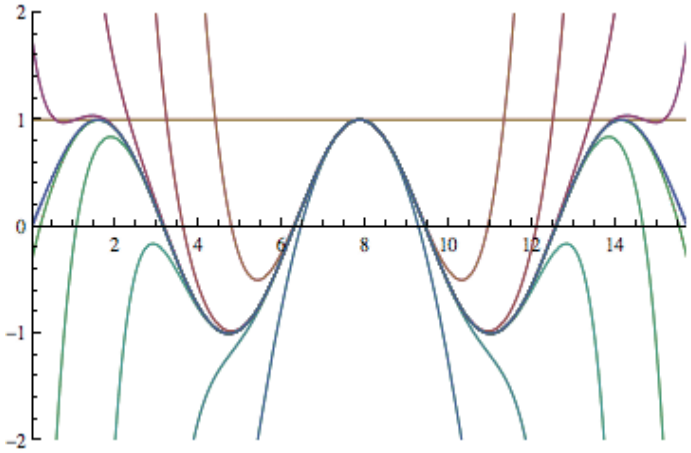
ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Δημιουργία Αριθμομηχανής στα πρότυπα του Scientific Calculator των Windows με την υλοποίηση των Μαθηματικών συναρτήσεων με μεθόδους Αριθμητικής Ανάλυσης



Του φοιτητή

ΘΕΟΔΩΡΑΚΗ ΝΙΚΟΛΑΟΥ

Αρ. Μητρώου: 05/2978

Επιβλέπων καθηγητής

ΓΟΥΛΙΑΝΑΣ ΚΩΣΤΑΝΤΙΝΟΣ

Θεσσαλονίκη 2012

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Κώδικα

Κεφάλαιο 1

1.1 Εισαγωγή

Στην εργασία αυτή θα υλοποιήσουμε σε Java μια επιστημονική αριθμομηχανή. Ο σκοπός της παρούσας εργασίας είναι διττός. Αφενός, να μελετήσουμε τις επιμέρους λειτουργίες της γλώσσας Java για την αναπαράσταση τόσο των ακέραιων αριθμών όσο και των αριθμών κινητής υποδιαστολής, όπως επίσης και να γνωρίσουμε τα ιδιαίτερα χαρακτηριστικά των σειρών Taylor και των αριθμητικών προσεγγίσεων των βασικών συναρτήσεων. Επιπρόσθετα, η εργασία αυτή μας επιτρέπει να εξοικειωθούμε με τις μεθόδους βελτιστοποίησης χρονοβόρων προγραμματιστικών διεργασιών.

Η εκλογή της γλώσσας Java έγινε με γνώμονα την ευρεία αποδοχή της από την προγραμματιστική κοινότητα καθώς και εκείνα τα ιδιαίτερα χαρακτηριστικά της, τα οποία επιτρέπουν την εκτέλεση των ίδιων εφαρμογών σε όλες ανεξαιρέτως τις πλατφόρμες και σε όλα τα λειτουργικά συστήματα τα οποία είναι εξοπλισμένα με την εικονική μηχανή της Java.

1.2 Abstract

In this Project, we will implement in Java a scientific calculator the purpose of this paper is twofold. First, we study the specific features of the Java language representation for both, integers and floating point numbers, as well as getting to know the characteristics of the Taylor series and numerical approaches of basic functions. Additionally, this work allows us to gain familiarity with programming optimization methods cumbersome processes.

The election of the Java language was driven wide acceptance by the programming community, and particularly those characteristics that allow the execution of the same application to every single platforms and all operating systems that are equipped with virtual machine of Java

Κεφάλαιο 2

Ανάλυση απαιτήσεων και σκοπιμότητας

2.1 Ανάλυση απαιτήσεων

Στην φάση της ανάλυσης των απαιτήσεων, κύριο μέλημα μας είναι ο προσδιορισμός των αναγκών του χρήστη, είτε αυτός πρόκειται για έναν απλό χρήστη, ο όποιος επιθυμεί να εκτελέσει άπλες αριθμητικές πράξεις, είτε για κάποιον ο όποιος έχει ως γνωστικό αντικείμενο τα μαθηματικά και ειδικότερα την αριθμητική ανάλυση, είτε για κάποιον του όποιου το γνωστικό αντικείμενο περιστρέφεται γύρω από τα υπολογιστικά συστήματα. Το συγκεκριμένο πληροφοριακό σύστημα οφείλει να είναι φιλικό προς τον εκάστοτε χρήστη, όποιος και εάν είναι αυτός, και κυρίως να είναι όσο το δυνατόν πιο εύχρηστο και ευέλικτο, έτσι ώστε να μπορέσει να καλύψει όλες τις ανάγκες του εκάστοτε χρήστη όσο σύνθετες και αν είναι αυτές.

2.2 Ανάλυση σκοπιμότητας

Η ανάλυση σκοπιμότητας έχει ως στόχο να ελέγξει εάν είναι εφικτή η υλοποίηση του συγκεκριμένου project. Η ανάλυση αυτή περιέχει τρεις φάσεις οι οποίες εξετάζουν, ελέγχουν και προσδιορίζουν εάν το συγκεκριμένο έργο θα γίνει αποδεκτό από τους χρηστές, εάν η συγκεκριμένη λύση που παρέχουμε είναι η καλύτερη δυνατή από οικονομικής πλευράς και τέλος προσδιορίζει εάν η πρόταση για την υλοποίηση έχει πρακτική εφαρμογή.

Το συγκεκριμένο project θα καταστεί χρήσιμο σε μεγάλο βαθμό σους χρηστές, καθώς τους παρέχει την δυνατότητα να επιλύουν από άπλες ως σύνθετες μαθηματικές εκφράσεις, ταχύτατα, διαθέτοντας παράλληλα μια ευρεία γκάμα πολλαπλών επιλογών, όντας παράλληλα ιδιαίτερα απλό στην χρήση του. Όσον αφορά το κόστος του, η λύση η όποια προτείνεται είναι η ευνοϊκότερη καθώς το πρόγραμμα που χρησιμοποιήθηκε για την ανάπτυξη της εφαρμογής διατίθεται δωρεάν.

2.3 Η επιλογή της γλώσσας προγραμματισμού

Η υλοποίηση γίνεται σε γλώσσα Java. Ένα από τα βασικά πλεονεκτήματα της Java, έναντι της συντριπτικής πλειοψηφίας των γλωσσών προγραμματισμού, αποτελεί η ανεξαρτησία της από το λειτουργικό σύστημα και τη πλατφόρμα: Πιο συγκεκριμένα, τα προγράμματα που δημιουργούνται με χρήστη της γλώσσας Java εκτελούνται με τον ίδιο ακριβώς τρόπο σε όλα τα λειτουργικά συστήματα Windows, Linux, Unix και Macintosh, χωρίς επανα-μεταγλώττιση του πηγαίου κώδικα, ούτε και τη τροποποίηση του,

που είναι απαραίτητη σε πολλές άλλες γλώσσες προγραμματισμού (όπως η C/C++).

Υπάρχουν πληροφορίες ότι σύντομα πολλές κονσόλες παιχνιδιών θα υποστηρίξουν στο μέλλον τη γλώσσα Java. Για να επιτευχθεί η ανεξαρτησία αυτή απαιτείται η ύπαρξη μιας συγκεκριμένης μεθοδολογίας η οποία θα επιτρέπει στα προγράμματα τα οποία είναι γραμμένα σε γλώσσα Java να εκτελούνται από κάθε υπολογιστή ανεξαρτήτως επεξεργαστή (Intel x86, IBM, Sun SPARC, Motorola) και λειτουργικού συστήματος (Windows, Unix, Linux, BSD, MacOS). Η βασική αιτία είναι ότι κάθε επεξεργαστής κατανοεί διαφορετικό κώδικα μηχανής. Έτσι, για παράδειγμα, ο συμβολικός κώδικας (assembly) που εκτελείται στα λειτουργικά συστήματα Windows είναι εντελώς διαφορετικός από τον αντίστοιχο ενός Macintosh. Η μεθοδολογία που έχει ακολουθηθεί βασίζεται στην ανάπτυξη μιας κατάλληλης Εικονικής Μηχανής (Virtual Machine ή VM) η οποία εκτελεί τα εκτελέσιμα αρχεία που παράγει η Java.

Μετά την ολοκλήρωση της συγγραφής ενός προγράμματος σε Java, ακολουθεί η μεταγλώττιση του, η οποία οδηγεί στη δημιουργία του bytecode. Ο bytecode είναι η “εκτελέσιμη” μορφή του κώδικα Java. Όταν πρόκειται να εκτελεστεί η εφαρμογή σε ένα μηχάνημα, η Java Virtual Machine (η οποία είναι απαραίτητο να είναι εγκατεστημένη σε αυτό) αναλαμβάνει να διαβάσει τον bytecode τον οποίο στη συνέχεια τον μετατρέπει σε γλώσσα μηχανής η οποία υποστηρίζεται από το λειτουργικό σύστημα και τον επεξεργαστή, έτσι ώστε να εκτελεστεί. Η παραπάνω αρχή λειτουργίας είναι η πρώτη που υιοθέτησε η εικονική μηχανή της Java και αποκαλείται σήμερα «Παραδοσιακή Εικονική Μηχανή» (Virtual Machine). Οι πιο σύγχρονες εκδόσεις της εικονικής μηχανής έχουν επιπλέον δυνατότητες ώστε να μεταγλωττίζουν εκ

των προτέρων τμήματα bytecode απευθείας σε κώδικα μηχανής (native code), με αποτέλεσμα να βελτιώνεται η ταχύτητα. Χωρίς την ιδιότητα αυτή, δεν θα ήταν δυνατή η εκτέλεση οποιουδήποτε λογισμικού γραμμένου σε Java. Αξίζει να σημειωθεί ότι η εικονική μηχανή JVM είναι λογισμικό που εξαρτάται από τη πλατφόρμα, δηλαδή για κάθε είδος λειτουργικού συστήματος και αρχιτεκτονικής του επεξεργαστή, υπάρχει διαφορετική έκδοση του. Έτσι, υπάρχουν διαφορετικές JVM για Windows, Linux, Unix, Macintosh, κινητά τηλέφωνα, παιχνιδομηχανές κλπ.

Όλες οι ενέργειες των χρηστών και των προγραμματιστών στη γλώσσα Java, πραγματοποιείται μέσω της εικονικής μηχανής. Η συγκεκριμένη αντιμετώπιση βοηθάει στην προάσπιση της ασφάλειας από πλευράς του συστήματος, καθώς η εικονική μηχανή γίνεται ο κύριος μεσάζοντας κατά την επικοινωνία χρήστη - υπολογιστή. Ο προγραμματιστής είναι αδύνατον να γράψει κώδικα ο οποίος θα έχει καταστροφικά αποτελέσματα για τον υπολογιστή, γιατί η εικονική μηχανή θα τον ανιχνεύσει και δε θα επιτρέψει την εκτέλεσή του. Από την άλλη πλευρά, ούτε ο χρήστης μπορεί να κατεβάσει κακόβουλο λογισμικό από το διαδίκτυο ή από άλλη πηγή και να εκτελέσει το κώδικα. Αυτό είναι ιδιαίτερα χρήσιμο για μεγάλα, κατανεμημένα συστήματα όπου πολλοί χρήστες χρησιμοποιούν το ίδιο πρόγραμμα.

Έτσι, οι βασικοί λόγοι που μας οδήγησαν στην υιοθέτηση της Java για την ανάπτυξη της εφαρμογής είναι οι εξής:

- Είναι εφικτό να εκτελέσουμε το πρόγραμμά μας σε κάθε λειτουργικό σύστημα και με οποιονδήποτε επεξεργαστή.

- Η γλώσσα Java προσφέρει εγγενώς εργαλεία για την ανάπτυξη γραφικού περιβάλλοντος. Αυτό προσφέρει σημαντική ευκολία στην ανάπτυξη του λογισμικού.
- Η γλώσσα Java, στις σύγχρονες εκδόσεις της, δεν στερείται ταχύτητας, όπως παλαιότερα, σε σύγκριση με άλλες γλώσσες προγραμματισμού (C/C++)

2.4 Επιλογή της πλατφόρμας

Για λόγους διευκόλυνσης της ανάπτυξης, έχει επιλέγει η πλατφόρμα Netbeans για την υλοποίηση της εφαρμογής. Η πλατφόρμα Netbeans αποτελεί ένα ολοκληρωμένο περιβάλλον εργασίας Java, με ενσωματωμένο debugger (“διαδικασία αποσφαλμάτωσης του κώδικα”) καθώς και ενσωματωμένη σχεδιαστική διαδικασία για την διευκόλυνση της δημιουργίας γραφικών περιβαλλόντων. Με χρήση της συγκεκριμένης, δωρεάν πλατφόρμας, διευκολύνεται η διαδικασία συγγραφής της εφαρμογής, ενώ ταυτόχρονα, απλοποιείται και η διαδικασία της αποσφαλμάτωσης του κώδικα και εντοπισμού σημείων καθυστέρησης. Το NetBeans είναι ένα open-source ολοκληρωμένο περιβάλλον ανάπτυξης σε γλώσσα Java και υποστηρίζει την ανάπτυξη όλων των διαφορετικών τύπων εφαρμογών της (όπως για παράδειγμα Java SE συμπεριλαμβανομένων JavaFX, Java ME , web , EJB και κινητών εφαρμογών). Η πλατφόρμα διαθέτει πληθώρα χαρακτηριστικών, όπως προχωρημένο σύστημα διαχείρισης project, υποστήριξη Maven (Maven είναι μια πλατφόρμα για τη διαχείριση πολύ μεγάλων έργων λογισμικού γραμμένων σε Java), δυνατότητες refactoring, καθώς και υποστήριξη για όλα τα γνωστά προγράμματα αποθετηρίων και

ενημέρωσης πηγαίου κώδικα (Υπάρχει υποστήριξη για CVS , Subversion -SVN , Mercurial και ClearCase).

Ένα άλλο σημαντικό χαρακτηριστικό του Netbeans είναι η επεκτασιμότητα του. Όλες οι λειτουργίες του παρέχονται από ανεξάρτητες οντότητες. Κάθε οντότητα παρέχει μια σαφώς καθορισμένη λειτουργία, όπως η βοήθεια για τη γλώσσα Java, η επεξεργασία ή η υποστήριξη για το CVS σύστημα εκδόσεων και SVN. Το NetBeans περιέχει όλες εκείνες τις εξ' ορισμού οντότητες που απαιτούνται για την ανάπτυξη εφαρμογών Java, επιτρέποντας στο χρήστη να δημιουργήσει νέες εφαρμογές άμεσα. Οι οντότητες επιτρέπουν επίσης, στο NetBeans να επεκταθεί και να προσθέσει νέα χαρακτηριστικά, όπως υποστήριξη για άλλες γλώσσες προγραμματισμού.

Κεφάλαιο 3

Οι συναρτήσεις της επιστημονικής αριθμομηχανής

3.1 Εισαγωγή

Οι βασικές συναρτήσεις τις οποίες θα αναπτύξουμε, πέρα από τις παραδοσιακές (πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση, υπόλοιπο διαίρεσης και αντίστροφος αριθμός) είναι οι τριγωνομετρικές συναρτήσεις του ημίτονου, του συνημιτόνου, της εφαπτομένης, της συνεφαπτομένης και οι αντίστροφες τους, καθώς και οι αντίστοιχες υπερβολικές, η εκθετική συνάρτηση, η συνάρτηση του δεκαδικού και του φυσικού (νεπέριου) λογαρίθμου, η τετραγωνική ρίζα, όπως και η ύψωση σε δύναμη.

Κατά την υλοποίηση, έχουμε βασιστεί στην ανάπτυξη ορισμένων συναρτήσεων κατά Taylor, ενώ για ορισμένες συναρτήσεις έχουμε βασιστεί στις εκδόσεις άλλων βασικών συναρτήσεων που έχουμε ήδη υλοποιήσει, αξιοποιώντας γνωστές ιδιότητες. Στη συνέχεια του κεφαλαίου, θα παρουσιάσουμε τη σχετική μαθηματική θεωρία και μεθοδολογία την οποία ακολουθήσαμε αναλυτικά, ενώ λεπτομέρειες που άπτονται στην υλοποίηση θα παρουσιαστούν στο αντίστοιχο κεφάλαιο της παρούσας εργασίας.

3.2 Η ανάλυση κατά Taylor

Στην επιστήμη των υπολογιστικών μαθηματικών, η σειρά Taylor (ή ανάλυση κατά Taylor) [1] είναι μια ιδιαίτερης σημασίας ανάλυση. Η σειρά Taylor αποτελεί την αναπαράσταση μίας συνάρτησης μιας μεταβλητής ως το άθροισμα άπειρων όρων μιας ακολουθίας, οι οποίοι υπολογίζονται με βάση τις τιμές των παραγώγων της σε ένα συγκεκριμένο σημείο.

Ιστορικά, οι σειρές Taylor έχουν εισαχθεί από τον Άγγλο μαθηματικό Brook Taylor, στις αρχές του 18ου αιώνα (1715). Στη περίπτωση που το κεντρικό σημείο της ανάλυσης είναι το μηδέν, τότε η σειρά, ονομάζεται και σειρά Mc-Laurin, από το όνομα του Σκοτσέζου μαθηματικού Κόλιν Mc-Laurin ο οποίος έκανε εκτεταμένη χρήση αυτής της ειδικής περίπτωσης των σειρών Taylor τον 18ο αιώνα.

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots$$

Η εφαρμογή των σειρών Taylor έχει μεγάλη πρακτική σημασία στην σύγχρονη υπολογιστική εποχή, καθώς προσφέρει αξιόπιστη προσέγγιση δύσκολων συναρτήσεων με χρήση απλών πολυωνύμων. Επιπρόσθετα, σύμφωνα με το θεώρημα του Taylor [2], αν η προσέγγιση μιας συνάρτησης f οριστεί μέχρι τον k όρο, τότε το σφάλμα προσέγγισης είναι της τάξης του τελευταίου όρου που χρησιμοποιήθηκε στη προσέγγιση, δηλαδή:

$$R_k(x) = o(|x-a|^k), \qquad x \to a$$

Στα πλαίσια της συγκεκριμένης εργασίας, θα επικεντρωθούμε στις σειρές Taylor με κέντρο το 0, (ή αλλιώς σειρές Mc-Laurin), καθώς η παραδοχή του μηδενικού κέντρου απλοποιεί σημαντικά τις διαδικασίες. Επιπρόσθετα, η ανάλυση θα διακόπτεται όταν το σφάλμα της προσέγγισης είναι υποδεέστερο μιας σταθερής μικρής ποσότητας, η οποία για τις ανάγκες της παρούσης εργασίας επιλέχθηκε να είναι ίση με 10^{-6} για μεταβλητές τύπου float και ίση με 10^{-15} για μεταβλητές τύπου double.

Στη συνέχεια του κεφαλαίου, θα αναφερθούμε πιο αναλυτικά στο θεώρημα Taylor πάνω στο οποίο βασίζονται τα αναπτύγματα Taylor και Mc-Laurin

3.3 Το θεώρημα Taylor

Έστω f(x) μια συνάρτηση η οποία είναι άπειρες φορές συνεχής και παραγωγίσιμη στο πεδίο ορισμού της Α. Έστω x_0 επίσης, ένα σημείο του Α. Τότε, για κάθε τιμή του x “κοντά” στο x_0, ισχύει:

$$f(x) = P_n^a(x) + R_n^a(x)$$

όπου $P_n^a(x) = f(a) + \frac{f^{(1)}(a)}{1!}(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n$

και $R_n^a(x) = \frac{f^{(n+1)}(c(x))}{(n+1)!}(x-a)^{n+1}$

Ο πρώτος όρος είναι το πολυώνυμο Taylor βαθμού n και κέντρου α της συνάρτησης f, ενώ ο δεύτερος όρος είναι το υπόλοιπο Taylor βαθμού n και κέντρου α της συνάρτησης f. Το υπόλοιπο

Taylor αποτελεί το θεωρητικό σφάλμα της προσέγγισης με χρήση σειρών. Η συνάρτηση c(x) είναι άγνωστη.

Ισοδύναμες έκφρασης του θεωρητικού σφάλματος Taylor αποτελούν το υπόλοιπο Lagrange και το υπόλοιπο Cauchy.

Το υπόλοιπο της σειράς Taylor κατά Lagrange είναι :

$R_n^a(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1}$, όπου ξ είναι ένα άγνωστο σημείο στο διάστημα (α,x) .

Το υπόλοιπο της σειράς Taylor κατά Cauchy είναι :

$$R_n^a(x) = \frac{(x-1)^{n+1}}{n!}(1-\delta)^n f^{(n+1)}(a+\delta(x-a))$$

3.4 Τα πλεονεκτήματα της ανάλυσης συναρτήσεων σε πολυωνυμικές μορφές

- Κάθε πολυωνυμική μορφή είναι συνάρτηση απείρως παραγωγίσιμη.
- Οι παράγωγοι των πολυωνυμικών συναρτήσεων υπολογίζονται εύκολα
- Τα πολυώνυμα είναι συναρτήσεις απείρως ολοκληρώσιμες
- Τα αόριστα ολοκληρώματα των πολυωνύμων είναι ευκόλως υπολογίσιμα

- Η πολυωνυμική προσέγγιση των συναρτήσεων, μας παρέχει μια εύκολα υλοποιήσιμη αριθμητική μέθοδο για την προσέγγιση της συνάρτησης με επιθυμητή ακρίβεια.

Ενώ τα πρώτα πλεονεκτήματα της πολυωνυμικής προσέγγισης των συναρτήσεων έχουν περισσότερο θεωρητικό ενδιαφέρον, το τελευταίο πλεονέκτημα της θα το αξιοποιήσουμε στην υπάρχουσα εργασία: Μια μεγάλη σειρά από συναρτήσεις θα υπολογίζονται με βάση την συγκεκριμένη ανάλυση, ενώ για τις υπόλοιπες θα αξιοποιούνται οι επιμέρους ιδιότητες.

3.5 Η εκθετική συνάρτηση

Για την ανάλυση της εκθετικής συνάρτησης σύμφωνα με το θεώρημα Taylor, σε πρώτο στάδιο πρέπει να υπολογίσουμε τις παραγώγους της συνάρτησης $f(x) = e^x$
Ισχύει ότι $f'(x) = f^{(2)}(x) = \cdots = f^{(n)}(x) = e^x$

Το επόμενο κρίσιμο βήμα είναι η εύρεση του κατάλληλου σημείου για τον υπολογισμό της σειράς. Την ιδανικότερη επιλογή στη περίπτωση αυτή αποτελούν εκείνες οι τιμές του x που μηδενίζουν κάποιες από τις παραγώγους. Όσον αφορά τη περίπτωση της εκθετικής συνάρτησης, αυτό δεν είναι εφικτό, καθώς είναι πάντα θετική για οποιαδήποτε πεπερασμένη τιμή. Έτσι, προσανατολιζόμαστε στη τιμή του x για την οποία οι παράγωγοι λαμβάνουν μια τιμή, η οποία διευκολύνει την ανάλυση. Για x=0 παρατηρούμε ότι όλοι οι παράγωγοι λαμβάνουν τιμή ίση με τη μονάδα, δηλαδή

$$f'(0) = f^{(2)}(0) = \cdots = f^{(n)}(0) = 1$$

Έτσι, η εκθετική συνάρτηση γράφεται στη μορφή:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Το επόμενο βήμα, είναι ο έλεγχος σύγκλισης της παραπάνω συνάρτησης. Ο έλεγχος πραγματοποιείται με τον έλεγχο της τιμής:

$$l = \lim_{n\to\infty} \left|\frac{a_{n+1}}{a_n}\right| = \lim_{n\to\infty} \left|\frac{x}{n+1}\right| = 0$$

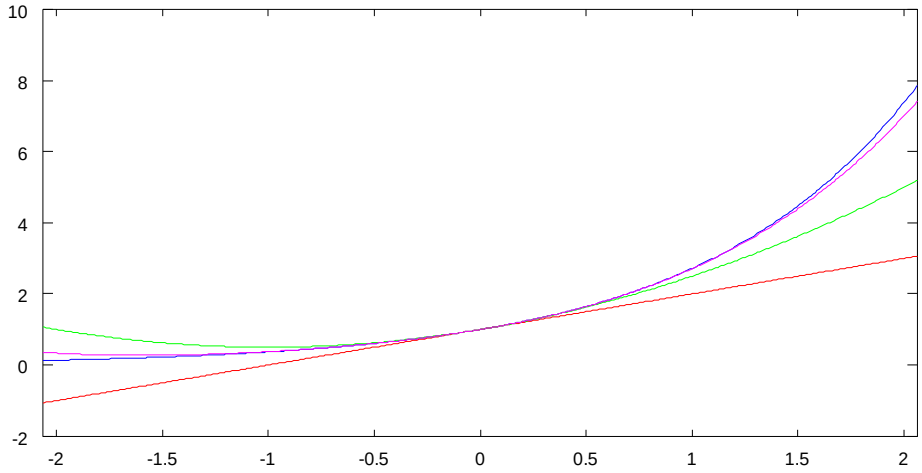
Επειδή *l*<1, σύμφωνα με το κριτήριο του d' Alembert, διαπιστώνουμε ότι η σειρά συγκλίνει για κάθε πεπερασμένη τιμή του x.

Με βάση την προηγούμενη ανάλυση, συνοψίζουμε ότι η σειρά Taylor για την εκθετική συνάρτηση είναι η :

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Η ανάλυση της εκθετικής συνάρτησης ισχύει για κάθε πεπερασμένη πραγματική τιμή του x.

Στη συνέχεια, επιβεβαιώνουμε σχηματικά την ακρίβεια της προσέγγισης του παραπάνω τύπου με γραφικό τρόπο:



Εικόνα 1: Η σύγκλιση της σειράς Taylor της εκθετικής συνάρτησης

Η μπλε γραμμή αναπαριστά την εκθετική συνάρτηση.

Η κόκκινη γραμμή αναπαριστά την προσέγγιση πρώτης τάξης (2 όροι της σειράς Taylor)

Η πράσινη γραμμή αναπαριστά την προσέγγιση δεύτερης τάξης (3 όροι της σειράς Taylor)

Η ιώδης γραμμή αναπαριστά την προσέγγιση τέταρτης τάξης (5 όροι της σειράς Taylor)

Παρατηρούμε στο διάγραμμα ότι με λίγους όρους της σειράς Taylor μπορούμε να επιτύχουμε άρτια προσέγγιση της εκθετικής συνάρτησης. Επιπρόσθετα, παρατηρούμε τη γρήγορη σύγκλιση της σειράς για μικρές τιμές του x, γεγονός που οφείλεται στην ύπαρξη του παράγοντα n! στο παρονομαστή της σειράς.

3.6 Η φυσική λογαριθμική συνάρτηση lnx

Στο επόμενο στάδιο, θα υπολογίσουμε την ανάλυση κατά Taylor της φυσικής λογαριθμικής συνάρτησης $f(x) = \ln x$.
Όπως και στην εκθετική συνάρτηση, αρχικά εξάγουμε όλες τις παράγωγους της f. Ισχύει ότι:

$$f'(x) = x^{-1}, \quad f^{(2)}(x) = -x^{-2}, \quad f^{(3)}(x) = 2x^{-2}, \quad f^{(4)}(x) = -2 \cdot 3x^{-4}$$

Παρατηρώντας την αλληλουχία των παραγώγων, συνάγεται το διαισθητικό συμπέρασμα ότι: $f^{(n)}(x) = (-1)^{n-1}(n-1)!\,x^{-n}$
Έπειτα, αποδεικνύουμε επαγωγικά τη διαίσθησή μας:

- Για n=1, η σχέση γράφεται $f'(x) = x^{-1}$, η οποία ισχύει
- Υποθέτουμε ότι ισχύει για n=k, δηλαδή ότι $f^{(k)}(x) = (-1)^{k-1}(k-1)!\,x^{-k}$
- Θα δείξουμε ότι ισχύει για n=k+1, θα δείξουμε δηλαδή ότι : $f^{(k+1)}(x) = (-1)^{k}(k)!\,x^{-k-1}$. Πράγματι, παραγωγίζοντας την υπόθεσή μας, αποδεικνύουμε το ζητούμενο.

$$f^{(k+1)}(x) = \frac{d}{dx}\{(-1)^{k-1}(k-1)!\,x^{-k}\} = (-1)^{k}(k)!\,x^{-k-1}$$

Το επόμενο στάδιο αποτελεί η επιλογή του κέντρου. Καθώς η συνάρτηση lnx έχει πεδίο ορισμού το σύνολο των θετικών πραγματικών αριθμών, μια ιδανική επιλογή αποτελεί η τιμή x_0=1. Έτσι,

$$f(x) = \ln x = f(x_0) + \sum_{n=1}^{\infty} \frac{(x-x_0)^n}{n!} f^{(n)}(x_0) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{(x-1)^n}{n}$$

Αν κάνουμε αντικατάσταση όπου x το x+1, προκύπτει ότι:

$$f(1+x) = \ln(1+x) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^n}{n}$$

Στη συνέχεια ελέγχουμε τη σύγκλιση της σειράς. Χρησιμοποιούμε, όπως και στη περίπτωση της εκθετικής σειράς, το κριτήριο του d' Alemebert.

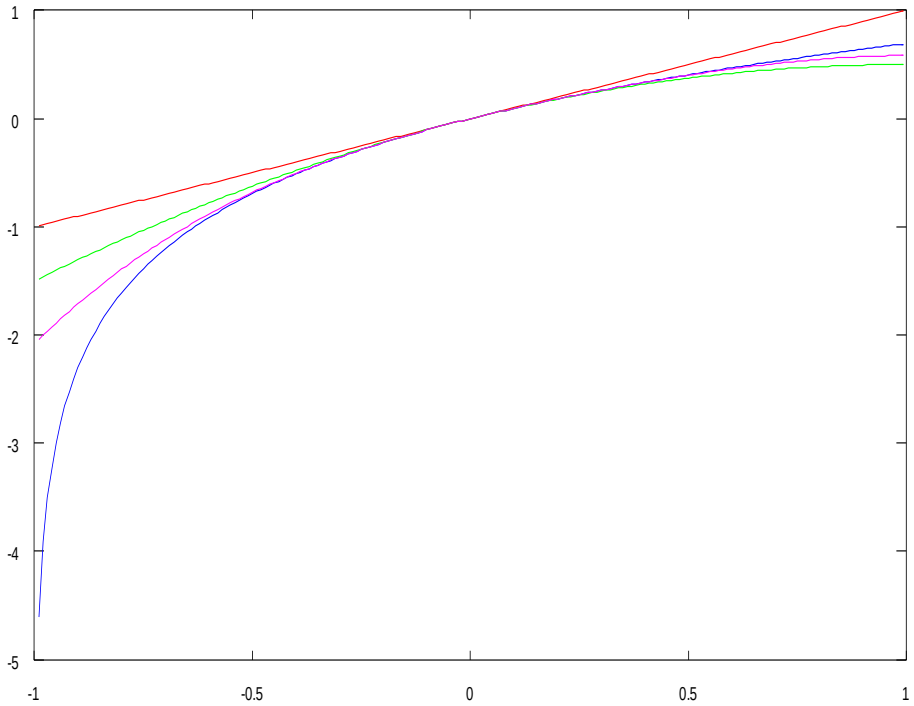
$$l = \lim_{n\to\infty}\left|\frac{a_{n+1}}{a_n}\right| = \lim_{n\to\infty}\left|\frac{n}{n+1}x\right| = |x|$$

Για να συγκλίνει η παραπάνω σειρά, πρέπει να ισχύει *l*<1, συνεπώς πρέπει να ισχύει |x|<1
Συνοψίζοντας, η lnx αναλύεται ως εξής:

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} + \cdots$$

Οι τιμές του x για τις οποίες η ανάλυση Taylor είναι δυνατή για τιμές του x στο διάστημα -1<x<1, επομένως, υπάρχει η δυνατότητα υπολογισμού της τιμής του λογαρίθμου μόνο για τιμές εισόδου στο διάστημα (0,2). Για τιμές εκτός των παραπάνω ορίων, αξιοποιούμε τον ορισμό ότι $\ln e^n = n$, καθώς και την ιδιότητα ότι $\ln ab = lna + lnb$ Έτσι, για κάθε αριθμό εκτός των ορίων, αναπτύσσουμε τη μορφή όπου b είναι ακέραιος αριθμός και α είναι ένας αριθμός μεταξύ 0 και 2.

Εν ακολουθία ελέγχουμε γραφικά την ορθότητα της προσέγγισης με σειρά Taylor της λογαριθμικής συνάρτησης.



Εικόνα 2: Η σύγκλιση της σειράς Taylor της λογαριθμικής συνάρτησης

Η μπλε γραμμή αναπαριστά την λογαριθμική συνάρτηση.

Η κόκκινη γραμμή αναπαριστά την προσέγγιση πρώτης τάξης (2 όροι της σειράς Taylor)

Η πράσινη γραμμή αναπαριστά την προσέγγιση δεύτερης τάξης (3 όροι της σειράς Taylor)

Η ιώδης γραμμή αναπαριστά την προσέγγιση τέταρτης τάξης (5 όροι της σειράς Taylor)

Παρατηρούμε ότι η προσέγγιση με τη σειρά Taylor της λογαριθμικής συνάρτησης είναι σημαντικά πιο αργή, σε σχέση με την εκθετική συνάρτηση. Αυτό οφείλεται κατά κύριο λόγο στην απουσία κατάλληλου όρου στο παρονομαστή της σειράς, ο οποίος αυξάνεται με ταχύτατους ρυθμούς. Αντίθετα, υπάρχει μόνο ο όρος n, ο οποίος αυξάνεται γραμμικά.

3.7 Η δεκαδική λογαριθμική συνάρτηση logx

Όσον αφορά τη δεκαδική λογαριθμική συνάρτηση, δε θα χρησιμοποιήσουμε ανάλυση taylor, καθώς η συνάρτηση logx υπολογίζεται με βάση το τύπο αλλαγής βάσης λογαρίθμων από τη lnx ως εξής:

$$\log x = \frac{\ln x}{\ln 10}$$

3.8 Η συνάρτηση sinx

Έπειτα, θα υπολογίσουμε την ανάλυση κατά Taylor της φυσικής λογαριθμικής συνάρτησης $f(x) = \sin x$.

Όπως και στις προηγούμενες συναρτήσεις, αρχικά εξάγουμε όλες τις παράγωγους της f. Ισχύει ότι:

$$f'(x) = \cos x,\ f^{(2)}(x) = -\sin x,\ f^{(3)}(x) = -\cos x,\ f^{(4)}(x) = \sin x$$

Παρατηρώντας την αλληλουχία των παραγώγων, συνάγουμε το διαισθητικό συμπέρασμα ότι: $f^{(n)}(x) = \sin(x + n\frac{\pi}{2})$

Και στη περίπτωση αυτή, θα αποδείξουμε επαγωγικά τη διαισθητική n-οστή παράγωγο της συνάρτησης του ημιτόνου.

- Αρχικά, για n=1, ισχύει

$$f'(x) = \cos(x) = \sin\left(\frac{\pi}{2} - x\right) = -\sin\left(x - \frac{\pi}{2}\right) = \sin\left(\pi + x - \frac{\pi}{2}\right) = sin\left(x + \frac{\pi}{2}\right)$$

- Έστω ότι $f^{(k)}(x) = \sin(x + k\frac{\pi}{2})$
- Θα δείξουμε ότι $f^{(k+1)}(x) = \sin(x + (k+1)\frac{\pi}{2})$. Πράγματι:

$$f^{(k+1)}(x) = \cos\left(x + k\frac{\pi}{2}\right) = \sin\left(\frac{\pi}{2} - x - k\frac{\pi}{2}\right)$$

$$= -\sin\left(x + \frac{(k-1)\pi}{2}\right) = \sin\left(x + \frac{(k+1)\pi}{2}\right)$$

Συνεπώς, πράγματι ισχύει ότι $f^{(n)}(x) = \sin(x + n\frac{\pi}{2})$

Το επόμενο βήμα είναι η εκλογή του κέντρου της ανάλυσης. Στη περίπτωση αυτή, οι καταλληλότερες επιλογές φαίνεται να είναι τα σημεία kπ/2, έτσι επιλέγουμε κέντρο x_0=0. Τότε,

$$f^{(n)}(0) = \begin{cases} 1 & n \bmod 4 = 1 \\ -1 & n \bmod 4 = 3 \\ 0 & n \bmod 2 = 0 \end{cases}$$

Έτσι, η συνάρτηση του ημιτόνου αναπτύσσεται ως εξής:

$$f(x) = \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots$$

Τη παραπάνω σειρά, μπορούμε να τη γράψουμε και ως εξής:

$$f(x) = \sin x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

.

Στη συνέχεια ελέγχουμε τη σύγκλιση της παραπάνω σειράς με βάση το κριτήριο του d' Alembert, σύμφωνα με το οποίο:

$$l = \lim_{n\to\infty} \left|\frac{a_{n+1}}{a_n}\right| = \lim_{n\to\infty} \left|\frac{x^2}{(2n+2)(2n+3)}\right| = 0$$

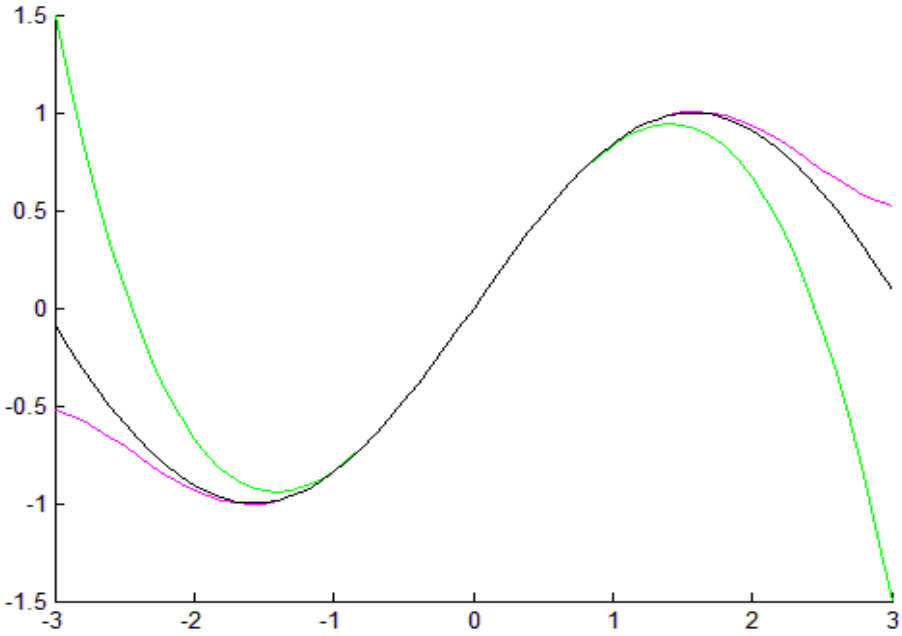
Άρα η σειρά συγκλίνει για κάθε πεπερασμένη τιμή του x.

Συνοψίζοντας, η ανάλυση κατά Taylor της συνάρτησης του ημιτόνου είναι:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots$$

για κάθε τιμή του x.

Εν συνεχεία ελέγχουμε τη σύγκλιση και την ακρίβεια της προσέγγισης της συνάρτησης του ημιτόνου με τη χρήση γραφικών παραστάσεων.



Εικόνα 3: Η σύγκλιση της σειράς του ημιτόνου

Η μαύρη γραμμή αναπαριστά την συνάρτηση του ημιτόνου.

Η κόκκινη γραμμή αναπαριστά την προσέγγιση πρώτης τάξης (1 όροι της σειράς Taylor)

Η πράσινη γραμμή αναπαριστά την προσέγγιση τρίτης τάξης (2 όροι της σειράς Taylor)

Η ιώδης γραμμή αναπαριστά την προσέγγιση έβδομης τάξης (4 όροι της σειράς Taylor)

Στη προσέγγιση κατά Taylor της συνάρτησης του ημιτόνου, παρατηρούμε σημαντική ακρίβεια στη προσέγγιση, καθώς με μόλις 5 όρους της σειράς, επιτυγχάνουμε σχεδόν απόλυτη ακρίβεια στο αποτέλεσμα. Αν συνυπολογίσουμε τις ιδιότητες της συνάρτησης, η ακρίβεια αυξάνεται ακόμη περισσότερο.

3.9 Η συνάρτηση cosx

Η εξαγωγή της ανάλυσης Taylor για τη συνάρτηση του συνημιτόνου, βασίζεται στο ανάπτυγμα του ημιτόνου και στη σχέση:

$$\frac{d}{dx}\sin x = \cos x$$

Όπως έχουμε ήδη εξάγει:

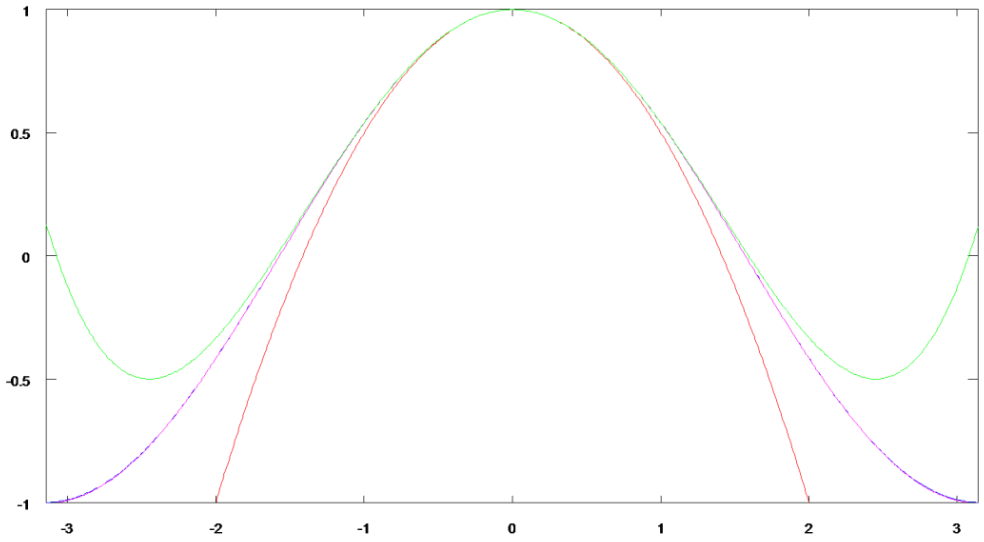
$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Παραγωγίζοντας την παραπάνω σχέση κατά μέλη, λαμβάνουμε:

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n (2n+1) x^{2n}}{(2n+1)!} = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

το οποίο ταυτίζεται με το ανάπτυγμα κατά Taylor της συνάρτησης του συνημιτόνου.

Έπειτα ελέγχουμε την ακρίβεια και τη σύγκλιση της προσέγγισης του συνημίτονου με τη χρήση γραφικών παραστάσεων.



Εικόνα 4: Η σύγκλιση της σειράς του συνημίτονου

Η μπλε γραμμή αναπαριστά την συνάρτηση του συνημίτονου.

Η κόκκινη γραμμή αναπαριστά την προσέγγιση δεύτερης τάξης (2 όροι της σειράς Taylor)

Η πράσινη γραμμή αναπαριστά την προσέγγιση τέταρτης τάξης (3 όροι της σειράς Taylor)

Η ιώδης γραμμή αναπαριστά την προσέγγιση όγδοης τάξης (5 όροι της σειράς Taylor)

Όπως και στη συνάρτηση του ημιτόνου, παρατηρούμε ότι μόλις με πέντε όρους της σειράς Taylor, μπορούμε να επιτύχουμε σχεδόν απόλυτη ακρίβεια στη προσέγγιση.

3.10 Η συνάρτηση tanx

Σχετικά με τη συνάρτηση της εφαπτομένης, όπως και με το δεκαδικό λογάριθμο δεν θα αξιοποιήσουμε το τύπο του Taylor άμεσα: Με βάση τις τριγωνομετρικές ιδιότητες η συνάρτηση της εφαπτομένης tanx υπολογίζεται με βάση τον τύπο:

$\tan x = \frac{\sin x}{\cos x}$.

3.11 Η συνάρτηση a^x

Όσον αφορά τη συνάρτηση της δύναμης, όπως και με τον δεκαδικό λογάριθμο δεν θα αξιοποιήσουμε τον τύπο του Taylor άμεσα: Με βάση τις ιδιότητες της εκθετικής συνάρτησης, η συνάρτηση a^x υπολογίζεται με βάση το τύπο:

$a^x = e^{x \ln a}$

3.12 Η συνάρτηση ρίζα $\sqrt{x}$

Για τη συνάρτηση της ρίζας, ακλουθούμε μια διαφορετική προσέγγιση προκειμένου να βρούμε το αποτέλεσμα, χρησιμοποιούμε την μέθοδο Newton-Rapson (μέθοδος τέμνουσας ή εφαπτόμενης).

Η μέθοδος αυτή αποτελεί μια από τις καλύτερες μεθόδους διαδοχικών προσεγγίσεων για την εύρεση ριζών μιας πραγματικής συνάρτησης. Ένα από τα βασικά πλεονεκτήματα της είναι ότι παρέχει την δυνατότητα να συγκλίνει αρκετά γρήγορα και η ταχύτητα αυτή εξαρτάται από το ποσό κοντά από την ζητούμενη λύση ξεκινάει η επαναληπτική διαδικασία. Βέβαια εάν η μέθοδος

ξεκινήσει μακριά από την ζητούμενη λύση υπάρχει περίπτωση να μη συγκλίνει τελικά.

Σχετικά με την αναλυτική κατασκευή της μεθόδου Newton-Rapson χρησιμοποιούμε το πολυώνυμο Taylor. Έστω μια συνάρτηση $f \in C^2[a,b]$ και έστω $x_0 \in [\alpha,\beta]$ μια προσεγγιση της ριζας x_1 τετοια ωστε:

$$f'(x_0) \neq 0$$

Και :

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|} = c|x_0 - x_1|$$

Θεωρούμε το πολυώνυμο Taylor πρώτου βαθμού για την $f(x)$ γύρω από το χ_0 :

$$f(x) = f(x_0) + (x_1 - x_0)f'(x_0) + \frac{(x_1 - x_0)^2}{2} f''(\xi)$$

Όπου το ξ βρίσκεται ανάμεσα στο x και στο x_0

Θεωρούμε τον όρο $(x_1 - x_0)^2$ αρκετα μικρο οποτε η σχεση μπορει να γραφτει :

$$0 \approx f(x_0) + (x_1 - x_0)^2 f'(x_1)$$

Λύνοντας ως προς χ_1 προκυπτει οτι :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \quad (1)$$

Αν x_n είναι μια προσέγγιση της ρίζας και x_{n+1} η νέα προσέγγιση, το x_{n+1} προκύπτει από τον επαναληπτικό τύπο :

$$x_{n+1=}x_n - \frac{f(x_n)}{f'(x_n)} \quad (2)$$

Το δεύτερο μέλος της (2) είναι η $g(x_n)$ της γενικής επαναληπτικής μεθόδου. Αν υποθέσουμε ότι η δεύτερη παράγωγος της $f(x_n)$ υπάρχει και είναι συνεχής, έχουμε την συνεχή συνάρτηση:

$$g'(x_n) = 1 - \frac{\left(f'(x_n)\right)^2 - f(x_n)f''(x_n)}{\left(f'(x_n)\right)^2} = \frac{f(x_n)f''(x_n)}{(f'(x_n))^2}$$

Για τη ρίζα ξ η παράγωγος γίνεται $g'(\xi) = 0$ αφού $f(\xi) = 0$. Αν ισχύει και $g''(\xi) \neq 0$ τότε η σύγκλιση είναι τουλάχιστον τετραγωνική (ταχεία σύγκλιση).

3.13 Η συνάρτηση τόξο ημιτόνου x

Με σκοπό την εύρεση του αναπτύγματος της συνάρτησης του αντίστροφου ημιτόνου, ακολουθούμε τη θεωρητική προσέγγιση.
Αρχικά υπολογίζουμε τις παραγώγους της συνάρτησης
$f(x) = \arcsin x$
Ισχύει ότι:

$$f'(x) = (1 - x^2)^{-\frac{1}{2}}$$

$$f^{(2)}(x) = x(1 - x^2)^{-\frac{3}{2}}$$

$$f^{(3)}(x) = (1 + 2x^2)(1 - x^2)^{-\frac{5}{2}}$$

$$f^{(4)}(x) = (9x + 6x^3)(1 - x^2)^{-\frac{7}{2}}$$

Η γενική μορφή της n-οστής παραγώγου της αντίστροφης ημιτονοειδούς αποδεικνύεται ότι δίνεται από τον τύπο:

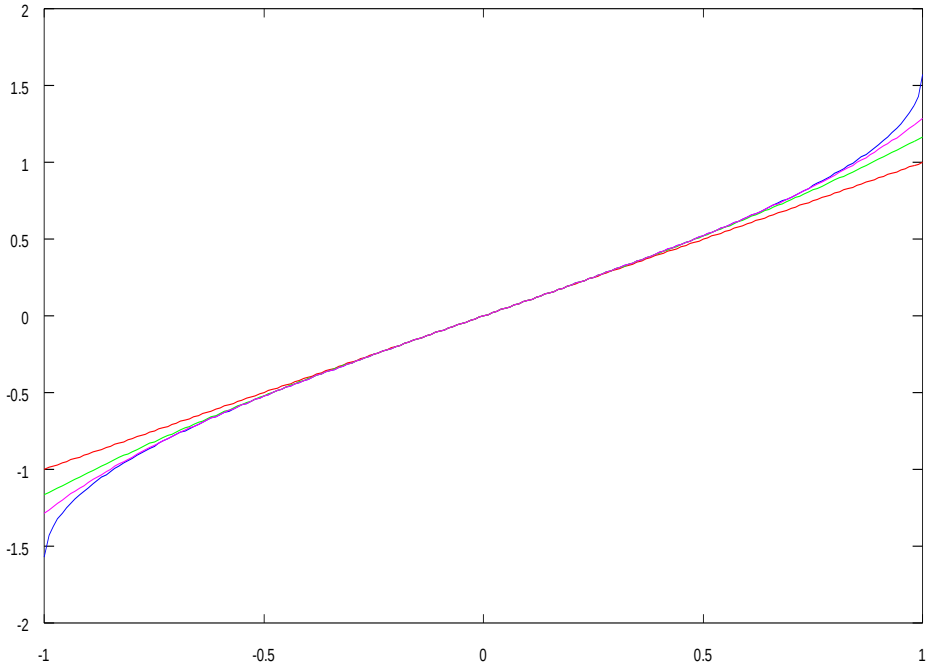
$$f^n(x) = (1 - x^2)^{-\frac{2n-1}{2}}(n-1)!\, x^{(n-1)mod\ 2} \times$$

$$\times \sum_{i=0}^{\frac{n-1-(n-1)mod\ 2}{2}} \frac{(n-1)!}{(2i)!\,4^{\frac{n-1-2i-(n-1)mod2}{2}}\left(\left(\frac{n-1-2i-(n-1)mod2}{2}\right)!\right)} x^{2i}$$

Έτσι, για x_0=0, ισχύει ότι $f^{(2k)}(0) = 0$ και $f^{(2k+1)}(0) = \frac{((2k)!)^2}{4^k(k!)^2}$.Με παρόμοιο τρόπο καταλήγουμε στον τύπο του Taylor, όπου η arcsinx αναλύεται με τον ακόλουθο τρόπο:

$$\arcsin x = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n(n!)^2(2n+1)} x^{2n+1}, |x| \leq 1$$

Στη συνέχεια ελέγχουμε την ακρίβεια της προσέγγισης με γραφικές μεθόδους:



Εικόνα 5: Η σύγκλιση της αντίστροφης ημιτονοειδούς συνάρτησης

Η μπλε γραμμή αναπαριστά την συνάρτηση του αντίστροφου ημιτόνου.

Η κόκκινη γραμμή αναπαριστά την προσέγγιση πρώτης τάξης (1 όρος της σειράς Taylor)

Η πράσινη γραμμή αναπαριστά την προσέγγιση τρίτης τάξης (2 όροι της σειράς Taylor)

Η ιώδης γραμμή αναπαριστά την προσέγγιση έβδομης τάξης (4 όροι της σειράς Taylor)

Όπως και στις προηγούμενες περιπτώσεις, χρειάζονται ελάχιστοι όροι για την ακριβή προσέγγιση της συνάρτησης του αντίστροφου ημιτόνου. Με μόλις 4 όρους και προσέγγιση με πολυώνυμο Taylor 7ης τάξης, παρατηρούμε ελάχιστες διαφορές, οι σημαντικότερες από τις οποίες εντοπίζονται στα άκρα του πεδίου ορισμού της συνάρτησης.

3.14 Η συνάρτηση τόξο συνημιτόνου x

Ομοίως, για την συνάρτηση του τόξου συνημιτόνου, ακολουθούμε τις τριγωνομετρικές ιδιότητες:

$$\arccos x = \frac{\pi}{2} - \arcsin x$$

.

3.15 Η συνάρτηση τόξο εφαπτομένης x

Η παραπάνω διαδικασία εφαρμόζεται και στην αντίστροφη εφαπτομένη και καταλήγουμε στην ακόλουθη ανάπτυξη Taylor της arctanx:

$$\arctan x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1}, |x| \leq 1$$

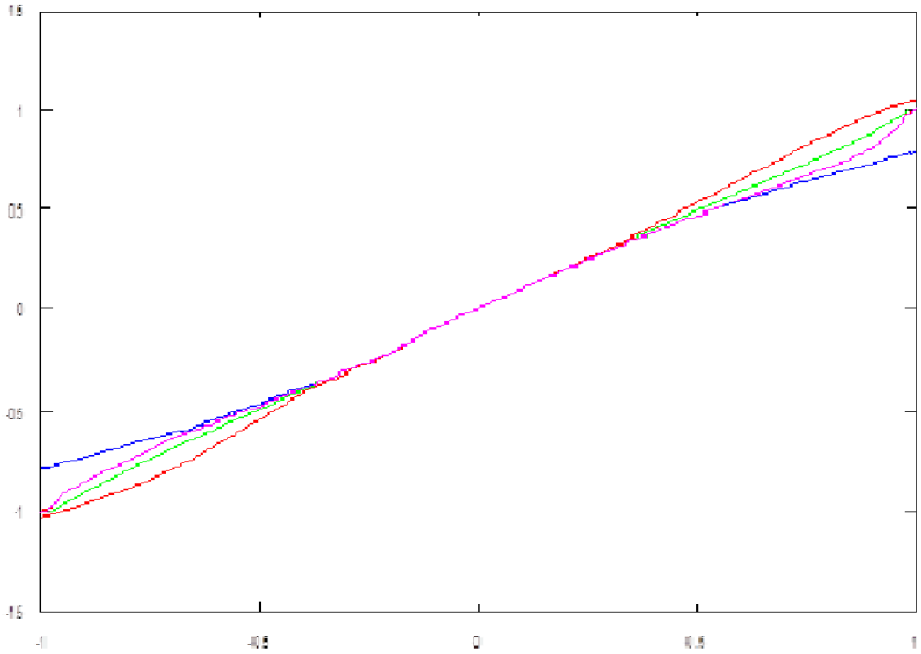
Στο ανάπτυγμα της συνάρτησης αυτής εισάγεται ένας ισχυρός περιορισμός.

Για την άρση του περιορισμού αυτού, αξιοποιούμε την ακόλουθη ιδιότητα της συνάρτησης:

$$\arctan\left(\frac{1}{x}\right) = \frac{\pi}{2} - \arctan x\,, x > 0$$

$$\arctan\left(\frac{1}{x}\right) = -\frac{\pi}{2} - \arctan x\,, x < 0$$

Με το τρόπο αυτό, μπορούμε να υπολογίσουμε τις τιμές της συνάρτησης για όλες τις τιμές του x



Εικόνα 6: Η σύγκλιση της αντίστροφης εφαπτομένης

Η μπλε γραμμή αναπαριστά την συνάρτηση του αντίστροφου ημιτόνου.

Η κόκκινη γραμμή αναπαριστά την προσέγγιση πρώτης τάξης (1 όρος της σειράς Taylor)

Η πράσινη γραμμή αναπαριστά την προσέγγιση τρίτης τάξης (2 όροι της σειράς Taylor)

Η ιώδης γραμμή αναπαριστά την προσέγγιση έβδομης τάξης (4 όροι της σειράς Taylor)

Στη συνέχεια, για το αρχικό διάστημα [-1,1] πραγματοποιούμε γραφικό τεστ σύγκλισης. Παρατηρούμε ότι εντοπίζουμε

αξιοσημείωτες διαφοροποιήσεις στα άκρα του διαστήματος σύγκλισης της σειράς, η οποία οφείλεται στο γεγονός ότι ο παρονομαστής αυξάνει με αργούς ρυθμούς. Ως αποτέλεσμα αυτού, είναι να απαιτούνται περισσότεροι όροι για τη καλύτερη προσέγγισή του, με χρήση της συγκεκριμένης δυναμόσειράς, καθώς οι τιμές του x απομακρύνονται από το 0.

3.16 Η συνάρτηση sinhx

Το υπερβολικό ημίτονο μπορεί να υπολογιστεί άμεσα από την σχέση:

$$\sinh x = \frac{e^x - e^{-x}}{2} = \frac{e^{2x} - 1}{2e^x}$$

3.17 Η συνάρτηση coshx

Το υπερβολικό συνημίτονο μπορεί να υπολογιστεί άμεσα από την σχέση:

$$\cosh x = \frac{e^x + e^{-x}}{2} = \frac{e^{2x} + 1}{2e^x}$$

3.18 Η συνάρτηση tahnhx

Η υπερβολική εφαπτομένη μπορεί να υπολογιστεί άμεσα από την σχέση:

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Για τις υπερβολικές συναρτήσεις, ακολουθούμε τις ιδιότητες τους.

Κεφάλαιο 4

Οι μετατροπές της επιστημονικής αριθμομηχανής

4.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφονται οι διαδικασίες μετατροπής μεταξύ των διαφόρων μορφών που υποστηρίζει η επιστημονική αριθμομηχανή

4.2 Μετατροπή ακεραίων αριθμών από δεκαδικό σε δυαδικό

Η διαδικασία μετατροπής ακέραιων δεκαδικών αριθμών στο δυαδικό σύστημα αρίθμησης, η οποία στην υλοποίησή μας πραγματοποιείται αυτόματα από τις εγγενείς διαδικασίες της γλώσσας, βασίζεται σε διαδοχικές διαιρέσεις του αριθμού εισόδου με τον αριθμό βάσης του δυαδικού συστήματος αρίθμησης, το δύο (2). Το πηλίκο που λαμβάνουμε από κάθε διαίρεση, διαιρείται εκ νέου με το 2 μέχρις ότου το πηλίκο να είναι μηδενικό. Το υπόλοιπο της πρώτης διαίρεσης με τον αριθμό 2 αποτελεί το λιγότερο σημαντικό ψηφίο (Less significant digit - LSD), ενώ το υπόλοιπο της τελευταίας διαίρεσης, αποτελεί το σημαντικότερο ψηφίο (Most significant digit - MSD). Στο παράδειγμα που ακολουθεί παρουσιάζεται αναλυτικά η διαδικασία μετατροπής:

Αριθμός εισόδου: 86

Είσοδος	Πηλίκο διαίρεσης με το 2	Υπόλοιπο διαίρεσης με το 2
86	43	0
43	21	1
21	10	1
10	5	0
5	2	1
2	1	0
1	0	1

Πίνακας 1:Μετατροπή από το δεκαδικό στο δυαδικό σύστημα αρίθμησης

Άρα η δυαδική έκφραση του αριθμού 86 είναι 01010110

Η παραπάνω διαδικασία ισχύει για τους θετικούς αριθμούς. Στο δυαδικό σύστημα αρίθμησης από πλευράς υλοποίησης στο υλικό τον υπολογιστών, υπάρχει η πρόβλεψη για αναπαράσταση των αρνητικών αριθμών, με βάση το συμπλήρωμα ως προς δύο. Δηλαδή, ένας αρνητικός αριθμός παριστάνεται με το συμπλήρωμα ως προς 2 της δυαδικής αναπαράστασης της απόλυτης τιμής του.

Έτσι, για τον αριθμό -86, υπολογίζουμε τη δυαδική αναπαράσταση του 01010110 και εν συνεχεία λαμβάνουμε το συμπλήρωμα ως προς 1, το οποίο προέρχεται από την αντικατάσταση όλων των δυαδικών ψηφίων με το αντίθετό τους. Δηλαδή, 10101001. Τέλος, λαμβάνουμε το συμπλήρωμα ως προς 2, απλώς προσθέτοντας μια μονάδα στο συμπλήρωμα ως προς 1. Άρα, για το -86 έχουμε την αναπαράσταση 10101010.

Στην υλοποίηση μας, οι αριθμοί αναπαρίστανται ως long integers με μήκος συνολικά 32 δυαδικά ψηφία.

4.3 Μετατροπή αριθμών κινητής υποδιαστολής από δεκαδικό σε δυαδικό

Στην επιστήμη των υπολογιστών, οι αριθμοί κινητής υποδιαστολής αναπαρίστανται σαν ένας αριθμός μεταξύ 1 και 2 πολλαπλασιαζόμενος με μια ακέραια (θετική ή αρνητική) δύναμη του 2, παρόμοια με την εκθετική παρουσίαση των αριθμών στο δεκαδικό σύστημα: 125 = 1.25*10^2. Προφανώς και στην περίπτωση του δεκαδικού συστήματος ο αριθμός είναι μεταξύ 1 και 10 και πολλαπλασιάζεται με μια ακέραια δύναμη του 10.

Εσωτερικά, η αναπαράσταση του αριθμού πραγματοποιείται με βάση τα πρότυπα της IEEE 754/2008, όπου περιγράφεται αναλυτικά η αναπαράσταση σε επίπεδο υλικού. Περιληπτικά, η αναπαράσταση αποτελείται από 3 βασικά μέρη:

- Το bit του προσήμου (1 bit, ανεξάρτητα από την αναπαράσταση του αριθμού)
- Τα bit του εκθέτη (8 bit για αριθμό μονής ακρίβειας, 11 bit για αριθμό διπλής ακριβείας και 15 bit για αριθμό τετραπλής ακριβείας).
- Τα bit του αριθμού μεταξύ 1 και 2 (23 bit για αριθμό μονής ακρίβειας, 52 bit για αριθμό διπλής ακρίβειας και 112 bit για αριθμό τετραπλής ακριβείας)

Το bit του προσήμου είναι 0 για θετικούς και 1 για αρνητικούς αριθμούς. Τα bit του αριθμού μεταξύ 1 και 2, περιλαμβάνουν τη δυαδική αναπαράσταση του δεκαδικού τμήματος του αριθμού μεταξύ 1 και 2 (χωρίς να καταγράφεται το ακέραιο τμήμα).

Τέλος, ο εκθέτης περιγράφεται με χρήση offset, ως εξής:

$$e_{new} = 2^{R-1} + e$$

όπου e είναι ο εκθέτης του αριθμού και e_{new} είναι ο ο αριθμός που αποθηκεύεται σύμφωνα με το πρότυπο της IEEE και R είναι ο αριθμός των bit που χρησιμοποιούνται για την αναπαράσταση του εκθέτη.

Στο επόμενο παράδειγμα, αναλύεται ο αριθμός 1.1 σε μορφή αριθμού κινητής υποδιαστολής μονής ακρίβειας (ο οποίος διαθέτει 1 bit για το πρόσημο, 8 bit για τον εκθέτη και 23 bit για τη mantissa)

Ανάπτυγμα:

Ο αριθμός 1.1 γράφεται ως $+1.1 \cdot 2^0$. Άρα, για το πρόσημο έχουμε το bit 0, για τον εκθέτη έχουμε τον αριθμό 0 και ο αριθμός που θα αναπτύξουμε σε δυαδική μορφή είναι ο 0.1

Πρόσημο:

Το πρόσημο του αριθμού είναι θετικό, άρα το bit του προσήμου είναι 0.

Εκθέτης:

Ο εκθέτης μετατρέπεται σε δυαδικό σύστημα έπειτα την εφαρμογή της ολίσθησης. Έτσι, με εφαρμογή της σχέσης ολίσθησης έχουμε:

$$e_{new} = 2^{R-1} - 1 + e = 127 + 0 = 127$$

Άρα, η αναπαράσταση του εκθέτη είναι 01111111

Mantissa:

Όσον αφορά τη μετατροπή της mantissa ακολουθούμε μια διαδικασία παρόμοια με τη μετατροπή των ακεραίων. Μόνο που στη περίπτωση των δεκαδικών, εκτελούμε πολλαπλασιασμούς αντί για διαιρέσεις. Τα δυαδικά ψηφία τα λαμβάνουμε από τα ακέραια μέρη των πολλαπλασιασμών, ενώ με τα δεκαδικά τμήματα επαναλαμβάνουμε την ίδια διαδικασία. Παρακάτω ακολουθεί η διαδικασία μετατροπής για τον αριθμό 0.1

Αριθμός	Γινόμενο x 2	Ακέραιο μέρος	Δεκαδικό μέρος
0.1	0.2	0	0.2
0.2	0.4	0	0.4
0.4	0.8	0	0.8
0.8	1.6	1	0.6
0.6	1.2	1	0.2
0.2	0.4	0	0.4
…			

Πίνακας 2:Παράδειγμα μετατροπής αριθμού κινητής υποδιαστολής από δεκαδικό σε δυαδικό

Παρατηρούμε ότι έπειτα από ορισμένα βήματα, επανεμφανίζεται αριθμός που έχει ήδη εμφανιστεί νωρίτερα στην ανάλυση. Το γεγονός αυτό σημαίνει ότι ο αριθμός που προσπαθούμε να αναπαραστήσουμε αποτελείται (μετά την υποδιαστολή) από άπειρα ψηφία περιοδικά επαναλαμβανόμενα. Αυτό είναι σύνηθες φαινόμενο σε τέτοιες μετατροπές.

Έτσι, η mantissa του ζητούμενου αριθμού είναι 10011001100110011001100.

Αν συνοψίσουμε, ο αριθμός 1.1 σε μορφή αριθμού κινητής υποδιαστολής μονής ακρίβειας είναι: 0011 1111 1000 1100 1100 1100 1100 1100.

4.4 Μετατροπή από δυαδικό σε οκταδικό

Η μετατροπή από δυαδικό σε οκταδικό πραγματοποιείται με την ομαδοποίηση του δυαδικού αριθμού σε τριάδες ψηφίων, ξεκινώντας από το τέλος. Στη συνέχεια κάθε τριάδα, μετατρέπεται σε οκταδικό αριθμό. Η αναπαράσταση που προκύπτει, αποτελεί την οκταδική έκφραση του δυαδικού αριθμού εισόδου.

Παράδειγμα, ο αριθμός 86. Έχει δυαδική αναπαράσταση 01010110. Η οκταδική του υπολογίζεται ως εξής:

1. 01| 010 |110
2. 1 | 2 |6
3. 126

Άρα η οκταδική αναπαράσταση του αριθμού 86 είναι 126

4.5 Μετατροπή από δυαδικό σε δεκαεξαδικό

Η μετατροπή σε δεκαεξαδικό σύστημα γίνεται παρόμοια με την μετατροπή στο οκταδικό, με τη διαφορά ότι η ομαδοποίηση γίνεται σε τετράδες δυαδικών ψηφίων.

01010110 → 0101 | 0110 → 56

Άρα η δεκαεξαδική του αριθμού 86 είναι 56

Κεφάλαιο 5

Υπολογισμός πολύπλοκων μαθηματικών παραστάσεων

5.1 Εισαγωγή

Για τον υπολογισμό πολύπλοκων μαθηματικών παραστάσεων, μια παραδοσιακά χρησιμοποιούμενη μεθοδολογία βασίζεται στη μεταθεματική (ή αλλιώς γνωστή ως επίθεματικη ή αντίστροφη πολωνική) γραφή τους. Πιο συγκεκριμένα, κάθε μαθηματική παράσταση μετατρέπεται από την ενδοθεματική μορφή της, στην μεταθεματική και στη συνέχεια, με χρήση ενός απλού αλγορίθμου που αξιοποιεί τις ιδιότητες της στοίβας (stack), πραγματοποιείται ο σχετικός υπολογισμός.

Πριν όμως προχωρήσουμε στη περιγραφή του αλγορίθμου υπολογισμού των πολύπλοκων αριθμητικών παραστάσεων, θα κάνουμε μια σύντομη ανασκόπηση της ενδοθεματικής και μεταθεματικής γραφής των παραστάσεων.

5.2 Ενδοθεματική γραφή

Η ενδοθεματική γραφή των παραστάσεων προϋποθέτει ότι κάθε τελεστής γράφεται ανάμεσα στις δύο μεταβλητές στις οποίες επιδρά. Η ενδοθεματική γραφή είναι ο συνηθισμένος τρόπος γραφής των μαθηματικών παραστάσεων που χρησιμοποιούμε στην καθημερινή ζωή και αξιοποιείται σχεδόν από το σύνολο του έντυπου υλικού για τη παρουσίαση των μαθηματικών σχέσεων. Έτσι, για παράδειγμα, η ακόλουθη απλή παράσταση είναι εκφρασμένη στην ενδοθεματική της μορφή:

$(A+B)C+DE-F$

5.3 Μεταθεματική μορφή

Η μεταθεματική μορφή (ή αντίστροφη Πολωνική) είναι μια μαθηματική σημειογραφία όπου κάθε μαθηματικός τελεστής ακολουθεί όλους τους τελεστές, σε πλήρη αντίθεση με την προθεματική μορφή (ή Πολωνική), όπου ο τελεστής προηγείται. Είναι επίσης γνωστή και ως Postfix και δεν είναι απαραίτητη η χρήση των παρενθέσεων. Η ονομασία «πολωνική» αναφέρεται στην εθνικότητα του επιστήμονα της λογικής Jan Łukasiewicz , που εφηύρε την Πολωνική μορφή στη δεκαετία του 1920.

Η αντίστροφη πολωνική γραφή προτάθηκε αρχικά το 1954 από τους Burks, Warren, και Wright [3] και αξιοποιήθηκε από τους FL Bauer και EW Dijkstra στις αρχές της δεκαετίας του 1960 για τη μείωση του αριθμού προσβάσεων στη μνήμη ενός υπολογιστή αλλά και για την αξιοποίηση της στοίβας για τον υπολογισμό πολύπλοκων εκφράσεων. Οι αλγόριθμοι και οι συμβολισμοί αυτού

του σχήματος επεκτάθηκαν, από τον αυστραλιανό φιλόσοφο και πληροφορικό Charles Hamblin στα μέσα της δεκαετίας του 1950.

Κατά τη διάρκεια του 1970 και του 1980, η αντίστροφη πολωνική γραφή είχε κάποια αποδοχή, ακόμη και από το ευρύ κοινό, καθώς είχε αξιοποιηθεί ευρέως σε φορητές αριθμομηχανές της εποχής όπως οι αριθμομηχανές της σειράς HP-10C.

Τα πλεονεκτήματα της αντίστροφης πολωνικής γραφής είναι πολλά:

- Οι υπολογισμοί πραγματοποιούνται μόλις οριστεί (εμφανιστεί) ο τελεστής.
- Η αυτόματη στοίβα επιτρέπει και την αυτόματη αποθήκευση των ενδιάμεσων αποτελεσμάτων για την μετέπειτα χρήση.
- Οι παρενθέσεις είναι περιττές: απλά εκτελούνται οι υπολογισμοί με τη σειρά που δίνονται. Ομοίως, η προτεραιότητα των τελεστών που απαιτείται στην ενδοθεματική μορφή, είναι πλεονασμός στην αντίστροφη πολωνική.

Η αντίστροφη πολωνική μορφή όμως έχει και μειονεκτήματα. Το κυριότερο από αυτά εντοπίζεται στο γεγονός ότι όταν γράφονται σε χαρτί, οι γειτονικοί αριθμοί χρειάζονται ένα απαραίτητο διαχωριστικό μεταξύ τους, ενώ στα χειρόγραφα, τα δεδομένα περιπλέκονται ακόμη περισσότερο.

5.4 Μετατροπή ενδοθεματικών παραστάσεων σε μεταθεματική γραφή

Ο βασικότερος αλγόριθμος μετατροπής είναι ο αλγόριθμος Shunting-yard ο οποίος έχει αναπτυχθεί από το Djikstra.

Παρακάτω περιγράφονται αναλυτικά τα βήματα του αλγορίθμου μετατροπής, θεωρώντας ότι “μαθηματικά στοιχεία” αποτελούν οι αριθμοί, οι τελεστές και οι παρενθέσεις. Ο αλγόριθμος, που χρησιμοποιεί μια στοίβα για την μετατροπή, σαρώνει την ενδοθεματική έκφραση, ξεκινώντας από την αρχή ως εξής:

1. Αν υπάρχει, λαμβάνουμε το επόμενο μαθηματικό στοιχείο από την ενδοθεματική έκφραση, διαφορετικά πηγαίνουμε στο βήμα 6
2. Αν το στοιχείο είναι αριθμός, τότε το μεταφέρουμε απευθείας στην μεταθεματική γραφή και συνεχίζουμε στο βήμα 1
3. Αν το στοιχείο είναι άνοιγμα παρένθεσης, τότε αυτή τοποθετείται στη στοίβα και συνεχίζουμε στο βήμα 1.
4. Αν το στοιχείο είναι κλείσιμο παρένθεσης, τότε

4.1. Όσο το τελευταίο στοιχείο της στοίβας δεν είναι το άνοιγμα παρένθεσης

a) Εξάγουμε το τελευταίο στοιχείο και το τοποθετούμε στην μεταθεματική γραφή

4.2. Εξάγουμε το τελευταίο στοιχείο της στοίβας και συνεχίζουμε στο βήμα 1

5. Αν το στοιχείο είναι τελεστής, τότε:

5.1. Αν η στοίβα είναι άδεια:

a) τοποθετούμε τον τελεστή στη στοίβα

b) συνεχίζουμε στο βήμα 1

5.2. Αν το τελευταίο στοιχείο της στοίβας είναι άνοιγμα παρένθεσης, ή τελεστής μικρότερης προτεραιότητας:

a) τοποθετούμε τον τελεστή στη στοίβα

b) συνεχίζουμε στο βήμα 1

5.3. Όσο η στοίβα έχει μέσα δεδομένα και το τελευταίο στοιχείο της στοίβας δεν είναι μικρότερης προτεραιότητας:

a) εξάγουμε το τελευταίο στοιχείο της στοίβας

b) το τοποθετούμε στην μεταθεματική γραφή

5.4. Τοποθετούμε τον τελεστή στη στοίβα

5.5. συνεχίζουμε στο βήμα 1

6. Όσο η στοίβα έχει μέσα δεδομένα:

6.1. εξάγουμε το τελευταίο στοιχείο της στοίβας

6.2. το τοποθετούμε στη μεταθεματική γραφή

Κεφάλαιο 6

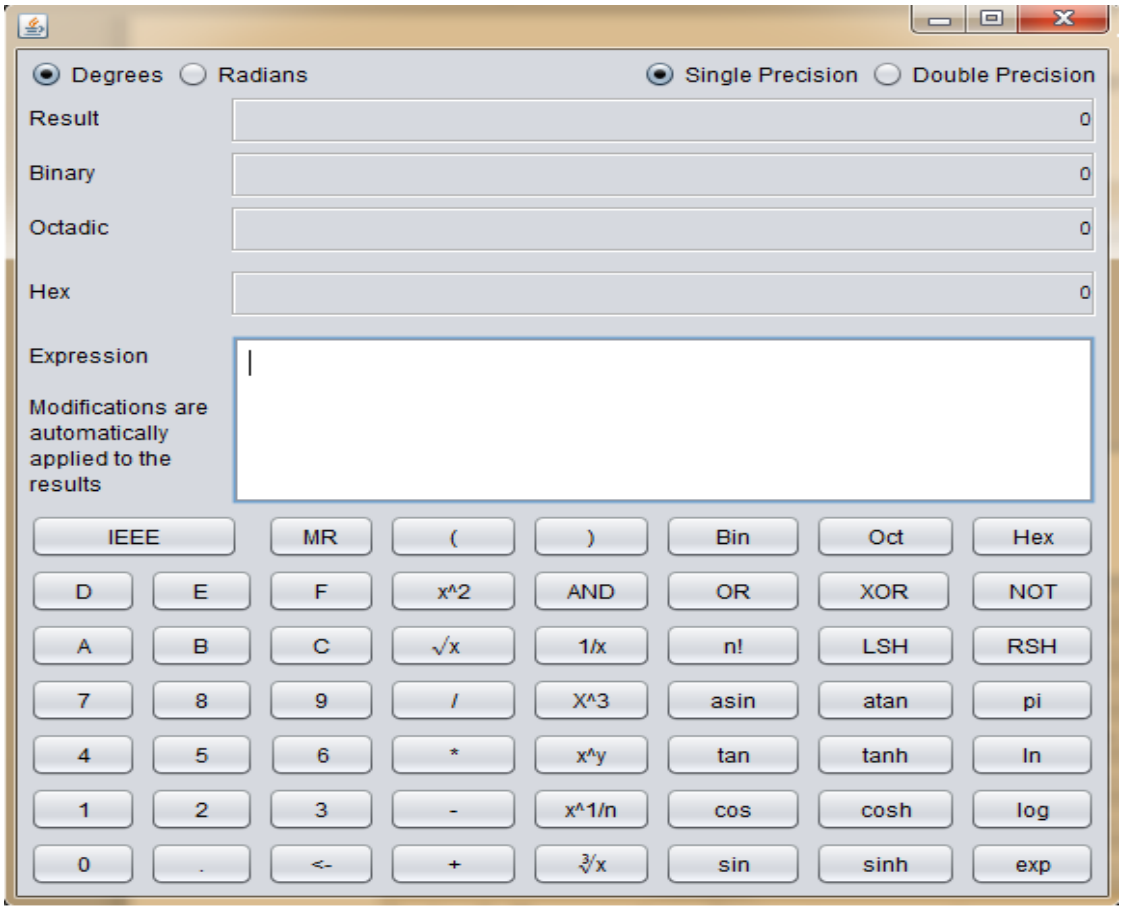
Υλοποίηση της επιστημονική αριθμομηχανής

6.1 Σχεδίαση της εφαρμογής

Για τη γραφική διεπαφή της εφαρμογής μας, δεν προχωρήσαμε σε σημαντικές καινοτομίες. Ακολουθήθηκε η αισθητική εμφάνιση των περισσότερων εφαρμογών επιστημονικής αριθμομηχανής, καθώς μια τέτοια προσέγγιση είναι ευρέως διαδεδομένη, προσιτή και εύληπτη από τους χρήστες. Προστέθηκαν όμως, σημαντικές λειτουργικές λεπτομέρειες, οι οποίες θα παρουσιαστούν στη συνέχεια.

Για το σχεδιασμό της διεπαφής αξιοποιήσαμε την ενσωματωμένη σχεδιαστική διαδικασία που διαθέτει η πλατφόρμα του Netbeans, η οποία παρέχει τη δυνατότητα στο χρήστη που τη χρησιμοποιεί να ενσωματώσει όλα τα στοιχεία εκείνα, τα οποία θα καταστήσουν τη διεπαφη λειτουργική ,εμφανίσιμη και κυρίως φιλική προς το χρήστη. Η βασική ιδέα για το σχεδιασμό της διεπαφης ήταν η δημιουργία ενός καλαίσθητου και εύχρηστου προγράμματος το οποίο θα μπορούσε να χρησιμοποιηθεί ανεξαρτήτως του επιπέδου μόρφωσης του εκάστοτε χρήστη, παρέχοντας όμως παράλληλα σύνθετες λειτουργίες. Στις ακόλουθες εικόνες, παρουσιάζονται ορισμένα screenshot από την εφαρμογή. Στην Εικόνα 7 παρουσιάζεται το βασικό παράθυρο της εφαρμογής.

Αυτό μπορεί να χωριστεί σε λογικές οντότητες όπως αυτό φαίνεται στην Εικόνα 8.



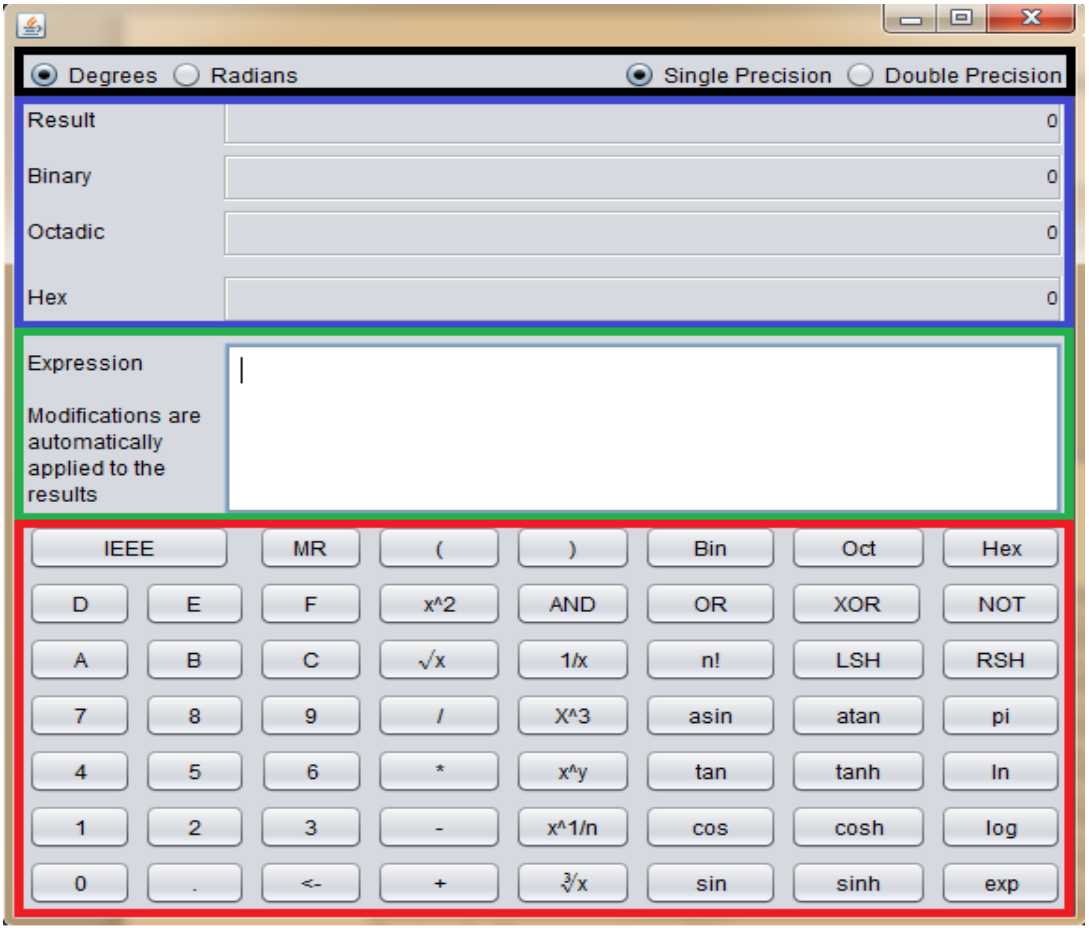
Εικόνα 7: Το γραφικό περιβάλλον της εφαρμογής

Το περιβάλλον εργασίας χωρίζεται σε τέσσερα λογικά τμήματα:

- Το πεδίο πληκτρολόγησης αριθμών/πράξεων: Αυτό το τμήμα περιλαμβάνει το σύνολο των πλήκτρων που μπορεί να περιλαμβάνει μια επιστημονική αριθμομηχανή. Θα αναφερθούμε διεξοδικά στο πεδίο αυτό στη συνέχεια του κεφαλαίου
- Το πεδίο επιλογής μονάδων αρίθμησης/μέτρησης: Το τμήμα αυτό περιλαμβάνει 2 διαφορετικές επιλογές: την ακρίβεια του συστήματος αρίθμησης (απλή ή διπλή) και το σύστημα

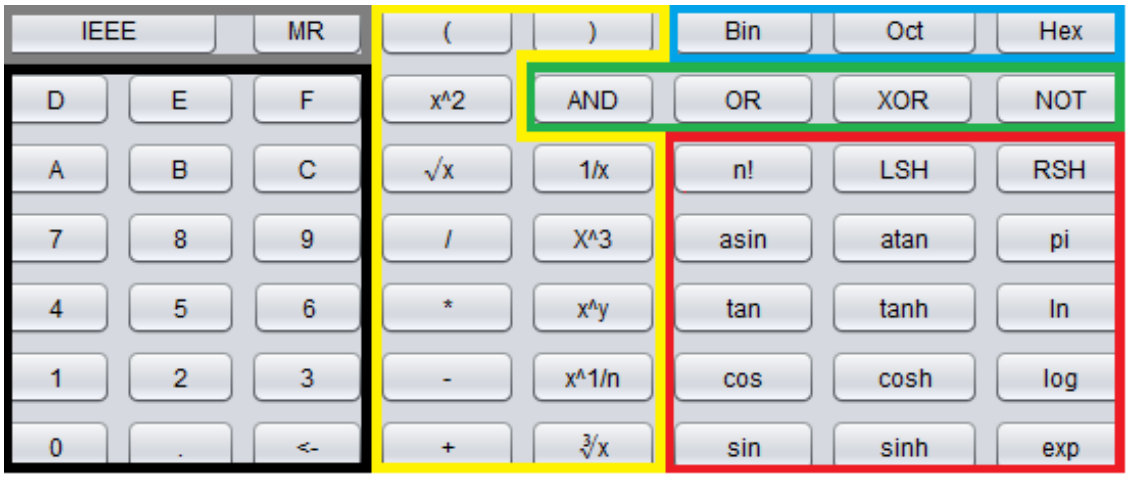
μέτρησης γωνιών (ακτίνια ή μοίρες). Ανάλογα με τις επιλογές του χρήστη, εμφανίζεται και το κατάλληλο αποτέλεσμα. Οι προεπιλεγμένες ρυθμίσεις είναι η απλή ακρίβεια σύστημα αρίθμησης και το σύστημα μέτρησης των γωνιών σε μοίρες.

- Το πεδίο εμφάνισης των αποτελεσμάτων
- Το πεδίο επεξεργασίας της έκφρασης υπολογισμού. Το συγκεκριμένο πεδίο αποτελεί σημαντική διαφοροποίηση συγκριτικά με τις περισσότερες αριθμομηχανές και προσεγγίζει μαθηματικά προγράμματα ανώτερου επιπέδου. Διεξοδική ανάλυση για τη συγκεκριμένη λειτουργία ακολουθεί στη συνέχεια του κεφαλαίου.



Εικόνα 8:Τα τέσσερα λογικά τμήματα της εφαρμογής

Το τμήμα πληκτρολόγησης μπορεί να χωριστεί σε έξι λογικά υποτμήματα, όπως φαίνεται στην Εικόνα 9. Το πρώτο τμήμα αναφέρεται στο κουμπί μνήμης όπου ο χρήστης μπορεί να εμφανίσει στο πεδίο εισαγωγής εκφράσεων κάποιον αριθμό τον οποίο έχει επεξεργαστεί στο δεύτερο παράθυρο της εφαρμογής, παράλληλα το τμήμα αυτό περιέχει την ρουτίνα η οποία καλεί το δεύτερο αυτό παράθυρο (παρακάτω θα γίνει η παρουσίαση του IEEE παραθύρου). Το δεύτερο τμήμα αναφέρεται στα πλήκτρα εισαγωγής αριθμών, το τρίτο στα πλήκτρα των βασικών πράξεων, το τέταρτο τμήμα στα πλήκτρα των πιο εξειδικευμένων (επιστημονικών) πράξεων, το πέμπτο τμήμα, των λογικών πράξεων και τέλος το έκτο τμήμα που περιλαμβάνει τις πράξεις συστημάτων αρίθμησης.



Εικόνα 9 : Τα έξι τμήματα του πεδίου πληκτρολόγησης

Η λογική στην οποία έχει βασιστεί η σχεδίαση της αριθμομηχανής ακολουθεί τη σχεδιαστική προσέγγιση των περισσότερων παρόμοιων εφαρμογών, καθώς είναι ευρέως διαδεδομένη, προσιτή και εύληπτη από τους χρήστες. Τα βασικά τμήματα της

αριθμομηχανής είναι διακριτά, ενώ ακόμη και το πεδίο πληκτρολόγησης διαχωρίζεται σε υποτμήματα τα οποία είναι ανεξάρτητα μεταξύ τους. Σύμφωνα με τις αρχικές εκτιμήσεις μας, ο υποψιασμένος χρήστης δεν θα χρειαστεί πολύ χρόνο για να εξοικειωθεί με το συγκεκριμένο περιβάλλον λειτουργίας και να αρχίσει άμεσα την χρήση του προγράμματος.

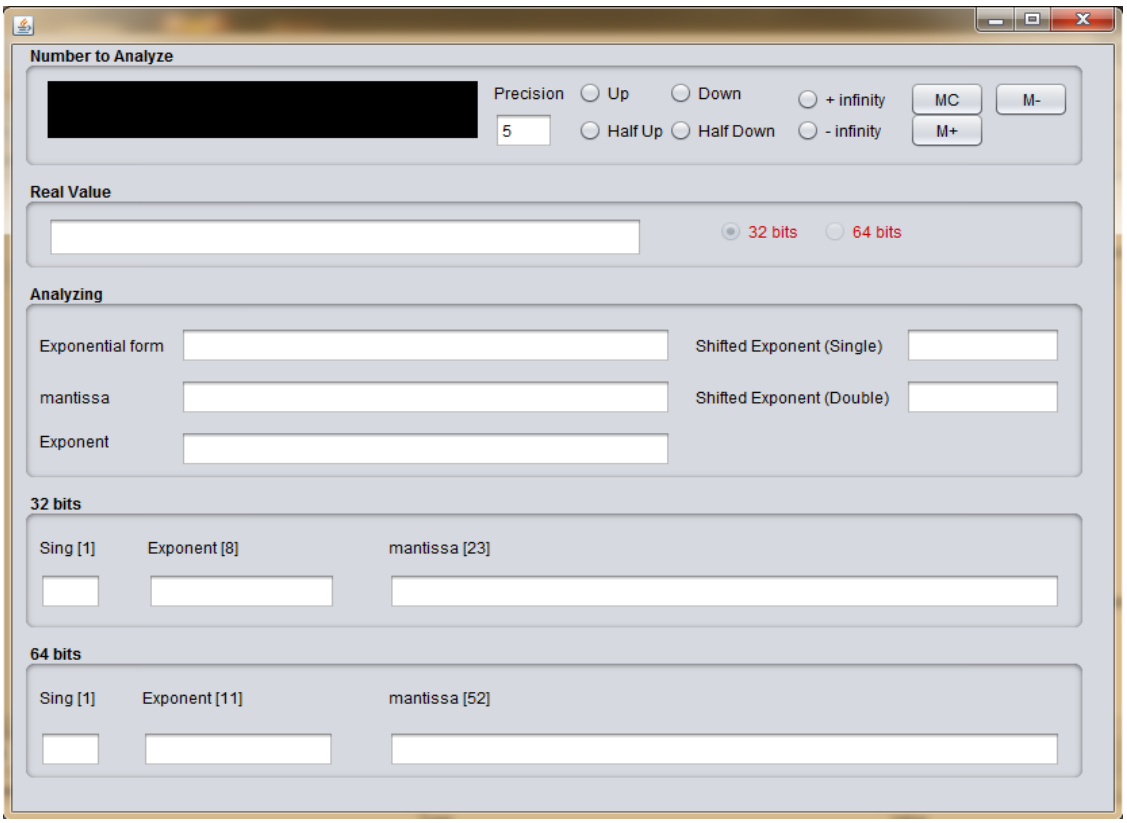
Μια σημαντική διαφοροποίηση από τις παραδοσιακές αριθμομηχανές, αποτελεί το πεδίο επεξεργασίας των μαθηματικών πράξεων. Το πεδίο αυτό δίνει την δυνατότητα στον χρήστη να πληκτρολογεί πολύπλοκες εκφράσεις οι οποίες στη συνέχεια υπολογίζονται αυτόματα. Η επεξεργασία του συγκεκριμένου πεδίου μπορεί να πραγματοποιηθεί είτε με τη χρήση των κουμπιών πληκτρολόγησης εκφράσεων του αντίστοιχου πεδίου της εφαρμογής, είτε με τη χρήση του πληκτρολογίου, όπου ο χρήστης εισάγει χειροκίνητα την έκφραση που θέλει να υπολογιστεί.

Εικόνα 10: Το τμήμα επεξεργασίας των μαθηματικών εκφράσεων

6.2 Το παράθυρο επεξήγησης αριθμών κινητής υποδιαστολής κατά IEEE

Όταν ο χρήστης το επιθυμεί, είναι σε θέση να εμφανιστεί ένα βοηθητικό παράθυρο το οποίο εξηγεί τη διαδικασία μετατροπής

και αναπαράστασης ενός αριθμού με τη μορφή κινητής υποδιαστολής απλής και διπλής ακρίβειας σύμφωνα με το πρότυπο της IEEE.



Εικόνα 11: Το Παράθυρο εμφανίσεις και επεξεργασίας των αριθμών με βάση το πρότυπο IEEE 754

Πιο συγκεκριμένα, το παράθυρο παρουσιάζει τη μετατροπή του αριθμού στην μορφή $\pm\alpha \cdot 2^b$ όπου $\alpha \in [1,2)$ και $b \in N$. Στη συνέχεια παρουσιάζεται ο αριθμός που αναπαριστά η mantissa (1-α), ο εκθέτης και ο πολωμένος εκθέτης, τόσο σε δεκαδική όσο και δυαδική μορφή. Τέλος, εμφανίζεται η δυαδική μορφή του αριθμού, χωρίζοντας με κενά τα επιμέρους τμήματα (πρόσημο, εκθέτης, mantissa). Ειδικότερα δίνει τη δυνατότητα στο χρήστη να επιλέξει την ακρίβεια του αριθμού που θα αναπαρασταθεί. Η default τιμή έχει καθοριστεί στα 5 δεκαδικά ψηφία ο χρήστης μπορεί να καθορίσει την τιμή που ο ίδιος επιθυμεί. Επίσης υπάρχει μια

πληθώρα δυνατοτήτων και επιλογών βάσει των οποίων καθορίζεται ο τρόπος με τον οποίο θα στρογγυλοποιηθεί ο αριθμός.

.Ο χρήστης μπορεί να επιλέγει μεταξύ των εξής τρόπων:

- ROUND_CEILLING
- ROUND_FLOOR
- ROUND_UP
- ROUND_HALF_UP
- ROUND_DOWN
- ROUND HALF_DOWH

Με τη μέθοδο ROUND_UP γίνεται στρογγυλοποίηση μακριά από το μηδέν.

Με τη μέθοδο ROUND_DOWN γίνεται στρογγυλοποίηση γύρω από το μηδέν.

Με τη μέθοδο ROUND_HALF_UP γίνεται στρογγυλοποίηση γύρω από το κοντινότερο γειτονικό αριθμό. Εκτός και εάν βρίσκεται μεταξύ δυο γειτονικών αριθμών σ' αυτή τη περίπτωση συμπεριφέρεται σαν τη ROUND_UP.

Με τη μέθοδο ROUND_HALF_DOWN γίνεται στρογγυλοποίηση γύρω από το κοντινότερο γειτονικό αριθμό. Εκτός και εάν βρίσκεται μεταξύ δυο γειτονικών αριθμών σ ' αυτή τη περίπτωση συμπεριφέρεται σαν τη ROUND_DOWN.

Με τη μέθοδο ROUND_CEILING γίνεται στρογγυλοποίηση προς το θετικό άπειρο. Εάν ο αριθμός είναι θετικός η μέθοδος

συμπεριφέρεται όπως και η ROUND_UP. Εάν ο αριθμός είναι αρνητικός, συμπεριφέρεται όπως και η ROUND_DOWN.
Με τη ROUND_FLOOR γίνεται στρογγυλοποίηση προς το αρνητικό άπειρο. Εάν ο αριθμός είναι θετικός ,η μέθοδος συμπεριφέρεται όπως και η ROUND_DOWN. Εάν ο αριθμός είναι αρνητικός συμπεριφέρεται όπως και η ROUND_UP.
Παρακάτω δίνεται παράδειγμα λειτουργίας των μεθόδων με κάποιους αριθμούς.

INPUT	ROUND_UP	ROUND_DOWN	ROUND_HALF_UP	ROUND_HALF_DOWN	ROUND_FLOOR	ROUND_CEILLING
7.6	8	8	8	8	8	7
7.5	8	7	8	7	8	7
7.4	7	7	7	7	8	7
-7.4	-7	-7	-7	-7	-7	-8
-7.5	-7	-8	-7	-7	-7	-8
-7.6	-8	-8	-8	-8	-7	-8

Πίνακας 3:Η στρογγυλοποίηση αριθμών με την χρήση των μεθόδων της BigDecimal

Το παράθυρο αυτό δίνει επίσης την δυνατότητα στο χρήστη να αποθηκεύσει στη μνήμη ή να αφαιρέσει από αυτή κάποιο προηγούμενο αποτελέσματα με σκοπό να το εμφανίσει στο πεδίο εκφράσεων του κεντρικού παραθύρου προκειμένου να εκτελέσει όποια πράξη ή πράξεις επιθυμεί.

6.3 Λεπτομέρειες υλοποίησης

6.3.1 Πληκτρολόγηση αριθμών και εισαγωγή πράξεων

Το πρώτο βήμα στην υλοποίηση της εφαρμογής αποτελεί ο χειρισμός των κουμπιών που περιέχουν αριθμούς (πλήκτρα 0-9, A-F), του πλήκτρου της υποδιαστολής, καθώς και των πλήκτρων που εκτελούν το σύνολο των πράξεων (απλών, επιστημονικών, boolean κλπ).

Ο χειρισμός αυτών των πλήκτρων είναι εφικτό να πραγματοποιηθεί με τους ακόλουθους τρόπους:

- Την εξομοίωση της απλής αριθμομηχανής, όπου οι πράξεις θα εκτελούνται με τη σειρά που εισάγονται,
- Την ενδιάμεση χρήση του πεδίου επεξεργασίας μαθηματικών εκφράσεων. Στη περίπτωση αυτή, κάθε πλήκτρο επιφέρει αλλαγές στο κείμενο του πλαισίου επεξεργασίας και η εκτέλεση των πράξεων πραγματοποιείται με βάση το κείμενο αυτό.

Για να επιτύχουμε την επιθυμητή λειτουργία, που προσφέρει δυνατότητες επεξεργασίας και εισαγωγής μαθηματικών εκφράσεων, ανεξάρτητα από τη πολυπλοκότητα τους, υιοθετήσαμε τη δεύτερη προσέγγιση.

Έτσι, κάθε πλήκτρο καλεί μια κατάλληλη συνάρτηση, τη συνάρτηση numberPressed με όρισμα τον αριθμό, τη συνάρτηση ή το σύμβολο που θα εισαχθεί στη μαθηματική έκφραση του πεδίου επεξεργασίας.

Πιο συγκεκριμένα, παρακάτω παρουσιάζεται ο κώδικας ενός τυχαίου πλήκτρου και γίνεται σχολιασμός του κώδικα της συνάρτησης numberPressed:

```
private void cmdNumber5ActionPerformed(java.awt.event.ActionEvent
evt) {
   numberPressed("5");
}
```

Κώδικας 1:Το Πλήκτρο 5

```
private void numberPressed(String i) {
      String expr = txtExpression.getText();
      int curPos = txtExpression.getCaretPosition();
      String newExpr = expr.substring(0,curPos) + i
            + expr.substring(curPos);
      txtExpression.setText(newExpr);
      txtExpression.setCaretPosition(curPos+i.length());
      performCalculation();
   }
```

Κώδικας 2:Η μέθοδος nmberPressed(String)

Η συνάρτηση numberPressed, αρχικά λαμβάνει το κείμενο που περιέχει το πεδίο επεξεργασίας των μαθηματικών εκφράσεων και επιχειρεί να το προσθέσει. Στην απλοϊκή προσέγγιση η προσθήκη θα πραγματοποιούταν στο τέλος. Έχουμε ήδη αναφέρει, όμως, ότι ο χρήστης θα έχει τη δυνατότητα να τροποποιεί τις μαθηματικές εκφράσεις. Για να είναι εφικτό να επιτευχθεί η διαγραφή και η εισαγωγή των δεδομένων σε οποιοδήποτε σημείο του πεδίου επεξεργασίας, λαμβάνεται υπόψη η θέση του δείκτη (το σημείο στο οποίο αναβοσβήνει ο cursor) με χρήση της συνάρτησης getCaretPosition. Έτσι, με αξιοποίηση των συναρτήσεων substring, τροποποιούμε κατάλληλα την επίμαχη μαθηματική έκφραση.

6.3.2 Ειδικά πλήκτρα

Στην εφαρμογή που έχουμε αναπτύξει, ορισμένα από τα πλήκτρα της αριθμομηχανής έχουν ιδιαίτερη λειτουργία. Μερικά από αυτά είναι τα πλήκτρα της μνήμης καθώς και το πλήκτρο Backspace. Για τις λειτουργίες της μνήμης θα γίνει αναφορά πιο αναλυτικά σε επόμενο υποκεφάλαιο.

Το πλήκτρο Backspace παρέχει τη δυνατότητα να διορθώνονται οι μαθηματικές εκφράσεις του πεδίου επεξεργασίας. Η αρχή λειτουργίας του είναι απλή: Όταν πληκτρολογείται το πλήκτρο backspace, διαγράφεται ένας χαρακτήρας από το σημείο που βρίσκεται ο δείκτης στο πεδίο επεξεργασίας. Για να επιτύχουμε την συγκεκριμένη λειτουργία, αναπτύξαμε τον ακόλουθο κώδικα:

```
private void
    ActionPerformed(java.awt.event.ActionEvent evt) {

    String expr = txtExpression.getText();
    int curPos = txtExpression.getCaretPosition();
    String newExpr = expr.substring(0,curPos-1)  +
          expr.substring(curPos);
    txtExpression.setText(newExpr);
    txtExpression.setCaretPosition(curPos-1);
    performCalculation();
}
```

Κώδικας 3:Το πλήκτρο backspace

Η λειτουργία του παραπάνω κώδικα ακολουθεί ακριβώς την ίδια φιλοσοφία που ακολουθεί και η συνάρτηση numberPressed, με τη διαφορά ότι η συγκεκριμένη συνάρτηση σβήνει έναν χαρακτήρα από το κείμενο αντί να τον εισάγει.

6.3.3 Πλήκτρα C, CE,± και =

Ο υποψήφιος χρήστης της εφαρμογής θα παρατηρήσει σίγουρα ότι από το γραφικό περιβάλλον εργασίας απουσιάζουν εντελώς τέσσερα πολύ συνηθισμένα πλήκτρα: Το πλήκτρο της ολικής διαγραφής όλων των αποτελεσμάτων και της τρέχουσας πληκτρολόγησης (πλήκτρο C), το πλήκτρο διαγραφής της τρέχουσας πληκτρολόγησης (πλήκτρο CE), το πλήκτρο αλλαγής πρόσημου του αριθμού καθώς και το πλήκτρο που παρουσιάζει στο χρήστη το τελικό αποτέλεσμα.

Η απουσία των συγκεκριμένων πλήκτρων είναι εσκεμμένη. Με την φιλοσοφία που έχουμε υιοθετήσει, ο χρήστης μπορεί να επεξεργάζεται την έκφραση κατά το δοκούν, να τροποποιεί αριθμούς, τελεστές, συναρτήσεις κοκ, με την απευθείας επεξεργασία του κατάλληλου πεδίου. Επομένως, η παρουσία των πλήκτρων C, CE καθίσταται περιττή.

Ομοίως, η αλλαγή του πρόσημου του αριθμού πραγματοποιείται με τη πληκτρολόγηση του συμβόλου της αφαίρεσης (-) πριν από τον αριθμό.

Επιπρόσθετα, προσθέσαμε στο πεδίο επεξεργασίας της μαθηματικής έκφρασης κατάλληλο ακροατή, ο οποίος μόλις ανιχνεύσει οποιαδήποτε μεταβολή στην μαθηματική έκφραση, ενεργοποιεί τη συνάρτηση υπολογισμού της. Με τη υιοθέτηση της συγκεκριμένη συμπεριφοράς, η παρουσία του πλήκτρου της ισότητας (=) είναι πια πλεονασμός.

6.3.4 Εκτέλεση πράξεων

Η εκτέλεση των πράξεων γίνεται αυτόματα μετά από κάθε τροποποίηση της μαθηματικής έκφρασης στο πεδίο επεξεργασίας. Η εκτέλεση των πράξεων που υιοθετήθηκε, βασίζεται στον υπολογισμό των εκφράσεων με βάση την αντίστροφη πολωνική γραφή τους. Πιο συγκεκριμένα, ο αλγόριθμος υπολογισμού περιλαμβάνει τα ακόλουθα βήματα:

1. Διαχωρισμός της μαθηματικής έκφρασης σε ενδοθεματική ακολουθία συμβόλων. Παράλληλα εκτελείται έλεγχος ακρίβειας των συμβόλων που έχουν εισαχθεί (έλεγχος ορθογραφίας) και στη συνέχεια πρωταρχικός έλεγχος ορθότητας της έκφρασης: Έλεγχος παρενθέσεων.
2. Μετατροπή της ενδοθεματικής έκφρασης σε αντίστροφη πολωνική γραφή. Παράλληλα εκτελείται ένας ακόμη έλεγχος ορθότητας: Πληρότητα τελεστών.
3. Υπολογισμός της έκφρασης με βάση την αντίστροφη πολωνική γραφή. Παράλληλα εκτελείται ο τελικός έλεγχος ορθότητας της έκφρασης.

Στη συνέχεια του κεφαλαίου θα παρουσιάσουμε αναλυτικά τις παραπάνω διαδικασίες.

Πριν προχωρήσουμε στην ανάλυση, θα γίνει παρουσίαση των απαραίτητων δομών που υλοποιήθηκαν για να υποβοηθήσουν τις παραπάνω διαδικασίες:

1. Απαρίθμηση MathematicalOperator: Η απαρίθμηση MathematicalOperator περιλαμβάνει όλους του πιθανούς τελεστές που μπορεί να έχει η μαθηματική έκφραση. Πιο συγκεκριμένα, οι πιθανοί τελεστές είναι οι εξής:
    1. PLUS: Αντιπροσωπεύει τον τελεστή + της πρόσθεσης

2. MINUS: Αντιπροσωπεύει τον τελεστή – της αφαίρεσης
3. MULTIPLICATION: Αντιπροσωπεύει τον τελεστή * του πολλαπλασιασμού
4. DIVISION: Αντιπροσωπεύει τον τελεστή / της διαίρεσης
5. MODULO: : Αντιπροσωπεύει τον τελεστή % για το υπόλοιπο της διαίρεσης ακεραίων
6. POWER: Αντιπροσωπεύει τον τελεστή ^ της ύψωσης σε δύναμη
7. SQRT: Αντιπροσωπεύει τον τελεστή της ρίζας σε δύναμη
8. SIN: Αντιπροσωπεύει τον τελεστή του ημιτόνου
9. COS: Αντιπροσωπεύει τον τελεστή του συνημιτόνου
10. TAN: Αντιπροσωπεύει τον τελεστή της εφαπτομένης
11. ASIN: Αντιπροσωπεύει τον τελεστή του αντίστροφου ημιτόνου
12. ACOS: Αντιπροσωπεύει τον τελεστή του αντίστροφου συνημιτόνου
13. ATAN: Αντιπροσωπεύει τον τελεστή της αντίστροφης εφαπτομένης
14. SINH: Αντιπροσωπεύει τον τελεστή του υπερβολικού ημιτόνου
15. COSH: Αντιπροσωπεύει τον τελεστή του υπερβολικού συνημιτόνου
16. TANH: Αντιπροσωπεύει τον τελεστή της υπερβολικής εφαπτομένης
17. EXP: Αντιπροσωπεύει τον τελεστή του e^x
18. LOG: : Αντιπροσωπεύει τον τελεστή του δεκαδικού λογαρίθμου
19. LN: Αντιπροσωπεύει τον τελεστή του φυσικού (νεπέριου) λογαρίθμου
20. LSH: Αντιπροσωπεύει τον τελεστή της αριστερής ολίσθησης κατά 1 bit

21. RSH: Αντιπροσωπεύει τον τελεστή της δεξιάς ολίσθησης κατά 1 bit
22. FACTORIAL: : Αντιπροσωπεύει τον τελεστή ! του παραγοντικού
23. BIN: Αντιπροσωπεύει τον τελεστή για την εισαγωγή δυαδικών εκφράσεων
24. HEX: Αντιπροσωπεύει τον τελεστή για την εισαγωγή δεκαεξαδικών εκφράσεων
25. OCT: Αντιπροσωπεύει τον τελεστή για την εισαγωγή οκταδικών εκφράσεων
26. AND: Αντιπροσωπεύει τον δυαδικό τελεστή AND
27. OR: Αντιπροσωπεύει τον δυαδικό τελεστή OR
28. XOR: Αντιπροσωπεύει τον δυαδικό τελεστή XOR
29. NOT: Αντιπροσωπεύει τον δυαδικό τελεστή NOT

2. Αντικείμενα MathematicElement: Με τη κλάση αυτή περιγράφουμε κάθε πιθανό μαθηματικό στοιχείο που μπορεί να περιέχει η μαθηματική έκφραση: Αριθμός, τελεστής, παρένθεση. ¨Στον Κώδικα 4 φαίνεται η υλοποίηση της κλάσης. Περιλαμβάνει 4 δυαδικές μεταβλητές (boolean) οι οποίες περιγράφουν το είδος του μαθηματικού στοιχείου που αντιπροσωπεύουν, μία μεταβλητή διπλής ακριβείας για την αναπαράσταση δυαδικών, οκταδικών και δεκαδικών αριθμών, μία μεταβλητή String για την αναπαράσταση δεκαεξαδικών αριθμών και μία μεταβλητή τύπου MathematicalOperator για την αναπαράσταση τελεστών. Τέλος, υλοποιούνται και δύο κατασκευαστές της κλάσης (constructors) οι οποίοι χειρίζονται την αρχικοποίηση των μεταβλητών κατά τη δημιουργία νέων αντικειμένων. Ο πρώτος, που δε δέχεται ορίσματα, βάζει όλες τις δυαδικές μεταβλητές ίσες με false, τους αριθμούς 0, και τον τελεστή NONE. Έτσι, το αρχικοποιημένο αντικείμενο δεν αντιπροσωπεύει κανένα μαθηματικό στοιχείο. Ο δεύτερος,

δέχεται σαν όρισμα μια μεταβλητή τύπου double και αρχικοποιεί το αντικείμενο σαν μαθηματικό στοιχείο αριθμού με τιμή ίδια με το όρισμα.

3. Τέλος, για το χειρισμό των σφαλμάτων των μαθηματικών εκφράσεων που εισάγονται από το χρήστη, κατασκευάσαμε κατάλληλη κλάση Εξαιρέσεων (Exception), αποκλειστικά για την εφαρμογή μας. Τη κλάση Mathematical Formula Exception.

```
class MathematicElement {

   public double number;
   public String myHexNumber;
   public boolean isOperator, isOpeningParenthesis,
              isClosingParenthesis, isNumber;
   public MathematicalOperator operator;

   public MathematicElement() {
      isOperator = isOpeningParenthesis =
             isClosingParenthesis = isNumber = false;
      operator = MathematicalOperator.NONE;
      number = 0;
      myHexNumber = "";
   }

   public MathematicElement(double x) {
      isOperator = isOpeningParenthesis =
                isClosingParenthesis = false;
      isNumber = true;
      operator = MathematicalOperator.NONE;
      number = x;
      myHexNumber = "";
   }
```

Κώδικας 4: Η κλάση MathematicElement

6.3.4.1 Διαχωρισμός της μαθηματικής έκφρασης

Ο διαχωρισμός των μαθηματικών εκφράσεων σε επιμέρους μαθηματικά στοιχεία πραγματοποιείται με τη χρήση τεσσάρων επιμέρους συναρτήσεων οι οποίες λειτουργούν ως εξής:

- Η πρώτη συνάρτηση ελέγχει αν μια δεδομένη συμβολοσειρά είναι έγκυρο μαθηματικό στοιχείο. Στη συγκεκριμένη υλοποίηση, ελέγχουμε αν το στοιχείο ισούται με κάποιο μαθηματικό τελεστή ή οποιαδήποτε από τις υποστηριζόμενες συναρτήσεις. Τέλος, ελέγχουμε αν πρόκειται για αριθμό, σύμφωνα με το σύστημα αρίθμησης (δυαδικό, οκταδικό, δεκαδικό ή δεκαεξαδικό).
- Η δεύτερη συνάρτηση από δεδομένη έκφραση, εξάγει το πρώτο έγκυρο μαθηματικό στοιχείο. Ο συγκεκριμένος κώδικας αρχικά ελέγχει αν το πρώτο γράμμα είναι έγκυρο. Στη συνέχεια σταδιακά αυξάνει το μέγεθος της ελεγχόμενης συμβολοσειράς, και τελικά επιστρέφει τη μέγιστη έγκυρη. Κατάλληλη τροποποίηση έχει εισαχθεί στην αρχή της συνάρτησης ώστε ο έλεγχος της εγκυρότητας να ξεκινάει από δεδομένο σημείο της συμβολοσειράς εισόδου. Αυτό πραγματοποιείται με την εξαγωγή του κατάλληλου substring και στη συνέχεια με τη χρήση αυτού.
- Η τρίτη συνάρτηση μετατρέπει μια έγκυρη συμβολοσειρά σε αντικείμενο μαθηματικό στοιχείο. Η συγκεκριμένη υλοποίηση κάνει διαδοχικούς ελέγχους, για κάθε έγκυρη τιμή επιστρέφει και το κατάλληλο αντικείμενο. Σε περίπτωση σφάλματος ενεργοποιείται Exception.
- Η τέταρτη συμβολοσειρά είναι ο ενορχηστρωτής της διαδικασίας, που καλώντας τις κατάλληλες συναρτήσεις, διασπά μια ολόκληρη μαθηματική έκφραση σε ένα πίνακα από μαθηματικά στοιχεία με βάση την σειρά τους στη δεδομένη έκφραση. Ο συγκεκριμένος κώδικας καλεί

κατάλληλα τη συνάρτηση εξαγωγής του επόμενου έγκυρου στοιχείου και εκτελεί βασικούς έλεγχους ορθότητας. Αν βρεθεί μη έγκυρο στοιχείο στη συμβολοσειρά, ενεργοποιείται κατάλληλο Exception που αποστέλλεται στο GUI.

6.3.4.2 Μετατροπή της ενδοθεματικής σε αντίστροφη πολωνική γραφή

Η ενδοθεματική γραφή μετατρέπεται σε αντίστροφη πολωνική σύμφωνα με τον αλγόριθμο Shunting-yard, που παρουσιάσαμε σε προηγούμενο κεφάλαιο. Η υλοποίηση του φαίνεται στον Κώδικα 10. Η συνάρτηση αυτή λαμβάνει σαν είσοδο μια λίστα με τα μαθηματικά στοιχεία σε ενδοθεματική μορφή και επιστρέφει τη σειρά των συμβόλων στην αντίστροφη πολωνική γραφή. Η συνάρτηση αυτή παράλληλα επιτελεί έλεγχο ορθότητας της ακολουθίας (αν δηλαδή υπάρχουν όλοι οι τελεστές που απαιτούνται). Για τη συγκεκριμένη υλοποίηση, απαιτείται η συνάρτηση υπολογισμού της προτεραιότητας του κάθε τελεστή

```
private static int getPriority(MathematicElement m)
            throws MathematicalFormulaException {
    if (m.isOpeningParenthesis) {
        return 10;
    }
    if (m.isClosingParenthesis) {
        return 0;
    }
    if (!m.isOperator) {
        throw new MathematicalFormulaException
                ("Uncompleted expression");
    }
```

```
        switch (m.operator) {
            case ACOS: case ASIN:
            case ATAN: case COS:
            case COSH: case EXP:
            case LN:   case LOG:
            case LSH:  case RSH:
            case SIN:
            case TAN:
            case TANH:
            case HEX:
            case BIN:
            case OCT:
            case NOT:
                return 50;
            case POWER:
            case MODULO:
                return 40;
            case MULTIPLICATION:
            case DIVISION:
            case AND:
                return 30;
            case PLUS:
            case MINUS:
            case OR:
            case XOR:
                return 20;

        }
        System.out.println(m);
        throw new
              MathematicalFormulaException("Uncompleted
                                    expression");
    }
```

Κώδικας 5: Η μέθοδος getPriority(MathematicElement)

```
private static MathematicElement[] ConvertInfixToPostfix (
```

```
            MathematicElement[] infix) throws
               MathematicalFormulaException {
      try {
         ArrayList<MathematicElement> postfixList =
                 new ArrayList<MathematicElement>();
         Stack<MathematicElement> stack = new
                      Stack<MathematicElement>();
         for (int i = 0; i < infix.length; i++) {
            if (infix[i].isNumber ||
                (infix[i].isOperator &&
                   infix[i].operator ==
                   MathematicalOperator.FACTORIAL)){
               postfixList.add(infix[i]);
               continue;
            }
            if (infix[i].isOpeningParenthesis) {
               stack.push(infix[i]);
               continue;
            }
            if (infix[i].isClosingParenthesis) {
               while (!
                (stack.peek().isOpeningParenthesis)){
                  postfixList.add(stack.pop());
               }
               stack.pop();
               continue;
            }
            while (!(stack.empty() ||
                    getPriority(infix[i]) >
                    getPriority(stack.peek()))) {
               postfixList.add(stack.pop());
            }
            stack.push(infix[i]);
         }
         while (!(stack.empty())) {
            postfixList.add(stack.pop());
         }
         return postfixList.toArray(new
```

```
                MathematicElement[postfixList.size()]);
        } catch (Exception xe) {
            throw new
              MathematicalFormulaException(xe.toString());
        }
    }
```

Κώδικας 6: Υλοποίηση του αλγορίθμου Shunting-yard

6.4.4.3 Υπολογισμός παραστάσεων σε αντίστροφη πολωνική μορφή

Το τελευταίο βήμα της διαδικασίας αποτελεί ο υπολογισμός της μαθηματικής έκφρασης, η οποία έχει μετατραπεί σε αντίστροφη πολωνική μορφή. Ο υπολογισμός βασίζεται στον ακόλουθο αλγόριθμο. Η ανάγνωση των συμβόλων γίνεται σειριακά ξεκινώντας από αριστερά:

1. Διαβάζουμε το επόμενο σύμβολο
2. Αν είναι αριθμός, το τοποθετούμε στη στοίβα
3. Αν είναι τελεστής, τότε
    1. Ελέγχουμε πόσους τελεστές χρειάζεται (στην περίπτωσή μας είναι 1, για τις συναρτήσεις και 2 για τις βασικές δεκαδικές και δυαδικές πράξεις)
    2. Εξάγουμε από τη στοίβα το αντίστοιχο πλήθος αριθμών.
    3. Εκτελούμε τη πράξη
    4. Εισάγουμε το αποτέλεσμα στη στοίβα
4. Αν υπάρχουν επιπλέον σύμβολα, επανερχόμαστε στο βήμα 1
5. Η στοίβα έχει μόνο ένα αποτέλεσμα, το ζητούμενο

Η υλοποίηση του παραπάνω αλγορίθμου χρησιμοποιεί τις συναρτήσεις που αναπτύξαμε οι οποίες βασίζονται στις σειρές

Taylor. Στο ακόλουθο πλαίσιο φαίνεται η υλοποίηση του αλγορίθμου.

```
private static double CalculatePostfixExpression
        (MathematicElement[] post,
         MathematicalNumberingSystem system)
            throws MathematicalFormulaException {
    Stack<MathematicElement> stack = new
                Stack<MathematicElement>();
    for (int i = 0; i < post.length; i++) {
       if (post[i].isNumber) {
          stack.push(post[i]);
          continue;
       }
       if (post[i].isOperator) {
          MathematicElement mx2 = stack.pop();
          double modifier;
          if(system==
              MathematicalNumberingSystem.DEGREES)
             modifier = pi/180;
          else
             modifier = 1;
          double x2 = mx2.number, x1, y;
          switch (post[i].operator) {
             case PLUS:
                x1 = stack.pop().number;
                y = x1 + x2;
                stack.push(new
                        MathematicElement(y));
                continue;
             case MINUS:
                x1 = stack.pop().number;
                y = x1 - x2;
                stack.push(new
                        MathematicElement(y));
                continue;
             case MULTIPLICATION:
```

```
            x1 = stack.pop().number;
            y = x1 * x2;
            stack.push(new
                    MathematicElement(y));
            continue;
        case DIVISION:
            x1 = stack.pop().number;
            y = x1 / x2;
            stack.push(new
                    MathematicElement(y));
            continue;
        case MODULO:
            x1 = stack.pop().number;
            y = ((int) Math.rint(x1)) %
                    ((int) Math.rint(x2));
            stack.push(new
                    MathematicElement(y));
            continue;
        case POWER:
            x1 = stack.pop().number;
            y = mypow(x1, x2);
            stack.push(new
                    MathematicElement(y));
            continue;
        case SIN:
            y = mySin(x2*modifier);
            stack.push(new
                    MathematicElement(y));
            continue;
        case COS:
            y = myCos(x2*modifier);
            stack.push(new
                    MathematicElement(y));
            continue;
        case TAN:
            y = myTan(x2*modifier);
            stack.push(new
                    MathematicElement(y));
```

```
            continue;
        case ASIN:
            y = myArcsin(x2)/modifier;
            stack.push(new
                    MathematicElement(y));
            continue;
        case ACOS:
            y = myArccos(x2)/modifier;
            stack.push(new
                    MathematicElement(y));
            continue;
        case ATAN:
            y = myArctan(x2)/modifier;
            stack.push(new
                    MathematicElement(y));
            continue;
        case SINH:
            y = mySinh(x2);
            stack.push(new
                    MathematicElement(y));
            continue;
        case COSH:
            y = myCosh(x2);
            stack.push(new
                    MathematicElement(y));
            continue;
        case TANH:
            y = myTanh(x2);
            stack.push(new
                    MathematicElement(y));
            continue;
        case EXP:
            y = myExp(x2);
            stack.push(new
                    MathematicElement(y));
            continue;
        case LOG:
            y = mylog(x2);
```

```
                stack.push(new
                        MathematicElement(y));
                continue;
            case LN:
                y = myln(x2);
                stack.push(new
                        MathematicElement(y));
                continue;
            case LSH:
              if(Math.abs(x2-Math.rint(x2))<1e-5)
                {
                   y = x2 * 2;
                } else {
                   long d =
                     Double.doubleToLongBits(e);
                   d *= 2;
                   y =
                     Double.longBitsToDouble(d);
                }
                stack.push(new
                        MathematicElement(y));
                continue;
            case RSH:
              if(Math.abs(x2-Math.rint(x2))<1e-5)
                {
                   int d = (int) x2;
                   d /= 2;
                   y = d;
                } else {
                   long d =
Double.doubleToLongBits(e);
                   d *= 2;
                   y = Double.longBitsToDouble(d);
                }
                stack.push(new MathematicElement(y));
                continue;
            case FACTORIAL:
                if (Math.abs(x2 - Math.rint(x2)) < 1e-5) {
```

```
                    y = myFactorial((int) x2);
                    stack.push(new MathematicElement(y));
                    continue;
                } else {
                    throw new MathematicalFormulaException("Factorial
function requires possitive integer numbers");
                }
            case AND:
                x1 = stack.pop().number;
                if ((Math.abs(x2 - Math.rint(x2)) < 1e-5) && (Math.abs(x1
- Math.rint(x1)) < 1e-5)) {
                    y = ((int) x1) & ((int) x2);
                    stack.push(new MathematicElement(y));
                    continue;
                } else {
                    y
Double.longBitsToDouble(Double.doubleToLongBits(x1)
Double.doubleToLongBits(x2));
                    stack.push(new MathematicElement(y));
                    continue;
                }
            case OR:
                x1 = stack.pop().number;
                if ((Math.abs(x2 - Math.rint(x2)) < 1e-5) && (Math.abs(x1
- Math.rint(x1)) < 1e-5)) {
                    y = ((int) x1) | ((int) x2);
                    stack.push(new MathematicElement(y));
                    continue;
                } else {
                    y=
Double.longBitsToDouble(Double.doubleToLongBits(x1)                    |
Double.doubleToLongBits(x2));
                    stack.push(new MathematicElement(y));
                    continue;
                }
            case XOR:
                x1 = stack.pop().number;
                if ((Math.abs(x2 - Math.rint(x2)) < 1e-5) && (Math.abs(x1
```

```
- Math.rint(x1)) < 1e-5)) {
                    y = ((int) x1) ^ ((int) x2);
                    stack.push(new MathematicElement(y));
                    continue;
                } else {
                    y=Double.longBitsToDouble(
                     Double.doubleToLongBits(x1) ^
                     Double.doubleToLongBits(x2));
                    stack.push(new
                          MathematicElement(y));
                    continue;
                }

case OCT: {
                    int xx = (int) Math.rint(x2);
                    int yy = myOct(xx);
                    stack.push(new
                            MathematicElement(yy));
                    continue;
                }
                case BIN: {
                    int xx = (int) Math.rint(x2);
                    int yy = myBin(xx);
                    stack.push(new
                            MathematicElement(yy));
                    continue;
                }
                case HEX:
                    y = myHex(mx2.myHexNumber);
                    stack.push(new
                             MathematicElement(y));
            }
         }
      }

      return stack.peek().number;
   }
```

Κώδικας 7: Υπολογισμός τις αντίστροφης πολυωνιμικής

6.4.5 Διαδικασίες Μνήμης

Στην επιστημονική αριθμομηχανή που αναπτύξαμε, έχουμε προσδώσει τη δυνατότητα να αποθηκεύει ενδιάμεσα αποτελέσματα στη μνήμη, όπως γίνεται με τις περισσότερες αριθμομηχανές.

Για να επιτύχουμε τη συγκεκριμένη συμπεριφορά, ορίσαμε μια επιπλέον μεταβλητή, την μεταβλητή memory, η οποία αποθηκεύει τα ενδιάμεσα αποτελέσματα.

Ο χρήστης έχει τη δυνατότητα να προσθέσει το υπάρχον αποτέλεσμα στον αριθμό που υπάρχει στη μνήμη (πλήκτρο M+), να αντιστρέψει τον αριθμό και να τον αποθηκεύσει στη μνήμη (πλήκτρο M-), να ανακαλέσει το αποτέλεσμα της μνήμης διατηρώντας τον αριθμό στην μνήμη (πλήκτρο MR) και τέλος να διαγράψει από τη μνήμη το αποθηκευμένο αποτέλεσμα (πλήκτρο MC).

Ο κώδικας των πράξεων των πλήκτρων είναι ο ακόλουθος:

```
//M+
{
      try {
          String d = Rounded.getText();
          this.memory = d;
       } catch (Exception ex) {
          return;
       }}
//M-
{
          try {
          String d = Rounded.getText();
          this.memory ="-"+ d;
       } catch (Exception ex) {
```

```
            return;
        }
 //MR
 {
           double a;
          int integer =0;
          if(k.Rounded.getText().length()!=0)
          {

           a=Double.parseDouble(k.memory);
           if(a==Math.round(a))
           {
            integer=(int)a;
             numberPressed(Integer.toString(integer));
           }
           else
           {
               numberPressed(Double.toString(a));
           }
 }
 //MC
 {
        memory = "";
 }
```

Κώδικας 8:Υλοποίηση αλγορίθμου για την προσθήκη αποτελεσμάτων στην μνήμη

6.3.6 Προβολή μη δεκαδικών αριθμητικών συστημάτων

Κατά την αλλαγή των αριθμητικών συστημάτων η εσωτερική αναπαράσταση των αριθμών δεν τροποποιείται. Αντίθετα, αλλάζει ο τρόπος εμφάνισης των αριθμών, οι οποίοι εμφανίζονται σύμφωνα με τις επιλογές του χρήστη.

Όσον αφορά τη προβολή των αριθμών με βάση το δεκαδικό σύστημα, χρησιμοποιούνται οι βασικές συναρτήσεις μετατροπής των αριθμών σε συμβολοσειρές (String), οι οποίες ανατίθενται στο κατάλληλο πεδίο της οθόνης.

Σχετικά με τη προβολή των αριθμών με βάση στο δυαδικό, οκταδικό και δεκαεξαδικό σύστημα, δύο διαφορετικές περιπτώσεις διακρίνονται:

- Ακέραιων αριθμών: Στη περίπτωση αυτή, ο αριθμός μετατρέπεται σε Long και στη συνέχεια καλείται η συνάρτηση toBinaryString(), που έχει η βασική κλάση Long της Java. Έτσι, προκύπτει η δυαδική αναπαράσταση η οποία στη συνέχεια μετατρέπεται (ανάλογα την επιλογή του χρήστη) σε οκταδική ή δεκαεξαδική μορφή

- Δεκαδικών αριθμών: Στη περίπτωση αυτή, ο αριθμός μετατρέπεται με κατάλληλη συνάρτηση σε Longbits και ακολουθείται η παραπάνω διαδικασία εκ νέου.

6.3.7 Βελτιστοποίηση υπολογισμών σχέσεων Taylor - Υπολογισμοί με αναδρομικό τύπο

Μετά από πειραματισμό κατά τις πρώτες εκδόσεις της εφαρμογής, διαπιστώθηκε ότι η χρονική διάρκεια η οποία απαιτείται για τους υπολογισμούς των σχέσεων Taylor ήταν εξαιρετικά μεγάλη και η καθυστέρηση ήταν εμφανής κατά τη χρήση της εφαρμογής. Η βασική αιτία του μεγάλου χρόνου εκτέλεσης εντοπίστηκε στο τμήμα υπολογισμού των σειρών Taylor με επαναληπτική διαδικασία και συγκεκριμένα, στον υπολογισμό της

τιμής της δύναμης και της τιμής του παραγοντικού σε κάθε επανάληψη. Η συγκεκριμένη διαδικασία αποδεικνύεται ότι είναι εξαιρετικά χρονοβόρα και επίπονη, και για αυτόν το λόγο κινηθήκαμε σε εύρεση εναλλακτικών μορφών υπολογισμού των συγκεκριμένων σειρών.

Σημαντική βοήθεια προς τη κατεύθυνση αυτή πρόσφερε η εξής παρατήρηση: δύο τυχαίοι διαδοχικοί όροι της σειράς Taylor ακολουθούν ένα συγκεκριμένο πρότυπο και συνεπώς είναι εφικτός ο υπολογισμός του επόμενου όρου της σειράς με βάση το προηγούμενο και μικρό υπολογιστικό κόστος. Με απλά λόγια, με βάση τους τύπους της σειράς Taylor, είναι εφικτό να εξάγουμε αναδρομικό υπολογιστικό τύπο, ο οποίος έχει αισθητά μικρότερο υπολογιστικό κόστος.

Έτσι, ο υπολογισμός των τύπων του Taylor πραγματοποιήθηκε με αναδρομικό υπολογιστικό τύπο, ο οποίος εξάγεται από τη σειρά σύμφωνα με την ακόλουθη διαδικασία

1. Εκφράζουμε τον τύπο του Taylor της επιθυμητής συνάρτησης σαν άθροισμα δηλαδή, άπειρων όρων μιας ακολουθίας.
2. Λαμβάνουμε την ποσότητα a_n,
3. Υπολογίζουμε την ποσότητα a_{n+1},
4. Στη συνέχεια υπολογίζουμε τον $\frac{a_{n+1}}{a_n}$,
5. Εξαιτίας της ύπαρξης παραγοντικών και διαδοχικών δυνάμεων του x μέσα στις ποσότητες a_{n+1} και a_n, ο λόγος που υπολογίζεται στο βήμα 4, δε περιέχει δυνάμεις στην n-οστή δύναμη, ούτε υπολογισμούς παραγοντικών. Έτσι, μπορούμε να εκφράσουμε τη ποσότητα a_{n+1} σα συνάρτηση της ποσότητας a_n με βάση τον προηγουμένως υπολογισμένο λόγο.

Στο παράδειγμα που ακολουθεί, υπολογίζεται αναλυτικά ο αναδρομικός τύπος για δύο περιπτώσεις: Της συνάρτησης e^x και της συνάρτησης sinx. Οι δύο αυτές συναρτήσεις έχουν επιλεγεί με σκοπό να δείξουμε τις διαφορές που προκύπτουν όταν η μεταβολή των όρων της σειράς Taylor είναι με μοναδιαίο βήμα (e^x) αλλά και με διπλό (sinx).

6.3.7.1 Βελτιστοποίηση εκθετικής συνάρτησης

- Αρχικά εξάγουμε τον όρο $a_n = \frac{x^n}{n!}$
- Στην συνέχεια υπολογίζουμε τον όρο $a_{n+1} = \frac{x^{n+1}}{(n+1)!}$
- Τέλος υπολογίζουμε τον λόγο $\frac{a_{n+1}}{a_n} = \frac{x}{n+1}$
- Άρα, για n=0 ισχύει ότι $a_0 = 1$ και για n>0 ισχύει ότι $a_{n+1} = \frac{x}{n+1} a_n$

Έτσι, ο κώδικας της εκθετικής συνάρτησης βελτιστοποιείται ως εξής:

```
public static double MyExp(double x)
{
      double result = 1;
      int i = 1;
      double term = x;
      while (term <-maxError || term > maxError)
      {
         result+=term;
         i++;
         term*=x/i;
      }
      return result;
}
```

Κώδικας 9:Η βελτιστοποιημένη εκθετική συνάρτηση

6.3.7.2 Βελτιστοποίηση της συνάρτησης του ημιτόνου

- Αρχικά εξάγουμε τον όρο $a_n = (-1)^n \frac{x^{2n+1}}{(2n+1)!}$
- Στην συνέχεια υπολογίζουμε τον όρο $a_{n+1} = (-1)^{n+1} \frac{x^{2n+3}}{(2n+3)!}$
- Τέλος υπολογίζουμε τον λόγο $\frac{a_{n+1}}{a_n} = \frac{-x^2}{(2n+2)(2n+3)}$
- Άρα, για n=0 ισχύει ότι $a_0 = x$ και για n>0 ισχύει ότι $a_{n+1} = \frac{-x^2}{(2n+2)(2n+3)} a_n$

Παρατηρούμε ότι ο βελτιστοποιημένος κώδικας:

- Δεν καλεί άλλες συναρτήσεις.
- Διαθέτει μόνο μια επανάληψη

Αντίθετα, η απλοϊκή υλοποίηση, σε κάθε επανάληψη καλεί δύο συναρτήσεις, η πρώτη εκ των οποίων περιέχει επανάληψη, ενώ η δεύτερη είναι αναδρομική. Προφανώς και η πολυπλοκότητα της υλοποίησης έχει μειωθεί από $O(n^2)$ σε πολυπλοκότητα $O(n)$.

Έτσι, ο κώδικας της συνάρτησης βελτιστοποιείται ως εξής:

```
static public double mySin(double x)
   {
      double retval=0;
      double dif=x;
      int i=1;
      while(dif>maxError || dif < -maxError)
      {
         retval+=dif;
         i+=2;
         dif *= -x*x/(i*(i-1));
```

```
}
if(retval>1)
    retval=1;
if(retval<-1)
    retval=-1;
return retval;
```

Κώδικας 10:Η βελτιστοποίηση του ημιτόνου

6.3.8 Περαιτέρω Βελτιστοποίηση υπολογισμών σχέσεων Taylor

Οι αλλαγές που εφαρμόστηκαν στη προηγούμενη παράγραφο, μείωσαν αισθητά το χρόνο υπολογισμού των συναρτήσεων, όταν οι τιμές εισόδου (x) είναι σχετικά μικρές. Όταν η τιμή του x αυξάνει, η ποσότητα $\frac{x^n}{n!}$ που εμφανίζεται στις περισσότερες σειρές μειώνεται με πιο αργούς ρυθμούς, επομένως είναι αναγκαίος ο υπολογισμός πολλών όρων των σειρών Taylor. Έτσι, αυξάνεται το υπολογιστικό κόστος, πολλές φορές σε σημαντικό βαθμό. Έτσι, για μια τιμή x=10, η ισχύει ότι η ποσότητα $\frac{x^n}{n!} < 10^{-12}$, για τιμές του n > 46, για x=5, αρκεί n > 30, ενώ για x=2, χρειάζεται να φτάσουμε μόνο μέχρι την τιμή του n=20.

Για την επίλυση του συγκεκριμένου προβλήματος ακολουθήσαμε μια διαφορετική μεθοδολογία για τις επιμέρους συναρτήσεις κατά περίπτωση, αξιοποιώντας τις ιδιότητες της κάθε συνάρτησης:

- Στις τριγωνομετρικές συναρτήσεις, αξιοποιήσαμε την περιοδικότητα που έχουν ανά 2π. Έτσι, αν ένας αριθμός x που έχει τεθεί ως είσοδο σε τριγωνομετρική συνάρτηση η οποία υπολογίζεται με σειρά Taylor είναι εκτός του διαστήματος [-π,π],

τότε ανάγεται σε αυτό το διάστημα και στη συνέχεια λαμβάνει χώρα ο υπολογισμός της σειράς.

- Για την εκθετική συνάρτηση, αξιοποιούμε την ιδιότητα $e^x = e^{\frac{x}{2}} e^{\frac{x}{2}}$. Έτσι, σε περιπτώσεις κατά τις οποίες η τιμή του x είναι μεγάλη (x>15), αντί να υπολογίσουμε το άθροισμα της σειράς για x το υπολογίζουμε για $\frac{x}{2}$ και υψώνουμε στο τετράγωνο το αποτέλεσμα του υπολογισμού. Η ύψωση στο τετράγωνο πραγματοποιείται με πολλαπλασιασμό του αποτελέσματος με τον εαυτό του.

Για τις συναρτήσεις του φυσικού λογαρίθμου και της αντίστροφης εφαπτομένης έχουμε ήδη αναφερθεί στο σχετικό κεφάλαιο, καθώς η ανάπτυξη της σε σειρά Taylor έχει περιορισμένο πεδίο ορισμού. Για τις υπόλοιπες αντίστροφες τριγωνομετρικές συναρτήσεις δεν απαιτείται κάποια περαιτέρω επέμβαση, καθώς είναι ήδη περιορισμένο το πεδίο ορισμού τους στο διάστημα [-1,1].

Στο κώδικα που ακολουθεί παρατηρούμε πως τροποποιείται ο κώδικας της βελτιστοποιημένης συνάρτησης υπολογισμού της εκθετικής συνάρτησης:

```
static public double myExp(double x)
   {
      return myExp(x,myMath.maxError);
   }
   static public double myExp(double x, double maxError)
   {
      if(x<-15 || x >15)
      {
         double r1= myExp(x/2,maxError/10);
         return r1*r1;
      }
      double result = 1;
      int i = 1;
      double term = x;
      while (term <-maxError || term > maxError)
      {
         result+=term;
         i++;
```

```
            term*=x/i;
        }
        return result;
    }
```

Κώδικας 11:Η μέθοδος της εκθετικής συνάρτησης

Με τη συγκεκριμένη αλλαγή, τροποποιήσαμε επιπλέον και την ευαισθησία της προσέγγισης, καθώς είναι επιθυμητό το σφάλμα να διατηρεί τις προδιαγραφές.

Επιπρόσθετα, καθώς η Java δεν επιτρέπει default τιμές σε μεταβλητές, δημιουργήσαμε μια δεύτερη συνάρτηση, για να μη τροποποιηθεί η βασική της κλήση.

6.3.9 Επαλήθευση των αποτελεσμάτων υπολογισμών με σειρές Taylor

Στο επόμενο στάδιο, αναπτύξαμε κατάλληλη συνάρτηση επαλήθευσης των αποτελεσμάτων των υπολογισμών που πραγματοποιούμε με χρήση των υλοποιήσεων μας (με σειρές Taylor) ως προς τις τιμές που υπολογίζονται από τις συναρτήσεις βιβλιοθήκης της Java. Ακολούθως εμφανίζεται ο πίνακας που παρουσιάζει το μέσο, το μέγιστο και το ελάχιστο σφάλμα υπολογισμού, καθώς και την τυπική του απόκλιση.

Συνάρτηση	Min Error	Max Error	Mean Error	Std Error
Εκθετική	0	$7.15\,10^{-7}$	$7.6\,10^{-9}$	$4.08\,10^{-8}$
Ημίτονο	0	$9.93\,10^{-13}$	$1.82\,10^{-13}$	$2.39\,10^{-13}$
Συνημίτονο	0	$9.9\,10^{-13}$	$1.95\,10^{-13}$	$2.47\,10^{-13}$
Λογαριθμική	0	$1.75\,10^{-12}$	$6.32\,10^{-13}$	$3.77\,10^{-13}$
Αντίστροφο ημίτονο	0	$7.77\,10^{-4}$	$3.85\,10^{-6}$	$1.72\,10^{-4}$
Αντίστροφη εφαπτομένη	0	$8.11\,10^{-4}$	$1.61\,10^{-4}$	$1.11\,10^{-4}$

Πίνακας 4: Επαλήθευση αποτελεσμάτων των μεθόδων που δημιουργήσαμε σε σχέση με τις αντίστοιχες μεθόδους που προσφέρει η java

Βλέποντας τα δεδομένα του πίνακα, αντιλαμβανόμαστε ότι θέτοντας το όριο σε σφάλμα επιπέδου 10^{-12} παρατηρούμε ότι στην πράξη λαμβάνουμε σφάλμα στην περιοχή αυτή, με μικρές αποκλίσεις στην εκθετική συνάρτηση και μεγαλύτερες αποκλίσεις στις αντίστροφες τριγωνομετρικές συναρτήσεις. Τα σφάλματα που παρατηρούνται οφείλονται στη περιορισμένη ακρίβεια των αριθμών διπλής ακριβείας (περίπου 15 σημαντικά ψηφία στο δεκαδικό σύστημα) και στο προσθετικό σφάλμα που εισάγεται με τον υπολογισμό των επιμέρους συναρτήσεων με τη χρήση των αθροισμάτων των όρων Taylor.

Κεφάλαιο 7

Οδηγός χρήσης της επιστημονική αριθμομηχανής

7.1 Εισαγωγή

Η επιστημονική αριθμομηχανή που αναπτύξαμε στα πλαίσια της υπάρχουσας εργασίας μπορεί να χρησιμοποιηθεί από όλους τους χρήστες, κάθε γνωστικού υπόβαθρου. Έχει τη δυνατότητα εκτέλεσης τόσο των απλών όσο και των σύνθετων υπολογισμών.

7.2 Βασικές πράξεις

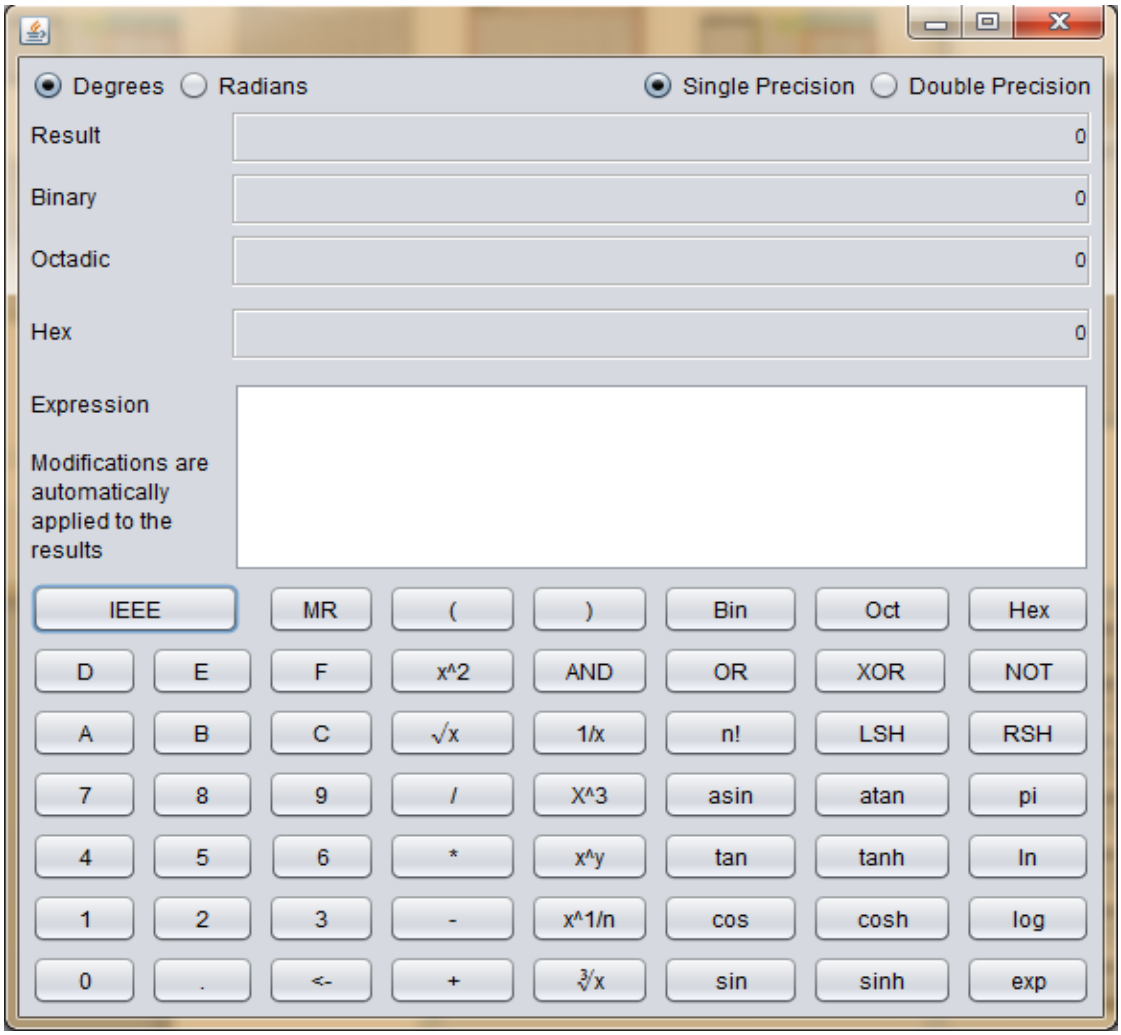
Οι βασικές πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμός και διαίρεση) εκτελούνται πατώντας το αντίστοιχο πλήκτρο (+,-*,/) μετά το πρώτο αριθμό της πράξης. Στη συνέχεια ακολουθεί ο δεύτερος αριθμός που λαμβάνει μέρος στην πράξη.

Αμέσως μετά από κάθε μεταβολή του κειμένου της έκφρασης, αν η έκφραση είναι σωστή, τότε εμφανίζεται το αποτέλεσμα.

7.3 Εισαγωγή αριθμών

Για εισαχθεί ένας αριθμός, πληκτρολογούνται τα ψηφία που τον αποτελούν με τη σειρά, όπως φαίνεται στην ακόλουθη εικόνα.

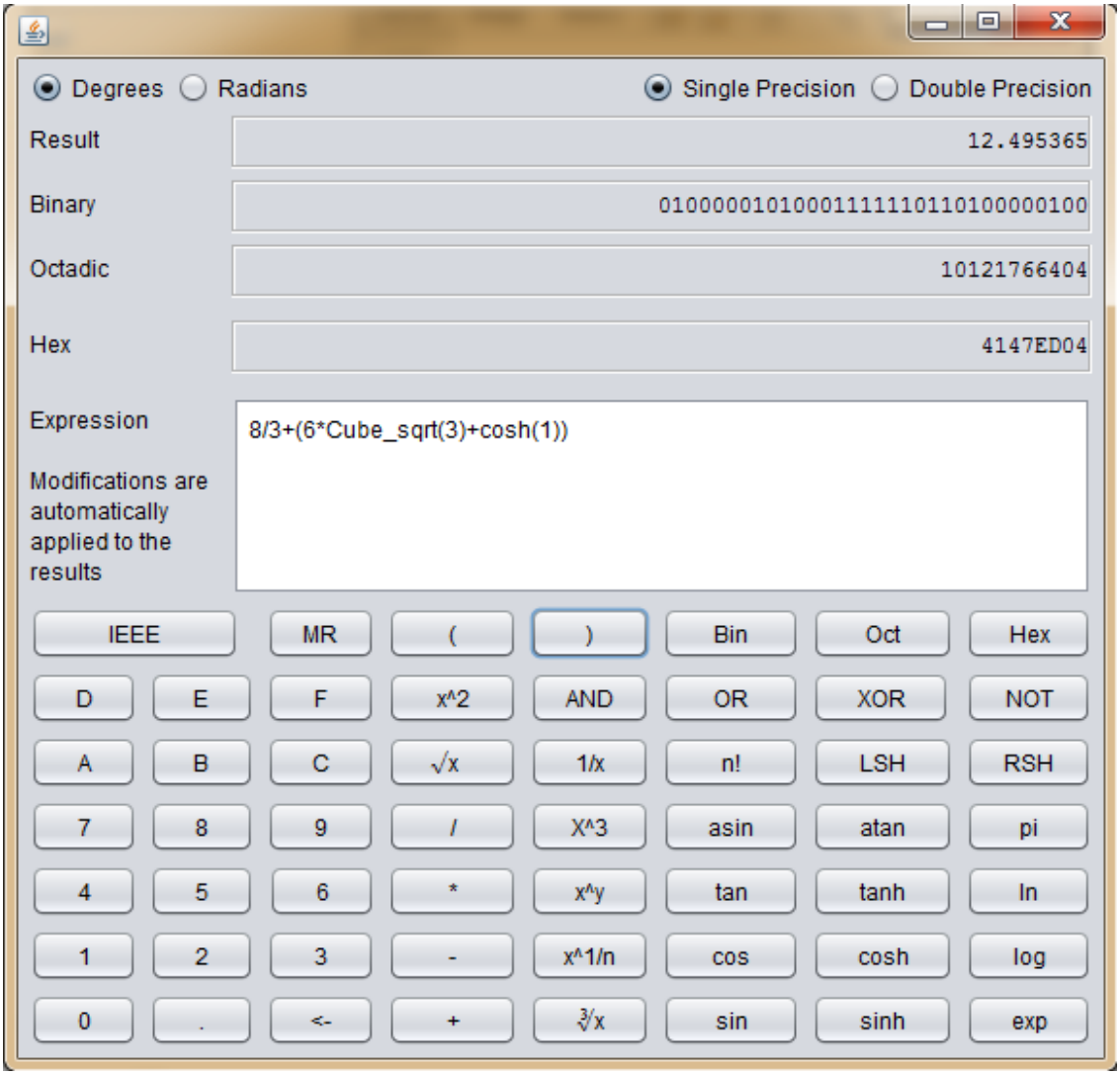
Εναλλακτικά μπορούμε να γράψουμε τους επιθυμητούς αριθμούς με απευθείας επεξεργασία από το πληκτρολόγιο του αντίστοιχου πεδίου. Είναι εφικτή επιπλέον η αντιγραφή-επικόλληση εκφράσεων από άλλες πηγές.



Εικόνα 12: Το interface της εφαρμογής και ο τρόπος εισαγωγής εκφράσεων

7.4 Επιστημονικές πράξεις

Σχετικά με την εκτέλεση των επιστημονικών πράξεων χρησιμοποιούμε τα αντίστοιχα πλήκτρα στο παραθυρικό περιβάλλον της εφαρμογής. Αρχικά εισάγεται η πράξη κι έπειτα ο αριθμός. Επιπρόσθετα, το όρισμα της συνάρτησης πρέπει να περικλείεται σε παρενθέσεις, αν είναι πολύπλοκο, διαφορετικά (αν είναι αριθμός) μπορεί να εισαχθεί κολλητά στο σύμβολο-τελεστή



Εικόνα 13: Το interface της εφαρμογής και ο τρόπος εισαγωγής εκφράσεων

Κεφάλαιο 8

Συμπεράσματα

Τα βασικά συμπεράσματα τα οποία είναι δυνατόν να διεξαχθούν από την ενασχόλησή μας με τις σειρές Taylor και το προγραμματισμό τους σε γλώσσα Java, καθώς και το προγραμματισμό της επιστημονικής αριθμομηχανής είναι τα ακόλουθα:

- Οι σειρές Taylor αποτελούν μια εξαιρετική προσέγγιση των συναρτήσεων που αναπαριστούν. Πιο συγκεκριμένα, διαπιστώσαμε πολύ μικρές αποκλίσεις μεταξύ της πραγματικής και της υπολογισμένης τιμής, για μικρές τιμές εισόδου (x).
- Η ακρίβεια των αποτελεσμάτων όμως μειώνεται όσο απομακρύνεται η τιμή της εισόδου από την τιμή 0. Αυτό το αποτέλεσμα όμως δεν οφείλεται σε αδυναμία της ανάλυσης Taylor, αλλά στις περιορισμένες δυνατότητες ακρίβειας που παρουσιάζουν οι σύγχρονοι υπολογιστές. Καθώς, το τελικό αποτέλεσμα προέρχεται από ένα άθροισμα πολλών όρων της σειράς Taylor, το σφάλμα ακριβείας κάθε όρου προστίθεται στο τελικό αποτέλεσμα (προσθετικό σφάλμα). Η επίδραση αυτού του σφάλματος είναι μεγαλύτερη όσο αυξάνεται ο αριθμός των όρων που απαιτούνται για τον υπολογισμό της σειράς. Για το λόγο αυτό, όσο μεγαλώνει η τιμή εισόδου x εξαιτίας του όρου $\frac{x^n}{n!}$, απαιτούνται περισσότεροι όροι για τον ακριβέστερο υπολογισμό του αποτελέσματος και επομένως υπάρχει μεγαλύτερο σφάλμα στο τελικό αποτέλεσμα.

- Ο απευθείας προγραμματισμός των σειρών Taylor έχει ως αποτέλεσμα την δημιουργία εξαιρετικά χρονοβόρων συναρτήσεων. Απαιτείται κατάλληλος, έξυπνος προγραμματισμός ώστε να επιταχυνθούν οι υπολογισμοί με τη χρήση αναδρομικών τύπων. Οι υπολογισμοί με χρήση αναδρομικών τύπων όμως, αυξάνει επιπλέον το προσθετικό σφάλμα.
- Η αντίστροφη πολωνική γραφή αποτελεί ένα εξαιρετικό εργαλείο για τον υπολογισμό των μαθηματικών εκφράσεων.

Βιβλιογραφία

1. Σημειώσεις Μαθήματος Αριθμητική Ανάλυση

2. Θεώρημα Taylor: http://en.wikipedia.org/wiki/Taylor's_theorem

3. Σειρές Taylor: http://el.wikipedia.org/wiki/%CE%A3%CE%B5%CE%B9%CF%81%CE%AC_Taylor

4. http://www.tutorialspoint.com/java/java_generics.htm

5. http://en.wikipedia.org/wiki/IEEE_floating_point

6. http://grouper.ieee.org/groups/754/

7. http://class.ece.iastate.edu/arun/CprE281F05/ieee754/ie4.html

8. http://steve.hollasch.net/cgindex/coding/ieeefloat.html

9. http://www.h-schmidt.net/FloatConverter/IEEE754.html

10. http://people.uncw.edu/tompkinsj/133/numbers/Reals.htm

11. http://www.java-forums.org/forum.php

12. http://www.forums.gr/forumdisplay.php?112-Java

13. http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Float.html

14. An Analysis of a Logical Machine Using Parenthesis-Free Notation," by Arthur W. Burks, Don W. Warren and Jesse B. Wright, 1954