται μεταξύ των x και x_n . Για $x = \xi$ η παραπάνω σχέση μας δίνει

$$f(\xi) = 0 = f(x_n) + (\xi - x_n)f'(x_n) + \frac{(\xi - x_n)^2}{2!}f''(\eta(\xi))$$

Ωστόσο, λόγω του ότι θέλουμε το ποσότητα $|x_n - \xi|$ να είναι αρκετά μικρό, ο όρος $\frac{(\xi - x_n)^2}{2!}f''(\eta(\xi))$ θα είναι πάρα πολύ κοντά στο μηδέν, επομένως μπορούμε να πούμε ότι:

$$0 \cong f(x_n) + (\xi - x_n)f'(x_n)$$

Κάνοντας τις πράξεις, προκύπτει ότι:

$$\xi = x_n - \frac{f(x_n)}{f'(x_n)}, f'(x_n) \neq 0$$

Με τον παραπάνω τρόπο, βλέπουμε πως κατασκευάζεται η ακολουθία x_n η οποία δίνεται από τον τύπο

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, n = 0,1,2, \dots, f'(x_n) \neq 0$$

η οποία αποτελεί την καρδιά της μεθόδου .

1.3.1 Η σύγκλιση της μεθόδου Newton – Raphson

Προφανώς το ότι βρήκαμε έναν τύπο ο οποίος μπορεί να εκφράζει την πραγματική ρίζα της εξίσωσης, δεν συνεπάγεται απόλυτα πως η ακολουθία η οποία βασίζεται σε αυτόν τελικά όντως συγκλίνει σε αυτή. Για τον λόγο αυτό, υπάρχει το παρακάτω Θεώρημα το οποίο μας εξασφαλίζει την αποτελεσματικότητα της ακολουθίας που έχουμε κατασκευάσει:

Θεώρημα:

Έστω $f(x) \in C^2[\xi - \varepsilon, \xi + \varepsilon], \varepsilon > 0$. Αν $f(\xi) = 0$ και $f'(\xi) \neq 0$, τότε υπάρχει $\varepsilon_0 \leq \varepsilon$ τέτοιο ώστε η μέθοδος Newton – Raphson να συγκλίνει στο ξ για κάθε αρχική προσέγγιση του $x_0 \in I_{\varepsilon_0} = [\xi - \varepsilon_0, \xi + \varepsilon_0]$. Επιπλέον, η σύγκλιση είναι τουλάχιστον τετραγωνική.

Αυτό που μας λέει το παραπάνω Θεώρημα, ουσιαστικά είναι πως κοντά στην πραγματική ρίζα της συνάρτησης $\boldsymbol{f(x)}$, η ακολουθία που κατασκευάσαμε , $\boldsymbol{x_n}$, συγκλίνει τελικά στην ρίζα αυτή, με το σφάλμα, ή αλλιώς την αποδεκτή τιμή που θα πάρει το ε σε κάθε τιμή της ακολουθίας $\boldsymbol{x_n}$ ικανοποιεί την παρακάτω σχέση:

$$\lim_{n\to\infty} \frac{|\varepsilon_{n+1}|}{|\varepsilon_n|^2} = \frac{1}{2}\left|\frac{f''(\xi)}{f'(\xi)}\right|$$

όπου $\boldsymbol{\varepsilon_{n+1} = x_{n+1} - \xi}$ και $\boldsymbol{\varepsilon_n = x_n - \xi}$.

Με βάση το παραπάνω, παρατηρούμε πως για να πετύχουμε αποτελεσματικά αποτελέσματα με την μέθοδο Newton – Raphson, θα πρέπει να γίνει μια πάρα πολύ καλή επιλογή του αρχικού στοιχείου της ακολουθίας μας, του $\boldsymbol{x_0}$. Για αυτό τον λόγο πρέπει να βρούμε έναν τρόπο ώστε να μπορούμε να κάνουμε την πιο σωστή επιλογή.

1.3.2 Κατάλληλη επιλογή του x

Θα προσπαθήσουμε σε αυτό το σημείο να βρούμε ένα αποτελεσματικό κριτήριο με το οποίο μπορούμε να διαλέξουμε κατάλληλο $\boldsymbol{x_0}$. Από την τελευταία σχέση που έχουμε, η οποία προκύπτει από την τετραγωνική σύγκλιση της ακολουθίας, έχουμε ότι

$$\lim_{n\to\infty} \frac{|\varepsilon_{n+1}|}{|\varepsilon_n|^2} = \frac{1}{2}\left|\frac{f''(\xi)}{f'(\xi)}\right| = M$$

όπου θέσαμε

$$\boldsymbol{M = \frac{1}{2}\left|\frac{f''(\xi)}{f'(\xi)}\right|}$$

Αν κάνουμε πράξεις, προκύπτει ότι:

$$\boldsymbol{|\varepsilon_{n+1}| \leq M\varepsilon_n{}^2}$$

Αυτό που θέλουμε ουσιαστικά, είναι με κάθε βήμα να φτάνουμε όλο και πιο κοντά στην πραγματική ρίζα, δηλαδή θέλουμε να ισχύει ότι:

$$\boldsymbol{\lim_{n\to\infty}|\varepsilon_{n+1}| = 0}$$

Επομένως για να γίνει αυτό, θα πρέπει να ισχύει ότι:

$$\boldsymbol{|M\varepsilon_0| < 1}$$

ή αλλιώς να ισχύει:

$$\boldsymbol{|\xi - x_0| < \frac{1}{M} = 2\left|\frac{f'(\xi)}{f''(\xi)}\right|}$$

Από την τελευταία παρατηρούμε πως η επιλογή του $\boldsymbol{x_0}$ επηρεάζεται κατά πολύ από την τιμή που θα λάβει θεωρητικά το M, μιας και δεν γίνεται στην πράξη να το γνωρίζουμε. Για τον λόγο αυτό, συνήθως χρησιμοποιούμε την μέθοδο της διχοτόμησης έτσι ώστε να βρούμε μια σχετική προσέγγιση της ρίζας που αναζητάμε, και στην συνέχεια με την ταχύτατη μέθοδο Newton – Raphson να βρούμε την τελική προσέγγιση που αναζητάμε.

1.3.3 Ο αλγόριθμος της μεθόδου Newton – Raphson

Στο παρόν εδάφιο, θα συνοψίσουμε την διαδικασία της Εσφαλμένης Θέσης σε έναν αλγόριθμο, έτσι ώστε στη συνέχεια να μπορέσουμε να τον υλοποιήσουμε στο προγραμματιστικό περιβάλλον της MATLAB.

Έστω πως δίνεται ως είσοδος η συνάρτηση $f(x) \in C^2[a,b]$ και μια αρχική προσέγγιση $x_0 \in [a,b]$ Για $n = 0,1,2,\ldots$, μέχρις ότου να επιτευχθεί η απαιτούμενη σύγκλιση με κατάλληλο κριτήριο, να εκτελούνται τα ακόλουθα:

1. Υπολογισμός του x_n με την βοήθεια του τύπου:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, n = 0,1,2, \dots, f'(x_n) \neq 0$$

1.3.4 Ο αλγόριθμος της μεθόδου Newton – Raphson στο MATLAB

Παρακάτω, θα παρουσιάσουμε τις εντολές με τις οποίες μπορούμε να υλοποιήσουμε τον αλγόριθμο Newton-Raphson με την χρήση του προγραμματιστικού περιβάλλοντος MATLAB.

Έχουμε:

```
function [root, steps] = Newton_Raphson(x0, tol, f, df)
 % INPUT:
 %   x0 = The initial approximation of the root
 %      tol = The value that we have as critera of termination
 %   f = The function that we want to find her root
 %   df = The derivative of the function
 % OUTPUT:
 %   root = The approximation of the real root
 %   steps = The number of the steps that where made to achive the root.

 steps = 0;
 xN = x0 - f(x0)/df(x0);
 while (abs(x0-xN) > tol) && (f(xN) ~= 0)
  x0 = xN;
  xN = x0 - f(x0)/df(x0);
  steps = steps + 1;
 endwhile
 root = xN;
```

Εικόνα 5 Ο αλγόριθμος Newton-Raphson στο MATLAB.

Όπως βλέπουμε παραπάνω, ο αλγόριθμος δέχεται σαν είσοδο μια αρχική προσέγγιση x_0 της ρίζας που ψάχνουμε, το μέγεθος του σφάλματος που δεχόμαστε με την τιμή *tol*, όπου πιο συγκεκριμένα ως κριτήριο θεωρούμε το μήκος του διαστήματος να είναι αρκετά μικρό, δηλαδή ο μέσος να συμπίπτει με τα άκρα του διαστήματος με τα αρχικά ψηφία τους να είναι αρκετά ίσα, τουλάχιστον σε 3 δεκαδικά ψηφία, το *f*, όπου με αυτό εισάγουμε την συνάρτηση με την οποία θέλουμε να δουλέψουμε και να βρούμε μια ρίζα της και σαν τελευταίο όρισμα έχουμε το *df* όπου με αυτό εισάγουμε την πρώτη παράγωγο της συνάρτησης μας. Σαν έξοδο, λαμβάνουμε δύο τιμές, την τιμή *root* η οποία είναι η προσέγγιση της πραγματικής ρίζας και την τιμή *steps* η οποία μας δίνει τον αριθμό των επαναλήψεων που έκανε ο αλγόριθμος, ώστε να μας δώσει τα αποτελέσματα που αναζητάμε.

Ένα παράδειγμα εκτέλεσης του παραπάνω αλγορίθμου είναι το εξής:

```
>> [r,n] = Newton_Raphson(2, 1e-5, @(x) x.^3 + 2*x.^2-3*x-1,  @(x) 3*x.^2+4*x-3)
r = 1.1987
n = 4
```

Εικόνα 6 Παράδειγμα εκτέλεσης του αλγορίθμου Newton-Raphson στο MATLAB.

Αρχικά, διαλέγουμε να αποθηκεύσουμε τις τιμές που επιστρέφει ο αλγόριθμος μας στο διάνυσμα $[r, n]$, όπου στην μεταβλητή r θα αποθηκεύσουμε την τιμή της ρίζα που βρήκαμε και στην μεταβλητή n θα αποθηκεύσουμε τον αριθμό των βημάτων που εκτελέστηκαν.

Στη συνέχεια βλέπουμε πως καλούμε τον αλγόριθμο με είσοδο την αρχική προσέγγιση της ρίζα ίση με 2, με τιμή για το σφάλμα ε το *1e-5* , το οποίο ισούται με $1 \times 10^{-5} = 0{,}00001$ και ως συνάρτηση της οποίας αναζητούμε την ρίζα της, την συνάρτηση $f(x) = x^3 + 2x^2 - 3x - 1$ με παράγωγο την συνάρτηση $f'(x) = 3x^2 + 4x - 3$.

Η έξοδος του αλγόριθμου βλέπουμε πως είναι ίση με r = *1.1987* και n = *4*, πράγμα το οποίο σημαίνει πως μέσα σε 4 επαναλήψεις, ο αλγόριθμος υπολόγισε πως η τιμή της ρίζα της εξίσωσης είναι ίσή με 1.1987. Αναφορικά θα πούμε πως η πραγματική ρίζα είναι ίση με ξ =1.1986912435. Επομένως η προσέγγιση r είναι αρκετά ακριβής σε σχέση με την πραγματική ρίζα ξ.

Ας δοκιμάσουμε να πάρουμε άλλες προσεγγίσεις για την ρίζα:

- Για προσέγγιση $x_0 = 1$ έχουμε:

```
>> [r,n] = Newton_Raphson(2, 1e-5, @(x) x.^3 + 2*x.^2-3*x-1,  @(x) 3*x.^2+4*x-3)
r = 1.1987
n = 3
```

Εικόνα 7 Παράδειγμα εκτέλεσης του αλγορίθμου Newton-Raphson στο MATLAB για αρχική προσέγγιση ρίζας $x_0 = 1$.

- Για προσέγγιση $x_0 = 3$ έχουμε:

```
>> [r,n] = Newton_Raphson(2, 1e-5, @(x) x.^3 + 2*x.^2-3*x-1,  @(x) 3*x.^2+4*x-3)
r = -0.2865
n = 3
```

Εικόνα 8 Παράδειγμα εκτέλεσης του αλγορίθμου Newton-Raphson στο MATLAB για αρχική προσέγγιση ρίζας $x_0 = 1$.

- Για προσέγγιση $x_0 = 0$ έχουμε:

```
>> [r,n] = Newton_Raphson(2, 1e-5, @(x) x.^3 + 2*x.^2-3*x-1,  @(x) 3*x.^2+4*x-3)
r = 1.1987
n = 5
```

Εικόνα 9 Παράδειγμα εκτέλεσης του αλγορίθμου Newton-Raphson στο MATLAB για αρχική προσέγγιση ρίζας $x_0 = 3$.

Αρχικά παρατηρούμε πως αν διαλέξουμε ως αρχική τιμή το 1, ο αλγόριθμος συγκλίνει πιο γρήγορα στην ρίζα, μιας και γεωμετρικά είναι πιο κοντά σε αυτή. Όμοια μας το επιβεβαιώνει αυτό και αν πάρουμε ως αρχική τιμή το 3. Αυτό που βλέπουμε είναι πως ο αλγόριθμος εκτελεί ένα βήμα παραπάνω απ' όταν είχαμε ως αρχική τιμή το 2.

1.3.5 Σχόλια – παρατηρήσεις

Το πρώτο αξιοσημείωτο συμπέρασμα που βγάζουμε από αυτά, είναι πως παρόλο που η αρχική προσέγγιση στην περίπτωση που διαλέξουμε το 3 είναι αρκετά μακριά από την πραγματική ρίζα, ο αλγόριθμος κατάφερε ταχύτατα να βρει την ζητούμενη προσέγγιση, σε σημείο ακόμα και να ξεπερνάμε κατά πολύ ακόμα και τον αλγόριθμο της Εσφαλμένης Θέσης, που υποτίθεται έχει να εντοπίσει την ρίζα μέσα σε ένα σχετικά μικρό διάστημα.

Το δεύτερο αξιοσημείωτο που παρατηρούμε, είναι πως αν θεωρήσουμε ως αρχική τιμή όχι το 1 αλλά το 0, η μέθοδος εστιάζει, με βάση την γεωμετρία της συνάρτησης μας, σε μια άλλη ρίζα, την -0.2865, μιας και σε αυτή είναι πιο κοντά.

Το τελευταίο καθώς και το γεγονός πως ο τύπος της ακολουθίας είναι ο

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, n = 0,1,2, \dots, f'(x_n) \neq 0$$

μας φανερώνουν μια μικρή λεπτομέρεια για την μέθοδο που εξετάζουμε, πως η ακολουθία που σχηματίζεται, εξαρτάται κατά έναν μεγάλο βαθμό από την συνάρτηση μας, και πιο συγκεκριμένα από την κλίση της και το σχήμα της.

2

2. Αριθμητική Επίλυση Γραμμικών Συστημάτων

Μετά την μελέτη και την εύρεση ριζών μιας συνάρτησης, το αμέσως λογικό άλμα το οποίο μπορούμε να κάνουμε είναι να προσπαθήσουμε να επιλύσουμε συστήματα αλγεβρικών γραμμικών εξισώσεων, μιας και είναι το αμέσως πιο συχνό πρόβλημα το οποίο καλούμαστε να λύσουμε σε πάρα πολλές εφαρμογές.

Ένα σύστημα γραμμικών εξισώσεων είναι ένα σύστημα το οποίο είναι της μορφής:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n = b_3$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n$$

όπου κάθε γραμμή αντιπροσωπεύει και μια γραμμική εξίσωση με την i - οστή γραμμή,

$$a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 + \cdots + a_{in}x_n = b_i$$

αποτελείται από τους αγνώστους $x_1, x_2, x_3, \ldots, x_n$, τους συντελεστές των αγνώστων $a_{i1}, a_{i2}, a_{i3}, \ldots, a_{in}$και τους σταθερούς όρους $b_1, b_2, b_3, \ldots, b_n$.

Επιπλέον θα ασχοληθούμε με γραμμικά συστήματα τα οποία είναι τετραγωνικά, δηλαδή ο πίνακας των συντελεστών

$$\mathrm{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

είναι τετραγωνικός πίνακας, και πιο συγκεκριμένα θα ασχοληθούμε με πυκνά γραμμικά συστήματα, συστήματα των οποίων ο πίνακας συντελεστών A αποτελείται από κυρίως από μη μηδενικά στοιχεία, πάνω από το 70-80% τους στο πλήθος.

Επίσης, όπως μπορούμε να συμβολίσουμε τους συντελεστές σε μορφή πίνακα, μπορούμε να κατασκευάσουμε και τους αντίστοιχους πίνακες για τους αγνώστους και τους σταθερούς όρους ως

$$\mathrm{x} = \begin{bmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ \vdots \\ \mathrm{x}_n \end{bmatrix}, \mathrm{b} = \begin{bmatrix} \mathrm{b}_1 \\ \mathrm{b}_2 \\ \vdots \\ \mathrm{b}_n \end{bmatrix}$$

και έτσι έχουμε το σύστημα στην πινακική μορφή $\mathrm{Ax} = \mathrm{b}$.

2.1 Η μέθοδος απαλοιφής Gauss

Ας θεωρήσουμε το γραμμικό σύστημα:

$$\alpha_{11}^{(1)} x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + \cdots + a_{1n}^{(1)} x_n = b_1^{(1)}$$

$$\alpha_{21}^{(1)} x_1 + \alpha_{22}^{(1)} x_2 + \alpha_{23}^{(1)} x_3 + \cdots + \alpha_{2n}^{(1)} x_n = b_2^{(1)}$$

$$\alpha_{31}^{(1)} x_1 + \alpha_{32}^{(1)} x_2 + \alpha_{33}^{(1)} x_3 + \cdots + \alpha_{3n}^{(1)} x_n = b_3^{(1)}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$\alpha_{n1}^{(1)} x_1 + \alpha_{n2}^{(1)} x_2 + \alpha_{n3}^{(1)} x_3 + \cdots + \alpha_{nn}^{(1)} x_n = b_n^{(1)}$$

το οποίο μπορούμε να το γράψουμε σε πινακική μορφή ως:

$$A^{(1)} x = b^{(1)}$$

Υποθέτουμε αρχικά ότι $\det A \neq 0$, έτσι ώστε να εξασφαλίσουμε πως θα υπάρχει λύση και πως αυτή η λύση θα είναι μοναδική. Επιπλέον, θεωρούμε πως ο πίνακας - διάνυσμα των σταθερών όρων $b^{(1)}$ είναι διάφορος του μηδενικού διανύσματος.

Η βασική ιδέα της μεθόδου είναι η εξής: Θα προσπαθήσουμε με βασικές γραμμοπράξεις, να κατασκευάσουμε ένα ισοδύναμο σύστημα, το οποίο θα αποτελείται από εξισώσεις οι οποίες θα αποτελούνται από λιγότερους αγνώστους, ώστε να λύνονται υπολογιστικά πιο εύκολα και γρήγορα. Πιο συγκεκριμένα, θα κατασκευάσουμε ένα σύστημα το οποίο θα είναι "άνω τριγωνικό", δηλαδή ο πίνακας A θα έχει άνω τριγωνική μορφή.

Δηλαδή από το παραπάνω σύστημα που έχουμε, θέλουμε να κατασκευάσουμε ένα σύστημα στο οποίο από την δεύτερη εξίσωση και κάτω, θα απαλειφθεί ο άγνωστος x_1, δηλαδή θα έχουμε το σύστημα:

$$\alpha_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + a_{13}^{(1)}x_3 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)}$$

$$\alpha_{22}^{(2)}x_2 + \alpha_{23}^{(2)}x_3 + \cdots + \alpha_{2n}^{(2)}x_n = b_2^{(2)}$$

$$\alpha_{32}^{(2)}x_2 + \alpha_{33}^{(2)}x_3 + \cdots + \alpha_{3n}^{(2)}x_n = b_3^{(2)}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$\alpha_{n2}^{(2)}x_2 + \alpha_{n3}^{(2)}x_3 + \cdots + \alpha_{nn}^{(2)}x_n = b_n^{(2)}$$

το οποίο μπορούμε να το γράψουμε σε πινακική μορφή ως:

$$A^{(2)}x = b^{(2)}$$

Με βάση αυτό το νέο σύστημα, το οποίο είναι ισοδύναμο με το πρώτο, θα προσπαθήσουμε να φτιάξουμε ένα ισοδύναμο και πάλι, μόνο που τώρα θα απαλειφθεί από την τρίτη εξίσωση και κάτω ο άγνωστος x_2, δηλαδή θα έχουμε το σύστημα:

$$\alpha_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + a_{13}^{(1)}x_3 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)}$$

$$\alpha_{22}^{(2)}x_2 + \alpha_{23}^{(2)}x_3 + \cdots + \alpha_{2n}^{(2)}x_n = b_2^{(2)}$$

$$\alpha_{33}^{(3)}x_3 + \cdots + \alpha_{3n}^{(3)}x_n = b_3^{(3)}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$\alpha_{n3}^{(3)}x_3 + \cdots + \alpha_{nn}^{(3)}x_n = b_n^{(3)}$$

το οποίο μπορούμε να το γράψουμε σε πινακική μορφή ως:

$$A^{(3)}x = b^{(3)}$$

Όπως παρατηρούμε, την πρώτη και την δεύτερη εξίσωση δεν τις επεξεργαστήκαμε σε αυτό το βήμα, μιας και δεν μας ενδιαφέρουν πλέον, αφού σιγά σιγά γίνονται μέρος του τριγωνικού πίνακα στον οποίο θέλουμε να καταλήξουμε.

Μετά από *r-1* επαναλήψεις, με $r < n$, θα έχουμε καταλήξει σε ένα σύστημα με την εξής μορφή:

$$\alpha_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1,r-1}^{(1)}x_{r-1} + a_{1,r}^{(1)}x_r + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)}$$

$$\alpha_{22}^{(2)}x_2 + \cdots + a_{1,r-1}^{(2)}x_{r-1} + a_{1,r}^{(2)}x_r + \cdots + \alpha_{2n}^{(2)}x_n = b_2^{(2)}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$a_{r-1,r-1}^{(r-1)}x_{r-1} + a_{r-1,r}^{(r-1)}x_r + \cdots + \alpha_{r-1n}^{(r-1)}x_n = b_{r-1}^{(r-1)}$$

$$a_{r,r}^{(r)}x_r + \cdots + \alpha_{rn}^{(r)}x_n = b_r^{(r)}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$a_{nr}^{(r)}x_r + \cdots + \alpha_{nn}^{(r)}x_n = b_n^{(r)}$$

το οποίο μπορούμε να το γράψουμε σε πινακική μορφή ως:

$$A^{(r-1)}x = b^{(r-1)}$$

Μετά από *n* επαναλήψεις, θα έχουμε καταλήξει σε ένα σύστημα με την εξής μορφή:

$$\alpha_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1,r-1}^{(1)}x_{r-1} + a_{1,r}^{(1)}x_r + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)}$$

$$\alpha_{22}^{(2)}x_2 + \cdots + a_{1,r-1}^{(2)}x_{r-1} + a_{1,r}^{(2)}x_r + \cdots + \alpha_{2n}^{(2)}x_n = b_2^{(2)}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$a_{r-1,r-1}^{(r-1)}x_{r-1} + a_{r-1,r}^{(r-1)}x_r + \cdots + \alpha_{r-1n}^{(r-1)}x_n = b_{r-1}^{(r-1)}$$

$$a_{r,r}^{(r)}x_r + \cdots + \alpha_{rn}^{(r)}x_n = b_r^{(r)}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$\alpha_{nn}^{(n)}x_n = b_n^{(n)}$$

το οποίο μπορούμε να το γράψουμε σε πινακική μορφή ως:

$$A^{(n)}x = b^{(n)}$$

όπου ο πίνακας $A^{(n)}$ είναι άνω τριγωνικός.

Τώρα το τελευταίο σύστημα μπορεί να λυθεί αρκετά εύκολα και γρήγορα, με την προς τα πίσω αντικατάσταση από τον τύπο

$$x_n = \frac{b_n^{(n)}}{\alpha_{nn}^{(n)}}$$

και για τους υπόλοιπους αγνώστους από τον τύπο

$$x_i = \frac{b_i^{(i)} - \sum_{j=i+1}^{n} a_{ij}^{(i)}x_j}{\alpha_{ii}^{(i)}}, i = n-1, n-2, \ldots, 1$$

Θα δούμε πιο αναλυτικά τώρα πως κατασκευάζονται τα ισοδύναμα συστήματα και αντίστοιχα οι πινακικές μορφές τους. Η μέθοδος απαλοιφής του Gauss ουσιαστικά κατασκευάζει μια ακολουθία από πίνακες συντελεστών $A^{(k)}$, $k = 1,2,\dots,n$, όπου $A^{(1)} = A$ και μια ακολουθία από πίνακες- διανύσματα σταθερών όρων $b^{(k)}, k = 1,2,\dots,n$, με τελικό στόχο ο $A^{(n)}$να είναι άνω τριγωνικός. Το πρώτο βήμα είναι να απαλειφθεί ο άγνωστος x_1από τις υπόλοιπες *n-1*επόμενες εξισώσεις. Για να καταφέρουμε να απαλείψουμε το x_1 από την *i-οστή* εξίσωση, προσθέτουμε την πρώτη εξίσωση, που την ονομάζουμε και *οδηγό εξίσωση,* σε αυτή ενώ ταυτόχρονα την έχουμε πολλαπλασιάσει με την ποσότητα

$$m_{\mathrm{i1}} = -\frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, i = 2,3,\dots,n$$

όπου $a_{11}^{(1)} \neq 0$. Το στοιχείο $a_{11}^{(1)}$ καλείται *οδηγό στοιχείο.*

Το αποτέλεσμα της παραπάνω πράξης παραπάνω θα είναι το εξής:

$$\left[a_{i2}^{(1)} + m_{\mathrm{i1}}a_{12}^{(1)}\right]x_2 + \cdots + \left[a_{in}^{(1)} + m_{\mathrm{i1}}a_{1n}^{(1)}\right]x_n = \ b_i^{(1)} + m_{\mathrm{i1}}b_1^{(1)}$$

όπου καταλήγουμε στο

$$a_{i2}^{(2)}x_2 + \cdots + a_{i2}^{(2)}x_n = \ b_i^{(2)}$$

με

$$a_{ij}^{(2)} = \ a_{ij}^{(1)} + m_{\mathrm{i1}}a_{1j}^{(1)}, i = 2,3,\cdots,n, j = 2,3,\cdots,n$$

$$b_i^{(2)} = \ b_i^{(1)} + m_{\mathrm{i1}}b_1^{(1)}, i = 2,3,\cdots,n$$

Με βάση αυτόν τον συνδυασμό πράξεων, καταφέρνουμε να απαλείψουμε τον άγνωστο x_1αφού

$$a_{i1}^{(2)} = a_{i2}^{(1)} + m_{\mathrm{i1}}a_{12}^{(1)} = 0$$

Στη συνέχεια, θεωρούμε το σύστημα με όλες τις εξισώσεις εκτός από την εξίσωση οδηγό, δηλαδή τις υπόλοιπες *n-1* εξισώσεις

$$a_{22}^{(2)}x_2 \ + \cdots + a_{2n}^{(2)}x_n \ = \ b_2^{(2)}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$a_{n2}^{(2)}x_2 + \cdots + a_{nn}^{(2)}x_n = b_n^{(2)}$$

Αν τώρα έχουμε ότι $a_{22}^{(2)} \neq 0$, τότε μπορούμε να ορίσουμε τους αντίστοιχους πολλαπλασιαστές

$$m_{\mathrm{i2}} = -\frac{a_{i2}^{(2)}}{a_{22}^{(2)}}, i = 3, \ldots, n$$

για να γίνει η απαλοιφή του αγνώστου x_1από τις επόμενες *n-2*εξισώσεις.

Επομένως, παρατηρούμε πως από αυτή την διαδικασία, σε κάθε βήμα, κατασκευάζεται και ένας αντίστοιχος πίνακας με τους πολλαπλασιαστές, για παράδειγμα έχουμε τον πίνακα:

$$M^{(1)} = \begin{bmatrix} 1 & 0 \\ m_{21} & \\ m_{31} & I_{n-1} \\ \vdots & \\ m_{\mathrm{n1}} & \end{bmatrix}$$

όπου I_{n-1} είναι ο μοναδιαίος πίνακας με *n-1* γραμμές και στήλες. Έτσι αν πολλαπλασιάσουμε τον πίνακα $M^{(1)}$από αριστερά με τον πίνακα $A^{(1)}$ και αντίστοιχα στο άλλο μέλος του συστήματος θα έχουμε ότι:

$$M^{(1)}A^{(1)}x = M^{(1)}b^{(1)}$$

δηλαδή θα προκύψει ότι:

$$A^{(2)}x = b^{(2)}$$

με

$A^{(2)} = M^{(1)}A^{(1)}$ και $b^{(2)} = M^{(1)}b^{(1)}$

και αντίστοιχα προκύπτουν οι υπόλοιποι πίνακες.

2.1.1 Ο αλγόριθμος της απαλοιφής Gauss

Ο παρακάτω αλγόριθμος παρουσιάζει την μέθοδο της απαλοιφής Gauss για την λύση του συστήματος $Ax = b$

1. Διαβάζουμε τους πίνακες $A = (a_{ij}), 1 \leq i, j \leq n, b = (b_i), 1 \leq i \leq n.$

2. Για $i = 1,2,\ldots,n$να τεθεί $a_{i,n+1} = b_i$

3. Για $r = 1,2,\ldots,n-1$εκτελούνται τα παρακάτω βήματα

   a) Υπολογίζουμε τους πολλαπλασιαστές

$$m_{\text{ir}} = -\frac{a_{ir}}{a_{rr}}$$

   b) Για $j = r+1, r+2, \ldots, n+1$να υπολογιστούν τα εξής:

$$a_{ij} = a_{ij} + m_{ir}a_{rj}$$

4. Να υπολογιστεί η οπισθοδρόμηση για την τελευταία γραμμή

$$x_n = \frac{a_{n,n+1}}{a_{nn}}$$

5. Για $\text{i} = \text{n}-1, \text{n}-2, \ldots, 1$ να υπολογιστεί

$$x_i = \frac{a_{i,n+1} - \sum_{j=i+1}^{n} a_{ij}x_j}{\alpha_{ii}}$$

6. Εκτύπωσε την λύση

Ας εξετάσουμε ένα ένα τα βήματα του αλγόριθμου. Αρχικά, για υπολογιστικούς λόγους και μόνο, επαυξάνουμε τον πίνακα τον συντελεστών *A* βάζοντας το διάνυσμα των σταθερών όρων *b* ως μια επιπλέον στήλη, στην θέση *n+1* .

Στη συνέχεια εκτελούμε με την σειρά τα εξής:

- Υπολογίζουμε τον πολλαπλασιαστή της γραμμής στην οποία θέλουμε να κρατήσουμε τον άγνωστο και να τον απαλείψουμε από τις υπόλοιπες από κάτω
- Εκτελούμε τις πράξεις έτσι ώστε στις εξισώσεις μετά από αυτή που βρισκόμαστε να απαλειφθεί με επιτυχία ο άγνωστος.

Αφού ολοκληρωθεί αυτή η διαδικασία, ξεκινάμε την οπισθοδρόμηση, δηλαδή ξεκινάμε να υπολογίζουμε τις τιμές των αγνώστων μας.

2.1.2 Ο αλγόριθμος της απαλοιφής Gauss στο MATLAB

Παρακάτω, θα παρουσιάσουμε τις εντολές με τις οποίες μπορούμε να υλοποιήσουμε τον αλγόριθμο της απαλοιφής Gauss με την χρήση του προγραμματιστικού περιβάλλοντος MATLAB.

Έχουμε:

```
function [x] = GaussE(A,b)
 % INPUTS
 %   A is a nxn matrix
 %   b is a nx1 matrix
 % OUTPUT
 %   x that is the solution of the Ax = b

 A_b = [A b]; % Create the augmented matrix
 n = length(A);
 x = zeros(n,1);
 for r = 1:(n-1)
   for i = r+1:n
    if A_b(r,r)~=0
     m = -A_b(i,r)/A_b(r,r); % Create multipliers
     for j = r:n+1
      A_b(i,j) = A_b(i,j) + m*A_b(r,j); % Update the row
     endfor
    endif
   endfor
 endfor
```

```
if A_b(n,n)~= 0
   x(n) = A_b(n+1)/A_b(n,n);
 endif
 for i=n-1:-1:1
   if A_b(i,i)~=0
     sum = 0;
     for j = i+1:n
      sum = sum + A_b(i,j)*x(j);
     endfor
     x(i) = (A_b(i,n+1)-sum)/A_b(i,i); % Calculate the x matrix
    endif
 endfor
```

Εικόνα 10 Ο αλγόριθμος της απαλοιφής Gauss στο MATLAB.

Ο αλγόριθμος μας βλέπουμε πως δέχεται 2 ορίσματα, το όρισμα *A*,που αντιστοιχεί στον πίνακα των συντελεστών του γραμμικού συστήματος του οποίου ψάχνουμε την λύση και τον πίνακα - διάνυσμα *b* που αντιστοιχεί στους σταθερούς όρους του συστήματος. Σαν αποτέλεσμα, ο αλγόριθμος επιστρέφει το διάνυσμα *x*, το οποίο αποτελεί την λύση του γραμμικού συστήματος.

Ένα παράδειγμα χρήσης του αλγορίθμου είναι το εξής:

```
>> A = [-1 2 -1; 2 -1 0; 1 7 -3]
A =

 -1  2 -1
  2 -1  0
  1  7 -3

>> b = [0; 1; 5]
b =

  0
  1
  5

>> x = GaussE(A,b)
x =

  1
  1
  1
```

Εικόνα 11 Παράδειγμα εκτέλεσης του αλγορίθμου απαλοιφής Gauss στο MATLAB.

Σε αυτό το παράδειγμα καλούμαστε να λύσουμε το γραμμικό σύστημα:

$$-x_1 + 2x_2 - x_3 = 0$$

$$2x_1 - 1x_2 = 1$$

$$x_1 + 7x_2 - 3x_3 = 5$$

Όπως βλέπουμε, το συγκεκριμένο γραμμικό σύστημα έχει ως πίνακα συντελεστών τον πίνακα:

$$A = \begin{bmatrix} -1 & 2 & -1 \\ 2 & -1 & 0 \\ 1 & 7 & -3 \end{bmatrix}$$

και πίνακα συντελεστών τον πίνακα:

$$b = \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix}$$

Μετά την εκτέλεση του αλγορίθμου μας, επιστρέφεται ως λύση το διάνυσμα:

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

το οποίο είναι όντως του γραμμικού συστήματος που εξετάζουμε.

2.1.3 Η ανάγκη για βελτίωση

Παρατηρούμε πως το πρώτο και πιο βασικό πρόβλημα - περιορισμό το οποίο πρέπει να αντιμετωπίσουμε στην απαλοιφή του Gauss, είναι η περίπτωση να εμφανιστεί μηδενικό στοιχείο ως οδηγό στοιχείο.

Πιο συγκεκριμένα, έστω πως βρισκόμαστε στο r βήμα και καλούμαστε να κατασκευάσουμε τους πολλαπλασιαστές m_{ir}, οι οποίοι δίνονται από την σχέση:

$$m_{ir} = -\frac{a_{ir}}{a_{rr}}$$

και παρατηρούμε πως το στοιχείο a_{rr} ισούται με 0. Αυτό έχει ως αποτέλεσμα την μη ορθή υλοποίηση του αλγορίθμου, μιας και ή θα μας προκύψει κάποιο μήνυμα σφάλματος ή απλά θα προκύψουν λανθασμένα αποτελέσματα στην λύση, αν έχουμε προνοήσει να βάλουμε σημεία ελέγχου για τις τιμές των στοιχείων του πολλαπλασιαστή. Ένα χειροπιαστό παράδειγμα τέτοιας περίπτωσης είναι το ακόλουθο.

```
>> A = [-1 2 -1; 2  0 -1 ; 1 7 -3]
A =
 -1   2  -1
  2   0  -1
  1   7  -3

>> b = [0; 1; 5]
b =
  0
  1
  5

>> x = GaussE(A,b)
x =
  0.6143
  0.4214
  0.2286

>> A*x == b
ans =
 0
 1
 0
```

Εικόνα 12 Παράδειγμα μη ορθής υλοποίησης του αλγορίθμου απαλοιφής Gauss στο MATLAB.

Όπως βλέπουμε, ο αλγόριθμος στην προκειμένη περίπτωση καλείται να λύση ένα γραμμικό σύστημα της μορφής:

$$-x_1 + 2x_2 - x_3 = 0$$

$$2x_1 \qquad - x_3 = 1$$

$$x_1 + 7x_2 - 3x_3 = 5$$

Επομένως, το συγκεκριμένο γραμμικό σύστημα έχει ως πίνακα συντελεστών τον πίνακα:

$$A = \begin{bmatrix} -1 & 2 & -1 \\ 2 & 0 & -1 \\ 1 & 7 & -3 \end{bmatrix}$$

και πίνακα συντελεστών τον πίνακα:

3

$$b = \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix}$$

Μετά την εκτέλεση του αλγορίθμου μας, επιστρέφεται ως λύση το διάνυσμα:

$$x = \begin{bmatrix} 0.6143 \\ 0.4214 \\ 0.2286 \end{bmatrix}$$

το οποίο όμως διαφέρει κατά πολύ από την σωστή λύση του γραμμικού συστήματος που εξετάζουμε, διότι βλέπουμε πως όταν εκτελούμε τον έλεγχο αν το γινόμενο *A*x είναι ίσο με το διάνυσμα *b*, παρατηρούμε πως δεν είναι ίσα, λόγω των μηδενικών που εμφανίζονται.

Για την ορθή και αποτελεσματική αντιμετώπιση θα κατασκευάσουμε δυο αναβαθμισμένες εκδόσεις του αλγορίθμου μας

1. Η μέθοδος Απαλοιφής Gauss με Μερική Οδήγηση
2. Η μέθοδος Απαλοιφής Gauss με Ολική Οδήγηση

2.2 Η απαλοιφή Gauss με μερική οδήγηση

Η μέθοδος της απαλοιφής Gauss με Μερική Οδήγηση έχει ως βασική ιδέα την εξής:

Έστω πως εκτελούμε το στο r βήμα του αλγορίθμου και καλούμαστε να κατασκευάσουμε τους πολλαπλασιαστές m_{ir}, οι οποίοι δίνονται από την σχέση:

$$m_{ir} = -\frac{a_{ir}}{a_{rr}}$$

και παρατηρούμε πως το στοιχείοa_{rr}ισούται με 0.Για να αποφύγουμε την διαίρεση με μηδενικό στοιχείο, θα προσπαθήσουμε να βρούμε στην ίδια στήλη ένα στοιχείο το οποίο δεν είναι μηδενικό.

Πιο συγκεκριμένα, θα αναζητήσουμε στα επόμενα στοιχεία της στήλης ένα στοιχείο που είναι διάφορο του μηδενός και όταν το βρούμε, θα αλλάξουμε τις δυο αυτές γραμμές μεταξύ τους. Η κίνηση αυτή είναι μαθηματικά ορθή, μιας και από την Γραμμική Άλγεβρα γνωρίζουμε ότι η αλλαγή γραμμών σε έναν πίνακα, δεν επιφέρει αλλαγές στο τελικό αποτέλεσμα του συστήματος.

2.2.1 Ο αλγόριθμος της απαλοιφής Gauss με μερική οδήγηση

Ο παρακάτω αλγόριθμος παρουσιάζει την μέθοδο της απαλοιφής Gauss με μερική οδήγηση για την λύση του συστήματος $Ax = b$

1. Διαβάζουμε τους πίνακες $A = (a_{ij}), 1 \leq i, j \leq n, b = (b_i), 1 \leq i \leq n.$
2. Για $i = 1,2,\ldots,n$ να τεθεί

$$a_{i,n+1} = b_i$$

3. Για $i = 1,2,\ldots,n$ να τεθεί

$$h(i) = i$$

4. Για $r = 1,2,\ldots,n-1$ να εκτελεστούν τα παρακάτω βήματα
   a) Έστω p ο μικρότερος ακέραιος για τον οποίο ισχύει ότι

$$r \leq p \leq n$$

   και ισχύει

$$|a(h(p),r)| = \max_{r \leq i \leq n} |a(h(i),r)|$$

   b) Αν $a(h(p),r) = 0$, τότε δεν υπάρχει μοναδική λύση.
   c) Αν $h(r) \neq h(p)$, τότε κάνουμε αλλαγή των γραμμών, δηλαδή έχουμε

$$q = h(r)$$

$$h(r) = h(p)$$

$$h(p) = q$$

   d) Για $i = r+1, r+2, \ldots, n$ να εκτελεστούν τα παρακάτω
      i. Να υπολογιστούν οι πολλαπλασιαστές

$$m(h(i),r) = -\frac{a(h(i),r)}{a(h(r),r)}$$

      ii. Για $j = r+1, r+2, \ldots, n$ να υπολογιστούν οι νέες τιμές των συντελεστών

$$a(h(i),j) = a(h(i),j) + m(h(i),r)a(h(r),j)$$

5. Να υπολογιστεί ο όρος

$$x_n = \frac{a(h(n), n+1)}{a(h(n), n)}$$

6. Για $i = n-1, n-2, \ldots, 1$ να υπολογιστούν οι υπόλοιποι όροι

$$x_i = \frac{a(h(i), n+1) - \sum_{j=n+1}^{n} a(h(i), j)x_j}{a(h(i), i)}$$

7. Εκτύπωσε τη λύση.

2.2.2 Ο αλγόριθμος της απαλοιφής Gauss με μερική οδήγηση στο MATLAB

Παρακάτω, θα παρουσιάσουμε τις εντολές με τις οποίες μπορούμε να υλοποιήσουμε τον αλγόριθμο της απαλοιφής Gauss με Μερική Οδήγηση με την χρήση του προγραμματιστικού περιβάλλοντος MATLAB.

Έχουμε:

```
function x = GaussEPartial(A,b)
% INPUTS
%   A is a nxn matrix
%   b is a nx1 matrix
% OUTPUT
%   x that is the solution of the Ax = b

 A_b = [A b]; % Create the augmented matrix
 [m,n] = size(A);
 x = zeros(m,1);

 for r = 1:m-1
  for p = r:n
   if (abs(A_b(r,r))<abs(A_b(p,r)))
    A_b([r p],:) = A_b([p r],:); % Change the rows
   endif
  endfor
  for i = r+1 : n
   mp = - A_b(i,r)/A_b(r,r); % Create multipliers
   for j = r:n+1
    A_b(i,j) = A_b(i,j) + mp*A_b(r,j);
   endfor
  endfor
 endfor

 x(m) = A_b(m,n+1)\A_b(m,m);
 for i=m-1:-1:1
  if A_b(i,i)~=0
   sum = 0;
   for j = i+1:n
    sum = sum + A_b(i,j)*x(j);
    endfor
    x(i) = (A_b(i,n+1)-sum)/A_b(i,i); % Calculate the x matrix
   endif
 endfor
```

Εικόνα 13 Ο αλγόριθμος της απαλοιφής Gauss με μερική οδήγηση στο MATLAB.

Ο αλγόριθμος μας βλέπουμε πως δέχεται 2 ορίσματα, το όρισμα *A*, που αντιστοιχεί στον πίνακα των συντελεστών του γραμμικού συστήματος του οποίου ψάχνουμε την λύση και τον πίνακα - διάνυσμα *b* που αντιστοιχεί στους σταθερούς όρους του συστήματος. Σαν αποτέλεσμα, ο αλγόριθμος επιστρέφει το διάνυσμα *x*, το οποίο αποτελεί την λύση του γραμμικού συστήματος, με την χρήση της μερικής οδήγησης.

Ένα παράδειγμα χρήσης του αλγορίθμου είναι το εξής:

```
>> A = [-1 2 -1; 2  0 -1 ; 1 7 -3]
A =

 -1   2  -1
  2   0  -1
  1   7  -3

>> b = [0; 1; 5]
b =

  0
  1
  5

>> x = GaussEPartial(A,b)
x =

  1.0000
  1.0000
  1.0000
```

Εικόνα 14 Παράδειγμα εκτέλεσης του αλγορίθμου απαλοιφής Gauss με μερική οδήγηση στο MATLAB.

Όπως βλέπουμε, ο αλγόριθμος στην προκειμένη περίπτωση καλείται να λύση ένα γραμμικό σύστημα της μορφής:

$$-x_1 + 2x_2 - x_3 = 0$$

$$2x_1 \qquad - x_3 = 1$$

$$x_1 + 7x_2 - 3x_3 = 5$$

Επομένως, το συγκεκριμένο γραμμικό σύστημα έχει ως πίνακα συντελεστών τον πίνακα:

$$A = \begin{bmatrix} -1 & 2 & -1 \\ 2 & 0 & -1 \\ 1 & 7 & -3 \end{bmatrix}$$

και πίνακα συντελεστών τον πίνακα:

$$b = \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix}$$

Μετά την εκτέλεση του αλγορίθμου μας, επιστρέφεται ως λύση το διάνυσμα:

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

το οποίο είναι η σωστή λύση του γραμμικού συστήματος.

Παρατηρούμε πως με την μέθοδο χωρίς οδήγηση, ο αλγόριθμός μας μας έδινε προφανώς λάθος αποτέλεσμα. Ωστόσο η νέα μας μέθοδος κατάφερε να λύσει με επιτυχία τα προβλήματα που προέκυψαν.

2.3 Η απαλοιφή Gauss με ολική οδήγηση

Ένας ακόμα αποτελεσματικός τρόπος για να αντιμετωπίσουμε τα προβλήματα που προκύπτουν κατά την εκτέλεση της απαλοιφής Gauss χωρίς οδήγηση, είναι να πάρουμε μια πιο γενική περίπτωση της απαλοιφής Gauss με Μερική Οδήγηση.

Όπως είδαμε παραπάνω, στην απαλοιφή Gauss με Μερική Οδήγηση, αναζητούσαμε σε κάθε βήμα το μεγαλύτερο στοιχείο της στήλης ώστε να πραγματοποιήσουμε τον υπολογισμό των πολλαπλασιαστών και την ανανέωση των στοιχείων του πίνακα. Τώρα σε κάθε μας βήμα θα αναζητάμε το μεγαλύτερο στοιχείο κάτω από την γραμμή στην οποία βρισκόμαστε, και με κατάλληλες αλλαγές γραμμών και στηλών, θα κατασκευάζουμε με ασφάλεια τους πολλαπλασιαστές και θα κάνουμε την ανανέωση των στοιχείων του πίνακα μας, μέχρι αυτό να λάβει μια άνω τριγωνική μορφή.

2.3.1 Ο αλγόριθμος της απαλοιφής Gauss με ολική οδήγηση

Ο παρακάτω αλγόριθμος παρουσιάζει την μέθοδο της απαλοιφής Gauss με ολική οδήγηση για την λύση του συστήματος $Ax = b$

1. Διαβάζουμε τους πίνακες $A = (a_{ij}), 1 \leq i, j \leq n, b = (b_i), 1 \leq i \leq n.$
2. Για $i = 1,2,\ldots,n$ να τεθεί

$$a_{i,n+1} = b_i$$

3. Για $i = 1,2,\ldots,n$ και $j = 1,2,\ldots,n$ να τεθεί

$$h(i) = i$$

$$g(j) = j$$

4. Για $r = 1,2,\ldots,n-1$ να εκτελεστούν τα παρακάτω βήματα

a) Έστω p, q οι μικρότεροι ακέραιοι για τους οποίους ισχύει ότι:

$$r \leq p, q \leq n$$

και ισχύει

$$|a(h(p), g(q))| = \max_{r \leq i,j \leq n} |a(h(i), g(j))|$$

b) Αν $a(h(p), g(q)) = 0$, τότε δεν υπάρχει μοναδική λύση.

c) Αν $h(r) \neq h(p)$, τότε κάνουμε αλλαγή των γραμμών, δηλαδή έχουμε

$$w = h(r)$$

$$h(r) = h(p)$$

$$h(p) = w$$

d) Αν $g(r) \neq g(q)$, τότε κάνουμε αλλαγή των στηλών, δηλαδή έχουμε

$$t = g(r)$$

$$g(r) = g(q)$$

$$g(p) = t$$

e) Για $i, j = r + 1, r + 2, \ldots, n$ να εκτελεστούν τα παρακάτω:

i. Να υπολογιστούν οι πολλαπλασιαστές

$$m(h(i), g(r)) = -\frac{a(h(i), g(j))}{a(h(r), g(r))}$$

ii. Για$j = r + 1, r + 2, \ldots, n$ να υπολογιστούν οι νέες τιμές των συντελεστών

$$a(h(i), g(j)) = a(h(i), g(j)) + m(h(i), g(r))a(h(r), g(j))$$

5. Να υπολογιστεί ο όρος

$$x_n = \frac{a(h(n), n + 1)}{a(h(n), n)}$$

6. Για $i = n - 1, n - 2, \ldots, 1$ να υπολογιστούν οι υπόλοιποι όροι

$$x_i = \frac{a(h(i), n + 1) - \sum_{j=n+1}^{n} a(h(i), j)x_j}{a(h(i), i)}$$

7. Εκτύπωσε τη λύση.

2.3.2 Ο αλγόριθμος της απαλοιφής Gauss με ολική οδήγηση στο MATLAB

Παρακάτω, θα παρουσιάσουμε τις εντολές με τις οποίες μπορούμε να υλοποιήσουμε τον αλγόριθμο της απαλοιφής Gauss με Ολική Οδήγηση με τη χρήση του προγραμματιστικού περιβάλλοντος MATLAB.

```
function x = GaussComplete(A,b)
% INPUTS
%   A is a nxn matrix
%   b is a nx1 matrix
% OUTPUT
%   x that is the solution of the Ax = b

 A_b = [A b]; % Create the augmented matrix
 [m,n] = size(A);
 x = zeros(m,1);
 for r = 1:m-1
  % Find the maximum element on the A matrix only
  for i = r:m
   for j = r:n
    max_val = A_b(r,r);
    max_i = r;
    max_j = r;
    if A_b(i,j) > max_val
     max_val = A_b(i,j);
     max_i = i;
     max_j = j;
    endif
   endfor
  endfor
  % Swap rows
  A_b([r max_i], :) = A_b([max_i r], :);
  %Swap columns
  v = A_b(:, r);
  A_b(:, r) = A(:, max_j);
  A_b(:, max_j) = v;
```

```
for i = r+1 : n
    mp = - A_b(i,r)/A_b(r,r); % Create multipliers
    for j = r:n+1
      A_b(i,j) = A_b(i,j) + mp*A_b(r,j);
    endfor
  endfor
endfor
x(m) = A_b(m,n+1)\A_b(m,m);
for i=m-1:-1:1
   if A_b(i,i)~=0
     sum = 0;
     for j = i+1:n
      sum = sum + A_b(i,j)*x(j);
      endfor
     x(i) = (A_b(i,n+1)-sum)/A_b(i,i); % Calculate the x matrix
    endif
 endfor
```

Εικόνα 15 Ο αλγόριθμος απαλοιφής Gauss με ολική οδήγηση στο MATLAB.

Όπως βλέπουμε, ο αλγόριθμος δέχεται ως ορίσματα τον πίνακα *A* που αντιστοιχεί στους συντελεστές του γραμμικού συστήματος και τον πίνακα *b* ο οποίος είναι ο πίνακας των σταθερών όρων. Ο αλγόριθμος επιστρέφει το διάνυσμα – στήλη *x*, το οποίο αποτελεί την λύση του συστήματος $Ax = b$.

Όπως αναπτύξαμε και πιο πάνω, ο αλγόριθμος ψάχνει σε κάθε βήμα το μέγιστο στοιχείο από αυτό που θα ανανεώσει και εκτελεί τις κατάλληλες αλλαγές σε γραμμές και στήλες για να κάνει αυτό το στοιχείο οδηγό. Στη συνέχεια αφού το κάνει οδηγό στοιχείο, συνεχίζει με τον υπολογισμό των πολλαπλασιαστών και ανανεώνει τις τιμές στις επόμενες γραμμές του πίνακα έτσι ώστε να μηδενιστούν τα κατάλληλα στοιχεία.

Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου φτάσουμε στην προτελευταία γραμμή, όπου εκεί ο αλγόριθμός μας σταματάει την αναζήτηση μέγιστου στοιχείου, μιας και ο πίνακας έχει έρθει σε άνω τριγωνική μορφή. Έπειτα, υπολογίζουμε τις τιμές του διανύσματος *x* από την τελευταία προς την πρώτη. Έτσι καταλήγουμε στην εύρεση της λύσης του συστήματος.

Ένα παράδειγμα χρήσης του αλγορίθμου είναι το εξής:

```
>> A = [-1 2 -1; 2  0 -1 ; 1 7 -3]
A =
 -1   2  -1
  2   0  -1
  1   7  -3
>> b = [0; 1; 5]
b =
  0
  1
  5
>> x = GaussComplete(A,b)
x =
  1
  1
  1
```

Εικόνα 16 Παράδειγμα εκτέλεσης του αλγορίθμου απαλοιφής Gauss με ολική οδήγηση στο MATLAB.

Όπως βλέπουμε, ο αλγόριθμος στην προκειμένη περίπτωση καλείται να λύση ένα γραμμικό σύστημα της μορφής

$$-x_1 + 2x_2 - x_3 = 0$$

$$2x_1 \qquad - x_3 = 1$$

$$x_1 + 7x_2 - 3x_3 = 5$$

Επομένως, το συγκεκριμένο γραμμικό σύστημα έχει ως πίνακα συντελεστών τον πίνακα

$$A = \begin{bmatrix} -1 & 2 & -1 \\ 2 & 0 & -1 \\ 1 & 7 & -3 \end{bmatrix}$$

και πίνακα συντελεστών τον πίνακα

$$b = \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix}$$

Μετά την εκτέλεση του αλγορίθμου μας, επιστρέφεται ως λύση το διάνυσμα

$$x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

το οποίο είναι η σωστή λύση του γραμμικού συστήματος.

Παρατηρούμε πως με την μέθοδο χωρίς οδήγηση, ο αλγόριθμός μας μας έδινε προφανώς λάθος αποτέλεσμα. Ωστόσο η νέα μας μέθοδος κατάφερε να λύσει με επιτυχία τα προβλήματα που προέκυψαν.

2.3.3 Μερική ή ολική οδήγηση;

Το εύλογο ερώτημα το οποίο προκύπτει μετά από την ανάπτυξη των δυο αυτών τροποποιημένων είναι ποια από τις δυο νέες μεθόδους πρέπει να χρησιμοποιούμε για την επίλυση συστημάτων;

Η απάντηση σε αυτό το ερώτημα δεν είναι απόλυτη, μιας και μπορούμε να έχουμε πολλά κριτήρια τα οποία την επηρεάζουν. Ωστόσο, κατά κύριο λόγο χρησιμοποιούμε την μερική οδήγηση, μιας και όταν αναζητά το μέγιστο στοιχείο, δεν ψάχνει όλο τον πίνακα, αλλά μόνο την στήλη στην οποία βρισκόμαστε.

Επομένως, από θέμα πράξεων και μόνο, η μερική οδήγηση είναι πιο εύχρηστη και αποδοτική σε αυτό το κομμάτι και για αυτό την προτιμούμε, ειδικά αν τα συστήματα που καλούμαστε να λύσουμε είναι αρκετά μεγάλα και αποτελούνται από πολλές εξισώσεις και αγνώστους.

2.4 Ο υπολογισμός του Αντιστρόφου Πίνακα A^{-1}

Μια σχετικά εύχρηστη εφαρμογή της επίλυσης γραμμικών συστημάτων, είναι ο υπολογισμός του αντίστροφου πίνακα ενός πίνακα *A*. Πιο συγκεκριμένα, για να βρούμε τον αντίστροφο πίνακα του πίνακα *A*, αρκεί να λύσουμε τα γραμμικά συστήματα

$$Ax_k = e_k$$

όπου το διάνυσμα e_k αντιστοιχεί στην *k*-οστή στήλη του πίνακα I_n, ο οποίος αντιστοιχεί στον μοναδιαίο πίνακα, διάστασης *n*.

2.4.1 Ο αλγόριθμος του υπολογισμού του αντιστρόφου πίνακα A^{-1}

Ο παρακάτω αλγόριθμος παρουσιάζει τον υπολογισμού του Αντίστροφου πίνακα A^{-1}, για τον πίνακα *A*.

1. Για $k = 1,2, \dots, n$ να λυθεί το γραμμικό σύστημα με την απαλοιφή Gauss με Μερική Οδήγηση
$$Ax_k = e_k$$
2. Επέστρεψε ως αντίστροφο τον πίνακα

4

$$A^{-1} = [x_1 \quad x_2 \quad \cdots \quad x_n]$$

2.4.2 Ο αλγόριθμος του υπολογισμού του αντιστρόφου πίνακα A^{-1} στο MATLAB

Παρακάτω, θα παρουσιάσουμε τις εντολές με τις οποίες μπορούμε να υλοποιήσουμε τον αλγόριθμο του υπολογισμού του αντίστροφου Πίνακα A^{-1} με την χρήση του προγραμματιστικού περιβάλλοντος MATLAB.

Έχουμε:

```
function inv_A = inverse_matrix(A)
 % INPUT
 %   A is a nxn matrix_type
 % OUTPUT
 %   inv_A is the inverse matrix of the matrix A

 n = length(A);
 e_base = zeros(n,1);
 x = zeros(n,1);
 inv_A = zeros(n,1);
 helper = inv_A;

 for i = 1:n
  e_base(i,1) = 1;
  if i > 1
    e_base(i-1,1) = 0;
  endif
  x = GaussEPartial(A,e_base);
  helper = [helper x];
 endfor
 inv_A = helper(:, 2:n+1);
```

Εικόνα 17 Ο αλγόριθμος υπολογισμού του αντιστρόφου πίνακα A^{-1} στο MATLAB.

Όπως βλέπουμε, ο αλγόριθμος δέχεται σαν είσοδο τον πίνακα A και κατασκευάζει αρχικά τα διανύσματα e_base και x, όπου το πρώτο θα μας βοηθήσει να κατασκευάσουμε τις στήλες του μοναδιαίου πίνακα και το δεύτερο θα μας βοηθήσει να κατασκευάσουμε τις στήλες του αντιστρέψιμου πίνακα A^{-1}. Επιπλέον κατασκευάζουμε τους πίνακες inv_A και helper, με βάση τους οποίους θα κατασκευαστεί ο αντίστροφος πίνακας.

Στη συνέχεια βλέπουμε πως μέσω μιας επαναληπτικής μεθόδου, κατασκευάζουμε τις αντίστοιχες στήλες του μοναδιαίου πίνακα και ταυτόχρονα λύνουμε τα γραμμικά

συστήματα που προκύπτουν και έτσι θα προκύψουν τα απαραίτητα διανύσματα-στήλες του αντίστροφου πίνακα.

Ένα παράδειγμα χρήσης του αλγορίθμου είναι το εξής:

```
>> A = [-1 2 -1; 1 5 9; 3 5 7]
A =

 -1  2 -1
  1  5  9
  3  5  7

>> Inv_A = inverse_matrix(A)
Inv_A =

 -0.166667  -0.316667   0.383333
  0.333333  -0.066667   0.133333
 -0.166667   0.183333  -0.116667
```

Εικόνα 18 Παράδειγμα εκτέλεσης του αλγορίθμου υπολογισμού του αντιστρόφου πίνακα A^{-1} στο MATLAB.

Όπως βλέπουμε, ως όρισμα στον αλγόριθμό μας δίνουμε τον πίνακα

$$A = \begin{bmatrix} -1 & 2 & -1 \\ 1 & 5 & 9 \\ 3 & 5 & 7 \end{bmatrix}$$

και το αποτέλεσμα που μας επιστρέφει ο αλγόριθμος είναι ο πίνακας

$$A^{-1} = \begin{bmatrix} -0.166667 & -0.316667 & 0.383333 \\ 0.333333 & -0.066667 & 0.133333 \\ -0.166667 & 0.183333 & -0.116667 \end{bmatrix}$$

Παρατηρούμε πως αν εκτελέσουμε τον πολλαπλασιασμό των δυο αυτών πινάκων, θα προκύψει το αποτέλεσμα

$$A^{-1}A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.5 Ο υπολογισμός της ορίζουσας det A

Άλλη μια ενδιαφέρουσα εφαρμογή της απαλοιφής Gauss, είναι ότι μπορούμε να εφαρμόσουμε την συγκεκριμένη μέθοδο για να υπολογίζουμε εύκολα την ορίζουσα ενός πίνακα.

Έστω πως θέλουμε να βρούμε την ορίζουσα ενός πίνακα A ο οποίος είναι ο πίνακας συντελεστών ενός γραμμικού συστήματος

$$Ax = b$$

Είδαμε πως μέσα από την απαλοιφή Gauss, προκύπτει ο πίνακας $A^{(n)}$ ο οποίος προκύπτει ως

$$A^{(n)} = MA^{(1)} = MA$$

Αν θέλουμε να υπολογίσουμε την ορίζουσα τώρα, βάση της τελευταίας σχέσης μας, προκύπτει ότι

$$detA^{(n)} = [\det M]\left[detA^{(1)}\right]$$

Ωστόσο, γνωρίζουμε πως πίνακας M είναι το αποτέλεσμα του γινομένου

$$M = M^{(n-1)} \cdots M^{(2)} M^{(1)}$$

όπου οι πίνακες $M^{(i)},\ i = 1,2,\cdots,n-1$ είναι κάτω τριγωνικοί μοναδιαίοι πίνακες.

Επομένως έχουμε ότι

$$\det M = \prod_{i=1}^{n-1} detM^{(i)} = 1$$

Επομένως προκύπτει ότι

$$detA^{(1)} = detA^{(n)}$$

Ωστόσο ο πίνακας $A^{(n)}$ είναι άνω τριγωνικός, άρα η ορίζουσα του είναι το γινόμενο της διαγώνιου του. Επομένως έχουμε ότι

$$\det A = \det A^{(1)} = detA^{(n)} = a_{11}^{(1)} a_{22}^{(2)} \cdots a_{nn}^{(n)}$$

δηλαδή η ορίζουσα του πίνακα μας είναι το γινόμενο των οδηγών στοιχείων μας.

2.5.1 Ο αλγόριθμος του υπολογισμού της ορίζουσας

Ο παρακάτω αλγόριθμος παρουσιάζει τον υπολογισμό της Ορίζουσας του πίνακα *A*.

1. Να λυθεί το γραμμικό σύστημα

$$Ax = b$$

για ένα τυχαίο διάνυσμα, π.χ. το διάνυσμα όπου όλα τα στοιχεία του ισούνται με 1

2. Από τον πίνακα $A^{(n)}$, πολλαπλασίασε όλα τα διαγώνια στοιχεία του.
3. Επέστρεψε το γινόμενο .

2.5.2 Ο αλγόριθμος του υπολογισμού της ορίζουσας στο MATLAB

Παρακάτω, θα παρουσιάσουμε τις εντολές με τις οποίες μπορούμε να υλοποιήσουμε τον αλγόριθμο του υπολογισμού της Ορίζουσα του Πίνακα A με την χρήση του προγραμματιστικού περιβάλλοντος MATLAB.

Έχουμε αρχικά μια μικρή αλλαγή στον αλγόριθμο της Μερικής Οδήγησης, έτσι ώστε να μπορέσουμε να κατασκευάσουμε πιο εύκολα και με λιγότερες περιττές εντολές την ζητούμενο αλγόριθμο μας. Η αλλαγή αυτή είναι η εξής:

```
function [x,A_n] = GaussEPartial(A,b)
% INPUTS
%   A is a nxn matrix
%   b is a nx1 matrix
% OUTPUT
%   x that is the solution of the Ax = b
%   A_n is the final matrix that the elimination is making

 A_b = [A b]; % Create the augmented matrix
 [m,n] = size(A);
 x = zeros(m,1);

 for r = 1:m-1
  for p = r:n
   if (abs(A_b(r,r))<abs(A_b(p,r)))
    A_b([r p],:) = A_b([p r],:); % Change the rows
   endif
  endfor
  for i = r+1 : n
   mp = - A_b(i,r)/A_b(r,r); % Create multipliers
   for j = r:n+1
    A_b(i,j) = A_b(i,j) + mp*A_b(r,j);
   endfor
  endfor
 endfor
 A_n = A_b(:, 1:n);

 x(m) = A_b(m,n+1)\A_b(m,m);
 for i=m-1:-1:1
  if A_b(i,i)~=0
   sum = 0;
   for j = i+1:n
    sum = sum + A_b(i,j)*x(j);
   endfor
   x(i) = (A_b(i,n+1)-sum)/A_b(i,i); % Calculate the x matrix
  endif
 endfor
```

Εικόνα 19 Τροποποίηση του αλγορίθμου απαλοιφής Gauss με μερική οδήγηση για πιο εύκολο υπολογισμό της ορίζουσας.

Η αλλαγή που κάναμε, ουσιαστικά ήταν να επιστρέφουμε και τον πίνακα $A^{(n)}$ μαζί με την λύση, μιας και ο προηγούμενος αλγόριθμος επέστρεφε μόνο την λύση του συστήματος, αλλά εμείς σε αυτή την περίπτωση θέλουμε να δουλέψουμε με τον πίνακα $A^{(n)}$. Με βάση τώρα αυτή την αλλαγή, μπορούμε εύκολα να υπολογίσουμε την ορίζουσα του πίνακα μας, με βάση τον αλγόριθμο που αναφέραμε προηγουμένως.

Έχουμε:

```
function deter = Determinant(A)
 % INPUT:
 %   A is a nxn matrix
 % OUTPUT:
 %   deter is the determinant of the matrix A

 n = length(A);
 b = ones(n,1);
 [x,A_n] = GaussEPartial(A,b);
 deter = 1;

 for i= 1:n
  deter = deter*A_n(i,i);
 endfor
```

Εικόνα 20 Ο αλγόριθμος υπολογισμού της ορίζουσας στο MATLAB.

Βλέπουμε πως ο αλγόριθμος αρχικά λαμβάνει ως όρισμα τον πίνακα A και στην συνέχεια κατασκευάζει ένα βοηθητικό διάνυσμα b ίσο με

$$b = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

με την βοήθεια του οποίου θα κατασκευάσουμε τον πίνακα από την απαλοιφή Gauss. Αφού βρούμε τον πίνακα $A^{(n)}$, στην συνέχεια υπολογίζουμε το γινόμενο των διαγώνιων στοιχείων του και έτσι προκύπτει η ορίζουσα του πίνακα.

Ένα παράδειγμα εφαρμογής του αλγορίθμου είναι το εξής:

```
>> A = [-1 2 -1; 1 5 9; 3 5 7]
A =

 -1   2  -1
  1   5   9
  3   5   7

>> determinant_A  = Determinant(A)
determinant_A = 60.000
>> determinant_A  == det(A)
ans = 1
```

Εικόνα 21 Παράδειγμα εκτέλεσης του αλγορίθμου της ορίζουσας στο MATLAB.

Όπως βλέπουμε, ως όρισμα στον αλγόριθμό μας δίνουμε τον πίνακα

$$A = \begin{bmatrix} -1 & 2 & -1 \\ 1 & 5 & 9 \\ 3 & 5 & 7 \end{bmatrix}$$

και το αποτέλεσμα που μας επιστρέφει ο αλγόριθμος είναι η τιμή της ορίζουσας η οποία είναι ίση με 60. Στη συνέχεια εκτελούμε έναν έλεγχο για το αν ο αλγόριθμος μας έχει επιστρέψει το σωστό αποτέλεσμα, ελέγχοντας το με την έτοιμη συνάρτηση της MATLAB για τον υπολογισμό της ορίζουσας ενός πίνακα, με τα αποτελέσματα να είναι σωστά.

3. Παραγοντοποίηση Πίνακα

Ένα θέμα το οποίο προκύπτει από την επίλυση γραμμικών συστημάτων είναι το αν υπάρχει ευκολότερος και ταυτόχρονα ασφαλέστερος τρόπος ώστε να μπορέσουμε να τα επιλύσουμε. Αυτή η σκέψη μας οδήγησε στην κατασκευή μιας νέας τακτικής επίλυσης τέτοιων προβλημάτων και αυτή είναι η διάσπαση του πίνακα των συντελεστών σε πίνακες που έχουν πιο εύχρηστη μορφή, όπως για παράδειγμα σε τριγωνικούς πίνακες.

3.1 Η LU παραγοντοποίηση

Υποθέτουμε πως έχουμε να λύσουμε το γραμμικό σύστημα $Ax = b$ και πως με την χρήση της απαλοιφής Gauss, το σύστημα μας μετασχηματίζεται στο τριγωνικό σύστημα

$$MAx = Mb$$

ή αλλιώς στο σύστημα

$$A^{(n)}x = b^{(n)}$$

για το οποίο ισχύει ότι

$$A = M^{-1}A^{(n)}$$

όπου λόγω της σχέσης $M = M^{(n)}M^{(n-1)} \cdots M^{(2)}M^{(1)}$ προκύπτει ότι

$$M^{-1} = \left[M^{(1)}\right]^{-1}\left[M^{(2)}\right]^{-1} \cdots \left[M^{(n-2)}\right]^{-1}\left[M^{(n-1)}\right]^{-1}$$

με

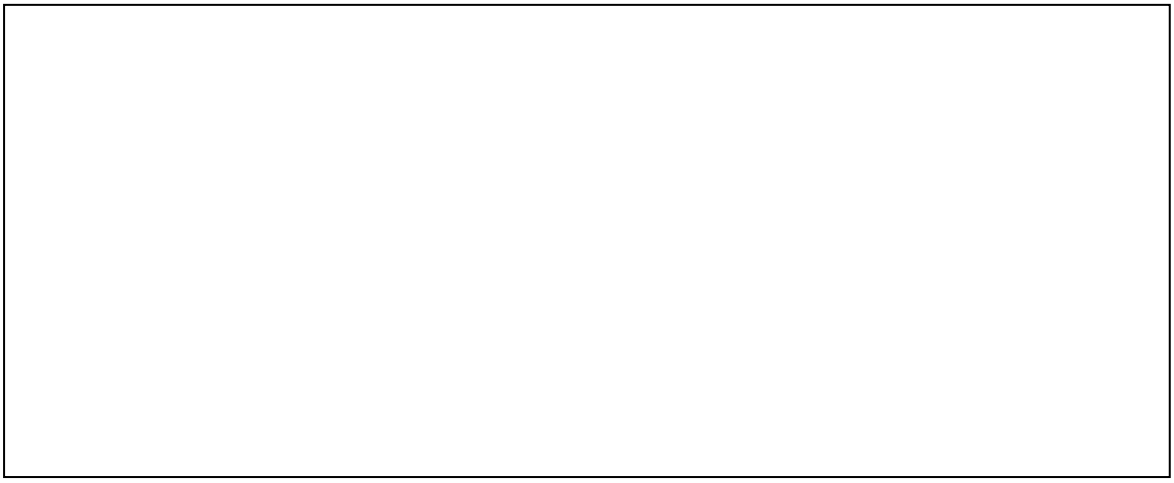
$$\left[M^{(r)}\right]^{-1} = \begin{bmatrix} I_{r-1} & & & & & \\ & 1 & & & & 0 \\ & -m_{r+1,r} & 1 & & & \\ 0 & -m_{r+2,r} & 0 & 1 & & \\ & \vdots & \vdots & & \ddots & \\ & -m_{n,r} & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Επομένως, αν κάνουμε τις πράξεις, βλέπουμε εύκολα πως προκύπτει ότι:

$$M^{-1} = \begin{bmatrix} 1 & & & & & \\ -m_{21} & 1 & & & 0 & \\ -m_{31} & -m_{32} & 1 & & & \\ -m_{41} & -m_{42} & -m_{43} & 1 & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ -m_{n1} & -m_{n2} & -m_{n3} & -m_{21} & \cdots & 1 \end{bmatrix}$$

Αυτό που βλέπουμε είναι πως τελικά ο αντίστροφος πίνακας M^{-1} είναι μοναδιαίος κάτω τριγωνικός και επιπροσθέτως γνωρίζουμε πως και ο πίνακας $A^{(n)}$ είναι ένας κάτω τριγωνικός πίνακας. Επομένως σε αυτό που καταλήγουμε είναι πως τελικά ο αρχικός πίνακας μας μπορεί να γραφτεί ως γινόμενο τριγωνικών πινάκων.

Τον παραπάνω ισχυρισμό, μας τον επαληθεύει το παρακάτω θεώρημα



Το γινόμενο αυτό ονομάζεται παραγοντοποίηση LU και θα χρησιμοποιούμε τον συμβολισμό

$$A = LU$$

όπου ο πίνακας L είναι μοναδιαίος κάτω τριγωνικός και ο πίνακας U είναι κάτω τριγωνικός.

Αυτός ο τρόπος γραφής μας βοηθάει αρκετά στο γεγονός ότι με βάση αυτή, το πιο βασικό πρόβλημα με το οποίο ασχολούμαστε, την επίλυση γραμμικών συστημάτων, μπορεί να λυθεί με πιο λίγες και απλές πράξεις. Ας δούμε πως μπορεί να γίνει αυτό.

Έστω πως θέλουμε να λύσουμε το γραμμικό σύστημα

$$Ax = b$$

Αρχικά διασπάμε το πίνακα A στην παραγοντοποιημένη μορφή,

$$A = LU$$

και έχουμε τελικά ότι το γραμμικό σύστημα γίνεται

$$LUx = b$$

ή ισοδύναμα προκύπτουν τα δυο τριγωνικά συστήματα

$$Ly = b$$

$$Ux = y$$

Προφανώς, και τα δυο αυτά τριγωνικά συστήματα λύνονται με ευκολία, και προκύπτει εύκολα η τελική λύση που ζητάμε. Το μόνο που μας απομένει, είναι να βρούμε έναν πιο αποτελεσματικό τρόπο ώστε να κάνουμε την παραγοντοποίηση μας, χωρίς να χρειάζεται να επικαλούμαστε την απαλοιφή Gauss.

3.1.1 Ο αλγόριθμος της LU παραγοντοποίησης

Ο παρακάτω αλγόριθμος παρουσιάζει τον υπολογισμό της παραγοντοποίησης LU ενός πίνακα *A*.

1. Διάβασε τα στοιχεία $(a_{ij}), 1 \leq i,j \leq n$ του πίνακα A και αρχικοποίησε τους πίνακες L και U, με την προϋπόθεση ότι τα διαγώνια στοιχεία του πίνακα L να είναι όλα ίσα με 1.

2. Να βρεθεί η τιμή για το u_{11} έτσι ώστε να ισχύει η σχέση

$$l_{11}u_{11} = u_{11} = a_{11}$$

   Αν $l_{11}u_{11} = 0$, τότε η παραγωντοποίηση είναι αδύνατη.

3. Για *j=1,2,…,n* να τεθεί

$$u_{1j} = \frac{a_{1j}}{l_{11}}$$

$$l_{j1} = \frac{a_{j1}}{u_{11}}$$

4. Για *r=2,3,…,n-1* να εκτελεστούν τα εξής:

   a. Να βρεθεί η τιμή για το u_{rr} έτσι ώστε να ισχύει η σχέση

$$u_{rr} = a_{rr} - \sum_{j=1}^{r-1} l_{rj}u_{jr}$$

      Αν $l_{\mathrm{rr}}u_{rr} = 0$, τότε η παραγωντοποίηση είναι αδύνατη.

   b. Για *p =r+1,r+2,…n* να τεθεί

$$u_{rp} = a_{rp} - \sum_{j=1}^{r-1} l_{rj}u_{jp}$$

5

$$l_{pr} = \frac{a_{pr} - \sum_{j=1}^{r-1} l_{pj} u_{jr}}{u_{rr}}$$

5. Να επιλεχθεί u_{nn} τέτοιο ώστε να ισχύει ότι

$$u_{nn} = a_{nn} - \sum_{j=1}^{n-1} l_{nj} u_{jn}$$

6. Εκτύπωσε τους πίνακες *L* και *U*.

3.1.2 Ο αλγόριθμος της LU παραγοντοποίησης στο MATLAB

Παρακάτω, θα παρουσιάσουμε τις εντολές με τις οποίες μπορούμε να υλοποιήσουμε τον αλγόριθμο του υπολογισμού της LU παραγοντοποίησης του Πίνακα *A* με την χρήση του προγραμματιστικού περιβάλλοντος MATLAB.

Έχουμε:

```
function [L,U] = LUfactorization(A)
 % INPUT
 %   A is the matrix that we want to factorize
 % OUTPUTS
 %   L is the lower triangular factor
 %   U is the upper triangular factor

 n = length(A);
 L = diag(diag(ones(n)));
 U = diag(diag(ones(n)));
 U(1,1) = A(1,1);
 for j = 2:n
  U(1,j) = A(1,j);
  L(j,1) = A(j,1)/U(1,1);
 endfor
 for r = 2:n-1
  % Calculate U(r,r)
  sum = 0;
  for j=1:r-1
   sum = sum + L(r,j)*U(j,r);
  endfor
  U(r,r) = A(r,r) - sum;
```

```
if U(r,r) == 0
    break;
  endif
  for p = r+1:n
   % Calculate U(r,p)
   sum = 0;
   for j = 1:r-1
    sum = sum + L(r,j)*U(j,p);
   endfor
   U(r,p) = A(r,p) - sum;
   % Calculate L(p,r)
   sum = 0
   for j = 1:r-1
    sum = sum + L(p,j)*U(j,r);
   endfor
   L(p,r) = (A(p,r) - sum)/U(r,r);
  endfor
endfor
% Calculate U(r,r)
sum = 0;
for j = 1:n-1
 sum = sum + L(n,j)*U(j,n);
endfor
U(n,n) = A(n,n) - sum;
```

Εικόνα 22 Ο αλγόριθμος της LU παραγοντοποίησης στο MATLAB.

Όπως βλέπουμε, ο αλγόριθμος δέχεται ως όρισμα τον πίνακα *A* και αρχικά ορίζουμε τους πίνακες στους οποίους θα αποθηκευτούν τα στοιχεία της παραγοντοποίησης του πίνακα μας. Στη συνέχεια εκτελούμε τους κατάλληλους υπολογισμούς τους οποίους στο προηγούμενο κομμάτι δείξαμε πως είναι κατάλληλα να πραγματοποιήσουν την παραγοντοποίηση μας. Δηλαδή να ξεκινήσουμε με τα διαγώνια στοιχεία του κάτω τριγωνικού παράγοντα να είναι ίσα με μονάδες και στην συνέχεια, με βάση την κάθε γραμμή που πίνακα *A* και τον πολλαπλασιασμό πινάκων, να υπολογίσουμε τα στοιχεία των παραγόντων.

Ας δούμε ένα παράδειγμα εφαρμογής του αλγορίθμου:

```
>> A = [2 3 4; 6 6 7; 8 9 10]
A =

   2   3   4
   6   6   7
   8   9  10

>> [L,U] = LUfactorization(A)
L =

   1   0   0
   3   1   0
   4   1   1

U =

   2   3   4
   0  -3  -5
   0   0  -1

>> L*U
ans =

   2   3   4
   6   6   7
   8   9  10
```

Όπως βλέπουμε, θα εφαρμόσουμε την παραγοντοποίηση LU στον πίνακα

$$A = \begin{bmatrix} 2 & 3 & 4 \\ 6 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix}$$

το αποτέλεσμα που επιστρέφει τους πίνακες

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 4 & 1 & 1 \end{bmatrix}, U = \begin{bmatrix} 2 & 3 & 4 \\ 0 & -3 & -5 \\ 0 & 0 & -1 \end{bmatrix}$$

Για να δούμε αν όντως το αποτέλεσμα είναι σωστό, αρκεί να πραγματοποιήσουμε τον πολλαπλασιασμό των πινάκων. Αυτό που προκύπτει είναι το εξής

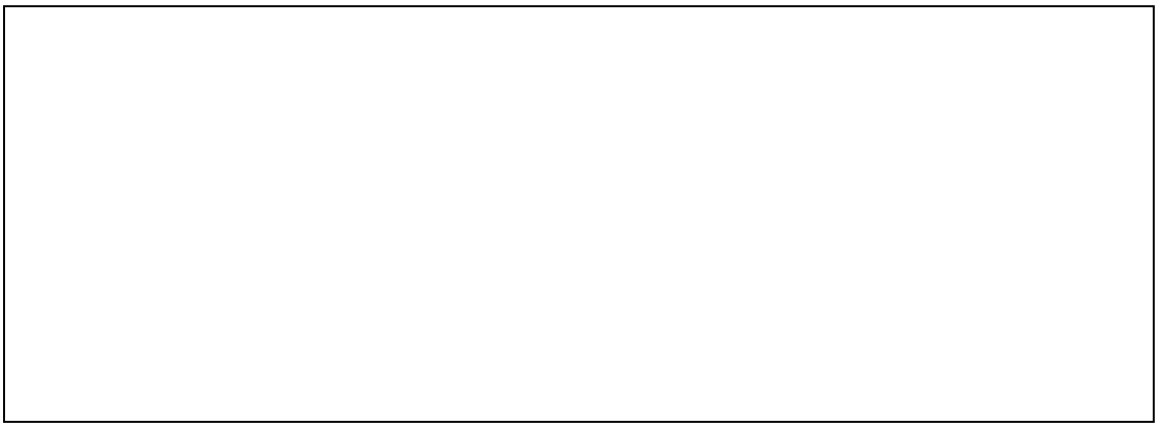
$$L * U = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 4 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 4 \\ 0 & -3 & -5 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 4 \\ 6 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix} = A$$

Επομένως βλέπουμε πως η παραγοντοποίηση μας είναι σωστή.

3.2 Παραλλαγές της LU παραγοντοποίησης – Αλγόριθμος Cholesky

Σε αυτό το σημείο, θα παραθέσουμε μια από τις πιο βασικές παραλλαγές της LU παραγοντοποίησης, την παραγοντοποίηση Cholesky για μη ιδιάζων πίνακες, δηλαδή για αντιστρέψιμους πίνακες. Γενικά, το να ασχολούμαστε με πίνακες οι οποίοι είναι μη ιδιάζων, είναι χρήσιμο διότι τα γραμμικά συστήματα τα οποία έχουν για πίνακα συντελεστών έναν τέτοιο πίνακα, έχουν μοναδική λύση.

Για να δούμε ωστόσο πιο αναλυτικά την παραγοντοποίηση Cholesky, θα πρέπει να την θεμελιώσουμε μαθηματικά με τα εξής τρία Θεωρήματα



Αυτό που μας λέει το παραπάνω θεώρημα είναι ουσιαστικά το εξής:

Ένας πίνακας A έχει μια παραγοντοποίηση LU όπου ισχύει ότι $A = LU$. Αν θεωρήσουμε έναν τον διαγώνιο, μη ιδιάζων πίνακα D, τότε αν θέσουμε $L' = LD$ και $U' = D^{-1}U$ τότε προκύπτει η νέα παραγοντοποίηση

$$A = LU = LDD^{-1}U = L'U'$$

Δηλαδή αυτό θα ονομάζεται μια LDU παραγοντοποίηση, και ισχύει λόγω του Θεωρήματος.

Για το επόμενο θεώρημα, θα χρειαστεί να αναφέρουμε τι είναι ένας θετικά ορισμένος πίνακας.

Με βάση αυτό, προκύπτει το εξής Θεώρημα:

Θεώρημα:

Αν ένας πίνακας *A* διάστασης *n x n* είναι θετικά ορισμένος, τότε ο πίνακας *A* είναι και μη ιδιάζων.

Αυτό που μας εξασφαλίζει το παραπάνω Θεώρημα είναι ουσιαστικά το πως οι πίνακες τους οποίους μελετάμε, εκτός από αντιστρέψιμοι, είναι και θετικά ορισμένοι, πράγμα που σημαίνει πως υπάρχει μια μεγαλύτερη ευελιξία στις πράξεις και έτσι θα αποφύγουμε καταστάσεις όπου θα προκύψουν αρκετά γινόμενα πινάκων και αυτό δεν θα είναι ίσα με μηδέν.

Το τελευταίο Θεώρημα που χρειαζόμαστε ως υπόβαθρο είναι το εξής:

Θεώρημα:

Ένας κύριος υποπίνακας ενός θετικά ορισμένου πίνακα είναι θετικά ορισμένος πίνακας.

Τώρα με βάση τα παραπάνω Θεωρήματα, προκύπτει η ισχύς της παραγοντοποίησης Cholesky.

Αυτό που μας λέει η παραγοντοποίηση Cholesky είναι πως μπορούμε να έχουμε μια παραγοντοποίηση για τον θετικά ορισμένο πίνακα *A*, τέτοια ώστε να προκύπτει από έναν μόνο πίνακα. Αυτό μας συμφέρει αρκετά από άποψη υπολογιστικής ισχύς.

3.2.1 Ο αλγόριθμος της παραγοντοποίησης Cholesky

Ο παρακάτω αλγόριθμος παρουσιάζει τον τρόπο υπολογισμού της παραγοντοποίησης Cholesky ενός θετικά ορισμένου πίνακα *A*.

1. Διάβασε τα στοιχεία του πίνακα $A = (a_{ij})$ καθώς και την τάξη *n* του πίνακα.
2. Να τεθεί

$$l_{11} = \sqrt{a_{11}}$$

3. Για $j = 2,3, \dots, n$ να τεθεί

$$l_{j1} = \frac{a_{j1}}{l_{11}}$$

4. Για $r = 2,3, \dots, n-1$ να εκτελεστούν τα παρακάτω
   a. Να τεθεί

$$l_{rr} = \left[a_{rr} - \sum_{j=1}^{r-1} l_{rj}^2 \right]^{1/2}$$

   b. Για $p = r+1, r+2, \dots, n$ να υπολογιστούν

$$l_{pr} = \frac{1}{l_{rr}} \left[a_{pr} - \sum_{j=1}^{r-1} l_{pj} l_{rj} \right]$$

5. Να τεθεί

$$l_{nn} = \left[a_{nn} - \sum_{j=1}^{n-1} l_{nj}^2 \right]^{1/2}$$

6. Να εκτυπωθούν τα υπολογισμένα στοιχεία.

Όπως παρατηρούμε, στον παραπάνω αλγόριθμο προκύπτουν υπολογισμοί ριζών. Σε αυτό το κομμάτι πρέπει να είμαστε λίγό προσεκτικοί, αν και το γεγονός ότι ο πίνακας είναι θετικά ορισμένος, καλύπτει στην πληθώρα τον περιπτώσεων το ασφάλεια εκτέλεσης του αλγορίθμου μας.

3.2.2 Ο αλγόριθμος της παραγοντοποίησης Cholesky στο MATLAB

Παρακάτω, θα παρουσιάσουμε τις εντολές με τις οποίες μπορούμε να υλοποιήσουμε τον αλγόριθμο του υπολογισμού της παραγοντοποίησης Cholesky του Πίνακα *A* με την χρήση του προγραμματιστικού περιβάλλοντος MATLAB.

Έχουμε :

```
function L = CholeskiFactor(A)
 % INPUT
 %   A is a nxn matrix
 % OUTPUT
 %   L is the Choleski factor

 n = length(A);
 L = zeros(n);

 L(1,1) = sqrt(A(1,1));
 for j=2:n
  L(j,1) = A(j,1)/L(1,1);
 endfor
 for r=2:n-1
  % Calculate L(r,r)
  sum = 0;
  for j=1:r-1
   sum = sum + L(r,j)**2;
  endfor
  L(r,r) = sqrt(A(r,r)-sum);
  %Calculate L(p,r)
  for p = r+1:n
   sum = 0;
   for j=1:r-1
    sum = sum + L(p,j)*L(r,j);
   endfor
   L(p,r) = (1/L(r,r))*(A(p,r) - sum);
  endfor
 endfor
 % Calculate L(n,n)
 sum = 0;
 for j=1:n-1
  sum = sum + L(n,j)**2;
 endfor
 L(n,n) = sqrt(A(n,n)-sum);
```

Εικόνα 24 Ο αλγόριθμος παραγοντοποίησης Cholesky στο MATLAB.

Όπως βλέπουμε, ο αλγόριθμος δέχεται σαν όρισμα τον πίνακα A και στην συνέχεια εκτελεί τα κατάλληλα βήματα και υπολογισμού έτσι ώστε να δημιουργηθεί ο παράγοντας L που ζητάμε.

Ας δούμε ένα παράδειγμα εφαρμογής του αλγορίθμου μας:

```
>> A = [4 2 -1; 2 4 1; -1 1 4]
A =

  4   2  -1
  2   4   1
 -1   1   4

>> L = CholeskiFactor(A)
sum = 0
L =

  2.0000       0         0
  1.0000   1.7321        0
 -0.5000   0.8660   1.7321

>> L*L.'
ans =

  4.0000   2.0000  -1.0000
  2.0000   4.0000   1.0000
 -1.0000   1.0000   4.0000
```

Εικόνα 25 Παράδειγμα εκτέλεσης του αλγορίθμου παραγοντοποίησης Cholesky.

Βλέπουμε πως ο αλγόριθμός μας δέχεται σαν είσοδο τον πίνακα

$$A = \begin{bmatrix} 4 & 2 & -1 \\ 2 & 4 & 1 \\ -1 & 1 & 4 \end{bmatrix}$$

και επιστρέφει ως αποτέλεσμα τον πίνακα

$$L = \begin{bmatrix} 2 & 0 & 0 \\ 1 & 1.7321 & 0 \\ -0.5 & 0.8660 & 1.7321 \end{bmatrix}$$

Για να επαληθεύσουμε αν το αποτέλεσμα είναι σωστό, εκτελούμε τον πολλαπλασιασμό

$$LL^T = \begin{bmatrix} 2 & 0 & 0 \\ 1 & 1.7321 & 0 \\ -0.5 & 0.8660 & 1.7321 \end{bmatrix} \begin{bmatrix} 2 & 1 & -0.5 \\ 0 & 1.7321 & 0.8600 \\ 0 & 0 & 1.7321 \end{bmatrix} = \begin{bmatrix} 4 & 2 & -1 \\ 2 & 4 & 1 \\ -1 & 1 & 4 \end{bmatrix} = A$$

Επομένως το αποτέλεσμά μας είναι σωστό.

4. Παρεμβολή

Ένα από τα βασικότερα προβλήματα που προκύπτουν γενικά σε διάφορα προβλήματα είναι να γίνει η πρόβλεψη τιμών μιας συνάρτησης, δεδομένου ότι η συνάρτηση αυτή παίρνει κάποιες τιμές σε συγκεκριμένα σημεία τα οποία γνωρίζουμε, αλλά ωστόσο η ίδια η συνάρτηση μας είναι άγνωστη.

Μια βασική ιδέα είναι πως για τα *n+1* σημεία x_i στα οποία η συνάρτησή μας *f(x)* λαμβάνει τιμές, να κατασκευάσουμε ένα πολυώνυμο $p_n(x)$, βαθμού το πολύ n, τέτοιο ώστε οι τιμές του στα σημεία x_i να συμπίπτουν με αυτές της συνάρτησης *f(x)*, δηλαδή

$$p_n(x_i) = f(x_i), i = 0,1, \dots, n$$

Το πολυώνυμο αυτό, το οποίο θα το ονομάζουμε πολυώνυμο παρεμβολής, είναι μοναδικό με αυτή την ιδιότητα και με βάση αυτό θα προσπαθούμε γενικά να προσεγγίσουμε την συνάρτηση *f(x)*.

4.1 Η παρεμβολή Lagrange

Θα προσπαθήσουμε να κατασκευάσουμε το πολυώνυμο της παρεμβολής με βάση την λεγόμενη Μέθοδο Lagrange. Η συγκεκριμένη μέθοδος αρχικά υποθέτει πως το πολυώνυμο το οποίο θα κατασκευάσουμε θα έχει την ιδιότητα

$$p_n(x_j) = f_j, j = 0,1, \dots, n$$

όπου f_j είναι η τιμή της συνάρτησης $f(x)$ στο σημείο x_j δηλαδή με την υπόθεση πως το πολυώνυμο παρεμβολής θα λαμβάνει τις τιμές της συνάρτησης στα δεδομένα σημεία που διαθέτουμε. Τότε το πολυώνυμο μας μπορεί να εκφραστεί σαν το εξής άθροισμα

$$p_n(x) = \sum_{i=0}^{n} L_i(x)\, f_i$$

Όπου έχουμε ότι οι ποσότητες $L_i(x), i = 0,1, \dots, n$ είναι πολυώνυμα βαθμού μικρότερου ή ίσου του *n* τα οποία καλούνται συντελεστές του Lagrange και ικανοποιούν τις σχέσεις

$$L_i(x_j) = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases} \quad 0 \leq i, j \leq n$$

Το $p_n(x)$ είναι το ζητούμενο πολυώνυμο παρεμβολής διότι έχουμε ότι

$$p_n(x_j) = \sum_{i=0}^{n} L_i(x_j)\, f_i = L_i(x_j) f_i = f_i,\ \ j = 0,1,\dots,n$$

Λόγω του ότι $L_i(x_j) = 0, i \neq j$ έχουμε ότι τα $x_j, j = 0,1,\dots,n\ i \neq j$ είναι οι ρίζες του $L_i(x)$, άρα

$$L_i(x) = A_i(x - x_0)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_n)$$

Ο προσδιορισμός της ποσότητας A_i γίνεται με την βοήθεια της πρώτης δίκλασης σχέσης και έχουμε ότι

$$L_i(x_i) = 1 = A_i(x_i - x_0)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)$$

Επομένως έχουμε ότι

$$A_i = \frac{1}{(x_i - x_0)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)}$$

Αν τώρα αντικαταστήσουμε αυτή την ποσότητα στην σχέση μας έχουμε ότι

$$L_i(x) = \frac{(x - x_0)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_n)}{(x_i - x_0)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)}$$

ή αλλιώς

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_i}{x_i - x_j}, \qquad i = 0,1,..,n$$

Επομένως για την παρεμβολή του Lagrange έχουμε ότι προκύπτει

$$f(x) = p_n(x) + R_{n+1}(x)$$

όπου η ποσότητα $R_{n+1}(x)$ είναι το σφάλμα μας, και έτσι ισχύει ότι

$$f(x) \cong p_n(x)$$

4.1.1 Ο αλγόριθμος της παρεμβολής Lagrange

Παρακάτω θα υπολογίσουμε το πολυώνυμο παρεμβολής του Lagrange για τις n τιμές μιας συνάρτησης f(x).

1. Διάβασε τα $x_i, i = 0,1,\dots,n$ καθώς και τις τιμές της συνάρτησης σε αυτά τα σημεία $f(x_i), i = 0,1,\dots,n$

6

2. Για $i = 0,1, \dots, n$ να υπολοιστούν οι συντελεστές

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_i}{x_i - x_j}, \qquad i = 0,1,..,n$$

3. Να υπολογιστεί το πολυώνυμο

$$p_n(x) = \sum_{i=0}^{n} L_i(x)\, f_i$$

4. Να εκτυπωθεί το πολυώνυμο.

4.1.2 Ο αλγόριθμος της παρεμβολής Lagrange στο MATLAB

Παρακάτω, θα παρουσιάσουμε τις εντολές με τις οποίες μπορούμε να υλοποιήσουμε τον αλγόριθμο του υπολογισμού της παρεμβολής Lagrange για μια συνάρτηση *f(x)* με την χρήση του προγραμματιστικού περιβάλλοντος MATLAB.

Έχουμε:

```
function F = LagrangePoly(x,f)
 % INPUTS
 %   x are the points where the function f(x) is taking her values
 %   f are the values of the function f(x) at the points x
 % OUTPUTS
 %   F that has the coefficients of the polynomial

 n = length(x);
 L = zeros(n,n);
 sum = zeros(1,n);
 L = zeros(n,n);

 for i = 1:n
  gin = 1;
  for j =1:n
   if i~=j
    gin = conv(gin, poly(x(j)))/(x(i)-x(j));
   endif
  endfor
  L(i,:) = gin*f(i);
 endfor

 for j = 1:n
  sum_j = 0;
  for i = 1:n
   sum_j = sum_j + L(i,j);
  endfor
  sum(1,j) = sum_j;
 endfor

 % Calculate the coefficients of the poolynomial
 F = flip(sum);
 % Print the polynomial
 for k = n:-1:2
  fprintf('+(%.2fx^%d)', F(k),k-1)
 endfor
 fprintf('+(%2f)\n', F(1))
```

Εικόνα 26 Ο αλγόριθμος υπολογισμού της παρεμβολής Lagrange στο MATLAB.

Όπως βλέπουμε, ο αλγόριθμος δέχεται ως ορίσματα τα διανύσματα x , το οποίο αποτελείται από τα σημεία x_i , $i = 0,1, \dots, n-1$ στα οποία η συνάρτηση $f(x)$ λαμβάνει τιμές και το διάνυσμα f, το οποίο αποτελείται από τις τιμές της συνάρτησης $f(x_i), i = 0,1, \dots, n-1$ στα σημεία του αρχικού μας διανύσματος. Ως αποτέλεσμα, ο αλγόριθμος επιστρέφει το διάνυσμα F το οποίο αποτελείται από τους συντελεστές κάθε μεταβλητή x^r , $r = 0,1, \dots, n-1$ που αποτελούν το πολυώνυμο παρεμβολής μας.

Εδώ ο αλγόριθμος κατασκευάστηκε λίγο διαφορετικά απ' ό,τι συνήθως, για λόγους υπολογιστικής ευκολίας και πλήρης αξιοποίησης των δυνατοτήτων του προγράμματος MATLAB.

Αρχικά, ο αλγόριθμος διαβάζει τα στοιχεία και κατασκευάζει με βάση αυτά το βοηθητικό διάνυσμα sum, το οποίο θα μας βοηθήσει να κατασκευάσουμε το ζητούμενο διάνυσμα συντελεστών *F*.

Στη συνέχεια, με την χρήση του τύπου

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_i}{x_i - x_j}, \qquad i = 0,1,..,n$$

κατασκευάσαμε μια επαναληπτική μέθοδος, η οποία μας δημιουργεί τους συντελεστές Lagrange από τους οποίους αποτελείται το πολυώνυμο παρεμβολής.

Έπειτα με μια επαναληπτική μέθοδο, κάνουμε την παραγοντοποίηση που χρειάζεται και έτσι κατασκευάζεται το διάνυσμα των συντελεστών του πολυωνύμου, το οποίο είναι στην μορφή

$$F = [a_0 \quad a_1 \quad \cdots \quad a_{n-1}]$$

μιας και το πολυώνυμο παρεμβολής θα είναι το

$$p_n(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0$$

Στη συνέχεια απλά εκτυπώνουμε αυτό το πολυώνυμο.

Ας δούμε ένα παράδειγμα εφαρμογής του αλγορίθμου μας.

```
>> x = [2;3;5]
x =

  2
  3
  5

>> f = [5;7;8]
f =

  5
  7
  8

>> p = LagrangePoly(x,f)
+(-0.50x^2)+(4.50x^1)+(-2.000000)
p =

 -2.0000   4.5000  -0.5000
```

Εικόνα 27 Παράδειγμα εκτέλεσης του αλγορίθμου υπολογισμού της παρεμβολής Lagrange στο MATLAB.

Τα αρχικά μας δεδομένα βλέπουμε πως είναι τα εξής στοιχεία:

Πίνακας 1 Αρχικά δεδομένα για την παρεμβολή Lagrange

	i=0	i=1	i=2
x	2	3	5
f(x)	5	7	8

Βλέπουμε πως ο αλγόριθμος μας έχει εκτυπώσει το πολυώνυμο:

$$p(x) = -0.5x^2 + 4.5x - 2$$

και στη συνέχεια βλέπουμε πως μας επιστρέφει το διάνυσμα των συντελεστών

$$F = [-2 \quad 4.5 \quad -0.5]$$

Στη συνέχεια με βάση αυτό το διάνυσμα και μερικές απλές εντολές του MATLAB, μπορούμε να κατασκευάσουμε το συγκεκριμένο πολυώνυμο και να λαμβάνουμε τιμές για διάφορες τιμές που μας ενδιαφέρουν.

Συμπεράσματα

Είδαμε πως με βασικές μαθηματικές αρχές, μπορούμε να κατασκευάσουμε λύσεις και αλγορίθμους για μια μεγάλη πληθώρα προβλημάτων, των οποίων η χρήση υπολογιστή για την λύση τους αρχικά φαίνεται αδύνατη.

Την υλοποίηση αυτών των αλγορίθμων καταφέραμε με ευκολία να την πραγματοποιήσουμε με την γλώσσα προγραμματισμού MATLAB μιας και είναι μια γλώσσα που βασίζεται αρκετά στις πράξεις πινάκων και τις εκτελεί γρήγορα και με μεγάλη ευκολία.

Βιβλιογραφία

1] Γ. Δ. Ακρίβης και Β. Α. Δουγαλής, Εισαγωγή στην αριθμητική ανάλυση, Ηράκλειο: Πανεπιστημιακές εκδόσεις Κρήτης, 1998.

2] Ν. Μυρσίλης, Αριθμητική ανάλυση, μία αλγοριθμική προσέγγιση, Αθήνα: Εκδόσεις ΕΚΠΑ, 2009.