ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Ενσωμάτωση του αλγορίθμου VAS στο Xcas»

Γεώργιος Νασόπουλος

A.M. : 1701361

A.E.M : 332

Επιβλέπων καθηγητής : Αλκιβιάδης Γ. Ακρίτας

ΠΕΡΙΕΧΟΜΕΝΑ

1.ΕΙΣΑΓΩΓΗ

1.1Αντικείμενο της εργασίας

Η εργασία αυτή έχει ως σκοπό την υλοποίηση του αλγορίθμου VAS των Vincent-Akritas-Strzebonski σε γλώσσα προγραμματισμού C++ με σκοπό την ενσωμάτωσή του στο πρόγραμμα «Giac/Xcas : ελεύθερο λογισμικό υπολογιστικής άλγεβρας» του Bernard Parisse, καθηγητή στο τμήμα «Institut Fourier de Mathématiques», στο πανεπιστήμιο της Grenoble. Ο αλγόριθμος αυτός έχει ήδη υλοποιηθεί σε γλώσσες προγραμματισμού μαθηματικής φύσης(με την πιο σημαντική το MATHEMATICA), αλλά ποτέ μέχρι τώρα σε αντικειμενοστραφή γλώσσα. Αυτό αποτελούσε μια πρόκληση, καθώς η γλώσσα C++ δεν είναι φτιαγμένη για τέτοιου είδους πράξεις με τη μέγιστη δυνατή ακρίβεια αλλά και εμβέλεια. Τελικώς όμως, παρά τα διάφορα προβλήματα που παρουσιάστηκαν η υλοποίηση ήταν επιτυχής.

1.2Διάρθρωση της εργασίας

Η εργασία αυτή αποτελείται από 3 μέρη:

1. ***ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ***:

Όπου παραθέτουμε την θεωρία της απομόνωσης και προσέγγισης πραγματικών ριζών πολυωνυμικών εξισώσεων.

2. ***ΔΗΜΟΣΙΕΥΣΕΙΣ ΣΧΕΤΙΚΑ ΜΕ ΤΟΝ ΑΛΓΟΡΙΘΜΟ V.A.S***:

Οι οποίες είναι οι σχετικές δημοσιεύσεις των Alkiviadis G. Akritas, Andreas I. Argyris, Adam W. Strzebonski, P. S. Vigklas,μεταφρασμένες.

3. ***ΜΕΤΑΦΟΡΑ ΨΕΥΔΟΚΩΔΙΚΑ VAS,FLQ,LMQ ΣΕ C++:***

Τέλος παρουσιάζουμε αναλυτικά την δομή του κώδικα σε C++ εξηγώντας την λογική και την λειτουργία των ενσωματωμένων ρουτινών.

1.3Ευχαριστίες.

Σε αυτό το σημείο, θα ήθελα να ευχαριστήσω τους καθηγητές Αλκιβιάδη Ακρίτα,Bernard Parisse αλλά και Βάνα Ντουφεξή οι οποίοι σε διάφορα στάδια της πορείας υλοποίησης της εργασίας αυτής η βοήθεια και καθοδήγησή τους ήταν καταλυτική.

1^{Ο} ΜΕΡΟΣ

Απομόνωση και προσέγγιση πραγματικών ριζών πολυωνυμικών

Εισαγωγή

Μέθοδοι για την εύρεση των πραγματικών ριζών — ή για την επίλυση — *πολυωνυμικών εξισώσεων, οποιονδήποτε βαθμού, στον δακτύλιο* Z[x] *περιγράφονται στα βιβλία Αριθμητικής Ανάλωσης και Θεωρίας Εξισώσεων. Η παρουσίαση μας τον θέματος αυτού διαφέρει σε τρία σημαντικά σημεία:*

- *Ακλουθώντας την "Γαλλική μαθηματική σχολή" κάνουμε διάκριση ανάμεσα στην* ***απομόνωση*** *και* ***προσέγγιση*** *των ριζών. Αντίθετα, στο μεγαλύτερο μέρος της βιβλιογραφίας ακολουθείται η "Αγγλική μαθηματική σχολή" όπου η προσέγγιση των ριζών ισοδυναμεί με την λύση των εξισώσεων.*

- *Διατυπώνουμε με ακρίβεια τα θεωρήματα των* Budan (1807) και Fourier (1820) που *δίνουν ένα* ***πάνω φράγμα*** (upper bound) στον *αριθμό των πραγματικών ριζών που έχει μία πολυωνυμική εξίσωση σε τυχαίο ανοικτό διάστημα* (f, r). *Βλέπε και το* Figure 6.1. *Τονίζουμε πως η διατύπωση τον θεωρήματος τον* Budan *δύσκολα βρίσκεται στην βιβλιογραφία διότι αντί αυτής υπάρχει η διατύπωση τον θεωρήματος τον* Fourier. *Το τελευταίο μάλιστα συχνά ονομάζεται θεώρημα τον* Budan *ή και θεώρημα των* Budan-Fourier.

- *Παρουσιάζουμε το θεώρημα τον* A.J.H. Vincent τον 1836, *με την βοήθεια του οποίου αναπτύσσονται δύο μέθοδοι απομόνωσης των πραγματικών ριζών πολυωνυμικών εξισώσεων με συνεχή κλάσματα. Η πρώτη μέθοδος ανήκει στον* Vincent (1836) *αλλά έχει εκθετικό χρόνο νεολογισμού. Η δεύτερη μέθοδος ανήκει στον γράφοντα — αναπτύχθηκε το* 1978 και *βελτιώθηκε με τον* Strzebonski το 1994 — και *όχι μόνο έχει πολυωνυμικό χρόνο υπολογισμού αλλά είναι και η* ***ταχύτερη μέθοδος απομόνωσης που υπάρχει.*** *Το θεώρημα τον* Vincent *είχε τόσο πολύ ξεχαστεί ώστε ακόμα και η Enzyclopaedie der mathematischen Wissenschaften το αγνοεί.*

Τονίζουμε πως και ο ίδιος ο Vincent *είχε τόσο ξεχαστεί ώστε το μικρό τον όνομα αρχικά αποδόθηκε σαν* M., *επειδή το Γαλλικό μαθηματικό περιοδικό στο οποίο δημοσιεύθηκε η εργασία του τον ανέφερε σαν* M. Vincent. *Πολύ αργότερα και ύστερα από υπόδειξη κατάλαβε ο γράφων ότι το* M. *ήταν για την Γαλλική λέξη "κύριος" ενώ στην πραγματικότητα ο* Vincent *είχε τρία μικρά ονόματα* Alexandre Joseph Hidulf ???.

Στην *συνέχεια θα εξετάσουμε λεπτομερώς τις δύο κλασσικές μεθόδους για την απομόνωση των πραγματικών ριζών πολυωνυμικών εξισώσεων με ακέραιους συντελεστές,* δηλαδή την *μέθοδο της* ***διχοτόμησης,*** Sturm (1829), και την *μέθοδο των* ***συνεχών κλασμάτων (σ.κ.),*** Vincent - Akritas - Strzebonski (1836, 1978, 1994).

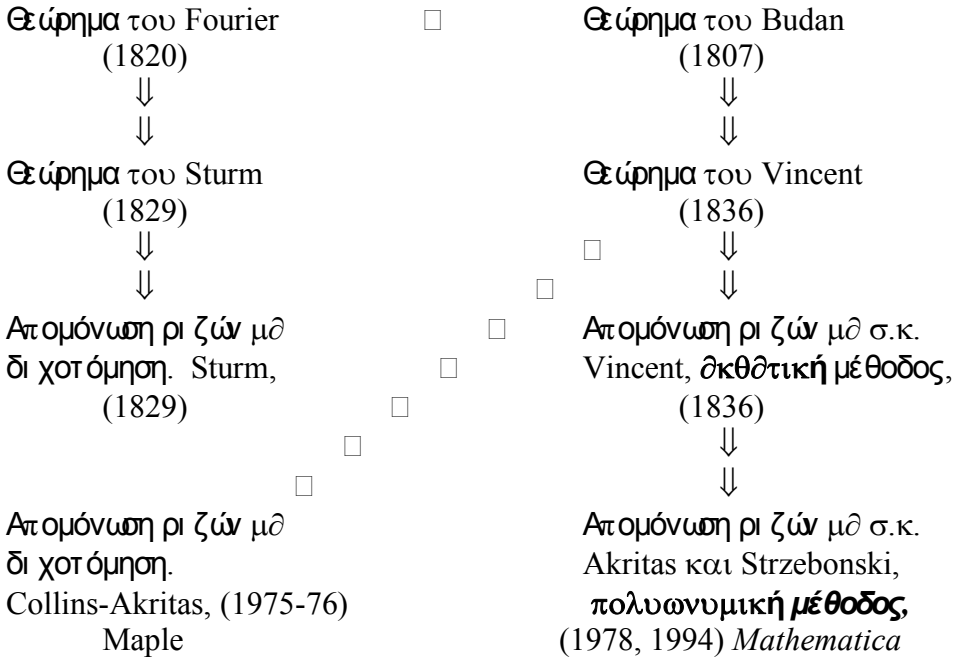


Figure 6.1. *Τα θεωρήματα των* Budan και Fourier και οι *δύο κλασσικές μέθοδοι απομόνωσης των πραγματικών ριζών πολυωνυμικών εξισώσεων. Η μέθοδος των* Collins-Akritas *βασίζεται σε μία ολική τροποποίηση του θεωρήματος του* Vincent, *δεν θεωρείται κλασσική και δεν καλύπτεται στο βιβλίο αυτό.*

Ιστορική ανασκόπηση και βασικές έννοιες

Τα αρχαιότερα δείγματα αλγεβρικών εξισώσεων εμφανίζονται το 1700 ή 1650 π. Χ. στον *πάπυρο* Rhind, στον *οποίο ο Αιγύπτιος* Ahmes *περιέλαβε και παλαιότερες εργασίες. Βρίσκουμε για παράδειγμα το εξής πρόβλημα: "Μία ποσότητα μαζί με το έβδομο της κάνουν 19. Ποία είναι η ποσότητα;" Προφανώς, το πρόβλημα είναι να λύσουμε την εξίσωση* $x + (\frac{1}{7})x = 19$ *όπως θα λέγαμε σήμερα. Λόγω έλλειψης κατάλληλου αλγεβρικού συμβολισμού οι Αιγύπτιοι έλυναν παρόμοιες εξισώσεις με έναν πολύπλοκο τρόπο που αργότερα έγινε γνωστός σαν μέθοδος της "λάθους θέσης."*

Αν και μερικοί υποστηρίζουν πως οι Έλληνες έλυσαν εξισώσεις δευτέρου βαθμού, εν γένει ούτε οι Αιγύπτιοι ούτε οι Έλληνες δεν έκαναν καμία σημαντική πρόοδο. Οι Άραβες πέτυχαν περισσότερα, αλλά μόνο την εποχή της Αναγέννησης σημειώθηκε αξιοσημείωτη πρόοδος, όταν οι Ιταλοί μαθηματικοί του δέκατου πέμπτου και δέκατου έκτου αιώνα (Tartaglia, Cardano και Ferrari) *πέτυχαν να λύσουν με ριζικά τις γενικές εξισώσεις* ***τρίτου*** *και* ***τετάρτου*** *βαθμού. Στο* ***βιβλίο του*** **Dunham** *περιγράφεται με ενδιαφέρον η διαμάχη, μεταξύ των* Tartaglia και Cardano, για την *προτεραιότητα της ανακάλυψης.*

Τον δέκατο έβδομο και δέκατο όγδοο αιώνα έγιναν πολλές προσπάθειες για την επίλυση της γενικής εξίσωσης ***πέμπτου*** *βαθμού. Με τις προσπάθειες αυτές αποκτήθηκε μια βαθύτερη κατανόηση της "φύσης" των ριζών των αλγεβρικών εξισώσεων (ειδικά με την εργασία των* Cardano και Descartes), *αλλά* παρ' *όλα αυτά κανένας δεν μπόρεσε να λύσει με ριζικά την εξίσωση πέμπτου βαθμού.*

Επόμενο ήταν να αναρωτηθούν οι μαθηματικοί αν είναι δυνατή μια τέτοια λύση. Η απάντηση δόθηκε το 1804 *από τον* Paolo Ruffini, ο *οποίος απέδειξε πως είναι αδύνατο να λυθεί η γενική εξίσωση πέμπτου βαθμού με ριζικά. Αργότερα, το* 1826, ο Abel *απέδειξε πως είναι αδύνατο να λυθούν με ριζικά γενικές αλγεβρικές εξισώσεις βαθμού μεγαλύτερου του* 4.

Στις αρχές του δέκατου ένατου αιώνα η προσοχή των μαθηματικών είχε ήδη στραφεί στις ***αριθμητικές μεθόδους*** *για την επίλυση πολυωνυμικών εξισώσεων με ακέραιους συντελεστές. Αυτή την περίοδο ο* Fourier *συνέλαβε την ιδέα να "σπάσει" το πρόβλημα σε* ***δύο***

υποπροβλήματα: *δηλαδή πρώτα να* ***απομονώσει*** *τις ρίζες και έπειτα να τις* ***προσεγγίσει*** *με όση ακρίβεια επιθυμείται .*

Απομόνωση (isolation) των *πραγματικών ριζών μιας πολυωνυμικής εξίσωσης είναι η διαδικασία εύρεσης πραγματικών διαστημάτων, μη τεμνομένων μεταξύ τους, έτσι ώστε κάθε διάστημα περιέχει ακριβώς μία πραγματική ρίζα, κα κάθε πραγματική ρίζα περιέχεται σε κάποιο διάστημα. Από την άλλη μεριά,* ***προσέγγιση*** (approximation) *είναι η διαδικασία σμίκρυνσης των διαστημάτων απομόνωσης τόσο, όσο να προσεγγισθούν ο ρίζες στον επιθυμητό βαθμό ακρίβειας.*

Το πρόβλημα της απομόνωσης των ριζών ήταν το σημαντικότερο κα τράβηξε την προσοχή των μαθηματικών. Με ξεκάθαρο τον σκοπό, ήρθε τώρα κα η επιτυχία. Στις αρχές του δέκατου ένατου αιώνα ο F.D. Budan και ο J.B.J. Fourier *παρουσίασαν δύο διαφορετικά (αλλά ισοδύναμα) θεωρήματα που μας επιτρέπουν να υπολογίσουμε τον* ***μέγιστο*** *δυνατό αριθμό πραγματικών ριζών που έχε μία εξίσωση με πραγματικούς συντελεστές μέσα σε ένα δεδομένο διάστημα.*

Το θεώρημα του Budan *δημοσιεύτηκε το* 1807 στην *εργασία* "Nouvelle methode pour la resolution des equations numeriques", *ενώ το θεώρημα του* Fourier *προδημοσιεύτηκε το* 1820 στο "Le bulletin des sciences par la societe philomatique de Paris." Εξ *αιτίας της σημασίας των δύο αυτών θεωρημάτων δημιουργήθηκε μεγάλη έριδα σχετικά με την προτεραιότητα ανακάλυψης. Όπως μας πληροφορεί ο* F. Arago στο *βιβλίο του Biographies of distinguished scientific men* (p. 383) ο Fourier *"θεώρησε απαραίτητο να λάβει βεβαιώσεις από τέως φοιτητές της Πολυτεχνικής Σχολής ή από καθηγητές του Πανεπιστημίου" γ α να απόδειξε ότι είχε διδάξει το θεώρημά του το* 1796, 1797 *και* 1803.

Όπως θα δούμε στην ενότητα 6.2, *βασιζόμενος στο θεώρημα του* Fourier, *ο* C.Sturm *παρουσίασε το* 1829 *ένα βελτιωμένο θεώρημα που μας επιτρέπει να υπολογίσουμε — με* ***διχοτόμηση*** — τον ***ακριβή*** *αριθμό των πραγματικών ριζών που έχε μία εξίσωση με ακέραιους συντελεστές μέσα σε ένα δεδομένο διάστημα. Έτσι ο* Sturm *είναι ο πρώτος που* ***έλυσε*** *το πρόβλημα της απομόνωσης των πραγματικών ριζών κα δικαιολογημένα έγινε παγκόσμια διάσημος.*

Έτσι, από το 1830 *μέχρι το* 1978 η *μέθοδος του* Sturm *ήταν η μόνη ευρέως γνωστή κα χρησιμοποιούμενη, με συνέπεια το θεώρημα του* Budan *να ξεχασθεί παντελώς —* και *μαζί του και το θεώρημα του* Vincent που *απορρέει από αυτό. Απ' ότι ξέρουμε, η διατύπωση του*

θεωρήματος του Budan *μπορεί να βρεθεί μόνο στο άρθρο του* Vincent, 1836, *κα στ ς εργασίες του συγγραφέα αυτού του μονογράμματος ενώ, σε αντίθεση, το θεώρημα του* Fourier *βρίσκεται σε όλα τα βιβλία τα σχετικά με την θεωρία των εξισώσεων.*

Στην ενότητα 6.3, *παρουσιάζουμε το θεώρημα του* Budan, *στο οποίο στηρίζεται το θεώρημα του* Vincent *του* 1836. *Το τελευταίο είναι η βάση της μεθόδου των* ***συνεχών κλασμάτων*** (Vincent-Akritas-Strzebonski) για την *απομόνωση των πραγματικών ριζών πολυωνυμικών εξισώσεων με ακέραιους συντελεστές, μ ας μεθόδου κατά πολύ καλλίτερης της μεθόδου του* Sturm. *Η μέθοδος των συνεχών κλασμάτων είναι η πιο γρήγορη στον κόσμο και χρησιμοποιείται στο Mathematica. Σε αντίθεση η μέθοδος του* Sturm *δεν χρησιμοποιείται πουθενά πλέον.*

Το θεώρημα του Vincent *είχε τόσο πολύ ξεχαστεί ώστε προ του* 1978 *δεν αναφέρεται από κανέναν συγγραφέα με εξαίρεση τον* **Obreschkoff** (1963) και **Uspensky** (1948). *Ο γράφων ανακάλυψε το θεώρημα του* Vincent στο *βιβλίο του* Uspensky το 1975-76 και στο *θέμα αυτό έγραψε την διδακτορική του διατριβή*(1978).

Σημειώνουμε παρενθετικά πως υπάρχει κα η μέθοδος των Collins-Akritas *για την απομόνωση των πραγματικών ριζών πολυωνυμικών εξισώσεων με* ***διχοτόμηση.*** *Η μέθοδος αυτή αναπτύχθηκε το* 1975-76, *στηρίζεται σε μία ολική τροποποίηση του θεωρήματος του* Vincent *κα χρησιμοποιείται στο* Maple.

Για την ιστορία αναφέρουμε πως υπάρχει μια αναφορά του Uspensky — στην *εισαγωγή του βιβλίου του — ότι ο ίδιος ανακάλυψε την μέθοδο απομόνωσης πραγματικών ριζών με συνεχή κλάσματα. Έτσι λοιπόν, αρχικά, η μέθοδος του* Vincent *εσφαλμένα ονομαζόταν* ***"μέθοδος του Uspensky"*** *ενώ η μέθοδος των* Collins-Akritas *ονομαζόταν* ***"τροποποιημένη μέθοδος του Uspensky".*** *Όμως όπως τόνισε ο γράφων, ο* Uspensky *δεν είχε υπ' όψη του το θεώρημα του* Budan *κα το μόνο που έκανε ήταν να διπλασίασε τον χρόνο υπολογισμού της μεθόδου του* Vincent. *Λεπτομέρειες θα δούμε στην ενότητα* 6.3.

6.2 Το *θεώρημα τον* Fourier *και η μέθοδος τον* Sturm *για την απομόνωση πραγματικών ριζών με διχοτόμηση*

Στην ενότητα αυτή *θα γνωρίσουμε το θεώρημα τον* Fourier, το *οποίο μας δίνει τον* ***μέγιστο δυνατό*** *πραγματικών ριζών που έχει μία εξίσωση με ακεραίους συντελεστές μέσα σε ένα διάστημα. Στην συνέχεια θα δούμε πως ο* Sturm *τροποποίησε το θεώρημα τον* Fourier ώστε να *παίρνουμε τον* ***ακριβή*** *αριθμό των πραγματικών ριζών μέσα στο υπό εξέταση διάστημα.*

6.2.1 Το θεώρημα τον Fourier

Το θεώρημα του Fourier *βασίζεται στην εργασία των* Cardano και Descartes την *οποία πρώτα παρουσιάζουμε. Χρειαζόμαστε τον ακόλουθο ορισμό:*

Ορισμός:

Έστω μία πεπερασμένη ή άπειρη ακολουθία πραγματικών αριθμών c_1, c_2, c_3, ... *Λέμε ότι ανάμεσα στους αριθμούς c και* c_r *(l< r) υπάρχει μία* ***μεταβολή πρόσημων*** (sign variation) ή *απλά* ***μεταβολή*** *όταν ισχύουν τα ακόλουθα:*

i. Αν $r = \{+1$, οι *αριθμοί* c_l *και* c_r *έχουν αντίθετα πρόσημα.*

ii. *Αν* $r \leq$??$\{ +2$ οι *αριθμοί* c_{l+1}, ..., c_{r-1} *είναι όλοι μηδέν και οι* c_l *και* c_r *έχουν αντίθετα πρόσημα.*

Ακολουθεί ένα πρόγραμμα στο Mathematical *για τον υπολογισμό των μεταβολών πρόσημου τόσο μιας αριθμητικής ακολουθίας όσο και ενός πολυωνύμου μιας μεταβλητής.*

```
variations[s_] : = Module [{lis , l, i , v = 0} ,
If[PolynomialQ[s, Variables[s]],
 lis = Select[CoefficientList[s, Variables[s]], # ≠ 0 &] ,
 lis = Select[s, # ≠ 0 &] ] ;
 l = Length[lis] ;
 If[l >1,
Do[
If[Not[Equal[Sign[lis[[i]]] ,
Sign[ lis [[i + 1]]]]], v + + ]  , {i, l - 1}]] ; v] /;
 ListQ[s]| |
```

```
Which[Length[Variables[s]]==0, AtomQ[Variables[s]] ,
Length[Variables[s] ==1, AtomQ[First[Variables[s]]]]
```

Παράδειγμα:

Έστω το πολυώνυμο $p(x) = x^3 - 7x + 7$, οι *συντελεστές του οποίου σχηματίζουν την πεπερασμένη ακολουθία* {1, 0, -7, 7}. *Στην ακολουθία αυτή υπάρχουν* 2 *μεταβολές πρόσημου. Πράγματι,*

```
variations[x^3- 7 x + 7]
```

2

Ο Gerolamo Cardano (1501-1576) *είναι ο πρώτος στην ιστορία των μαθηματικών που σύνδεσε τον* ***αριθμό*** $\boldsymbol{\rho_+}$ ***των θετικών ριζών*** *ενός πολυωνύμου* $p(x)$ *με τον* ***αριθμό*** $\boldsymbol{\nu}$ ***των μεταβολών πρόσημων*** *στην ακολουθία των συντελεστών του* $p(x)$.

Συγκεκριμένα ο Cardano *ήξερε τις εξής δύο περιπτώσεις:*

- αν στους *συντελεστές ενός πολυωνύμου δεν υπάρχει* ***καμία*** *μεταβολή πρόσημου, δηλαδή* $\boldsymbol{\nu = 0}$, *τότε δεν υπάρχει* ***καμία*** *θετική ρίζα, δηλαδή* $\boldsymbol{\rho_+ = 0}$, και
- *αν στους συντελεστές ενός πολυωνύμου υπάρχει* ***μία*** *μεταβολή πρόσημου, δηλαδή* $\boldsymbol{\nu = 1}$, *τότε υπάρχει* ***μία*** *θετική ρίζα, δηλαδή* $\boldsymbol{\rho_+ = 1}$.

Οι περιπτώσεις αυτές όσο και αν φαίνονται απλές χρησιμοποιούνται — όπως θα δούμε στην ενότητα 6.3 — σαν *κριτήριο τερματισμού στην πιο γρήγορη μέθοδο απομόνωσης πραγματικών ριζών — αυτή με συνεχή κλάσματα. Επίσης χρησιμοποιούνται και σαν κριτήριο τερματισμού στην μέθοδο των* Collins-Akritas.

Παράδειγμα:

Έτσι για παράδειγμα το πολυώνυμο $p(x) = x^3 + 7x + 7$, δεν *έχει καμία θετική ρίζα, ενώ το πολυώνυμο* $p(x) = x^3 - 7x - 7$, *έχει μία θετική ρίζα.*

Αυτό που έκανε ο Rene Descartes (1596-1650) *ήταν να γενικεύσει το "θεώρημα" του* Cardano. *Συγκεκριμένα, στον* Descartes *οφείλουμε την σχέση* $\boldsymbol{\nu = \rho_+ + 2\lambda}$, *όπουλ* $\in Z \geq 0$, την *οποία θα αποδείξουμε πιο κάτω με την βοήθεια του θεωρήματος του* Fourier.

Όπως αναφέραμε στην προηγούμενη ενότητα, το θεώρημα τον Fourier *δημοσιεύθηκε το* 1820 *και βρίσκεται σε όλα τα βιβλία τα σχετικά με την θεωρία των εξισώσεων είτε με το όνομα* Budan - Fourier είτε μόνο με το *όνομα* Budan. Πιοτού το *διατυπώσουμε θα παρουσιάσουμε δύο βοηθητικά λήμματα.* Αν και *ενδιαφερόμαστε μόνο για πολυώνυμα με ακέραιους συντελεστές θα αποδείξουμε τα λήμματα αυτά στην γενικότερη περίπτωση με πραγματικούς συντελεστές.*

Λήμμα 6.2.1:

Έστω ότι p(x) = 0 *είναι μία πολυωννμική εξίσωση βαθμού* n > 0 με *πραγματικούς συντελεστές και έστω ότι έχει μία πραγματική ρίζα α πολλαπλότητας* m. *Τότε για* ε > 0 τα *πολυώνυμα* p(x) και $p^{(1)}$ (x) — *όπου εν γένει* $p^{(i)}(x)$ *είναι η* i-στή *παράγωγος τον p(x) — έχουν* ***αντίθετα*** *πρόσημα στο διάστημα (α* - ε, α) και ***ίδια*** *πρόσημα στο διάστημα* (α, α + ε).

Απόδειξη:

Εφαρμόζουμε το ανάπτυγμα κατά Taylor, *δηλαδή*

$$p(\alpha \pm x) = \sum_{i=0}^{n} \frac{p^{i}(\alpha)}{i!} x^{i}.$$

Τότε λόγω της πολλαπλότητας m *της ρίζας α, έχουμε*

$$p(\alpha \pm \varepsilon) = \sum_{i=m}^{n} \frac{p^{i}(\alpha)}{i!} (\pm\varepsilon)^{i} = \frac{p^{m}(\alpha)}{m!} (\pm\varepsilon)^{m} + \equiv + \frac{p^{n}(\alpha)}{n!} (\pm\varepsilon)^{n}$$

και

$$p^{1}(\alpha \pm \varepsilon) = \sum_{i=m}^{n} \frac{p^{i}(\alpha)}{(i-1)!} (\pm\varepsilon)^{i-1} = \frac{p^{m}(\alpha)}{(m-1)!} (\pm\varepsilon)^{m-1} + \equiv + \frac{p^{n}(\alpha)}{(n-1)!} (\pm\varepsilon)^{n-1}$$

Βλέπουμε όμως πως για e αρκετά μικρό τα πρόσημα στις δύο τελευταίες εξισώσεις εξαρτώνται μόνο από το πρόσημο τον όρου $p^{(m)}$ *(α) — τον πρώτον όρο που δεν έχει ρίζα το α — και συνεπώς είναι αντίθετα στο διάστημα (α* - ε, *α) και ίδια στο διάστημα* (α, α + ε).//

Παράδειγμα:

Έστω p(x) το ακόλουθο πολυώνυμο που έχει την ρίζα α = 2, *πολλαπλότητας* 4.

```
p[x_] := (x^3 - 7 x + 7) (x - 2)^4 //Expand
```

Για $\varepsilon = 0.025$, *από τα γραφήματα των* $p(x)$ *και* $p^{(1)}(x)$, *βλέπουμε πως τα πρόσημα είναι αντίθετα στο διάστημα* $(2 - \varepsilon, 2)$ *και ίδια στο διάστημα* $(2, 2 + \varepsilon)$. (Το *γράφημα της* $p^{(1)}(x)$, *είναι με* ***διακεκομμένη γραμμή)***

```
ε = 0.025;
plotO = Plot[{p[x]} , {x, 2 - ε,2 + ε } ,
AxesOrigin{2, 0}, DisplayFunction:->Identity] ;

plotl = Plot[{p' [x]} , {x, 2 - ε , 2  + ε } , AxesOrigin - {2, 0} ,
PlotStyle — {Dashing[{0.05,0.05}] =, DisplayFunction :-*Identity ] ;

Show[{plot0, plot1} , DisplayFunction :-* $DisplayFunction] ;
```

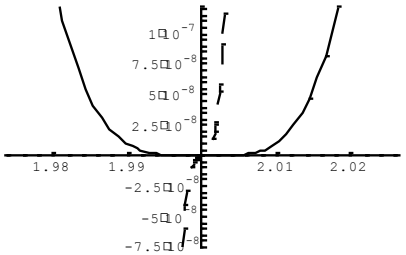


Figure 6.2. Τα *γραφήματα της p(x) και της παραγώγου της* $p^{(1)}(x)$ *(με διακεκομμένη γραμμή). Στο διάστημα (2 - e, 2) τα πρόσημα των p(x) και* $p^{(1)}(x)$ *είναι αντίθετα* ενώ στο *διάστημα (2, 2 + e) τα πρόσημα είναι ίδια.*

***Λήμμα* 6.2.2:**

Έστω πάλι ότι $p(x) = 0$ *είναι μία πολυωνυμική εξίσωση βαθμού* $n > 0$ *με πραγματικούς συντελεστές και έστω ότι έχει μία πραγματική ρίζα α πολλαπλότητας* $m > 1$. Θεωρούμε τα m πολυώνυμα $p(x)$, $p^{(1)}(x)$,, ..., $p^{(m-1)}(x)$, *όπου και πάλι το* $p^{(i)}(x)$, *συμβολίζει την i-στή παράγωγο του* $p(x)$. *Τότε για* $\varepsilon > 0$, αν στα m *πολυώνυμα αντικαταστήσουμε την μεταβλητή x με τιμές από το διάστημα* $(\alpha - \varepsilon, \alpha)$ τα *πρόσημα της αριθμητικής ακολουθίας που προκύπτει είναι* ***εναλλασσόμενα****, ενώ αν αντικαταστήσουμε την μεταβλητή x με τιμές από το διάστημα* $(\alpha, \alpha + \varepsilon)$ τα *πρόσημα της αριθμητικής ακολουθίας που προκύπτει* ***συμπίπτουν*** *με το πρόσημο του* $p^{(m)}(a)$, *όπου* $p^{(m)}(x)$, *είναι η πρώτη συνάρτηση για την οποία το α δεν είναι ρίζα.*

Απόδειξη:

Εφαρμόζουμε επαναληπτικά το Λήμμα 6.2.1.//

Παράδειγμα (συνέχεια):

Συνεχίζουμε με τα γραφήματα των επόμενων μερικών παραγώγων του p(x). Κάθε ένα γράφημα το συγκρίνουμε με το γράφημα της προηγουμένης παραγώγου και βλέπουμε πως στο διάστημα (α - ε, α) τα πρόσημα είναι αντίθετα, ενώ στο διάστημα (α, α + ε) τα πρόσημα είναι ίδια. Τα γραφήματα της ***πρώτης*** *και* ***τρίτης*** *παραγώγου είναι με* ***διακεκομμένες γραμμές.***

```
plot2 = Plot [p' [x] , {x, 2 - ε, 2  + ε } ,
AxesOrigin    {2, 0} , DisplayFunction :-> Identity ]

Show[{plot1, plot2} , DisplayFunction :-> $DisplayFunction] ;
```

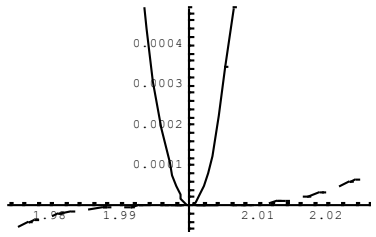


Figure 6.3. *Τα γραφήματα της $p^{(1)}(x)$ (με διακεκομμένη γραμμή) και της παραγώγου της $p^{(2)}(x)$.*

```
plot3 = Plot[{p ''' [x]} , {x, 2 - ε , 2  + ε } , AxesOrigin - {2, 0} ,
PlotStyle — {Dashing[{0.05,0.05}] = , DisplayFunction :- > Identity ] ;

Show[{plot2, plot3}, DisplayFunction:- > $DisplayFunction] ;
```

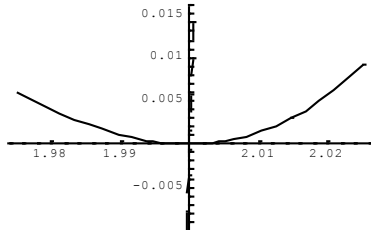


Figure 6.4. *Τα γραφήματα της $p^{(2)}(x)$ και της παραγώγου της $p^{(3)}(x)$ (με διακεκομμένη γραμμή).*

```
plot4 = Plot [p '''' [x] , {x, 2 - ε , 2  + ε } ,
                  AxesOrigin — {2, 0} , DisplayFunction :- > Identity ] ;

Show[{plot3, plot4} , DisplayFunction :- >$DisplayFunction] ;
```

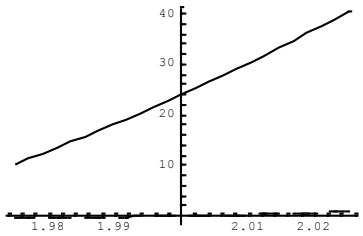


Figure 6.5. *Τα γραφήματα του $p^{(3)}(x)$ (με διακεκομμένη γραμμή) και της παραγώγου της $p^{(4)}(x)$.*

Οι παράγωγοι $p^{(3)}(x)$και $p^{(4)}(x)$είναι οι τελευταίες για τις οποίες ισχύει ότι στο διάστημα (2 - ε, 2) τα πρόσημά τους είναι αντίθετα ενώ στο διάστημα (2, 2+ ε) είναι ίδια. Στο Figure 6.6 *συγκρίνουμε τις* $p^{(4)}(x)$ *και* $p^{(5)}(x)$ *και βλέπουμε ότι* τα *πρόσημα είναι τα ίδια στο διάστημα* (2 - ε, 2 + ε).

```
plot5 = Plot[{p''''[x]} , {x, 2 - ε , 2  + ε } , AxesOrigin    {2, 0} ,
PlotStyle —> {Dashing[{0.05,0.05}] = , DisplayFunction :-> Identity ] ;

Show[{plot4, plot5} , DisplayFunction :-> $DisplayFunction] ;
```

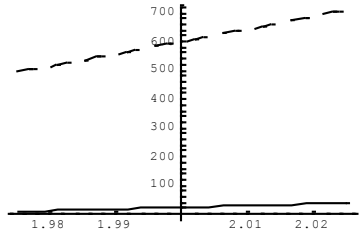


Figure 6.6. Τα *γραφήματα της* $p^{(4)}(x)$ *και της παραγώγου της* $p^{(5)}(x)$ *(με διακεκομμένη γραμμή*

Ορισμός:

Έστω p(x) = 0 είναι μία *πολυωνυμική εξίσωση βαθμού n* > 0 με *πραγματικούς συντελεστές. Ονομάζουμε* ***ακολουθία*** **Fourier** την *ακολουθία των n + 1 συναρτήσεων*

$$F_{seq}(x) = \{p(x), p^{(1)}(x), p^{(2)}(x), ..., p^{(n)}(x)\}$$

όπου $p^{(i)}(x)$ είναι η i-στή παράγωγος του p(x)

Παράδειγμα:

Έστω πάλι το πολυώνυμο $p(x) = x^3 - 7x + 7$. Η *ακολουθία* Fourier είναι

$$F_{seq}(x) = \{x^3 - 7x + 7, 3x^2 - 7, 6x, 6\}$$

***Θεώρημα του* Fourier (1820):** *(Υπολογισμός ενός πάνω φράγματος στον αριθμό των πραγματικών ριζών που έχει μία εξίσωση σε ένα ανοιχτό διάστημα.)*

Έστω $p(x) = 0$ *μία πολυωνυμική εξίσωση βαθμού* $n > 0$ με *πραγματικούς συντελεστές. και έστω ότι έχει μία πραγματική ρίζα α πολλαπλότητας* $m > 1$. Αν στην *ακολουθία* Fourier $F_{seq}(x) = \{p(x), p^{(1)}(x), p^{(2)}(x), ..., p^{(n)}(x)\}$ *αντικαταστήσουμε το x με τυχαίους πραγματικούς αριθμούς l, r (*$l < r$*), τότε προκύπτουν οι αριθμητικές ακολουθίες* $F_{seq}(l)$ *και* $F_{seq}(r)$ *με* v_l *και* v_r *μεταβολές πρόσημου αντίστοιχα. Ισχύουν τα ακόλουθα:*

- **i.** *Η ακολουθία* $F_{seq}(l)$ *δεν μπορεί να έχει λιγότερες μεταβολές πρόσημου από την ακολουθία* $F_{seq}(r)$. Δηλαδή, $v_l \geq v_r$.
- **ii.** *Ο αριθμός ρ των πραγματικών ριζών της εξίσωσης* $p(x) = 0$ που *βρίσκονται στο διάστημα* (l, r) *ποτέ δεν μπορεί να είναι μεγαλύτερος από τον αριθμό των μεταβολών πρόσημου που χάνονται στην* $F_{seq}(x)$ *κατά την μετάβασή μας από την αντικατάσταση* $x \leftarrow l$ *στην αντικατάσταση* $x \leftarrow r$. Δηλαδή, $\rho \leq v_l - v_r$.
- iii. *Όταν ο αριθμός ρ των πραγματικών ριζών της εξίσωσης* $p(x)=0$ που *βρίσκονται στο διάστημα* (l, r) *είναι γνήσια μικρότερος από τον αριθμό των μεταβολών πρόσημου που χάνονται στην* $F_{seq}(x)$ *κατά την μετάβασή μας από την αντικατάσταση* $x \leftarrow l$ *(στην αντικατάσταση* $x \leftarrow r$*, τότε η διαφορά είναι άρτιος αριθμός.* Δηλαδή, $\rho = v_l - v_r - 2\lambda$, *όπου* $\lambda \varepsilon Z_{>0}$.

Απόδειξη:

Όταν το x μεταβάλλεται στο διάστημα ((, r) ο αριθμός των μεταβολών προσήμου στην ακολουθία Fourier, $F_{seq}(x)$, *αλλάζει* ***μόνο*** *όταν το x είναι ρίζα του p(x) ή μιας των παραγώγων του. Εξετάζουμε τις δύο αυτές περιπτώσεις:*

***Περίπτωση* 1η.** *Έστω ότι α είναι ρίζα του p(x) πολλαπλότητας m. Από το Λήμμα* 6.2.2 *προκύπτει πως όταν το x μεταβάλλεται στο διάστημα (α - ε, α + ε), για αρκετά μικρό ε > 0, υπάρχουν m μεταβολές πρόσημου στην ακολουθία* Fourier *αμέσως πριν το πέρασμα από την ρίζα α, και δεν υπάρχει καμία μεταβολή πρόσημου στην ακολουθία* Fourier *αμέσως μετά το πέρασμα από την ρίζα α. Συνεπώς, στην ακολουθία* Fourier, $F_{seq}(x)$, *χάνονται m μεταβολές πρόσημου.*

***Περίπτωση* 2η.** *Έστω ότι το α είναι τώρα ρίζα μιας παραγώγου, πολλαπλότητας m.* Δηλαδή, για *κάποιο i,* $0 < i < n$, *έχουμε* $p^{(i-1)}(x) \neq 0$ και $p^{(i)}(x)= 0$.

Θεωρούμε την ακολουθία

$D_{seq}(x) = \{p^{(i-1)}(x), p^{(i)}(x), p^{(i+1)}(x),\ldots, p^{(i+m)}(x)\}$

που *είναι* ***υποακολουθία*** *της* $F_{seq}(x)$, και *όπου* $p^{(i+m)}(x)$*είναι το πρώτο πολυώνυμο για το οποίο α δεν είναι ρίζα. Προσέξτε πως όταν το x μεταβάλλεται στο διάστημα (α - ε, α + ε), για αρκετά μικρό ε > 0,* τα *πρόσημα των* $p^{(i-1)}(x)$ *και* $p^{(i+m)}(x)$ *δεν αλλάζουν επειδή τα πολυώνυμα αυτά δεν μηδενίζονται.* Με την *βοήθεια του Λήμματος* 6.2.2, και *λαμβάνοντας* υπ' *όψη την τιμή της m (άρτια ή περιττή) καθώς επίσης και το αν τα πρόσημα των* $p^{(i-1)}(x)$ και $p^{(i+m)}(x)$ *είναι ίδια ή αντίθετα, προκύπτει πως στην ακολουθία* $D_{seq}(x)$ *χάνεται* ***άρτιος*** *αριθμός πρόσημων όταν το x μεταβάλλεται στο διάστημα (α - ε, α + ε), για αρκετά μικρό e > 0.* (Βλέπε και το *παράδειγμα που ακολουθεί.)*

Συνεπώς, όταν το x μεταβάλλεται σε τυχαίο διάστημα (l, r), ο *αριθμός των μεταβολών πρόσημου που χάνονται στην ακολουθία* $F_{seq}(x)$ ***είτε*** *είναι ίσος με τον αριθμό ρ των πραγματικών ριζών του p(x) στο διάστημα αυτό* ***είτε*** *ξεπερνάει τον ρ κατά άρτιο αριθμό.//*

Προσοχή:
Όταν το x μεταβάλλεται σε τυχαίο διάστημα (f, r) με την *ακολουθία* $F_{seq}(x)$ *μπορούμε να βρούμε τον ακριβή αριθμό των ριζών μόνο στις εξής* ***δύο*** *περιπτώσεις:* ***(α)*** *αν δεν χάνεται καμία μεταβολή πρόσημου τότε δεν υπάρχει καμία ρίζα στο (l,* r), και ***(β)*** *αν χάνεται μία μεταβολή πρόσημου τότε υπάρχει μία πραγματική ρίζα στο* (l, r). **Τα *αντίστροφα των* (α) *και (β) δεν ισχύουν!***

Παράδειγμα: *(της* **2ης *περίπτωσης*** *του θεωρήματος του* Fourier)

Έστω το πολυώνυμο p(x) = $x^4 - 8x^3 + 24x^2$. *Το α* = 2 δεν *είναι ρίζα του, αλλά είναι ρίζα πολλαπλότητας*

m = 2 της *παραγώγου του* $p^{(2)}(x)$ — *δηλαδή χρησιμοποιώντας τον συμβολισμό του παραπάνω θεωρήματος, i* = 2 *στην ακολουθία* $D_{seq}(x)$.

```
p [x_] = x^4 - 8 x^3 + 24 x^2 ;
{p[2] , p'[2] ,p''[2] , p'''[2] , p'''[2]}
```

```
{48, 32, 0, 0, 24}
```

Έτσι στο παράδειγμα αυτό έχουμε $D_{seq}(x) = \{p^{(1)}(x), p^{(2)}(x), p^{(3)}(x), p^{(4)}(x)\}$.

```
Dseq[x_] : = {p'[x] , p''[x] , p '''[x] , p''''[x]}
```

Θα *δούμε πως όταν το x μεταβάλλεται σε τυχαίο διάστημα* (2 - ε, 2 + ε), ο *αριθμός των μεταβολών πρόσημου που χάνονται στην ακολουθία* $D_{seq}(x)$ *είναι άρτιος. Πράγματι, στο διάστημα (2 - ε, 2) έχουμε 2 μεταβολές πρόσημου,*

```
Dseq[1.98]
```

```
{32., 0.0048, -0.48, 24}
```

```
variations[Dseq[1.98]]
```

```
2
```

ενώ στο *διάστημα (2, 2 + ε)* — με *βάση το* Λήμμα 6.2.2 — τα πολυώνυμα $p^{(2)}(x)$ *και* $p^{(3)}(x)$ *παίρνουν το πρόσημο του* $p^{(4)}(x)$ *και έτσι δεν έχουμε μεταβολές πρόσημου.*

```
Dseq[2.02]
```

```
{32.,    0.0048,    0.48,    24}
```

```
variations[ Dseq[ 2.02]]
```

```
0
```

Συνεπώς χάθηκαν 2 μεταβολές πρόσημου. Προσέξτε πως τα πρόσημα των $p^{(1)}(x)$ *και* $p^{(4)}(x)$ *είναι ίδια και δεν αλλάζουν επειδή τα πολυώνυμα αυτά δεν μηδενίζονται στο α = 2.*

Παράδειγμα: *(υπολογισμός ενός πάνω φράγματος στον αριθμό των πραγματικών ριζών μέσα σε ένα διάστημα με την βοήθεια του θεωρήματος του* Fourier)

Έστω το πολυώνυμο $p(x) = x^3 - 7x + 7$. *Για να βρούμε ένα πάνω φράγμα στον αριθμό των πραγματικών ριζών που έχει το πολυώνυμο αυτό στο διάστημα* (0, 2) *με το θεώρημα του* Fourier *υπολογίζουμε την ακολουθία* Fourier, $F_{seq}(x)$, *του πολυωνύμου αυτού*

```
p[x_] = x  - 7 x + 7;
Fseq[x_] := {p[x] , p'[x] , p'''[x], p''''[x]} Fseq[x]
```

$\{7 - 7x + x^3, -7 + 3x^2, 6x, 6\}$

και κατόπιν υπολογίζουμε τον αριθμό των μεταβολών πρόσημου που χάνονται στην $F_{seq}(x)$ *κατά την* μετάβασή του *x από το* 0 στο 2. *Συγκεκριμένα έχουμε*

```
Fseq[0]
```

{7, -7, 0, 6}

```
variations[Fseq[0]]
```

2

και

```
Fseq[2]
```

{1, 5, 12, 6}

```
variations[ Fseq[ 2]]
```

0

Δηλαδή χάνονται δύο μεταβολές πρόσημου, και αυτό σημαίνει πως στο διάστημα (0, 2) το *πολυώνυμο είτε έχει δύο πραγματικές ρίζες ή καμία, κάτι που πρέπει να διερευνηθεί. Στο Mathematical ένα πρόγραμμα που υπολογίζει την ακολουθία* Fourier *είναι το εξής:*

```
createFourierSequence [p_] : =
Module [{Fseq = { } , q = p, r, v = First [Variables [p]]} ,
r = D[p , v] ; AppendTo [Fseq, {q, r}] ;
While [Exponent [r, v] > 0 , r = D[r, v] ; AppendTo [Fseq, r] ] ;
Fseq // Flatten] / ;
AtomQ[First[Variables[p]]]
```

και για το τελευταίο παράδειγμα έχουμε:

```
p[x_] = x^3 - 7 x + 7;
Fseq[ x_] = createFourierSequence[ p[ x]] ; variations[
Fseq[ 0]] - variations[ Fseq[ 2]]
```

2

Το θεώρημα του Fourier *μπορεί να χρησιμοποιηθεί για την απόδειξη του ακόλουθου θεωρήματος.*

***Θεώρημα των* Cardano-Descartes:** *(Υπολογισμός ενός πάνω φράγματος στον αριθμό των θετικών ριζών που έχει μία εξίσωση.)*

Έστω

$p(x) = c_n x^n + c_{n-1}x^{n-1} + \ldots + c_1 x + c_0$ *μία πολυωνυμική εξίσωση βαθμού* $n > 0$ *με πραγματικούς συντελεστές. Αν ν είναι ο αριθμός των μεταβολών πρόσημου στην ακολουθία των συντελεστών* $\{c_n, c_{n-i}, \ldots, c_1, c_0\}$ *— όπου οι μηδενικοί συντελεστές έχουν παραληφθεί —* και ρ_+ *είναι ο αριθμός των θετικών ριζών της* $p(x) = 0$*, τότε* $\boldsymbol{\nu = \rho_+ + 2\lambda}$*, όπου* $\lambda \in Z_{>0}$.

Απόδειξη:

Αυτό που ζητάμε είναι ένα πάνω φράγμα στον αριθμό των πραγματικών ριζών της $p(x) = 0$ *μέσα στο διάστημα* $(0, \infty)$*. Η ακολουθία* Fourier στην *περίπτωσή μας είναι* $F_{seq}(x) = \{p(x), p^{(i)}(x), p^{(2)}(x), \ldots, p^{(n)}(x)\}$*, όπου*

$p(x) = c_n x^n + c_{n-1}x^{n-1} + \ldots + c_1 x + c_0,$
$p^{(i)}x) = nc_n x^{n-1} + \ldots + c_i,$
$p^{(2)}(x) = n(n-1)c_n x^{n-2} + \ldots + (2!)c_2,$

…
$p^{(n)}(x) = (n!)\, c_n.$

Με βάση το θεώρημα του Fourier *υπολογίζουμε πρώτα την ακολουθία*

$Fse_q(0) = \{ c_0, c_i, (2!)c_2, \ldots, (n!)\, c_n \}$

που έχει ν μεταβολές πρόσημου, και έπειτα την ακολουθία $F_{seq}(\infty)$ *που δεν έχει μεταβολή πρόσημου — επειδή όλοι οι όροι έχουν το πρόσημο του* c_n*. Συνεπώς από το θεώρημα του Fourier έχουμε*

$$\boldsymbol{\nu = \rho_+ + 2\lambda},$$

όπου $\lambda \in Z_{>0}$.//

Προσέξτε πως το θεώρημα των Cardano-Descartes μας δίνει τον ***ακριβή*** *αριθμό των θετικών ριζών* ***μόνο*** *στις περιπτώσεις που* $v = 0$ *ή* $v = i$ *— οπότε κατ ανάγκη* $\lambda = 0$.

6.2.2 Το *θεώρημα τον* Sturm

Από ιστορική άποψη αξίζει να αναφέρουμε πως τα δύο βασικά θέματα μελέτης στην ζωή του Fourier ήταν η θεωρία της θερμότητας κα η θεωρία αριθμητικής επίλυσης *εξισώσεων. Και τα δύο αυτά θέματα συνεχίστηκαν αργότερα από τον Sturm, ο οποίος είχε προσωπικές και επιστημονικές σχέσεις με τον Fourier. Το 1829 οι χειρόγραφες εργασίες του Fourier σχετικά με την αριθμητική επίλυση εξισώσεων είχαν διαδοθεί σε αρκετούς ειδικούς επί του θέματος, ανάμεσα στους οποίους ήταν και ο Sturm — ο οποίος αναφέρει ξεκάθαρα πόσο πολύ επηρεάστηκε από τις εργασίες του Fourier.*

Αυτό που έκανε ο Sturm στην θεωρία αριθμητικής επίλυσης *εξισώσεων ήταν να αντικαταστήσει την ακολουθία Fourier με την ακολουθία*

$S_{seq}(x) = \{p(x), p^{(1)}(x), r_1(x), ..., r_k(x)\}$,

η οποία αποκαλείται ***ακολουθία*** **Sturm.** *Η ακολουθία αυτή προκύπτει με εφαρμογή του Ευκλείδειου αλγόριθμου στα πολυώνυμα* $p(x)$ *και* $p^{(1)}(x)$, *ορίζοντας τα* $r_i(x)$, $1 \leq i \leq k$, *σαν τα* ***αρνητικά*** *των υπολοίπων που προκύπτουν.*
Δηλαδή η ακολουθία Sturm ορίζεται από τις ακόλουθες σχέσεις:

$p(x) = p^{(l)}(x)\, q_1(x) - r_1(x)$,
$p^{(1)}(x) = r_1(x) q_2(x) - r_2(x)$,
$r_1(x) = r_2(x) q_3(x) - r_3(x)$,

$r_{k-2}(x) = r_{k-1}(x) q_k(x) - r_k(x)$,

Το πλεονέκτημα της ακολουθίας Sturm είναι ότι μπορούμε τώρα να αποκτήσουμε τον ***ακριβή*** *αριθμό των πραγματικών ριζών που έχει* η *εξίσωση* $p(x)=0$ *μέσα σε ένα δεδομένο διάστημα. Προσέξτε πως αν ο βαθμός του πολυωνύμου* $p(x)$ *είναι* n *τότε, συνήθως, η ακολουθία Sturm*

αποτελείται συνολικά από n + 1 συναρτήσεις — δεδομένου ότι στην διαδικασία εύρεσης ενός μέγιστου κοινού διαιρέτη (μ.κ.δ.) των p(x) και $p^{(1)}$ ο βαθμός κάθε υπολοίπου είναι συνήθως κατά μονάδα μικρότερος από τον βαθμό του προηγουμένου υπολοίπου. Επιπλέον, αν δεν υπάρχουν πολλαπλές ρίζες το (χ) είναι *σταθερά. (Λεπτομέρειες για την εύρεση ενός μ.κ.δ. δύο πολυωνύμων θα δούμε σε ένα από τα επόμενα κεφάλαια. Η μετατροπή ενός πολυωνύμου με πολλαπλές ρίζες σε ένα με απλές θα συζητηθεί σε άλλη ενότητα αυτού του κεφαλαίου.)*

Παράδειγμα:

Η ακολουθία Sturm για το *πολυώνυμο* $p(x) = \chi^3 - 7\chi + 7$ *είναι* $S_{seq}(x) = \{p(x), p^{(1)}(x), r_1(x), r_2(x)\}$, *όπου* $p^{(1)}(x)$ *είναι η πρώτη παράγωγος του p(x) και τα υπόλοιπα υπολογίζονται ως εξής: Το* $r_1(x)$ *είναι*

```
p [x_] = x^3 - 7 x + 7; pp [x_] = D[p[x],x];
rl = -PolynomialMod[p [x] ,pp[x]]
```

```
- 7 + 14 x
```

ή ισοδύναμα $r_1(x) = 2x - 3$. *Προσέξτε πως αν υπολογίσουμε και το πηλίκο* $q_1(x)$,

```
ql = PolynomialQuotient [p [x] ,pp[x] ,x]
```

$$\frac{x}{3}$$

τότε ισχύει η προαναφερθείσα σχέση

$p(x) = p^{(1)}(x)\ q_1(x) - r_1(x)$:

```
p[x] == Evaluate [(pp [x] q1 - r1)//Expand]
```

```
True
```

Αντίστοιχα το $r_2(x)$ *υπολογίζεται*

```
r2 = -PolynomialMod[pp [x] ,r1]
```

ή *ισοδύναμα* $r_2(x) = 1$. *Αν δε υπολογίσουμε και το πηλίκο* $q_2(x)$,

```
q2 = PolynomialQuotient [pp [x] ,r1, x]
```

τότε ισχύει η δεύτερη (κα τελευταία για το παράδειγμα αυτό) προαναφερθείσα σχέση $p^{(1)}(x)= r_1(x)q_2(x) - r_2(x)$*:*

```
pp [x] == Evaluate [(r1 q2 - r2)// Expand]
```

True

Άρα η ακολουθία Sturm για το πολυώνυμο $p(x) = x^3 - 7x + 7$ *είναι* $S_{seq}(x) = \{x^3 -7x + 7, 3x^2 - 7, 2x - 3, 1\}$*. Στο Mathematical ένα πρόγραμμα που υπολογίζει την ακολουθία Sturm είναι το εξής:*

```
createSturmSequence[ p_ := Module[{Sseq = { } ,
q = p, r, v = First[Variables[p]]},
r = D[p, v] ; AppendTo [Sseq, {q, r}] ;
While [Exponent [r, v] > 0 , temp = -  PolynomialMod[q, r] ;
 AppendTo[ Sseq, temp ; q =  r; r = temp ] ;
Sseq // Flatten/ ; AtomQ[ First[ Variables[ p]
```

Οι (συντελεστές των μελών της ακολουθίας που υπολογίζονται είναι όμως είναι ρητοί και όχι ακέραιοι.

```
Sseq[ x_ ]= createSturmSequence[ x^3 - 7 x + 7]]
```

$[7 - 7x + x^3, -7 + 3x^2, -7 + \frac{14x}{3}, \frac{1}{4}]$

Όπως θα δούμε σε επόμενο κεφάλαιο, ο απλούστερος — αλλά όχι ο πιο αποτελεσματικός —τρόπος για να πάρουμε ακέραιους συντελεστές στα υπόλοιπα είναι ο εξής: πριν την διαίρεση πολλαπλασιάζουμε τον διαιρετέο με τον κύριο συντελεστή του διαιρέτη υψωμένο στην δύναμη $m - n + 1$*, όπου* m *είναι ο βαθμός του διαιρετέου κα* n *είναι ο βαθμός του διαιρέτη, κα* **2ον.** *μετά την διαίρεση διαιρούμε τους συντελεστές του υπολοίπου με τον μέγιστο κοινό διαιρέτη τους. Έτσι τώρα το πρόγραμμα γίνεται*

```
createSturmSequenceIC [p_] : = Module[
{cl, gcd, lc, m, n, q = p, r, Sseq = {} , v  = First [Variables [p]]}
r = D[p, v] ; AppendTo [Sseq, {q, r}] ;
m = Exponent[ q, v] ; n =  Exponent[ r, v] ;
While[n > 0 ,
lc = Last[ CoefficientList[ r, v]] ;
temp = - PolynomialMod[lc^m"n+1 q, r] ;
```

```
cl = CoefficientList[ temp, v] ;
gcd = GCD[ Apply[ Sequence, cl]] ;
cl = cl/gcd;
temp = Fold[ v #1 + #2 &, 0 , Reverse[cl]] // Expand; AppendTo[
Sseq, temp] ; q =  r; r = temp; m = n;
n = Exponent[ r, v] ] ;
Sseq // Flatten ] /  ; AtomQ[First[Variables[p]]]
```

και *η ακολουθία* Sturm είναι:

```
Sseq[x_] = createSturmSequenceIC[x^3- 7 x + 7]
```

$$\{7 - 7x + x^3, -7 + 3x^2, -3 + 2x, 1\}$$

Οι *4 χαρακτηριστικές ιδιότητες των μελών της ακολουθίας* Sturm:

Έστω p(x) = 0 μία εξίσωση με ρητούς συντελεστές και χωρίς πολλαπλές ρίζες. Για τις *συναρτήσεις της ακολουθίας* Sturm,i'seqCx) = *{p(x), $p^{(1)}(x)$, $r_1(x)$, ..., $r_k(x)$}, ισχύουν τα εξής :*

i. Αν α είναι μία ρίζα του πολυωνύμου *p(x) τότε για αρκετά μικρό* $\in > 0$ *τα πολυώνυμα p(x) και* $p^{(1)}(x)$*, έχουν αντίθετα πρόσημα στο διάστημα (α -* $\in$*, α) και ίδια πρόσημα στο διάστημα (α, α +* $\in$*).*

Απόδειξη:

Αυτό είναι το Λήμμα 6.2.1.

ii. *Δύο διαδοχικά μέλη της ακολουθίας* Sturm δεν *μπορούν να μηδενίζονται ταυτόχρονα.*

Απόδειξη:

Υποθέτουμε το αντίθετο. Δηλαδή έστω ότι $r_i(x) = 0$ *και* $r_{i+1}(x) = 0$*. Τότε από τις σχέσεις ορισμού των υπολοίπων έχουμε:* $r_i(x) = r_{i+1}(x)q_{i+2}(x) - r_{i+2}(x) = 0$*, που συνεπάγεται* $r_{i+2}(x) = 0$*. Συνεπώς,* $r_{i+3}(x) = 0 \ldots$ *,* $r_k(x) = 0$*. Αυτή είναι όμως και* η *αντίφαση, διότι* $r_k(x)$ *είναι σταθερά. Προφανώς, το ίδιο ισχύει και για τα πολυώνυμα p(x) και* $p^{(1)}(x)$*.//*

iii. *Αν για τυχαίο i,* $i \neq k$*, η συνάρτηση* $r_i(x)$ *της ακολουθίας* Sturm *μηδενίζεται για κάποια τιμή* x_0*, τότε οι γειτονικές της συναρτήσεις στην ακολουθία, εκτιμώμενες στην ίδια τιμή, έχουν αντίθετα πρόσημα.*

Απόδειξη:

Πράγματι, έστω ότι $r_i(x) = 0$*. Τότε από την σχέση ορισμού των υπολοίπων έχουμε:*

$r_{i-1}(x_0) = r_i(x_0)q_{i+1}(x_0) - r_{i+1}(x_0) = 0$ *απ όπου προκύπτει* $r_{i-1}(x_0) = -r_{i+1}(x_0)$. *Προφανώς, το ίδιο ισχύει και για το πολυώνυμο* $p^{(1)}(x)$.//

iv. *Η τελευταία συνάρτηση* $r_k(x)$ *δεν μηδενίζεται και συνεπώς δεν αλλάζει πρόσημο.*

Απόδειξη:

Προφανής διότι το πολυώνυμο $p(x)$ *δεν έχει πολλαπλές ρίζες .//*

Έχοντας απόδειξε τις χαρακτηριστικές αυτές ιδιότητες των μελών της ακολουθίας Sturm *είμαστε έτοιμο γ α το ακόλουθο:*

***Θεώρημα του* Sturm (1829):**

Έστω η πολυωνυμική εξίσωση $p(x) = 0$ *με ακέραιους συντελεστές κα χωρίς πολλαπλές ρίζες. Τότε, για τον αριθμό ρ των πραγματικών ριζών της στο διάστημα ((, f) ισχύει η ισότητα*

$$\boldsymbol{P = v_l - v_r}$$

όπου v_l *,*v_r *είναι οι μεταβολές πρόσημου των αριθμητικών ακολουθιών* $S_{seq}(l)$ *και* $S_{seq}(r)$ *αντίστοιχα.*

Απόδειξη:

Αυτό που πρέπει να δείξουμε είναι πως όταν το x μεταβάλλεται από το l στο r, η ακολουθία Sturm ***χάνει*** *μία μεταβολή πρόσημου όταν το x περνάει από μία ρίζα α της* $p(x) = 0$ *και, σε αντίθεση από την ακολουθία* Fourier, ***δεν χάνει*** *καμία μεταβολή πρόσημου όταν το x περνάει από μία ρίζα ενός άλλου μέλους της ακολουθίας.*

Πράγματι, από την πρώτη ιδιότητα ***(i)*** — ή το *Λήμμα* 6.2.i — *συμπεραίνουμε πως όταν το x περνάει από μία ρίζα α της* $p(x) = 0$ *χάνεται ακριβώς μία μεταβολή πρόσημου.*

Έτσι ας υποθέσουμε πως το x περνάει από μία ***(όχι κατ' ανάγκη απλή)*** *ρίζα αι της* $r_i(x) = 0$. *Από τις ιδιότητες* ***(ii)*** *και* ***(iii)*** *έπεται πως* $r_{i-1}(a_i)$ *και* $r_{i+1}(a_i)$ *είναι μη μηδενικά και έχουν αντίθετα πρόσημα. Διαλέγουμε τότε ένα μικρό διάστημα* $(a_i - \in, + \in), \in > 0$, *όπου οι δύο αυτές συναρτήσεις δεν μηδενίζονται, που σημαίνει πως δεν αλλάζουν πρόσημα κα δημιουργούμε τον ακόλουθο πίνακα (*Figure 6.7*):*

X	r_{i-1}	r_i	r_{i-1}	r_{i-1}	r_i	r_{i-1}
$\xi_i - \varepsilon$	+	±	-	-	±	+
ξ_i	+	0	-	-	0	+
$\xi_i + \varepsilon$	+	Υ(±)	-	-	Υ(±)	+

Figure 6.7. Στις δύο στήλες της συνάρτησης r_i, τα πρόσημα (±) δηλώνουν ρίζα με πολλαπλότητα.

Από τον πίνακα του σχήματος 6.7 βλέπουμε πως όταν το x περνάει από μία (όχι κατ' ανάγκη απλή) ρίζα της $r_i(x) = 0$ ***δεν χάνεται μεταβολή πρόσημου,*** *και αυτό συμπληρώνει την απόδειξή μας. II*

Παράδειγμα:

Έστω πάλι το πολυώνυμο $p(x) = x^3 - 7x + 7$ η ακολουθία Sturm του οποίου, όπως είδαμε, είναι $S_{seq}(x) = [x^3 - 7x + 7, 3x^2 - 7, 2x - 3, 1]$. Με βάση το θεώρημα του Sturm ο ακριβής αριθμός των πραγματικών ριζών που έχε το p(x) στο διάστημα (0, 2) είναι $v_0 - v_2$, όπου v_0, v_2 είναι οι μεταβολές πρόσημου των αριθμητικών ακολουθιών $S_{seq}(0)$ και $S_{seq}(2)$ αντίστοιχα. Στο συγκεκριμένο παράδειγμα $S_{seq}(0) = [7, -7, -3, 1]$ με 2 μεταβολές πρόσημου, $v_0 = 2$, και $S_{seq}(2) = [1, 5, 1, 1]$ με καμία μεταβολή πρόσημου, $v_2 = 0$. Συνεπώς υπάρχουν $v_0 - v_2 = 2$ πραγματικές ρίζες στο διάστημα (0, 2). Το αποτέλεσμα αυτό επιβεβαιώνεται και με το Mathematica:

```
Sseq[x_] = createSturmSequenceIC[x^3 - 7 x + 7] ;
variations[Sseq[0]] - variations[Sseq[2]]
```

2

Ο ίδιος ο Sturm ανέφερε πως το θεώρημα αυτό του 1829 ήταν παράπλευρο αποτέλεσμα των εκτεταμένων ερευνών του στην περιοχή των εξισώσεων δ αφορών δευτέρας τάξης. Το αίτημα να μην έχε η εξίσωση p(x) = 0 πολλαπλές ρίζες δεν είναι περιορισμός της γενικότητας, διότι στην αντίθετη περίπτωση — όπως θα δούμε στην επόμενη ενότητα — τις "βγάζουμε" με μία εύκολη "παραγοντοποίηση" (square free factorization) και ύστερα εφαρμόζουμε το θεώρημα του Sturm.

Το 1835 ο Sturm τροποποίησε το θεώρημά του έτσι ώστε να μπορεί να προσδιορίζει τον αριθμό των ζευγών μιγαδικών ριζών που έχε η εξίσωση p(x) = 0, στην περίπτωση που η ακολουθία Sturm έχει n + 1 μέλη, όπου n = deg(p(x)).

***Θεώρημα του* Sturm (1835):**

Έστω η πολυωνυμική εξίσωση p(x) = 0 βαθμού n, με ακέραιους συντελεστές και χωρίς πολλαπλές ρίζες. Τότε, ο αριθμός των ***ζευγών μιγαδικών ριζών*** *του p(x) ισούται με τον αριθμό των μεταβολών πρόσημου στην ακολουθία των πρώτων όρων των n (συναρτήσεων* $[p^{(1)}(x), r_i(x), ..., r_k(x)\}$ *της ακολουθίας Sturm* $S_{seq}(x)$.

Απόδειξη:

Η αλήθεια αυτού του κανόνα βασίζεται στο γεγονός ότι η μία από τυχαίες δύο γειτονικές συναρτήσεις της ακολουθίας Sturm έχε βαθμό άρτιο κα η άλλη περιττό. Επομένως, αν οι δύο αυτές συναρτήσεις έχουν το ίδιο πρόσημο για $x = +\infty$, *θα έχουν αντίθετα πρόσημα για* $x = -\infty$, *και αντίστροφα. Έτσι αν εκτιμήσουμε την ακολουθία Sturm,* $S_{seq}(x)$, *στα σημεία* $x = +\infty$ *και* $x = -\infty$ *κάθε* ***μεταβολή πρόσημου*** *σε μία ακολουθία αντιστοιχεί σε* ***σταθερότητα πρόσημου*** *στην άλλη (βλέπε και το παράδειγμα που ακολουθεί). Δηλαδή ο αριθμός σταθεροτήτων πρόσημου στην ακολουθία Sturm εκτιμημένης στο σημείο* $x = -\infty$ *ισούται με τον αριθμό μεταβολών πρόσημου στην ακολουθία Sturm εκτιμημένης στο σημείο* $x = +\infty$.

Έστω ότι i είναι ο αριθμός των μεταβολών πρόσημου στην ακολουθία $S_{seq}(+\infty)$. *Ο μεταβολές αυτές προέρχονται από τα πρόσημα των συντελεστών των υψηλότερων βαθμών ως προς x (των πρώτων όρων), στις n συναρτήσεις* $[p^{(1)}(x), r_i(x), ..., r_k(x)\}$ *της ακολουθίας Sturm* $S_{seq}(x)$ — *όπου οι πρώτοι όροι των p(x) και* $p^{(1)}(x)$ *θεωρούνται θετικοί.*

Μόλις είδαμε όμως ότι στην ακολουθία $S_{seq}(-\infty)$ *ο αριθμός σταθεροτήτων πρόσημου θα είναι i, ή ισοδύναμα η ακολουθία* $S_{seq}(-\infty)$ *θα έχει n - i μεταβολές πρόσημου. (Εδώ θεωρούμε δεδομένο ότι στην ακολουθία Sturm υπάρχουν n + 1 συναρτήσεις και ότι στην* $S_{seq}(x)$ *ο αριθμός των μεταβολών πρόσημου συν τον αριθμό των σταθεροτήτων πρόσημου ισούται με n.)*

Από το θεώρημα όμως του Sturm του 1829 ξέρουμε πως ο αριθμός των πραγματικών ριζών της p(x) = 0 που βρίσκονται στο διάστημα (-ο, +∞) ισούται με $\boldsymbol{v}_{-\infty} - \boldsymbol{v}_{+\infty}$, *όπου* $\boldsymbol{v}_{-\infty}$, $\boldsymbol{v}_{+\infty}$ *είναι οι μεταβολές πρόσημου των αριθμητικών ακολουθιών* $S_{seq}(-\infty)$ *και* $S_{seq}(\infty)$ *αντίστοιχα. Στην προκειμένη περίπτωση το p(x) έχε n - 2i πραγματικές ρίζες, που σημαίνει πως έχε 2i μιγαδικές ρίζες που εμφανίζονται σε ζευγάρια. Επομένως έχει i ζευγάρια μιγαδικών ριζών.//*

Παράδειγμα:

Έστω το πολυώνυμο $p(x) = x^5 - 3x^3 + 7x^2 - 28x + 28$ *και η αντίστοιχη ακολουθία Sturm,* $S_{seq}(x)$

```
p[x_] = (x - 2 i) (x + 2 i) (x^3 - 7 x + 7)//Expand;
Sseq[ x_] = createSturmSequenceIC[ p[ x]]
```

$\{28 - 28x + 7x^2 - 3x^3 + x^5, -28 + 14x - 9x^2 + 5x^4, -140 + 112x - 21x^2 + 6x^3, -4564 + 2352x + 493x^2, 520304 - 350941x, -1\}$

Προσέξτε τις μεταβολές και σταθερότητες πρόσημου στις ακολουθίες $S_{seq}(-\infty)$ *και* $S_{seq}(+\infty)$.

```
Limit [Sseq [x] , x -> -∞]
```

$\{-\infty, \infty, -\infty, \infty, \infty, -1\}$

```
Limit[ Sseq[ x] , x -> ∞]
```

$\{\infty, \infty, -\infty, \infty, -\infty, -1\}$

Το *πολυώνυμό μας έχει ένα ζευγάρι μιγαδικών ριζών και αυτό φαίνεται από το γεγονός ότι η ακολουθία* $S_{seq}(+\infty)$ *έχει μία μεταβολή πρόσημου. Πράγματι, είτε έχουμε*

```
variations[ Limit[ Sseq[ x] , x -> ∞] ]
```

1

είτε

```
variations[Map[Last[CoefficientList[#, x]] &, Drop[Sseq[x],1]]]
```

1

όπου η παραπάνω αριθμητική ακολουθία αποτελείται από τους συντελεστές των πρώτων όρων των 5 τελευταίων συναρτήσεων της ακολουθίας Sturm

```
Map[ Last[ CoefficientList[ #, x]] &, Drop[ Sseq[ x] ,1]]
```

{ 5, 6, 493, -350941, -1}

6.2.3 Η κλασσική μέθοδος διχοτόμησης του Sturm για την απομόνωση των πραγματικών ριζών ενός πολυωνύμου

Το θεώρημα του Sturm μπορεί να χρησιμοποιηθεί για την απομόνωση των πραγματικών ριζών πολυωνυμικών εξισώσεων με ακέραιους συντελεστές κα χωρίς πολλαπλές ρίζες. Στην ενότητα αυτή περιγράφουμε την κλασσική μέθοδο διχοτόμησης του Sturm, ενώ στην επόμενη ενότητα, 6.2.4, θα μελετήσουμε έναν αλγόριθμο για την διάσπαση ενός πολυωνύμου σε παράγοντες χωρίς πολλαπλές ρίζες.

Σύμφωνα με την αρχική, κλασσική, πρόταση του Sturm, ο π ο αποτελεσματικός τρόπος για την απομόνωση των πραγματικών ριζών του πολυωνύμου p(x) είναι να απομονώσουμε ***πρώτα τις θετικές ρίζες*** *του p(x) και* ***ύστερα τις αρνητικές*** *- αφού πρώτα τις κάνουμε θετικές με την αντικατάσταση x←-x.*

Το πρώτο και άμεσο πλεονέκτημα της πρότασης του Sturm είναι ότι για τα συμμετρικά πολυώνυμα — δηλαδή εκείνα για τα οποία ισχύει p(x) = p(-x) ή με άλλα λόγια εκείνα γ α τα οποία ο θετικές τους ρίζες ισούνται (σε απόλυτες τιμές) με τις αρνητικές τους — η *απομόνωση των αρνητικών ριζών καθίσταται περιττή!*

Ακολουθώντας την πρόταση του Sturm, και υποθέτοντας πως p(x)≠ 0 — διότι *αλλιώς αντικαθιστούμε το p(x) με το* $\frac{p\,x}{x}$ και *επιστρέφουμε το διάστημα (0, 0)-υπολογίζουμε* ***πρώτα*** *την ακολουθία Sturm για το p(x) και ύστερα βρίσκουμε κάποιο διάστημα (0,* b_+*),* $b_+ \in$ Q, που περιέχει *τις θετικές ρίζες. Στην ενότητα 6.2.5 θα μάθουμε να υπολογίζουμε ένα πάνω φράγμα στις τιμές των θετικών ριζών ενός πολυωνύμου. Αν p(*b_+*) = 0, επιστρέφουμε το διάστημα (*b_+*,* b_+*).*

Στην ***συνέχεια*** *η απομόνωση των θετικών ριζών του p(x) προχωράει ως εξής: Με την βοήθεια της ακολουθίας Sturm υπολογίζουμε τον ακριβή αριθμό των θετικών ριζών* $\rho+ = v_0 - v_{b+}$ *στο διάστημα*
(0, b_+*). Αν* $\rho_+ = 0$*, το διάστημα (0,* b_+*) δεν έχει θετικές ρίζες και δεν λαμβάνεται πλέον υπ όψη. Αν* $\rho_+ = 1$*, τερματίζει η απομόνωση των θετικών ριζών κα επιστρέφουμε το διάστημα (0,* b_+*). Αν* $\rho_+ > 1$*, ελέγχουμε αρχικά αν το μεσαίο σημείο* $m_p = \frac{b+}{2}$ *είναι ρίζα του p(x). Αν το αποτέλεσμα του τεστ είναι θετικό, επιστρέφουμε το διάστημα (*m_p*,* m_p*), κα εξετάζουμε τα υποδιαστήματα (0,* m_p*) και (*m_p*,* b_+*).*

Σε κάθε ένα από τα δύο αυτά υποδιάστηματα υπολογίζεται ο ακριβής αριθμός ρ_{v+} *των πραγματικών ριζών. Αν* $\rho_+ = 0$ *για κάποιο από αυτά, το αντίστοιχο υποδιάστημα δεν λαμβάνεται πλέον υπ όψη. Αν* $\rho_{v+} > 1$*, ενεργούμε όπως και πριν.*

Αφού απομονώσουμε τις θετικές ρίζες κάνουμε την αντικατάσταση x — -x, οπότε ο αρνητικές ρίζες γίνονται θετικές, ύστερα βρίσκουμε κάποιο διάστημα (0, b_-), $b_- \in Q$, που περιέχει τις "τέως" αρνητικές — αλλά τώρα θετικές — ρίζες, κα επαναλαμβάνουμε την ίδια διαδικασία.

Η μέθοδος σταματάει όταν έχουμε απομονώσει όλες τις πραγματικές ρίζες. Ο αριθμός των υποδιαιρέσεων — κα εν γένει ο θεωρητικός χρόνος για την απομόνωση των ριζών — εξαρτάται από το πόσο κοντά είναι οι ρίζες. Στην ενότητα 622.6 υπολογίζουμε ένα κάτω φράγμα στην απόσταση μεταξύ δύο τυχαίων ριζών ενός πολυωνύμου.

Ακολουθώντας την πρόταση του Sturm το αρχικό διάστημα (0, b_+) που χρησιμοποιούμε πρώτα για τις θετικές ρίζες, είναι εντελώς διαφορετικό από το αντίστοιχο αρχικό διάστημα (0, b_-) που χρησιμοποιούμε στη συνέχεια για τις αρνητικές ρίζες. Συνεπώς, με αυτόν τον τρόπο ***περιορίζουμε στο ελάχιστο τις άσκοπες υποδιαιρέσεις διαστημάτων και τους ελέγχους για ύπαρξη ριζών σε αυτά.***

Η πρόταση αυτή του Sturm εφαρμόσθηκε από τους Γάλλους μαθηματικούς του 19ου αιώνα, αλλά μετά περιέπεσε σε λήθη. Επανήλθε στο "φως" από τον γράφοντα, το 1978, στην διδακτορική του διατριβή.

Έτσι, τον εικοστό αιώνα και μέχρι το 1978, σαν αρχικό διάστημα χρησιμοποιούταν το διάστημα (-b, b), όπου b είναι ένα φράγμα στις ***απόλυτες τιμές των ριζών*** *του p(x). Στην σχετική βιβλιογραφία υπάρχουν διάφορα θεωρήματα για τον υπολογισμό διαφόρων τιμών αυτού του b, τα οποία δεν θα μελετήσουμε εδώ.*

Το μειονέκτημα όμως, της στρατηγικής αυτής είναι ότι οι αρνητικές και οι θετικές ρίζες δεν είναι κατ' ανάγκη ομοιόμορφα κατανεμημένες. Έτσι, σε απόλυτες τιμές, μπορεί να συμβεί οι μεν αρνητικές να είναι πολύ μεγάλες οι δε θετικές πολύ μικρές (ή κα αντίστροφα). Στην περίπτωση αυτή ***άσκοπα*** *θα* ***υποδιαιρούμε*** *πολλά διαστήματα, τα οποία κα θα* ***ελέγχουμε*** *για ρίζες που δεν έχουν.*

Ακολουθεί η κλασσική μέθοδος της διχοτόμησης του Sturm για την απομόνωση των πραγματικών ριζών πολυωνυμικών εξισώσεων με ακέραιους συντελεστές κα χωρίς πολλαπλές ρίζες.

Αλγόριθμος: Ο κλασσικός αλγόριθμος του* Sturm (1829) *για τις θετικές ρίζες.

Είσοδος: p(x) = 0, μία πολυωνυμική εξίσωση με ακέραιους συντελεστές, χωρίς πολλαπλές ρίζες και

p(0)≠ 0.

*Έξοδος: Τα διαστήματα απομόνωσης των **θετικών** ριζών του p(x) ή ο ακριβείς θετικές ρίζες σε μορφή διαστημάτων.*

1. *Ορίζουμε την λίστα των διαστημάτων απομόνωσης των ριζών rootIsolationIntervals = { }, καθώς και την λίστα των διαστημάτων προς εξέταση intervalsToBeProcessed = { }. Αρχικά οι δύο αυτές λίστες είναι κενές. Υπολογίζουμε* επίσης την *ακολουθία Sturm του p(x) με την βοήθεια της συνάρτησης createSturmSequenceIC[].*

2. *Με την συνάρτηση CauchyPositiveRootUpperBound[], που περιγράφεται στην ενότητα 6.2.5, υπολογίζουμε ένα πάνω φράγμα, $b_+ \in Q$, στις τιμές των θετικών ριζών του p(x). Το b_+ υπολογίζεται κατά τέτοιο τρόπο ώστε είναι **γνησία** μεγαλύτερο από κάθε θετική ρίζα του p(x). Επιπλέον ορίζουμε το αρχικό διάστημα (l, r) = (0, b_+), και το επισυνάπτουμε στην λίστα intervalsToBeProcessed.*

3. *(* επεξεργασία διαστήματος *)*

*Μέχρις ότου η λίστα intervalsToBeProcessed = { }, δηλαδή μέχρι να "αδειάσει", **επαναλαμβάνουμε** την εξής διαδικασία: Παίρνουμε το πρώτο διαθέσιμο διάστημα ((, f). Με την βοήθεια του θεωρήματος του Sturm (1829) και της συνάρτησης variations[] υπολογίζουμε τον αριθμό των θετικών ριζών $\rho_+ = v_l - v_r$ στο διάστημα (l, f). Αν $\rho_+ = 0$, το διάστημα (l, f) δεν έχει θετικές ρίζες και δεν λαμβάνεται πλέον υπ᾽ όψη. Αν $\rho_+ = 1$, επισυνάπτουμε το διάστημα (l, r) στην λίστα rootIsolationIntervals. Αν $\rho_+ > 1$, τότε (α) θέτουμε $m_p = \frac{l+r}{2}$, υποδιαιρούμε το διάστημα (l, r) στα υποδιαστήματα (l, m_p) και (m_p, r), τα οποία επισυνάπτουμε στην λίστα intervalsToBeProcessed, διαστημάτων προς εξέταση, κα **(β)** στην περίπτωση που $p(m_p) = 0$ επισυνάπτουμε το διάστημα (m_p, m_p) στην λίστα rootIsolationIntervals.*

*Για την απομόνωση των **αρνητικών ριζών** πρώτα εξετάζουμε αν p(x) = p(-x). Αν ισχύει η ισότητα, αυτό σημαίνει πως ο αρνητικές ρίζες είναι συμμετρικές*

με τις θετικές για τις οποίες έχουμε ήδη υπολογίσει τα διαστήματα απομόνωσής τους. Άρα στην περίπτωση αυτή τα διαστήματα απομόνωσης των αρνητικών ριζών βρίσκονται στοιχειωδώς. Αν p(x)≠ p(-x) τότε θέτουμε p(x)<- p(-x), επαναλαμβάνουμε τον παραπάνω αλγόριθμο ακόμα μία φορά και στο τέλος απεικονίζουμε τα διαστήματα απομόνωσης των ριζών στον αρνητικό ημιάξονα.

Όσον αφορά το 0, εύκολα ελέγχουμε αν p(0) = 0, και στην περίπτωση αυτή θέτουμε p(x)= $\frac{p\ x}{x}$ *.Ακολουθεί ο παραπάνω αλγόριθμος εφαρμοσμένος στο Mathematica. Γι α τον αλγόριθμο αυτό χρειάζεται να έχουν ενεργοποιηθεί οι συναρτήσεις variations[], CauchyPositiveRootUpperBound[], και createSturmSequenceIC[].*

```
SturmPositiveRootIsolation[p_] :=
Module[{b, intervalsToBeProcessed, left, midPoint, posroots, right, rootIsolationIntervals, Sseq,
v = First[Variables[p]]},

(*    step 1,  initialization *)
rootIsolationIntervals = {} ; intervalsToBeProcessed=
{};
Sseq = createSturmSequenceIC[ p] ;

(*    step 2,  initialization continued *) b =
CauchyPositiveRootUpperBound[ p] ;

left = 0; right = b;
AppendTo[ intervalsToBeProcessed, { left, right}] ;

(*    step 3, processing of intervals *)
While[intervalsToBeProcessed≠{},

{ left, right} = First[ intervalsToBeProcessed];
intervalsToBeProcessed = Rest[ intervalsToBeProcessed]; posroots
= variations [Sseq / . v  - > left] − variations [Sseq / . v  - >
right] ;

Switch[ posroots,0, Null,1,AppendTo[ rootIsolationIntervals,
                                          left□right
{ left, right}] ; Continue[] ,, midPoint=      2      ;
AppendTo[intervalsToBeProcessed, { left, midPoint}] ; AppendTo[
intervalsToBeProcessed, { midPoint, right}] ; If [(p / . v  - >
midPoint) == 0 ,  AppendTo[rootIsolationIntervals, {midPoint,
midPoint}]]]]; Sort[rootIsolationIntervals]]
```

Έτσι βλέπουμε πως οι ***θετικές*** *ρίζες του πολυωνύμου p(x) =* x^3 *-7x + 7 βρίσκονται στα διαστήματα απομόνωσης* $(0, \frac{3}{2})$ *και* $(\frac{3}{2},3)$:

```
p [x_] = x^3 - 7 x + 7; SturmPositiveRootIsolation[p [x]]
```

$\{\{0, \frac{3}{2}\}, \{\frac{3}{2}, 3\}\}$

ενώ η *μοναδικά* ***αρνητική*** *βρίσκεται στο διάστημα* (-4, 0).

```
SturmPositiveRootIsolation[ p[ - x]]
```

```
{{0, 4}}
```

Προφανώς το 0 *δεν είναι ρίζα τον p(x).*

```
p[0] == 0
```

```
False
```

Τελειώνουμε την περιγραφή, τον αλγόριθμο αυτού τονίζοντας ότι *η εφαρμογή τον θα μπορούσε να βελτιωθεί στο εξής σημείο: Έστω (l, r) ένα διάστημα.* Κατ' *αρχάς υπολογίζουμε τον αριθμό των μεταβολών πρόσημων των ακολουθιών* Sseq(l) *και* Sseq(r). Αν *χρειαστεί να υποδιαιρέσουμε το διάστημα στα υποδιαστήματα (l,* midPoint), και (midPoint, *r) τότε υπολογίζουμε ξανά τον αριθμό των μεταβολών πρόσημων των ακολουθιών* Sseq(f) και Sseq(midPoint) *ή* / και Sseq(r) και Sseq(midPoint). *Δηλαδή υπολογίζουμε δύο φορές τον αριθμό των μεταβολών πρόσημων των ακολουθιών* Sseq(l) και Sseq(r). *Αυτό μπορεί να αποφευχθεί, αλλά χάνεται* η *απλότητα τον αλγόριθμο — κάτι που μας ενδιαφέρει περισσότερο εδώ.*

***Ανάλυση του χρόνου υπολογισμού της μεθόδου του* Sturm:**

Όπως αναφέραμε, η μέθοδος τον Sturm *είναι μία μέθοδος απομόνωσης των πραγματικών ριζών ενός πολυωνύμου p(x) με διχοτόμηση. Ο αριθμός των διχοτομήσεων εξαρτάται από το πόσο κοντά είναι οι ρίζες τον* p(x). *Αν το p(x) έχει k διαφορετικές ρίζες, α\, ..., , k* > 2, *ορίζουμε την* ***ελάχιστη απόσταση των ριζών*** (minimum root separation) τον p(x) ως

$\Delta = \min_{1 \le i < j \le k} |\alpha_i - \alpha_j| > 0$

Αν $k = 1$, *τότε* $\Delta = \infty$. *Όπως θα δούμε στην ενότητα* 6.2.6, αν $n \geq 2$ *είναι ο βαθμός του πολυωνύμου* p(x), *τότε ένα κάτω φράγμα στο* Δ *δίνεται από τον τύπο:*

$$\Delta \geq \sqrt{3} \times n^{\frac{-(n+2)}{2}} \times |p(x)|_1^{-(n-1)}$$

Αντιστρέφοντας την παραπάνω ανισότητα και παίρνοντας λογαρίθμους έχουμε

$$\log \Delta^{-1} \leq \log 3^{-\frac{1}{2}} + \frac{n+2}{2} \log n + (n-1) \log |p(x)|_1 \quad \acute{\eta}$$

$$\log \Delta^{-1} = O(n \log n + n \log |p(x)|_\infty),$$

όπου αντί τον $|p(x)|_1$ *χρησιμοποιούμε το* $|p(x)|_\infty$ που *είναι της ίδιας τάξης μεγέθους. Επιπλέον, λαμβάνοντας* υπ' *όψη ότι το β-μήκος τον βαθμού των πολυωνύμων για τις περιπτώσεις που εξετάζουμε είναι* $\lambda(n) = 1$, *ή* $\log n = 1$, *προκύπτει*

$\log \Delta^{-1} = O(n \log |p(x)|_\infty)$.

Επειδή η *τάξη μεγέθους των λογαρίθμων ως προς οποιανδήποτε βάση είναι η ίδια, έπεται πως ισχύει*

$\log_2 \Delta^{-1} = O(n \log |p(x)|_\infty)$,

που *σημαίνει πως η παραπάνω έκφραση είναι ένα* ***πάνω φράγμα στον αριθμό των διχοτομήσεων*** *που γίνονται για την απομόνωση των ριζών του πολυωνύμου* p(x) *με την ελάχιστη απόσταση. Δηλαδή, για την απομόνωση των δύο πλησιέστερων ριζών τον* p(x), ***ο αριθμός των διχοτομήσεων*** *είναι περίπου όσο το γινόμενο τον βαθμού τον* p(x) *επί τον αριθμό των ψηφίων του μεγαλύτερου συντελεστή του.*

Ας δούμε τώρα τον χρόνο υπολογισμού κάθε βήματος του αλγορίθμου. Στο ***πρώτο βήμα*** *υπολογίζουμε την ακολουθία* Sturm τον p(x), Sseq(x), το *κόστος της οποίας — όπως θα δούμε στο κεφάλαιο* 7 *— είναι*

$O(n^5 \log^2 |p(x)|_\infty)$.

Στο ***δεύτερο βήμα*** *υπολογίζουμε ένα πάνω φράγμα* b *στις τιμές των θετικών ριζών τον* p(x), *το κόστος τον οποίον — όπως θα δούμε στην ενότητα* 6.2.5 *— είναι* O(n).

Στο ***τρίτο βήμα*** *εκτιμούμε την ακολουθία* Sturm τον p(x), Sseq(x), σε *διάφορα ρητά σημεία — που είναι είτε ακραία είτε μεσαία σημεία διαστημάτων. Έστω* $\frac{a}{d}$, $d > 0$, μη *μηδενικός,* ***θετικός*** *ρητός αριθμός και* $p(x) = \sum_{i=0}^{n} c_i x^i$ *τυχαίο πολυώνυμο μέλος της ακολουθίας* Sseq(x). *Τότε το*

πρόσημο τον $p(\frac{a}{d})$ *είναι το ίδιο με το πρόσημο τον* $p(x)= \sum_{i=0}^{n} c_i a^i d^{n-i}$, *το οποίο υπολογίζεται μόνο με αριθμητική ακεραίων! Από την ενότητα* 4.1 *ξέρουμε πως το κόστος ενός τέτοιου υπολογισμού με την μέθοδο* Ruffini-Horner *είναι* $O(n^2 \log^2 e \log |p(x)|_\infty)$, *όπου* $e = \max(a, d)$ *και* $\log e \leq |\log \Delta^{-1}| = O(n \log |p(x)|_\infty)$.

Συνεπώς, ο χρόνος υπολογισμού ***κάθε μιας*** *από τις εκτιμήσεις των πολυωνύμων της ακολουθίας* Sturm *τον* p(x), *σε ρητό σημείο είναι*

$O(n^4 \log^2 |p(x)|_\infty)$.

Αν τώρα λάβουμε υπ' όψη ότι: **1ον,** *η ακολουθία* Sturm *τον* p(x), Sseq(x), *έχει* $n + 1$ *πολυώνυμα,* **2ον,** *για τις δύο πλησιέστερες ρίζες θα γίνουν το πολύ* $O(n \log |p(x)|_\infty)$ *διχοτομήσεις, και* **3ον,** *στην χειρότερη περίπτωση οι ρίζες ανά δύο — δηλαδή συνολικά* $\frac{n}{2}$ *ζεύγη — θα βρίσκονται σε απόσταση Δ, τότε έπεται πως ο χρόνος απομόνωσης των πραγματικών ριζών τον p(x) με την μέθοδο διχοτόμησης τον* Sturm *είναι*

$O(n^7 \log^3 |p(x)|_\infty)$.

6.2.4 *Διάσπαση πολυωνύμου σε παράγοντες ελεύθερους από τετράγωνα*

Ένα πολυώνυμο p(x) ονομάζεται ***ελεύθερο από τετράγωνα*** (squarefree) *αν δεν υπάρχει πολυώνυμο q(x) θετικού βαθμού έτσι ώστε το* $q^2(x)$ *να διαιρεί το* p(x). *Εκτός από την απομόνωση των ριζών, η διαδικασία διάσπασης ενός πολυωνύμου σε παράγοντες ελεύθερους από τετράγωνα εφαρμόζεται και στην ολοκλήρωση ρητών συναρτήσεων. Προτού παρουσιάσουμε τον αλγόριθμο χρειαζόμαστε λίγη θεωρία.*

Θεώρημα:

Έστω J μια περιοχή μοναδικής παραγοντοποίησες χαρακτηριστικής μηδέν και έστω p(x) e J[x] *ένα μη σταθερό και* ***αρχέγονο*** (primitive) *πολυώνυμο (οι συντελεστές τον είναι πρώτοι μεταξύ τους). Έστω επί πλέον*

$P(x)= p1^{e1}(x)\, p2^{e2}(x) \ldots pn^{en}(x)$

η μοναδική παραγοντοποίηση τον p(x) σε γινόμενο ***μη αναγώγιμων*** (irreducible) *παραγόντων (όπου* p_f (x) *συμβολίζει τον παράγοντα* $p_i(x)$ *υψωμένο στην δύναμη* e_i*), και* $p'(x)$ *η πρώτη παράγωγός τον. Τότε*

$\gcd(p(x), p'(x)) = p1^{e1-1}(x)\, p2^{e2-1}(x) \ldots pn^{en-1}(x)$

Απόδειξη:

Έστω $q(x) = \prod_{i=2}^{n} pi^{ei}(x)$ *και* $r(x) = \gcd(p(x), p'(x))$. *Τότε έχουμε* $p(x) = p1^{e1}(x)\, q(x)$ *και*

$P'(x) = p1^{e1}(x)\, q'(x) + e1p1^{e1-1}(x)\, p1'(x)\, q(x)$

από όπου συνεπάγεται πως το $p1^{e1-1}(x)$ ***διαιρεί*** *το* $r(x)$. *Με την εις άτοπο απαγωγή θα δείξουμε πως το* $p1^{e1}(x)$ ***δεν διαιρεί*** *το r(x). Έστω λοιπόν πως το* $p1^{e1}(x)$ *διαιρεί το r(x). Τότε το* $p1^{e1}(x)$ *διαιρεί την παράγωγο* $p'(x)$, *απ' όπου συνεπάγεται πως το* $p1^{e1}(x)$ *διαιρεί την έκφραση* $e1p1^{e1-1}(x)\, p1'(x)\, q(x)$. *Μετά από απαλοιφή όμως προκύπτει ότι το* $p_1(x)$ *διαιρεί την έκφραση* $e_1 p'_1(x) q(x)$.

Επί πλέον, επειδή τα πολυώνυμα $p_i(x)$ *είναι πρώτα μεταξύ τους, ισχύει* $\gcd(p_1(x), q(x)) = 1$ *και συνεπώς πρέπει το* $p_1(x)$ *να διαιρεί την έκφραση* $e_1\, p'_1(x)$. *Αυτό όμως είναι αδύνατο διότι* $\deg(p_1(x)) > \deg(p'_1(x))$ *Άρα στο r(x) ο βαθμός τον* $p_1(x)$ *είναι* $e_1 - 1$ και λόγω *συμμετρίας ισχύει* $r(x) = p1^{e1-1}(x)\, p2^{e2-1}(x) \ldots pn^{en-1}(x)$, *πράγμα που θέλαμε να* δείξουμε.//

Από το παραπάνω θεώρημα συμπεραίνουμε πως αν $\gcd(p(x), p'(x)) = 1$, *τότε το* $p(x)$ ***δεν έχει πολλαπλές ρίζες,*** *και αντίστροφα. Ισχύουν επίσης τα εξής:*

Λήμμα 1:

Οι απλές ρίζες ενός πολυωνύμου δεν είναι ρίζες της παραγώγου τον.

Λήμμα 2

Έστω J σώμα και $p(x) \in J[x]$ *ένα* ***μη αναγώγιμο*** *πολυώνυμο που διαιρεί το* $s(x) \in J[x]$. *Τότε το* $p^2(x)$ *διαιρεί το s(x) εάν και μόνο εάν το p(x) διαιρεί το s(x).*

Απόδειξη:

Επειδή το p(x) διαιρεί το s(x) μπορούμε να γράψουμε $s(x) = p(x)q(x)$, *απ' όπου προκύπτει* $S'(x) = p'(x)q(x) + p(x)q'(x)$. Συνεπώς, αν το $p^2(x)$ *διαιρεί το s(x) τότε το p(x) διαιρεί το q(x) και προφανώς το p(x) διαιρεί το* $s'(x)$. *Αντιστρόφως, αν το p(x) διαιρεί το* $s'(x)$ *τότε το p(x) διαιρεί την έκφραση* $p'(x)q(x)$, που *συνεπάγεται πως το p(x) διαιρεί είτε το p(x) είτε το* $q'(x)$ ***(βλέπε άσκηση ??***). *Επειδή όμως* $\deg(p(x)) > \deg(p'(x))$ *έπεται πως το p(x) διαιρεί το* $q(x)$, και *συνεπώς το* $p^2(x)$ *διαιρεί το* $s(x)$

Ας δούμε τώρα τον αλγόριθμο διάσπασης ενός πολυωνύμου σε παράγοντες ελεύθερους από τετράγωνα.

Έστω J μια περιοχή μοναδικής παραγοντοποίησης χαρακτηριστικής μηδέν και έστω p(x) ∈J[x], ένα αρχέγονο πολυώνυμο μιας μεταβλητής και θετικού βαθμού. Υποθέτουμε επί πλέον πως

$P(x)= p1^{e1}(x)\ p2^{e2}(x) \ldots pn^{en}(x)$ *είναι η μοναδική παραγοντοποίηση τον p(x) σε γινόμενο* ***μη αναγώγιμων*** *παραγόντων $p_i(x)$, κάθε ένας των οποίων έχει θετικό βαθμό $e_i > 0$. Ορίζουμε $e = \max(e_1, ..., e_n)$* και για *κάθε* $1 \leq i \leq e$.

$J_i = \{j : e_i = 1\}$, $S_i(x) = \Pi_{j \in J_i} P_j(x)$

Προφανώς, *ισχύει* η *σχέση*

$$p(x) = \prod_{i=2}^{e} Si^{i}(x)$$

η οποία ονομάζεται ***ανάλυση σε παράγοντες ελεύθερους από τετράγωνα*** (squarefree factorization).

Τα πολυώνυμα $s_i(x)$ είναι οι ***παράγοντες οι ελεύθεροι από τετράγωνα*** (squarefree factors) και *μερικοί από αυτούς μπορεί να είναι ίσοι με την μονάδα. Το $s_1(x)$ είναι το γινόμενο όλων των παραγόντων με απλές ρίζες, το $s_2(x)$ είναι το γινόμενο όλων των παραγόντων με διπλές ρίζες, κοκ.*

Τα πολυώνυμα $s_i(x)$ υπολογίζονται με την βοήθεια τον θεωρήματος ως εξής: Πρώτα υπολογίζουμε το r(x), τον μέγιστο κοινό διαιρέτη (μκδ) των p(x), και $p'(x)$

$$r(x) = \gcd(p(x), p'(x)) = \prod_{i=1}^{n} pi^{e-i}(x) = \prod_{i=1}^{e} sj^{i-1}(x),$$

όπου δεν εμφανίζεται το $s_1(x)$. Κατόπιν, βρίσκουμε το t(x), που *είναι για το p(x) ο μεγαλύτερος διαιρέτης τον ελεύθερος από τετράγωνα*

$$t(x) = \frac{p(x)}{r(x)} = \prod_{i=1}^{n} p_i(x) = \prod_{i=1}^{e} Si(x).$$

Τέλος, βρίσκοντας τον μκδ των r(x) και t(x)

$$v(x) = \gcd(r(x), t(x)) = \prod_{i=1}^{e} Si(x).$$

προκύπτει πως $s_1(x) = \frac{t(x)}{v(x)}$ *. Δηλαδή ο πρώτος παράγοντας ελεύθερος από τετράγωνα,* S1(x), *βρίσκεται με παραγώγιση, υπολογισμούς μκδ και διαίρεση.*

Επαναλαμβάνοντας την παραπάνω διαδικασία με το r(x) στον ρόλο τον p(x) βρίσκουμε τον δεύτερο παράγοντα ελεύθερο από τετράγωνα, $s_2(x)$, κοκ.

Προσοχή:

Αν ακολουθήσουμε την παραπάνω διαδικασία θα εκτελέσουμε ***δύο*** *υπολογισμούς μκδ για κάθε παράγοντα* $s_i(x)$*. Επειδή όμως γνωρίζουμε την μορφή των πολυωνύμων* $r(x)$ *μπορούμε να εκτελέσουμε* ***έναν*** *υπολογισμό μκδ για κάθε παράγοντα* $s_i(x)$ *κάνοντας τις αντικαταστάσεις* $r(x) \leftarrow \frac{r\,x}{v\,x}$ *και* $t(x) \leftarrow v(x)$*.Έτσι ο αλγόριθμος είναι ως εξής:*

Αλγόριθμος: Διάσπαση πολυωνύμου σε παράγοντες ελεύθερους από τετράγωνα.

Είσοδος: $p(x) \in J[x]$*, ένα αρχέγονο πολυώνυμο θετικού βαθμού με συντελεστές από το J, μια περιοχή μοναδικής παραγοντοποίησης χαρακτηριστικής μηδέν. Έξοδος: Η παραγοντοποίηση* $p(x)= \prod_{i=1}^{e} \text{si}^{\text{i}}(\text{x})$*, σε παράγοντες ελεύθερους από τετράγωνα. Μερικά από τα πολυώνυμα* $s_i(x)$ *μπορεί να είναι ίσα με την μονάδα*

1. (* *έναρξη* *)

sqfreefactorList = {}; r(x) ←gcd(p(x), p'(x)); t(x) $\leftarrow \frac{p(x)}{\text{r(x)}}$.

2. (* *υπολογισμός των Sj(x)* *)

Εφόσον *ο βαθμός τον* $r(x)$ *είναι μεγαλύτερος τον μηδενός* deg(r(x)) > 0, ***επαναλαμβάνουμε*** *την εξής διαδικασία: Υπολογίζουμε το Sj(x) με τις εντολές* $(x) \leftarrow \gcd(r(x),t(x))$; $^{S_j}(x) \leftarrow \frac{t\,x}{v\,x}$, το *επισυνάπτουμε στην λίστα* sqfreefactorList και *ανανεώνουμε τα πολυώνυμα* $r(x) \leftarrow \frac{r\,x}{v\,x}$ *και* $t(x) \leftarrow v(x)$. **Αλλιώς** *επισυνάπτουμε στην λίστα* sqfreefactorList *το πολυώνυμο t(x).*

3. (* *τέλος* *)

Την λίστα sqfreefactorList *την μετατρέπουμε κατόπιν στο γινόμενο* $\prod_{i=1}^{e} \text{si}^{\text{i}}(\text{x})$*,, όπου e ισούται με το μήκος της λίστας.*

Ακολουθεί ο παραπάνω αλγόριθμος εφαρμοσμένος στο Mathematica:

```
squareFreeFactors[f_] : =
Module [{e, facList = {} , j = 1, r, s, sqfreefactorList = {},
p = f, t, v, var = First [Variables [f ]] } ,
```

```
(* step 1 *)
r = PolynomialGCD [p, D [p, var]] ;
t = PolynomialQuotient[ p, r, var] ;

(* step 2 *)

While[Exponent[r, var] >0 ,
v = PolynomialGCD[ r, t] ;
s = PolynomialQuotient[t, v, var] ;
AppendTo[ sqfreefactorList, s] ;
r = PolynomialQuotient[ r, v, var] ;t = v ] ;
AppendTo[ sqfreefactorList, t] ;

(* step 3 *)
Map[(If[# =!= 1, AppendTo[facList,  #^j ]] ;j += 1) &, sqfreefactorList] ;
Apply[Times, facList]^1
```

Παράδειγμα:

Εφαρμόζοντας τον αλγόριθμο squareFreeFactors[] στο *πολυώνυμο p(x)* = $x^5 - x^4 - 2x^3 + 2x^2 + x - 1$ *προκύπτουν τα εξής ενδιάμεσα αποτελέσματα:*

*Την **πρώτη** φορά που εκτελείται ο βρόγχος* While[] , ο *βαθμός τον p(x) είναι* 5, $r(x) = x^3 - x^2 - x + 1$, $t(x) = x^2 - 1$, $v(x) = x^2 - 1$ και $s_1(x) = 1$, που *σημαίνει ότι δεν υπάρχουν παράγοντες πρώτον βαθμού.*

*Την **δεύτερη** φορά που εκτελείται ο βρόγχος* While[] , *ο βαθμός τον p(x) είναι* 3, $r(x) = x - 1$, $t(x) = x^2 - 1$, $v(x) = x - 1$ και $s_2(x) = x + 1$, που *σημαίνει ότι το* $(x + 1)^2$ *είναι παράγοντας δευτέρου βαθμού.*

*Την **τρίτη** —* και *τελευταία — φορά που εκτελείται ο βρόγχος* While[] , ο *βαθμός τον p(x) είναι* 1, $r(x) = 1$, $t(x) = x - 1$, $v(x) = 1$ και $s_3(x) = x - 1$, που *σημαίνει ότι το* $(x - 1)^3$ *είναι παράγοντας τρίτον βαθμού.*

Πράγματι, το πολυώνυμο p(x) = $x^5 - x^4 - 2x^3 + 2x^2 + x - 1$ *αναλύεται στο εξής γινόμενο παραγόντων ελεύθερων από τετράγωνα:*

```
p[x_] = x^5 - x^4 - 2 x^3 + 2 x^2 + x - 1; squareFreeFactors [p[x]]
```

$(-1+x)^3\ (1+x)^2$

Το *αποτέλεσμα είναι το ίδιο με αυτό που προκύπτει χρησιμοποιώντας την συνάρτηση* FactorSquareFree[] τον *Mathematica.*

```
FactorSquareFree[ p[ x]]
```

$(-1+x)^3\ (1+x)^2$

Ανάλυση του χρόνου υπολογισμού της παραγοντοποίησης σε παράγοντες ελεύθερους από τετράγωνα:

Είναι προφανές πως ο χρόνος υπολογισμού τον αλγορίθμου αυτού καθορίζεται από τον χρόνο υπολογισμού των μκδ (gcd) στο 2ο *βήμα. Επιπλέον, σημειώνουμε πως η πρώτη εκτέλεση τον μκδ είναι και η πιο χρονοβόρα — από όλες τις άλλες.*

Αν n= deg(p(x)), *τότε n είναι ένα πάνω φράγμα στον αριθμό εκτελέσεων τον βρόγχου* While[]. *Διακρίνουμε* 2 *περιπτώσεις:*

***Περίπτωση* 1η:** *Αν p(x)* ∈ *J[x], και J είναι σώμα, τότε ο μκδ* gcd(p(x), *p* (x)) *υπολογίζεται σε χρόνο O(* n^2*) και επομένως το 2ο βήμα —* και *συνεπώς ο αλγόριθμος — εκτελείται σε χρόνο O(* n^3).

Περίπτωση 2η: *Αν p(x)* ∈ J[x], και *J* = Z, τότε, *όπως θα δούμε στο κεφάλαιο* 7, ο μκδ gcd(p(x), p (x)) *υπολογίζεται σε χρόνο O(* $n^5 \log^2 |p(x)|_\infty$*) και επομένως το* 2*ο βήμα — και συνεπώς ο αλγόριθμος — εκτελείται σε χρόνο O(* $n^6 \log^2 |p(x)|_\infty$).

6.2.5 Πάνω *και κάτω φράγμα στις τιμές των θετικών ριζών ενός πολυωνύμου*

Στην ενότητα αυτή *παρουσιάζουμε και αποδεικνύουμε το θεώρημα του* Cauchy *για την εύρεση* ενός πάνω *φράγματος στις τιμές των θετικών ριζών μίας πολυωνυμικής εξίσωσης p(x)* = 0, με *ακέραιους συντελεστές. Σημειώνουμε πως το σημαντικό αυτό αποτέλεσμα βρισκόταν* ***μόνο*** *στο βιβλίο του* N. Obreschkoff στα ***Γερμανικά*** *και πως πριν το* 1978 δεν *υπήρχε σε κανένα βιβλίο της* Αγγλικής *βιβλιογραφίας! Στην* Αγγλική *βιβλιογραφία, όπως αναφέραμε, υπήρχαν τύποι μόνο για την εύρεση ενός φράγματος στις απόλυτες τιμές των ριζών.*

Εμείς χρησιμοποιούμε το θεώρημα του Cauchy και για την *εύρεση ενός κάτω φράγματος στις τιμές των θετικών ριζών της* p(x) = 0.

***Θεώρημα* (Cauchy):**

Έστω $p(x) = c_n x^n + c_{n-1}x^{n-1} + \ldots + c_1 x + c_0 = 0$ *μία πολυωνυμική εξίσωση βαθμού* $n > 0$, με ***ακέραιους*** *συντελεστές και* $c_{n-k} < 0$ για *τουλάχιστον ένα* k, $1 \le k \le n$. *(Προσέξτε πως* $c_n > 0$!)
Αν λ είναι ο αριθμός των αρνητικών συντελεστών τότε

$$b = \max_{\{1 \le k \le n\}:\, c_{n-k} < 0} \sqrt[k]{\frac{-\lambda c_{n-k}}{c_n}}$$

είναι ένα πάνω φράγμα στις τιμές των θετικών ριζών της p(x) = 0.

Απόδειξη:

Από τον τρόπο ορισμού του b συνεπάγεται πως

$$b^k \ge \left(\frac{-\lambda c_{n-k}}{c_n}\right)$$

για κάθε k έτσι ώστε $c_n k < 0$. Για αυτά τα k, η παραπάνω ανισότητα γράφεται και σαν

$$b^k \ge \left(\frac{-\lambda c_{n-k}}{c_n}\right) b^{n-k}$$

Αθροίζοντας για όλα τα κατάλληλα k έχουμε

$$\lambda b^n \ge \lambda \sum_{\{1 \le k \le n\}:\, c_{n-k} < 0} \left(\frac{-c_{n-k}}{c_n}\right) b^{n-k} \quad \text{ή}$$

$$b^n \ge \sum_{\{1 \le k \le n\}:\, c_{n-k} < 0} \left(\frac{-c_{n-k}}{c_n}\right) b^{n-k} \quad \text{ή}$$

Δηλαδή αν *διαιρέσουμε την p(x) = 0 με το c_n, κάνοντας μονάδα τον κύριο συντελεστή της, και αντικαταστήσουμε το x με το b, x — b, τότε ο πρώτος όρος, δηλαδή το b^n, θα είναι* ***μεγαλύτερος από,*** *ή* ***ίσος με,*** *το άθροισμα των απόλυτων τιμών των όρων με αρνητικούς συντελεστές. Συνεπώς, για όλα τα*

x > b, το p(x) > 0.//

Το πάνω *φράγμα b υπολογίζεται ευκολότατα από τον τύπο ορισμού του με το ακόλουθο πρόγραμμα στο Mathematica. Προσέξτε πως για να αποφύγουμε την περίπτωση της ισότητας στην έκφραση* $b^n \ge \sum_{\{1 \le k \le n\}:\, c_{n-k} < 0} \left(\frac{-c_{n-k}}{c_n}\right) b^{n-k}$ *τέλος του προγράμματος παίρνουμε σαν πάνω φράγμα την ποσότητα άνω φράγμα* [**65 / 64 b**]

```
CauchyPositiveRootUpperBound[p_] : =
Module [{b = 0 , cl, lc, lamda, n, cpos, tb}]
cl = CoefficientList[p, Variables[p]] ;
n = Length[cl] ; l c = Last[cl] ;
If [lc < 0 , cl = - cl; lc = -  lc] ;
lamda = Length[Select[cl, # < 0 & ]] ;
```

```
If [lamda == 0 , Return [b]] ;
cpos = Position[cl,    ? (# < 0 &) ] ;
```

```
Do [tb =  sqrt[n-cpos[[k]]]( - lamda cl[[cpos[[k]]]] / lc ) // N // First;
If [tb > b, b = tb] , {k, lamda}] ; Ceiling[65 / 64 b]]
```

Έτσι βλέπουμε πως το 3 *είναι ένα πάνω φράγμα στις τιμές των* ***θετικών*** *ριζών του* $p(x) = x^3 - 7x + 7$

```
p [x_] = x^3 - 7 x + 7;CauchyPositiveRootUpperBound[ p[ x]]
```

3

ενώ το 4 είναι ένα πάνω φράγμα στις τιμές των ***αρνητικών*** *ριζών του* $p(x) = x^3 - 7x + 7$, και *βρίσκεται αντικαθιστώντας το x με το -x, x — -x.*

```
CauchyPositiveRootUpperBound[p[-x]]
```

4

Ανάλυση του χρόνου υπολογισμού ενός πάνω φράγματος στις τιμές των θετικών ριζών πολυωνύμου:

Είναι *προφανές πως ο χρόνος υπολογισμού του αλγορίθμου αυτού καθορίζεται από τον χρόνο υπολογισμού των ριζών. Επειδή εδώ έχουμε αριθμητική κινητής υποδιαστολής, ένας υπολογισμός ριζικού εκτελείται σε χρόνο* 0(1) και *επομένως το πολύ n τέτοιοι υπολογισμοί εκτελούνται σε χρόνο 0(n).*

Όπως θα δούμε στην ενότητα 6.3, στην *μέθοδο απομόνωσης των ριζών με συνεχή κλάσματα θα χρειαστούμε και κάτω φράγματα στις τιμές των θετικών ριζών ενός πολυωνύμου p(x). Αυτό το κάτω φράγμα στις τιμές των θετικών ριζών του p(x) θα μπορούσε να βρεθεί* ***αντιστρέφοντας*** *το πάνω φράγμα στις τιμές των θετικών ριζών της πολυωνυμικής εξίσωσης* $\chi^n \rho(\frac{1}{x}) = 0$, *όπου n* = deg(p(x)).

Αντί αυτού όμως παρουσιάζουμε έναν διαφορετικό αλγόριθμο, όπου δουλεύουμε με τους συντελεστές του $\chi^n \cdot \rho(-\chi)$, *παίρνοντας τώρα σαν κύριο συντελεστή,* lc = First[cl] — *αντί του* lc = Last[cl] που *χρησιμοποιούσαμε για το πάνω φράγμα. Επιπλέον, προσέξτε πως στο τέλος το όριο είναι* $2^{[-\log 2b]}$.

```
CauchyPositiveRootLowerBound[p_] : =
Module [{b = 0 , cl, lc, lamda, n, cpos, tb} , cl = CoefficientList[p,
Variables[p]] ;
n = Length[cl] ; lc = First[cl] ;
```

```
If [lc < 0 , cl = - cl; lc = -  lc] ; lamda =
Length[Select[cl, # < 0 &]] ;
If [lamda == 0 , Return [b]] ; cpos = Position[cl,    ? (# <
0 &) ] ;
Do [tb =  ⁿ⁻ᶜᵖᵒˢ[[k]]⁻¹√(-lamda cl[[cpos[[k]]]]/lc)  // N // First;lc
If [tb > b, b = tb] , {k, lamda}] ; 2 ^ (-Ceiling [Log [2, b]])]
```

Έτσι βλέπουμε πως το 1 *είναι τόσο ένα κάτω φράγμα στις τιμές των* ***θετικών*** *ριζών του ρ(χ) =* $\chi^3 - 7\chi + 7$

```
p [x_] = x^3 - 7 x + 7;CauchyPositiveRootLowerBound[p[x]]
```

1

όσο και ένα κάτω φράγμα στις τιμές των ***αρνητικών*** *ριζών του ρ(χ) =* $\chi^3 - 7\chi + 7$, και *βρίσκεται αντικαθιστώντας το χ με το -χ, χ< -χ.*

```
CauchyPositiveRootLowerBound[p [-x]]
```

1

6.2.6 Κάτω *φράγμα στην απόσταση μεταξύ δύο τυχαίων ριζών ενός πολυωνύμου*

Στην ενότητα αυτή θα παρουσιάσουμε το θεώρημα του K. Mahler (1964) για τον *υπολογισμό ενός κάτω φράγματος στην* ***ελάχιστη απόσταση Δ των ριζών*** *ενός πολυωνύμου* ρ(χ). *Το αποτέλεσμα αυτό είναι σημαντικότατο γιατί χρησιμοποιείται στην ανάλυση του χρόνου υπολογισμού όχι μόνο της μεθόδου* Sturm *αλλά και κάθε άλλης μεθόδου απομόνωσης των πραγματικών ριζών ενός πολυωνύμου.*

Για την απόδειξη του θεωρήματος του Mahler *χρειαζόμαστε το θεώρημα του* Hadamard *για τις ορίζουσες, το οποίο παρουσιάζουμε χωρίς απόδειξη, τους ορισμούς της διακρίνουσας και του μέτρου ενός πολυωνύμου, και τέλος την ανισότητα του* Landau.

Θεώρημα **(Hadamard,** ***για ορίζουσες)***

Αν τα στοιχεία d_{ij}, ij = 1, 2, ..., n *του πίνακα :*

$$m = \begin{bmatrix} d_{11} & \cdots & d_{1n} \\ \vdots & \ddots & \vdots \\ d_{n1} & \cdots & d_{nn} \end{bmatrix}$$

είναι τυχαίοι μιγαδικοί αριθμοί, τότε για την ορίζουσα d = det(m) *ισχύει η ανισότητα:*

$$|d| \leq \sqrt{\prod_{j=1}^{n}\left(\sum_{i=1}^{n}|\boldsymbol{d_{ij}}|^2\right)}$$

Η ισότητα ισχύει εάν και μόνο εάν

$\sum_{i=1}^{n} \boldsymbol{d_i}\, \boldsymbol{d'_{ik}}$ $1 \le j < k \le n$

Όπου ο $\boldsymbol{d'_{ik}}$ *είναι ο συζυγής μιγαδικός του* $\boldsymbol{d_{ik}}$

Ορισμός (διακρίνουσας):

Η ***διακρίνουσα*** (discriminant), discr(p(x)), ενός *πολυωνύμου*
$p(x) = c_n \prod_{i=1}^{n}(x - a_i)$
ορίζεται από την σχέση

$\text{discr}(p(x)) = c_n^{2n-2} \prod_{i=1}^{n} \prod_{j=i+1}^{n}(a_i - a_j)^2$

η *οποία γενικεύει τον τύπο* b^2 - 4ac, της *διακρίνουσας πολυωνύμου δευτέρου βαθμού. (Προσέξτε πως* c_n *είναι ο συντελεστής του* x^n*.)*

Τονίζουμε πως η *διακρίνουσα μας δίνει ένα μέτρο του πόσο κοντά είναι οι ρίζες του πολυωνύμου p(x).*

Ορισμός (μέτρου ενός πολυωνύμου):
Έστω $p(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0 = c_n \prod_{i=1}^{n}(x - a_i) \in C[x]$, *ένα πολυώνυμο με μιγαδικούς συντελεστές,* $c_n \neq 0$, και *ρίζες* α_i. *Το* ***μέτρο*** *του*p(x), μ(ρ(x)), *ορίζεται από τον τύπο*

$\mu(\rho(x)) = |c_n| \prod_{i=1}^{n} max(1, |a_i|)$

Επιπλέον ισχύει και ***η ανισότητα του*** **Landau (1905)**

$c_n \prod_{i=1}^{n}(x - a_i)$

όπου $| p(x) |_2$ *είναι η Ευκλείδεια νορμ. Την ανισότητα του* Landau θα *χρησιμοποιήσουμε στο ακόλουθο:*

Θεώρημα **(Mahler, 1964):**
Έστω $p(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0$ e Z[x], *ένα πολυώνυμο βαθμού* $n > 2$, μιας *μεταβλητής, ελεύθερο από τετράγωνα, και με ακέραιους συντελεστές. Αν Δ είναι ελάχιστη απόσταση των ριζών του* p(x), *τότε ισχύει η ανισότητα*

$$\Delta \geq \sqrt{3} n^{-\frac{(n+2)}{2}} \left| p(x) \right|_1^{-(n-1)} \quad (\quad \text{M} \quad)$$

όπου $|\mathrm{p}(\mathrm{x})|_1$ *είναι η αθροιστική νορμ του p(x) — δηλαδή το άθροισμα των απολύτων τιμών των συντελεστών.*

Απόδειξη:

Έστω ότι $\alpha_1, \alpha_2, \ldots, \alpha_n$ *είναι οι ρίζες του* p(x). *Τις αριθμούμε έτσι ώστε*

$$|a_1| \geq |a_2| \geq \ldots \geq |a_N| > 1 \geq |a_{N+1}| \geq \ldots \geq |a_n|$$

και *ορίζουμε την ορίζουσα*

$$\mathrm{vdm}(\mathrm{p}(\mathrm{x})) = \coprod_{i=1}^{n} \prod_{j=i+1}^{n} (a_i - a_j)$$

με την συνθήκη vdm(p(x)) = 1, *για την* ***μη επιτρεπτή*** *περίπτωση* n = 2. *Από την ενότητα* 4.2 του *πρώτου τόμου, θυμόμαστε πως vdm(p(x)) είναι η ορίζουσα του* Vandermonde

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \ldots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \ldots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \ldots & \alpha_n^{n-1} \end{pmatrix}.$$

Επιπλέον ορίζουμε την $\mathrm{vdm}^*(\mathrm{p}(\mathrm{x})) = (a_1 a_2 \cdot\cdot\, a_N)^{-(n-1)} \mathrm{vdm}(\mathrm{p}(\mathrm{x}))$ *που είναι η ορίζουσα του πίνακα*

$$\begin{pmatrix} \alpha_1^{-(n-1)} & \alpha_1^{-(n-2)} & \alpha_1^{-(n-3)} & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_N^{-(n-1)} & \alpha_N^{-(n-2)} & \alpha_N^{-(n-3)} & \cdots & 1 \\ 1 & \alpha_{N+1} & \alpha_{N+1}^2 & \cdots & \alpha_{N+1}^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^{n-1} \end{pmatrix}$$

Δηλαδή, ο *τελευταίος πίνακας προκύπτει αφού πολλαπλασιάσουμε την* /-στη *σειρά του* Vandermonde επί $a^{-(n-1)}$, $1 \leq \iota \leq$ N. Επειδή η απόλυτος τιμή κάθε στοιχείου στον τελευταίο πίνακα είναι ≤ 1 *έπεται από την ανισότητα* (H) *του* Hadamard *πως*

$$|\mathrm{vdm}^*(\mathrm{p}(\mathrm{x}))| \leq \sqrt{n^n} = \mathrm{n}^{\mathrm{n}/2}.$$

Εδώ η ισότητα ισχύει αν

$|a_1| = |a_2| = \ldots = |a_n|$

και

$$\sum_{k=0}^{n-1} \overline{a_1^K} a_j^k = 0 \quad \text{για } 1 \le i < j \le n$$

Για $i = 1$ *(και απαλείφοντας τον παρανομαστή του αντιστρόφου* α^{-1}) το *παραπάνω άθροισμα γίνεται*

$$\sum_{k=0}^{n-1} \overline{a_1^K} a_j^k = \sum_{k=0}^{n-1} (a_1^{-1} a_j)^k = \frac{(\frac{a_j}{a_1})^n - 1}{(\frac{a_j}{a_1}) - 1} = 0$$

Πολλαπλασιάζοντας *την τελευταία έκφραση με* $\frac{a_j}{a_1}$, *προκύπτει* $(\frac{a_j}{a_1})^n = 1$,

απ' όπου *βλέπουμε πως τα* $\frac{a_j}{a_1}\ 1 \le j \le n$ είναι Οι n *διαφορετικές ρίζες της εξίσωσης* $x'' - 1 = 0$.

Επομένως,

$$x'' - 1 = \prod_{j=1}^{n} x - \frac{a_j}{a_1}$$

$$a_1^n x^n - a_1^n = \prod_{j=1}^{n} (x - a_j)$$

Άρα, $|\text{vdm}^*(p(x))| = \le \sqrt{n^n} = n^{n/2}$.μόνο στην *περίπτωση που* $p(x) = c_n x'' + c_0$, *και* $|c_n| = |c_0| \ne 0$.

Έστω τώρα r,s έτσι ώστε $1 \le r < s \le n$. Θα *βρούμε ένα πάνω φράγμα στην έκφραση*

$\cup \frac{\text{vdm}^{\perp}[p\ x}{r\ s} \cup$ *και* με την *βοήθειά* του θα *αποδείξουμε το θεώρημα.*

Στον πίνακα Vandermonde *αφαιρούμε την* s-στή σειρά *από την r-στή σειρά και το αποτέλεσμα είναι η* ***νέα*** *σειρά* r, *που αποτελείται από τα στοιχεία*

$$0,\ r - s,\ r^2 - s^2,\ \ldots,\ r^{n-1} - s^{n-1}$$

τα *οποία είναι όλα τους πολλαπλάσια του* a_r - a_s. *Διαιρούμε την σειρά r με* a_r - a_s *και έστω* q_0, q_1, ..., q_{n-1} τα *νέα στοιχεία της σειράς* αυτής, *όπου* $q_0 = 0$, και

$q_i = \frac{\square_r^i \square \square_s^i}{\square_r \square \square_s} = \sum_{k=0}^{i-1} a_r^{i-1-k} a_s^k$ *για* $1 \le i \le n-1$

Η ορίζουσα του ***νέου πίνακα*** *— που διαφέρει από τον ορχικό ως προς την σειρά r — είναι προφανώς* $\frac{\mathrm{vdm}\ p\ x}{\square_r \square \square_s}$. Αν *στην συνέχεια, στον νέο αυτό πάνα^ διαιρέσουμε — όπως και πριν —* την 1η, 2η, ..., *N-στη σειρά με τους όρους* $a_1^{n-1}, a_2^{n-1}, ..., a_N^{n-1}$ *αντίστοιχα προκύπτει Πίνακας η ορίζουσα του οποίου είναι* $\frac{\mathrm{vdm}\square\ p\ x}{\square_r \square \square_s}$! *Στον νέο αυτό πίνακα η r-στη σειρά από τα στοιχεία*

$q_0 a_r^{-(n-1)}, q_1 a_r^{-(n-1)}, ..., q_{n-1} a_r^{-(n-1)}$ *αν* $r > N$ *ή από τα στοιχεία* $q_0, q_1, ..., q_{n-1}$

Επειδή δε, $r > N \cup \square_r \cup \ge \cup \square_s \cup$ και $\cup \square_r \cup = \begin{matrix} \square 1 \square\square\square r \square N \\ \square 1 \square\square\square r \square N \end{matrix}$ *έπεται πως οι απόλυτες τιμές των διαδοχικών στοιχείων της r-στής σειράς του νέου πίνακα δεν υπερβαίνουν τις τιμές* Ο, *1,* 2, *n* - 2, *n* - *1, αντίστοιχα. Αυτό "αποδεικνύεται" και με το Mathematical με την χρήση της συνάρτησης* Assuming[] — όπου αντί για α_1. *και* α_8 *χρησιμοποιούμε a και b :*

```
(*  r ≤ N  *)
Clear[n, i]; n = 10; answer = {};
Do[Assuming[a ∈ Reals && b ∈ Reals && Abs[a] > Abs[b] &&
   Abs[a] > 1 && i ∈ Integers && i > 1,
  AppendTo[answer, Refine[a^-(n-1) ∑(k=0, i-1) a^(i-1-k) b^k ≤ i]]], {i, 2, n}]; answer
```

```
{True, True, True, True, True, True, True, True, True}
```

```
(*  r > N  *)
Clear[n, i]; n = 10; answer = {};
Do[Assuming[a ∈ Reals && b ∈ Reals && Abs[a] > Abs[b] &&
   Abs[a] < 1 && i ∈ Integers && i > 1,
  AppendTo[answer, Refine[∑(k=0, i-1) a^(i-1-k) b^k ≤ i]]], {i, 2, n}]; answer
```

```
{ True, True, True, True, True, True, True, True, True}
```

Όσον *αφορά τις απόλυτες τιμές των διαδοχικών στοιχείων των υπόλοιπων n - 1 σειρών του νέου πίνακα — εκτός r-στής σειράς — αυτές δεν υπερβαίνουν την μονάδα. Επομένως, από την ανισότητα* (H) του Hadamard *έχουμε*

$$\frac{\mathrm{vdm}(p(x))}{\Delta_{r,s}}^2 \le (n-1)^{n-1} \sum_{i=0}^{n-1} i^2 < n^{n-1} \sum_{i=0}^{n-1} i^2.$$

όπου το άθροισμα $\sum_{i=0}^{n-1} i^2$ *είναι μόνο για την r-στή σειρά και ο πρώτος όρος*

$$(n-1)^{n-1} = \sum_{j=0}^{n-2} \sum_{i=0}^{n-2} 1$$

είναι για τις υπόλοιπες. Επειδή όμως το άθροισμα

$$\sum_{i=0}^{n-1} i^2 = \frac{n(n-1)(2n-1)}{6} \quad \textit{προκύπτει}$$

$$\frac{\mathrm{vdm}(p(x))}{\Delta_{r,s}}^2 < \frac{n^{n-2}}{3}$$

Λύνουμε την παραπάνω ανισότητα ως προς $\Delta_{r,s}^2$ και *έχουμε*

$$\Delta_{r,s}^2 > 3\, n^{-n-2}\, \mathrm{vdm}(p(x))^2,$$

ή

$$\Delta_{r,s}^2 > 3\, n^{-(n+2)} (a_1 a_2 \cdots a_N)^{-2(n-1)} |\, \mathrm{vdm}(p(x))\,|^2.$$

ή

$$|\, a_r - a_s\,|^2 > 3\, n^{-(n+2)} (c_n\, \alpha_1\, \alpha_2 \cdots a_N)^{-2(n-1)}\, c_n^{2n-2} \,|\, vdm(p(x))\,|^2.$$

Από *τους τύπους (μ) του μέτρου και* (D) *της διακρίνουσας βλέπουμε πως η παραπάνω ανισότητα γράφεται και σαν*

$$|\, \alpha_\gamma - \alpha,\,|^2 > 3\, n^{-(n+2)}\, (\mu(\, p(x)))^{-2(n-1)}\, c_n^{2n-2} \mathrm{discr}(\, p(x)).$$

Επειδή οι συντελεστές του *p(x) είναι ακέραιοι,* |discr(p(x))| > 1. Επιπλέον *επιλέγοντας τους δείκτες r, s έτσι ώστε Δ =* $|\, \alpha,, - \alpha\alpha \,|$ *και χρησιμοποιούντες την ανισότητα του* Landau (L) *αποδεικνύεται η ζητούμενη ανισότητα* (M).//

Παράδειγμα:

Για το πολυώνυμο p(x) = $x^3 - 7x + 7$ *βλέπουμε πως*

$$\Delta \ge \sqrt{3^3 - 15^2} = \sqrt{3^3 15^4} \approx 0.000493827$$

Αυτό σημαίνει πως οι ρίζες του p(x) δεν μπορούν να έχουν απόσταση

μικρότερη από 0.000493827.

6.3 Το *θεώρημα τον* Budan *και η μέθοδος των* Vincent-Akritas-Strzebonski *για την απομόνωση πραγματικών ριζών με συνεχή κλάσματα*

Στην προηγούμενη ενότητα εξετάσαμε το θεώρημα του Fourier, *από το οποίο απορρέουν το θεώρημα του* Sturm *και η μέθοδος του* Sturm *για την απομόνωση πραγματικών ριζών με διχοτόμηση. Η μέθοδος αυτή ήταν* η *πρώτη που αναπτύχθηκε και αποτέλεσε σταθμό στην ιστορία των μαθηματικών. Χρησιμοποιήθηκε πολύ μέχρι το* 1976.

Όμως το 1975/76 — με *τροποποίηση του θεωρήματος του* Vincent που *μόλις είχε ανακαλύψει ο γράφων — αναπτύχθηκε η μέθοδος των* Collins-Akritas για την *απομόνωση πραγματικών ριζών με διχοτόμηση. Η μέθοδος αυτή —* που *είναι πολύ πιο γρήγορη από την μέθοδο του* Sturm *— εφαρμόσθηκε στο σύστημα υπολογιστικής άλγεβρας* maple, και *έκτοτε χρησιμοποιείται ευρέως. Με το πέρασμα του χρόνου αποδείχθηκε πως* ***η μέθοδος των* Collins-Akritas *είναι η πιο γρήγορη μέθοδος για την απομόνωση πραγματικών ριζών με διχοτόμηση.***

Στην ενότητα αυτή θα εξετάσουμε το θεώρημα του Budan *από το 0ποίο απορρέουν το θεώρημα του* Vincent *και η μέθοδος των* Vincent-Akritas-Strzebon-ski για την

απομόνωση πραγματικών ριζών με συνεχή κλάσματα. Ο Vincent (1836) *ήταν ο πρώτος που ανέπτυξε την μέθοδο των συνεχών κλασμάτων αλλά ο χρόνος υπολογισμού της μεθόδου του ήταν εκθετικός. Στην συνέχεια ο γράφων βελτίωσε την μέθοδο αυτή* (1978) *και την έκανε όχι μόνο πολυωνυμική, αλλά και την* ***πιο γρήγορη μέθοδο απομόνωσης πραγματικών ριζών στον κόσμο.*** *Με τον* Strzebonski (1994) *βελτιώθηκε η μέθοδος ως προς ένα ακόμα σημείο.*

6.3.1 Το *θεώρημα τον* Budan

Όπως έχουμε αναφέρει το θεώρημα του Budan *εμφανίστηκε πριν από το θεώρημα του* Fourier, *αλλά παρά όλα αυτά είχε ξεχαστεί. Στην βιβλιογραφία αναφερόταν μεν το όνομα* Budan *αλλά η διατύπωση του θεωρήματος ήταν εκείνη του* Fourier. *Ο γράφων βρήκε την διατύπωση του θεωρήματος του* Budan *στο άρθρο του* Vincent *του* 1836 *κα την παρουσιάζει — λίγο διασκευασμένα — σε αναλογία με το θεώρημα του* Fourier:

Θεώρημα τον **Budan (1807):** *(Υπολογισμός ενός πάνω φράγματος στον αριθμό των πραγματικών ριζών που έχε μία εξίσωση σε ένα ανοιχτό διάστημα.)*

Στην πολυωνυμική εξίσωση $p(x) = 0$, *βαθμού* $n > 0$, *κάνουμε τις δύο αντικαταστάσεις,*

$x \leftarrow x + l$ *και* $x \leftarrow x + r$, *όπου* l *και* r *είναι πραγματικοί αριθμοί έτσι ώστε* $l < r$. Αν v_l *και* v_r *είναι οι μεταβολές πρόσημου στις ακολουθίες των συντελεστών των πολυωνύμων* $p(x + l)$ *και* $p(x + r)$, *αντίστοιχα, τότε ισχύουν τα ακόλουθα :*

i. *Το πολυώνυμο* $p(x + l)$ *δεν μπορεί να έχει λιγότερες μεταβολές πρόσημου από το πολυώνυμο* $p(x + r)$. Δηλαδή, $v_l \geq v_r$.

ii. *Ο αριθμός* ρ *των πραγματικών ριζών της εξίσωσης* $p(x) = 0$ *που βρίσκονται* στο *διάστημα* (l, r) *ποτέ δεν μπορεί να είναι μεγαλύτερος από τον αριθμό των μεταβολών πρόσημου που χάνονται κατά την μετάβασή μας από το πολυώνυμο* $p(x + l)$ *στο πολυώνυμο* $p(x + r)$. Δηλαδή, $\rho \leq v_l - v_r$.

iii. *Όταν ο αριθμός* ρ *των πραγματικών ριζών της εξίσωσης* $p(x) = 0$ *που βρίσκονται στο διάστημα* (f, r) *είναι γνήσια μικρότερος από τον αριθμό των μεταβολών πρόσημου που χάνονται κατά την μετάβασή μας από το πολυώνυμο*

$p(x + l)$ *στο πολυώνυμο* $p(x + f)$, *τότε η διαφορά είναι άρτιος αριθμός.*

Δηλαδή,

$\rho = v_l - v_r - 2\lambda$, *όπου* $\lambda \in Z_{>0}$.

Ισοδυναμία:

Το θεώρημα του Budan *είναι* ***ισοδύναμο*** *με το θεώρημα του* Fourier. *Αυτό φαίνεται από το γεγονός ότι για το πολυώνυμο* p(x), *βαθμού* $n > 0$, *στην ακολουθία* Fseq(t) (που *προκύπτει από την αντικατάσταση* $x \leftarrow t$) *οι* $n + 1$ *αριθμοί* ***έχουν το ίδιο πρόσημο,*** *και είναι ανάλογο με τους αντίστοιχους* συντ∂λ∂στές του πολυωνύμου

$$p(x + t) = \sum_{i=0}^{n} \frac{p^{i} t}{i!} x^{i},$$

όπως προκύπτουν από τον μετασχηματισμό Taylor — ή *την αντικατάσταση* $x \leftarrow x + t$.

Επομένως, το θεώρημα του Budan μας *δίνει επίσης ένα* ***πάνω φράγμα*** *στον αριθμό των πραγματικών ριζών που έχει η εξίσωση* $p(x) = 0$ *μέσα στο διάστημα* (f, r). *Προσέξτε όμως πως αντί για ακολουθίες παραγώγων χρησιμοποιούνται οι αντικαταστάσεις* $x \leftarrow x + 1$ *και* $x \leftarrow x + r$. *Οι αντικαταστάσεις αυτές ονομάζονται* **Mobius** ή ***γραμμικές κλασματικές αντικαταστάσεις*** *και στην επόμενη υποενότητα εξετάζουμε την επίδρασή τους στις ρίζες των πολυωνύμων.*

Προσοχή:

Με τις αντικαταστάσεις $x \leftarrow x + \{$ *και* $x \leftarrow x + \rho$. *μπορούμε να βρούμε τον ακριβή αριθμό των ριζών στο διάστημα* (f, r) *μόνο στις εξής* ***δύο*** *περιπτώσεις:* **(α)** αν δεν *χάνεται καμία μεταβολή πρόσημου τότε δεν υπάρχει καμία ρίζα στο (*l, r), και ***(β)*** *αν χάνεται μία μεταβολή πρόσημου τότε υπάρχει μία πραγματική ρίζα στο (*l, r). ***Τα αντίστροφα των (α) και (β) δεν ισχύουν!***

Παράδειγμα: *(υπολογισμός ενός πάνω φράγματος στον αριθμό των πραγματικών ριζών μέσα σε ένα διάστημα με την βοήθεια του θεωρήματος του* Budan)
Έστω πάλι το πολυώνυμο $p(x) = x^3 - 7x + 7$. *Για να βρούμε, με το θεώρημα του* Budan, *ένα πάνω φράγμα στον αριθμό των πραγματικών ριζών που έχει το πολυώνυμο αυτό στο διάστημα* (0, 2) *υπολογίζουμε την διαφορά των μεταβολών πρόσημου στα πολυώνυμα* $p(x + 0)$ και $p(x + 2)$:

```
p [x_] = x^3 - 7 x + 7;
variations [p [x + 0]] - variations [p [x + 2]]
```

2

Δηλαδή χάνονται δύο μεταβολές πρόσημου, και αυτό σημαίνει πως στο διάστημα (0, 2) *το πολυώνυμο είτε έχει δύο πραγματικές ρίζες ή καμία, κάτι που πρέπει να διερευνηθεί. Υπενθυμίζουμε πως οι συντελεστές του πολυωνύμου* που *προκύπτει από την αντικατάσταση*

$x \leftarrow x + \alpha$ *υπολογίζονται με την μέθοδο των* Rufini-Horner που *εξετάσαμε στην ενότητα* 4.1 του *πρώτου τόμου.*

6.3.2 *Αντικαταστάσεις* Mobius *και η επίδρασή τους στις ρίζες πολνωννμικών εξισώσεων*

Όπως θα δούμε στην επόμενη υποενότητα 6.3.3, *σημαντικότατο ρόλο στο θεώρημα του* Vincent *παίζουν οι αντικαταστάσεις της μορφής* $x \leftarrow \alpha + \frac{1}{x}$. *Οι αντικαταστάσεις αυτές ανήκουν στην κατηγορία των* ***γραμμικών κλασματικών αντικαταστάσεων*** *ή* ***αντικαταστάσεων*** **Mobius,** *που*
ονομάστηκαν έτσι προς τιμήν του A.F. Mobius (1790-1868) ο *οποίος πρώτος τις μελέτησε στην προβολική γεωμετρία.*

Ονομάζουμε ***αντικατάσταση*** **Mobius** την *έκφραση*

$$x \leftarrow \mathrm{M}(x) = \frac{a\,x \square b}{c\,x \square d},$$

όπου a, b, c, d είναι μιγαδικοί αριθμοί έτσι ώστε η ορίζουσά τους είναι διάφορη του *μηδενός — δηλαδή έχουμε* $ad - bc \neq 0$. Την *αντικατάσταση αυτή συμβολίζουμε ως* $x \leftarrow \mathrm{M}(x)$, $\det(\mathrm{M}) \neq 0$.

Για τυχαίο $x \in \mathbb{C}$, η *αντικατάσταση* $x \leftarrow \mathrm{M}(x)$ *ορίζεται από την έκφραση του ορισμού, με την προϋπόθεση ότι* $cx + d \neq 0$. Αλλιώς, *ορίζουμε* $\mathrm{M}(\frac{\square d}{c}) = \infty$. *Αν* $c = 0$, *τότε πρέπει να έχουμε* $ad \neq 0$— *επειδή* η *ορίζουσα πρέπει να είναι διάφορη του μηδενός — και η αντικατάσταση ορίζεται από την έκφραση*

$$x \leftarrow \mathrm{M}(x) = \frac{a}{d} x \square \frac{b}{d}.$$

Προσέξτε πως στην περίπτωση $c = 0$ *ισχύει* $\mathrm{M}(\infty) = \infty$, *ενώ αν* $c \neq 0$,, *τότε* $\mathrm{M}(\infty) = \frac{a}{c}$.

Είναι προφανές ότι σε κάθε αντικατάσταση Mobius ***αντιστοιχεί*** *ο τετράγωνος πίνακας των συντελεστών του, και αυτό πλήρως ορίζει την αντικατάσταση. Ορίζουμε λοιπόν*

$$\mathbf{M} = \{M(x) : x \leftarrow M(x),\ M = \begin{matrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{matrix},\ \det(M)\neq 0,\ x \in X\},$$

το σύνολο όλων των *αντικαταστάσεων* Mobius και X' = X⇐{∞},, το *συμπαγές μιγαδικό επίπεδο.*

Προσοχή.

Στην συνεχεία ο συμβολισμός $\begin{matrix} a & b \\ c & d \end{matrix}(x)$ ***υπονοεί*** *την σχέση:*

$$\begin{matrix} a & b \\ c & d \end{matrix}(x) = \frac{a\,x + b}{c\,x + d}$$

Θεώρημα'

Το σύνολο *M. των αντικαταστάσεων* Mobius *αποτελεί ομάδα ισομορφική με την ομάδα τετραγώνων πινάκων τάξης* 2.

Απόδειξη'

Στο σύνολο Μεισάγουμε *κατ αρχάς μια σχέση* ***ισότητας*** *ως* εξής: Δύο *αντικαταστάσεις* A(*x*), B(*x*) ∈ **M** *ταυτίζονται, δηλαδή* ∀*x* ∈ X' ισχύει A(x) = B(x), εάν και μόνο εάν ∃λ ∈ X, λ ≠ 0, *έτσι ώστε A = λB* — όπου η *τελευταία είναι ισότητα πινάκων·*

Ο λόγος που *ορίζουμε την ισότητα έτσι είναι διότι αν σε ένα πολυώνυμο p(x) κάνουμε την αντικατάσταση* $x \leftarrow \begin{matrix} a & b \\ c & d \end{matrix}(x)$ ή την αντικατάσταση $x \leftarrow \lambda \begin{matrix} a & b \\ c & d \end{matrix}(x)$ το *αποτέλεσμα είναι το ίδιο. Αυτό φαίνεται από το ακόλουθο παράδειγμα όπου κάνουμε τις αντικαταστάσεις* και $x \leftarrow \begin{matrix} 0 & 1 \\ 1 & 1 \end{matrix}(x)$ και $x \leftarrow 3\begin{matrix} 0 & 1 \\ 1 & 1 \end{matrix}(x)$ στο *πολυώνυμο* $p(x) = x^3 - 7x + 7$. Προσέξτε πως για να *αποφύγουμε ρητές εκφράσεις πολλαπλασιάζουμε επί* $(x + 1)^3$.

```
Clear p:=x^3-7*x+7; p(1/(1-x)):k:=1:Simplify
```

$$1 - 7x - 14x^2 - 7x^3$$

```
p(-3/(3*x)):k:=1:Simplify
```

$$1 - 7x - 14x^2 - 7x^3$$

Κατόπιν ορίζουμε το ***γινόμενο*** *δύο αντικαταστάσεων* A(*x*), B(*x*) ∈ **M** *ως το γινόμενο των πινάκων τους* AB(x). *Προσέξτε όμως τον* ***τρόπο εφαρμογής*** *του γινομένου*

αντικαταστάσεων: ***πρώτα*** *εφαρμόζεται* η *αντικατάσταση* A(x) και ***ύστερα*** *η αντικατάσταση* B(x). Το *γινόμενο αυτό είναι* επίσης *μια αντικατάσταση διότι*

$$x \leftarrow \mathrm{AB}(x) = \mathrm{A}(\mathrm{B}(x)) = \frac{a_{11}\,\mathrm{B}\,x + a_{12}}{a_{21}\,\mathrm{B}\,x + a_{22}} =$$

$$= \frac{a_{11}\frac{b_{11}x + b_{12}}{b_{21}x + b_{22}} + a_{12}}{a_{21}\frac{b_{11}x + b_{12}}{b_{21}x + b_{22}} + a_{22}} = \frac{a_{11}b_{11} + a_{12}b_{21}\,x + a_{11}b_{12} + a_{12}b_{22}}{a_{21}b_{11} + a_{22}b_{21}\,x + a_{21}b_{12} + a_{22}b_{22}} =$$

$$= \begin{matrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{matrix} (x) = (\mathrm{AB})(x)$$

$$\mathrm{I} = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}.$$

Σαν το *μοναδιαίο στοιχείο του συνόλου* Λ4 *ορίζουμε την ταυτοτική αντικατάσταση* $x \leftarrow x$, που αντιστοιχεί στον πίνακα

Από τον *τρόπο που ορίσαμε την ισότητα βλέπουμε πως* $\lambda\mathrm{I}$, $\lambda \in$ Χκαι $\lambda \neq 0$, συμπίπτει με την ταυτοτική *αντικατάσταση*. Για *τυχαία αντικατάσταση* $x \leftarrow \mathrm{M}(x) \in \mathbf{M}$,, όπου $\mathrm{M} = \begin{matrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{matrix}$ η *αντίστροφη αντικατάσταση είναι* $x \leftarrow M^{-1}(x)$, όπου το M^{-1} είναι **είτε**

$$M^{-1} = \frac{1}{\det M} \begin{matrix} m_{22} & -m_{12} \\ -m_{21} & m_{11} \end{matrix}$$

δηλαδή ο αντίστροφος πίνακας του M, **είτε,** λόγω *ορισμού της ισότητας,*

$$M^{-1} = \begin{matrix} m_{22} & -m_{12} \\ -m_{21} & m_{11} \end{matrix}$$

Προφανώς $\det(M^{-1}) \neq 0$./

Η *ομάδα δεν είναι Αβελιανή διότι γενικά* $\mathrm{A}(x)\,\mathrm{B}(x) \neq \mathrm{B}(x)\,\mathrm{A}(x)$.

Ορίσμός:

Οι *ακόλουθοι τρεις αντικαταστάσεις* Mobius *ονομάζονται* ***γεννήτριες αντικαταστάσεις*** (generating substitutions) της *ομάδας* **Λ1:**

i. ***Αντίστροφη*** (inversion): $x \leftarrow \frac{1}{x} = \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} (x)$

ii. **Μετάθεση** (translation): $x \leftarrow x + a = \begin{matrix} 1 & a \\ 0 & 1 \end{matrix} (x)$

iii. **Τάνυσμα** (stretching): $x \leftarrow a\,x = \begin{matrix} a & 0 \\ 0 & 1 \end{matrix} (x)$

Όταν το *« e* C το τάνυσμα λέγεται ***στροφή*** (rotation).

Το θεώρημα που ακολουθεί τονίζει την σημασία των γεννητριών αντικαταστάσεων.

Θεώρημα (γεννητριών αντικαταστάσεων):

Κάθε αντικατάσταση M(x) ∈ **M** *προκύπτει από κατάλληλο πολλαπλασιασμό των γεννητριών αντικαταστάσεων.* (Μην ξεχνάτε πως σε ένα ***γινόμενο αντικαταστάσεων, αυτές εφαρμόζονται μία-μία από τα αριστερά προς τα δεξιά.)*** *Επομένως κάθε αντικατάσταση προκύπτει από μία ακολουθία γεννητριών αντικαταστάσεων.*

Απόδειξη:

Έστω

$$\mathrm{M} = \begin{matrix} a & b \\ c & d \end{matrix}.$$

Για να αποδείξουμε το θεώρημα διακρίνουμε τις εξής δύο περιπτώσεις:

c = **0:** *Στην περίπτωση αυτή εύκολα μπορούμε να δούμε πως* M = $M_1 M_2$, *όπου*

$$M_1 = \begin{matrix} 1 & \frac{b}{d} \\ 0 & 1 \end{matrix} \quad \text{και} \quad M_2 = \begin{matrix} \frac{a}{d} & 0 \\ 0 & 1 \end{matrix}.$$

Πράγματι, το γινόμενό τους είναι

$$M_1 M_2 = \begin{matrix} \frac{a}{d} & \frac{b}{d} \\ 0 & 1 \end{matrix} = \frac{1}{d} \begin{matrix} a & b \\ 0 & d \end{matrix} = \begin{matrix} a & b \\ 0 & d \end{matrix},$$

που *σημαίνει πως α* 0, η *αντικατάσταση* M(x) *είναι ισοδύναμη προς μία* ***μετάθεση,*** M_1(x), ***ακολουθούμενη*** *από ένα* ***τάνυσμα,*** M_2(x). $c \neq 0$**:**

Στην *περίπτωση αυτή μπορούμε επίσης εύκολα να δούμε η αντικατάσταση* M(x) *είναι ισοδύναμη προς μία* ***μετάθεση,*** M_1(x), ***ακολουθούμενη*** *από μία* ***αντίστροφη,*** M_2(x), ***ακολουθούμενη*** *από μία ακόμα* ***μετάθεση,*** M_3(x), ***ακολουθούμενη*** *από ένα* ***τάνυσμα,*** M_4(x), *όπου*

$$M_1 = \begin{pmatrix} 1 & \frac{a}{c} \\ 0 & 1 \end{pmatrix}, \quad M_2 = \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix},$$

$$M_3 = \begin{matrix} 1 & \frac{c\,d}{b\,c - a\,d} \\ 0 & 1 \end{matrix} \quad \text{και} \quad M_4 = \begin{matrix} \frac{c^2}{b\,c - a\,d} & 0 \\ 0 & 1 \end{matrix}$$

Πράγματι, το γινόμενό τους είναι

$$M_1 M_2 M_3 M_4 = \begin{matrix} \frac{a}{c} & 1 \\ 1 & 0 \end{matrix} M_3 M_4 = \begin{matrix} \frac{a}{c} & \frac{b\,c}{b\,c - a\,d} \\ 1 & \frac{c\,d}{b\,c - a\,d} \end{matrix} M_4$$

$$= \begin{vmatrix} \frac{a\,c}{b\,c\,a\,d} & \frac{b\,c}{b\,c\,a\,d} \\ \frac{c^2}{b\,c\,a\,d} & \frac{c\,d}{b\,c\,a\,d} \end{vmatrix} = \frac{c}{b\,c\,a\,d} \begin{matrix} a & b \\ c & d \end{matrix} = \begin{matrix} a & b \\ c & d \end{matrix}$$

Παράδειγμα:

Ας θεωρήσουμε την αντικατάσταση $x \leftarrow \frac{1}{1+x}$ *στην οποία αντιστοιχεί ο πίνακας* $\begin{matrix} 0 & 1 \\ 1 & 1 \end{matrix}$ *. Εφαρμόζοντας το παραπάνω θεώρημα έχουμε την ακόλουθη ανάλυση σε γινόμενο γεννητριών αντικαταστάσεων:*

$M_1(x) = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} + x$ *ταυτοτική αντικατάσταση*

$M_2(x) = \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} + x$ *, αντιστροφή,*

$M_3(x) = \begin{matrix} 1 & 1 \\ 0 & 1 \end{matrix} x$ *, μοναδιαία μετάθεση, και*

$M_4(x) = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} + x$ *ταυτοτική αντικατάσταση*

Συνεπώς, αντικατάσταση $x \leftarrow \frac{1}{1+x}$ *είναι ισοδύναμη με μία αντιστροφή,* $M_2(x)$*, ακολουθούμενη από μία μετάθεση κατά μονάδα,* $M_3(x)$*.*

Από τα παραπάνω γίνεται φανερό πως οι **k** *αντικαταστάσεις*

$$x \leftarrow 1 + x,\ x \leftarrow 1 + x,\ \ldots,\ x \leftarrow 1 + x \text{ — } 1 + \chi$$

ακολουθούμενες από την αντικατάσταση $x \leftarrow \frac{1}{1+x}$ *— είναι* ***ισοδύναμες*** *με την αντικατάσταση*

$x \leftarrow \boldsymbol{k} + \frac{1}{x}$ *ακολουθούμενη από την* $x \leftarrow 1 + x$*.*

Ας δούμε τώρα την επίδραση των αντικαταστάσεων Mobius, *ή ισοδύναμα των γεννητριών αντικαταστάσεων, στις ρίζες μιας πολυωνυμικής εξίσωσης* $p(x) = 0$ με ***ακέραιους*** *συντελεστές,* ***βαθμού n*** *και με* ***μία*** *μεταβλητή. Έστω λοιπόν ότι*

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + \equiv + c_0 = c_n (x - \alpha_1)(x - \alpha_2) \equiv (x - \alpha_n) = 0.$$

Αν στο $p(x)$ *κάνουμε την αντικατάσταση* $x \leftarrow \frac{1}{x}$ *,* ***(αντιστροφή)*** *τότε προκύπτει το πολυώνυμο*

$p(\frac{1}{x}) = p_i(x) = \frac{c_n}{x^n} + \frac{c_{n-1}}{x^{n-1}} + \equiv + \frac{c_1}{x} + c_0,$

$= c_n \frac{1}{x^n} (1 - x\alpha_1)(1 - x\alpha_2) \equiv (1 - x\alpha_n)$

$= c_n (\alpha_1 \alpha_2 \equiv \alpha_n) \frac{(-1)^n}{x^n} (x - \frac{1}{\alpha_1})(x - \frac{1}{\alpha_2}) \equiv (x - \frac{1}{\alpha_n})$

$= c_0 \frac{(-1)^n}{x^n} (x - \frac{1}{\alpha_1})(x - \frac{1}{\alpha_2}) \equiv (x - \frac{1}{\alpha_n}) = 0,$

Επειδή $c_0 = c_n(\alpha_1 \alpha_2 \equiv \alpha_n)$ Πολλαπλασιάζοντας την παραπάνω εξίσωση επί $(-1)^n x^n$ έχουμε

$(-1)^n x^n p(\frac{1}{x}) =$

$c_0 x^n + c_1 x^{n-1} + \equiv + c_n = c_0 (x - \frac{1}{\alpha_1})(x - \frac{1}{\alpha_2}) \equiv (x - \frac{1}{\alpha_n}) = 0.$

Δηλαδή βλέπουμε πως ύστερα από την αντικατάσταση $x \leftarrow \frac{1}{x}$ *,* ***οι συντελεστές και οι ρίζες του πολυωνύμου αντιστρέφονται!***

Όταν στο p(x) κάνουμε την αντικατάσταση $x \leftarrow k + x$, $k \in \Theta$, ***(μετάθεση)*** *τότε προκύπτει το πολυώνυμο*

$p(k + x) = p_t(x) = c_n(k + x - \alpha_1)(k + x - \alpha_2) \equiv (k + x - \alpha_n)$

$= c_n(x - (\alpha_1 - k))(x - (\alpha_2 - k)) \equiv (x - (\alpha_n - k)) = 0.$

Δηλαδή, *ύστερα από μία μετάθεση το* ***πραγματικό μέρος των ριζών*** *τον νέον πολυωνύμου p(k + x)* ***θα μικρύνει ή θα μεγαλώσει,*** *ανάλογα με το αν το k είναι θετικό ή αρνητικό, αντίστοιχα.*

Προσέξτε πως οι συντελεστές τον νέον πολυωνύμου είναι *ακέραιοι μόνο στην περίπτωση που*

$k \in Z$. Στην *περίπτωση που* $k \in \Theta$, $k = \frac{a}{b} \neq 1$, για να κάνουμε τους συντελεστές *ακεραίους πρέπει να πολλαπλασιάσουμε το νέο πολυώνυμο που προκύπτει επί* b^n.

```
p(x-3*x+7; 2)(p(x-3/2))|Expand
```

$-1 - 2x - 36x^2 - 8x^3$

Τέλος, όταν στο p(x) κάνουμε την *αντικατάσταση* $x \leftarrow kx$, $k \in \Theta$ και $k \neq 0$, ***(τάνυσμα)*** *τότε προκύπτει το πολυώνυμο*

$$p(k\,x) = p_s(x) = c_n\,(k\,x)^n + c_{n-1}\,(k\,x)^{n-1} + \equiv + c_0$$

$$= c_n(k\,x - \alpha_1)(k\,x - \alpha_2) \equiv (k\,x - \alpha_n)$$

$$= c_n\,k^n\,(x - \frac{\alpha_1}{k})(x - \frac{\alpha_2}{k}) \equiv (x - \frac{\alpha_n}{k}) = 0.$$

Βλέπουμε λοιπόν πως *ύστερα από ένα τάνυσμα οι ρίζες τον νέον πολυωνύμου p(k x)* ***θα μικρύνουν ή θα μεγαλώσουν*** *ανάλογα με το αν το k είναι > 1 ή < 1, αντίστοιχα.*

Και σε αυτήν την περίπτωση οι συντελεστές τον νέον πολυωνύμου είναι ακέραιοι μόνο στην περίπτωση που k ∈ Z. Στην περίπτωση που k ∈ Θ, k = $\frac{a}{b}$ *□ 1,, για να κάνουμε τους συντελεστές ακεραίους πρέπει να πολλαπλασιάσουμε* το *νέο πολυώνυμο που προκύπτει επί* b^n*. Για παράδειγμα*

```
p(k_):=(x^3-7*x-7;2^3*p(3/2*x))Expand
```

```
p(k_):=(x^3-7*x-7;2^3*p(3/2*x))Expand
```

$$56 - 84\,x - 27\,x^3$$

ή

```
p(k_):=(x^3-7*x-7;3^3*p(2/3*x))Expand
```

$$189 - 126\,x - 8\,x^3$$

Αναφέρουμε εν παραδώ ότι αν k = b > 0, όπου b είναι ένα ***πάνω*** *φράγμα στις απόλυτες τιμές των ριζών τον p(x), τότε το νέο πολυώνυμο p(bx) = $p_s(x)$ = 0 θα έχει όλες τις ρίζες τον μέσα στον μοναδιαίο κύκλο.*

Έχοντας δει την επίδραση των γεννητριών αντικαταστάσεων, στις ρίζες μιας πολυωνυμικής εξίσωσης

p(x) = 0 με ακέραιους συντελεστές και με μία μεταβλητή θα εξετάσουμε στην συνέχεια πως ***εκτελούνται*** *αυτές οι αντικαταστάσεις.*

Η αντιστροφή, x ← $\frac{1}{x}$ *και το τάνυσμα, x ← kx,, εκτελούνται ευκολότατα — η μεν πρώτη (αντικατάσταση) με απλή αντιστροφή της τάξης (σειράς) των συντελεστών τον πολυωνύμου η δε δεύτερη κλιμακώνοντας τους συντελεστές με δυνάμεις τον k.*

Η μετάθεση, x ← k + x,, είναι η αντικατάσταση με το μεγαλύτερο ενδιαφέρον. Εκτός από τον υπολογισμό της με το ανάπτυγμα κατά Taylor, εκτελείται πολύ αποτελεσματικά και με την μέθοδο των Ruffim-Homer — εδάφιο 4.1 του πρώτου τόμου.

6.3.3 Το *θεώρημα του* Vincent: *επέκτασή του και εφαρμογή του*

Στο εδάφιο αυτό θα εξετάσουμε το θεώρημα τον Vincent τον 1836, το οποίο αποτελεί την βάση της μεθόδου των συνεχών κλασμάτων για την απομόνωση των πραγματικών ριζών πολυωνυμικών εξισώσεων.

Αρχίζουμε με μία πιο λεπτομερή μελέτη τον θεωρήματος των Cardano-Descartes. Όπως είδαμε, βάσει τον θεωρήματος αυτού ο αριθμός ρ+ των θετικών ριζών μιας πολυωνυμικής εξίσωσης p(x) = 0 δεν μπορεί να είναι μεγαλύτερος από τον αριθμό ν των μεταβολών πρόσημου της ακολουθίας των συντελεστών τον, και στην περίπτωση μη ισότητας ισχύει η σχέση **ν =** □ **+ 2λ,,** *όπου* λ ∈ □ □0.

Το θεώρημα των Cardano-Descartes είναι σχετικά αδύνατο διότι μας δίνει τον ακριβή αριθμό των ριζών μόνο στις εξής δύο περιπτώσεις:

- αν στους συντελεστές *ενός πολυωνύμου δεν υπάρχει* ***καμία*** *μεταβολή πρόσημου, δηλαδή* **ν = 0,** *τότε δεν υπάρχει* ***καμία*** *θετική ρίζα, δηλαδή* □□ **= 0,** *και*
- *αν στους* συντελεστές *ενός πολυωνύμου υπάρχει* ***μία*** *μεταβολή πρόσημου, δηλαδή* **ν = 1,** *τότε υπάρχει* ***μία*** *θετική ρίζα, δηλαδή* □□ **= 1.**

Όπως θα δούμε στην συνέχεια οι περιπτώσεις αυτές χρησιμοποιούνται (εκτός των άλλων) σαν κριτήριο τερματισμού στην μέθοδο απομόνωσης πραγματικών ριζών με συνεχή κλάσματα. ***Το αξιοσημείωτο είναι πως από τις παραπάνω δύο περιπτώσεις μόνο στην πρώτη ισχύει και το αντίστροφο!*** *Δηλαδή ισχύει το εξής:*

Λήμμα **(Stodola):**
Αν η πολυωνυμική εξίσωση

$$p(x) = c_n \square x^n + c_{n\square 1} \square x^{n\square 1} + \equiv + c_0 = 0, \qquad (c_n > 0)$$

με πραγματικούς συντελεστές C_i, $0 \le i \le n$, *έχει ρίζες με* ***μόνο αρνητικά πραγματικά μέρη,*** *τότε όλοι οι συντελεστές της είναι θετικοί και επομένως δεν παρουσιάζουν καμία μεταβολή πρόσημου.*

Απόδειξη:
Έστω ότι $-\alpha_\rho$, $\rho = 1, 2, \ldots, k$ *είναι οι αρνητικές ρίζες, και* □□□ ± ™□, θ = 1, 2, …, *{είναι οι μιγαδικές ρίζες της* $p(x) = 0$, *όπου* $\alpha_\rho > 0$ *και* □ > 0*για όλα τα* ρ *και* q. *Το*

πολυώνυμο p(x) μπορεί να γραφτεί σαν το γινόμενο γραμμικών όρων όλων των ριζών τον, δηλαδή σαν

$$p(x) = C_n \prod_{i=1} x \square \square \square \prod_{i=1} x \square \square^2 \square \square^2$$

όπου όλοι οι παράγοντες έχουν θετικούς συντελεστές. Συνεπώς, οι συντελεστές του γινομένου θα είναι όλοι τους θετικοί και δεν θα παρουσιάζουν καμία μεταβολή πρόσημου Jl

Χωρίς προϋποθέσεις, το αντίστροφο της δεύτερης περίπτωσης δεν ισχύει:

*δηλαδή αν ένα πολυώνυμο έχει **μία** θετική ρίζα τότε **δεν συνεπάγεται** πως αναγκαστικά θα υπάρχει στους συντελεστές τον **μία** μεταβολή πρόσημου. Αυτό φαίνεται καθαρά από το πολυώνυμο*

$(x - 1)(x - ™)(x + ™) = x^3 - x^2 + x - 1,$

*το οποίο ενώ έχει **μία θετική ρίζα** —* και *δύο μιγαδικές — παρουσιάζει **τρεις μεταβολές πρόσημου.***

Με ορισμένες προϋποθέσεις όμως ισχύει και στην δεύτερη περίπτωση το αντίστροφο. Ισχύει το εξής:

Λήμμα **(Akritas - Danielopoulos):**

*Έστω p(x) = 0 μία πολυωνυμική εξίσωση βαθμού n > 1, με πραγματικούς συντελεστές, χωρίς πολλαπλές ρίζες και η οποία έχει μόνο **μία θετική** ρίζα* $\alpha \neq 0$ *και* **n - 1 *ρίζες*** $\alpha_1, \alpha_2,$ **...** $\square_{n\square 1}$, ***με αρνητικό πραγματικό μέρος*** — εκ των *οποίων οι μιγαδικές ρίζες εμφανίζονται σε συζυγή ζεύγη. Έστω επιπλέον ότι* οι *n - 1 ρίζες με το αρνητικό πραγματικό μέρος μπορούν να γραφτούν σαν*

$\square_j = -(1 + \square_j), \quad j = 1, 2, \ldots, n - 1$

με όλα τα β_j *να ικανοποιούν την **ανισότητα*** $\cup \square_j \cup < \square_n$, *όπου*

$\square_n = (1 + \frac{1}{n}\ 1\ n\square 1 - 1.$

Δηλαδή, οι n - 1 ρίζες με το αρνητικό πραγματικό μέρος βρίσκονται όλες μέσα σε έναν κύκλο με κέντρο το -1 και ακτίνα ε_η. Τότε η ακολουθία των συντελεστών τον p(x) ***παρουσιάζει ακριβώς μία μεταβολή πρόσημου.***

Απόδειξη:

Εκτός από έναν σταθερό παράγοντα το πολυώνυμο γράφεται και ως

$p(x) = (x - \alpha)(x - \square_1)\equiv(x - \square_{n\square 1})$

$= (x - \alpha)(x + 1\square\square_1)\equiv(x + 1\square\square_{n\square 1})$

$= (x - \alpha)(x^{n\square 1} + c_1\square x^{n\square 2} + \equiv + c_{n\square 1})$.

Για τους συντελεστές c_k, $1 \le k \le n - 1$, *ισχύει*

$$c_k = \sum 1\square\square_1\ \square 1\square\square_2\ \square\ 1\square\square_k ,$$

όπου το *άθροισμα αποτελείται από* $\binom{n\square 1}{k}$ *όρους.* Προφανώς, το *πολυώνυμο p(x) μπορεί να γραφτεί επιπλέον και ως*

$p(x) = x^n + (c_1 - \alpha) x^{n\square 1} + (c_2 - c_1\alpha) x^{n\square 2} + \equiv + (c_{n\square 1} - c_{n\square 2}\alpha)x - c_{n\square 1}\alpha.$

Αν τώρα *αποδείξουμε πως* $c_k > 0$, $1 \le k \le n - 1$, ***και*** *πως ο λόγος* $\frac{c_k}{c_{k\square 1}}$ *όπου* $c_0 = 1$ *ελαττώνεται για αυξάνοντα k, τότε προφανώς το πολυώνυμο p(x)* ***έχει ακριβώς μία μεταβολή πρόσημου.***

Ck > **0:** *Χρησιμοποιώντας το* λήμμα *του* Stodola *που αναφέραμε παραπάνω η απόδειξη του*

Ck > 0, $1 \le k \le n - 1$, *είναι άμεση διότι αυτοί είναι συντελεστές του πολυωνύμου μέσα στην παρένθεση* $p(x) = (x - \alpha)(x^{n-1} + c_1 x^{n-2} + \ldots + c_{n-1})$ *που έχει ρίζες μόνο με αρνητικά πραγματικά μέρη.*

$\frac{c_{k\square 1}}{c_k} < \frac{c_k}{c_{k\square 1}}$ *Για να αποδείξουμε πως ο λόγος* $\frac{c_k}{c_{k\square 1}}$ *όπου* $c_0 = 1$, *ελαττώνεται για αυξάνοντα*

k, $1 \le k \le n - 1$, *πρέπει να βρούμε με τι ισούνται οι συντελεστές* c_k. *Παρατηρούμε πως για κάθε*

έναν από τους $\binom{n\square 1}{k}$ *όρους του αθροίσματος έχουμε*

$\cup(1 + \square_1)(1 + \square_2)\equiv(1 + \square_k) - 1\cup \le (1 + \cup \square_1\cup)(1 + \cup \square_2\cup)\equiv(1 + \cup \square_k\cup) - 1$

και επειδή εξ υποθέσεως έχουμε $\cup \square_j \cup < \square_n$, $1 \le j \le n-1$ *προκύπτει*

$(1 + \cup \square_1 \cup)(1 + \cup \square_2 \cup) \equiv (1 + \cup \square_k \cup) - 1 \quad \le (1 + \square_n \quad - 1$

$\le (1 + \square_n \quad ^{n \square 1} - 1 = \frac{1}{n}$

Επομένως μπορούμε να γράψουμε

$c_k = \prod_{k}^{n \square 1} (1 + \square_k)$,

όπου $\cup \square_k \cup \le \frac{1}{n}$ *Για μία ακόμα φορά φαίνεται πως* $c_k > 0$, $1 \le k \le n$

Χρησιμοποιώντας την έκφραση $c_k = \prod_{k}^{n \square 1} (1 + \square_k)$,*προκύπτει*

$$\frac{c_k}{c_{k \square 1}} = \frac{n \square k}{k} \frac{1 \square\square_k}{1 \square\square_{k \square 1}}$$

Και $\frac{c_{k \square 1}}{c_k} = \frac{n \square k \square 1}{k \square 1} \frac{1 \square\square_{k \square 1}}{1 \square\square_k}$

Επομένως για να δείξουμε πως $\frac{c_{k \square 1}}{c_k} < \frac{c_k}{c_{k \square 1}}$ *πρέπει να αποδείξουμε πως*

$$\frac{k\, n \square k \square 1}{k \square 1\, \lceil n \square k} < \frac{1 \square\square_k{}^2}{1 \square\square_{k \square 1}\, \lceil 1 \square\square_{k \square 1}}.$$

Αυτό όμως *ισχύει* διότι αφ' ενός μεν *έχουμε*

$$\frac{k\, n \square k \square 1}{k \square 1\, \lceil n \square k} = 1 - \frac{n}{k \square 1\, \lceil n \square k} \le 1 - \frac{4 \lceil n}{n \square 1^{\,2}} = \frac{n \square 1^{\,2}}{n \square 1^{\,2}}$$

επειδή $\frac{n}{k \square 1\, \lceil n \square k} \ge \frac{4 \lceil n}{n \square 1^{\,2}}$,, *όπως βλέπουμε και με το Mathematica*

```
Assuming ⌊⌊n, k⌋⌋ Integers && n ⩾ k && k ⩾ 1, Refine ⌊⌊ n / (k □ 1 ⌈ n □ k) ⌋⌋ □ ⌊⌊ 4 ⌈ n / (n □ 1)^2 ⌋⌋
```

True

αφετέρου δε, λόγω της σχέσης $\cup \square_k \cup \le \frac{1}{n}$,

$$\frac{1 \square\square_k{}^2}{1 \square\square_{k \square 1}\, \lceil 1 \square\square_{k \square 1}} > \frac{1 \square \frac{1}{n}{}^2}{1 \square \frac{1}{n}{}^2} = \frac{n \square 1^{\,2}}{n \square 1^{\,2}}.$$

Επειδή λοιπόν αφ' ενός μεν $c_k > 0$, $1 \le k \le n - 1$, αφ' *ετέρου δε ο λόγος* $\frac{c_k}{c_{k \square 1}}$ όπου $c_o = 1$, *ελαττώνεται για αυξάνοντα k*, $1 \le k \le n - 1$, το *πολυώνυμο* p(x) *παρουσιάζει μία μεταβολή πρόσημου.//*

Το *θεώρημα του* Vincent που *ακολουθεί στηρίζεται στα δύο προηγούμενα λήμματα.*

***Θεώρημα του* Vincent (1836):**

Αν σε μία πολυωνυμική εξίσωση με ρητούς συντελεστές και χωρίς πολλαπλές ρίζες *κάνουμε διαδοχικές αντικαταστάσεις της μορφής*

$x \leftarrow a_1 + \frac{1}{x}$, $x \leftarrow a_2 + \frac{1}{x}$, $x \leftarrow a_3 + \frac{1}{x}$, …

όπου $a_1 \geq 0$ είναι *τυχαίος* ***μη αρνητικός*** *ακέραιος και* a_2, a_3, … είναι *τυχαίοι* ***θετικοί*** *ακέραιοι,* $a_i > 0$, $i > 1$, τότε το νέο *πολυώνυμο που προκύπτει* ***είτε*** *δεν έχει καμία,* ***είτε*** *έχει μία μεταβολή πρόσημου. Στην τελευταία περίπτωση η εξίσωση έχει ακριβώς μία θετική ρίζα που παρίσταται από το συνεχές κλάσμα*

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}$$

ενώ στην πρώτη περίπτωση δεν υπάρχει θετική ρίζα.

Απόδειξη:

Η απόδειξη βρίσκεται στο άρθρο του Vincent του 1836 και *παραλείπεται.* Αντί *αυτής θα παρουσιάσουμε πιο κάτω την απόδειξη ενός γενικότερου θεωρήματος.//*

Προφανώς, το *θεώρημα του* Vincent *απομονώνει μόνο τις θετικές ρίζες ενός πολυωνύμου* p(x). *Οι αρνητικές ρίζες απομονώνονται — όπως ακριβώς πρότεινε ο* Sturm *— αφού γίνουν πρώτα θετικές, με την αντικατάσταση* $x \leftarrow -x$ *στο* p(x). *Η γενικότητα του θεωρήματος του* Vincent *δεν περιορίζεται από τον όρο να μην έχει το πολυώνυμο p(x) πολλαπλές ρίζες, διότι στην αντίθετη περίπτωση το διασπάμε σε παράγοντες ελεύθερους από τετράγωνα — και απομονώνουμε τις ρίζες κάθε ενός από αυτούς.*

Ο ίδιος ο Vincent *αναφέρει πως το θεώρημα αυτό το υπαινίχθηκε ο* Fourier *το* 1827, *αλλά ποτέ δεν το απέδειξε — ή αν το απέδειξε, η απόδειξη δεν βρέθηκε ποτέ. Επιπλέον η βασική ιδέα του θεωρήματος χρησιμοποιήθηκε πολύ πιο πριν από τον* Lagrange.

Η εξάρτηση του θεωρήματος του Vincent *από εκείνο του* Budan *φαίνεται εύκολα αν κάθε αντικατάσταση της μορφής* $x \leftarrow a_i + \frac{1}{x}$ *αντικατασταθεί από τις ισοδύναμες αντικαταστάσεις*

$\{x \leftarrow a_i + x, x \leftarrow \frac{1}{x}\}$

Μιλώντας ***διαισθητικά,*** *ο σκοπός των διαδοχικών αντικαταστάσεων της μορφής*

$x \leftarrow a_i + \frac{1}{x}$ *που γίνεται στο πολυώνυμο* p(x), ***βαθμού n,*** *για να απομονωθούν οι ρίζες του είναι* ***διπλός:***

σκοπός 1ος: *αναγκάζει τις αρνητικές ρίζες να* ***μπουν*** *σε έναν κύκλο με κέντρο το* -1 και *ακτίνα*

$\square_n = (1 + \frac{1}{n}\ 1\ n\square 1$ - 1— *βλέπε και το λήμμα των* Akritas-Danielopoulos που *αναφέραμε πιο πάνω*

σκοπός 2ος: *αναγκάζει τις θετικές ρίζες να* ***κατανεμηθούν*** *σύμφωνα με έναν από τους εξής δύο τρόπους:* ***είτε*** *μία από τις θετικές ρίζες είναι στο διάστημα* (0, 1) *ενώ οι υπόλοιπες είναι στο διάστημα* (1, ∞) ***είτε*** *μία από τις θετικές ρίζες είναι στο διάστημα* (1, ∞)*ενώ οι υπόλοιπες είναι στο διάστημα* (0, 1) — *εξαιρώντας την περίπτωση να είναι το* 1 *ρίζα του p(x).*

Αν μία από τις θετικές ρίζες είναι στο διάστημα (0, 1), *ενώ οι υπόλοιπες είναι* στο *διάστημα*

(1, ∞), *τότε από την επιπλέον αντικατάσταση της μορφής* $x \leftarrow \frac{1}{1\square x}$ *προκύπτει ένα πολυώνυμο με μία θετική ρίζα και μία μεταβολή πρόσημου! Αντίθετα, αν μία από τις θετικές ρίζες είναι στο διάστημα* (1, ∞),, *ενώ οι υπόλοιπες είναι στο διάστημα* (0, 1), *τότε από την επιπλέον αντικατάσταση της μορφής* $x \leftarrow 1+x$ *προκύπτει ένα πολυώνυμο με μία θετική ρίζα και μία μεταβολή πρόσημου!*

Στο θεώρημα του Vincent *εύκολα γεννιέται η ερώτηση σχετικά με τον μέγιστο αριθμό αντικαταστάσεων που πρέπει να γίνουν για να προκύψει το πολυώνυμο με το πολύ μία μεταβολή πρόσημου.*

Η απάντηση στο ερώτημα αυτό δόθηκε από τον Uspensky που γενίκευσε το *θεώρημα του* Vincent *και απέκτησε ένα πάνω φράγμα στον αριθμό των αντικαταστάσεων. Όμως η παρουσίασή του* Uspensky *είχε ορισμένα λάθη στην διατύπωση και απόδειξη τα οποία διορθώθηκαν από τον γράφοντα. Ακολουθεί το γενικευμένο θεώρημα:*

Θεώρημα **(Vincent-Uspensky-Akritas):**

Έστω $p(x) = 0$ *μία πολυωνυμική εξίσωση βαθμού* $n > 1$, με *ρητούς συντελεστές και χωρίς πολλαπλές ρίζες και έστω επιπλέον ότι* $\Delta > 0$ *είναι η* ***ελάχιστη απόσταση*** *των ριζών του* $p(x)$ *— όπως αυτή ορίσθηκε στην ενότητα* 6.2.3. *Ας συμβολίσουμε επιπλέον με* m *τον μικρότερο δείκτη έτσι ώστε*

$$F_{m+1}\frac{\Delta}{2} > 1, \quad \text{και} \quad F_{m+1}F_m\Delta > 1 + \frac{1}{\alpha_n}, \qquad (1)$$

όπου F_k *είναι το* k*-στό μέλος της ακολουθίας* Fibonacci 1, 1, 2, 3, 5, 8, 13, ... και

$$\alpha_n = \left(1 + \frac{1}{n}\right)^{n+1} - 1, \qquad (2)$$

η ακτίνα τον κύκλου γύρω από το -1 — που αναφέραμε πριν από το θεώρημα αυτό. Τέλος έστω $a_i \geq 0$ *τυχαίος* ***μη αρνητικός*** *ακέραιος και* $a_2, a_3, \ldots, a_m$ *τυχαίοι* ***θετικοί*** *ακέραιοι,* $a_i > 0$, $1 < i \leq m$. *Τότε με την αντικατάσταση*

$$x \leftarrow a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_m + \frac{1}{x}}}},$$

που είναι *ισοδύναμη με την ακολουθία των διαδοχικών αντικαταστάσεων της μορφής* $x \leftarrow a_i + \frac{1}{x}$, $1 \leq i \leq m$, *από την εξίσωση* $p(x) = 0$ *προκύπτει η εξίσωση* $p_{ti}(x) = 0$,, *που έχει* ***το πολύ μία μεταβολή πρόσημου***.

Απόδειξη:

Για να αποδείξουμε το θεώρημα αρκεί να δείξουμε πως μετά τις διαδοχικές αντικαταστάσεις της μορφής $x \leftarrow a_i + \frac{1}{x}$, $1 \leq i \leq m$ *τα πραγματικά μέρη όλων των μιγαδικών ριζών γίνονται αρνητικά, όπως επίσης γίνονται αρνητικές και όλες οι πραγματικές ρίζες* ***εκτός από το πολύ μία. Σημειώστε*** *πως οι ρίζες της εξίσωσης* $p_i(x) = 0$, ***"μαζεύονται"*** *μέσα σε έναν μικρό κύκλο γύρω από το -1.*

Πράγματι, έστω ότι $\frac{p_k}{q_k}$ *είναι η* k*-στή* ***συγκλίνουσα*** *στο συνεχές κλάσμα*

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \frac{1}{a_k}}}$$

Από το εδάφιο 3.5 του πρώτου τόμου ξέρουμε πως για $k \geq 0$, $p_{-1} = 0$, $p_0 = 1$, $q_{-1} = 1$ και $q_0 = 0$,, *ισχύουν οι τύποι*

$p_{k+1} = a_{k+1}\, p_k + p_{k-1}$,

$q_{k+1} = a_{k+1}\, q_k + q_{k-1}$.

Επειδή $q_1 = 1$ και $q_2 = a_2 \geq 1$, *έπεται πως* $q_k \geq F_k$. *Επιπλέον,* η *αντικατάσταση (3)* *μπορεί να γραφεί ως*

$$x \leftarrow \frac{p_m x + p_{m-1}}{q_m x + q_{m-1}}$$

από την οποία προκύπτει το πολυώνυμο $p_{ti}(x)$*Δηλαδή*

Προφανώς, με την αντίστροφη αντικατάσταση

$p_{ti}(x) = p\left(\frac{p_m x + p_{m-1}}{q_m x + q_{m-1}}\right)$.

αποκτούμε πάλι το πολυώνυμο p(x). Δηλαδή

$$x \leftarrow -\frac{p_{m-1} - q_{m-1} x}{p_m - q_m x}$$

Προσέξτε πως η *αντίστροφη αντικατάσταση που αναφέραμε αντιστοιχεί στον πίνακα* $\begin{matrix} q_{m-1} & -p_{m-1} \\ -q_m & p_m \end{matrix}$, τον *αντίστροφο του πίνακα* $\begin{matrix} p_m & p_{m-1} \\ q_m & q_{m-1} \end{matrix}$ με την *ισότητα που ορίσαμε.*

Επομένως, αν α είναι μία ρίζα της εξίσωσης $p(x) = 0$, *η* ***ποσότητα***

$$\alpha = -\frac{p_{m-1} - q_{m-1}\alpha}{p_m - q_m\alpha}$$

που *ορίζεται είναι η* ***αντίστοιχη ρίζα*** *της εξίσωσης* $p_{ti}(x) = 0$. *Θα εξετάσουμε τις* ***περιπτώσεις*** *όπου η ρίζα α της εξίσωσης* $p(x) = 0$ *είναι μιγαδική ή πραγματική και θα δείξουμε πως οι αντίστοιχες ρίζες της εξίσωσης* $p_{ti}(x) = 0$ *έχουν όλες αρνητικό πραγματικό μέρος —* ***εκτός από μία το πολύ πραγματική ρίζα*** — και *είναι μέσα σε έναν μικρό κύκλο γύρω από το -1.*

Μιγαδική ρίζα:

Έστω ότι α είναι μία μιγαδική ρίζα της εξίσωσης $p(x) = 0$. *Έστω δηλαδή* $\alpha = a \pm ib$, $b \neq 0$.

Στην περίπτωση αυτή κάνοντας τις πράξεις βλέπουμε πως το ***πμ(α),*** *το* ***πραγματικό μέρος*** *της αα, της αντίστοιχης ρίζας της εξίσωσης* $p_{ti}(x) = 0$, *είναι*

$$\pi\mu(\alpha) = -\frac{p_{m-1} - q_{m-1}a - p_m - q_m a - q_{m-1} q_m b^2}{p_m - q_m a^2 - q_m^2 b^2}$$

Θέλουμε να αποδείξουμε πως πμ(α) < 0. *Στην περίπτωση που στην* (5) $p_{m-1} - q_{m-1}a - p_m - q_m a \geq 0$*προφανώς το πμ(α) είναι αρνητικό και τελειώσαμε.*

Στην περίπτωση όμως που $p_{m+1} - q_{m+1}a < p_m - q_m a < 0$,, *πρέπει να αποδείξουμε πως*

$q_{m+1}(q_m b^2 > \cup p_{m+1} - q_{m+1}a < p_m - q_m a \cup$.*Προς τούτο προσέξτε πως η τιμή του a περιέχεται ανάμεσα στις δύο διαδοχικές συγκλίνουσες* $\frac{p_{m+1}}{q_{m+1}}$ *και* $\frac{p_m}{q_m}$ *η διαφορά των οποίων σε απόλυτη τιμή είναι*

$\frac{p_{m+1}}{q_{m+1}} - a < \frac{1}{q_{m+1}q_m}$ και $\frac{p_m}{q_m} - a < \frac{1}{q_{m+1}q_m}$,

Από τις οποίες συνεπάγεται ότι

$\cup p_{m+1} - q_{m+1}a < p_m - q_m a \cup < \frac{1}{q_{m+1}q_m} \le 1.$

Επομένως, έχουμε τις ανισότητες Από τις (5) και (6) *συμπεραίνουμε πως το πμ(ά) θα είναι αρνητικό* ***αν***

$q_{m+1}(q_m b^2 > 1.$

Για να *αποδείξουμε την τελευταία ανισότητα προσέξτε πως* επειδή Δ είναι η *ελάχιστη απόσταση των ριζών του p(x) έχουμε*

$\cup(a + \text{™}b) - (a - \text{™}b)\cup = \cup 2\text{™}b\cup = 2\cup b\cup \ge \Delta,$

από την οποία *προκύπτει* $\cup b\cup \ge \frac{\Delta}{2}$. *Επιπλέον ξέρουμε πως* $q_m \ge q_{m+1} \ge F_{m+1}$, *και από την (1) πως* $F_{m+1}\frac{\Delta}{2} > 1$. *Επομένως ισχύει* $F_{m+1}\cup b\cup > 1$, *από την οποία έπονται και οι ανισότητες*

$q_{m+1}\cup b\cup > 1$ και $q_m\cup b\cup > 1.$

Από αυτές τις τελευταίες δύο ανισότητες προκύπτει $q_{m+1}(q_m b^2 > 1$, *το οποίο αποδεικνύει πως το πμ(α)* ***είναι αρνητικό.*** *Αυτό φυσικά ισχύει για* ***όλες*** *τις μιγαδικές ρίζες της εξίσωσης* $p_{ti}(x)=0$.*Υπενθυμίζουμε πως η* $p_{ti}(x)=0$.*προκύπτει από την p(x) = 0 με αντικατάσταση της μορφής (3).*

Πραγματική ρίζα:

Ας θεωρήσουμε κατ' αρχάς την περίπτωση ότι για ***όλες*** *τις πραγματικές ρίζες* ξ_i, *της εξίσωσης p(x) = 0 ισχύει η ανισότητα*

$p_{m+1} - q_{m+1}\xi_i < p_m - q_m \xi_i > 0.$

Από τις (4) και (5) έπεται πως όλες οι πραγματικές ρίζες της εξίσωσης $p_{ti}(x) = 0$ *θα είναι αρνητικές. Επιπλέον ξέρουμε πως όλες οι μιγαδικές ρίζες της εξίσωσης* $p_i(x) = 0$ *έχουν αρνητικό πραγματικό μέρος. Συνεπώς, από το Λήμμα του Stodola προκύπτει πως δεν υπάρχει μεταβολή πρόσημου στο πολυώνυμο* $p_{ti}(x)$

Έστω λοιπόν τώρα ότι για ***κάποια*** *πραγματική ρίζα α της εξίσωσης* $p(x) = 0$ *ισχύει*

$$p_{m+1} \cdot q_{m+1} - p_m \cdot q_m \le 0. \quad (7)$$

Τότε *προφανώς η ρίζα α περιέχεται ανάμεσα στις δύο διαδοχικές συγκλίνουσες* $\frac{p_{m+1}}{q_{m+1}}$ και $\frac{p_m}{q_m}$ *,* ***είναι θετική*** *και ισχύει* $\frac{p_m}{q_m} - \alpha < \frac{1}{q_{m+1} q_m}$ *. Έστω ότι* $\xi_k \neq \alpha$,είναι μία **άλλη** ρίζα, *πραγματική* ή *μιγαδική, της εξίσωσης* $p(x) = 0$*, διάφορη της ρίζας* α*, και έστω ότι*

$$\xi_k = -\frac{p_{m+1} - q_{m+1} \xi_k}{p_m - q_m \xi_k}$$

είναι η αντίστοιχη ρίζα της $p_{ti}(x) = 0$.. *Τότε αν λάβουμε υπ όψη ότι*

$$p_m q_{m+1} - p_{m+1} q_m = (-1)^m,$$

και *προσθέσουμε στην ρίζα* ξ_k την *ποσότητα* $\frac{q_{m+1}}{q_m}$ *προκύπτει η ισότητα*

$$\xi_k + \frac{q_{m+1}}{q_m} = \frac{(-1)^m}{q_m \left(p_m - q_m \xi_k\right)}$$

που γράφεται και

$$\xi_k = -\frac{q_{m+1}}{q_m} \left(1 - \frac{(-1)^m}{q_{m+1} \left(q_m \frac{p_m}{q_m} - \xi_k\right)}\right) = -\frac{q_{m+1}}{q_m} (1 + \varepsilon_k),$$

όπου

$$\varepsilon_k = \frac{(-1)^{m+1}}{q_{m+1} \left(q_m \frac{p_m}{q_m} - \xi_k\right)}.$$

Επειδή δε ξέρουμε πως $q_m \ge q_{m+1} \ge F_{m+1}$*,και από την (l) πως* $F_{m+1} \frac{\Delta}{2} > 1$*, ισχύει*

$$\frac{p_m}{q_m} - \xi_k = \frac{p_m}{q_m} - \alpha + \alpha - \xi_k \ge \Delta - \left|\frac{p_m}{q_m} - \alpha\right| \ge \Delta - \frac{1}{q_{m+1} q_m} > 0$$

και συνεπώς

$$|\varepsilon_k| \le \frac{1}{q_{m+1} \left(q_m \Delta - 1\right)} \le \frac{1}{F_{m+1} \left(F_m \Delta - 1\right)}.$$

Από την παραπάνω ανισότητα και την ανισότητα $F_{m+1} F_m \Delta > 1 + \frac{1}{n}$ της (l) *συνεπάγεται πως* $|\varepsilon_k| \le \varepsilon_n$.

Εκτός *λοιπόν από την* ***θετική*** *ρίζα* α, όλες οι *υπόλοιπες ρίζες* ξ_k *της εξίσωσης* $p_{ti}(x) = 0$ που *αντιστοιχούν στις ρίζες* a_k *της εξίσωσης* $p(x) = 0$ *— και που είναι όλες διάφορες της ρίζας α — είναι* της *μορφής*

$$\xi_k = -\frac{q_{m+1}}{q_m} (1 + \varepsilon_k), \quad |\varepsilon_k| \le \varepsilon_n, \quad 1 \le k \le n - 1. \quad (8)$$

Δηλαδή, οι *η - 1 ρίζες της εξίσωσης* $p_{ti}(x) = 0$ — που *προκύπτει από την* $p(x) = 0$ *με αντικατάσταση της μορφής* (3) — *έχουν* όλες τους *αρνητικό πραγματικό μέρος και έχουν μαζευτεί γύρω από το -1.*

Ορίζουμε τώρα το πολυώνυμο

$$q_{ti}(x) = (x - \square)(x + (1 + \square_1)) \equiv (x + (1 + \square_k)),$$

το *οποίο πληροί ης συνθήκες του Λήμματος των* Akritas-Danielopoulos *και επομένως* ***παρουσιάζει μία μεταβολή πρόσημου!*** *Επειδή όμως ισχύει*

$$p_{ti}(x) = \frac{q_{m+1}}{q_m}^{n+1} q(x)$$

έπεται πως και το $p_{ti}(x)$ *επίσης* ***παρουσιάζει μία μεταβολή πρόσημου!***

Το μόνο που μένει να εξετάσουμε είναι η περίπτωση που η (7) είναι ισότητα, δηλαδή

$$p_{m+1} \cdot q_{m+1} - p_m \cdot q_m = 0.$$

Αν $p_{m+1} \cdot q_{m+1} = 0$ *τότε έπεται πως* $\square = -\frac{p_{m+1} \cdot q_{m+1}}{p_m \cdot q_m} = 0$,, *και η εξίσωση* $p_{ti}(x) = 0$ *δεν έχει μεταβολή πρόσημου (Λήμμα του Stodola). Αν πάλι* $p_m \cdot q_m = 0$,, *τότε έπεται πως*

$\square = -\frac{p_{m+1} \cdot q_{m+1}}{p_m \cdot q_m} = \infty$, *και η εξίσωση* $p_{ti}(x) = 0$ ***μετασχηματίζεται σε εξίσωση βαθμού n - 1.*** *Επειδή δε όλες οι ρίζες της μετασχηματισμένης εξίσωσης έχουν αρνητικό πραγματικό μέρος, συμπεραίνουμε (Λήμμα του Stodola) πως η* $p_{ti}(x) = 0$ *δεν παρουσιάζει καμία μεταβολή πρόσημου. Έτσι απεδείχθηκε το θεώρημα εντελώς.//*

Από το παραπάνω θεώρημα βλέπουμε πως m είναι ένα πάνω φράγμα στον αριθμό των αντικαταστάσεων της μορφής $x \leftarrow a_i + \frac{1}{x}$ *που πρέπει να εκτελεσθούν έτσι ώστε το πολυώνυμο που προκύπτει να έχει το πολύ μία μεταβολή πρόσημου. ισχύει το εξής:*

Λήμμα:

Με *τις προϋποθέσεις του παραπάνω θεωρήματος ισχύει*

$$m = O\left(n \log n \cdot \square \cdot n \log p x \square\right)$$

Απόδειξη:

Εξ ορισμού m *είναι ο μικρότερος δείχτης έτσι ώστε να ισχύουν και οι δύο ανισότητες στην (1). Προφανώς, μία από αυτές* τις *δύο ανισότητες — και πιθανόν και οι δύο —* δεν θα *ισχύει αν ελαττώσουμε το* m *κατά ένα. Έστω* ότι *"χαλάει η πρώτη ανισότητα της (1) και γίνεται*

$F_{m-2} \frac{\Delta}{2} \le 1.$ *(9)*

Εφαρμόζοντας την σχέση $F_k = \frac{\varphi^k}{\sqrt{5}}$ *,, όπου f = 1.618… και η στρογγύλευση γίνεται προς τον πλησιέστερο ακέραιο, προκύπτει* $\varphi^{m-2} \le 2\sqrt{5}\frac{1}{\Delta}$ *και συνεπώς,*

$m \le 2 + \log_\varphi 2 + \frac{1}{2}\log_\varphi 5 - \log_\varphi \Delta$.. *(10)*

Επιπλέον από το θεώρημα του Mahler έχουμε

$\Delta \ge \sqrt{3}\, n^{n-2}\, 2^{2}\, p_1(x)^{n-1}$ *(11)*

οπότε συνδυάζοντας τις *(10) και (11) αποδεικνύουμε το ζητούμενο —* αν *αντί του* $p_1(x)$ *χρησιμοποιούμε το* $p_\varphi(x)$ *που είναι της ίδιας τάξης μεγέθους. Το ίδιο αποτέλεσμα προκύπτει αν υποθέσουμε* σm *"χαλάει η δεύτερη ανισότητα της (1).//*

Λαμβάνοντας υπ όψη ότι το *β-μήκος του βαθμού των πολυωνύμων για* τις *περιπτώσεις που εξετάζουμε είναι ,* $\lambda(n) = 1$, ή $\log n = 1$, *προκύπτει*

$m = O\left(n \log p_\varphi(x)\right)$ *. (12 Το θεώρημα του Vincent μπορεί να χρησιμοποιηθεί για την απομόνωση των πραγματικών ριζών μιας πολυωνυμικής εξίσωσης. Προσέξτε* ότι *εδώ —* σε *αντίθεση με το θεώρημα του Sturm —* ***δεν έχουμε άλλη επιλογή*** *από το να απομονώσουμε πρώτα* τις *θετικές και ύστερα* τις *αρνητικές ρίζες βάνοντας την αντικατάσταση* $x \leftarrow -x$*). Για να δείτε την εφαρμογή τον προσέξτε τα ακόλουθα σημεία από την παραπάνω απόδειξη:*
Η αντικατάσταση (3) τον *θεωρήματος που αποδείξαμε μπορεί να γραφτεί και ως*

$x \leftarrow \frac{p_m(x)-p_{m-1}}{q_m(x)-q_{m-1}}$,, (13)

όπου $\frac{p_k}{q_k}$ *είναι* η *k-στή* ***συγκλίνουσα*** *στο συνεχές κλάσμα*

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}$$

και όπως *αναφέραμε, για* $k \ge 0$, $p_{-1} = 0$, $p_0 = 1$, $q_{-1} = 1$ και $q_0 = 0$, *ισχύουν οι τύποι*
$p_{k-1} = a_{k-1}\, p_k + p_{k-1}$, (14)

$q_{k-1} = a_{k-1}\, q_k + q_{k-1}$.

a. Η απόσταση μεταξύ δύο διαδοχικών συγκλινουσών είναι

$$\frac{p_{m-1}}{q_{m-1}} - \frac{p_m}{q_m} = \frac{1}{q_{m-1} q_m}$$

Προφανώς, οι μικρότερες τιμές των q_m εμφανίζονται όταν $\forall i,\ a_i = 1$.. Τότε $q_m = F_m$, ο m-στός αριθμός Fibonacci. *Αυτό εξηγεί διαισθητικά την σχέση μεταξύ των αριθμών* Fibonacci και την *απόσταση Δ* των *ριζών.*

b. Έστω $p_{ti}(x) = 0$ η εξίσωση που προκύπτει από την $p(x) = 0$ με αντικατάσταση της μορφής (13) και έστω επιπλέον ότι ισχύουν οι προϋποθέσεις τον θεωρήματος που αποδείξαμε και ότι το $p_{ti}(x)$ παρουσιάζει μία μεταβολή πρόσημου. Τότε το πολυώνυμο $p_{ti}(x)$ παρουσιάζει μία μεταβολή πρόσημου. Τότε το πολυώνυμο $p_{ti}(x)$ θα έχει ***ακριβώς*** *μία θετική ρίζα α στην οποία αντιστοιχεί μία θετική ρίζα α τον πολυωνύμου $p(x)$. Οι ρίζες των δύο αυτών πολυωνύμων σχετίζονται με τον τύπο*

$$\alpha = -\frac{p_{m-1}\alpha + q_{m-1}}{p_m \alpha + q_m},$$

δηλαδή την σχέση (4) που *συναντήσαμε στην απόδειξη τον θεωρήματος. Προσέξτε πως η αντικατάσταση (13)* ***απεικονίζει το διάστημα*** $(0, \infty)$— *μέσα στο οποίο βρίσκεται η μοναδική θετική ρίζα* α του $p_{ti}(x)$— ***πάνω στο διάστημα*** *με άκρα τα σημεία* $\frac{p_{m-1}}{q_{m-1}}$ και $\frac{p_m}{q_m}$ *— μέσα στο οποίο βρίσκεται μία θετική ρίζα α τον $p(x)$. Τα άκρα τον διαστήματος,* $\frac{p_{m-1}}{q_{m-1}}$ και $\frac{p_m}{q_m}$ *βρίσκονται αν στην έκφραση* $\frac{p_m x + p_{m-1}}{q_m x + q_{m-1}}$ *της* (13) *αντικαταστήσουμε το x πρώτα με το 0 και ύστερα με το* ∞*αντίστοιχα.*

6.3.4 *Απομόνωση των θετικών ριζών με συνεχή κλάσματα*

Από την παραπάνω συζήτηση είναι φανερό πως η απομόνωση των θετικών ριζών ενός πολυωνύμου $p(x)$ με συνεχή κλάσματα δεν είναι τίποτε άλλο από τον ***υπολογισμό των μερικών πηλίκων*** $a_1, a_2, \ldots, a_m$ *για αντικαταστάσεις της μορφής* (3) που *οδηγούν σε πολυώνυμα $p_{ti}(x)$* με ***ακριβώς μία μεταβολή πρόσημου.***

Υψίστης σημασίας είναι το γεγονός πως το θεώρημα τον Budan — με την *δική τον ιδιόμορφη διατύπωση — χρησιμοποιείται σαν ένα είδος* ***"τερματικού τεστ'"*** *στον υπολογισμό κάθε ενός*

*από τα μερικά πηλίκα $a_1, a_2, \ldots, a_m$. Για παράδειγμα, αν τα πολυώνυμα $p_{\text{ti}}(x + a_i)$ και $p_{\text{ti}}(x + a_i + 1)$ έχουν τον ίδιο αριθμό μεταβολών πρόσημου, θέτουμε $a_i \leftarrow a_i + 1$ και συνεχίζουμε τον υπολογισμό τον a_i . Αν όμως πηγαίνοντας από **κάποιο ενδιάμεσο** πολυώνυμο $p_{\text{ti}}(x + a_i)$στο πολυώνυμο $p_{\text{ti}}(x + a_i + 1)$ **"χάσουμε"** μεταβολές πρόσημου, **τότε** η τιμή τον τυχαίου μερικού πηλίκου a_i έχει υπολογιστεί. Έχοντας υπολογίσει το a_i εκτελούμε την αντικατάσταση $x \leftarrow \frac{1}{x-1}$ για να αρχίσουμε τον υπολογισμό του επόμενου μερικού πηλίκου a_{i+1} ή να σταματήσουμε.*

*Υπάρχουν **δύο τρόποι υπολογισμού** των μερικών πηλίκων a_i — και συνεπώς δύο μέθοδοι απομόνωσης των θετικών ριζών ενός πολυωνύμου $p(x)$ με συνεχή κλάσματα. Ο πρώτος τρόπος αναπτύχθηκε από τον Vincent το 1836 ενώ ο δεύτερος αναπτύχθηκε από τον γράφοντα το 1978. Η διαφορά ανάμεσα στους δύο αυτούς τρόπους υπολογισμού των a_i είναι ανάλογη της διαφοράς που υπάρχει ανάμεσα στα ολοκληρώματα κατά Riemann και κατά Lebesgue. Δηλαδή, όπως ξέρουμε, το άθροισμα 1 + 1 + 1 + 1 + 1 μπορεί να υπολογισθεί κατά δύο τρόπους: κ**ατά** **Riemann** υπολογίζεται ως 1 + 1 = 2, 2 + 1 = 3, 3 + 1 = 4, 4 + 1 = 5, ενώ **κατά** **Lebesgue** υπολογίζεται ως 5·1 = 5.*

*Εκτός από τους **δύο** προαναφερθέντες τρόπους υπολογισμού των μερικών πηλίκων a_i, στην συνέχεια αναφέρουμε και μία "αποτυχημένη" προσπάθεια του Uspensky. Η αποτυχία αυτή οφείλεται στο γεγονός ότι ο Uspensky δεν γνώριζε το θεώρημα τον Budan.*

***Υπολογισμός τυχαίου μερικού πηλίκου a_i κατά* Vincent:**

Ο Vincent στο άρθρο τον τού 1836 υπολογίζει την τιμή του τυχαίου μερικού πηλίκου a_i με μοναδιαίες αυξήσεις της μορφής $a_i \leftarrow a_i + 1$. Σε κάθε τέτοια αύξηση αντιστοιχεί η αντικατάσταση $x \leftarrow x + 1$ που εκτελείται σε κάποιο ενδιάμεσο πολυώνυμο $p_{\text{ti}}(x)$ και ακολουθεί έλεγχος για πιθανό "χάσιμο" μεταβολών πρόσημου — χάσιμο που σηματοδοτεί και το τέλος τον υπολογισμού τον εν λόγω a_i.

*Τονίζουμε πως **μόνο αν** πηγαίνοντας από κάποιο ενδιάμεσο πολυώνυμο $p_{\text{ti}}(x + a_i)$ στο πολυώνυμο*

$p_{\text{ti}}(x + a_i\ p_{\text{ti}}\lfloor)$ *"χι* a_i ***ούν"*** *μεταβολές πρόσημου* ***τι*** $\frac{1}{x\square 1}$ ***και μόνον τότε, εκτελεί*** *ο Vincent στο πολυώνυμο* $(x + \quad)$ *την αντικατάσταση* $x \leftarrow \quad$ *, με σκοπό να αρχίσει τον υπολογισμό του επόμενου μερικού πηλίκου* a_{i+1} *ή να σταματήσει.*

Η μέθοδος αυτή τον Vincent οδηγεί σε μία ***εκθετική*** *μέθοδο απομόνωσης των θετικών ριζών, κάτι που είχε γίνει αντιληπτό και από τον Sturm και από τον Uspensky. Η εκθετική αυτή συμπεριφορά εμφανίζεται μόνο στην περίπτωση μεγάλων μερικών πηλίκων* a_i*. Για μικρά* a_i *η μέθοδος τον Vincent είναι πολύ ικανοποιητική.*

Παράδειγμα:

Έστω ότι θέλουμε να απομονώσουμε τις ρίζες του πολυωνύμου $p(x) = (x - \alpha)(x - \beta)$.

Επιλέγουμε $\alpha = 10^{\lfloor} + \varepsilon$, *όπου* $0 < \varepsilon < 1$, $\kappa \in \square\ \square 0$ *και αρκετά μεγάλο, και* $\beta = \alpha + \varepsilon$.

Στην περίπτωση αυτή, το πρώτο μερικό πηλίκο a_1 *είναι το ακέραιο μέρος της ρίζας α, δηλαδή* $a_1 = \delta\alpha\tau = 10^{\lfloor}$. .

Για να υπολογίσουμε το a_1 *με τον τρόπο τον Vincent θέτουμε* $a_1 \leftarrow 1$, $p_{\text{ti}}(x) \leftarrow p(x)$,*υπολογίζουμε το* $p_{\text{ti}}(x + 1)$*και ελέγχουμε για μείωση τον αριθμού των μεταβολών πρόσημου. Επειδή τα πολυώνυμα* $p_{\text{ti}}(x)$ *και* $p_{\text{ti}}(x + 1)$ *έχουν τον ίδιο αριθμό μεταβολών πρόσημου ξέρουμε από το θεώρημα τον Budan πως δεν υπάρχει ρίζα τον p(x) στο διάστημα (0, 1).*

Αυξάνουμε λοιπόν το a_1 *κατά μονάδα,* $a_1 \leftarrow a_1 + 1$, *θέτουμε* $p_{\text{ti}}(x) \leftarrow p_{\text{ti}}(x + 1)$, *υπολογίζαμε το* $p_{\text{ti}}(x + 1)$, *και ελέγχουμε ξανά για μείωση τον αριθμού των μεταβολών πρόσημου. Η διαδικασία αυτή επαναλαμβάνεται* 10^k *φορές και — αν το k επιλεχθεί κατάλληλα — μπορεί να διαρκέσει χρόνια στον γρηγορότερο υπολογιστή.*

Υπολογισμός τυχαίου μερικού πηλίκου a_i ***κατά*** **Uspensky:**

Όπως είδαμε παραπάνω η τιμή τον τυχαίου μερικού πηλίκου a_i *υπολογίζεται με την χρήση τον θεωρήματος τον Budan. Έτσι, αν τα πολυώνυμα* $p_{\text{ti}}(x)$ *και* $p_{\text{ti}}(x + 1)$ *έχουν τον ίδιο αριθμό μεταβολών πρόσημου ο Vincent προχωρεί στην επόμενη μοναδιαία αύξηση* $a_i \leftarrow a_i + 1$ —

και *φυσικά και στην επόμενη αντικατάσταση* $x \leftarrow x + 1$. *Και αυτό επειδή ξέρει πως δεν υπάρχει ρίζα τον* $p_{\text{ti}}(x)$*στο διάστημα (0, 1).*

Ο Uspensky στο βιβλίο τον (1949) υπολογίζει την τιμή του τυχαίου μερικού πηλίκου a_i *επίσης με μοναδιαίες αυξήσεις της μορφής* $a_i \leftarrow a_i + 1$ — και *φυσικά αντικαταστάσεις της μορφής* $x \leftarrow x + 1$. *Μην γνωρίζοντας όμως το θεώρημα τον Budan, ο Uspensky* ***δεν μπορεί να συμπεράνει*** *ότι το* $p_{\text{ti}}(x)$ *δεν έχει ρίζα στο διάστημα (0, 1), αν τα πολυώνυμα* $p_{\text{ti}}(x)$ και $p_{\text{ti}}(x + 1)$ *έχουν τον ίδιο αριθμό μεταβολών πρόσημου. Έτσι λοιπόν, για να βεβαιωθεί πως το* $p_{\text{ti}}(x)$ *δεν έχει ρίζα στο διάστημα (0, 1), ο Uspensky κάνει* ***σε κάθε βήμα,*** *εκτός από την αντικατάσταση* $x \leftarrow x + 1$, ***και*** *την αντικατάσταση*

$x \leftarrow \frac{1}{x \square 1}$ *από την οποία στην προκειμένη περίπτωση προκύπτει ένα πολυώνυμο χωρίς καμία μεταβολή πρόσημου. Διευκρινίζουμε πως με την αντικατάσταση* $x \leftarrow \frac{1}{x \square 1}$ *οι ρίζες του* $p_{\text{ti}}(x)$ *που είναι >1 γίνονται* ***αρνητικές*** *ενώ οι ρίζες που είναι < 1 γίνονται > 1.*

Έτσι ο Uspensky το μόνο που κατόρθωσε ήταν να ***διπλασιάσει*** *τον χρόνο υπολογισμού της μεθόδου τον Vincent. Συνεπώς οι ισχυρισμοί τον —* στην *εισαγωγή του βιβλίου του — ότι δήθεν ανακάλυψε μία νέα μέθοδο για την απομόνωση των ριζών δεν ευσταθούν.*

Η ***συμβολή του*** **Uspensky** *έγκειται στα εξής:*

a. *Για την αντικατάσταση* $x \leftarrow x + 1$ *χρησιμοποίησε την μέθοδο των Ruffini-Horner, ενώ ο Vincent το ανάπτυγμα Taylor.*
b. *Για να εξαφανίσει την εκθετική συμπεριφορά της μεθόδου πρότεινε αντί των αντικαταστάσεων*

$x \leftarrow x + 1$ *να γίνονται αντικαταστάσεις της μορφής* $x \leftarrow x + \kappa$, *όπου k επιλέγεται τυχαία και διαδοχικά αυξάνεται — κάτι που δεν πέτυχε. Προφανώς δεν είχε γίνει κατανοητή η γεωμετρική σημασία των μερικών πηλίκων* a_i.

Υπολογισμός τυχαίου μερικού πηλίκου a_i ***κατά τον γράφοντα:***

Στην διδακτορική τον διατριβή, 1978, ο γράφων ***με την βοήθεια του θεωρήματος του*** **Cauchy** *(ενότητα 6.2.5) υπολόγισε το τυχαίο μερικό πηλίκο* a_i *σαν ένα* ***κάτω φράγμα, (b,*** *στις τιμές των* ***θετικών*** *ριζών κάποιου ενδιάμεσου πολυωνύμου* $p_{\text{ti}}(x)$.
Θεωρητικά λοιπόν ο υπολογισμός τον a_i *γίνεται άμεσα θέτοντες* $a_i \leftarrow \{\beta$, $\{\beta \geq 1$, *και αυτό αντιστοιχεί στην αντικατάσταση* $x \leftarrow x + \{\beta$ *που εκτελείται στο* $p_{\text{ti}}(x)$.
Δεδομένου ότι οι αντικαταστάσεις $x \leftarrow x + 1$ *και* $x \leftarrow x + \{\beta$, $\{\beta \geq 1$, *εκτελούνται στον ίδιο περίπου χρόνο είναι προφανές ότι η εκθετική συμπεριφορά της μεθόδου εξαφανίσθηκε.*

Προσέξτε πως p_{ti} : $\forall i,\ a_i = \delta$ s τ, *όπου* a_5 *είναι η μικρότερη θετική ρίζα κάποιου ενδιάμεσου πολυωνύμου* (x).*Επειδή γενικά το κάτω φράγμα δεν μας δίνει το ακέραιο μέρος της μικρότερης ρίζας, το θεώρημα του* s auchy *θα χρειαστεί να εφαρμοσθεί περισσότερες από μία φορά για τον υπολογισμό το* $_0{}^9\delta$ τ.*Έτσι, στο παράδειγμα που είδαμε παραπάνω, για να υπολογιστεί το* δατ, $\alpha = 5\cdot 1$ + ε,, *χρειάστηκαν 18 εφαρμογές.*

***Συνθήκη* 1η:**

Δεδομένου ότι ο αριθμός εφαρμογών τον θεωρήματος τον Cauchy δεν μπορεί να προβλεφθεί και είναι πολύ μικρός σχετικά με την τιμή τον a_i, *στην* ***θεωρητική ανάλυση*** *της μεθόδου θα θεωρήσουμε πως για το κάτω φράγμα ισχύει* $\{\beta = \delta$ s τ.. *Αυτό δεν περιορίζει την γενικότητα διότι, όπως είδαμε, το κόστος κάθε εφαρμογής τον θεωρήματος τον Cauchy είναι πολύ μικρό.*

Αν αναλογισθούμε ότι ο σκοπός των αντικαταστάσεων είναι ***είτε*** *μία από τις θετικές ρίζες να μπει στο διάστημα (0, 1) ενώ οι υπόλοιπες να μπουν στο διάστημα* (1, ∞) ***είτε*** *μία από τις θετικές ρίζες να μπει στο διάστημα* (1, ∞) *ενώ οι υπόλοιπες να μπουν στο διάστημα (0, 1) — βλέπε και την συζήτηση πριν την απόδειξη τον θεωρήματος τον Vincent — τότε εύλογα δικαιολογείται η ερμηνεία που δώσαμε στα μερικά πηλίκα. Τα ακόλουθα λήμματα είναι σχετικά.*

Λήμμα (για την αντιστροφή πραγματικών ριζών):

Έστω p(x) μία πολυωνυμική εξίσωση μιας μεταβλητής, βαθμού δ ≥ 2, *με ακέραιους συντελεστές και χωρίς πολλαπλές ρίζες, η οποία έχει t πραγματικές ρίζες μέσα στο διάστημα* (0, 1), 2 ≤ τ ≤ δ,

Δ > 0, η ελάχιστη απόστασή τους. Τότε αν στο p(x) κάνουμε την αντιστροφή $x \leftarrow \frac{1}{x}$,*οι t ρίζες απεικονίζονται στο διάστημα* (1, ∞) , *όπου τώρα η ελάχιστη απόστασή τους είναι Δ' > Δ.*

Απόδειξη:

Έστω ότι $0 <$ 1 $< \equiv <$ [$<$ [$< \equiv <$ [$<$ [$< \equiv <$ [< 1 *είναι οι t ρίζες του p(x) μέσα στο διάστημα (0, 1), και έστω ότι* Δ = [- [, *δνώ* $\Delta' = \frac{1}{\square\square} - \frac{1}{\square\square}$. *Η απόδειξη τον λήμματος φαίνεται αμέσως από το*

Λήμμα(για την αντιστροφή μιγαδικών ριζών):

Έστω p(x) μία πολυωνυμική εξίσωση μιας μεταβλητής, βαθμού $\delta \geq 2$, *με ακέραιους συντελεστές και χωρίς πολλαπλές ρίζες, η οποία έχει δύο μιγαδικές συζυγείς ρίζες,* α_1 *και* α_2 *μέσα στον κύκλο με κέντρο* $(\frac{1}{2}, 0)$ *και ακτίνα* $\frac{1}{2}$, *και έστω επιπλέον* $\delta = \square_1 - \square_2$ *.Τότε αν στο p(x) κάνουμε την αντιστροφή* $x \leftarrow \frac{1}{x}$ *,οι 2 μιγαδικές ρίζες απεικονίζονται στο ημιεπίπεδο με πραγματικό μέρος>1, όπου τώρα η απόστασή τους είναι* $\delta' > \delta$.

Απόδειξη:

Όμοια με την προηγούμενη.//

Ακολουθεί μια λεπτομερέστερη περιγραφή της απομόνωσης πραγματικών ριζών με συνεχή κλάσματα.

Ας θεωρήσουμε ένα άπειρο δυαδικό δένδρο σε κάθε κορυφή τον οποίον αντιστοιχούμε μία τριάδα της μορφής $\{f(x), M(x), V_f\}$, *όπου το πολυώνυμο f (x) προκύπτει από το αρχικό πολυώνυμο p(x) ύστερα από την αντικατάσταση* $x \leftarrow M(x) = \frac{a\,x \square b}{c\,x \square d}$ *,, και* V_f *είναι ο αριθμός των μεταβολών πρόσημου στην ακολουθία των συντελεστών τον f(x).*

Από την προηγούμενη συζήτηση γνωρίζουμε πως αν $f(x) = p(\frac{a\,x \square b}{c\,x \square d})$,*τότε οι θετικές ρίζες του f(x) αντιστοιχούν σε* ***εκείνες*** *τις θετικές ρίζες τον p(x) που βρίσκονται μέσα στο διάστημα με άκρα* $\frac{b}{d}$ και $\frac{a}{c}$ *. Προσέξτε πως η διάταξη των* $\frac{b}{d}$, $\frac{a}{c}$ ***δεν είναι γνωστή,*** *και έτσι για ευκολία το διάστημα αυτό θα το συμβολίζουμε ως interval(a, b, c, d). Υπενθυμίζουμε πως τα άκρα αυτά προέρχονται από την έκφραση*

$\frac{a\,x \square b}{c\,x \square d}$ *αντικαθιστώντας το x με το 0 και το* ∞ *αντίστοιχα.*

Αν p(x) είναι το αρχικό πολυώνυμο με ν μεταβολές πρόσημου στην ακολουθία των συντελεστών τον, τότε στην ρίζα τον δυαδικού δένδρου αντιστοιχεί η τριάδα $\{f(x) — p(x), M(x) — x, V_f — \nu\}$.

Ο δρόμος από κάθε ***κορυφή*** *ή* ***κόμβο*** *(node) προς τον δεξιό απόγονο αντιστοιχεί στην αντικατάσταση* $x \leftarrow x+1$, *ενώ ο δρόμος προς τον αριστερό απόγονο αντιστοιχεί στην αντικατάσταση* $x \leftarrow \frac{1}{x \square 1}$ *. Προσέξτε πως για κάθε μερικό πηλίκο* a_i, *μια σειρά από* a_i *διαδοχικές αντικαταστάσεις της μορφής* $x \leftarrow x+1$ *ακολουθούμενες από την* $x \leftarrow \frac{1}{x \square 1}$ *είναι ισοδύναμες με την* $x \leftarrow a_i + \frac{1}{x}$ *ακολουθουμένη από την* $x \leftarrow x+1$.

Όλες οι κόμβοι που ανήκουν σε κάποιο ***δρόμο*** *(path), πεπερασμένο ή άπειρο, θα θεωρούνται μέλη μη τεμνομένων συνόλων τριών τύπων. Ένα σύνολο τύπον* V_0, V_1 *ή* V_n *περιέχει κόμβους*

που αντιστοιχούν σε πολυώνυμα με 0, 1 ή περισσότερες μεταβολές πρόσημου, αντίστοιχα. Τα σύνολα τύπον V_0 ή V_1 καλούνται ***τερματικά σύνολα.*** *Στην περίπτωση που σύνολα ανήκουν στον ίδιο δρόμο λέμε πως το σύνολο X* ***προηγείται*** *του συνόλου Y εάν και μόνον εάν $\forall x \in X$ και $\forall y \in Y$ το μήκος του δρόμου(x) < το μήκος τον δρόμου(y).Σε ένα τερματικό σύνολο, ο κόμβος με την μικρότερη απόσταση από την κορυφή λέγεται* ***τερματικός κόμβος.***

ΣΧΗΜΑ 7.3.1 από το βιβλίο

Σχετικά με το δυαδικό αυτό δένδρο έχουμε και την ακόλουθη υπόθεση, όπου το πολυώνυμο $f(x)$ αντιστοιχεί σε κάποιο κόμβο και τα πολυώνυμα $f(x+1)$ και $f(\frac{1}{x+1})$ *είναι οι δύο απόγονοί του. Η υπόθεση αυτή χρησιμοποιείται στην ανάλυση της χρήσης μνήμης του υπολογιστή από την μέθοδο των συνεχών κλασμάτων.*

Υπόθεση **(Strzebonski-Akritas):**

Έστω το πολυώνυμο $f(x)$ βαθμού n, $f(0) \neq 0$, με ρητούς συντελεστές και ***χωρίς πολλαπλές ρίζες*** *και έστω επιπλέον ότι με $sv(f(x))$ συμβολίζουμε τις μεταβολές πρόσημου στην ακολουθία των συντελεστών του. Τότε για τα πολυώνυμα* $f(x+1)$ και $f(\frac{1}{x+1})$ *ισχύει* ***η ανισότητα των μεταβολών πρόσημου***

$$sv(f(x+1)) + sv((x+1)^n f(\tfrac{1}{x+1})) \leq sv(f(x)).$$

Συζήτηση:

Η ισχύς της υπόθεσης αυτής έχει επιβεβαιωθεί από έναν πάρα πολύ μεγάλο αριθμό πειραμάτων. Δυστυχώς όμως η απόδειξη μας διαφεύγει. Θα μπορούσε να είναι ως εξής:

Όπως ξέρουμε από την απόδειξη των θεωρημάτων του Fourier και Cardano-Des-cartes — για την *γενική περίπτωση που επιτρέπονται πολλαπλές ρίζες (ενότητα* 6.2) — ο *αριθμός των μεταβολών πρόσημου του $f(x)$* ***είτε*** *ισούται ακριβώς με το αριθμό των θετικών ριζών του $f(x)$* ***είτε*** *το υπερβαίνει κατά κάποιο άρτιο αριθμό που οφείλεται σε κάποιες ρίζες ορισμένων παραγώγων του $f(x)$.*

Για παράδειγμα το πολυώνυμο

```
f(x_) := x - 7/3 x^2 + x^3 : Expand
```

$$-\frac{7}{3}\,x+\frac{7\,x^2}{3}-x^3$$

με μία θετική ρίζα και τρεις μεταβολές πρόσημου οφείλει τις δύο παραπάνω μεταβολές πρόσημου στην ρίζα 0.25662 της *πρώτης του παραγώγου*

```
Solve [f'[x] == 0] / / N
```

```
{{x -- 0.25662}, {x -- 1.29894}}
```

```
Solve[f''[x] == 0] / / N
```

```
{{x - 0.777778}}
```

Αυτό μαίνεται *από το γεγονός ότι κατά το "πέρασμα" από την ρίζα* 0.25662 *η ακολουθία* Fourier *χάνει δύο μεταβολές πρόσημου*

```
Fseq[x_] = createFourierSequence[ f [x]] ;
variations[Fseq[0.25]] - variations[Fseq[0.26]]
2
```

Λέμε *λοιπόν πως η ρίζα* 0.25662 της *πρώτης παραγώγου* ***συνεισφέρει*** *στο* $f(x)$ *2 μεταβολές πρόσημου. Η δεύτερη ρίζα της πρώτης παραγώγου,* 1.29894, *καθώς και η μοναδική ρίζα της δεύτερης παραγώγου,* 0.777778, δεν *συνεισφέρουν καμία μεταβολή πρόσημου*

```
variations[Fseq[1.29]] - variations[Fseq[1.30]]
```

```
0
```

```
variations[Fseq[0.77]] - variations[Fseq[0.78]]
```

```
0
```

Από το θεώρημα Fourier *λοιπόν συμπεραίνουμε πως ο αριθμός των μεταβολών πρόσημου,*

$sv(f(x))$, *αποτελείται από τις συνεισφορές των θετικών ριζών του* $f(x)$ — ***μία μεταβολή πρόσημου ανά θετική ρίζα*** — και *από τις πιθανές συνεισφορές κάποιων ριζών*

ορισμένων παραγώγων του f (x) — ***άρτιος αριθμός μεταβολών πρόσημου ανά θετική ρίζα παραγώγου.***

***Απόπειρα απόδειξης με την ακολουθία* Fourier:**

Όσον αφορά την αντικατάσταση x — x + 1, *όπως βλέπουμε από το ανάπτυγμα κατά* Taylor *του*

f(x + 1)

```
Series [f[x + 1], {x, 0 , 4}] // Normal
```

$f[1] + x f'[1] + \frac{1}{2} x^2 f''[1] + \frac{1}{6} x^3 f^{(3)}[1] + \frac{1}{24} x^4 f^{(4)}[1]$

το *πολυώνυμο f (x* + 1) θα έχει τόσες *μεταβολές πρόσημου όσες έχει και* η Fseq(l) — η *ακολουθία* Fourier του *f (x) υπολογισμένη στην τιμή x* = 1:

{ f[x], f'[x], f''[x], f'''[x], f''''[x]}/.x→1

Από το θεώρημα του Fourier *ξέρουμε όμως πως για κάθε πολυώνυμο f (x) και την αντίστοιχη ακολουθία του* Fseq(x) ισχύει *sv*(Fseq(0)) ≥ *sv*(Fseq(1))και *επομένως*
sv(f(x+1)) = *sv*(Fseq(1)) ≤ *sv*(Fseq(0)) = *sv*(f(x)).

Μέχρις *εδώ ο τρόπος αυτός πάει καλά. Για την αντικατάσταση όμως x* ← $\frac{1}{x+1}$ *από την ακολουθία* Fourier { f[x], f'[x], f''[x], f'''[x], f''''[x]}/.x→1

και το ανάπτυγμα κατά Taylor του $(x + 1)^n f(\frac{1}{x+1})$

```
Series [(x + 1)^4 f[1/(x + 1)], {x, 0, 4}] // Normal
```

$f[1] + x (4 f[1] - f'[1]) + x^2 (6 f[1] - 3 f'[1] + \frac{f''[1]}{2}) + \ldots$

$$x^3 \left(\frac{f'''}{6}\right) + x^4 \left(\frac{f''''}{24}\right) + \cdots$$

που *γράφεται και σαν*

Collect[%,{ f[1], f'[1], f''[1], f'''[1], f''''[1]}]

$$\left(1 - 4x + 6x^2 - 4x^3 + x^4\right) f(1) + \left(x - 3x^2 + 3x^3 - x^4\right) f'(1) + \cdots$$

$$\left(\frac{x^2}{2} - x^3 + \frac{x^4}{2}\right) f''(1) + \left(\frac{x^3}{6} - \frac{x^4}{6}\right) f'''(1) + \frac{1}{24} x^4 f''''(1) + \cdots$$

βλέπουμε πως δεν μπορούμε να κάνουμε καμία *πρόβλεψη για τον αριθμό των μεταβολών πρόσημου του* $(x+1)^n f\left(\frac{1}{x-1}\right)$.*Πρέπει να ξέρουμε τις τιμές των παραγώγων για* $x = 1$

Απόπειρα απόδειξης με τις ρίζες:

Για ευκολία χωρίζουμε τις θετικές ρίζες $\rho+$ *του* $f(x)$ σε *μικρότερες, ίση και μεγαλύτερες από το* 1. *Δηλαδή* $\nu = \nu_1 + 1 + \nu_1$. *Κατά τον ίδιο τρόπο χωρίζουμε και τις θετικές ρίζες των παραγώγων του* $f(x)$ που ***συνεισφέρουν*** *στις μεταβολές πρόσημου. Δηλαδή έχουμε* $\nu = \nu_1 + \nu_1 + \nu_1$.Συνεπώς, ο *αριθμός των μεταβολών πρόσημου είναι*s

$v(f(x)) = (\square\square 1 + 2\lfloor\square\square 1) + (1 + 2\kappa\ \square 1) + (\square\square 1 + 2\lfloor\square\square 1)$.

Με την *αντικατάσταση* $x \leftarrow x + 1$ οι *ρίζες του* f (x) και *εκείνες των παραγώγων του που βρίσκονται στο διάστημα* (0, 1) θα *μετακινηθούν στο διάστημα* (-1, 0), και έτσι το *πολυώνυμο* $f(x + 1)$ θα έχει sv(f (x + 1)) *μεταβολές πρόσημου, όπου*

$sv(f(x + 1)) \leq \square 1 + 2\lfloor\square\square 1$.

Από την άλλη, με την *αντικατάσταση* $x \leftarrow \frac{1}{x \square 1}$ — που *είναι ισοδύναμη με την αντικατάσταση*
$x \leftarrow \frac{1}{x}$ *ακολουθούμενη από την* $x \leftarrow x + 1$ *(βλέπε* 6.3.1) — *έρχεται η σειρά των ριζών του* $f(x)$ *και εκείνων των παραγώγων του που βρίσκονται στο διάστημα* (1, ∞) *να μετακινηθούν στο διάστημα* (-1, 0). *Έτσι το πολυώνυμο* $(x + 1)^n f(\frac{1}{x \square 1})$ θα *έχει*
$sv((x + 1)^n f(\frac{1}{x \square 1}))$*μεταβολές πρόσημου, όπου*

$sv((x + 1)^n f(\frac{1}{x \square 1})) \leq \square 1 + 2\lfloor\square\square 1$.

Στην *περίπτωση αυτή όμως μπορεί να συμβεί το εξής: μία θετική ρίζα παραγώγου που μένει στο διάστημα* (0, ∞) *μετά την αντικατάσταση* $x \leftarrow \frac{1}{x \square 1}$ *-σταματάει να συνεισφέρει μεταβολές πρόσημου στο* $(x + 1)^n f(\frac{1}{x \square 1})$και το *"έργο" αυτό "αναλαμβάνει" μια θετική ρίζα* ***άλλης*** *παραγώγου. Βλέπε για παράδειγμα το πολυώνυμο,*
$f(x) = 924x^7 - 86x^6 - 962x^5 - 715x^4 + 112x^3 + 831\ x^2 - 518x + 91$ (Strzebonski). Πως *μπορούμε να διαβεβαιώσουμε πως με την αλλαγή συνεισφοράς δεν εμφανίζονται περισσότερες μεταβολές πρόσημου;*

Από τα παραπάνω βλέπουμε πως η αντικατάσταση $x \leftarrow \frac{1}{x \square 1}$ *είναι η προβληματική περίπτωση. Αν μπορέσουμε να δείξουμε πως* $sv((x + 1)^n f(\frac{1}{x \square 1})) \leq sv(f(x)) - sv(f(x + 1))$, *τότε θα έχουμε το ζητούμενο*

$sv(f(x+1)) + sv((x+1)^{n} f(\frac{1}{x+1})) \le sv(f(x))$,

και η ισότητα θα ισχύει μόνον όταν □□ = □□1 + □□1 και □□ = 0.//.//

Ακολουθεί μία ακόμα συνθήκη πριν παρουσιάσουμε τον αλγόριθμο.

Συνθήκη 2η:

Έστω $p(x) = 0$ *μία πολυωνυμική εξίσωση βαθμού* $n > 1$, με *ρητούς συντελεστές και χωρίς πολλαπλές ρίζες που αντιστοιχεί στην κορυφή του δυαδικού δένδρου που προαναφέραμε. Έστω επιπλέον ότι η αντικατάσταση*

$$x \leftarrow a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_h + \frac{1}{x}}}},$$

όπου $a_1 \ge 0$ *τυχαίος μη αρνητικός ακέραιος και* a_2, a_3, …, a_m *τυχαίοι θετικοί ακέραιοι,*

$a_i > 0$, $1 < i \le h \le m$, *(το* m *ορίζεται από την* (1) του *θεωρήματος του* Vincent) *μετασχηματίζει την πολυωνυμική εξίσωση* $p(x) = 0$ σε *μία άλλη που αντιστοιχεί σε* V_0 *ή* V_1 *τερματικό κόμβο. Τότε για όλα τα* $1 \le i \le h$ *ισχύει*

$a_i = O$ p x □□

Συζήτηση:

Η δεύτερη αυτή συνθήκη, όπως και η 1η, μας *χρειάζεται για την* ***θεωρητική ανάλυση*** *της μεθόδου των συνεχών κλασμάτων.*

Όπως είδαμε η μέθοδος των συνεχών κλασμάτων κατά Vincent *παρουσιάζει εκθετική συμπεριφορά σχετικά με τον* ***αριθμό*** *των αντικαταστάσεων της μορφής* $x \leftarrow x + 1$ *που πρέπει να εκτελεσθούν για τον υπολογισμό κάποιου μερικού πηλίκου* a_i.

Η εκθετική συμπεριφορά της μεθόδου των συνεχών κλασμάτων εξαλείφθηκε από τον γράφοντα που υπολογίζει, με την βοήθεια του θεωρήματος του Cauchy *(ενότητα* 6.2.5), τα a_i *σαν τα κάτω φράγματα θετικών ριζών πολυωνύμων. Έτσι, στον ίδιο περίπου χρόνο που χρειάζεται να εκτελεσθεί* ***μία*** *αντικατάσταση της μορφής* $x \leftarrow x + 1$ *εκτελείται* η *αντικατάσταση* $x \leftarrow x + a_i$.

Υπάρχει όμως και το εξής πρόβλημα: *το* ***μέγεθος*** *των ακέραια*$_{a_i}$ *συντελεστών των πολυωνύμων που προκύπτουν από την αντικατάσταση* $x \leftarrow x +$ *αυξάνεται και για πάρα πολύ μεγάλα* a_i ***επιβραδύνει*** *τους υπολογισμούς*

Γεννιέται λοιπόν το ερώτημα αν υπάρχει ***ένα πάνω φράγμα στις τιμές των αι*** *για να μπορέσουμε να προβλέψουμε τον χρόνο υπολογισμού της μεθόδου.*

Διακρίνουμε δύο περιπτώσεις :

A. *Οι ρίζες του πολυωνύμου επιλέγονται από εμάς και το πολυώνυμο γράφεται ως* $p(x) = (x - \square_1) \equiv (x - \square_n)$. *Στην περίπτωση αυτή είναι προφανές πως ισχύει* $a_i = O \quad p\ x\ [\square$ *, διότι όπως ξέρουμε η σταθερά του πολυωνύμου ισούται με το γινόμενο των ριζών και* $p\ x\ \square$ *είναι ο μεγαλύτερος συντελεστής σε απόλυτη τιμή.*

B. *Οι ρίζες είναι τυχαίες. Στην περίπτωση αυτή τα πράγματα είναι λίγο πιο σύνθετα. Από την μία μεριά είναι γνωστό πως όταν αναπτύσσουμε άρρητους αριθμούς σε συνεχή κλάσματα εμφανίζονται μεγάλα μερικά πηλίκα για τα οποία* ***δεν είναι γνωστό κανένα πάνω φράγμα.*** *Για παράδειγμα, το* 432ο *μερικό πηλίκο στην ανάπτυξη του π είναι* 20776, *κάτι που δεν θα μπορούσαμε να το φανταστούμε.*

```
Last[ContinuedFraction[π, 432]]
```

```
20776
```

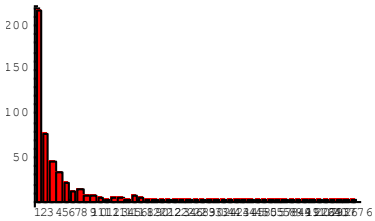
Από την άλλη μεριά έχουμε το θεώρημα των Gauss-Kuzmin που μας *λέει πως, για σχεδόν όλους τους αριθμούς, η* ***πιθανότητα*** *να είναι το* a_i, *το* i*-στό μερικό πηλίκο, ίσο με τον θετικό ακέραιο* j *είναι*

$$\log_2 \frac{j\square 1^{\,2}}{j\ j\square 2}.$$

Αυτό σημαίνει πως, για σχεδόν όλους τους αριθμούς, $a_i = 1$ *με πιθανότητα* 0.41. *Ομοίως* $a_i = 20776$ με *πιθανότητα* $3.34\text{-}10^{-9}$ ή ≈ 0.

Η στατιστική ανάλυση των μερικών πηλίκων του π συμφωνεί με το θεώρημα των Gauss-Kuzmin.

```
<<Statistics`DataManipulation`
freq=Frequencies[ContinuedFraction[π,500]];
<<Graphics`Graphics`
BarChart[freq];
```



Από τα 500 *μερικά πηλίκα τα* 216 ήταν 1, δηλαδή *ποσοστό περίπου* 43%.

Γενικά λοιπόν, το *θεώρημα των* Gauss-Kuzmin μας λέει πως δεν *μπορούμε να βρούμε ένα πάνω φράγμα στις τιμές των* a_t. ***Ειδικά όμως για την περίπτωση μας,*** *επειδή χρησιμοποιούμε πολύ λίγα μερικά πηλίκα για να απομονώσουμε τις ρίζες, το ίδιο το θεώρημα των* Gauss-Kuzmin μας λέει πως η *πιθανότητα εμφάνισης μεγάλων μερικών πηλίκων είναι ίση με μηδέν. Άρα λοιπόν η συνθήκη μας είναι δικαιολογημένη.//*

Συμπέρασμα: *Από την συζήτηση της* 2ης *συνθήκης προκύπτει ότι* ***η μέθοδος των συνεχών κλασμάτων θα είναι λίγο αργή για πάρα πολύ μεγάλα*** *μερικά πηλίκα. Το συμπέρασμα αυτό επιβεβαιώνεται και πειραματικά στην επόμενη ενότητα όπου εμείς επιλέγουμε ρίζες* — και *άρα αναγκάζουμε τα μερικά πηλίκα να είναι* — της *τάξης* 10^{300} (1000 bits)! **Σε *όλες τις άλλες περιπτώσεις είναι η μέθοδος των συνεχών κλασμάτων είναι η πιο γρήγορη στον κόσμο.***

Ακολουθεί ο αλγόριθμος για την απομόνωση των θετικών ριζών όπου αλλάζουμε λίγο την μορφή της τριάδας {*f* (χ), M(x), *Vf*} και — *δεδομένου ότι* $M(x) = \frac{a\,x + b}{c\,x + d}$, για μη *αρνητικούς ακέραιους a, b, c, και d έτσι ώστε* $ad - bc \neq 0$ — την *γράφουμε σαν* {a, b, c, d, *f*, v}. *Όπως και πριν το διάστημα με άκρα* $\frac{b}{d}$, και $\frac{a}{c}$ (η *διάταξη των οποίων* ***δεν είναι γνωστή))*** *(συμβολίζεται ως interval(a, b, c,* d). *Η κλιμάκωση των ριζών στο 4ο βήμα του αλγορίθμου οφείλεται στον* Strzebonski (1994). *Η τιμή της παραμέτρου (*b_0 *στο βήμα αυτό βρίσκεται εμπειρικά και στον αλγόριθμό μας είναι (*$b_0 = 16$.

Αλγόριθμος: Ο κλασσικός αλγόριθμος των **Vincent - Akritas - Strzebonski (1836, 1978, 1994)** ***για τις θετικές ρίζες.***

Είσοδος: $p(x) = 0$, *μία πολυωνυμική εξίσωση με ακέραιους συντελεστές, χωρίς πολλαπλές ρίζες και* $p(0) \neq 0$.

Έξοδος: *Τα διαστήματα απομόνωσης των* ***θετικών*** *ριζών του* $p(x)$ *ή οι ακριβείς θετικές ρίζες σε μορφή διαστημάτων.*

1. *Αρχικοποιούμε την λίστα των διαστημάτων απομόνωσης των ριζών rootIsolationIntervals* = {}. *Θέτουμε f* — p(x) και *υπολογίζουμε ν* — svf). *Αν ν* = 0 *επιστρέφουμε την άδεια λίστα* {}. *Αν ν* = 1 *επιστρέφουμε την λίστα* $\{(0, ub)\}$, *όπου ub είναι ένα πάνω φράγμα στις θετικές ρίζες του f που υπολογίζεται με την συνάρτηση* CauchyPositiveRootUpperBound[]. *Θέτουμε την "τριάδα"* {1, 0, 0, 1, *f*, *ν*} *στην λίστα των διαστημάτων προς εξέταση intervalsToBeProcessed.*
2. (* *επεξεργασία διαστήματος (βήματα* 3-10) *)
   Αν η λίστα intervalsToBeProcessed είναι άδεια επιστρέφουμε την λίστα rootIsolationIntervals, αλλιώς ***επαναλαμβάνουμε*** *την εξής διαδικασία: βγάζουμε την* ***πρώτη*** *"τριάδα"* $\{a, b, c, d, f, v\}$ *από την λίστα intervalsToBeProcessed.*

3. Με την *συνάρτηση* CauchyPositiveRootLowerBound[] *υπολογίζουμε ένα κάτω φράγμα (*b *στις θετικές ρίζες του* f.
4. *Αν* {β > 0 θέτουμε $f(x) \leftarrow f(\{\beta \cdot x)$, $a \leftarrow \{\beta \cdot a$, $c \leftarrow \{\beta \cdot c$, και $\{\beta \leftarrow 1$.
5. Αν $\{\beta \geq 1$ *θέτουμε* $f(x) \leftarrow f(x + \{\beta)$, $b \leftarrow \{\beta \cdot a + b$, *Υπολογίζουμε* $v \leftarrow sv(f)$. *Αν* $v = 0$ *πηγαίνουμε στο βήμα* 2. *Αν* $v = 1$ *επισυνάπτουμε το διάστημα interval(*a, b, c, d*) στην λίστα rootIsolationIntervals και πηγαίνουμε στο βήμα 2*
6. *Υπολογίζουμε* $f_1(x) \leftarrow f(x + 1)$και *θέτουμε* $a_1 \leftarrow a$, $b_1 \leftarrow a + b$, $c_1 \leftarrow c$, $d_1 \leftarrow c + d$,, *και*
   $r \leftarrow 0$. *Αν* $f_1(0) = 0$, *επισυνάπτουμε το διάστημα* $\{\frac{b_1}{d_1}, \frac{b_1}{d_1}\}$*στην λίστα rootIsolationIntervals και θέτουμε* $f_1(x) \leftarrow \frac{f_1(x}{x}$,και r — 1. *Υπολογίζουμε* $v \leftarrow sv(f)$ και θέτουμε
   $v_2 \leftarrow$ v - v_1 - r, $a_2 \leftarrow b$, $b_2 \leftarrow a + b$, $c_2 \leftarrow d$, και $d_2 \leftarrow c + d$.
7. *Αν* $v_2 > 1$, *υπολογίζουμε* $f_2(x) \leftarrow (x + 1)^m f(\frac{1}{x+1})$ *όπου είναι ο βαθμός του* f *Αν* f2(0) = 0, *θέτουμε* $f_2(x) \leftarrow \frac{f_2(x}{x}$ *και υπολογίζουμε* $v_2 \leftarrow sv(f_2)$.
8. *Αν* $v_1 < v_2$,*ανταλλάσσουμε* (swap) το$\{a_1, b_1, c_1, d_1, f_1, v_1\}$ με το $\{a_2, b_2, c_2, d_2, f_2, v_2\}$.
9. *Αν* $v_1 = 0$, *πηγαίνουμε στο βήμα* 2. *Αν* $v_1 = 1$ *επισυνάπτουμε το διάστημα interval(*a_1, b_1, c_1, d_1*) στην λίστα rootIsolationIntervals,αλλιώς επισυνάπτουμε* την *"τριάδα"*$\{a_1, b_1, c_1, d_1, f_1, v_1\}$ στην ***αρχή*** *της λίστας intervalsToBeProcessed.*

10. *Αν* $v_2 = 0$, *πηγαίνουμε στο βήμα* 2. *Αν* $v_2 = 1$ *επισυνάπτουμε το διάστημα* $interval(a_2, b_2, c_2, d_2)$ *στην λίστα* rootIsolationIntervals,*αλλιώς επισυνάπτουμε την "τριάδα"*$\{a_2, b_2, c_2, d_2, f2, v_2\}$ στην ***αρχη*** *της λίστας* intervalsToBeProcessed. *Πηγαίνουμε στο βήμα* 2.

Όπως και στην μέθοδο του Sturm, *για την απομόνωση των* ***αρνητικών ριζών*** *πρώτα εξετάζουμε αν* $p(x) = p(-x)$. *Αν ισχύει η ισότητα, αυτό σημαίνει πως οι αρνητικές ρίζες είναι συμμετρικές με τις θετικές για τις οποίες έχουμε ήδη υπολογίσει τα διαστήματα απομόνωσής τους. Άρα στην περίπτωση αυτή τα διαστήματα απομόνωσης των αρνητικών ριζών βρίσκονται στοιχειωδώς. Αν* $p(x) \neq p(-x)$ *τότε θέτουμε* $p(x) \leftarrow p(-x)$, *επαναλαμβάνουμε τον παραπάνω αλγόριθμο ακόμα μία φορά και στο τέλος απεικονίζουμε τα διαστήματα απομόνωσης των ριζών στον αρνητικό ημιάξονα.*

Όσον αφορά το 0, *εύκολα ελέγχουμε αν* $p(0) = 0$, *και στην περίπτωση αυτή θέτουμε* $p(x) = \frac{p\,x}{x}$.

Απαιτήσεις σε μνήμη:

Με την βοήθεια της υπόθεσης των Strzebonski-Akritas *μπορούμε να αποδείξουμε πως ο μέγιστος αριθμός των τριάδων της μορφής* $\{f(x), M(x), v_f\}$ *ή ισοδύναμα της μορφής* $\{a, b, c, d, f, v\}$, *που πρέπει να αποθηκευτούν στην λίστα* intervalsToBeProcessed *κατά την διάρκεια εκτέλεσης της μεθόδου των συνεχών κλασμάτων είναι το πολύ*

$1 + \log_2 n$,

όπου n *είναι ο βαθμός τον αρχικού πολυωνύμου.*

Πράγματι, αν υποθέσουμε πως ισχύει η ανισότητα των μεταβολών πρόσημου που αναφέραμε έπεται πως ο αριθμός των μεταβολών πρόσημου κάθε τριάδας αποθηκευμένης στην λίστα είναι τουλάχιστον ίσος με τον ολικό αριθμό των μεταβολών πρόσημου σε όλες τις τριάδες που προηγούνται. Επειδή ο αριθμός των μεταβολών πρόσημου στην ***πρώτη*** *τριάδα της λίστας είναι τουλάχιστον 2, και ο ολικός αριθμός των μεταβολών πρόσημου σε όλες τις τριάδες της λίστας είναι το πολύ* n, *όπου* n *είναι ο βαθμός τον αρχικού πολυωνύμου, έπεται πως ο αριθμός των τριάδων στην λίστα είναι το πολύ* $\log_2 n$. *Επομένως ο μέγιστος αριθμός μετασχηματισμένων πολυωνύμων που χρειάζεται να αποθηκεύσουμε στην λίστα είναι το πολύ 1 +* $\log_2 n$.

Ακολουθεί ο παραπάνω αλγόριθμος σε ***αναγωγική μορφή*** (recursive) *εφαρμοσμένος στο* Mathematica. *Για τον αλγόριθμο αυτό χρειάζεται η νέα συνάρτηση*

```
intrv[a_,b_] := If[a>b,{b,a},{a,b}]
```

και να *έχουν ενεργοποιηθεί οι παλαιότερες συναρτήσεις* variations[], CauchyPositiveRootUpperBound[], και CauchyPositiveRootLowerBound[].

```
VASposRootIsol[p_] := Module[
  {a, b, c, d, a1, b1, c1, d1, a2, b2, c3, d2, at, bt, ct, dt,
   intervalsToBeProcessed = {}, lb, rootIsolationIntervals = {},
   f = p, f1, f2, ft, r, ub, v, v1, v2, vt, x},

  (*  step 1*)
  x = First[Variables[f]];
  v = variations[f];
  ub = CauchyPositiveRootUpperBound[f];
  If[v == 0, Return[rootIsolationIntervals]];
  If[v == 1, Return[{{0, ub}}]];
  PrependTo[intervalsToBeProcessed, {1, 0, 0, 1, f, v}];
```

```
(*  step 2  *)
While[intervalsToBeProcessed ≠ {},
 {a, b, c, d, f, v} = First[intervalsToBeProcessed];
 intervalsToBeProcessed = Rest[intervalsToBeProcessed];

 (*  step 3  *)
 lb = CauchyPositiveRootLowerBound[f];

 (*  step 4  *)
 If[lb > 16,
  f = (f /. x → lb x) // Expand; a = lb a; c = lb c; lb = 1];

 (*  step 5  *)
 If[lb ≥ 1,
  f = (f /. x → lb + x) // Expand; b = lb a + b; d = lb c + d];
 If[(f /. x → 0) == 0, AppendTo[rootIsolationIntervals, {b/d, b/d}]];
  f = Cancel[f/x]]; v = variations[f];
 If[v == 0, Continue[];
  If[v == 1,
   AppendTo[rootIsolationIntervals, If[c ≠ 0, intrv[a/c, b/d],
     {b, b + CauchyPositiveRootUpperBound[f]}]]; Continue[]]];

 (*  step 6  *)
 f1 = (f /. x → x + 1) // Expand;
 a1 = a; b1 = a + b; c1 = c; d1 = c + d; r = 0;
 If[(f1 /. x → 0) == 0, AppendTo[rootIsolationIntervals,
   {b1/d1, b1/d1}]; f1 = Cancel[f1/x]; r = 1];
 v1 = variations[f1]; v2 = v - v1 - r; a2 = b;
 b2 = a + b; c2 = d; d2 = c + d;
```

```
(*  step 7  *)
If[v2 > 1,
 f2 = ((x + 1)^Exponent[f,x] (f /. x → 1/(x + 1))) // Expand // Simplify];
If[(f2 /. x → 0) == 0, f2 = Cancel[f2/x];
 v2 = variations[f2]];

(*  step 8  *)
If[v1 < v2, {at, bt, ct, dt, ft, vt} = {a1, b1, c1, d1, f1, v1};
 {a1, b1, c1, d1, f1, v1} = {a2, b2, c2, d2, f2, v2};
 {a2, b2, c2, d2, f2, v2} = {at, bt, ct, dt, ft, vt}];

(*  step 9  *)
If[v1 == 0, Continue[]];
If[v1 == 1,
 AppendTo[rootIsolationIntervals, If[c1 ≠ 0, intrv[a1/c1, b1/d1],
   {b1, b1 + CauchyPositiveRootUpperBound[f1]}]],
 PrependTo[intervalsToBeProcessed, {a1, b1, c1, d1, f1, v1}]];

(*  step 10  *)
If[v2 == 0, Continue[]];
If[v2 == 1,
 AppendTo[rootIsolationIntervals, If[c2 ≠ 0, intrv[a2/c2, b2/d2],
   {b2, b2 + CauchyPositiveRootUpperBound[f2]}]],
 PrependTo[intervalsToBeProcessed, {a2, b2, c2, d2, f2, v2}]]
];
Sort[rootIsolationIntervals]
]
```

Έτσι βλέπουμε πως οι ***θετικές*** *ρίζες του πολυωνύμου p(x) =* x^3 -7x + 7 *βρίσκονται στα διαστήματα απομόνωσης* (1, $\frac{3}{2}$) και ($\frac{3}{2}$, 2):

```
p [x_] = x^3 - 7 x + 7; VASposRootIsol [p [x]]
```

```
LL, 3/2 L L3/2, 2 L
```

ενώ η *μοναδική* ***αρνητική*** *βρίσκεται στο διάστημα* (-4, 0).

```
VASposRootIsol[ p[ -x]]
```

```
{{0, 4}}
```

Ανάλυση του *χρόνου υπολογισμού της μεθόδου τ:*

Έστω p(x) το πολυώνυμο βαθμού n του οποίου θέλουμε να απομονώσουμε τις ρίζες. Από την ενότητα 4.1 του πρώτου τόμου ξέρουμε πως η αντικατάσταση της μορφής $x \leftarrow a_i + x$ *εκτελείται σε χρόνο*

$$O\left(n^3 \log^2 |a_i| + n^2 \log a_i \log p(x)\right).$$

Από την (12) ξέρουμε πως για κάθε πραγματική ρίζα του p(x) ξέρουμε πως πρέπει να εκτελέσουμε το πολύ m τέτοιες αντικαταστάσεις, όπου

$$m = O\left(n \log p(x)\right).$$

Επιπλέον, από την δεύτερη συνθήκη ξέρουμε πως για κάθε ρίζα και για κάθε αντικατάσταση της μορφής $x \leftarrow a_i + x$ *ισχύει*

$$a_i = O\left(p(x)\right).$$

Συνδυάζοντας τα παραπάνω αποτελέσματα βλέπουμε πως μια ρίζα του p(x) μπορεί να απομονωθεί σε χρόνο

$$O\left(n^4 \log^3 p(x)\right).$$

Επομένως, επειδή το p(x) έχει το πολύ n πραγματικές ρίζες έπεται πως απομονώνονται σε χρόνο

$$O\left(n^5 \log^3 p(x)\right)$$

Τόσο θεωρητικά όσο mi *εμπειρικά η μέθοδος των συνεχών κλασμάτων για την απομόνωση των πραγματικών ριζών είναι η γρηγορότερη στον κόσμο*

6.4 *Σύγκριση διαφόρων μεθόδων για την απομόνωση πραγματικών ριζών με διχοτόμηση*

Στην ενότητα αυτή παρουσιάζουμε πίνακες, ***διαφόρων εποχών,*** *που συγκρίνουν τις μεθόδους: των συνεχών κλασμάτων, του* Sturm, των Collins-Akritas και μιας *τροποποίησης της τελευταίας από τους* Rouillier και Zimmermann. Το *συμπέρασμα είναι πως η μέθοδος των*

συνεχών κλασμάτων **ήταν,** και ***εξακολουθεί να είναι*** *η ταχύτερη μέθοδος απομόνωσης πραγματικών ριζών στον κόσμο!*

***Άνοιξη του* 1978:**

Οι πρώτοι τρεις πίνακες είναι από την διδακτορική διατριβή του γράφοντα. Έγιναν χρησιμοποιώντας το σύστημα **sac-1** σε *υπολογιστή* IBM S/370 Model 175 και *συγκρίνουν την μέθοδο των συνεχών κλασμάτων — χωρίς φυσικά την βελτίωση του* Strzebonski – με *εκείνη του* Sturm. *Προσέξτε πως στους πίνακες αυτούς ο βαθμός των πολυωνύμων είναι το πολύ* 20 *— βαθμός "απίστευτα" μεγάλος για την εποχή εκείνη!*

***Πολυώνυμα με ρίζες τυχαία επιλεγμένες από το διάστημα* (0, 10)**

Βαθμός πολυωνύμου	***συνεχή κλάσματα διχοτόμηση*** **V-A**	**Sturm**
5	0.71	0.73
10	23.22	22.50
15	95.35	151.42
20	288.49	> 600

Πίνακας 6.4.1. *Κάθε πολυώνυμο βαθμού* n, *(όπου* n = 5, 10, 15 και 20) στον *πίνακα αυτό σχηματίστηκε παίρνοντας το γινόμενο αντίστοιχου αριθμού γραμμικών όρων*

Πολυώνυμα με 10-ψήφιους συντελεστές τυχαία επιλεγμένους

Βαθμός πολυωνύμου	***συνεχή κλάσματα διχοτόμηση* V-A**	**Sturm**
5	0.26	2.05
10	0.46	33.28
15	0.94	156.40
20	2.36	524.42

Πίνακας 6.4.2. *Οι συντελεστές κάβε πολυωνύμου στον πίνακα αυτό είναι όλοι τους διάφοροι του μηδενός, 10-ψήφιοι* και *τυχαία επιλεγμένοι.*

Από τους δύο παραπάνω πίνακες είναι προφανές πως η μέθοδος των συνεχών κλασμάτων είναι κατά πολύ γρηγορότερη της μεθόδου του Sturm. Πως *συγκρίνεται όμως με την μέθοδο των*

Collins-Akritas — μιας *ακόμη μεθόδου διχοτόμησης* — που *είχε αναπτυχθεί μόλις δύο χρόνια νωρίτερα και που είναι και αυτή γρηγορότερη της μεθόδου του* Sturm;

Η *απάντηση την εποχή εκείνη δόθηκε έμμεσα ως εξής: Τα πολυώνυμα του Πίνακα* 6.4.2 *είναι τα ίδια με εκείνα που είχαν χρησιμοποιηθεί για να συγκριθεί η μέθοδος των* Collins-Akritas *με την μέθοδο του* Sturm. *Έτσι στον Πίνακα* 6.4.3 *συγκρίνουμε τους λόγους των χρόνων της μεθόδου των συνεχών κλασμάτων και της μεθόδου των* Collins-Akritas προς τους *αντίστοιχους χρόνους της μεθόδου του* Sturm.

***Σύγκριση της μεθόδου των συνεχών κλασμάτων με την μέθοδο των* Collins-Akritas *για τα πολυώνυμα του Πίνακα* 6.4.2**

Βαθμός πολυωνύμου	***συνεχή κλάσματα / Sturm***	**Collins-Akritas / Sturm**
5	0.13	0.28
10	0.014	0.10
15	0.004	0.05
20	0.0045	0.03

Πίνακας 6.4.3. *Σύγκριση των λόγων των χρόνων της μεθόδου των συνεχών κλασμάτων και της μεθόδου των* Collins-Akritas προς τους *αντίστοιχους χρόνους της μεθόδου του* Sturm

Αν και η *σύγκριση των δύο μεθόδων δεν ήταν* εκτενής, *από τον Πίνακα* 6.4.3 *βλέπουμε πως εμπειρικά η μέθοδος των συνεχών κλασμάτων είναι καλλίτερη της μεθόδου των* Collins-Akritas. *Δεδομένου ότι ο χρόνος υπολογισμού της τελευταία είναι* $O(n^6 \log^2 | p(x) \|_m)$, το *συμπέρασμά μας συμφωνεί πλήρως με την θεωρητική ανάλυση των μεθόδων.*

***Άνοιξη του* 2002:**

Στην *πρόσφατη εργασία τους οι* Rouillier και Zimmermann (2002), *παρουσιάζουν μία νέα μέθοδο απομόνωσης πραγματικών ριζών που είναι* ***αφ' ενός*** *μεν τόσο γρήγορη όσο και η μέθοδος των* Collins-Akritas, ***αφετέρου*** *δε η καλλίτερη όσον αφορά την χρήση μνήμης του υπολογιστή.*

Οι πίνακες που ακολουθούν συγκρίνουν την μέθοδο των συνεχών κλασμάτων (CF), *όπως τροποποιήθηκε με την βελτίωση του* Strzebonski, με την *μέθοδο* REL των Rouillier και Zimmermann. *Και οι δύο μέθοδοι έγιναν μέρος του πυρήνα του Mathematica. Για σύγκριση βρίσκονται στην ιστοσελίδα*

http://members.wolfram.com/webMathematica/Users/adams/RootIsolation.jsp

Οι δύο μέθοδοι δοκιμάστηκαν στα πολυώνυμα Chebyshev, Laguerre, Wilkinson και Mignotte, που *χρησιμοποίησαν οι* Rouillier και Zimmermann *καθώς επίσης και σε τρεις τύπους τυχαίων πολυωνύμων που χρησιμοποιήθηκαν στην διδακτορική διατριβή του γράφοντα.*

Όλοι οι υπολογισμοί έγιναν σε έναν 850 MHz Athlon PC με 256 MB RAM. *Οι πληροφορίες σχετικά με την μνήμη του υπολογιστή που χρησιμοποιήθηκε αποκτήθηκαν με την συνάρτηση* MaxMemoryUsed του *Mathematica. Στην αρχή των υπολογισμών ο πυρήνας του Mathematica καταλαμβάνει* 1.6 MB

Ειδικά Πολυώνυμα

Πολυώνυμα	***Βαθμός***	***Αρ. ριζών***	**CF** T (s)/M (MB)	**REL** T (s)/M (MB)
Chebyshev	1000	1000	2172/9.2	7368/8.5
Chebyshev	1200	1200	4851/12.8	15660/11.8
Laguerre	900	900	3790/8.7	22169/14.1
Laguerre	1000	1000	6210/10.4	34024/17.1
Wilkinson	800	800	73.4/3.24	3244/10
Wilkinson	900	900	143/3.66	5402/12.5
Wilkinson	1000	1000	256/4.1	8284/15.1
Mignotte	300	4	0.12/1.75	803/7.7
Mignotte	400	4	0.22/1.77	3422/15.8
Mignotte	600	4	0.54/1.89	26245/49.1

Πίνακας 6.4.4. *Για τα ειδικά πολυώνυμα η μέθοδος των συνεχών κλασμάτων είναι γρηγορότερη από την* REL *από* 3 *φορές* — για τα *πολυώνυμα* Chebyshev — *μέχρι περίπου* 50000 *φορές για τα πολυώνυμα* Mignotte.

Όπως αναφέραμε, η μέθοδος των συνεχών κλασμάτων απομονώνει πρώτα τις θετικές ρίζες και ύστερα τις αρνητικές. Αν φυσικά το πολυώνυμο είναι συμμετρικό απομονώνουμε ***μόνο τις θετικές*** *του ρίζες. Τα πολυώνυμα* Chebyshev *είναι συμμετρικά και έτσι εκμεταλλευόμαστε το γεγονός αυτό, κάτι που* δεν *κάνει η μέθοδος* REL

Πολυώνυμα με τυχαία παραγόμενους συντελεστές (συν/στές)

Συν/στές	***Βαθμός***	***Αρ. ριζών***	**CF**	**REL**

(αρ. bits)			T (s)/M (MB)	T (s)/M (MB)
10	500	3.6	0.78/2.2	1.66
10	1000	4.4	6.67/3.75	34.2
10	2000	5.6	215/11.4	5(
1000	500	3.2	0.56/2.28	2.19
1000	1000	3.6	12.7/5.1	31.4
1000	2000	6	329/14.2	5:

Πίνακας 6.4.5. *Γι α πολυώνυμα με τυχαία παραγόμενους συντελεστές* η *μέθοδος των συνεχών*

Στον Πίνακα 6.4.5 *κάθε αποτέλεσμα ήταν ο μέσος όρος συνόλου* 5 πολυωνύμων. *Ο αριθμός των ριζών ήταν επίσης ο μέσος. Τα ίδια τυχαία πολυώνυμα χρησιμοποιήθηκαν και γι α τις* 2 *μεθόδους.*

Πολυώνυμα με τυχαία παραγόμενους συντελεστές και μοναδιαίο κύριο συντελεστή (συν/στή)

Συν/στές (αρ. bits)	Βαθμός	**Αρ. ριζών**	**CF** T (s)/M (MB)	**REL** T (s)/M (MB
10	500	5.2	1.43/2.48	8.48/3.84
10	1000	4.8	7.12/3.74	80.7/10.1
10	2000	6.8	263/11.4	1001/37.1
1000	100	4.4	0.01/1.75	56.8/5.5
1000	200	6	0.086/1.93	252/17
1000	500	5.6	0.57/2.28	1917/96.8

Πίνακας 6.4.6. Η περίπτωση των πολυωνύμων με τυχαία παραγόμενους συντελεστές και μοναδιαίο κύριο συντελεστή αποδείχθηκε ιδιαίτερα "σκληρή" για την REL που ήταν πολύ χειρότερη από την CF.

Η *βραδύτητα της μεθόδου* REL στον *Πίνακα* 6.4.6 *δεν είναι τυχαία. Πολυώνυμα με τυχαία παραγόμενους (συντελεστές και μοναδιαίο* κύριο *συντελεστή, έχουν* ***και*** *πολύ μεγάλες* ***και*** *πολύ μικρές ρίζες με συνέπεια μία μέθοδος διχοτόμησης να αρχίζει με ένα πάρα πολύ μεγάλο διάστημα που πρέπει να διχοτομηθεί πολλές φορές προτού απομονωθούν οι μικρές ρίζες.*

Από τα παραπάνω φαίνεται πως η μέθοδός μας των συνεχών κλασμάτων είναι σχεδόν πάντα γρηγορότερη από τις μεθόδους που βασίζονται στην διχοτόμηση. Και στην πράξη, η χρήση της μνήμης του υπολογιστή

FLQ (FASTEST QUADRATIC COMPLEXITY BOUND), η ταχύτερη μέθοδος εύρεσης φράγματος τετραγωνικής πολυπλοκότητας στις τιμές των θετικών ριζών των πολυωνύμων

Alkiviadis G. Akritas, Andreas I. Argyris, Adam W. Strzebonski

Περίληψη. Σε αυτή την εργασία παρουσιάζουμε την FLQ, μια μέθοδο φράγματος τετραγωνικής πολυπλοκότητα για τις τιμές των θετικών ριζών των πολυωνύμων. Αυτό το φράγμα είναι μία επέκταση της FirstLambda, η αντίστοιχη μέθοδος φράγματος γραμμικής πολυπλοκότητας και, κατά συνέπεια, προέρχεται από το θεώρημα 3 που αναφέρεται πιο κάτω. Έχουμε εφαρμόσει την FLQ στον αλγόριθμο Vincent-Ακρίτας-Strzebonski- Συνεχή Κλάσματα (VAS-CF) για την απομόνωση των πραγματικών ριζών πολυωνύμων και συγκρίναμε τη συμπεριφοράς του με την θεωρητικά αποδεδειγμένη βέλτιστη μέθοδο, LMQ. Τα πειραματικά αποτελέσματα δείχνουν ότι ενώ η FLQ τρέχει κατά μέσο όρο ταχύτερα (ή πολύ γρηγορότερα) από ό, τι η LMQ, ωστόσο η ποιότητα των φραγμάτων που υπολογίζονται από τα δύο είναι περίπου ίδια΄ εξάλλου, αποκαλύφθηκε ότι όταν ο αλγόριθμος VAS-CF στα πολυώνυμα που ορίσαμε ως σημείο αναφοράς χρησιμοποιώντας FLQ, LMQ και min (FLQ? LMQ) και οι τρεις εκδόσεις τρέχουν εξίσου καλά και, ως εκ τούτου, είναι ασαφές ποια θα πρέπει να χρησιμοποιηθεί στον VAS-CF.

1. Εισαγωγή.

Ο υπολογισμός ενός άνω φράγματος, UB, στις τιμές των (πραγματικών) θετικών ριζών ενός πολυωνύμου p (x) είναι μια πολύ σημαντική πράξη επειδή μπορεί να χρησιμοποιηθεί για να απομονωθούν αυτές τις ρίζες- δηλαδή, να βρούμε τα διαστήματα σχετικά με τους θετικούς άξονες που το καθένα περιέχει ακριβώς μία θετική ρίζα.

Για παράδειγμα, ας υποθέσουμε ότι οι θετικές ρίζες του p (x) βρίσκονται στο ανοιχτό διάστημα] 0,UB[και ότι έχουμε μια δοκιμασία για τον καθορισμό του αριθμού των ριζών στα οποιοδήποτε διάστημα] a,b [. Στη συνέχεια, μπορούμε να απομονώσουμε αυτές τις ρίζες από επανειλημμένες υποδιαιρέσεις του διαστήματος] 0, UB[μέχρις ότου κάθε προκύπτων διάστημα να περιέχει ακριβώς μία ρίζα και κάθε πραγματική ρίζα να περιέχεται σε κάποιο διάστημα. Το φράγμα, UB, έχουν πρακτική αξία, διότι μπορούμε τώρα να δουλέψουμε με ένα ορισμένο διάστημα] 0,UB[, αντί του $]0,+\infty[$.

Προφανώς, όσο πιο ευκρινέστερο είναι το άνω φράγμα, UB, τόσο πιο αποτελεσματική είναι η μέθοδος απομόνωσης πραγματικών ριζών, διότι όλο και λιγότερες διχοτομήσεις θα πραγματοποιηθούν. Παρακαλούμε σημειώστε ότι η μέθοδος διχοτόμησης χρησιμοποιεί το ανώτερο φράγμα μόνο μια φορά και φανταστείτε την εξοικονόμηση χρόνου που σημειώνεται αν η μέθοδος απομόνωσης εξαρτάται σε μεγάλο βαθμό από επαναλαμβανόμενους υπολογισμούς των εν λόγω φραγμάτων!

Τέτοια είναι και η περίπτωση του αλγορίθμου Vincent-Ακρίτας-Strzebo nski Συνεχόμενων Κλασμάτων (VAS-CF) για την απομόνωση των θετικών ριζών των πολυωνυμικών εξισώσεων. Η μέθοδος αυτή βασίζεται στο θεώρημα Vincent του 1836, [25], η οποία αναφέρει:

Θεώρημα 1. Αν σε ένα πολυώνυμο, p (x), βαθμού n, με ρητούς συντελεστές και χωρίς πολλαπλές ρίζες εκτελούμε διαδοχικές αντικαταστάσεις του τύπου

$$x \leftarrow a_1 + \frac{1}{x},\ x \leftarrow a_2 + \frac{1}{x},\ x \leftarrow a_3 + \frac{1}{x},\ldots,$$

όπου α1 ≥ 0 είναι ένας αυθαίρετα μη αρνητικός ακέραιος και α2, α3,. . . είναι αυθαίρετα θετικοί ακέραιοι αριθμοί, αi> 0, i> 1, τότε το πολυώνυμο που προκύπτει είτε δεν παραλλαγές πρόσημου είτε έχει μια παραλλαγή. Στην τελευταία περίπτωση η εξίσωση έχει ακριβώς μία θετική ρίζα, η οποία αναπαρίσταται από το συνεχές κλάσμα

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}},$$

ενώ στην πρώτη περίπτωση δεν υπάρχουν θετικές ρίζες.

Σημειώστε ότι αν αντιπροσωπεύουμε με τον όρο $\frac{ax+b}{cx+d}$ το συνεχές κλάσμα που οδηγεί στον μετασχηματισμένο πολυώνυμο $f(x) = (cx+d)^n p\left(\frac{ax+b}{cx+d}\right)$, με μία παραλλαγή πρόσημου, τότε η μοναδική θετική ρίζα της *f (x)* -στο διάστημα $]0,\infty[$- αντιστοιχεί σε αυτή την θετική ρίζα του *p (x)* η οποία βρίσκεται στο ανοικτό διάστημα με άκρα $\frac{b}{d}$ *and* $\frac{a}{c}$. Αυτά τα σημεία τερματισμού δεν έχουν ταξινομηθεί και λαμβάνονται από τον όρο $\frac{ax+b}{cx+d}$ αντικαθιστώντας το x με 0 και ∞, αντίστοιχα.
Βλέπε την βιβλιογραφία [1], [2], το κεφάλαιο 7 του [3], καθώς και τα έγγραφα από Alesina & Galuzzi, [10] και [4] Για μια πλήρη ιστορική έρευνα του θέματος και τις λεπτομέρειες εφαρμογής, αντίστοιχα.

Ως εκ τούτου, με το θεώρημα του Vincent μπορούμε να απομονώσουμε τις (θετικές) ρίζες του δεδομένου πολυώνυμο p (x). Οι αρνητικές ρίζες απομονώνονται- όπως πρότεινε ο Sturm- αφού τους μετατρέψουν σε θετικές με την αντικατάσταση του $x \leftarrow -x$ να εκτελείται στο p (x). Η απαίτηση το ότι το p (x) δεν έχει πολλαπλές ρίζες, δεν περιορίζει την γενικότητα του θεωρήματος, διότι στην αντίθετη περίπτωση θα κάνουμε square-free factorization και στη συνέχεια θα απομονώσουμε τις ρίζες του καθενός από τους square-free παράγοντες.
Το 1978, [1], [2], διαπιστώθηκε ότι κάθε μερικό πηλίκο i είναι το ακέραιο μέρος ενός πραγματικού αριθμού- δηλαδή $a_i = \lfloor a_s \rfloor$ όπου a_s είναι ο μικρότερη θετική ρίζα κάποιου πολυώνυμο *f (x)* - και, επομένως, ότι μπορεί να υπολογιστεί ως το χαμηλότερο φράγμα, ℓb, σχετικά με τις τιμές των θετικών ριζών ενός πολυωνύμου. Έτσι, αν υποτεθεί ότι $\ell b = \lfloor a_s \rfloor$ (ιδανικά κάτω φράγμα) θέτουμε τώρα, $a_i \leftarrow \ell b, \ell b \geq 1 \cdot$, και κάνουμε την αντικατάσταση $x \leftarrow x + \ell b, \ell b \geq 1$ -οποία διαρκεί περίπου το ίδιο με την αντικατάστασης $x \leftarrow x + 1$. Αργότερα, η υπόθεση του ιδανικού κάτω φράγματος εγκαταλείφτηκε [4].

Ένα κατώτερο φράγμα,lb, για τις τιμές των θετικών ριζών ενός πολυωνύμου *f(x)*, βαθμού n, που βρίσκεται αφού πρώτα υπολογίσουμε ένα άνω φράγμα, ub, στις θετικές ρίζες του $x^n f\left(\frac{1}{x}\right)$ και θέτοντας $\ell b = \frac{1}{ub}$. Άρα αυτό που χρειάζεται είναι μία αποτελεσματική μέθοδος για τον υπολογισμό των άνω φραγμάτων στις τιμές μόνο των θετικών ριζών των πολυωνυμικών εξισώσεων .

Πρέπει να δοθεί έμφαση ότι φράγματα στις τιμές μόνο των θετικών ριζών των πολυωνύμων είναι σπάνια στην βιβλιογραφία. Το φράγμα του Cauchy στις τιμές των θετικών ριζών Ενός πολυωνύμου, χρησιμοποιούταν μέχρι προσφάτως τστην μέθοδο απομόνωσης πραγαμτικών ριζών VAS-CF [4].Στην υλοποίηση SYNAPS της μεθόδου VAS-CF, [24], οι Emiris και Tsigaridas χρησιμοποίησαν το φράγμα του Kioustelidis,[16] και ελενξαν ανεξαρτήτως των αποτελέσματα που είχαν οι Akritas και Strzebonski[4]. Παρακαλώ σημειώστε ότι και τα δύο φράγματα που προαναφέρθηκαν παραπάνω, των Cauchy και Kioustelidis, είναι γραμμικής πολυπλοκότητας.

ΠΑΡΑΘΕΜΑ

1. Με τους κατάλληλους μετασχηματισμούς $p(x) \equiv p(-x) = 0 \, and \; p(x) \equiv x^n p\left(-\frac{1}{x}\right) = 0$ κάποιος μπορεί να βρει τα $-ub$ και πάνω $\_\frac{1}{ub}$ φράγματα των αρνητικών ριζών x_ του *p(x)* αντίστοιχα,
   $-ub \leq x- \leq -\frac{1}{ub}$

2. Δείτε επίσης την δουλειά του Sharma, [20] και [21], όπου χρησιμοποίησε την χειρότερό δυνατό θετικό κατώτερο φράγμα για να αποδείξει ότι η μέθοδος VAS-CF παραμένει πολυωνυμική σε χρόνο!

Ειδικά το φράγμα του Kioustelidis εμφανίστηκε 1986, [16], αλλά πέρασε μάλλον απαρατήρητο από τον Hong, [15], όταν ανέπτυξε το πρώτο φράγμα τετραγωνικής πολυπλοκότητας των θετικών ριζών των πολυωνύμων(FLQ).

Σε πρόσφατη δουλειά, [9], [5], ένα θεώρημα από τον Stefanescu του 2005, [22], επεκτάθηκε και γενικεύτηκε με τέτοιο τρόπο έτσι ώστε όλες οι -τότε υπάρχοντες- μέθοδοι γραμμικής πολυπλοκότητας για υπολογισμό φραγμάτων στις τιμές των θετικών ριζών ενός πολυωνύμου να προκύπτουν από αυτό. Βασισμένα στο θεώρημα 3, τα FL και LM, δύο νέα φράγματα γραμμικής πολυπλοκότητας αναπτύχτηκαν΄ και χρησιμοποιώντας το ελάχιστό τους στο VAS-CF όχι μόνο επιταχύνθηκε η απομόνωση πραγματικών ριζών κατά περισσότερο από 15% - σε σύγκριση με την έκδοση του VAS-CF που υλοποιεί το φράγμα του Cauchy[7] – αλλά έγινε και πάντα ταχύτερη από την Vincent-Collins-Akritas διχοτόμησης method (VCA-Bisect), [6].

Πρόσφατα, με κίνητρο από την δουλειά του Hong', [15], αναπτύχθηκαν νέες μέθοδοι τετραγωνικής πολυπλοκότητας για τον υπολογισμό φραγμάτων στις τιμές των θετικών ριζών των πολυωνύμων. Αυτές οι μέθοδοι- όπως και αυτή που υλοποιήθηκε από τον Hong- προκύπτουν και αυτές από το θεώρημα 3 και παρουσιάζονται αλλού [7]. Έχει επιδειχτεί ότι- με εξαίρεση το FLQ- μεταξύ των μεθόδων τετραγωνικής πολυπλοκότητας , η εκτίμηση του LMQ είναι πάντα η καλύτερη.

Στην 2η ενότητα παρουσιάζουμε το θεώρημα 3 από το οποίο προκύπτουν όλες οι μέθοδοι για τον υπολογισμό φραγμάτων στις τιμές των θετικών ριζών ενός πολυωνύμων. Τότε παρουσιάζουμε τα φράγματα γραμμικής πολυπλοκότητας- | First Lambda, (FL) και Local Max, (LM)- μαζί με τα αντίστοιχα φραγματά τους τετραγωνικής- |FLQ και LMQ. Παρακαλώ σημειώστε ότι ο LMQ παρουσιάστηκε πρώτα αλλού, [7].

Στην 3η ενότητα παρουσιάζουμε τν κώδικα για τα φράγματα τετραγωνικής πολυπλοκότητας FLQ και LMQ.

Τέλος, στην 4η ενότητα συγκρίνουμε τις εκτιμήσεις των FLQ και LMQ μαζί με τον χρόνο που χρειάζεται για να τα υπολογίσουμε΄ επιπλέον, συγκρίνουμε την απόδοση τους στην VAS-CF μέθοδο απομόνωσης πραγματικών ριζών.

2. Θεωρητικό υπόβαθρο. Στην βιβλιογραφία υπάρχουν φράγματα στις απόλυτες τιμές των ριζών [13] , [17], [26], και φράγματα μόνο στις θετικές ρίζες των πολυωνύμων, [16], [18]. Παρόλο της περιορισμένης χρήσης του, η πιο πρόσφατη προσθήκη στον μεταγενέστερο τύπο των φραγμάτων έγινε από τον Stefanescu, [22]. Απέδειξε το ακόλουθο θεώρημα:

Θεώρημα 2 (Stefanescu, 2005).

Έστω το p(x) ε R[x] έτσι ώστε το πλήθος των αλλαγών πρόσημου των συντελεστών του να είναι ζυγό. Εάν

$$p(x) = c_1 x^{d_1} - b_1 x^{m_1} + c_2 x^{d_2} - b_2 x^{m_2} + \ldots + c_k x^{dk} - b_k x^{mk} + g(x),$$

Με $g(x) \in R+[x], c_i > 0, b_i > 0, d_i > m_i > d_{i+1}$ για κάθε i ο αριθμός

$$B_3(p) = \max\left\{\left(\frac{b_1}{c_1}\right)^{1/(d_1-m_1)}, \ldots, \left(\frac{b_k}{c_k}\right)^{1/(d_k-m_k)}\right\}$$

Είναι ένα άνω φράγμα για τις θετικές τιμές των ριζών του πολυωνύμου p για οποιαδήποτε επιλογή των $c_1,...,c_k$.

Το θεώρημα του Stefanescu εισάγει την ιδέα του ταιριάσματος ή ζευγαρώματος ενός θετικού συντελεστή με τον αρνητικό συντελεστή ενός όρου κατώτερης τάξης΄ πάντως το θεώρημα του Stefanescu δούλεψε μόνο για πολυώνυμα με ένα ζυγό αριθμό αλλαγών προσήμου.

Το θεώρημα του Stefanescu γενικεύτηκε με την έννοια του θεωρήματος 3 , πέρα από τις εφαρμογές σε πολυώνυμα με οπουδήποτε αριθμό αλλαγών προσήμου[9]. Για να το επιτύχουμε , εισάχθηκε η ιδέα του σπασίματος ενός θετικού συντελεστή, όπου κάθε από τα πολλά τμήματα του συντελεστή ζευγαρώνει με τους αρνητικούς συντελεστές όρων κατώτερης τάξης, [5].

Θεώρημα 3 Έστω το p(x)

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_0, \quad (a_n > 0)$$

Ένα πολυώνυμο με πραγματικούς συντελεστές και έστω d(p) και t(p) που υποδηλώνουν των βαθμό και τον αριθμό των όρων τους, αντίστοιχα.
Περισσότερο, υποθέτουμε ότι το p(x) μπορεί να γραφτεί ως

$$(4)\, p(x) = q_1(x) - q_2(x) + q_3(x) - q_4(x) + ... + q_{2m-1}(x) - q_{2m}(x) + g(x),$$

Όπου όλα τα πολυώνυμα $q_i(x), i = 1, 2, ..., 2m$ και g(x) έχουνε μόνο θετικούς.
Επιπροσθέτως, υποθέτουμε ότι για i = 1, 2, ..., m έχουμε

$$q_{2i-1}(x) = c_{2i-1,1} x^{e_{2i-1,1}} + ... + c_{2i-1,t(q_{2i-1})} x^{e_{2i-1,t(q_{2i-1})}}$$

και

$$q_{2i}(x) = b_{2i,1} x^{e_{2i,1}} + ... + b_{2i,t(q_{2i})} x^{e_{2i,t(q_{2i})}},$$

Όπου $e_{2i-1,1} = d(q_{2i-1})$ και $e_{2i,1} = d(q_{2i})$ και ο εκθέτης κάθε όρου στο $q_{2i-1}(x)$ είναι μεγαλύτερος από τον εκθέτη κάθε όρου του $q_{2i}(x)$. Εάν για όλους τους δείκτες i = 1, 2, ..., m έχουμε

$$t(q_{2i-1}) \geq t(q_{2i}),$$

Τότε ένα άνω φράγμα των τιμών των θετικών ριζών του p(x) δίνεται από

$$(5)$$

$$ub = \max_{\{i=1,2,...,m\}} \left\{ \left(\frac{b_{2i,1}}{c_{2i-1,1}} \right)^{\frac{1}{c_{2i-1,1} - e_{2i,1}}}, ..., \left(\frac{b_{2i,t(q_{2i})}}{c_{2i-1,t(q_{2i})}} \right)^{\frac{1}{c_{2i-1,t(2q_{2i})} - e_{2i,t(q_{2i})}}} \right\},$$

Για κάθε αλλαγή των θετικών συντελεστών $c_{2i-1,j},\ j = 1,2,...,t(q_{2i-1})$.
Αλλιώς, για κάθε ένα από τους δείκτες για τους οποίου έχουμε

$$t(q_{2i-1}) < t(q_{2i}),$$

Σπάμε έναν από τους συντελεστές του $q_{2i-1}(x)$ σε $t(q_{2i}) - t(q_{2i-1}) + 1$ τμήματα, έτσι ώστε τώρα $t(q_{2i}) = t(q_{2i-1})$ και να εφαρμόσουμε την ίδια φόρμουλα (5) που δίνετε παρακάτω.
Για απόδειξη αυτού του θεωρήματος βλέπε[5]. Παρακαλώ σημειώστε ότι η μερική

επέκταση του θεωρήματος 2 που παρουσιάζεται στο[9] δεν απειλεί την περίπτωση ότι $t(q_{2i-1}) < t(q_{2i})$

Κρίσιμη παρατήρηση. Το ζευγάρωμα θετικών με αρνητικούς συντελεστές και το σπάσιμο ενός θετικού συντελεστή στον απαραίτητο αριθμό τμημάτων- για να το ταιριάξουμε με τον αντίστοιχο αριθμό των αρνητικών συντελεστών- είναι οι ιδέες-κλειδιά για το θεώρημα . Γενικώς, υπάρχει ανάλογη φόρμουλα με το (5) για τις περιπτώσεις όπου: (a)ζευγαρώνουμε συντελεστές από τα μη προσκείμενα πολυώνυμα $q_{2i-1}(x)$ και $q_{2i}(x)$, για $1 \le i < i,$ και (b) σπάμε έναν η περισσότερους θετικούς συντελεστές σε περισσότερα τμήματα για να ζευγαρώσουν με τους αρνητικούς συντελεστές όρων κατώτερης τάξης.
Μεταξύ άλλων, τα ακόλουθα φράγματα γραμμικής και τετραγωνικής πολυπλοκότητας όρια στις τιμές των θετικών ριζών των πολυωνύμων προέρχονται από το θεώρημα 3.

2.1. Δύο φράγματα γραμμικής πολυπλοκότητας που προέρχονται από το θεώρημα 3.

Διάφορα φράγματα γραμμικής πολυπλοκότητας μπορούν να παρθούν από το θεώρημα 3΄αυτά που περιγράφονται παρακάτω, έχουν περιγραφεί και αλλού, [5], αλλά όχι στο πλαίσιο της πολυπλοκότητα. Τα παρουσιάζουμε εδώ ξανά, σύντομα, για συμπληρωματικότητα:

FL. "**First–**λ" : εφαρμογή του Θεωρήματος 2. Για ένα πολυώνυμο P (x), όπως στην εξίσωση (2), με λ αρνητικούς συντελεστές θα ασχοληθούμε πρώτα όλες τις περιπτώσεις για τις οποίες t(q2i) > t(q2i−1), με διάσπαση του τελευταίου συντελεστή c2i−1,t(q2i), του q2i−1(x), σε t(q2i) − t(q2i−1) + 1 *ίσα τμήματα.* Τότε ζευγαρώνουμε κάθε ένα από τους λ θετικούς συντελεστές του p (x), που αντιμετωπίσαμε καθώς κινούμασταν σε μη αυξανομένη τάξης στους εκθετών, με τον πρώτο ασύμφωνο συντελεστή

LM. "**Local-Max**" : εφαρμογή του Θεωρήματος 2. Για ένα πολυώνυμο P (x), όπως στην εξίσωση (1), ο συντελεστής $-a_k$ του όρου $-a_{k^{x^k}}$ στο P (x) – όπως δίνεται στην (1)- ζευγαρλωνει με τον συντελεστή $\frac{a_m}{2^t}$, του όρου $a_{m^{x^m}}$, όπου a_m είναι ο μεγαλύτερος θετικός συντελεστής με n ≥ m > k και t να αποτελούν το αριθμό των φορών που ο συντελεστής a_m χρησιμοποιείται.

Αυτά τα δύο φράγματα έχουν δοκιμαστεί εκτενώς- σε διάφορες κλάσεις συγκειμένων και τυχαίων πολυωνύμων- και έχει βρεθεί ότι ο συνδυασμός τους, min(FL, LM), είναι ο καλύτερος από τα φράγματα γραμμικής πολυπλοκότητας, [5]επιπρόσθετα, μία επιτάχυνση κατά 15% επιτεύχθηκε με την VAS-CF/min(FL;LM), που είναι, η μέθοδος συνεχών κλασμάτων απομόνωσης θετικών ριζών χρησιμοποιώντας το φράγμα min(FL;LM)- όταν συγκρίθηκε με την VAS-CF/Cauchy, η μέθοδος συνεχών κλασμάτων υλοποιημένη με το φράγμα του Cauchy's, [7].

2.2. Τα δύο φράγματα τετραγωνικής που προέρχονται από το Θεώρημα 3

Σε αυτήν την υποενότητα παρουσιάζουμε τα FLQ και LMQ, τις δύο υλοποιήσεις τετραγωνικής πολυπλοκότητας των FL and LM αντίστοιχα, που επίσης προέρχονται από το θεώρημα 3΄ άλλα φράγματα τετραγωνικής πολυπλοκότητας περιγράφονται αλλού [7]. Γενικώς , οι εκτιμήσεις που παίρνουμε από τα φράγματα τετραγωνικής πολυπλοκότητας είναι καλύτερες από αυτές που παίρνουμε από τις αντίστοιχες τους γραμμικής πολυπλοκότητας, καθώς υπολογίζονται με πολύ μεγαλύτερη προσπάθεια.

FLQ. First-Lambda" Υλοποίηση τετραγωνικής πολυπλοκότητας του θεωρήματος 3 .

Για ένα πολυώνυμο p(x), όπως στο (4), με αρνητικούς συντελεστές πρώτα ασχολούμαστε με όλες τις περιπτώσεις για τις οποίες $t(q_{2i}) < t(q_{2i-1})$, σπάζοντας τον τελευταίο συντελεστή $c_{2i-1,t(q_{2i})}$, του $q_{2i-1}(x)$, σε $d_{2i-1,t(q_{2i})} = t(q_{2i}) - t(q_{2i-1}) + 1$ ίσα τμήματα. Τότε κάθε αρνητικός συντελεστής $a_\mu < 0$ ζευγαρώνει με κάθε ένα από τους προηγούμενους $\min(\mu, \lambda)$ θετικούς συντελεστές a_ν όπως διαιρούνται από d_ν - που είναι, ότι κάθε από τους προηγούμενους $\min(\mu, \lambda)$ θετικούς συντελεστές a_ν σπάζει σε d_ν ίσα τμήματα, όπου d_ν αρχικοποιείται με 1 και οι τιμές τους αλλάζουνε μόνο εάν ο θετικός συντελεστής a_ν σπάει σε ίσα τμήματα όπως διαπιστώθηκε στο Θεώρημα??΄ το $u(\nu)$ που αναφέρεται στον αριθμό τον φορών a_ν μπορούν να χρησιμοποιηθούν για να υπολογιστεί το ελάχιστο, αρχικοποιείται σε d_ν και οι τιμές τους που μειώνονται κάθε φορά κατά a_ν χρησιμοποιούνται στον υπολογισμό του ελαχίστου- και το ελάχιστο κυριαρχεί έναντι όλων των ν΄ Αντίστοιχα, το μέγιστο κυριαρχεί έναντι όλων των μ

Δηλαδή έχουμε

$$ub_{FLQ} = \max_{\{a_\mu < o\}} \min_{\{a_\nu > 0 : \nu > \min(\mu,\lambda) : u(\nu) \neq 0\}} \sqrt[\nu-\mu]{-\frac{a\mu}{\frac{a_\nu}{d\nu}}}$$

LMQ. "Local-Max" Τεραγονικής πολυπλοκότητας εφαρμογή του Θεωρήματος 3. Για ένα πολυώνυμο p (x), όπως στην εξίσωση (3), κάθε αρνητικός συντελεστής $a\mu < 0$ ζευγαρώνει με κάθε μία από τους προηγούμενους θετικούς συντελεστές a_ν διαιρεμένους με $2t_\nu$, -δηλαδή, κάθε θετικός συντελεστής a_ν είναι "χωρίζεται" σε άνισα μέρη, όπως γίνεται με ακριβώς μόνο με το τοπικό μέγιστο συντελεστή στην μέθοδο φράγματος τοπικού μεγίστου΄ το t_ν αρχικά έχει οριστεί ως 1 και προστίθεται κάθε φορά με το θετικό συντελεστή a_ν που χρησιμοποιείται - και ο ελάχιστος αναλαμβάνει όλα τα ν΄ στη συνέχεια, ο ανώτατος αναλαμβάνει όλα τα μ

Δηλαδή έχουμε

$$ub_{LMQ} = \max_{\{a_\mu < o\}} \min_{\{a_\nu > 0 : \nu > \mu\}} \sqrt[\nu-\mu]{-\frac{a\mu}{\frac{a_\nu}{2^{t_\nu}}}}$$

Από τις δύο παραπάνω περιγραφές είναι ξεκάθαρο ότι το FLQ ελέγχει τους πρώτους $\min(\mu, \lambda)$ συντελεστές, ενώ το LMQ ελέγχει όλους τους προηγούμενους θετικούς συντελεστές.

Οπότε, το FLQ είναι πιο γρήγορο(ή αρκετά γρηγορότερο) από το LMQ. Επιπροσθέτως, καθώς τα άλλα φράγματα τετραγωνικής πολυπλοκότητας που περιγράφονται στο[7] δουλεύουν όπως και το φράγμα LMQ, είναι προφανές ότι το FLQ είναι το γρηγορότερο φράγμα τετραγωνικής πολυπλοκότητας.

3. Αλγοριθμική υλοποίηση των FLQ και LMQ.Σε αυτήν την ενότητα παρουσιάζουμε τον κώδικα για τα LMQ και FLQ, το δεύτερο σε δύο μέρη.

Input : A univariate polynomial $p(x) = a_k x^k + a_{k-1}x^{k-1} + \ldots + a_0, (a_k > 0)$
Output: An upper bound $tempmax$, on the values of the positive roots of the polynomial

```
1  initializations;
2  cl ⟵ {a_0, a_1, a_2, ..., a_{k-1}, a_k};
3  timesused ⟵ {1, 1, 1, ..., 1};
4  tempmax = 0;
5  if k + 1 ≤ 1 then return tempmax = 0;
6  for m ⟵ k to 1 do
7  |  if cl(m) < 0 then
8  |  |  tempmin = ∞;
9  |  |  for n ⟵ k + 1 to m + 1 do
10 |  |  |  temp = (-cl(m) / (cl(n) / 2^timesused(n)))^(1/(n-m));
11 |  |  |  timesused(n) + +;
12 |  |  |  if tempmin > temp then tempmin = temp;
13 |  |  end
14 |  |  if tempmax < tempmin then tempmax = tempmin;
15 |  end
16 end
17 return tempmax;
```

Algorithm 3.1. LMQ implementation

Input : A univariate polynomial
$p(x) = a_k x^k + a_{k-1} x^{k-1} + \ldots + a_0, (a_k > 0)$
Output: An upper bound $tempmax$, on the values of the positive roots of the polynomial

```
1  initializations;
2  cl ⟵ {a_0, a_1, a_2, ..., a_{k-1}, a_k};
3  λ ⟵ number of negative elements of cl;
4  usedVector ⟵ {0, 0, 0, ..., 0};
5  for i ⟵ 1 to k + 1 do
6  |  if cl(i) > 0 then usedVector(i) = 1;
7  end
8  if k + 1 ≤ 1 or λ = 0 then return tempmax = 0;
9  i = k + 1;
10 templamda = 0;
11 flag = 0;
12 while templamda < λ do // make sure t(q_{2i-1}) ≥ t(q_{2i}) holds for
   all i
13 |  if cl(i) > 0 then
14 |  |  if flag = 0 then posCounter + +;
15 |  |  else if flag = 1 then
16 |  |  |  if negCounter > posCounter then
17 |  |  |  |  usedVector(positionLastPositiveCoef) =
   |  |  |  |  negCounter − posCounter + 1;
18 |  |  |  end
19 |  |  |  negCounter = 0;
20 |  |  |  posCounter = 1;
21 |  |  |  flag = 0;
22 |  |  end
23 |  |  positionLastPositiveCoef = i;
24 |  else if cl(i) < 0 then
25 |  |  flag = 1;
26 |  |  negCounter + +;
27 |  |  templamda + +;
28 |  end
29 |  i − −;
30 end
31 if negCounter > posCounter then
32 |  usedVector(positionLastPositiveCoef) =
   |  negCounter − posCounter + 1;
33 end
```

Algorithm 3.2. FLQ implementation part 1

```
34 sumPosCoeff = 0;
35 i = k + 1;
   // Last of the first-λ coefficients
36 while sumPosCoeff < λ do
37 |  if usedVector(i) ≠ 0 then
38 |  |  sumPosCoeff+ = usedVector(i);
39 |  |  flPos = i;
40 |  end
41 |  i − −;
42 end
   /* If the last of the first-λ coefficients is a broken one
      (usedVector(flPos) > 1), there might be a chance that the
      sum of the positive coefficients (including broken ones) is
      more than λ.For Example:
      Let the signs of p be + + + - + + - - - + + + -
      the 5^th positive coefficient will be broken into 2 pieces
      (usedVector(8) = 2). However, the sum of the first-λ (5 non
      broken)positive coefficients is 6(incl. broken). As a
      result we are going to use the last of the positive first-λ
      coefficients timesToUse(8) − (sum − λ) = 1 time only.      */
43 timesToUse(flpos)− = (sumPosCoeff − λ);
44 denomVector ⟵ usedVector;
45 m = k;
46 tempmax = 0;
47 while λ > 0 do
48 |  if cl(m) < 0 then
49 |  |  tempmin = ∞;
50 |  |  for n = k + 1 to max(m + 1, flPos) do
51 |  |  |  if usedVector(n) > 0 then
52 |  |  |  |  tempB = (−cl(m) / (cl(n) / denomVector(n)))^(1/(n−m));
53 |  |  |  |  if tempmin > tempB then
54 |  |  |  |  |  tempmin = tempB;
55 |  |  |  |  |  tempN = n;
56 |  |  |  |  end
57 |  |  |  end
58 |  |  end
59 |  |  usedVector(tempN) − −;
60 |  |  λ − −;
61 |  |  if tempmax < tempmin then tempmax = tempmin;
62 |  end
63 |  m − −;
64 end
65 return tempmax;
```

Algorithm 3.3. FLQ implementation part 2

4. **Εμπειρικά αποτελέσματα**.

Τα πειραματικά αποτελέσματα που παρουσιάζονται σε αυτήν την ενότητα διαιρούνται σε δύο κατηγορίες : Πίνακες 1-2 και πίνακες 3-7. Στους πίνακες 1 και 2 παρουσιάζουμε τις εκτιμήσεις που υπολογίζονται από τα FLQ και LMQ για διάφορες κλάσεις συγκεκριμένων και τυχαίων/συνηθισμένων πολυωνύμων. Επιπροσθέτως, ο χρόνος που απαιτείται για κάθε εκτίμηση αναφέρεται στις παρενθέσεις. Αυτοί οι υπολογισμοί έγιναν σε έναν υπολογιστή P4 Northwood 2.4GHz @ 2.7GHz, 1GB RAM.
Τα ακόλουθα τυχαία/συνήθη πολυώνυμα χρησιμοποιήθηκαν:

- **sRand:**

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$
$$\{a_n, a_{n-1}, \ldots, a_0\} \in [-2^{20}, 2^{20}]$$

με τυχαία και διασπορά = 1001.

- **usRand**:

$$p(x) = x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$
$$\{a_{n-1}, \ldots, a_0\} \in [-2^{20}, 2^{20}]$$

με τυχαία και διασπορά = 1001.

- **pRand I:**

$$p(x) = \prod_{\text{degree}} (x - n)$$
$$n \in [-2^{10}, 2^{10}]$$

με τυχαία και διασπορά = 1001.

- **pRand II:**

$$p(x) = \prod_{\text{degree}} (x - n)$$
$$n \in [-2^{1000}, 2^{1000}]$$

με τυχαία και διασπορά = 1001.

- **Custom Poly I:**

$$p(x) = x^3 + (10^{100})x^2 - (10^{100})x - 1$$

- **Custom Poly II:**

$$p(x) = x^9 + 3x^8 + 2x^7 + x^6 - 4x^4 + x^3 - 4x^2 - 3$$

Table. 1. Bounds for positive roots of various types of polynomials. *MPR* stands for the maximum positive root, computed numerically.

		Degrees									
Polynomial	Bounds	10	100	200	300	400	500	600	700	800	900
Laguerre	LMQ	200	2×10^3	8×10^3	18×10^3	32×10^3	50×10^3	72×10^3	98×10^3	128×10^3	162×10^3
		(0.)	(0.563)	(3.562)	(11.187)	(25.594)	(49.782)	(87.344)	(142.453)	(220.766)	(329.719)
	FLQ	100	10^3	4×10^3	9×10^3	16×10^3	25×10^3	36×10^3	49×10^3	64×10^3	81×10^3
		(0.)	(0.015)	(0.031)	(0.078)	(0.109)	(0.172)	(0.25)	(0.328)	(0.406)	(0.5)
	MPR	29.92	374.98	767.82	1162.8	1558.81	1955.44	2352.5	2749.87	3147.48	3545.29
Chacyshevi	LMQ	2.23607	2.07107	10	12.2474	14.1421	15.8114	17.3205	18.7083	20	21.2132
		(0.)	(0.078)	(0.515)	(1.547)	(3.453)	(6.453)	(10.688)	(15.891)	(23.406)	(33.75)
	FLQ	1.58114	5	7.07107	8.66025	10	11.1803	12.2474	13.288	14.1421	15
		(0.)	(0.)	(0.015)	(0.047)	(0.62)	(0.11)	(0.141)	(0.172)	(0.234)	(0.281)
	MPR	0.987688	0.998877	0.999069	0.999089	0.999092	0.999095	0.999097	0.999097	0.999098	0.999099
Chacyshevii	LMQ	2.12132	7.03562	9.97497	12.227	14.1244	15.7956	17.3061	18.6949	19.9875	21.2014
		(0.015)	(0.079)	(0.531)	(1.563)	(3.453)	(6.468)	(10.719)	(16.687)	(24.656)	(34.734)
	FLQ	1.5	4.97494	7.05337	8.64581	9.98749	11.1692	12.2372	13.2103	14.1333	14.9917
		(0.)	(0.016)	(0.015)	(0.047)	(0.078)	(0.109)	(0.141)	(0.187)	(0.219)	(0.265)
	MPR	0.959403	0.999516	0.999878	0.999945	0.999969	0.99998	0.999986	0.99999	0.999992	0.999994
Wilkinson	LMQ	110	10100	40200	90300	160400	250500	360600	490700	640800	810900
		(0.)	(6.391)	(52.234)	(179.75)	(438.516)	(878.)	(1549.89)	(2508.52)	(3833.52)	(5569.42)
	FLQ	55	5050	20100	45150	80200	125250	180300	245350	320400	405450
		(0.)	(0.016)	(0.047)	(0.078)	(0.014)	(0.203)	(0.282)	(0.359)	(0.516)	(0.656)
	MPR	10	100	200	300	400	500	600	700	800	900
Mignotte	LMQ	1.77828	1.04811	1.02353	1.01557	1.01164	1.00929	1.00773	1.00662	1.00579	1.00514
		(0.)	(0.)	(0.015)	(0.)	(0.)	(0.016)	(0.)	(0.)	(0.)	(0.015)
	FLQ	1.63089	1.04073	1.01995	1.01321	1.00988	1.00789	1.00656	1.00562	1.00491	1.00437
		(0.)	(0.016)	(0.)	(0.016)	(0.15)	(0.016)	(0.016)	(0.015)	(0.016)	(0.015)
	MPR	1.5763	1.0362	1.0177	1.0117	1.0088	1.0070	1.0058	1.0050	1.0044	1.0039
xRand	LMQ	2.01011	14.3073	1.39904	3.54546	3.65744	7.75602	2.5257	1.53975	1.70317	1.65478
		(0.)	(0.219)	(3.016)	(2.11)	(3.828)	(6.)	(8.625)	(11.735)	(14.765)	(18.734)
	FLQ	2.45417	25.1062	1.04472	3.54546	2.5862	7.75602	2.03158	1.6409	1.48873	1.17328
		(0.)	(0.031)	(0.157)	(0.281)	(0.547)	(0.843)	(1.157)	(1.656)	(1.5)	(2.375)
	MPR	1.6173	8.15106	0.982276	1.1221	1.71021	1.012339	0.983633	0.983628	1.010844	0.983628
mRand	LMQ	1.57532	1916790	1.40849	1.78915	105264	1272940	1803160	12.6533	1197500	790432
		(0.)	(0.25)	(1.032)	(2.125)	(3.797)	(5.891)	(8.5)	(11.531)	(14.875)	(18.765)
	FLQ	3.16629	1916790	1.06293	15.6504	105264	1272940	901580	6.32665	598750	790432
		(0.016)	(0.047)	(0.156)	(0.266)	(0.531)	(0.828)	(1.156)	(1.641)	(2.046)	(2.328)
	MPR	0.925381	1.018871	0.982509	0.9841	1.026689	1.011621	1.331235	1.755769	1.013423	1.228396

Table. 2. cont. Bounds for positive roots of various types of polynomials. *MPR* stands for the maximum positive root, computed numerically.

		Degrees		
Polynomial	Bounds	100	200	500
pRand I	LMQ	5984.55	8818.63	14435.4
		(10.297)	(75.453)	(1053.55)
	FLQ	4231.72	5555.39	9093.75
		(0.562)	(1.453)	(21.156)
	MPR	998	998	1019
		20	50	100
pRand II	LMQ	$2.4785759004986 78 \times 10^{301}$	$4.381225845096125 \times 10^{301}$	4.62669×10^{301}
		(7.812)	(138.86)	(1359.34)
	FLQ	$1.925678288070229 \times 10^{301}$	$3.210297938962179 \times 10^{301}$	3.51814×10^{301}
		(1.844)	(12.578)	(169.594)
	MPR	0.99777×10^{301}	1.05601×10^{301}	1.05601×10^{301}
Custom Poly I	LMQ	2		
		(0.)		
	FLQ	1		
		(0.)		
	MPR	1		
Custom Poly II	LMQ	1.3218		
		(0.)		
	FLQ	1.1487		
		(0.)		
	MPR	1.06815		

Οι πίνακες 3-7 δείχνουν τον χρόνο που χρειάζεται για την VAS-CF μέθοδο για την απομόνωση των πραγματικών ριζών διαφόρων κλάσεων συγκειμένων και τυχαίων πολυωνύμων- που περιγράφονται στην επικεφαλίδα του πίνακα – όταν χρησιμοποιούν τα φράγματα FLQ, LMQ και min(FLQ,LMQ). Αυτοί οι υπολογισμοί έγιναν με Windows XP laptop computer με 1.8 Ghz Pentium M processor, και 2 GB RAM.

Table 3. Special Polynomials

Polynomial	Degree	No. of Roots	FLQ $T(s)$	LMQ $T(s)$	$FLQ + LMQ$ $T(s)$
Laguerre	100	100	0.191	0.19	0.18
Laguerre	500	500	34.69	36.192	33.589
Laguerre	1000	1000	610.047	703.772	662.152
Chebychev I	100	100	0.15	0.161	0.17
Chebychev I	500	500	29.773	27.79	26.528
Chebychev I	1000	1000	422.577	384.073	386.285
Chebychev II	100	100	0.14	0.171	0.16
Chebychev II	500	500	27.76	30.303	27.149
Chebychev II	1000	1000	408.628	381.839	382.57
Wilkinson	100	100	0.03	0.05	0.03
Wilkinson	500	500	4.717	4.777	4.817
Wilkinson	1000	1000	56.341	57.343	57.392
Mignotte	100	100	0.011	0.01	0.01
Mignotte	500	500	0.24	0.19	0.17
Mignotte	1000	1000	1.022	0.811	0.821

Table 4. Polynomial with randomly generated coefficients

Coefficients (bit length)	Degree	No. of Roots (average)	FLQ $T(s)$	LMQ $T(s)$	$FLQ + LMQ$ $T(s)$
10	500	5.2	0.5628	0.4228	0.4244
10	1000	6.4	3.7152	2.4256	2.3254
10	2000	4	41.498	20.9762	21.08
1000	500	3.6	0.4384	0.2428	0.2442
1000	1000	5.2	2.8442	1.566	1.5804
1000	2000	4.8	20.161	10.6712	10.291

FLQ, the Fastest Quadratic Complexity Bound...

Table 5. Monic polynomials with randomly generated coefficients

Coefficients (bit length)	Degree	No. of Roots (average)	FLQ $T(s)$	LMQ $T(s)$	$FLQ + LMQ$ $T(s)$
10	500	5.6	0.4586	0.3788	0.3864
10	1000	6.8	4.0158	2.736	2.7358
10	2000	7.2	60.453	25.0124	23.6416
1000	500	5.2	0.6028	0.4166	0.4146
1000	1000	5.2	1.943	1.3658	1.38
1000	2000	5.6	20.6296	13.7298	13.2852

Table 6. Products of factors ($x^{20}-$randomly generated integer root)

Coefficients (bit length)	Degree	No. of Roots	FLQ $T(s)$	LMQ $T(s)$	$FLQ + LMQ$ $T(s)$
20	500	50	4.178	4.6408	3.8814
20	700	70	14.4728	13.5738	14.1442
20	1000	100	60.251	52.5754	51.6798
1000	300	30	7.05	5.5642	6.371
1000	400	40	18.933	15.9492	16.65
1000	500	50	47.5842	37.6782	41.3212

Table 7. Products of factors ($x-$randomly generated integer root)

Coefficients (bit length)	Degree	No. of Roots	FLQ $T(s)$	LMQ $T(s)$	$FLQ + LMQ$ $T(s)$
10	100	100	0.2682	0.3528	0.276
10	200	200	1.3278	1.2878	1.2918
10	500	500	19.7542	21.3006	19.2678
1000	20	20	0.032	0.03	0.028
1000	50	50	0.8732	1.0298	0.8552
1000	100	100	14.529	17.321	14.1182

5. **Συμπεράσματα**

Από τα δεδομένα που παρουσιαστήκαν στην προηγούμενη ενότητα γίνεται προφανές ότι η ποιότητα των εκτιμήσεων και του FLQ και του LMQ είναι περίπου ίδια αλλά το FLQ τρέχει γρηγορότερα (ή αρκετά πιο γρήγορα) από το LMQ. Όμως όταν τα φράγματα FLQ, LMQ και min(FLQ,LMQ) υλοποιούνται στην μέθοδο VAS-CF απομόνωσης πραγματικών ριζών δεν μπορούμε να συμπεράνουμε ποιο από τα δύο πρέπει να χρησιμοποιηθεί.

Εκτεταμένοι έλεγχοι των διάφορων υλοποιήσεων της VAS-CF γραμμικής και τετραγωνικής πολυπλοκότητας αποκάλυψαν ότι το V AS/ CF=LMQ είναι γρηγορότερο για όλες τις κλάσεις των πολυωνύμων , εκτός όταν υπάρχουν πολύ μεγάλες ρίζες΄ σε αυτήν την περίπτωση το V AS/ CF=min(FL,LM) είναι γρηγορότερο κατά μία πολύ μικρή διαφορά΄ στην πραγματικότητα, επιτεύχθηκε μία επιτάχυνση κατά 40% όταν το was attained when VAS/CF=LMQ συγκρίθηκε με το V AS/CF=Cauchy, που είναι η αρχική υλοποίηση, [8].

Επιπροσθέτως, όπου δείχνουμε αλλού, , [6], το V AS/ CF=min(FL,LM) είναι πάντα γρηγορότερο από την μέθοδο διχοτόμησης Vincent-Collins-Akritas (VCA-bisec)5, [12], ή από οπουδήποτε παραλλαγή της, [19]. Για αυτό, τα τρέχοντα αποτελέσματα μας, μειώνουν το κενό μεταξύ των VAS-CF και VCA-bisec.

ΑΝΑΦΟΡΕΣ

[1] Akritas A. G. Vincent's Theorem in Algebraic Manipulation. Ph.D. Thesis, Department of Operations Research, North Carolina State University, Raleigh, NC, 1978.

[2] Akritas A. G. The fastest exact algorithms for the isolation of the real roots of a polynomial equation. Computing, 24 (1980), 299{313.
[3] Akritas A. G. Elements of Computer Algebra with Applications. John Wiley Interscience, New York, 1989.

[4] Akritas A. G., A. Strzebonski. A comparative study of two real root isolation methods. Nonlinear Analysis: Modelling and Control, 10, No 4 (2005), 297{304.

[5] Akritas A. G., A. Strzebonski, P. Vigklas. Implementations of a New Theorem for Computing Bounds for Positive Roots of Polynomials. Computing, 78, (2006), 355{367.

[6] Akritas A. G., A. Strzebonski, P. Vigklas. Advances on the Continued Fractions Method Using Better Estimations of Positive Root Bounds. In: Proceedings of the 10th International Workshop on Computer Algebra in Scienti_c Computing, CASC'2007 (Eds V. G. Ganzha, E. W. Mayr, E. V. Vorozhtsov), Bonn, Germany, September 16-20, 2007. LNCS 4770, Springer Verlag, Berlin, Heidelberg, 24{30.

[7] Akritas A. G., A. Strzebonski, P. Vigklas. `Quadratic Complexity Bounds on the Values of Positive Roots of Polynomials (submitted).

[8] Akritas A. G., A. Strzebonski, P. Vigklas. Improving the Performance of the Continued Fractions Method Using New Bounds of Positive Roots (submitted).

[9] Akritas A. G., P. Vigklas. A Comparison of Various Methods for Computing Bounds for Positive Roots of Polynomials. Journal of Universal Computer Science, 13, No 4 (2007), 455{467
[10] Alesina A., M. Galuzzi. A new proof of Vincent's theorem. L'Enseignement Mathemathique, 44, (1998), 219{256.

[11] Boulier F. Syst_emes polynomiaux : que signi_e \r_esoudre"?. Universit_e Lille 1, UE INFO 251, 2007.

[12] Collins,G. E., A. G. Akritas. Polynomial real root isolation using Descartes' rule of signs. In: Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computations, Yorktown Heights, N.Y., 1976, 272{ 275.

[13] Demidovich P., A. Maron. Computational Mathematics. Mir Publishers, Moscow, 1987.

[14] von zur Gathen J., J. Gerhard. Fast Algorithms for Taylor Shifts and Certain Di_erence Equations. In: Proceedings of ISSAC'97, Maui, Hawaii, U.S.A., 1997, 40{47.

[15] Hong H. Bounds for absolute positiveness of multivariate polynomials. J. Symb. Comput., 25, No 5 (1998), 571{585.

[16] Kioustelidis B. Bounds for positive roots of polynomials. J. Comput. Appl. Math., 16, No 2 (1986), 241{244.

[17] Mignotte M. Mathematics for Computer Algebra. Springer-Verlag, New York, 1992.

[18] Obreschkoff N. Verteilung und Berechnung der Nullstellen reeller Polynome. VEB Deutscher Verlag der Wissenschaften, Berlin, 1963.

[19] Rouillier F., P. Zimmermann. E_cient isolation of polynomial's real roots. Journal of Computational and Applied Mathematics, 162, (2004), 33{ 50.

[20] Sharma V. Complexity of Real Root Isolation Using Continued Fractions. In: ISAAC07, preprint, 2007.

[21] Sharma V. Complexity Analysis of Algorithms in Algebraic Computation. Ph.D. Thesis, Department of Computer Sciences, Courant Institute of Mathematical Sciences, New York University, 2007.

[22] S_tef_anescu D. New bounds for positive roots of polynomials. Journal of Universal Computer Science, 11, No 12 (2005), 2132{2141.

[23] S_tef_anescu D. Bounds for Real Roots and Applications to Orthogonal Polynomials. In: Proceedings of the 10th International Workshop on Computer Algebra in Scienti_c Computing, CASC 2007 (Eds V. G. Ganzha, E. W. Mayr, E. V. Vorozhtsov), Bonn, Germany, September 16{20, 2007, LNCS 4770, Springer Verlag, Berlin, Heidelberg, 377{391.

[24] Tsigaridas E. P., I. Z. Emiris. Univariate polynomial real root isolation: Continued fractions revisited. In: ESA 2006 (Eds Y. Azar and T. Erlebach), LNCS 4168, 2006, 817{828.

[25] Vincent A. J. H. Sur la resolution des _equations numeriques. Journal de Mathematiques Pures et Appliquees, 1, (1836), 341{372.

[26] Yap C-K. Fundamental problems of Algorithmic Algebra. Oxford University Press, 2000.

Βελτίωση της απόδοσης της μεθόδου των συνεχών κλασμάτων χρησιμοποιώντας νέα φράγματα για θετικές ρίζες

A.G. Akritas, A.W. Strzebonski, P. S. Vigklas

Abstract: Στην εργασία αυτή συγκρίνουμε τέσσερις εφαρμογές της μεθόδου συνεχών κλασμάτων των Vincent-Akritas-Strzebonski για απομόνωση πραγματικών ριζών χρησιμοποιώντας τέσσερα διαφορετικά φράγματα(δύο γραμμικής και δύο τετραγωνικής πολυπλοκότητας) για τις τιμές των θετικών ριζών των πολυωνύμων. Τα όριο τετραγωνικής πολυπλοκότητας χρησιμοποιήθηκαν για να δούμε εάν η ποιότητα των εκτιμήσεων τους ανταποκρίνεται στην τετραγωνική πολυπλοκότητα τους. Πράγματι ο πειραματισμός με τις διάφορες κατηγορίες ειδικών και τυχαίων πολυωνύμων αποκάλυψε ότι η μέθοδος VAS συνεχών κλασμάτων χρησιμοποιώντας LMQ, η τετραγωνικής πολυπλοκότητας παραλλαγή του τοπικού άνω φράγματος, πέτυχε συνολική μέγιστη επιτάχυνση της τάξης του 40%, σε σχέση με την αρχική εφαρμογή χρησιμοποιώντας το γραμμικό φράγμα της μεθόδου Cauchy.

Λέξεις-κλειδιά: θεώρημα του Vincent, πολυωνυμική απομόνωση πραγματικών ριζών, μέθοδος συνεχών κλασμάτων, άνω φράγματα για τις θετικές ρίζες, φράγματα γραμμικής και τετραγωνικής πολυπλοκότητας.

1 Εισαγωγή

Ξεκινάμε πρώτα αναθεωρώντας μερικά βασικά γεγονότα της μεθόδου VAS-ΣΚ για την απομόνωση θετικών ριζών των πολυωνύμων. Η μέθοδος αυτή βασίζεται στο θεώρημα Vincent του 1836, το οποίο αναφέρει:

Θεώρημα 1. Αν σε ένα πολυώνυμο, p (x), βαθμού n, με ρητούς συντελεστές και χωρίς πολλαπλές ρίζες εκτελούμε διαδοχικές αντικαταστάσεις του τύπου

$$x \leftarrow a_1 + \frac{1}{x}, x \leftarrow a_2 + \frac{1}{x}, a_3 + \frac{1}{x}, \ldots,$$

όπου α1 ≥ 0 είναι ένας αυθαίρετα μη αρνητικός ακέραιος και α2, α3,. . . είναι αυθαίρετα θετικοί ακέραιοι αριθμοί, αi> 0, i> 1, τότε το πολυώνυμο που προκύπτει είτε δεν παραλλαγές πρόσημου είτε έχει μια παραλλαγή. Στην τελευταία περίπτωση η εξίσωση έχει ακριβώς μία θετική ρίζα, η οποία αναπαρίσταται από το συνεχές κλάσμα

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}},$$

ενώ στην πρώτη περίπτωση δεν υπάρχουν θετικές ρίζες.

Δείτε τα έγγραφα των Alesina & Galuzzi, [2] και το κεφάλαιο 7 του [3] Για μια πλήρη ιστορική έρευνα του θέματος και των λεπτομερειών της εφαρμογής, αντίστοιχα. Το πράγμα που πρέπει να σημειώσουμε είναι ότι οι ποσότητες αi (τα μερική πηλίκα των συνεχών κλασμάτων) υπολογίζονται από την επανειλημμένη εφαρμογή μιας μεθόδου των κάτω φραγμάτων για τις τιμές των θετικών ριζών ενός πολυωνύμου.

Το φράγμα του Cauchy (γραμμικής πολυπλοκότητας) στις τιμές των θετικών ριζών, χρησιμοποιήθηκε μέχρι πρόσφατα στην μέθοδο VAS-ΣΚ απομόνωσης πραγματικών ριζών, [4]. Στην SYNAPS εφαρμογή της μεθόδου VAS-ΣΚ, [5], Οι Εμίρης και Τσιγαρίδας χρησιμοποίησαν το φράγμα του Kioustelidis » (γραμμικής πολυπλοκότητας) , [6] και επαλήθευσαν ανεξάρτητα τα αποτελέσματα που επιτεύχθηκαν από τους Akritas-Strzebonski, [4] 2.

Στην εργασία αυτή εφαρμόζουμε την μέθοδο VAS-ΣΚ απομόνωσης πραγματικών ριζών χρησιμοποιώντας συγχρόνως φράγματα γραμμικής και τετραγωνικής πολυπλοκότητας στις τιμές των θετικών ριζών ενός πολυωνύμων - τα οποία όλα βασίζονται στο Θεώρημα 2 που ακολουθεί, [9], [10], [11] - και ανακαλύψαμε ότι η τελευταία βελτιώσει σημαντικά τις επιδόσεις του!

Η υπόλοιπη εργασία είναι δομημένη ως εξής:

Στο κεφάλαιο 2 παρουσιάζεται ο αλγόριθμος VAS-ΣΚ όπως διαπιστώθηκε στη [4] ΄ αυτό γίνεται τόσο για πληρότητα του παρόντος εγγράφου όσο και να διορθώσουμε ένα τυπογραφικό λάθος που

εμφανίστηκε στο βήμα 5 [4].

Παρουσιάζουμε επίσης το θεωρητικό υπόβαθρο του νέου φράγματος τετραγωνικής πολυπλοκότητας. Στην παράγραφο 3, συγκρίνουμε τέσσερις διαφορετικές υλοποιήσεις του αλγορίθμου VAS-ΣΚ χρησιμοποιώντας δύο φράγματα γραμμικής και δύο τετραγωνικής πολυπλοκότητας- όλα κατάλληλα προσαρμοσμένα για τον υπολογισμό κάτω φράγματα για τις τιμές των θετικών ριζών.

Τέλος, στο κεφάλαιο 4 παρουσιάζουμε τα συμπεράσματά μας ΄ δηλαδή όταν το LMQ, η τετραγωνικής πολυπλοκότητας παραλλαγή του μέγιστου τοπικού μας ορίου, υλοποιείται στην μέθοδο VAS-ΣΚ, επιτυγχάνουμε μία μέση επιτάχυνση της τάξης του 40% σε σχέση με την αρχική εκδοχή του VAS-ΣΚ, όπου χρησιμοποιούμε το φράγμα Cauchy (γραμμικής πολυπλοκότητας).

2 Αλγοριθμικό και θεωρητικό υπόβαθρο

Στην Ενότητα 2.1 παρουσιάζουμε τον αλγόριθμο VAS-ΣΚ - όπως ορίστηκε στο [4] - και να διορθώνουμε το τυπογραφικό λάθος στο Βήμα 5 που είχε εμφανιστεί στην εν λόγω παρουσίαση ΄ στην συνέχεια, θα εξηγήσουμε πως χρησιμοποιείται το νέο φράγμα στις θετικές ρίζες. Επίσης παρουσιάζουμε το Θεώρημα 2 από τα οποία
προέρχονται όλα τα φράγματα για τις τιμές των θετικών ριζών. Τα φράγματα γραμμικής πολυπλοκότητας που παρουσιάζονται στην Ενότητα 2.2 ενώ τα φράγματα τετραγωνικής πολυπλοκότητας παρουσιάζονται στην Ενότητα 2.3.

Παράθεμα

1) Ένα κατώτερο όριο, ℓβ, στις τιμές των θετικών ρίζες ενός πολυωνύμου f (x), βαθμού n, βρίσκεται αφού πρώτα υπολογίζουμε ένα άνω φράγμα, UB, στις τιμές των θετικών ριζών $x^n f\left(\frac{1}{x}\right)$ και στη συνέχεια θέτουμε ℓ b = 1/UB.
2) Δείτε επίσης το Sharma, [7] και [8], όπου χρησιμοποιείται το χειρότερο δυνατό θετικό κάτω φράγμα για να αποδείξει ότι η μέθοδος VAS-ΣΚ είναι ακόμα πολυωνυμική στο χρόνο!

2.1 Περιγραφή του αλγορίθμου VAS -Συνεχών Κλασμάτων, VAS-ΣΚ

Χρησιμοποιώντας το συμβολισμό του άρθρου [4], έστω η f ∈ Z[x] \ {0}.. Από το sgc(f), παίρνουμε τον αριθμό των αλλαγών πρόσημου στην ακολουθία των μη μηδενικών συντελεστών της f. Για μη αρνητικούς ακεραίους a, b, c, και d, τέτοια ώστε ad − bc 6= 0, βάζουμε

$$\text{int}\, ru(a,b,c,d) := \Phi_{a,b,c,d}((0,\infty)),$$

Όπου

$$\Phi_{a,b,c,d} : (0,\infty) \ni x \to \frac{ax+b}{cx+d} \in \left(\min\left(\frac{a}{c},\frac{b}{d}\right), \max\left(\frac{a}{c},\frac{b}{d}\right)\right),$$

Και από τα δεδομένα του διαστήματος δημιουργούμε μία λίστα {a, b, c, d, p, s}, όπου p είναι ένα πολυώνυμο τέτοιο ώστε οι ρίζες της f στο intrv(a, b, c, d) είναι οι εικόνες των θετικών ριζών του p μέσω των $\Phi_{a,b,c,d}$ και s = sgc(p).

Η τιμή της παραμέτρου α0 που χρησιμοποιείται στο βήμα 4 παρακάτω, πρέπει να επιλέγεται εμπειρικά. Στην εφαρμογή μας α0= 16.

Αλγορίθμου VAS -Συνεχών Κλασμάτων, VAS-ΣΚ

Είσοδος: Ένα squarefree πολυώνυμο f ∈ Z[x] \ {0}

Έξοδος: Ο λίστα rootlist των διαστημάτων απομόνωσης των θετικών ρίζών της f

1) Όρισε την rootlist ως άδεια λίστα. Υπολόγισε το s ← sgc(f). Εάν s = 0 επέτρεψε μία άδεια λίστα. Εάν s = 1 επέστρεψε {(0, ∞)}. Τοποθέτησε τα δεδομένα διαστήματος(interval data) {1, 0, 0, 1, f, s} στην intervalstack.
2) Εάν η intervalstack είναι άδεια, επέστρεψε την rootlist, αλλιώς πάρε τα δεδομένα διαστήματος {a, b, c, d, p, s}από την intervalstack.
3) Υπολόγισε ένα κάτω φράγμα $\alpha \in Z$ στις θετικές ρίζες της p.
4) Εάν $\alpha > \alpha_0$ θέσε όπου p(x) ← p(α x), a ← α a, c ← α c, και $\alpha \leftarrow 1$
5) Εάν $\alpha \geq 1$, θέσε όπου p(x) ← p(x + α), b ← α a + b, και d ← α c + d. Εάν p(0) = 0, πρόσθεσε [b/d, b/d] στην rootlist, και θέσε όπου p(x) ← p(x)/x. Υπολογισε το s ← sgc(p). Εάν s = 0 πήγαινε στο βήμα 2. Εάν s = 1 πρόσθεσε το intrv(a, b, c, d) στο rootlist και πήγαινε στο βήμα 2.
6) Υπολόγισε το $p_1(x) \leftarrow p(x+1)$, και θέσε $a_1 \leftarrow a$, $b_1 \leftarrow a+b$, $c_1 \leftarrow c$, $d_1 \leftarrow c+d$, και $r \leftarrow 0$. Εάν $p_1(0) = 0$, πρόσθεσε $[b_1/d_1, b_1/d_1]$ στην rootlist, και θέσε όπου $p_1(x) \leftarrow p_1(x)/x$, και $r \leftarrow 1$. Υπολόγισε το $s_1 \leftarrow sgc(p_1)$, και θέσε όπου $s_2 \leftarrow s - s_1 - r$, $a_2 \leftarrow b$, $b_2 \leftarrow a+b$, $c_2 \leftarrow d$, and $d_2 \leftarrow c+d$.
7) Εάν $s_2 > 1$, θέσε όπου $p_2(x) \leftarrow (x+1)^m p\left(\frac{1}{x+1}\right)$,όπου m είναι ο βαθμός του πολυωνύμου p. Εάν $p_2(0) = 0$, θέσε όπου $p_2(x) \leftarrow p_2(x)/x$. Υπολόγισε το $s_2 \leftarrow sgc(p_2)$.
8) Εάν $s_1 < s_2$, κάνε swap τα $\{a_1, b_1, c_1, d_1, p_1, s_1\}$ με τα $\{a_2, b_2, c_2, d_2, p_2, s_2\}$.
9) Εάν $s_1 = 0$ πήγαινε στο βήμα 2. Εάν $s_1 = 1$ πρόσθεσε intrv(a_1, b_1, c_1, d_1) στην rootlist, αλλιώς βάλε τα δεδομένα διαστήματος(interval data) $\{a_1, b_1, c_1, d_1, p_1, s_1\}$στην intervalstack.
10) Εάν $s_2 = 0$ πήγαινε στο βήμα 2. Εάν $s_2 = 1$ πρόσθεσε τα intrv(a_2, b_2, c_2, d_2) στην rootlist, αλλιώς βάλε τα δεδομένα διαστήματος(interval data) $\{a_2, b_2, c_2, d_2, p_2, s_2\}$ στην intervalstack. Πήγαινε στο βήμα 2.

Παρακαλώ σημειώστε ότι το κατώτερο φράγμα α, στις θετικές ρίζες του p (x) υπολογίζεται στο Βήμα 3, και χρησιμοποιείται στο Βήμα 5.

Όπως αναφέρθηκε στην εισαγωγή, το φράγμα του Cauchy ήταν το πρώτο που χρησιμοποιήθηκε στο VAS-ΣΚ. Ωστόσο, τα νέα φράγματα για τις τιμές των θετικών ρίζών αναπτύχθηκαν από εμάς πρόσφατα΄ για λεπτομέρειες, βλ. [9], [10] και [11]. Το σημαντικό πράγμα που προέκυψε από τις εργασίες μας είναι ότι όλα τα φράγματα για τις τιμές των θετικών ρίζες προέρχονται από τα ακόλουθα [11]:

Θεώρημα 2.

Έστω p(x)

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0 \quad (a_n > 0) \qquad (1)$$

είναι ένα πολυώνυμο με πραγματικούς συντελεστές και έστω d(p) και t(p), υποδηλώνουν το βαθμό και τον αριθμό των όρων του, αντίστοιχα. Επιπλέον, υποθέτουμε ότι p (x) μπορεί να γραφεί ως

$$p(x) = q_1(x) - q_2(x) + q_3(x) - q_4(x) + \ldots + q_{2m-1}(x) - q_{2m}(x) + g(x), \qquad (2)$$

Όπου όλα τα πολυώνυμα $q_i(x)$, i = 1, 2, . . . , 2m και το g (x) έχουν μόνο θετικές συντελεστές. Επιπλέον, ας υποθέσουμε ότι για i = 1, 2,. . . , m έχουμε

$$q_{2i-1}(x) = c_{2i-1,1}x^{e_{2i-1,1}} + \ldots + c_{2i-1,t(q_{2i-1})}x^{e_{2i-1,t(q_{2i-1})}}$$

και

$$q_{2i}(x) = b_{2i,1}x^{e_{2i,1}} + \ldots + b_{2i,t(q_{2i})}x^{e_{2i,t(q_{2i})}},$$

Όπου $e_{2i-1,1} = d(q_{2i-1})$ και $e_{2i,1} = d(q_{2i})$ και ο εκθέτης κάθε όρου στο $q_{2i-1}(x)$ είναι μεγαλύτερο από τον εκθέτη κάθε όρου στο $q_{2i}(x)$.
Αν για όλους τους δείκτες i = 1, 2,. . . , m, έχουμε
$t(q_{2i-1}) \geq t(q_{2i})$
τότε, ένα άνω φράγμα των τιμών των θετικών ριζών του p (x) δίνεται από

$$ub = \max_{\{i=1,2,\ldots,m\}} \left\{ \left(\frac{b_{2i-1}}{c_{2i-1}} \right)^{\frac{1}{e_{2i-1,1}-e_{2i-1}}}, \ldots, \left(\frac{b_{2i,t(q_{2i})}}{c_{2i-1,t(q_{2i})}} \right)^{\frac{1}{e_{2i-1,t(q_{2i})}-e_{2i,t(q_{2i})}}} \right\}, \quad (3)$$

για οποιαδήποτε μετάθεση των θετικών συντελεστών $c_{2i-1,j},\ j = 1,2,\ldots,t(q_{2i-1}).$
Σε αντίθετη περίπτωση, για κάθε ένα από τα i δεικτών για τα οποία έχουμε
$t(q_{2i-1}) < t(q_{2i}),$
έχουμε χωρίσει έναν από τους συντελεστές του $q_{2i-1}(x)$ σε $t(q_{2i}) - t(q_{2i-1}) + 1$ τμήματα, έτσι ώστε τώρα $t(q_{2i}) = t(q_{2i-1})$ και έτσι ώστε να μπορούμε να εφαρμόσουμε τον ίδιο τύπο (3) που δίνεται ανωτέρω.

Για μια απόδειξη αυτής της οπτικής της θεωρίας [11]. Μεταξύ άλλων, τα εξής φράγματα γραμμικής και τετραγωνικής πολυπλοκότητας για τις αξίες των θετικών ριζών των πολυωνύμων προέρχονται από αυτή.

2.2 Φράγματα γραμμικής πολυπλοκότητας που προέρχονται από το θεώρημα 2

Διάφορα φράγματα γραμμικής πολυπλοκότητας μπορούν να ληφθούν από το παραπάνω θεώρημα΄ αυτά περιγράφονται παρακάτω έχουν παρουσιαστεί αλλού, [11], αλλά όχι στο πλαίσιο της πολυπλοκότητας.
Εμείς τα παρουσιάζουμε εδώ και πάλι, σύντομα, για λόγους πληρότητας:

C. Cauchy

Η εφαρμογή του θεωρήματος 2 από την μέθοδο του Cauchy : "Οδηγός- συντελεστή".

Για ένα πολυώνυμο p (x), όπως στην εξίσωση (1), με λ αρνητικούς συντελεστές, η μέθοδος του Cauchy πρώτα σπάει τον οδηγό- συντελεστή, αn, σε λ ίσα μέρη και στη συνέχεια ζευγαρώνει κάθε μέρος με τον πρώτο ασύμφωνο συντελεστή, [12].

Δηλαδή, έχουμε:

$$ub_C = \max_{\{1 \leq k \leq n : a_{n-k} < 0\}} \sqrt[k]{-\frac{\lambda a_{n-k}}{a_n}}$$

Ή ισοδύναμα

$$ub_C = \max_{\{1\le k\le n: a_{n-k}<0\}} \sqrt[k]{-\frac{a_{n-k}}{\frac{a_n}{\lambda}}}$$

K. Kioustelidis

Η εφαρμογή του θεωρήματος 2 από την μέθοδο του K. Kioustelidis: "Οδηγός- συντελεστή".

Για ένα πολυώνυμο p (x), όπως στην εξίσωση (1), η μέθοδος του Kioustelidis ταιριάζει τον συντελεστή $-a_{n-k}$ του όρου $-a_{n-k}x^{n-k}$ στο p(x) με $\frac{a_n}{2^k}$ τον μεγαλύτερος συντελεστής διαιρεμένο δια το 2^k .

Η εφαρμογή του θεωρήματος 2 από την μέθοδο του K. Kioustelidis: "Οδηγός- συντελεστή", διαφέρει από την μέθοδο του Cauchy, μόνο στο γεγονός ότι τώρα ο "Οδηγός- συντελεστής", έχει πλέον σπάσει σε άνισα μέρη, χωρίζοντάς το με διαφορετικές δυνάμεις του 2, [6],

$$ub_K = 2\max_{\{1\le k\le n: a_{n-k}<0\}} \sqrt[k]{-\frac{a_{n-k}}{a_n}}$$

Ή ισοδύναμα

$$ub_K = 2\max_{\{1\le k\le n: a_{n-k}<0\}} \sqrt[k]{-\frac{a_{n-k}}{\frac{a_n}{2^k}}}$$

Πέραν των παραπάνω παραθέτουμε τις δύο δικές μας εφαρμογές γραμμικής πολυπλοκότητα του Θεώρημα 2, ο συνδυασμός των οποίων (δηλαδή, για το ελάχιστο ποσό) αποδίδει το καλύτερο άνω φράγμα γραμμικής πολυπλοκότητας στις τιμές των θετικών ριζών ενός πολυωνύμου, [11].

Παρακαλούμε ας σημειωθεί ότι σε γενικές γραμμές μπορούμε να επιτύχουμε καλύτερα φράγματα, αν στο θεώρημα 2 έχουμε ζεύγος συντελεστών από
μη προσκείμενα(adjacent) πολυώνυμα $q_{2\ell-1}(x)$
και $q_{2i}(x), 1\le \ell < i-$, $1\le e\langle i-$ και αυτό γίνεται στο ακόλουθους ορισμούς:

FL. "**First–**λ" : εφαρμογή του Θεωρήματος 2. Για ένα πολυώνυμο P (x), όπως στην εξίσωση (2), με λ αρνητικούς συντελεστές θα ασχοληθούμε πρώτα όλες τις περιπτώσεις για τις οποίες t(q2i) > t(q2i−1), με διάσπαση του τελευταίου συντελεστή c2i−1,t(q2i), του q2i−1(x), σε t(q2i) − t(q2i−1) + 1 *ίσα τμήματα.* Τότε ζευγαρώνουμε κάθε ένα από τους λ θετικούς συντελεστές του p (x), που αντιμετωπίσαμε καθώς κινούμασταν σε μη αυξανομένη τάξης στους εκθετών, με τον πρώτο ασύμφωνο συντελεστή

LM. "**Local-Max**" : εφαρμογή του Θεωρήματος 2. Για ένα πολυώνυμο P (x), όπως στην εξίσωση (1), ο συντελεστής $-a_k$ του όρου $-a_k x^k$ στο P (x) – όπως δίνεται στην (1)- ζευγαρλωνει με τον συντελεστή $\frac{a_m}{2^t}$, του όρου $a_m x^m$, όπου a_m είναι ο μεγαλύτερος θετικός συντελεστής με n ≥ m > k και t να αποτελούν το αριθμό των φορών που ο συντελεστής a_m χρησιμοποιείται.

Έχουμε δοκιμάσει εκτενώς - σε επιμέρους κατηγορίες ειδικών και τυχαία πολυωνύμων- και τις τέσσερις μεθόδους για φράγματα γραμμικής πολυπλοκότητας που σημειώνονται παραπάνω και τα ακόλουθα είναι μια περίληψη από τα πορίσματά μας,[11]:

- Η μέθοδος του φράγματος του «Kioustelidis» είναι γενικά, καλύτερη(ή πολύ καλύτερη) από του Cauchy΄ αυτό συμβαίνει γιατί η πρώτη σπάει τον "Οδηγός- συντελεστή" σε άνισα μέρη, ενώ η δεύτερη τον χωρίζει σε ίσα μέρη.

- Η δική μας μέθοδος του φράγματος «first-λ» όπως το όνομα δείχνει, χρησιμοποιεί επιπλέον συντελεστές και, ως εκ τούτου, δεν αποτελεί έκπληξη ότι είναι, σε γενικές γραμμές, η καλύτερη (ή πολύ καλύτερη) από τα δύο προηγούμενες μεθόδους. Στις λίγες περιπτώσεις όπου η μέθοδος Kioustelidis είναι καλύτερη από την «first-λ» , το τοπικό μας μέγιστο παίρνει πάλι το προβάδισμα.

Κατά συνέπεια, δεδομένου του γραμμικού κόστος εκτέλεσής τους, οι min (FL, LM) είναι οι καλύτερες μέθοδοι μεταξύ για εύρεση φραγμάτων γραμμικής πολυπλοκότητας [11], και το χρησιμοποιούμε στο τμήμα 3 για να προσδιορίσουμε το σημείο αποκοπής μεταξύ γραμμικής και τετραγωνικής πολυπλοκότητας φραγμάτων. Επιπλέον, στο τμήμα 3 βλέπουμε ότι η εφαρμογή των min (FL, LM) στην μέθοδο συνεχών κλασμάτων απομόνωσης για απομόνωση πραγματικών ριζών, με το VAS-CF/min (FL, LM), παίρνουμε την ταχύτητα κατά πολύ περισσότερο από 15% καλύτερη σε σύγκριση με το V AS -CF / Cauchy.

2.3 Φράγματα τετραγωνικής πολυπλοκότητας που προέρχονται από το θεώρημα 2

Στην υποενότητα αυτή παρουσιάζουμε τα δύο φράγματα τετραγωνικής πολυπλοκότητας που θα χρησιμοποιηθούν στο πείραμά μας. Δείτε την βιβλιογραφία, [13], για μια πιο ολοκληρωμένη συζήτηση των δευτεροβάθμιων φραγμάτων πολυπλοκότητας. Ξεκινάμε με την KQ, την τετραγωνικό έκδοση της μεθόδου φράγματος του Kioustelidis ΄για τις τιμές των θετικών ριζών ενός πολυωνύμου. Απ 'όσο γνωρίζουμε αυτό παρουσιάστηκε για πρώτη φορά από το Χονγκ, [14,σ 574], και μπορεί να διατυπωθεί ως εξής:

KQ. Kioustelidis Τεραγονικής πολυπλοκότητας εφαρμογή του Θεωρήματος 2. Για ένα πολυώνυμο p (x), όπως στην εξίσωση (1), κάθε αρνητικός συντελεστής $a_i < 0$ ζευγαρώνει με κάθε ένα από τους προηγούμενους συντελεστές a_j διαιρεμένους με 2^{j-i} -δηλαδή, κάθε θετικός συντελεστής a_j "χωρίζεται" σε άνισα μέρη, όπως γίνεται με ακριβώς τον "Οδηγός- συντελεστή" στην μέθοδο φράγματος του Kioustelidis- και ο ελάχιστος αναλαμβάνει έναντι όλων των j, στην συνέχεια ο μεγαλύτερος αναλαμβάνει έναντι όλων των i

Δηλαδή, έχουμε

$$ub_{KQ} = 2 \max_{\{ai<0\}} \min_{\{a_j>0: j>i\}} \sqrt[j-i]{-\frac{a_i}{a_j}},$$

Ή ισοδύναμα

$$ub_{KQ} = \max_{\{ai<0\}} \min_{\{a_j>0: j>i\}} \sqrt[j-i]{-\frac{a_i}{\frac{a_j}{2^{j-1}}}}.$$

Το θεώρημα του Χονγκ 2,2, αν εξεταστεί προσεκτικά, αποκαλύπτει ότι για τα πολυώνυμα μίας μεταβλητής είναι η τετραγωνική μέθοδος φράγματος του Kioustelidis ,KQ΄ εξάλλου, η σχέση τους με το Θεώρημα 2 είναι αρκετά προφανές.

Η παρατήρηση αυτή μας οδηγεί στο δική μας τετραγωνικής πολυπλοκότητας μέθοδο φράγματος LMQ, που βασίζεται στο θεώρημα 2, [13], το οποίο μπορεί να διατυπωθεί ως εξής:

LMQ. "Local-Max" Τεραγονικής πολυπλοκότητας εφαρμογή του Θεωρήματος 2. Για ένα πολυώνυμο p (x), όπως στην εξίσωση (1), κάθε αρνητικός συντελεστής ai <0 ζευγαρώνει με κάθε μία από τους προηγούμενους θετικούς συντελεστές aj διαιρεμένους με 2^{t_j} ,-δηλαδή, κάθε θετικός συντελεστής aj είναι "χωρίζεται" σε άνισα μέρη, όπως γίνεται με ακριβώς μόνο με το τοπικό μέγιστο συντελεστή στην μέθοδο φράγματος τοπικού μεγίστου΄ το tj αρχικά έχει οριστεί ως 1 και προστίθεται κάθε φορά με το θετικό συντελεστή aj που χρησιμοποιείται - και ο ελάχιστος αναλαμβάνει όλα τα j΄ στη συνέχεια, ο ανώτατος αναλαμβάνει όλα τα i

Δηλαδή, έχουμε

$$ub_{LMQ} = \max_{\{ai<0\}} \min_{\{a_j>0: j>i\}} \sqrt[j-i]{-\frac{a_i}{\frac{a_j}{2^{t_j}}}}.$$

Αφού $2^{t_j} \le 2^{j-i}$ -όπου i και j είναι οι δείκτες που υλοποιούν το μέγιστο του min΄η ισότητα ισχύει όταν δεν λείπουν όροι στο πολυώνυμο- είναι σαφές ότι τα όρια που υπολογίζεται με το LMQ είναι ευκρινέστερα με το συντελεστή $2^{\frac{j-i-t_j}{j-i}}$ απ' ότι εκείνες που υπολογίζονται από το KQ του «Kioustelidis »΄ ωστόσο ο VAS-ΣΚ υλοποιείται και με τις δύο μεθόδους τετραγωνικής πολυπλοκότητας.

Έχουμε εφαρμόσει επίσης τον γρηγορότερο αλγόριθμο τετραγωνικής πολυπλοκότητας FLQ, αλλά παραδεχόμαστε από τη συμπεριφορά του ότι είναι παρόμοια με εκείνη του LMQ και δε μπορεί να συγκριθεί άμεσα με KQ΄ λεπτομέρειες μπορούν να βρεθούν αλλού, [15].

3 Εμπειρικά αποτελέσματα

Στην ενότητα αυτή συγκρίνουμε τέσσερεις υλοποιήσεις της μεθόδου απομόνωσης πραγματικών ριζών VAS-CF χρησιμοποιώντας δύο γραμμικής και δύο τετραγωνικής πολυπλοκότητας φράγματα στις τιμές των θετικών ριζών των πολυωνύμων.

Τα δύο φράγματα γραμμικής πολυπλοκότητας είναι τα εξής: του Cauchy και του min (FL,LM), το ελάχιστο όριο του *F*irst *L*ambda κα τα τοπικά μέγιστα φράγματα [11],], ενώ τα δύο φράγματα τετραγωνικής πολυπλοκότητας είναι τα εξής: η KQ, που είναι παραλλαγή τετραγωνικής πολυπλοκότητας της μεθόδου Kioustelidis΄, που μελετήθηκε από το Χονγκ, [14], και η LMQ, που είναι η έκδοση τετραγωνικής πολυπλοκότητα ς τοπικού μεγίστου, [13].

Η επιλογή μας από τα διάφορα φράγματα στην υλοποιήσεις του VAS-CF είναι δικαιολογημένη, δεδομένου τα εξής:

1. Από τα φράγματα γραμμικής πολυπλοκότητας συμπεράναμε ότι:

   (α) Η μέθοδος του Cauchy, C, που πρέπει να χρησιμοποιείται ως σημείο αναφοράς, αφού χρησιμοποιούταν για τα τελευταία 30 χρόνια, και
   (β) Η μέθοδος μας min(FL, LM) , [11], που είναι οι καλύτερες μέθοδοι μεταξύ των γραμμικής

πολυπλοκότητας φραγμάτων, προκειμένου να δούμε πότε εφαρμογή τους αυτή θα ξεπεράσουν αυτήν των δύο φραγμάτων τετραγωνικής πολυπλοκότητας.

2. Από τα φράγματα τετραγωνικής πολυπλοκότητας συμπεράναμε ότι:
(α) Η μέθοδος KQ Kioustelidis , και
(β) Η μέθοδος μας LMQ , προκειμένου να συγκρίνουμε τις επιδόσεις τους΄, όπως εξηγείται στην προηγούμενη ενότητα η LMQ υπολογίζει με μεγαλύτερη ακρίβεια από την KQ.

Έχουμε εφαρμόσει όλες τις εκδόσεις του VAS- CF, ως μέρος του πυρήνα Mathematica. Όλοι χρησιμοποιούν την ίδια εφαρμογή του αλγορίθμου των Shaw και Traub για τις Taylor shifts (βλέπε [16]). Ακολουθήσαμε τη συνήθη πρακτική και χρησιμοποιήσαμε ως σημείο αναφοράς τα Laguerre3, Chebyshev (πρώτο4 και δεύτερο5 είδος), Wilkinson6 και Mignotte7 πολυώνυμα, καθώς και αρκετούς τύπους πολυώνυμα που δημιουργούνται τυχαία με βαθμούς {100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000}. Για τα τυχαία πολυώνυμα το μέγεθος των συντελεστών κυμαίνεται μεταξύ -2^20 και 2^20.

Όλες οι υπολογισμοί έγιναν σε φορητό υπολογιστή με Windows XP , 1,8 Ghz, Pentium M επεξεργαστή, και 2 GB μνήμης RAM. Η αλλαγή στη χρήση μνήμης ήταν αμελητέα σε κάθε περίπτωση, και ως εκ τούτου, δεν περιλαμβάνεται στους παρακάτω πίνακες. Η μέση επιτάχυνση υπολογίστηκε με τον τύπο: Επιτάχυνση = 100 · (LMQ - Cauchy) / Cauchy, από την οποία παραλείπεται το αρνητικό πρόσημο.

Παράθεμα :

1. $L_0(x)=1, L_1(x)=1-x, and\ L_{n+1}(x)=\frac{1}{n+1}((2n+1-x)L_n(x)-nL_{n-1}(x))$
2. $T_0(x)=1, T_1(x)=x, and\ T_{n+1}(x)=2xT_n(x)-T_{n-1}(x)$
3. $U_0(x)=1, U_1(x)=2x, and\ U_{n+1}(x)=2xU_n(x)-U_{n-1}(x)$
4. $W(x)=\prod_{i=1}^{n}(x-i)$
5. $M_n(x)=x^n-2(5x-1)^2$

Πίνακας 1.

Ειδικά Πολυώνυμα. Μερικά ενδεικτικά αποτελέσματα δίνονται για βαθμούς 100, 1000, 1500 και 2000. Η μέση ταχύτητα για αυτόν τον πίνακα είναι περίπου 32%

Polynomial Class	Degree	Time (s)			
		Cauchy	*FL + LM*	*KQ (Hong)*	*LMQ*
Laguerre	100	0.23	0.19	0.19	0.17
Laguerre	1000	979	665	729	633
Laguerre	1500	7194	4903	5356	4569
Laguerre	2000	27602	21007	22712	19277
Chebyshev I	100	0.19	0.17	0.16	0.11
Chebyshev I	1000	517	460	496	299
Chebyshev I	1500	3681	3333	3381	2188
Chebyshev I	2000	16697	15010	14571	10473
Chebyshev II	100	0.42	0.17	0.15	0.10
Chebyshev II	1000	529	437	443	296
Chebyshev II	1500	3772	3198	3190	2166
Chebyshev II	2000	16559	14492	14370	10184
Wilkinson	100	0.03	0.03	0.03	0.03
Wilkinson	1000	54.6	44.5	43.7	43.3
Wilkinson	1500	339	295	270	265
Wilkinson	2000	1361	1305	1241	1242
Mignotte	100	0.008	0.004	0.008	0.004
Mignotte	1000	0.79	0.78	0.81	0.66
Mignotte	1500	2.05	2.12	2.06	1.77
Mignotte	2000	4.52	4.37	4.47	3.69

Πίνακας 2.

Πολυώνυμα με τυχαίους 10-bit συντελεστές. Η μέση ταχύτητα για αυτόν τον πίνακα είναι περίπου 39%

Degree	No. of roots Avg (Min/Max)	Time (s), Avg (Min/Max)			
		Cauchy	*FL + LM*	*KQ (Hong)*	*LMQ*
100	4.4 (2/6)	0.01 (0.00/0.01)	0.01 (0.01/0.02)	0.01 (0.01/0.02)	0.01 (0.01/0.01)
200	4.0 (2/8)	0.06 (0.02/0.18)	0.06 (0.03/0.16)	0.05 (0.03/0.14)	0.04 (0.03/0.09)
300	4.8 (4/6)	0.14 (0.07/0.24)	0.12 (0.06/0.22)	0.13 (0.07/0.19)	0.09 (0.07/0.13)
400	4.4 (4/6)	0.17 (0.12/0.21)	0.18 (0.12/0.25)	0.17 (0.12/0.20)	0.16 (0.12/0.20)
500	4.8 (2/8)	0.70 (0.21/1.96)	0.54 (0.20/1.22)	0.35 (0.21/0.56)	0.32 (0.20/0.50)
600	5.2 (4/6)	0.96 (0.46/1.41)	0.86 (0.51/1.25)	0.60 (0.42/0.84)	0.53 (0.42/0.72)
700	4.0 (2/6)	0.95 (0.45/1.68)	0.81 (0.44/1.33)	0.82 (0.44/1.25)	0.69 (0.50/0.91)
800	5.2 (4/8)	1.97 (0.67/4.09)	1.68 (0.74/3.33)	1.22 (0.71/2.25)	1.02 (0.72/1.70)
900	3.6 (2/6)	2.56 (0.68/7.15)	2.27 (0.72/6.13)	1.44 (0.71/2.55)	1.19 (0.67/1.87)
1000	6.4 (4/8)	4.07 (1.63/9.02)	3.56 (1.54/7.64)	2.86 (1.57/4.51)	2.06 (1.38/3.18)
1500	4.0 (2/6)	10.6 (2.73/26.1)	7.51 (2.33/13.9)	5.78 (2.35/10.1)	5.24 (2.43/7.77)
2000	6.8 (4/12)	53.8 (7.54/137)	45.5 (7.90/118)	23.3 (7.67/53.9)	19.1 (7.61/40.2)

Πίνακας 3.

Πολυώνυμα με τυχαίους 1000-bit συντελεστές. Η μέση ταχύτητα για αυτόν τον πίνακα είναι περίπου 48%

Degree	No. of roots Avg (Min/Max)	Time (s), Avg (Min/Max)			
		Cauchy	*FL + LM*	*KQ (Hong)*	*LMQ*
100	4.0 (4/4)	0.01 (0.00/0.02)	0.01 (0.00/0.02)	0.01 (0.00/0.02)	0.01 (0.00/0.02)
200	3.6 (2/6)	0.06) (0.03/0.12)	0.05 (0.02/0.10)	0.04 (0.02/0.06)	0.03 (0.01/0.06)
300	4.8 (2/8)	0.12 (0.04/0.32)	0.11 (0.04/0.28)	0.10 (0.04/0.23)	0.09 (0.04/0.17)
400	4.4 (2/6)	0.29 (0.06/0.54)	0.25 (0.06/0.44)	0.24 (0.06/0.44)	0.16 (0.06/0.25)
500	5.2 (4/8)	0.68 (0.16/1.20)	0.55 (0.17/0.95)	0.45 (0.21/0.92)	0.32 (0.21/0.48)
600	3.6 (2/4)	0.76 (0.19/2.09)	0.54 (0.18/0.96)	0.43 (0.19/0.66)	0.39 (0.18/0.52)
700	3.6 (0/6)	1.26 (0.25/2.82)	1.28 (0.19/2.51)	0.85 (0.19/1.54)	0.68 (0.19/1.29)
800	4.4(2/6)	3.03 (0.29/5.50)	2.53 (0.26/4.76)	1.08 (0.34/1.68)	0.93 (0.27/1.53)
900	5.6 (4/8)	4.55 (1.05/9.32)	3.72 (1.02/7.53)	2.23 (1.00/3.09)	1.59 (0.76/2.68)
1000	3.6 (2/6)	2.42 (0.46/4.62)	2.06 (0.44/3.92)	1.27 (0.40/2.00)	1.04 (0.42/1.68)
1500	5.6 (4/8)	16.1 (2.30/40.2)	9.41 (1.99/18.2)	7.17 (2.10/11.9)	5.63 (1.96/10.8)
2000	5.2(4/6)	23.3 (4.12/79.8)	19.4 (4.08/65.4)	13.2 (4.33/33.4)	10.4 (4.11/20.2)

Πίνακας 4.

Monic πολυώνυμα με τυχαίους 10-bit συντελεστές. Η μέση ταχύτητα για αυτόν τον πίνακα είναι περίπου 31%

Degree	No. of roots Avg (Min/Max)	Time (s), Avg (Min/Max)			
		Cauchy	*FL + LM*	*KQ (Hong)*	*LMQ*
100	4.8 (2/8)	0.01 (0.01/0.02)	0.01 (0.00/0.02)	0.01 (0.01/0.02)	0.01 (0.00/0.02)
200	5.6 (4/6)	0.08 (0.03/0.16)	0.06 (0.03/0.13)	0.06 (0.03/0.09)	0.05 (0.03/0.08)
300	4.8(4/6)	0.12 (0.08/0.22)	0.12 (0.08/0.21)	0.12 (0.08/0.15)	0.10 (0.08/0.15)
400	4.8 (4/6)	0.19 (0.19/0.29)	0.19 (0.16/0.26)	0.18 (0.15/0.26)	0.17 (0.16/0.20)
500	5.2 (4/10)	0.44 (0.18/1.38)	0.42 (0.18/1.19)	0.33 (0.18/0.74)	0.32 (0.19/0.63)
600	5.6 (4/8)	0.99 (0.30/2.04)	0.76 (0.30/1.21)	0.65 (0.31/0.94)	0.49 (0.30/0.72)
700	5.2 (4/8)	1.14 (0.43/1.63)	0.99 (0.42/1.47)	0.92 (0.46/1.30)	0.73 (0.49/0.94)
800	5.6(4/8)	1.45 (0.66/1.99)	1.29 (0.64/1.62)	1.22 (0.65/1.42)	0.90 (0.69/1.03)
900	4.4 (2/6)	1.01 (0.74/1.18)	1.01 (0.71/1.31)	1.05 (0.69/1.36)	1.09 (0.72/1.33)
1000	5.6 (4/8)	3.40 (1.18/7.09)	3.02 (1.03/5.94)	(1.10/4.28) (1.10/4.28)	(1.10/2.70) (1.10/2.70)
1500	6.8 (6/8)	14.8 (6.06/27.3)	11.8 (5.86/17.1)	8.43 (5.98/12.9)	6.80 (4.90/9.24)
2000	7.6(4/14)	54.8 (6.12/137)	47.6 (6.09/120)	23.9 (6.15/56.0)	19.4 (6.03/42.2)

Πίνακας 5.

Monic πολυώνυμα με τυχαίους 1000-bit συντελεστές. Η μέση ταχύτητα για αυτόν τον πίνακα είναι περίπου 60%

Degree	No. of roots Avg (Min/Max)	Time (s), Avg (Min/Max)			
		Cauchy	*FL + LM*	*KQ (Hong)*	*LMQ*
100	6.0 (4/8)	0.03 (0.02/0.04)	0.01 (0.01/0.03)	0.01 (0.00/0.02)	0.01 (0.00/0.02)
200	5.2 (4/8)	0.09 (0.02/0.22)	0.07 (0.02/0.19)	0.04 (0.02/0.11)	0.04 (0.03/0.06)
300	5.6(4/8)	0.19 (0.06/0.46)	0.14 (0.07/0.28)	0.12 (0.06/0.24)	0.14 (0.07/0.26)
400	5.2 (4/8)	0.41 (0.08/1.00)	0.24 (0.06/0.54)	0.21 (0.06/0.44)	0.15 (0.06/0.28)
500	5.6 (4/8)	0.62 (0.18/1.00)	0.39 (0.12/0.68)	0.45 (0.12/0.74)	0.26 (0.12/0.37)
600	4.8 (4/6)	1.03 (0.24/3.09)	0.52 (0.17/1.21)	0.37 (0.17/0.68)	0.32 (0.17/0.59)
700	5.2 (2/10)	1.27 (0.20/2.67)	1.02 (0.21/1.84)	0.86 (0.19/1.43)	0.65 (0.19/1.09)
800	5.6(4/8)	2.92 (0.43/5.41)	2.40 (0.39/4.46)	1.02 (0.38/2.02)	0.79 (0.38/1.38)
900	6.0 (4/8)	4.22 (0.84/9.86)	2.67 (0.80/5.78)	1.84 (0.88/2.47)	1.43 (0.74/2.22)
1000	5.6 (4/6)	4.23 (2.21/5.86)	2.90 (1.34/4.21)	2.23 (1.34/3.52)	1.44 (1.15/1.84)
1500	6.8 (6/8)	17.1 (26.06/41.8)	11.4 (5.25/28.2)	8.28 (4.86/15.7)	5.44 (3.30/10.4)
2000	6.4(6/8)	30.6 (4.80/102)	24.0 (4.59/80.9)	16.7 (4.60/47.4)	12.6 (5.02/35.9)

Πίνακας 6.

Τα γινόμενα των όρων $x^{20} - r$, με τυχαία 20-bit r. Η μέση ταχύτητα για αυτόν τον πίνακα είναι περίπου 48%

Degree	No. of roots Avg (Min/Max)	Time (s), Avg (Min/Max)			
		Cauchy	*FL + LM*	*KQ (Hong)*	*LMQ*
100	10	0.05 (0.02/0.09)	0.05 (0.02/0.09)	0.03 (0.02/0.04)	0.02 (0.02/0.02)
200	20	0.31 (0.18/0.39)	0.27 (0.16/0.38)	0.24 (0.16/0.32)	0.15 (0.12/0.20)
300	30	1.07 (0.58/1.37)	0.89 (0.60/1.11)	0.87 (0.60/1.04)	0.57 (0.56/0.60)
400	40	2.22 (1.92/2.58)	1.97 (1.86/2.27)	1.94 (1.86/2.08)	1.50 (1.35/1.70)
500	50	8.51 (6.03/11.5)	5.32 (4.24/7.28)	4.55 (4.25/5.32)	3.24 (2.87/3.74)
600	60	13.9 (11.7/17.0)	9.15 (8.28/10.2)	8.96 (8.43/9.35)	6.43 (5.96/6.84)
700	70	24.6 (21.7/29.1)	17.2 (13.7/21.2)	16.3 (13.3/19.7)	12.1 (10.6/14.0)
800	80	38.0 (33.7/44.2)	26.3 (23.6/30.4)	24.4 (19.4/30.3)	17.2 (15.1/19.3)
900	90	53.7 (40.4/63.8)	43.5 (37.0/51.5)	37.0 (28.8/45.8)	29.5 (23.1/36.6)
1000	100	89.6 (70.9/103)	69.1 (52.2/78.5)	63.9 (45.4/76.5)	50.0 (42.1/58.9)
1500	150	577 (468/696)	456 (378/533)	429 (360/473)	353 (311/402)
2000	200	2228 (1917/2711)	1907 (1674/2342)	1808 (1614/2279)	1464 (1204/1767)

Πίνακας 7.

Τα γινόμενα των όρων $x^{20} - r$, με τυχαία 1000-bit r. Η μέση ταχύτητα για αυτόν τον πίνακα είναι περίπου 25%

Degree	No. of roots Avg (Min/Max)	Time (s), Avg (Min/Max)			
		Cauchy	*FL + LM*	*KQ (Hong)*	*LMQ*
100	10	0.08 (0.05/0.10)	0.08 (0.05/0.12)	0.11 (0.06/0.31)	0.09 (0.03/0.23)
200	20	1.65 (0.96/2.14)	1.42 (0.97/2.09)	1.28 (1.02/0.1.45)	1.31 (1.10/1.50)
300	30	7.54 (5.08/10.8)	5.20 (4.46/5.65)	4.88 (3.67/5.49)	4.24 (3.92/4.69)
400	40	15.7 (10.8/19.7)	15.7 (13.3/17.5)	14.7 (12.7/17.3)	12.7 (11.0/14.1)
500	50	42.4 (29.2/64.7)	44.5 (35.2/48.7)	35.5 (32.8/40.5)	35.0 (27.5/49.7)
600	60	117 (91.9/154)	106 (82.6/134)	103 (90.0/121)	92.0 (86.5/97.0)
700	70	248 (208/332)	252 (221/282)	240 (205/264)	189 (168/205)
800	80	549 (351/753)	481 (410/590)	474 (412/542)	382 (364/432)
900	90	1138 (971/1271)	855 (721/967)	834 (718/931)	670 (646/723)
1000	100	1661 (1513/1913)	1335 (1123/1673)	1265 (1066/1440)	1065 (947/1146)
1500	150	9004 (8233/9705)	8360 (7281/8999)	8230 (7357/9652)	6141 (5659/6470)

Εικ. 8.

Τα γινόμενα των όρων x - r με τυχαίο ακέραιο αριθμό r. Η μέση ταχύτητα για αυτόν τον πίνακα είναι περίπου 35%

Degree	No. of roots Avg (Min/Max)	Time (s), Avg (Min/Max)			
		Cauchy	*FL + LM*	*KQ (Hong)*	*LMQ*
10	100	0.46 (0.28/0.94)	0.24 (0.18/0.28)	0.34 (0.27/0.41)	0.34 (0.30/0.41)
10	200	1.46 (1.24/1.85)	1.40 (1.28/1.69)	1.41 (1.26/1.71)	1.40 (1.20/1.69)
10	500	18.1 (16.5/18.9)	18.1 (16.6/18.8)	21.2 (17.5/24.4)	22.1 (18.7/24.2)
1000	20	0.07 (0.04/0.14)	0.02 (0.02/0.03)	0.03 (0.02/0.04)	0.03 (0.02/0.04)
1000	50	3.69 (2.38/6.26)	0.81 (0.60/1.28)	0.88 (0.52/1.28)	0.81 (0.52/1.11)
1000	100	47.8 (37.6/56.9)	13.8 (10.3/19.2)	17.6 (12.4/25.9)	15.8 (11.3/21.3)

4 Συμπεράσματα

Ανακοίνωση των ακόλουθων γενικών παρατηρήσεων:

- Η μόνη διαφορά μεταξύ των μεθόδων φραγμάτων που ικανοποιείται πάντα είναι ότι η μέθοδος KQι είναι πάντα μεγαλύτερη ή ίση από την LMQ,
- Για ένα ανεξάρτητο πολυώνυμο ένα καλύτερο φράγμα μπορεί να οδηγήσει σε χειρότερες επιδόσεις σε VAS-CF
- Τα αποτελέσματα χρονισμού είναι επιρρεπή σε σφάλματα μέτρησης, κάτι που πλήττει ιδιαίτερα τους μικρούς χρονισμούς.

Λαμβάνοντας αυτές τις παρατηρήσεις υπόψη, θα μπορούσε κανείς να προβλέψει ότι η χρήση της μεθόδου LMQ θα πρέπει να δώσει την καλύτερη απόδοση στον VAS-CF, εκτός από τα ανεξάρτητα σημεία δεδομένων για τα οποία αυτό δεν είναι αλήθεια.

Πράγματι, από τους πίνακες στο τμήμα 3, είναι προφανές ότι η εφαρμογή του VAS-CF χρησιμοποιώντας την δική μας μέθοδο LMQ είναι η ταχύτερη για όλες τις δοκιμασμένες κατηγορίες των πολυωνύμων, εκτός από την περίπτωση πάρα πάρα πολύ μεγάλων ριζών, Πίνακας 8. Για όλες τις περιπτώσεις, η μέση συνολική βελτίωση του χρόνου υπολογισμού είναι περίπου 40%. Στην περίπτωση των πάρα πάρα πολύ μεγάλων ριζών ο VAS-CF που χρησιμοποίησε την LMQ είναι μια πολύ κοντά στην VAS-CF χρησιμοποιώντας το φράγμα μας γραμμικής πολυπλοκότητας min (FL, LM).

Επιπλέον, όπως φαίνεται και αλλού, [17], χρησιμοποιώντας μόνο το φράγμα μας min (FL, LM) ο

αλγόριθμος VAS-CF είναι πάντα πιο γρήγορος από τον Vincent-Collins-Akritas8, [19], ή σε οποιοδήποτε παραλλαγή του, [20]. Ως εκ τούτου, τα τρέχοντα αποτελέσματά μας διευρύνουν το χάσμα μεταξύ των δύο αυτών μεθόδων

1. J.H. Vincent, Sur la r´esolution des ´equations num´eriques, *J. Math. Pure Appl.*, **1**, pp. 341–372, 1936.
2. Alesina, M. Galuzzi, A new proof of Vincent's theorem, *L'Enseignement Mathemathique*, **44**, pp. 219–256, 1998.
3. A.G. Akritas, *Elements of Computer Algebra with Applications*, John Wiley Interscience, New York, 1989.
4. A.G. Akritas, A.W. Strzebonski, A comparative study of two real root isolation methods, *Nonlinear Anal. Model. Control*, **10**(4), pp. 297–304, 2005.
5. E. P. Tsigaridas, I. Z. Emiris, Univariate polynomial real root isolation: Continued fractions revisited, in: *ESA 2006*, Y. Azar, T. Erlebach (Eds.), LNCS 4168, pp. 817–828, 2006.
6. B. Kioustelidis, Bounds for positive roots of polynomials, *J. Comput. Appl. Math.*, **16**(2), pp. 241–244, 1986.
7. V. Sharma, Complexity of Real Root Isolation Using Continued Fractions, in: *Proc. Annual ACM ISSAC, Waterloo, Canada*, C.W. Brown (Ed.), pp. 339–346, 2007
8. V. Sharma, *Complexity Analysis of Algorithms in Algebraic Computation*, Ph.D. Thesis,

Department of Computer Sciences, Courant Institute of Mathematical Sciences, New York University, 2007.
9. D. S¸ tef˘anescu, New bounds for positive roots of polynomials, *J. Univers. Comput. Sci.*, **11**(12), pp. 2132–2141, 2005.
10. A.G. Akritas, P. S. Vigklas, A comparison of various methods for computing bounds for positive roots of polynomials, *J. Univers. Comput. Sci.*, **13**(4), pp. 455–467, 2007.
11. A.G. Akritas, A.W. Strzebonski, P. S. Vigklas, Implementations of a new theorem for computing bounds for positive roots of polynomials, *Computing*, **78**, pp. 355–367, 2006.
12. N. Obreschkoff, *Verteilung und Berechnung der Nullstellen reeller Polynome*, VEB Deutscher Verlag der Wissenschaften, Berlin, 1963.
13. A.G. Akritas, A.W. Strzebonski, P. S. Vigklas, Quadratic complexity bounds on the values of the positive roots of polynomials (submitted).
14. H. Hong Bounds for absolute positiveness of multivariate polynomials, *J. Symb. Comput.*, **25**(5), pp. 571–585, 1998.
15. A.G. Akritas, A. I. Argyris, A.W. Strzebo´nski, FLQ, the Fastest Quadratic Complexity Bound on the Values of Positive Roots of Polynomials (to appear).
16. J. von zur Gathen, J. Gerhard, Fast algorithms for Taylor shifts and certain difference equations, in: *Proceedings of ISSAC'97, Maui, Hawaii, U.S.A.*, pp. 40–47, 1997.
17. A.G. Akritas, A.W. Strzebonski, P. S. Vigklas, Advances on the continued fractions method using better estimations of positive root bounds, in: *Proceedings of the 10th International Workshop on Computer Algebra in Scientific Computing, CASC 2007, Bonn, Germany, September 16–20, 2007*, V.G. Ganzha, E.W. Mayr, E.V. Vorozhtsov (Eds.), LNCS 4770, Springer Verlag, Berlin, pp. 24–30, 2007.
18. .F. Boulier, *Syst`emes polynomiaux : que signifie "r´esoudre"?*, Universit´e Lille 1, UE INFO 251, 2007.
19. G. E. Collins, A.G. Akritas, Polynomial real root isolation using Descartes' rule of signs, in: *Proceedings of the 1976 ACMSymposium on Symbolic and Algebraic Computations*, Yorktown Heights, N.Y., pp. 272–275, 1976.
20. F. Rouillier, P. Zimmermann, Efficient isolation of polynomial's real roots, *J. Comput. Appl. Math.*, **162**, pp. 33–50, 2004.

Εισαγωγή

Ο αλγόριθμος VAS(Vincent, Akritas, Strzebonski) που υλοποιείται σε αυτήν την εργασία σε γλώσσα προγραμματισμού C++, αφορά ακέραιους αριθμούς. Η λειτουργία του ουσιαστικά είναι να βρίσκει ρίζες πολυωνύμων, όχι όμως με την ακριβή τιμή τους αλλά σε μορφή διαστημάτων στα οποία περιέχονται. Όπως αναφέρθηκε στο Α' μέρος της εργασίας, η προσεγγιστική του φύση οφείλεται στα θεωρήματα του Taylor και Vincent που στην εποχή στην οποία δημιουργήθηκαν η ακριβής επίλυση δεν ήταν εφικτή με τα τότε υπολογιστικά συστήματα. Βασικά ο αλγόριθμος βρίσκει τις θετικές ρίζες και εάν βάλουμε όπου χ το (-χ) βρίσκει και τις θετικές ρίζες του f(-x), δηλαδή τις αρνητικές.

Παράδειγμα

Έστω το πολυώνυμο σε μορφή γινομένων

f (x)= (x + 17)^3 (x + 32) (x +35)^2 (x - 8)^4 (x - 14)^5 (x - 11)^2

ή

στην αναλυτική του μορφή

f (x)= 20x ^3 + 32x^2 - 8x - 11

Τότε ο αλγόριθμος VAS θα βρει :

Θετικές ρίζες

[(7.9) U (10,12) U (13,15)]

Αρνητικές ρίζες

[(-16.-18) U (-31,-33) U (-34,-36)]

Περιγραφή των δομικών στοιχείων

Η υλοποίηση του αλγορίθμου αποτελείται από τα εξής υποπρογράμματα :

1. "rootslist.h", το οποίο είναι η υλοποίηση μιας διασυνδεδεμένης λίστας FIFO.
2. "list.h", το οποίο αποτελεί ένα αφηρημένο τύπο δεδομένων μορφής LIFO(STACK).
3. "Rational.h", το οποίο αφορά μια κλάση για έναν ιδιόμορφο τύπο δεδομένων, τους «ρητούς» .
4. "Polynomial.h", το οποίο ως επέκταση του προηγούμενου, δημιουργεί συναρτήσεις χρησιμοποιώντας την δομή των «ρητών» αριθμών.
5. "bound.h", το οποίο υλοποιεί τους αλγορίθμους LMQ,FLQ(άνω και κάτω φραγμάτων και για τους δύο)
6. "VAS.cpp", το οποίο συνιστά το βασικό πρόγραμμα με την μεταφορά των βημάτων του αλγορίθμου VAS αλλά χρησιμοποιώντας βεβαίως όλα τα παραπάνω αρχεία.

Η σειρά με την οποία αναφέρονται δεν είναι τυχαία, αλλά τα παρουσιάζουμε από το πιο απλό στο πιο σύνθετο.

Γενικές σημειώσεις

1. Υλοποίηση αλγορίθμου ως πακέτο υποπρογραμμάτων

Θα μπορούσαμε να φτιάξουμε όλο το πρόγραμμα σε ένα και μόνο αρχείο τύπου ".cpp", όμως μία βασική αρχή του σωστά δομημένου προγραμματισμού είναι να έχουμε ξεχωριστά αυτόνομα προγράμματα που αλληλεπιδρούν μεταξύ τους. Έτσι μπορούμε να βελτιστοποιούμε ή να διορθώσουμε κάποιο κομμάτι που επιθυμούμε αλλά ταυτόχρονα κάνουμε πιο εύκολη την κατανόηση του από κάποιον αναγνώστη.

2. Εισαγωγή πολυωνύμου:

Όπως θα παρουσιαστεί και αργότερα το αρχείο "Polynomial.h" έχουμε δημιουργήσει δύο διαφορετικούς τρόπους εισαγωγής του πολυωνύμου στο πρόγραμμα

Α)*Εισαγωγή με όρους:* Βάζουμε κάθε όρο του πολυωνύμου που έχει την μορφή

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0$$

Πρώτα δίνουμε το πλήθος των όρων, μετά τον συντελεστή και τέλος τον εκθέτη κάθε όρου ξεχωριστά. Επαναλαμβάνουμε την διαδικασία για όσους όρους έχουμε εισάγει είτε ξεκινώντας από τον μεγαλύτερο φτάνοντας στον μικρότερο, είτε με τυχαία σειρά(είναι δική μας επιλογή). Έτσι και αλλιώς θα καλέσουμε συνάρτηση που ταξινομεί κατά φθίνουσα σειρά τους όρους αυτούς.

Β)*Εισαγωγή από αρχείο μορφής "txt":*Μια πιο πρακτική μέθοδος είναι να αποθηκεύσουμε τα πολυώνυμα που επιθυμούμε να βρούμε τις ρίζες τους σε αρχεία κειμένου με ονόματα π.χ. poly01.txt, poly02.txt,…, polyN.txt

με την μορφή

(16384 *x^ 15 + -61440 *x^ 13 + 92160 *x^ 11 + -70400 *x^ 9 + 28800 *x^ 7 + -6048 *x^ 5 + 560 *x^ 3 + -15 *x^ 1 + 0 *x^ 0)^ 1

end

Δηλαδή από τον μεγαλύτερο όρο στον μικρότερο, αφήνοντας ένα κενό μεταξύ τους και βάζοντας τον αρνητικό τελεστή «-1» όταν έχουμε αρνητικό αριθμό. Όλη η παραπάνω έκφραση εσωκλείεται σε παρενθέσεις, αφήνοντας ένα κενό στην αρχή και το τέλος, και την υψώνουμε σε οποιονδήποτε εκθέτη θέλουμε. Έτσι και αλλιώς θα απλοποιηθεί το πολυώνυμο μέσω της διαδικασίας απαλοιφής παραγόντων ελεύθερων από τετράγωνα που περιγράφεται στο Α΄ΜΕΡΟΣ στις σελίδες αλλά και πως υλοποιείται στον κώδικα μας στην αμέσως επόμενη ενότητα. Όταν τελειώσουμε βάζουμε την λέξη-κλειδί "end" για να δηλώσουμε το τέλος φόρτωσης αρχείου.

Εναλλακτικά μπορούμε να εισάγουμε το πολυώνυμο με την μορφή

(8192 *x^ 14 + -28672 *x^ 12 + 39424 *x^ 10 + -26880 *x^ 8 + 9408 *x^ 6 + -1568 *x^ 4 + 98 *x^ 2 + -1 *x^ 0)^ 1

(1 *x^ 11 + -707 *x^ 10 + -345 *x^ 9 + -673 *x^ 8 + 564 *x^ 7 + 870 *x^ 6 + -147 *x^ 5 + 200 *x^ 4 + -561 *x^ 3 + 270 *x^ 2 + 285 *x^ 1 + -212 *x^ 0)^ 3

(1 *x^ 30 + -50 *x^ 2 + 20 *x^ 1 + -2 *x^ 0)^ 4

end

Δηλαδή ως γινόμενο πολυωνύμων όπου επαναλαμβάνουμε την διαδικασία σε κάθε ένα ξεχωριστά όπως θα αναλύσουμε και στην συνέχεια.

3. Ειδική εκδοχή του "square free decomposition"

Στην διεθνή βιβλιογραφία, η μέθοδος αυτή(διαδικασία απαλοιφής παραγόντων ελεύθερων από τετράγωνα) υλοποιείται ως εξής:

Ένα πολυώνυμο της μορφής

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0$$

το οποίο έχει m ρίζες

$\alpha_0, \alpha_1,..., \alpha_{m-1}$

πολλαπλότητας από 1εώς k μετατρέπεται σε γινόμενο

$$p(x) = (x-\alpha_0)^1(x-\alpha_1)^2 \cdots (x-\alpha_{m-2})^{k-1}(x-\alpha_{m-1})^k$$

Στην συνέχεια απαλείφουμε τις πολλαπλότητες

$$p(x) = (x-\alpha_0)(x-\alpha_1)^{\cdots} (x-\alpha_{m-2})(x-\alpha_{m-1})$$

και επαναφέρουμε το πολυώνυμου στην αρχική(αναλυτική) του μορφή

$$p(x) = b_n x^n + b_{n-1}x^{n-1} + \ldots + b_1 x + b_0$$

Όπου ο μέγιστος συντελεστής δεν είναι κατά ανάγκη ίδιος.

Στο δικό μας πρόγραμμα όμως κάνουμε κάτι το διαφορετικό.

Φτάνουμε το πολυώνυμο στην μορφή

$$p(x) = (x-\alpha_0)^1(x-\alpha_1)^2 \cdots (x-\alpha_{m-2})^{k-1}(x-\alpha_{m-1})^k$$

και για κάθε έναν από αυτούς τους όρους, αφού βέβαια απαλείψουμε τους εκθέτες, καλούμε τον αλγόριθμο VAS. Αξίζει να σημειωθεί ότι κάθε ένας από αυτούς τους όρους μπορεί να μην είναι τόσο απλός αλλά να έχει την γενική μορφή:

$$p(x) = c_n x^n + c_{n-1}x^{n-1} + \ldots + c_1 x + c_0$$

με ρίζες όμως αυτήν την φορά πολλαπλότητας 1.

4. *Εισαγωγή βοηθητικής μαθηματικής βιβλιοθήκης*

Δυστυχώς η C++ όπως και όλες οι αντικειμενοστραφείς γλώσσες προγραμματισμού δεν έχουν όλες εκείνες τις μαθηματικές συναρτήσεις που μας χρειάζονται ως βοηθητικές για να πετύχουμε την επιθυμητή εμβέλεια και ακρίβεια στις πράξεις που υλοποιούμε.

Για αυτές τις λειτουργίες υπάρχουν έτοιμα μαθηματικά προγράμματα, όπως είναι το MATHEMATICA και το MATLAB, τα οποία έχουν ενσωματωμένες όλες εκείνες τις ρουτίνες(και παραπάνω) που χρειαζόμαστε.

Βέβαια, για αυτό το λόγω έχουν εργαστεί ομάδες προγραμματιστών, στηριζόμενοι στην μνήμη RAM που χρησιμοποιούν αυτά τα είδη των

προγραμμάτων. Για αυτόν τον λόγο η εγκατάστασή τους απαιτεί πολύ χώρο στον υπολογιστή μας αλλά και μεγάλο ποσοστό χρήσης της CPU.

Όπως μπορεί να παρατηρηθεί, κάθε εντολή σε MATHEMATICA αντιστοιχεί σε περισσότερες από μία στην C++.Όμως και πάλι, υπάρχουν προδιαγραφές της C++, που δεν μπορούμε να ξεπεράσουμε και αφορούν την εμβέλεια των αριθμών που επεξεργαζόμαστε.

Αυτό που μπορούμε να κάνουμε είναι, εισάγοντας το μαθηματικό πακέτο "gmp" , να αντιμετωπίσουμε τα θέματα ακρίβειας των αριθμών, χρησιμοποιώντας τις ρουτίνες δημιουργίας και επεξεργασίας ρητών αριθμών στην μέγιστη δυνατή εμβέλεια.

Όμως και αυτό το πακέτο, έχει κάποιες ελλείψεις καθώς δεν έχει ακόμη εισάγει συναρτήσεις όπως "log", "exp" και "ceiling". Αυτές οι συναρτήσεις είναι απαραίτητες για να υπολογίσουμε τα φράγματα *των ριζών που τυχαίνει να έχουν μεγάλη ακρίβεια και μετά από μία εκθετική και λογαριθμική πράξη και την οποία η στρογγύλευση της C++ χάνει σε πολλά ψηφία.*

Αυτό ισχύει και για άλλα πακέτα όπως το symbpyth….

5. *Υλοποίηση της πράξης P(x+α) με την μέθοδο Taylor Shift*

Σε συγκεκριμένα βήματα του αλγορίθμου VAS, χρειάζεται να κάνουμε την πράξη P(x+α) με $a \in R$. *Εκτός από την απλή αντικατάσταση* $x \leftarrow x+a$, *υπάρχει και η μέθοδος αυτή που σε υπολογιστικά προγραμματιστικά συστήματα, έχει αποδειχθεί πολύ πιο γρήγορη!*

Παραθέτουμε ένα ορισμό και το αντίστοιχο θεώρημα για P(x+1)

Ορισμός 1.

Για οποιονδήποτε μη αρνητικό ακέραιο n έστω

$$I_n = \{(i,j) \mid i,j \geq 0 \wedge i+j \leq n\}$$

και

$$A(x) = a_n x^n + \ldots + a_1 x + a_0,$$

το οποίο είναι ένα πολυώνυμο ακεραίων που ορίζουμε για $k \in \{0,\ldots,n\}$ και $(i,j) \in I_n$,

$$\begin{aligned} a_{-1,k} &= 0, \\ a_{k,-1} &= a_{n-k}, \\ a_{i,j} &= a_{i,j-1} + a_{i-1,j}, \end{aligned}$$

όπως φαίνεται στο σχήμα 1.

$$\begin{array}{ccccccc} & & 0 & 0 & 0 & 0 \\ & & \downarrow & \downarrow & \downarrow & \downarrow \\ a_n & \rightarrow & a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{n-1} & \rightarrow & a_{1,0} & a_{1,1} & a_{1,2} & \\ a_{n-2} & \rightarrow & a_{2,0} & a_{2,1} & & \\ a_{n-3} & \rightarrow & a_{3,0} & & & \end{array}$$

ΣΧΗΜΑ 1.

Στο θεώρημα 1 που ακολουθεί, το πρότυπο των προσθέσεων στο τρίγωνο του Pascal ,δηλαδή το $a_{i,j} = a_{i,j-1} + a_{i-1,j}$, μπορεί να χρησιμοποιηθεί για να υλοποιήσουμε την *Taylor Shift*.

Θεώρημα1.

Έστω n ένας μη αρνητικός ακέραιος, και έστω

$A(x) = a_n x^n + \ldots + a_1 x + a_0,$,το οποίο είναι ένα πολυώνυμο ακεραίων βάση του ορισμού που προηγήθηκε, ισχύει :

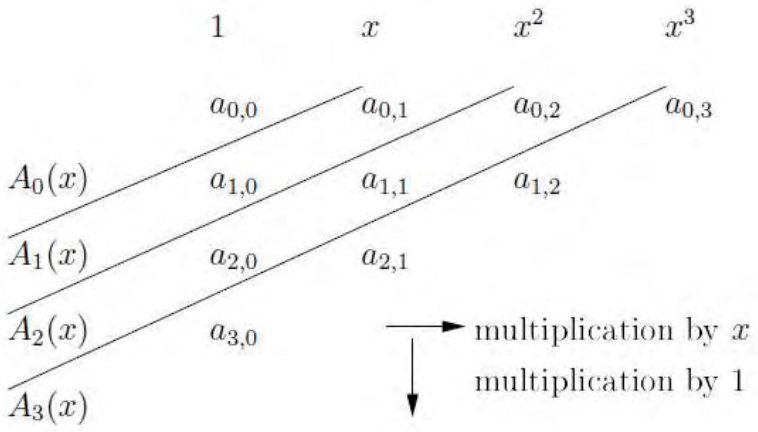
$$A(x+1) = \sum_{h=0}^{n} a_{n-h,h} x^h.$$

Απόδειξη 1.

Η απόδειξη ισχύει προφανώς για n=0. Έτσι μπορούμε να υποθέσουμε ότι ισχύει n>0.

Για κάθε $k \in \{0, \ldots, n\}$ έστω $A_k(x) = \sum_{h=0}^{k} a_{k-h,h} x^h$. Το σχήμα 2 δείχνει ότι οι συντελεστές του πολυωνύμου A_k αντιστοιχούν στην k^{στη} διαγώνιο του πίνακα του σχήματος 1. Τότε για όλα τα $k \in \{0, \ldots, n-1\}$, έχουμε

$A_{k+1}(x) = (x+1)A_k(x) + a_{n-(k+1)}$.Τώρα μία απλή επαγωγή στο k δείχνει ότι $A_k(x) = \sum_{h=0}^{k} a_{n-k+h}(x+1)^h$ για όλα τα $k \in \{0, \ldots, n\}$. Πιο συγκεκριμένα , $A_n(x) = \sum_{h=0}^{n} a_h (x+1)^h = A(x+1)$.



ΣΧΗΜΑ 2. Οι συντελεστές του πολυωνύμου $A_k(x)$ στην απόδειξη του θεωρήματος1 αντιστοιχούν στην k^{στη} διαγώνιο . Ο πολλαπλασιασμός του $A_k(x)$ με (x+1), μπορεί να διερμηνευτεί ως μια πρόσθεση που ακολουθεί μία μετακίνηση(“shift”) στα δεξιά και μία μετακίνηση προς τα κάτω.

Η διαφορά της ***P(x+α)*** με *P(x+1) είναι ότι*

Περιγραφή των δομικών στοιχείων

Υποπρόγραμμα 1^{ο} : "rootslist.h"

Η κλασική μορφή της λίστας FIFO, όπου ξεκινώντας από ένα στοιχείο-κεφαλή φτάνουμε στο στοιχείο-ουρά σαρώνοντας τόσα, όσο είναι βέβαια και το μέγεθος της.

Το πρόγραμμα αυτό το χρησιμοποιείται για να συνδέσει τα διαστήματα των ριζών που θα βρει αργότερα ο αλγόριθμος VAS(πρώτα των θετικών και μετά αρνητικών).

Τα δομικά στοιχεία της "rootslist" είναι :

A) "class rootslist" *: [κλάση]*

Κλάση που ενσωματώνει τις απαραίτητες συναρτήσεις που υλοποιούνται σε αυτό το αρχείο.

Πρώτα από όλα, ορίζεται ο αφηρημένος τύπος δεδομένων της λίστας **"struct node"** με δυο «αριθμούς» (που εξηγούνται στο αρχείο "Rational.h") ανά κόμβο και ένα δείκτη που δείχνει στο επόμενο στοιχείο.

```
1  #ifndef ROOTSLIST_H_
2  #define ROOTSLIST_H_
3
4  #include "Rational.h"
5  class rootslist{
6
7      struct node{
8          Rational l,r;
9          node* next;
10         node(Rational s, Rational t, node* n) : l(s), r(t), next(n) {}
11     };
12     node* head;
```

B) "rootslist()" : *[συνάρτηση]*

Βασικός "constractor" της κλάσης. Αρχικοποίηση της λίστας με την κεφαλή να είναι κενή.

```
13  public:
14      rootslist(){
15          head= NULL;
16      }
```

Γ) "~rootslist()" : *[συνάρτηση]*

Καταστροφή της λίστας("constractor") όταν αυτή δεν θα είναι πλέον χρήσιμη, στα πρότυπα του garbage collector.

```
22      ~rootslist(){
23          while(head){
24              node* temp(head);
25              head=head->next;
26              delete temp;
27          }
28      }
29  };
```

Δ) "bool empty()" *: [συνάρτηση]*

Λογική πράξη που επιστρέφει αληθές όταν η λίστα είναι άδεια.

```
18      bool empty() const { return ( head==NULL ); }
```

Ε) "void add(Rational, Rational, bool)" *: [συνάρτηση]*

Προσθήκη στοιχείου στο τέλος της λίστας. Η παράμετρος "bool" ορίζει εάν είναι θετική η ρίζα. Εάν δεν είναι, βάζουμε αρνητικό πρόσημο ως ρητό "fake" μπροστά από κάθε αριθμό.

```
32  void rootslist::add(Rational l, Rational r, bool positive){
33
34      node *neo;
35
36      if (positive){
37          neo = new node(l, r, NULL);
38      }
39
40      else {
41          Rational fake(-1,1);
42          Rational a,b;
43          a=r*fake;
44          b=l*fake;
45          neo = new node(a, b, NULL);
46      }
47      if( empty() ) head = neo;
48      else {
49          node *tmp= head;
50          while(tmp->next && (tmp->next->r<=l && tmp->next->r<=l)) tmp=tmp->next;
51          neo->next=tmp->next;
52          tmp->next=neo;
53      }
54  }
```

Στ) "void print()" *:[συνάρτηση]*

Εκτύπωση της λίστας ή εκτύπωση μηνύματος ότι δεν βρέθηκαν ρίζες (θετικές ή αρνητικές) . Σε ορισμένες περιπτώσεις, το αριστερό με το δεξί φράγμα ταυτίζονται, για αυτό και κάνουμε αυτόν τον έλεγχο.

```
57  void rootslist::print(){
58
59      if( empty() ){
60          cout<<"empty (Couldn't find any roots).";
61          return;
62      }
63      cout<<" {";
64      node* temp(head);
65      if(temp->l!=temp->r){
66          cout<<" [";
67          temp->l.disp();
68          cout<<" , ";
69          temp->r.disp();
70          cout<<"]";
71      }
72      else {
73          cout<<" {";
74          temp->l.disp();
75          cout<<"}";
76      }
77      temp=temp->next;
78      while (temp){
79          if(temp->l!=temp->r){
80              cout<<" U [";
81              temp->l.disp();
82              cout<<" , ";
83              temp->r.disp();
84              cout<<"]";
85          }
86          else{
87              cout<<" U {";
88              temp->l.disp();
89              cout<<"}";
90          }
91          temp=temp->next;
92      }
93      cout<<" }";
94  }
```

Υποπρόγραμμα 2^{ο} :"list.h"

Αντίστοιχα με την μορφή της λίστας FIFO, ορίζουμε και μία στοίβα LIFO όπου ξεκινώντας από ένα στοιχείο-κεφαλή φτάνουμε στο στοιχείο-ουρά σαρώνοντας τόσα

στοιχεία, όσο είναι βέβαια και το μέγεθος της, αλλά αυτήν την φορά η κεφαλή βρίσκεται στην κορυφή της.

Το πρόγραμμα αυτό το χρησιμοποιείται για να ορίσει την INTERVALSTACK. Κάθε στοιχείο της λίστας αυτής χρησιμεύει για να βρεθούν τα άνω και κάτω φράγματα σε κάθε ρίζα του πολυωνύμου, βάση του θεωρήματος του Vincent, που θα βρει αργότερα ο αλγόριθμος VAS.

Ορίζουμε ένα αφηρημένο τύπο δεδομένων "template"

Τα δομικά στοιχεία της "list.h" είναι :

A) "class Node" : *[κλάση]*

Ο αφηρημένος τύπος δεδομένων της λίστας ανά κόμβο με παραμέτρους που ορίζονται στο αρχείο "VAS.cpp" και ένας δείκτης που δείχνει στο επόμενο στοιχείο.

```
1  #ifndef LIST_H_
2  #define LIST_H_
3  template <class T>
4  class Node {
5
6      public:
7          T data;
8          Node<T> *next;
9          Node() {
10             next = NULL;
11         }
12 };
```

B) "class LinkedList" : *[κλάση]*

Κλάση που ενσωματώνει τις απαραίτητες συναρτήσεις που υλοποιούνται σε αυτό το αρχείο μαζί με ένα στιγμιότυπο του αφηρημένου τύπου δεδομένων "Node" και την δήλωση της μεταβλητής του μεγέθους της στοίβας.

```
15  template <class T>
16  class LinkedList {
17
18      public:
19          int size;
20          Node<T> *head;
21          LinkedList();
22          ~LinkedList();
23          int addHead(T x);
24          int removeHead(T &x);
25          bool isEmpty();
26  };
```

Γ) "LinkedList()" : *[συνάρτηση]*

Βασικός "constractor" της κλάσης. Αρχικοποίηση της στοίβας.

```
28  template <class T>
29  LinkedList<T>::LinkedList() {
30      head = NULL;
31      size=0;
32  }
```

Δ) "int addHead(T)" : *[συνάρτηση]*

Προσθήκη στοιχείου στην κορυφή της στοίβας. Ως παράμετρο παίρνει τον αφηρημένο τύπο "template".Όταν το νέο στοιχείο που προστίθεται είναι και το πρώτο, επιστρέφει 1 αλλιώς επιστέφει 0.

```
34  template <class T>
35  int LinkedList<T>::addHead(T x) {
36      Node<T> *newNode = new Node<T>;
37      newNode->data = x;
38      if (!head) {
39          size++;
40          head = newNode;
41          return 1;
42      }
43      newNode->next = head;
44      size++;
45      head = newNode;
46      return 0;
47  }
```

Ε) "int removeHead(T)" : *[συνάρτηση]*

Αφαίρεση στοιχείου από την κορυφή της στοίβας. Ως παράμετρο παίρνει τον αφηρημένο τύπο "template".Όταν η στοίβα είναι άδεια, επιστρέφει -1 αλλιώς επιστέφει 0.

```
49  template <class T>
50  int LinkedList<T>::removeHead(T &x) {
51      if (!head) return -1;
52      Node<T> *tmp = head;
53      x = head->data;
54      head = head->next;
55      size--;
56      if (size==0) head=NULL;
57      delete tmp;
58      return 0;
59  }
```

Στ) "bool isEmpty()" : *[συνάρτηση]*

Λογική πράξη που επιστρέφει αληθές όταν η στοίβα είναι άδεια.

```
61  template <class T>
62  bool LinkedList<T>::isEmpty() {
63      if (!head) return true;
64      return false;
65  }
```

Η) "~LinkedList()" : *[συνάρτηση]*

Καταστροφή της στοίβας όταν αυτή δεν θα είναι πλέον χρήσιμη, στα πρότυπα του garbage collector.

```
67  template <class T>
68  LinkedList<T>::~LinkedList() {
69      T x;
70      while(head) removeHead(x);
71  }
72
```

Υποπρόγραμμα 3^{ο} : "Rational.h"

Σε αυτό το αρχείο ορίζουμε έναν ειδικό τύπο αριθμό, τους «ρητούς». Κάθε αριθμός αποτελείται από τον αριθμητή και τον παρανομαστή του, απεικονίζεται δηλαδή με την μορφή κλάσματος. Θα τον παρουσιάζουμε από και πέρα σε αυτήν την μορφή, ακόμη και εάν δεν έχει δεκαδικά ψηφία υποδιαστολής, όπου σε αυτήν την περίπτωση ο παρανομαστής είναι 1. Ο λόγος που το κάνουμε αυτό είναι γιατί ορισμένοι αριθμοί έχουν απεριόριστα δεκαδικά ψηφία(π.χ. ο αριθμός «π») ή απλά περισσότερα από όσα αντέχει η C++. Έτσι, όταν κάνουμε στρογγυλοποίηση χάνουμε σημαντικά σε ακρίβεια. Σε έναν αλγόριθμο όπου μας ενδιαφέρει η καλύτερη δυνατή προσέγγιση μίας ρίζας μέσα σε ορισμένα φράγματα, κάτι τέτοιο δεν είναι επιθυμητό.

Το πρόγραμμα αυτό το χρησιμοποιείται για να ορίσει εκτός από αυτόν τον τύπο δεδομένων και τις πράξεις που εκτελούνται μεταξύ τους. Αυτές τις πράξεις χρησιμοποιεί αργότερα το αρχείο “Polynomial.h” που υλοποιεί ακόμη πιο σύνθετες με την μορφή συναρτήσεων.

Η διαφορά μιας συνάρτησης που έχει τον χαρακτηρισμό “const” , σε σύγκριση με αυτήν που δεν έχει, είναι ότι

Αρχικά καλούμε τα πακέτα :

1. “iostream.h”
2. “stdlib.h” και “stdio.h”

3. “gmp.h” : Το πακέτο αυτό είναι μία μαθηματική διεπαφή στα πρότυπα του “sympyt”. Αυτές οι βιβλιοθήκες έχουν έτοιμες συναρτήσεις που έχουν υλοποιηθεί με σκοπό αφενός να παρέχουν στο προγραμματιστή έτοιμες ρουτίνες με την καλύτερη δυνατή υλοποίηση σε ταχύτητα και χώρο, αλλά και να καλύψουν το κενό ορισμένων γλωσσών προγραμματισμού(όπως η C++) στην ακρίβεια και αποτελεσματικότητα μαθηματικών συναρτήσεων.

```
1  #ifndef RATIONAL_H_
2  #define RATIONAL_H_
3  #include <iostream>
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <gmp.h>
```

Τα δομικά στοιχεία της “Rational.h”είναι :

A) "class Rational " : *[κλάση]*

Κλάση που ενσωματώνει τις απαραίτητες συναρτήσεις που υλοποιούνται σε αυτό το αρχείο .Στην συνέχεια έχουμε την δήλωση του "namespace", της μεταβλητής "id", της μεταβλητής "idc" και της ρουτίνας "mpq_rt" του πακέτου "gmp" που ορίζει την μορφή ενός ρητού αριθμού.

```
 7  using namespace std;
 8
 9  int idc =0;
10
11  class Rational
12  {
13      private:
14
15          int id;
16      public:
17
18
19          mpq_t rational;
20
21          Rational();
22          Rational(long, long);
23
24          Rational(const Rational &o);
25
26          double toDouble();
27
28          Rational getRational(double );
29
30          void simplify();
31
32          void disp();
33          void disp() const;
34
35          void dispDouble();
36          void dispDouble() const;
37
38          ~Rational()
```

Έχουμε επίσης και τις υπερφορτώσεις των βασικών πράξεων μεταξύ αριθμών, αυτή την φορά όμως για ρητούς.

```
71          Rational &operator=(const Rational &);
72          Rational operator+(const Rational &);
73          Rational operator-(const Rational &);
74          Rational operator*(const Rational &);
75          Rational operator/(const Rational &);
76          Rational operator+(const Rational &)const ;
77          Rational operator-(const Rational &)const ;
78          Rational operator*(const Rational &)const ;
79          Rational operator/(const Rational &)const ;
80          Rational &operator+=(const Rational &);
81          Rational &operator-=(const Rational &);
82          Rational &operator*=(const Rational &);
83          Rational &operator/=(const Rational &);
84          bool operator==(const Rational &);
85          bool operator!=(const Rational &);
86          bool operator>(const Rational &);
87          bool operator<(const Rational &);
88          bool operator>=(const Rational &);
89          bool operator<=(const Rational &);
90          bool operator==(const Rational &) const;
91          bool operator!=(const Rational &) const;
92          bool operator>(const Rational &) const;
93          bool operator<(const Rational &) const;
94          bool operator>=(const Rational &) const;
95          bool operator<=(const Rational &) const;
96  };
```

Πιο αναλυτικά :

Β) "Rational(long, long) *: [συνάρτηση]*

Ο βασικός constractor της κλάσης Rational.Ως παραμέτρους παίρνει δύο αριθμούς τύπου long(δηλαδή 10ψηφίων) για αριθμητή και παρονομαστή, αντίστοιχα. Χρησιμοποιούμε τις συναρτήσεις "mpq_init" και "mpq_set_si" του πακέτου "gmp" για την αρχικοποίηση και την δημιουργία του ρητού αριθμού ,αντίστοιχα .Στην πρώτη ορίζουμε απλώς τον αριθμό.

```
 98
 99  Rational::Rational(){
100      id = idc++;
101
102      mpq_init(rational);
103  }
104
105  Rational::Rational(long a, long b){
106      id = idc++;
107      mpq_init(rational);
108      mpq_set_si(rational, a, b);
109  }
```

Γ) "Rational(const)" : *[συνάρτηση]*

Αντίστοιχα με την προηγούμενη συνάρτηση, ορίζουμε τις σταθερές ως ρητούς αριθμούς.

```
43          Rational(const Rational &o)
44          {
45              id = idc++;
46              mpq_init(rational);
47              mpq_set (rational, o.rational);
48          }
```

Δ) "double toDouble()" : *[συνάρτηση]*

Μετατροπή του τύπου ενός ρητού αριθμού από long σε double (ή αλλιώς long long).Χρησιμοποιούμε την συνάρτηση "mpq_get_d" του πακέτου "gmp" για να πάρουμε τον ρητό αριθμό. Αυτό το κάνουμε για να έχουμε μεγαλύτερη εμβέλεια σε πράξεις που θα γίνουν παρακάτω

```
111  double Rational::toDouble(){
112      return mpq_get_d(rational);
113  }
```

Ε) "void simplify()" : *[συνάρτηση]*

Απλοποίηση ενός ρητού αριθμού. Στην περίπτωση που ο ρητός μπορεί να απλοποιηθεί(γιατί αριθμητής και παρανομαστής έχουν μέγιστο κοινό διαιρέτη

μεγαλύτερο της μονάδας), καλούμε την "mpq_canonicalize" του πακέτου "gmp" που ουσιαστικά κάνει την δουλειά του "gcd"(μ.κ.δ).

```
121  void Rational::simplify(){
122      mpq_canonicalize(rational);
123  }
```

Στ) "void disp()" : *[συνάρτηση]*

Εκτύπωση των ρητών αριθμών. Χρησιμοποιούμε τις συναρτήσεις "mpq_t" , "mpq_init" και "mpq_clear" του πακέτου "gmp" για να ορίσουμε ,να αρχικοποιήσουμε και να καταστρέψουμε τον αριθμό "0". Στην συνέχεια με την "mpq_cmp" συγκρίνουμε τους αριθμούς με το μηδέν για να τους εκτυπώσουμε τελικά με αύξουσα σειρά με την "mpq_out_str".

```
125  void Rational::disp(){
126      mpq_t zero;
127      mpq_init(zero);
128      if(mpq_cmp (rational,zero)>=0)
129          cout <<"+";
130      mpq_out_str (NULL, 10, rational);
131      mpq_clear(zero);
132  }
133
134  void Rational::disp() const{
135    mpq_t zero;
136      mpq_init(zero);
137      if(mpq_cmp (rational,zero)>=0)
138          cout <<"+";
139      mpq_out_str (NULL, 10, rational);
140      mpq_clear(zero);
141  }
```

Η) "void dispDouble()" : *[συνάρτηση]*

Εκτύπωση των ρητών αριθμών που έχουν γίνει προηγουμένως "double". Καλούμε την συνάρτηση "mpq_get_d" του πακέτου "gmp" για να πάρουμε τον ρητό αριθμό.

```
144  void Rational::dispDouble(){
145      if(mpq_get_d(rational)>=0)
146          cout <<"+";
147      printf("%f",mpq_get_d(rational));
148  }
149
150  void Rational::dispDouble() const{
151      if(mpq_get_d(rational)>=0)
152          cout <<"+";
153      printf("%f",mpq_get_d(rational));
154  }
```

Θ) "Rational operator= (const Rational)" : *[συνάρτηση]*

Ορισμός της πράξης ανάθεσης .Χρησιμοποιούμε την συνάρτηση "mpq_set" του πακέτου "gmp" για να πάρουμε τον ρητό και να τον επιστρέψουμε στο δείκτη "*this".

```
157  Rational &Rational::operator=(const Rational &passed){
158      mpq_set (rational, passed.rational);
159      return *this;
160  }
```

Ι) "Rational operator+ (const Rational)" : *[συνάρτηση]*

Ορισμός της πράξης πρόσθεσης .Θέτουμε έναν αριθμό "Result" ως ρητό. Χρησιμοποιούμε την συνάρτηση "mpq_add" του πακέτου "gmp" για να προσθέσουμε δύο ρητούς μεταξύ τους(οι δύο τελευταίες παράμετροι) και να τους αναθέσουμε στον "Result" τον οποίο και τελικά θα επιστρέψουμε.

```
162  Rational Rational::operator+(const Rational & passed){
163      Rational Result = Rational();
164      mpq_add (Result.rational, rational, passed.rational);
165      return Result;
166  }
167
168  Rational Rational::operator+( const Rational & passed) const{
169      Rational Result = Rational();
170      mpq_add (Result.rational, rational, passed.rational);
171      return Result;
172  }
```

Ια) "Rational operator- (const Rational)" : *[συνάρτηση]*

Ορισμός της πράξης αφαίρεσης. Θέτουμε έναν αριθμό "Result" ως ρητό. Χρησιμοποιούμε την συνάρτηση "mpq_sub" του πακέτου "gmp" για να προσθέσουμε δύο ρητούς μεταξύ τους(οι δύο τελευταίες παράμετροι) και να τους αναθέσουμε στον "Result" τον οποίο και τελικά θα επιστρέψουμε.

```
174  Rational Rational::operator-(const Rational & passed){
175      Rational Result = Rational();
176      mpq_sub (Result.rational, rational, passed.rational);
177      return Result;
178  }
179
180  Rational Rational::operator-(const Rational& passed) const{
181      Rational Result = Rational();
182      mpq_sub (Result.rational, rational, passed.rational);
183      return Result;
184  }
```

Ιβ) "Rational operator* (const Rational)" : *[συνάρτηση]*

Ορισμός της πράξης πολλαπλασιασμού .Θέτουμε έναν αριθμό "Result" ως ρητό. Χρησιμοποιούμε την συνάρτηση "mpq_mul" του πακέτου "gmp" για να πολλαπλασιάσουμε δύο ρητούς μεταξύ τους(οι δύο τελευταίες παράμετροι) και να τους αναθέσουμε στον "Result" τον οποίο και τελικά θα επιστρέψουμε.

```
186  Rational Rational::operator*(const Rational & passed){
187      Rational Result = Rational();
188      mpq_mul (Result.rational, rational, passed.rational);
189      return Result;
190  }
191
192  Rational Rational::operator*(const Rational& passed) const{
193      Rational Result = Rational();
194      mpq_mul (Result.rational, rational, passed.rational);
195      return Result;
196  }
```

Ιγ) "Rational operator/ (const Rational)" : *[συνάρτηση]*

Ορισμός της πράξης διαίρεσης. Θέτουμε έναν αριθμό "Result" ως ρητό. Χρησιμοποιούμε την συνάρτηση "mpq_div" του πακέτου "gmp" για να διαιρέσουμε δύο ρητούς μεταξύ τους(οι δύο τελευταίες παράμετροι) και να τους αναθέσουμε στον "Result" τον οποίο και τελικά θα επιστρέψουμε.

```
198 Rational Rational::operator/(const Rational & passed){
199     Rational Result = Rational();
200     mpq_div (Result.rational, rational, passed.rational);
201     return Result;
202 }
203 Rational Rational::operator/(const Rational& passed) const{
204     Rational Result = Rational();
205     mpq_div (Result.rational, rational, passed.rational);
206     return Result;
207 }
```

Ιδ) "*Rational operator+= (const Rational)" : *[συνάρτηση]*

Ορισμός της πράξης πρόσθεσης . Η διαφορά με την προηγούμενη είναι ότι ταυτίζονται ο πρώτος όρος της πράξης και η μεταβλητή αποθήκευσης. Επιστρέφουμε το αποτέλεσμα σε δείκτη.

```
209 Rational &Rational::operator+=(const Rational & passed){
210     mpq_add (rational, rational, passed.rational);
211     return *this;
212 }
```

Ιε) "*Rational operator-= (const Rational)" : *[συνάρτηση]*

Ορισμός της πράξης αφαίρεσης. Η διαφορά με την προηγούμενη είναι ότι ταυτίζονται ο πρώτος όρος της πράξης και η μεταβλητή αποθήκευσης. Επιστρέφουμε το αποτέλεσμα σε δείκτη.

```
214 Rational &Rational::operator-=(const Rational & passed){
215     mpq_sub (rational, rational, passed.rational);
216     return *this;
217 }
```

Ιστ) "*Rational operator*= (const Rational)" : *[συνάρτηση]*

Ορισμός της πράξης πολλαπλασιασμού. Η διαφορά με την προηγούμενη είναι ότι ταυτίζονται ο πρώτος όρος της πράξης και η μεταβλητή αποθήκευσης. Επιστρέφουμε το αποτέλεσμα σε δείκτη.

```
219  Rational &Rational::operator*=(const Rational & passed){
220      mpq_mul (rational, rational, passed.rational);
221      return *this;
222  }
```

Ιζ) "*Rational operator/(const Rational)" : *[συνάρτηση]*

Ορισμός της πράξης διαίρεσης. Η διαφορά με την προηγούμενη είναι ότι ταυτίζονται ο πρώτος όρος της πράξης και η μεταβλητή αποθήκευσης. Επιστρέφουμε το αποτέλεσμα σε δείκτη.

```
224  Rational &Rational::operator/=(const Rational & passed){
225      mpq_div (rational, rational, passed.rational);
226      return *this;
227  }
```

Ιη) "bool operator== (const Rational)" : *[συνάρτηση]*

Λογική πράξη σύγκρισης για ισότητα. Χρησιμοποιούμε την συνάρτηση "mpq_equal" του πακέτου "gmp".

```
229  bool Rational::operator==(const Rational & passed){
230      if(mpq_equal (rational, passed.rational))
231          return true;
232      else
233          return false;
234  }
235
236  bool Rational::operator==(const Rational &passed) const {
237      if(mpq_equal (rational, passed.rational))
238          return true;
239      else
240          return false;
241  }
```

Ιθ) "bool operator!= (const Rational)" : *[συνάρτηση]*

Λογική πράξη σύγκρισης για ανισότητα. Χρησιμοποιούμε την συνάρτηση "mpq_equal" του πακέτου "gmp".

```
243  bool Rational::operator!=(const Rational & passed){
244      if(mpq_equal (rational, passed.rational))
245          return false;
246      else
247          return true;
248  }
249
250  bool Rational::operator!=(const Rational &passed) const {
251      if(mpq_equal (rational, passed.rational))
252          return false;
253      else
254          return true;
255  }
```

Κα) "bool operator> (const Rational)" : *[συνάρτηση]*

Λογική πράξη σύγκρισης για μεγαλύτερο. Χρησιμοποιούμε την συνάρτηση "mpq_cmp" του πακέτου "gmp".

```
257  bool Rational::operator>(const Rational & passed){
258      if(mpq_cmp (rational, passed.rational)>0)
259          return true;
260      else
261          return false;
262  }
263
264  bool Rational::operator>(const Rational &passed) const {
265      if(mpq_cmp (rational, passed.rational)>0)
266          return true;
267      else
268          return false;
269  }
```

Κβ) "bool operator< (const Rational)" : *[συνάρτηση]*

Λογική πράξη σύγκρισης για μικρότερο. Χρησιμοποιούμε την συνάρτηση "mpq_cmp" του πακέτου "gmp".

```
271 bool Rational::operator<(const Rational & passed){
272     if(mpq_cmp(rational, passed.rational)<0)
273         return true;
274     else
275         return false;
276 }
277
278 bool Rational::operator<(const Rational &passed) const {
279     if(mpq_cmp(rational, passed.rational)<0)
280         return true;
281     else
282         return false;
283 }
```

Κγ) ""bool operator>= (const Rational)" : *[συνάρτηση]*

Λογική πράξη σύγκρισης για μεγαλύτερο ή ίσο. Χρησιμοποιούμε την συνάρτηση "mpq_cmp" του πακέτου "gmp".

```
285 bool Rational::operator>=(const Rational & passed){
286     if(mpq_cmp (rational, passed.rational)>=0)
287         return true;
288     else
289         return false;
290 }
291
292 bool Rational::operator>=(const Rational &passed) const
293     if(mpq_cmp (rational, passed.rational)>=0)
294         return true;
295     else
296         return false;
297 }
```

Κδ) "bool operator<= (const Rational)" : *[συνάρτηση]*

Λογική πράξη σύγκρισης για μικρότερο ή ίσο. Χρησιμοποιούμε την συνάρτηση "mpq_cmp" του πακέτου "gmp".

```
299 bool Rational::operator<=(const Rational & passed){
300     if(mpq_cmp (rational, passed.rational)<=0)
301         return true;
302     else
303         return false;
304 }
305
306 bool Rational::operator<=(const Rational &passed) const {
307     if(mpq_cmp (rational, passed.rational)<=0)
308         return true;
309     else
310         return false;
311 }
```

Υποπρόγραμμα 4^{ο} : "Polynomial.h"

Σε αυτό το αρχείο, βασιζόμενοι στα βασικές πράξεις του αρχείου “Rational.h”, υλοποιούμε όλες εκείνες τις συναρτήσεις οι οποίες είναι απαραίτητες για να ολοκληρώσουμε στην συνέχεια τα βήματα του αλγορίθμου VAS και τις οποίες θα καλέσουμε στο αρχείο “VAS.cpp”.

Όπως θα παρατηρήσετε, το πρόγραμμα αυτό είναι το πιο «φορτωμένο» από συναρτήσεις. Έτσι έχουμε έτοιμες όλες τις απαιτούμενες ρουτίνες που επεξεργάζονται με διάφορους τρόπους πολυώνυμα για το αρχείο “VAS.cpp”. Παρουσιάσουμε ταυτόχρονα, εναλλακτικές υλοποιήσεις συγκεκριμένων ρουτινών ανάλογα με τον τύπο δεδομένων(“Rational”, “double”, “long”) .

Αρχικά καλούμε τα πακέτα :

1. “fstream.h”
2. “string.h”
3. “vector”
4. “Rational.h” :από όπου καλούμε τις βασικές πράξεις μεταξύ ρητών αριθμών
5. “list.h” :

```
#ifndef POLYNOMIAL_H_
#define POLYNOMIAL_H_

#include <fstream>
#include <string.h>
#include <vector>
#include "Rational.h"
#include "list.h"

```

Στην συνέχεια δηλώνουμε τους σταθερούς ρητούς αριθμούς «μηδέν», «ένα» και «άπειρο» αλλά και το “namespace”, “theid”

```
const Rational zero(0,1);
const Rational one(1,1);
const Rational inf(333333333,1);
using namespace std;

int theid = 0;
```

Τα δομικά στοιχεία της “Polynomial.h”είναι :

A) "class IndexInTable" : *[κλάση]*

Ειδική κλάση για αναζήτηση εκθέτη σε πίνακα.

```
17  class IndexInTable {
18   public:
19         bool found;
20         int index;
21  };
22
```

B) " class Polynomial" : *[κλάση]*

Κλάση που ενσωματώνει τις απαραίτητες συναρτήσεις που υλοποιούνται σε αυτό το αρχείο . Επίσης έχουμε τις δηλώσεις των "maxTerms"(σταθερή μεταβλητή για μέγιστο πλήθος όρων πολυωνύμου), "numTerms" και "exp[maxTerms]" (για τον μεταβλητό πλήθος όρων και αντίστοιχων εκθετών), "coeff"(δείκτης που περνά σαν παράμετρος σε συναρτήσεις που επεξεργάζονται πολυώνυμα) και "id".

```
22
23  class Polynomial{
24
25   public:
26
27         static const int maxTerms = 400;
28
29         Polynomial();
30         Polynomial(const Polynomial &o);
31
32         void MergeSort(int [], int , int);
33         void Merge(int [], int , int , int );
34         void QuickSort(int [], int , int);
35         void BubbleSort(int []);
36         void InsertionSort(int [],int);
37
38         Polynomial operator+(const Polynomial &) const;
39         Polynomial operator-(const Polynomial &) const;
40         Polynomial operator*(const Polynomial &) const;
41         Polynomial operator/(const Polynomial &) const;
42         Polynomial operator%(const Polynomial &) const;
43         Polynomial operator^(int) const;
44         Polynomial &operator=(const Polynomial &);
45         Polynomial &operator+=(const Polynomial &);
46         Polynomial &operator-=(const Polynomial &);
47         Polynomial &operator*=(const Polynomial &);
48
49         bool isnot(int [], int);
50         bool operator==(const Polynomial &) const;
51
52         Polynomial derivative() const;
53
54         Polynomial fneg() const;
55
56         Polynomial monic() const;
57
58         Polynomial gcd(const Polynomial &, const Polynomial &);
```

```
60          Polynomial xplusA(Rational);
61          Polynomial xplusA2(Rational);
62
63          Polynomial xtimesA(Rational);
64
65          Polynomial xtoMdivx();
66
67          Polynomial sqf() const;
68
69          void enterTerms();
70          void enterTermsDouble();
71          void enterTermsRational();
72
73          int load();
74          int loadDouble();
75          int loadRational();
76
77          void refress();
78
79          void zeroCoef();
80
81          void mononym(int, const Rational &);
82
83          int sgc();
84
85          void printPolynomial() const;
86          void printPolynomialDouble() const;
87
88          int maxDegIndex() const;
89
90          int getNumTerms() const;
91
92          int getTermExp(int) const;
93
94          Rational getTermCoeff(int) const;
```

```
96           void setCoeff(int, Rational);
97
98           int getDegree() const;
99
100          int numTerms;
101
102          int exp[maxTerms];
103
104          Rational *coeff;
105
106          static void polyCombine(Polynomial &);
107          static void polyCombine2(Polynomial &);
108          static void polyCombine3(Polynomial &);
109
110          static bool constantPoly(Polynomial &w);
111
112          int id;
113
114          ~Polynomial();
115  };
```

Γ) "Polynomial()" : *[συνάρτηση]*

Ο βασικός constractor της κλάσης Polynomial.Αρχικοποιούμε με μηδέν τα "coeff"(συντελεστές), "exp"(αντίστοιχοι εκθέτες) και "numTerms"(πλήθος όρων).

```
117  Polynomial::Polynomial(){
118
119      theid++;
120      id = theid;
121
122      coeff = new Rational[maxTerms];
123
124      for(int t = 0; t < maxTerms; t++){
125        coeff[t] = zero;
126        exp[t] = 0;
127      }
128      numTerms = 0;
129  }
```

Δ) "Polynomial(const Polynomial *)" : *[συνάρτηση]*

Ο βασικός constractor της κλάσης Polynomial.Η διαφορά με πριν είναι ότι παίρνει σαν παράμετρο ένα δείκτη.

```
131  Polynomial::Polynomial(const Polynomial &o){
132
133      theid++;
134      id = theid;
135
136      coeff = new Rational[maxTerms];
137
138      for(int t = 0; t < maxTerms; t++){
139          coeff[t] = o.coeff[t];
140          exp[t] = o.exp[t];
141      }
142      numTerms = o.numTerms;
143  }
```

Ε) "Mergesort" : *[συνάρτηση]*

Αλγόριθμος ταξινόμησης για τους όρους του πολυωνύμου, από τον μεγαλύτερο στον μικρότερο.

```
145 void Polynomial::MergeSort(int a[], int low, int high){
146
147     int mid;
148
149     if(low<high){
150         mid=(low+high)/2;
151         MergeSort(a,low,mid);
152         MergeSort(a,mid+1,high);
153         Merge(a,low,high,mid);
154     }
155     return;
156 }
```

Στ) "Merge" : *[συνάρτηση]*

Αλγόριθμος ταξινόμησης για τους όρους του πολυωνύμου, από τον μεγαλύτερο στον μικρότερο.

```
158 void Polynomial::Merge(int a[], int low, int high, int mid){
159
160     int i, j, k, c[50];
161     i=low;
162     j=mid+1;
163     k=low;
164
165     while((i<=mid)&&(j<=high)){
166         if(a[i]<a[j]){
167             c[k]=a[i];
168             k++;
169             i++;
170         }
171         else{
172             c[k]=a[j];
173             k++;
174             j++;
175         }
176     }
177     while(i<=mid){
178         c[k]=a[i];
179         k++;
180         i++;
181     }
182     while(j<=high){
183         c[k]=a[j];
184         k++;
185         j++;
186     }
187     for(i=low;i<k;i++){
188         a[i]=c[i];
189     }
190 }
```

Ζ) "QuickSort" : *[συνάρτηση]*

Αλγόριθμος ταξινόμησης για τους όρους του πολυωνύμου, από τον μεγαλύτερο στον μικρότερο.

```
192 void Polynomial::QuickSort(int arrayData[], int left, int right) {
193
194     int i, j, p, temp;
195     i = left;
196     j = right;
197     p = arrayData[(left+right)/2];
198
199     do {
200       while(arrayData[i] > p && i < right)  i++;
201       while(p > arrayData[j] && j > left)  j--;
202       if(i <=j) {
203           temp = arrayData[i];
204           arrayData[i] = arrayData[j];
205           arrayData[j] = temp;
206           i++;
207           j--;
208         }
209     } while (i <= j);
210     if(left <j)
211       QuickSort(arrayData, left, j);
212     if(i < right)
213       QuickSort(arrayData, i, right);
214 }
```

Η) "BubbleSort" : *[συνάρτηση]*

Αλγόριθμος ταξινόμησης για τους όρους του πολυωνύμου, από τον μεγαλύτερο στον μικρότερο.

```
216 void Polynomial::BubbleSort(int array[]){
217
218     int i,j;
219
220     for(i=2;i<maxTerms;i++){
221         for(j=1;j<i;j++){
222             if(array[i]>array[j]){
223                 int temp=array[i];
224                 array[i]=array[j];
225                 array[j]=temp;
226             }
227         }
228     }
229 }
```

Θ) "InsertionSort" : *[συνάρτηση]*

Αλγόριθμος ταξινόμησης για τους όρους του πολυωνύμου, από τον μεγαλύτερο στον μικρότερο.

```
231 void Polynomial::InsertionSort(int x[],int length){
232
233     int key,i;
234
235     for(int j=2;j<length;j++)
236     {
237         key=x[j];
238         i=j-1;
239         while(x[i]>key && i>=0)
240         {
241                 x[i+1]=x[i];
242             i--;
243         }
244         x[i+1]=key;
245     }
246 }
```

I) "Polynomial operator+ (const Polynomial *)" : *[συνάρτηση]*

Ορισμός της πράξης αθροίσματος μεταξύ πολυωνύμων. Ως παράμετρο παίρνει δείκτη ως προς ένα πολυώνυμο, το οποίο συγκρίνεται με .Η πράξη αυτή γίνεται όρο με όρο, εκθέτη με εκθέτη και όταν τελειώσει το πρώτο πολυώνυμο απλώς παραθέτουμε το δεύτερο. Τέλος, επειδή το πολυώνυμο που προκύπτει("temp") είναι αταξινόμητο, καλούμε την "polyCombin3" που μελετούμε παρακάτω.

```
248 Polynomial Polynomial::operator+(const Polynomial &r) const {
249
250     Polynomial temp;
251     bool expExists;
252     int s;
253
254     temp.coeff[0] = coeff[0] + r.coeff[0];
255
256     for(s = 1; (s < maxTerms) && (r.exp[s] != 0); s++){
257         temp.coeff[s] = r.coeff[s];
258         temp.exp[s] = r.exp[s];
259     }
260     for(int x = 1; x < maxTerms; x++){
261         expExists = false;
262
263         for(int t = 1; (t < maxTerms) && (!expExists); t++)
264           if(exp[x] == temp.exp[t]){
265             temp.coeff[t] += coeff[x];
266             expExists = true;
267           }
268         if(!expExists){
269           temp.exp[s] = exp[x];
270           temp.coeff[s] += coeff[x];
271           s++;
272         }
273     }
274     polyCombine3(temp);
275     return temp;
276 }
```

Ιι) "Polynomial operator- (const Polynomial *)" : *[συνάρτηση]*

Ορισμός της πράξης αφαίρεσης μεταξύ πολυωνύμων. Ως παράμετρο παίρνει δείκτη ως προς ένα πολυώνυμο, το οποίο συγκρίνεται με .Η πράξη αυτή γίνεται όρο με όρο, εκθέτη με εκθέτη και όταν τελειώσει το πρώτο πολυώνυμο απλώς παραθέτουμε το δεύτερο.

```
278 Polynomial Polynomial::operator-(const Polynomial &r) const {
279
280     Polynomial temp;
281     bool expExists;
282     int s;
283
284     temp.coeff[0] = coeff[0] - r.coeff[0];
285
286     for(s = 1; (s < maxTerms) && (exp[s] != 0); s++){
287         temp.coeff[s] = coeff[s];
288         temp.exp[s] = exp[s];
289     }
290     for (int x = 1; x < maxTerms; x++){
291         expExists = false;
292
293         for(int t = 1; (t < maxTerms) && (!expExists); t++)
294
295           if(r.exp[x] == temp.exp[t]){
296               temp.coeff[t] -= r.coeff[x];
297               expExists = true;
298           }
299           if(!expExists){
300               temp.exp[s] = r.exp[x];
301               temp.coeff[s] -= r.coeff[x];
302               s++;
303           }
304     }
305     return temp;
306 }
```

Ιβ) "Polynomial operator* (const Polynomial *)" : *[συνάρτηση]*

Ορισμός της πράξης πολλαπλασιασμού μεταξύ πολυωνύμων. Ως παράμετρο παίρνει δείκτη ως προς ένα πολυώνυμο, το οποίο συγκρίνεται με .Η πράξη αυτή γίνεται όρο με όρο, εκθέτη με εκθέτη και όταν τελειώσει το πρώτο πολυώνυμο απλώς παραθέτουμε το δεύτερο. Τέλος, επειδή το πολυώνυμο που προκύπτει ("temp") είναι αταξινόμητο, καλούμε την "polyCombin2" που μελετούμε παρακάτω.

```
308 Polynomial Polynomial::operator*(const Polynomial &r) const {
309
310     Polynomial temp;
311     int s = 1;
312
313     for(int x = 0; (x < maxTerms) && (x == 0 || coeff[x] != zero); x++)
314     {
315         for(int y = 0; (y < maxTerms) && (y == 0 || r.coeff[y] != zero); y ++)
316         {
317             if(coeff[x] * r.coeff[y] != zero)
318             {
319                 temp.refress();
320                 if((exp[x] == 0) && (r.exp[y] == 0))
321                 {
322                     temp.coeff[0] += coeff[x] * r.coeff[y];
323                 }
324                 else
325                 {
326                     temp.coeff[s] = coeff[x] * r.coeff[y];
327                     temp.exp[s] = exp[x] + r.exp[y];
328                     s++;
329                 }
330             }
331         }
332     }
333     polyCombine2(temp);
334     return temp;
335 }
```

Ιγ) "Polynomial operator/ (const Polynomial *)" : *[συνάρτηση]*

Ορισμός της πράξης διαίρεσης μεταξύ πολυωνύμων. Ως παράμετρο παίρνει δείκτη ως προς ένα πολυώνυμο, το οποίο συγκρίνεται με .Η πράξη αυτή γίνεται όρο με όρο, εκθέτη με εκθέτη και όταν τελειώσει το πρώτο πολυώνυμο απλώς παραθέτουμε το δεύτερο. Χρησιμοποιούμε τις μεταβλητές "numDegree", "denDegree", τα οποία αποτελούν τον εκθέτη του διαιρέτη, και διαιρετέου αντίστοιχα και ανάλογα με την σύγκριση τους , διακρίνουμε περιπτώσεις.

```
337 Polynomial Polynomial::operator /(const Polynomial &r) const {
338
339     int numDegree = getDegree(), denDegree = r.getDegree();
340     Polynomial ret;
341
342     if(numDegree< denDegree)
343     {
344         ret.zeroCoef();
345         return ret;
346     }
347     if(numDegree == denDegree)
348     {
349         ret.mononym(0,this->getTermCoeff(numDegree)/r.getTermCoeff(numDegree));
350         return ret;
351     }
352
353     Rational *num, *den,*piliko;
354     num = new Rational[numDegree+1];
355     den = new Rational[denDegree+1];
356     piliko = new Rational[numDegree-denDegree+1];
357
358     for(int i = 0; i <= numDegree; i++)
359         num[i] = getTermCoeff(i);
360     for(int i = 0; i <= denDegree; i++)
361         den[i] = r.getTermCoeff(i);
362     for(int i = numDegree;i >=denDegree; i--)
363     {
364         piliko[i-denDegree] = num[i]/den[denDegree];
365         for(int j = 0; j <= denDegree; j++)
366         {
367             num[i-j] -=  piliko[i-denDegree]*den[denDegree-j];
368         }
369     }
370     ret.zeroCoef();
371     for(int i = 0; i <= numDegree-denDegree; i++)
372     {
373         ret.coeff[i] = piliko[i];
374         ret.exp[i] = i;
375     }
376     return ret;
377 }
```

Ιδ) "Polynomial operator% (const Polynomial *)" : *[συνάρτηση]*

Ορισμός της πράξης διαίρεσης μεταξύ πολυωνύμων όπου επιστρέφεται το υπόλοιπο. Ως παράμετρο παίρνει δείκτη ως προς ένα πολυώνυμο, το οποίο συγκρίνεται με .Η πράξη αυτή γίνεται όρο με όρο, εκθέτη με εκθέτη και όταν τελειώσει το πρώτο πολυώνυμο απλώς παραθέτουμε το δεύτερο. Χρησιμοποιούμε τις μεταβλητές "numDegree", "denDegree", τα οποία αποτελούν τον εκθέτη του διαιρέτη, και διαιρετέου αντίστοιχα και ανάλογα με την σύγκριση τους , διακρίνουμε περιπτώσεις.

```
379 Polynomial Polynomial::operator%(const Polynomial &r) const {
380
381     int numDegree = this->getDegree(), denDegree = r.getDegree();
382     Polynomial ret;
383
384     if(numDegree< denDegree)
385
386     {
387         ret.zeroCoef();
388         return ret;
389     }
390     if(numDegree == denDegree)
391     {
392         ret.mononym(0,this->getTermCoeff(numDegree)/r.getTermCoeff(numDegree));
393         return ret;
394     }
395
396     Rational *num, *den,*piliko;
397     num = new Rational[numDegree+1];
398     den = new Rational[denDegree+1];
399     piliko = new Rational[numDegree-denDegree+1];
400
401     for(int i = 0; i <= numDegree; i++)
402         num[i] = getTermCoeff(i);
403     for(int i = 0; i <= denDegree; i++)
404         den[i] = r.getTermCoeff(i);
405     for(int i = numDegree;i >=denDegree; i--)
406     {
407         piliko[i-denDegree] = num[i]/den[denDegree];
408         for(int j = 0; j <= denDegree; j++)
409         {
410             num[i-j] -=  piliko[i-denDegree]*den[denDegree-j];
411         }
412     }
413     ret.zeroCoef();
414     for(int i = 0; i < denDegree; i++)
415     {
416         ret.coeff[i] = num[i];
417         ret.exp[i] = i;
418     }
419     return ret;
420 }
```

Ιε) " Polynomial operator^ (int)" : *[συνάρτηση]*

Ορισμός της πράξης ύψωσης σε μία δύναμη ενός πολυωνύμου. Ως παράμετρο παίρνει τον δείκτη ύψωσης. Η διαδικασία επαναλαμβάνεται σε δύο στάδια. Πρώτα όσο a*2<e όπου . Μετά όσο a<e έχουμε

```
422 Polynomial Polynomial::operator^(int e) const{
423
424     Polynomial temp,f;
425     int a=1;
426
427     temp = *this;
428     f=*this;
429
430     while(a*2<e){
431         temp=temp*temp;
432         a*=2;
433     }
434     while(a<e){
435         temp=temp*f;
436         a++;
437     }
438     return temp;
439 }
```

Ιστ) "Polynomial operator= (const Polynomial *)" : *[συνάρτηση]*

Ορισμός της πράξης ανάθεσης σε ένα πολυώνυμο. Ως παράμετρο παίρνει δείκτη ως προς ένα πολυώνυμο, το οποίο ανατίθεται όρο με όρο σε ένα «νέο» που τελικά και επιστρέφεται σε δείκτη.

```
441 Polynomial &Polynomial::operator=(const Polynomial &r) {
442
443     exp[0] = r.exp[0];
444     coeff[0] = r.coeff[0];
445
446     for(int s = 1; s < maxTerms; s++){
447         if(r.exp[s] != 0){
448             exp[s] = r.exp[s];
449             coeff[s] = r.coeff[s];
450         }
451         else{
452             if(exp[s] == 0) break;
453             exp[s] = 0;
454             coeff[s] = zero;
455         }
456     }
457     return *this;
458 }
```

Ιζ) "Polynomial operator+= (const Polynomial *)" : *[συνάρτηση]*

Ορισμός της πράξης πρόσθεσης. Η διαφορά με την προηγούμενη είναι ότι ταυτίζονται ο πρώτος όρος της πράξης και η μεταβλητή αποθήκευσης. Το αποτέλεσμα επιστέφεται σε ένα δείκτη.

```
460 Polynomial &Polynomial::operator+=(const Polynomial &r){
461     *this = *this + r;
462     return *this;
463 }
```

Ιη) "Polynomial operator-= (const Polynomial *)" : *[συνάρτηση]*

Ορισμός της πράξης αφαίρεσης. Η διαφορά με την προηγούμενη είναι ότι ταυτίζονται ο πρώτος όρος της πράξης και η μεταβλητή αποθήκευσης. Το αποτέλεσμα επιστέφεται σε ένα δείκτη.

```
465 Polynomial &Polynomial::operator-=(const Polynomial &r)
466     *this = *this -r;
467     return *this;
468 }
```

Ιθ) "Polynomial operator*= (const Polynomial *)" : *[συνάρτηση]*

Ορισμός της πράξης πολλαπλασιασμού. Η διαφορά με την προηγούμενη είναι ότι ταυτίζονται ο πρώτος όρος της πράξης και η μεταβλητή αποθήκευσης. Το αποτέλεσμα επιστέφεται σε ένα δείκτη.

```
470 Polynomial &Polynomial::operator*=(const Polynomial &r){
471     (*this) = (*this) * r;
472     return (*this);
473 }
```

Κ) "Polynomial operator== (const Polynomial *)" : *[συνάρτηση]*

Λογική πράξη για επαλήθευση ισότητας.

```
475 bool Polynomial::operator==(const Polynomial &r) const {
476
477     Polynomial temp;
478     bool equals,test=true;
479
480     equals=(coeff[0] == r.coeff[0]);
481
482     for(int x = 1; x < maxTerms; x++){
483             test=false;
484       for(int t = 1; (t < maxTerms) && (equals); t++){
485         if(exp[x] == r.exp[t]){
486             equals=equals && (coeff[x] == r.coeff[t]);
487             test=true;
488         }
489       }
490       if (!test) return false;
491      }
492     return equals;
493 }
```

Κα) "bool isnot(int [] ,int)" : *[συνάρτηση]*

Λογική πράξη για επαλήθευση εύρεσης όρου σε ένα πολυώνυμο.

```
495  bool Polynomial::isnot(int a[], int b){
496
497      for (int i=0;i<maxTerms;i++){
498          if (b==a[i]) return false;
499      }
500      return true;
501  }
```

Κβ) " Polynomial derivative () const " : *[συνάρτηση]*

Εύρεση και επιστροφής της παραγώγου ενός πολυωνύμου. Χρησιμοποιούμε το "fake".

```
504  Polynomial Polynomial::derivative() const {
505
506      Polynomial temp;
507      temp.zeroCoef();
508      Rational fake(1,1);
509      int s;
510
511      for(s = 1; (s < maxTerms); s++){
512          if(exp[s]>1){
513              fake=Rational(exp[s],1);
514              temp.coeff[s] = coeff[s]*fake;
515              temp.exp[s] = exp[s]-1;
516          }
517          else if(exp[s]==1){
518              fake=Rational(exp[s],1);
519              temp.coeff[0] += coeff[s]*fake;
520              temp.exp[0] = 0;
521          }
522      }
523      return temp;
524  }
```

Κγ) " Polynomial fneg () const " : *[συνάρτηση]*

Εύρεση και επιστροφή του αντιθέτου(ως προς τα πρόσημα) ενός πολυωνύμου. Χρησιμοποιούμε το "fake" για να κάνουμε τον πολλαπλασιασμό με «-1».

```
526 Polynomial Polynomial::fneg() const {
527
528     Polynomial temp=*this;
529     Rational fake(-1,1);
530     int s;
531
532     for(s = 1; (s < maxTerms); s++){
533         if(exp[s]%2!=0){
534             temp.coeff[s] = coeff[s]*fake;
535         }
536     }
537     return temp;
538 }
```

Κδ) " Polynomial monic () const" : *[συνάρτηση]*

Εύρεση και επιστροφής ενός πολυωνύμου που γίνεται μονικό. Χρησιμοποιούμε το "inv" .Η συνάρτηση αυτή χρησιμοποιείται στην λειτουργία του μεγίστου κοινού διαιρέτη("gcd").

```
540 Polynomial Polynomial::monic() const{
541
542     Polynomial f=*this;
543     int deg=exp[f.maxDegIndex()];
544     Rational inv(0,1);
545
546     for(int x = 0; x < maxTerms; x++){
547         if(f.exp[x]==deg) {
548             inv+=f.coeff[x];
549         }
550     }
551     for(int x = 0; x < maxTerms; x++){
552         if(f.coeff[x]!=zero) f.coeff[x]=f.coeff[x]/inv;
553     }
554     return f;
555 }
```

Κε) " Polynomial gcd (const Polynomial *, const Polynomial *)" : *[συνάρτηση]*

Εύρεση και επιστροφής του μέγιστου κοινού διαιρέτη δύο πολυωνύμων. Χρησιμοποιούμε το "inv" και "polyCombine2" για

```
557 Polynomial Polynomial::gcd(const Polynomial &a, const Polynomial &b){
558
559     Polynomial zero;
560     zero.zeroCoef();
561     Polynomial a1=a.monic();
562     Polynomial b1=b.monic();
563     Polynomial q=a1/b1;
564     Polynomial r=a1%b1;
565     polyCombine2(b1);
566     polyCombine2(r);
567
568     if (r==zero) return b1;
569     else return gcd(b1,r);
570 }
```

Κστ) " Polynomial xplusA (Rational)" : *[συνάρτηση]*

Εύρεση του πολυωνύμου p(x+a).Η πράξη αυτή είναι απαραίτητη για κάποια βήματα του αλγορίθμου VAS που καλούμε στην συνέχεια στο αρχείο "VAS.cpp". Χρησιμοποιούμε τις μεταβλητές "oldPascal" και "thisPascal" για

```
571 Polynomial Polynomial::xplusA(Rational a){
572
573     Polynomial ret;
574     ret.zeroCoef();
575     int k = getDegree();
576     int *thisPascal, *oldPascal;
577
578     for(int i = 0; i <=k ;i++)
579     {
580         if(i >0) oldPascal = thisPascal;
581         thisPascal = new int[i+1];
582         thisPascal[0] = 1;
583         if(i > 0)
584         {
585             thisPascal[i] = 1;
586             for(int j = 1; j <i; j++)
587                 thisPascal[j] = oldPascal[j]+oldPascal[j-1];
588             delete [] oldPascal;
589         }
590         Rational rr(one);
591         for(int j = i; j >= 0; j--)
592         {
593             ret.exp[j] = j;
594             ret.coeff[j] += getTermCoeff(i)*rr*Rational(thisPascal[j],1);
595             rr *=a;
596         }
597
598     }
599     return ret;
600 }
```

Κζ) " Polynomial xplusA2 (Rational)" : *[συνάρτηση]*

Εναλλακτική εύρεση του πολυωνύμου p(x+a) με βάση την μέθοδο "Taylorshift".Η πράξη αυτή γίνεται για να κερδίσουμε ταχύτητα σε σύγκριση με την προηγούμενη. Επειδή όμως είναι απαραίτητο να φτιάξουμε ένα πολυώνυμο σε τόσους όρους, όσο είναι και ο βαθμός του(ενώ μέχρι τώρα τους μηδενικούς τους παραλείπαμε), περνούμε αυτήν την φορά από αυτήν την διαδικασία.

```
602 Polynomial Polynomial::xplusA2(Rational a) {
603
604     int k;
605     k = getTermExp(maxDegIndex());
606
607     IndexInTable myIndexInTable;
608
609     Rational *coeff2=new Rational[k+1];
610     Rational **table = new Rational*[k+1];
611
612     for(int rr = 0; rr <=k; rr++) table[rr] = new Rational[k+1];
613
614         Rational *coeff3=new Rational[k+1];
615         Rational *coeff4=new Rational[k+1];
616         int *exp4=new int[k+1];
617         Polynomial myPolynomial;
618
619     for (int i=0;i<=k-1;i++) {
620
621                 myIndexInTable = searchForTermInExp(k-i,exp,k);
622
623                 if (myIndexInTable.found) {
624                    coeff2[i] = coeff[myIndexInTable.index];
625                 }
626                 else if (!myIndexInTable.found) {
627                    coeff2[i] = zero;
628                 }
629     }
630
631     coeff2[k] = coeff[0];
632
633     for (int i=0;i<=0;i++) {
634         table[i][0] = one;
635
636         for (int j=1;j<=k;j++) {
637             table[i][j] = (a * table[i][j-1]) + coeff2[j];
638         }
639     }
```

```
641     for (int i=1;i<=k;i++) {
642         table[i][0] = one;
643
644         for (int j=1;j<=k;j++) {
645             table[i][j] = (a * table[i][j-1]) + table[i-1][j];
646         }
647     }
648
649     for (int i=0;i<=k;i++){
650
651         coeff3[i] = table[k-i][i];
652     }
653
654         int j = 0;
655
656         coeff4[0] = coeff3[k];
657         exp4[0] = 0;
```

```
659         for (int i=1;i<=k;i++) {
660            if (j <= k-1) {
661               while ((coeff3[j] == zero) && (j <= k-1)) {
662                  j++;
663               }
664
665               if (j <= k-1) {
666                  coeff4[i] = coeff3[j];
667                  exp4[i] = k-j;
668                  j++;
669               }
670               else {
671                  coeff4[i] = zero;
672                  exp4[i] = 0;
673               }
674            }
675            else {
676               coeff4[i] = zero;
677               exp4[i] = 0;
678            }
679         }
680
681         myPolynomial.numTerms = k;
682
683         for (int i=0;i<=k;i++) {
684            myPolynomial.coeff[i] = coeff4[i];
685         }
686
687         for (int i=0;i<=k;i++) {
688            myPolynomial.exp[i] = exp4[i];
689         }
690
691         delete [] coeff2;
692         delete [] coeff3;
693         delete [] coeff4;
694         delete [] exp4;
695         for(int rr = 0; rr <=k; rr++) delete [] table[rr];
696         delete [] table;
697
698
699         return myPolynomial;
700   }
```

Κη) " Polynomial xtimesA(Rational)" : *[συνάρτηση]*

Εύρεση του πολυωνύμου p(x*a).Η πράξη αυτή είναι απαραίτητη για κάποια βήματα του αλγορίθμου VAS που καλούμε στην συνέχεια στο αρχείο "VAS.cpp". Χρησιμοποιούμε τις συναρτήσεις "mpz_init","mpq_canonicalize", "mpq_get_num","mpq_get_den","mpz_pow_ui","mpq_set_num","mpq_set_den", "mpz_clear" από το πακέτο "gmp", όπως έχουν εξηγηθεί στο αρχείο "Rational.h".

```
Polynomial Polynomial::xtimesA(Rational a){

    Polynomial ret(*this);
    mpz_t num,den;
    mpz_init(num);
    mpz_init(den);

    for(int i = 0; i < maxTerms; i++)
    {
        if(ret.exp[i]>0 && ret.coeff[i] != zero)
        {
            Rational a1=a;

            mpq_canonicalize(a1.rational);
            mpq_get_num(num,a1.rational);
            mpq_get_den(den,a1.rational);
            mpz_pow_ui(num,num,exp[i]);
            mpz_pow_ui(den,den,exp[i]);
            mpq_set_num(a1.rational,num);
            mpq_set_den(a1.rational,den);
            ret.coeff[i]*=a1;
        }
    }
    mpz_clear(num);
    mpz_clear(den);
    return ret;
}
```

Κθ) " Polynomial xtoMdivx()" : *[συνάρτηση]*

```
Polynomial Polynomial::xtoMdivx(){

    Polynomial temp(*this);
    int deg = temp.getDegree();

    for(int i =0; i < maxTerms; i++)
    {
        if(temp.coeff[i] != zero)
            temp.exp[i] = deg - temp.exp[i];
    }
    return temp.xplusA(one);
}
```

Λ) " Polynomial sqf() const" : *[συνάρτηση]*

Υλοποίηση της παραδοσιακής εκδοχής της μεθόδου έτσι όπως περιγράφεται στην διεθνή βιβλιογραφία και αναλύθηκε νωρίτερα στις «*Γενικές σημειώσεις».*

Λα) " void enterTerms()" : *[συνάρτηση]*

Εισαγωγή πολυωνύμου όρο με όρο. Πρώτα βάζουμε το πλήθος των όρων, μετά τον συντελεστή και τέλος τον αντίστοιχο εκθέτη(με την σειρά της προτίμησης μας).

Λβ) " void enterTermsDouble()" : *[συνάρτηση]*

Εισαγωγή πολυωνύμου όρο με όρο. Η διαφορά με την προηγούμενη είναι ότι οι αριθμοί είναι "double".

Λγ) " void enterTermsRational()" : *[συνάρτηση]*

Εισαγωγή πολυωνύμου όρο με όρο. Η διαφορά με την προηγούμενη είναι ότι οι αριθμοί είναι "Rational".

Λδ) " int load()" : *[συνάρτηση]*

Εισαγωγή πολυωνύμου από αρχείο με όνομα"poly.txt". Η μορφή του πολυωνύμου ακολουθεί μία αυστηρή μορφή όπως αναλύθηκε νωρίτερα στις «*Γενικές σημειώσεις».*

" int loadDouble()" : *[συνάρτηση]*

Εισαγωγή πολυωνύμου από αρχείο με όνομα"poly1.txt". Η διαφορά με την προηγούμενη είναι ότι οι αριθμοί είναι "double".

" int loadRational()" : *[συνάρτηση]*

Εισαγωγή πολυωνύμου από αρχείο με όνομα"poly1.txt".Η διαφορά με την προηγούμενη είναι ότι οι αριθμοί είναι "Rational".

" void refress()" : *[συνάρτηση]*

Απλοποίηση των συντελεστών ενός πολυωνύμου(εάν είναι δυνατόν) μέσω "gcd".

" void zeroCoef()" : *[συνάρτηση]*

Αρχικοποίηση των συντελεστών ενός πολυωνύμου ως κενοί.

" void mononym(int , const Rational *)" : *[συνάρτηση]*

Μετατροπή πολυωνύμου σε μονώνυμο.

"int sgc()" : *[συνάρτηση]*

Υπολογισμός μεταβολών πρόσημου.

"void printPolynomial() const" : *[συνάρτηση]*

Εκτύπωση πολυωνύμου από τον πρώτο μέχρι τον τελευταίο όρο.

“void printPolynomialDouble() const” : *[συνάρτηση]*

Εκτύπωση πολυωνύμου από τον πρώτο μέχρι τον τελευταίο όρο. Η διαφορά με την προηγούμενη είναι ότι οι αριθμοί είναι “double”.

“void setCoeff(int, Rational)” : *[συνάρτηση]*

Τοποθέτηση του συντελεστή που δίνεται σαν δεύτερη παράμετρος στον όρο που δίνεται σαν πρώτη παράμετρος, εάν υπάρχει.

“int maxDegIndex() const” : *[συνάρτηση]*

Εύρεση και επιστροφή του μέγιστου βαθμού του πολυωνύμου, εφόσον έχει μπει με τυχαία σειρά όρων.

“int getDegree()” : *[συνάρτηση]*

Εύρεση μέγιστου εκθέτη(βαθμού) πολυωνύμου, .

“int getNumTerms() const” : *[συνάρτηση]*

Εύρεση και επιστροφή του πλήθους των όρων του πολυωνύμου .

“int getTermExp(int) const” : *[συνάρτηση]*

Εύρεση και επιστροφή του εκθέτη του όρου που δίνεται σαν παράμετρος, εάν υπάρχει.

“Rational getTermCoeff(int) const” : *[συνάρτηση]*

Εύρεση και επιστροφή του συντελεστή του όρου που δίνεται σαν παράμετρος, εάν υπάρχει. Θεωρούμε ότι είναι ρητός αριθμός, καθώς μπορεί να έχουμε και κλασματικούς συντελεστές.

“ void polyCombine(Polynomial *)” : *[συνάρτηση]*

“ void polyCombine2(Polynomial *)” : *[συνάρτηση]*

“ void polyCombine3(Polynomial *)” : *[συνάρτηση]*

“ bool constantPoly(Polynomial *)” : *[συνάρτηση]*

“ IndexInTable searchForTermInExp(int , int [], int)” : *[συνάρτηση]*

Εύρεση και επιστροφή ενός συγκεκριμένου συντελεστή, εάν υπάρχει. Ως παράμετροι εισάγονται ο συντελεστής, ο αντίστοιχος εκθέτης και το πλήθος των όρων του πολυωνύμου.

“~Polynomial()” : *[συνάρτηση]*

Καταστροφή βασικού constractor της κλάσης όταν αυτός δεν θα είναι πλέον χρήσιμος, στα πρότυπα του garbage collector.

Υποπρόγραμμα 5^{ο} : "bound.h"

Σε αυτό το αρχείο, υλοποιούμε ένα από τα βασικότερα σημεία του αλγορίθμου VAS. Η εύρεση άνω και κάτω φράγματος είναι απαραίτητη καθώς κάθε ρίζα ενός πολυωνύμου υπολογίζεται προσεγγιστικά, μέσα σε ένα διάστημα και όχι με την ακριβή της τιμή. Οι βοηθητικοί αλγόριθμοι FLQ, LMQ που παρουσιάζονται στις δημοσιεύσεις που προηγήθηκαν, ουσιαστικά βρίσκουν τα κάτω φράγματα για θετικές ρίζες. Για να βρούμε τα άνω, αρκεί να βασιστούμε στην απλή παραδοχή ότι το πολυώνυμο που παίρνουμε ως είσοδο είναι το

$$g(x) = x^n \times f(\frac{1}{x})$$

Όπου βέβαια $f(x)$ είναι το πολυώνυμο που παίρνει ως είσοδο ο αλγόριθμος κάτω φράγματος.

Τελικά τα άνω και κάτω φράγματα συνδέονται με την σχέση $lb = \frac{1}{ub}$.

Όταν θέλουμε να βρούμε τα αντίστοιχα φράγματα για αρνητικές ρίζες, ουσιαστικά τρέχουμε τον VAS βάζοντας όπου $x -> (-x)$ και στην συνέχεια ο ίδιος αλγόριθμος θα καλέσει τους FLQ, LMQ.

Όπως αναλύεται στις δημοσιεύσεις, ενώ ο αλγόριθμος FLQ(THE FASTEST QUADRATIC COMPLEXITY BOUND ON THE VALUES OF POSITIVE ROOTS OF POLYNOMIALS) είναι ο γρηγορότερος από όλους τους αντίστοιχους τετραγωνικής πολυπλοκότητας, τελικά χρησιμοποιείται ο αλγόριθμος LMQ (THE LOCAL MAX QUADRATIC COMPLEXITY BOUND) για εύρεση φραγμάτων από τον VAS καθώς οι **Akritas**, **Strzebonski**, **Vigklas** απέδειξαν με την έρευνά τους ότι τρέχει βέλτιστα για την αντίστοιχη υλοποίησή τους στο MATHEMATICA.

Όπως έχουμε αναφέρει στα δύο προηγούμενα αρχεία λειτουργούμε με αριθμούς τύπου "Rational". Όμως για να πετύχουμε μέγιστη δυνατή εμβέλεια λειτουργούμε με "double"(20 ψηφία δεκαδικού μέρους). Αυτοί οι αριθμοί, λειτουργούν για Windows 64 bits ενώ τα Windows 32 bits φτάνουν μέχρι 10 ψηφία("long").Αυτό, μπορεί να δημιουργήσει προβλήματα καθώς εάν τρέχει αυτός ο κώδικας σε Windows 32 bits, τότε η C++ θα κάνει απότομη στρογγυλοποίηση, κόβοντας τα 4 τελευταία bytes και κρατώντας τα 4 πρώτα πιο σημαντικά για να μετατρέψει έναν αριθμό "double(long long)" σε "long".

Για να εξηγήσουμε βήμα-βήμα το πως υλοποιούνται οι αλγόριθμοι FLQ, LMQ παραθέτουμε τους ψευδοκώδικές τους για να αναφερθούμε αργότερα στα δομικά στοιχεία αυτού του αρχείου, σε κάθε γραμμή ξεχωριστά.

Input : A univariate polynomial
$p(x) = a_k x^k + a_{k-1}x^{k-1} + \ldots + a_0, (a_k > 0)$
Output: An upper bound $tempmax$, on the values of the positive roots of the polynomial

```
1  initializations;
2  cl ⟵ {a_0, a_1, a_2, ..., a_{k-1}, a_k};
3  λ ⟵ number of negative elements of cl;
4  usedVector ⟵ {0, 0, 0, ..., 0};
5  for i ⟵ 1 to k + 1 do
6  |  if cl(i) > 0 then usedVector(i) = 1;
7  end
8  if k + 1 ≤ 1 or λ = 0 then return tempmax = 0;
9  i = k + 1;
10 templamda = 0;
11 flag = 0;
12 while templamda < λ do // make sure t(q_{2i-1}) ≥ t(q_{2i}) holds for
   all i
13 |  if cl(i) > 0 then
14 |  |  if flag = 0 then posCounter + +;
15 |  |  else if flag = 1 then
16 |  |  |  if negCounter > posCounter then
17 |  |  |  |  usedVector(positionLastPositiveCoef) =
   |  |  |  |  negCounter − posCounter + 1;
18 |  |  |  end
19 |  |  |  negCounter = 0;
20 |  |  |  posCounter = 1;
21 |  |  |  flag = 0;
22 |  |  end
23 |  |  positionLastPositiveCoef = i;
24 |  else if cl(i) < 0 then
25 |  |  flag = 1;
26 |  |  negCounter + +;
27 |  |  templamda + +;
28 |  end
29 |  i − −;
30 end
31 if negCounter > posCounter then
32 |  usedVector(positionLastPositiveCoef) =
   |  negCounter − posCounter + 1;
33 end
```

Algorithm 3.2. FLQ implementation part 1

```
34 sumPosCoff = 0;
35 i = k + 1;
   // Last of the first-λ coefficients
36 while sumPosCoff < λ do
37 |  if usedVector(i) ≠ 0 then
38 |  |  sumPosCoff + = usedVector(i);
39 |  |  flPos = i;
40 |  end
41 |  i − −;
42 end
   /* If the last of the first-λ coefficients is a broken one
      (usedVector(flPos) > 1), there might be a chance that the
      sum of the positive coefficients (including broken ones) is
      more than λ.For Example:
      Let the signs of p be + + + − + + + − − − + + + −
      the 5^th positive coefficient will be broken into 2 pieces
      (usedVector(8) = 2). However, the sum of the first-λ (5 non
      broken)positive coefficients is 6(incl. broken). As a
      result we are going to use the last of the positive first-λ
      coefficients timesToUse(8) − (sum − λ) = 1 time only.      */
43 timesToUse(flpos)− = (sumPosCoff − λ);
44 denomVector ⟵ usedVector;
45 m = k;
46 tempmax = 0;
47 while λ > 0 do
48 |  if cl(m) < 0 then
49 |  |  tempmin = ∞;
50 |  |  for n = k + 1 to max(m + 1, flPos) do
51 |  |  |  if usedVector(u) > 0 then
   |  |  |  |  tempB = (−cl(m) / cl(n) / denomVector(n))^{1/(n−m)};
52 |  |  |  |  if tempmin > tempB then
53 |  |  |  |  |  tempmin = tempB;
54 |  |  |  |  |  tempN = n;
55 |  |  |  |  end
56 |  |  |  end
57 |  |  end
58 |  |  usedVector(tempN) − −;
59 |  |  λ − −;
60 |  |  if tempmax < tempmin then tempmax = tempmin;
61 |  end
62 |  m − −;
63 end
64 return tempmax;
```

Algorithm 3.3. FLQ implementation part 2

Input : A univariate polynomial $p(x) = a_k x^k + a_{k-1}x^{k-1} + \ldots + a_0, (a_k > 0)$
Output: An upper bound $tempmax$, on the values of the positive roots of the polynomial

```
1  initializations;
2  cl ⟵ {a_0, a_1, a_2, ..., a_{k-1}, a_k};
3  timesused ⟵ {1, 1, 1, ..., 1};
4  tempmax = 0;
5  if k + 1 ≤ 1 then return tempmax = 0;
6  for m ⟵ k to 1 do
7      if cl(m) < 0 then
8          tempmin = ∞;
9          for n ⟵ k + 1 to m + 1 do
10             temp = (-cl(m) / cl(n))^(1/(n-m));
11             timesused(n) + +;
12             if tempmin > temp then tempmin = temp;
13         end
14         if tempmax < tempmin then tempmax = tempmin;
15     end
16 end
17 return tempmax;
```

Algorithm 3.1. LMQ implementation

Αρχικά καλούμε τα πακέτα :

1. “math.h”
2. “Rational.h” :από όπου καλούμε τις βασικές πράξεις μεταξύ ρητών αριθμών
3. “Polynomial” : από όπου καλούμε τις βασικές πράξεις μεταξύ ρητών πολυωνύμων

```
1  #ifndef BOUND_H_
2  #define BOUND_H_
3
4  #include <math.h>
5  #include "Rational.h"
6  #include "Polynomial.h"
```

Και αρχικοποιούμε μία σταθερή μεταβλητή “negone” που θα χρησιμοποιήσουμε ως αναφορά στον αρνητικό αριθμό «-1».

```
8  const Rational negone(-1,1);
```

Τα δομικά στοιχεία της "Polynomial.h"είναι :

A) "double boundFLQ(Polynomial)" : *[συνάρτηση]*

Υλοποίηση του κάτω φράγματος για τον αλγόριθμο FLQ. Ως είσοδο παίρνει ένα πολυώνυμο και επιστρέφει το κάτω όριο ως "double". Παρακάτω παρουσιάζουμε την αντιστοιχία των γραμμών μεταξύ ψευδοκώδικα και C++.

Ψευδοκώδικας Γραμμή(ές)	**C++ Γραμμή(ές)**
1	12-18
	20-29
2	32
3	39
4	30-31
5-7	32-35
8	
9	
10-11	45-46
12-33	47-69
34-35	70-71
36-42	72-78
43	
44	79-81
45-46	84-103

B) "double boundLMQ(Polynomial *)" : *[συνάρτηση]*

Υλοποίηση του κάτω φράγματος για τον αλγόριθμο LMQ. Ως είσοδο παίρνει έναν δείκτη σε πολυώνυμο και επιστρέφει το κάτω όριο ως "double". Παρακάτω παρουσιάζουμε την αντιστοιχία των γραμμών μεταξύ ψευδοκώδικα και C++.

Ψευδοκώδικας Γραμμή(ές)	**C++ Γραμμή(ές)**
1	119-122
	123-132
2	136
3	133-135
5	
6-16	138-154

Γ) "double upboundLMQ(Polynomial)" : *[συνάρτηση]*

Υλοποίηση του άνω φράγματος για τον αλγόριθμο LMQ. Ως είσοδο παίρνει έναν δείκτη σε πολυώνυμο και επιστρέφει το άνω όριο ως "double".

Υποπρόγραμμα 6^{ο} : "VAS.cpp"

Σε αυτό το αρχείο, υλοποιούμε τον βασικό αλγόριθμο **VAS** μεταφέροντας τα βήματα του ψευδοκώδικα. Όλα τα αρχεία που έχουν προηγηθεί έχουν σαν σκοπό να χρησιμοποιηθούν εδώ απλοποιώντας την δομή του τρέχοντος κώδικα.

Σε αυτό το πρόγραμμα καλούμε τέλος την "main" η οποία καλεί την βασική συνάρτηση "VAS" και εκτυπώνουμε τα αποτελέσματα, πρώτα για τις θετικές ρίζες και στην συνέχεια για τις αρνητικές.

Όταν θέλουμε να βρούμε τις αρνητικές ρίζες, ουσιαστικά τρέχουμε τον VAS βάζοντας όπου $x->(-x)$.

Για να εξηγήσουμε βήμα-βήμα το πως υλοποιείται ο αλγόριθμος VAS, παραθέτουμε τον ψευδοκώδικα για να αναφερθούμε αργότερα στα δομικά στοιχεία αυτού του αρχείου, σε κάθε γραμμή ξεχωριστά.

Algorithm Continued Fractions (VAS-CF)

Input: A squarefree polynomial $f \in Z[x] \setminus \{0\}$
Output: The list $rootlist$ of the isolation intervals of the positive roots of f

1. Set $rootlist$ to an empty list. Compute $s \leftarrow sgc(f)$. If $s = 0$ return an empty list. If $s = 1$ return $\{(0, \infty)\}$. Put interval data $\{1, 0, 0, 1, f, s\}$ on $intervalstack$.
2. If $intervalstack$ is empty, return $rootlist$, else take interval data $\{a, b, c, d, p, s\}$ off $intervalstack$.
3. Compute a lower bound $\alpha \in \mathbb{Z}$ on the positive roots of p.
4. If $\alpha > \alpha_0$ set $p(x) \leftarrow p(\alpha x)$, $a \leftarrow \alpha a$, $c \leftarrow \alpha c$, and $\alpha \leftarrow 1$.
5. If $\alpha \geq 1$, set $p(x) \leftarrow p(x + \alpha)$, $b \leftarrow \alpha a + b$, and $d \leftarrow \alpha c + d$. If $p(0) = 0$, add $[b/d, b/d]$ to $rootlist$, and set $p(x) \leftarrow p(x)/x$. Compute $s \leftarrow sgc(p)$. If $s = 0$ go to Step 2. If $s = 1$ add $intrv(a, b, c, d)$ to $rootlist$ and go to Step 2.
6. Compute $p_1(x) \leftarrow p(x + 1)$, and set $a_1 \leftarrow a$, $b_1 \leftarrow a + b$, $c_1 \leftarrow c$, $d_1 \leftarrow c + d$, and $r \leftarrow 0$. If $p_1(0) = 0$, add $[b_1/d_1, b_1/d_1]$ to $rootlist$, and set $p_1(x) \leftarrow p_1(x)/x$, and $r \leftarrow 1$. Compute $s_1 \leftarrow sgc(p_1)$, and set $s_2 \leftarrow s - s_1 - r$, $a_2 \leftarrow b$, $b_2 \leftarrow a + b$, $c_2 \leftarrow d$, and $d_2 \leftarrow c + d$.
7. If $s_2 > 1$, compute $p_2(x) \leftarrow (x + 1)^m p(\frac{1}{x+1})$, where m is the degree of p. If $p_2(0) = 0$, set $p_2(x) \leftarrow p_2(x)/x$. Compute $s_2 \leftarrow sgc(p_2)$.
8. If $s_1 < s_2$, swap $\{a_1, b_1, c_1, d_1, p_1, s_1\}$ with $\{a_2, b_2, c_2, d_2, p_2, s_2\}$.
9. If $s_1 = 0$ goto Step 2. If $s_1 = 1$ add $intrv(a_1, b_1, c_1, d_1)$ to $rootlist$, else put interval data $\{a_1, b_1, c_1, d_1, p_1, s_1\}$ on $intervalstack$.
10. If $s_2 = 0$ goto Step 2. If $s_2 = 1$ add $intrv(a_2, b_2, c_2, d_2)$ to $rootlist$, else put interval data $\{a_2, b_2, c_2, d_2, p_2, s_2\}$on $intervalstack$. Go to Step 2.

Αρχικά καλούμε τα πακέτα :

1. "bound.h" , από όπου καλούμε τις "boundLMQ" και "upboundLMQ".
2. "rootslist.h", από όπου καλούμε την λίστα που συνδέει τις ρίζες αλλά και την ρουτίνα εκτύπωσης.
3. "list.h", από όπου καλούμε τον αφηρημένο τύπο για να δημιουργήσουμε τον τύπο δεδομένων της στοίβας.
4. "Rational.h" :από όπου καλούμε τις βασικές πράξεις μεταξύ ρητών αριθμών.
5. "Polynomial" : από όπου καλούμε τις βασικές πράξεις μεταξύ ρητών πολυωνύμων.

```
1  #include "bound.h"
2  #include "rootslist.h"
3  #include "list.h"
4  #include "Rational.h"
5  #include "Polynomial.h"
```

Τα δομικά στοιχεία της "Polynomial.h"είναι :

A) Interval *: [κλάση]*

Υλοποίηση του τύπου δεδομένων ενός κόμβου της INTERVALSTACK, έτσι όπως περιγράφεται από τις δημοσιεύσεις. Ουσιαστικά αποτελεί την υλοποίηση του αφηρημένου τύπου δεδομένων της στοίβας που ορίζεται από το αρχείο "list.h" και καλούμε αργότερα στην συνάρτηση "VAS".

B) void put(Rational , Rational , Rational , Rational , Polynomial , int) : *[συνάρτηση]*

Ορισμός του αντικειμένου της κλάσης "Interval που περιέχει 4 ρητούς, ένα πολυώνυμο και ένα ακέραιο, όπως περιγράφεται στις δημοσιεύσεις.

```
class Interval {

public:

    Rational a,b,c,d;
    Polynomial p;
    int s;

    void put(Rational x, Rational y, Rational w, Rational z, Polynomial p1, int s1){
        a=x;
        b=y;
        c=w;
        d=z;
        p=p1;
        s=s1;
    }
};
```

Γ) Root *: [κλάση]*

Υλοποίηση του τύπου δεδομένων ενός κόμβου της λίστας που συνδέει τα διαστήματα ριζών, έτσι όπως δημιουργούνται από την συνάρτηση "VAS". Ουσιαστικά αποτελεί την υλοποίηση του αφηρημένου τύπου δεδομένων της διασυνδεδεμένης λίστας που ορίζεται από το αρχείο "rootslist.h" και καλούμε αργότερα στην συνάρτηση "VAS".

Δ) "void setlimits(Rational , Rational) " : *[συνάρτηση]*

Ορισμός των ορίων-φραγμάτων κάθε ρίζας όπου βέβαια και οι δύο αριθμοί είναι ρητοί.

Ε) "void printinterval()" : *[συνάρτηση]*

Εκτύπωση των φραγμάτων που περιέχουν την ρίζα.

```
25  class Root {
26
27  public:
28
29      Rational a,b;
30
31      void setlimits(Rational x, Rational y){
32          a=x;
33          b=y;
34      }
35
36      void printinterval(){
37          cout<<"[";
38          a.disp();
39          cout<<" , ";
40          b.disp();
41          if (b==inf) cout<<")";
42          else cout<<"]";
43      }
44  };
```

Στ) " void printroot(LinkedList<Root>)" : *[συνάρτηση]*

Μέθοδος εκτύπωσης των διαστημάτων κάθε ρίζας, έτσι όπως ορίζεται στο αρχείο "rootslist.h".

```
47  void printroot(LinkedList<Root> lista){
48      Root a;
49      if (lista.isEmpty()) cout<<"\nNo Positive Roots:\n";
50      else{
51          while (!lista.isEmpty()){
52              lista.removeHead(a);
53              a.printinterval();
54          }
55      }
56  }
```

Z) void newVAS(Polynomial *,int , bool , rootslist *) : *[συνάρτηση]*

Υλοποίηση των βημάτων του αλγορίθμου VAS, έτσι όπως περιγράφεται από τις δημοσιεύσεις. Ως παραμέτρους παίρνει ένα δείκτη σε πολυώνυμο, ένα ακέραιο , μία λογική μεταβλητή και ένα δείκτη στην λίστα εκτύπωσης για τα διαστήματα. Παρακάτω παρουσιάζουμε την αντιστοιχία των γραμμών από τον ψευδοκώδικα στην C++.

H) void myswap(T *, T *) : *[συνάρτηση]*

Υλοποίηση της "swap" , της ανταλλαγής των τιμών δύο μεταβλητών που δίνονται ως παράμετροι με την μορφή δεικτών. Αυτή η συνάρτηση χρησιμοποιείται στην

Θ) int main()[συνάρτηση] : *[συνάρτηση]*

Απεικόνιση των αποτελεσμάτων που έχουμε υπολογίσει με τον αλγόριθμο "newVAS". Χρησιμοποιούμε σταδιακά τις συναρτήσεις

i. "mononym" (δύο φορές για το πολυώνυμο πριν και μετά την sqf),
ii. "loadRational"(φόρτωση πολυώνυμου από αρχείο "poly.txt"),
iii. "sqf"(square free decomposition για απλοποίηση του πολυωνύμου), "printPolynomial"(εκτύπωση πολυωνύμου)
iv. "newVAS" (αλγόριθμος VAS για θετικές ρίζες)
v. "fneg"(δημιουργία του πολυώνυμου "p(x)")
vi. "newVAS""(αλγόριθμος VAS για αρνητικές ρίζες).

Επαναλαμβάνουμε την διαδικασία όσο υποδηλώνει η μεταβλητή "resp".