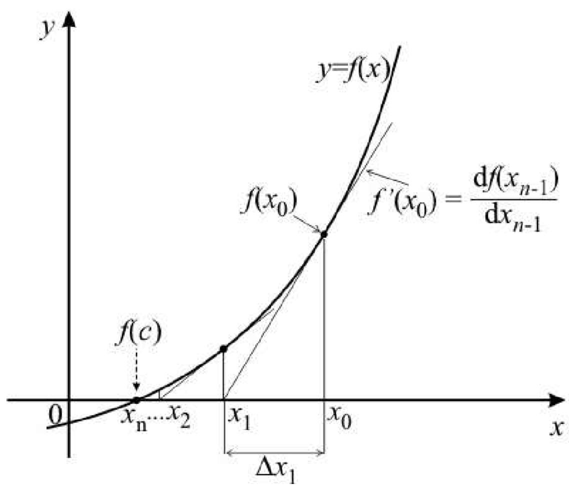
# 1η Εργασία Στο Μάθημα Της Αριθμητικής Ανάλυσης

Ονοματεπώνυμο : Παπαδόπουλος Αναστάσιος

ΑΕΜ : 3654

Διδάσκων : Τέφας Αναστάσιος

13 Δεκεμβρίου 2020




ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

# Περιεχόμενα


**1 Εισαγωγή** **3**

**2 Άσκηση 1** **4**

2.1 Μέθοδος της διχοτόμησης . . . . . 5

2.2 Μέθοδος Newton-Raphson . . . . . 9

2.3 Μέθοδος της τέμνουσας . . . . . 14

2.4 Υπολογισμός σύγκλισης της μεθόδου **Newton-Raphson** για κάθε ρίζα . . . . . 18

2.5 Σύγκριση επαναλήψεων για κάθε ρίζα . . . . . 21

2.5.1 **Μέθοδος της διχοτόμησης** . . . . . 21

2.5.2 **Μέθοδος Newton-Raphson** . . . . . 22

2.5.3 **Μέθοδος της τέμνουσας** . . . . . 23

**3 Άσκηση 2** **24**

3.1 Τροποποιημένη μέθοδος Newton-Raphson . . . . . 25

3.2 Τροποποιημένη μέθοδος της διχοτόμησης . . . . . 27

3.3 Τροποποιημένη μέθοδος της τέμνουσας . . . . . 29

3.4 Σύγκλιση τροποποιήμενης μεθόδου της διχοτομήσης . . . . . 31

3.5 Πειραματική σύγκριση της ταχύτητας σύγκλισης των τροποιήμενων με τις κλασσικές μεθόδους . . . . . 33

**4 Άσκηση 3** **36**

4.1 **PA = LU** αποσύνθεση . . . . . 36

4.2 Αποσύνθεση **Cholesky** . . . . . 39

4.3 Μέθοδος **Gauss-Seidel** . . . . . 40

# 1 Εισαγωγή

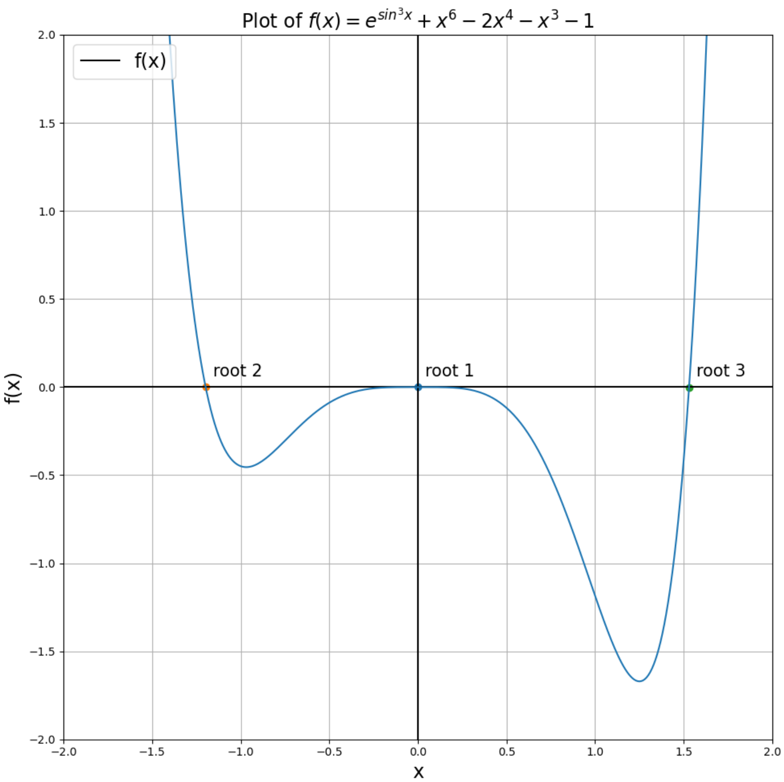
Η εργασία αυτή αναπτύχθηκε κατά την παρακολούθηση του μάθηματος της Αριθμητικής Ανάλυσης του 3ου εξαμήνου του τμήματος Πληροφορικής ΑΠΘ. Περιλαμβάνει υλοποιήσεις και πειραματισμούς με αλγορίθμους Αριθμητικής Ανάλυσης που αναφέρονται σε θεματικές ενότητες όπως της επίλυσης μη γραμμικών εξισώσεων ( **Άσκηση 1 και 2** ) για παράδειγμα η μέθοδος της διχοτόμησης, της επίλυσης γραμμικών συστημάτων ( **Άσκηση 3** ) για παράδειγμα η μέθοδος της $\mathbf{PA} = \mathbf{LU}$ αποσύνθεσης και τέλος της ιδιοανάλυσης ( **Άσκηση 4** ). Οι αλγόριθμοι αυτοί καθώς και οι γραφικές παραστάσεις που περιέχονται στο παρόν έγγραφο έχουν υλοποιηθεί με την γλώσσα προγραμματισμού **Python** με χρήση των βιβλιοθηκών **NumPy** και **Matplotlib**. Στα παραδοτέα περιλαμβάνεται το παρόν **pdf** αρχείο καθώς και το αντίστοιχο **.tex** και για κάθε μία άσκηση υπάρχει ένας υποφάκελος όπου περίεχει τα αντίστοιχα **.py** αρχεία με τον πηγαίο κώδικα.

## 2 Άσκηση 1

Στην **1η άσκηση** μας δίνεται αρχικά η συνάρτηση

$$f(x) = e^{sin^3 x} + x^6 - 2x^4 - x^3 - 1$$

και ζητείται η γραφική παράσταση της στο διάστημα [-2,2].




Σχήμα 1: Γραφική παράσταση της συνάρτησης $\mathbf{f(x)}$

Από την οποία παρατηρούμε στο *Σχήμα 1* ότι η συνάρτηση έχει 3 ρίζες.Στην συνέχεια ζητείται να υπολογισθούν αυτές οι ρίζες με ακρίβεια 5ου δεκαδικού ψηφίου χρησιμοποιώντας την μέθοδο της διχοτόμησης, την μέθοδο Newton-Raphson και τέλος την μέθοδος της τέμνουσας.

## 2.1 Μέθοδος της διχοτόμησης

Η μέθοδος της διχοτόμησης γενικά προϋποθέτει την ύπαρξη ενός διαστήματος **[a,b]** στο οποίο η συνάρτηση έχει τουλάχιστον μία ρίζα. Για να βρούμε ένα τέτοιο διάστημα χρησιμοποιούμε το θεώρημα **Bolzano** σε συνδυασμό με την γραφική παράσταση της **f(x)**. Όπως παρατηρούμε στο *Σχήμα 1* η **f(x)** έχει μία ρίζα στο διάστημα **[-1.5,-1.0]**. Εξάλλου, στο διάστημα **[-1.5,-1.0]** η **f(x)** είναι συνεχής με

$$\mathbf{f(\text{-}1.5)} * \mathbf{f(\text{-}1.0)} < 0$$

συνεπώς σύμφωνα με το θέωρημα **Bolzano** έχει μία τουλάχιστον ρίζα στο διάστημα αυτό. Για να βρούμε την ρίζα χρησιμοποιούμε την συνάρτηση ***bisection*** από το αρχείο ***bisection.py*** με ορίσματα την συνάρτηση **f(x), a = -1.5, b = -1.0** , ενώ το **default** όρισμα **eps** έχει την τιμή που χρειάζεται για ακρίβεια **5** δεκαδικών ψηφίων. Η συνάρτηση επιστρέφει την ρίζα της **f(x)** στο διάστημα **(a,b)** και τον αριθμό των επαναλήψεων που χρειάστηκαν για να βρέθει η ρίζα με σφάλμα λιγότερο από **eps**.

```
root, iterations_num = bisection(g, -1.5, -1.0)
```

Σχήμα 2: Παράδειγμα κλήσης της συνάρτησης ***bisection***

Μετά την κλήση της συνάρτησης ***bisection*** τα αποτελέσματα είναι τα εξής:

```
Root = -1.1976
Iterations = 17
```

Σχήμα 3: Αποτελέσματα κλήσης της συνάρτησης ***bisection*** στο διάστημα **[-1.5,-1.0]**.

Όπου παρατηρούμε ότι η ρίζα της **f(x)** στο διάστημα **[-1.5,-1.0]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **-1.1976** καθώς και ότι η μέθοδος της διχοτόμησης χρειάστηκε **17** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Στην συνέχεια παρατηρούμε στο *Σχήμα 1* ότι η **f(x)** έχει μία ακόμα ρίζα στο διάστημα **[1.25,1.75]**. Πέρα από την γραφική παράσταση, με παρόμοιο τρόπο χρησιμοποιώντας το θεώρημα **Bolzano** μπορούμε να αποδείξουμε ότι η **f(x)** όντως έχει μία τουλάχιστον ρίζα στο διάστημα **[1.25,1.75]**. Επομένως καλούμε πάλι την συνάρτηση ***bisection*** αυτή την φορά όμως με ορίσματα **f(x), a = 1.25, b = 1.75** .

```
root, iterations_num = bisection(g, 1.25, 1.75)
```

Σχήμα 4: Παράδειγμα κλήσης της συνάρτησης ***bisection***

Από την οποία παίρνουμε τα εξής αποτελέσματα:

```
Root = 1.5301
Iterations = 17
```

Σχήμα 5: Αποτελέσματα κλήσης της συνάρτησης ***bisection*** στο διάστημα **[1.25,1.75]**.

Όπου παρατηρούμε ότι η ρίζα της **f(x)** στο διάστημα **[1.25,1.75]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **1.5301** καθώς και ότι η μέθοδος της διχοτόμησης χρειάστηκε **17** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Τέλος, μπορούμε εύκολα να αποδείξουμε ότι η τρίτη και τελευταία ρίζα της **f(x)** είναι το **0** καθώς **f(0) = 0**. Στην περίπτωση μας όμως δεν υπάρχει κανένα διάστημα που ταυτόχρονα να περιέχει το **0** και το θεώρημα **Bolzano** να εγγυάται την ύπαρξη τουλάχιστον μιας ρίζας καθώς δεν υπάρχουν **x1,x2** με

$$\mathbf{x1} \neq \mathbf{x2}, \quad \mathbf{x1}, \mathbf{x2} \in \textbf{[-2,2]}$$

και

$$\mathbf{f(x1)} * \mathbf{f(x2)} < 0$$

και

$$\mathbf{0} \in \mathbf{[x1,x2]}$$

Συνεπώς η κλασσική μέθοδος της διχοτόμησης δεν εγγυάται σύγκλιση σε αυτή την περίπτωση καθώς και δεν μπορεί να χρησιμοποιηθεί για την εύρεση της ρίζας $\mathbf{x} = \mathbf{0}$ αφού προϋποθέτει την ύπαρξη ενός διαστήματος στο οποίο να ισχύει το θεώρημα **Bolzano** για την $\mathbf{f(x)}$. Αν προσπαθήσουμε να καλέσουμε την συνάρτηση ***bisection*** για ένα διάστημα που δεν ισχύει το θεώρημα **Bolzano** τα αποτελέσματα είναι μη προβλέψιμα και ανάλογα την περίπτωση και τις τιμές του διαστήματος **[a,b]** η μέθοδος μπορεί να μην συγκλίνει σε σωστή λύση ή να συγκλίνει αλλά όχι με την επιθυμητή ακρίβεια.

```
Root = 0.05000
Iterations = 15
```

Σχήμα 6: Αποτελέσματα κλήσης της συνάρτησης ***bisection*** στο διάστημα **[-0.05,0.05]**.

```
Root = 0.50000
Iterations = 18
```

Σχήμα 7: Αποτελέσματα κλήσης της συνάρτησης ***bisection*** στο διάστημα **[-0.5,0.5]**.

Η συνάρτηση ***bisection*** που έχει υλοποιηθεί στο αρχείο ***bisection.py*** ελέγχει αν στο διάστημα **[a,b]** ισχύει

$$\mathbf{f(x1)} * \mathbf{f(x2)} > 0$$

και αν ναι επιστρέφει την τιμή **nan** ως ρίζα και την τιμή **-1** ως τον αριθμό των επαναλήψεων. Η συνάρτηση έχει τροποποιηθεί επίσης ώστε να ελέγχει αν τα άκρα του αρχικού διαστήματος **[a,b]** αποτελούν ρίζες της **f(x)**. Αν προσπαθήσουμε τώρα να καλέσουμε την συνάρτηση ***bisection*** στην τροποποιημένη μορφή της παρατηρούμε τα εξής:

```
Root = nan
Iterations = -1
```

Σχήμα 8: Αποτελέσματα κλήσης της τροποποιήμενης συνάρτησης ***bisection*** στο διάστημα **[-0.5,0.5]**.

```
Root = 0.00000
Iterations = 0
```

Σχήμα 9: Αποτελέσματα κλήσης της τροποποιήμενης συνάρτησης ***bisection*** στο διάστημα **[0,1]**.

όπου στην πρώτη κλήση δεν ισχύει το θέωρημα **Bolzano** ενώ στην δεύτερη το ένα άκρο του διαστήματος είναι ρίζα της **f(x)**. Τέλος, τονίζεται ότι αν και τα δύο άκρα του διαστήματος είναι ρίζες της **f(x)** επιστρέφεται η τιμή του **a**.

## 2.2 Μέθοδος Newton-Raphson

Η μέθοδος **Newton-Raphson** αποτελεί μία περίπτωση μεθόδου σταθερού σημείου. Προφανώς προϋποθέτει κι αυτή, όπως και η μέθοδος της διχοτόμησης, την ύπαρξη ενός διαστήματος **[a,b]** στο οποίο η συνάρτηση έχει τουλάχιστον μία ρίζα. Κατ΄ αναλογία με την μέθοδο της διχοτόμησης για να βρούμε και πάλι ένα τέτοιο διάστημα χρησιμοποιούμε το θεώρημα **Bolzano** σε συνδυασμό με την γραφική παράσταση της **f(x)**. Εκτός από την ύπαρξη διαστήματος με τις προαναφερόμενες ιδιότητες, η μέθοδος **Newton-Raphson** απαιτεί η εκάστοτε συνάρτηση **f(x)** να είναι δύο φορές παραγωγίσιμη σε αυτό το διάστημα με

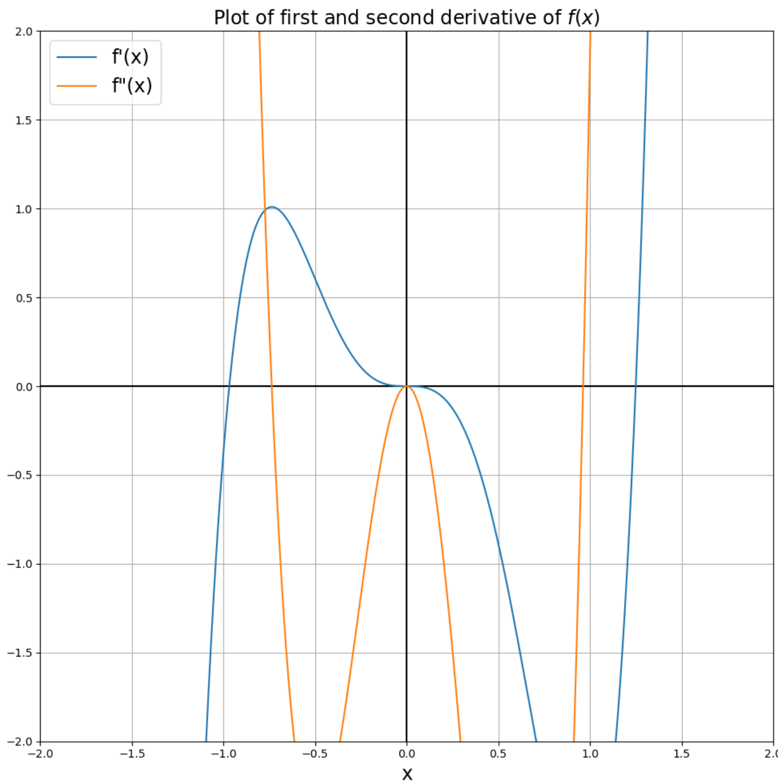
$$f'(x), f''(x) \neq \mathbf{0} \quad \forall \mathrm{x} \in \mathbf{[a,b]}$$

Καθώς και ένα αρχικό σημείο $x_0$ από το οποίο θα ξεκινήσει η επαναληπτική μέθοδος με την εξής ιδιότητα

$$\mathrm{f}(x_0) * f''(x_0) > \mathbf{0}$$

Όμοια με πριν παρατηρούμε στο *Σχήμα 1* ότι η δοσμένη **f(x)** έχει μία ρίζα στο διάστημα **[-1.5,-1.0]** καθώς και ότι ισχύει το θεώρημα **Bolzano** σε αυτό το διάστημα. Επίσης, παρατηρούμε στο *Σχήμα 10* ότι ισχύει

$$f'(x), f''(x) \neq \mathbf{0} \quad \forall \mathrm{x} \in [-1.5,-1.0]$$




Σχήμα 10: Γραφική παράσταση της πρώτης και δεύτερης παραγώγου της συνάρτησης **f(x)**

Επομένως αρκεί να βρούμε ένα σημείο $x_0$ με

$$x_0 \in \textbf{[-1.5,-1.0]}$$

τέτοιο ώστε να ικανοποιείται η συνθήκη

$$\mathrm{f}(x_0) * f''(x_0) > \mathbf{0}$$

Ένα τέτοιο σημείο είναι το

$$x_0 = \textbf{-1.4}$$

Για να βρούμε την ρίζα χρησιμοποιούμε την συνάρτηση ***newton_raphson*** από το αρχείο ***newton_raphson.py*** με ορίσματα την συνάρτηση **f(x)**, την παράγωγο της **f'(x)** και το αρχικό σημείο $x_0$= **-1.4**, το **default** όρισμα **eps** έχει την τιμή που χρειάζεται για ακρίβεια **5** δεκαδικών ψηφίων ενώ το **default** όρισμα **max_iterations** αντιστοιχεί στον μέγιστο αριθμό επαναλήψεων που θα γίνουν σε περίπτωση που έχει δοθεί κάποιο αρχικό σημείο $x_0$ χωρίς να τηρούνται οι προϋποθέσεις που εγγυώνται σύγκλιση για την μέθοδο **Newton-Raphson**, με τιμή τον αριθμό **50**. Αν πληρούνται όλες οι προϋποθέσεις η συνάρτηση επιστρέφει την ρίζα της **f(x)** στο αντίστοιχο διάστημα **(a,b)** ( εδώ στο **(-1.5,-1.0)** ), αφού οι προαναφερόμενες προϋποθέσεις εγγυώνται την ύπαρξη **μοναδικής** ρίζας στο διάστημα αυτό, καθώς και τον αριθμό των επαναλήψεων που χρειάστηκαν για να βρέθει η ρίζα με σφάλμα λιγότερο από **eps**.

```
root, iterations_num = newton_raphson(g, gprime, -1.4)
```

Σχήμα 11: Παράδειγμα κλήσης της συνάρτησης ***newton_raphson*** με αρχική τιμή $x_0$ = **-1.4**.

Μετά την κλήση της συνάρτησης ***newton_raphson*** τα αποτελέσματα είναι τα εξής:

```
Root = -1.1976
Iterations = 4
```

Σχήμα 12: Αποτελέσματα κλήσης της συνάρτησης ***newton_raphson*** με αρχική τιμή $x_0$ = **-1.4**.

Όπου παρατηρούμε ότι η ρίζα της **f(x)** στο διάστημα **[-1.5,-1.0]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **-1.1976** καθώς και ότι η μέθοδος **Newton-Raphson** χρειάστηκε **4** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Στην συνέχεια παρατηρούμε στο *Σχήμα 1* ότι η **f(x)** έχει μία ακόμα ρίζα στο διάστημα **[1.25,1.75]** και πως ισχύει το θέωρημα **Bolzano** σε αυτό το διάστημα, όμως δεν ικανοποιείται η πρώτη συνθήκη οπότε θα χρειαστεί να περιορίσουμε το διάστημα στο **[1.3,1.75]** όπου έχουμε

$$f'(x), f''(x) \neq \mathbf{0} \quad \forall \mathrm{x} \in \mathbf{[1.3,1.75]}$$

Επομένως, το επόμενο βήμα είναι η επιλογή του αρχικού σημείου $x_0$ έτσι ώστε να ικανοποιείται και η τρίτη συνθήκη. Ένα τέτοιο σημείο είναι το $x_0 = \mathbf{1.7}$. Καλούμε λοιπόν την συνάρτηση ***newton_raphson*** με αρχικό σημείο το $x_0 = \mathbf{1.7}$.

```
root, iterations_num = newton_raphson(g, gprime, 1.7)
```

Σχήμα 13: Παράδειγμα κλήσης της συνάρτησης ***newton_raphson*** με αρχική τιμή $x_0 = \mathbf{1.7}$.

Μετά την κλήση της συνάρτησης ***newton_raphson*** τα αποτελέσματα είναι τα εξής:

```
Root = 1.5301
Iterations = 4
```

Σχήμα 14: Αποτελέσματα κλήσης της συνάρτησης ***newton_raphson*** με αρχική τιμή $x_0 = \mathbf{1.7}$.

Όπου παρατηρούμε ότι η ρίζα της **f(x)** στο διάστημα **[1.3,1.75]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **1.5301** καθώς και ότι η μέθοδος **Newton-Raphson** χρειάστηκε **4** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Τέλος, αφού η μέθοδος **Newton-Raphson** προϋποθέτει κι αυτή, όπως και η μέθοδος της διχοτόμησης, την ύπαρξη διαστήματος στο οποίο ισχύει το θεώρημα **Bolzano** για να εγγυάται η σύγκλιση ( εφόσον ισχύουν και οι υπόλοιπες προϋποθέσεις ) αντιμετωπίζουμε πάλι πρόβλημα με την ρίζα $\mathbf{x} = \mathbf{0}$. Αν προσπαθήσουμε να καλέσουμε την συνάρτηση ***newton_ raphson*** με αρχικό σημείο κάποιο σημείο στο οποίο δεν τηρούνται όλες οι προϋποθέσεις τα αποτελέσματα είναι μη προβλέψιμα και είτε μπορεί να έχουμε σύγκλιση σε λάθος αποτέλεσμα είτε σύγκλιση χωρίς την επιθυμητή ακρίβεια.

| $x_0$ | Αποτέλεσμα κλήσης συνάρτησης ***newton_ raphson*** | Αριθμός Επαναλήψεων |
|---|---|---|
| 0.25 | 0.00005 | 29 |
| 0.5 | 0.00009 | 30 |
| -0.5 | -0.00005 | 31 |
| -0.25 | -0.00004 | 29 |

Σχήμα 15: Αποτέλεσματα κλήσεων της συνάρτησης ***newton_ raphson*** χωρίς να τηρούνται όλες οι προϋποθέσεις

Όπως παρατηρούμε και στο σχήμα 15, καμία κλήση δεν βρίσκει την ρίζα της $\mathbf{f(x)}$ με την επιθυμητή ακρίβεια. Η συνάρτηση ***newton_ raphson*** που έχει υλοποιηθεί στο αρχείο ***newton_ raphson.py*** έχει τροποποιηθεί έτσι ώστε να ελέγχει αν το αρχικό σημείο $x_0$ αποτελεί ρίζα της $\mathbf{f(x)}$. Έτσι, αν τώρα καλέσουμε την συνάρτηση ***newton_ raphson*** με αρχικό σημείο το $x_0$= $\mathbf{0}$ και παίρνουμε τα εξής αποτελέσματα:

```
Root = 0.00000
Iterations = 0
```

Σχήμα 16: Αποτελέσματα κλήσης της τροποποιήμενης συνάρτησης
***newton_ raphson***
με αρχικό σημείο $x_0 = \mathbf{0}$.

## 2.3 Μέθοδος της τέμνουσας

Η μέθοδος της τέμνουσας απότελει μία παραλλαγή της μεθόδου **Newton-Raphson** και χρησιμοποιείται συνήθως όταν είναι δύσκολο ή αδύνατο να υπολογιστεί η πρώτη παράγωγος **f(x)** και αλλάζει την πρώτη παράγωγο της **f(x)** στην αναδρομική ακολουθία της μεθόδου **Newton-Raphson** με την παρακάτω παράσταση :

$$\frac{f(x+h) - f(x)}{(x+h)-x} \quad ,\mathrm{h} \to 0$$

Οπότε τελικά η αναδρομική ακολουθία της μεθόδου της τέμνουσας είναι η παρακάτω :

$$x_{n+1} = x_n - \frac{f(x_n) * (x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

Οι προϋποθέσεις που εγγυώνται την σύγκλιση της μεθόδου της τέμνουσας είναι ίδιες με αυτές της μεθόδου **Newton-Raphson** με την διαφορά ότι απαιτούνται δύο αρχικά σημεία $x_0$ και $x_1$. Συνήθως τα δύο αρχικά αυτά σημεία είναι τα άκρα του διαστήματος **[a,b]** όπου εφαρμόζουμε το θεώρημα **Bolzano**. Για την ρίζα της **f(x)** στο διάστημα **[-1.5,-1.0]** καλούμε την συνάρτηση ***secant*** από το αρχείο ***secant.py*** με ορίσματα την συνάρτηση **f(x)**, και τα δύο αρχικά σημεία $x_0$= **-1.5** και $x_1$= **-1.0** καθώς πληρούνται όλες οι προϋποθέσεις της μεθόδου της τέμνουσας σε αυτό το διάστημα, το **default** όρισμα **eps** έχει την τιμή που χρειάζεται για ακρίβεια **5** δεκαδικών ψηφίων ενώ το **default** όρισμα **max_iterations** αντιστοιχεί στον μέγιστο αριθμό επαναλήψεων που θα γίνουν σε περίπτωση που έχουν δοθεί αρχικά σημεία $x_0$ και $x_1$ χωρίς να τηρούνται οι προϋποθέσεις που εγγυώνται σύγκλιση , με τιμή τον αριθμό **50**. Αν πληρούνται όλες οι προϋποθέσεις η συνάρτηση επιστρέφει την ρίζα της **f(x)** στο αντίστοιχο διάστημα **(a,b)** ( εδώ στο **(-1.5,-1.0)** ), αφού οι προαναφερόμενες προϋποθεσεις εγγυώνται την ύπαρξη **μοναδικής** ρίζας στο διάστημα αυτό, καθώς και τον αριθμό των επαναλήψεων που χρειάστηκαν για να βρέθει η ρίζα με σφάλμα λιγότερο από **eps**.

```
root, iterations_num = secant(g, -1.5, -1.0)
```

Σχήμα 17: Παράδειγμα κλήσης της συνάρτησης ***secant*** με αρχικά σημεία $x_0$ = **-1.5** και $x_1$ = **-1.0** .

Μετά την κλήση της συνάρτησης ***secant*** τα αποτελέσματα είναι τα εξής:

```
Root = -1.1976
Iterations = 10
```

Σχήμα 18: Αποτελέσματα κλήσης της συνάρτησης ***secant*** με αρχικά σημεία $x_0$ = **-1.5** και $x_1$ = **-1.0** .

Όπου παρατηρούμε ότι η ρίζα της **f(x)** στο διάστημα **[-1.5,-1.0]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **-1.1976** καθώς και ότι η μέθοδος της τέμνουσας χρειάστηκε **10** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Στην συνέχεια για την επόμενη ρίζα της **f(x)** επιλέγουμε το διάστημα **[1.3,1.9]** και καλούμε την συνάρτηση ***secant*** με ορίσματα την συνάρτηση **f(x)**, και τα δύο αρχικά σημεία $x_0$= **1.3** και $x_1$= **1.9**.

```
root, iterations_num = secant(g, 1.3, 1.9)
```

Σχήμα 19: Παράδειγμα κλήσης της συνάρτησης ***secant*** με αρχικά σημεία $x_0$ = **1.3** και $x_1$ = **1.9** .

Μετά την κλήση τα αποτελέσματα είναι τα εξής:

```
Root = 1.5301
Iterations = 8
```

Σχήμα 20: Αποτελέσματα κλήσης της συνάρτησης ***secant*** με αρχικά σημεία $x_0$ = **1.3** και $x_1$ = **1.9** .

Ώπου παρατηρούμε ότι η ρίζα της $\mathbf{f(x)}$ στο διάστημα **[1.3,1.9]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **1.5301** καθώς και ότι η μέθοδος της τέμνουσας χρειάστηκε **8** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Τέλος, με την μέθοδο της τέμνουσας αντιμετωπίζουμε το ίδιο πρόβλημα που αντιμετωπίζουν και οι δύο προαναφερόμενες μέθοδοι για την ρίζα $\mathbf{x} = \mathbf{0}$. Αν προσπαθήσουμε να καλέσουμε την συνάρτηση ***secant*** με αρχικά σημεία κάποια σημεία στα οποία δεν τηρούνται όλες οι προϋποθέσεις τα αποτελέσματα είναι μη προβλέψιμα και είτε μπορεί να έχουμε σύγκλιση σε λάθος αποτέλεσμα είτε σύγκλιση χωρίς την επιθυμητή ακρίβεια.

| $x_0$ | $x_1$ | Αποτέλεσμα κλήσης συνάρτησης ***secant*** | Αριθμός Επαναλήψεων |
|---|---|---|---|
| -0.25 | 0.25 | -0.00002 | 44 |
| -0.5 | 0.5 | 0.00008 | 46 |

Σχήμα 21: Αποτέλεσματα κλήσεων της συνάρτησης ***secant*** χωρίς να τηρούνται όλες οι προϋποθέσεις

Όπως παρατηρούμε και στο σχήμα 20, καμία κλήση δεν βρίσκει την ρίζα της $\mathbf{f(x)}$ με την επιθυμητή ακρίβεια. Η συνάρτηση ***secant*** που έχει υλοποιηθεί στο αρχείο ***secant.py*** έχει τροποποιηθεί έτσι ώστε να ελέγχει αν κάποιο από τα αρχικά σημεία $x_0$,$x_1$ αποτελεί ρίζα της $\mathbf{f(x)}$. Έτσι, αν τώρα καλέσουμε την συνάρτηση ***secant*** με αρχικά σημεία τα $x_0 = \mathbf{0}$ και $x_1 = \mathbf{1.5}$ παίρνουμε τα εξής αποτελέσματα:

```
Root = 0.00000
Iterations = 0
```

Σχήμα 22: Αποτελέσματα κλήσης της τροποποιήμενης συνάρτησης ***secant*** με αρχικά σημεία $x_0 = \mathbf{0}$ και $x_1 = \mathbf{1.5}$ .

Όπου το $x_0 = \mathbf{0}$ αποτελεί ρίζα της $\mathbf{f(x)}$. Τέλος, παρόμοια με την μέθοδο της διχοτόμησης αν και τα δύο αρχικά σημεία $x_0$,$x_1$ αποτελούν ρίζες επιστρέφεται η τιμή του $x_0$.

## 2.4 Υπολογισμός σύγκλισης της μεθόδου Newton-Raphson για κάθε ρίζα

Για την μέθοδο **Newton-Raphson** ζητείται επίσης να δειχθεί για ποιες ρίζες συγκλίνει τετραγωνικά και για ποιες όχι. Αρχικά αναφέρουμε τις τρεις ρίζες της συνάρτησης **f(x)** με ακρίβεια 5ου ψηφίου. Στην συνέχεια αναφέρουμε το θεώρημα

| |
|---|
| $x_0$ = **-1.1976** |
| $x_1$ = **1.5301** |
| $x_2$ = **0.0000** |

Σχήμα 23: Ρίζες της συνάρτησης f(x) στο διάστημα [-2,2]

που θα χρησιμοποιηθεί για τον υπολογισμό της τάξης σύγκλισης της κάθε ρίζας.

**Θεώρημα :** Έστω ότι $x_n \neq x^*$ για κάθε φυσικό αριθμό n, όπου $x^*$ σταθερό σημείο της συνάρτησης **φ(x)** που χρησιμοποιείται στην μέθοδο σταθερού σημείου, και έστω ότι ισχύει :

$$\lim_{n\to\infty} \frac{x_{n+1} - x^*}{(x_n - x^*)^p} = a \neq 0,$$

τότε η τάξη σύγκλισης της ακολουθίας $x_n$ είναι ακριβώς **p**.

Πιο συγκεκριμένα για την μέθοδο **Newton-Raphson** χρησιμοποιώντας το θεώρημα **Taylor** με κέντρο το σημείο $x^*$, όπου $x^*$ ρίζα της **f(x)**, αποδεικνύεται ότι για την αναδρομική ακολουθία της μεθόδου **Newton-Raphson** το προηγούμενο όριο ισούται για $p = 2$ με :

$$\frac{f''(x^*)}{2f'(x^*)} \neq 0,$$

εφόσον $f'(x^*) \neq 0$.

Με αυτά που αναφέρθηκαν πλέον μπορούμε να προχωρήσουμε στον υπολογισμό της σύγκλισης της μεθόδου **Newton-Raphson** για κάθε μία από τις τρεις ρίζες.

Υπολογίζουμε αρχικά την πρώτη και την δεύτερη παράγωγο της **f(x)** στο διάστημα [**-2.2**]:

$$f'(x) = 3e^{sin^3 x} * sin^2 x * cosx + 6x^5 - 8x^3 - 3x^2$$

$$f''(x) = 9e^{sin^3 x} * sin^4 x * cos^2 x + 3e^{sin^3 x} * sin(2x) * cosx - 3e^{sin^3 x} * sin^3 x + 30x^4 - 24x^2 - 6x$$

- $x_0$ **= -1.1976**

  Ισχύει

  $$f'(-1.1976) \approx 35.62888 \neq 0,$$

  επομένως μπορούμε να χρησιμοποιήσουμε τον προηγούμενο τύπο για την σύγκλιση της μεθόδου **Newton-Raphson** για την ρίζα $x_0$. Οπότε έχουμε

  $$\lim_{n\to\infty} \frac{x_{n+1} - x_0}{(x_n - x_0)^2} = \frac{f''(x_0)}{2f'(x_0)} = \frac{35.62888}{-2 * 4.92044} = -3.62049 \neq 0$$

  επομένως η μέθοδος **Newton-Raphson** συγκλίνει τετραγωνικά για την ρίζα $x_0$ **= -1.1976**.

- $x_1$ **= 1.5301**

  Ισχύει

  $$f'(1.5301) \approx 14.97241 \neq 0,$$

  επομένως μπορούμε να χρησιμοποιήσουμε τον προηγούμενο τύπο για την σύγκλιση της μεθόδου **Newton-Raphson** για την ρίζα $x_1$. Οπότε έχουμε

  $$\lim_{n\to\infty} \frac{x_{n+1} - x_1}{(x_n - x_1)^2} = \frac{f''(x_1)}{2f'(x_1)} = \frac{91.03090}{2 * 14.97241} = 3.03995 \neq 0$$

  επομένως η μέθοδος **Newton-Raphson** συγκλίνει τετραγωνικά για την ρίζα $x_1 = \mathbf{1.5301}$.

- $x_2 = \mathbf{0.0000}$

  Ισχύει

  $$f'(0.0000) = 0.0000 = 0,$$

  επομένως δεν μπορούμε να χρησιμοποιήσουμε τον προηγούμενο τύπο για την σύγκλιση της μεθόδου **Newton-Raphson** για την ρίζα $x_2$. Αν υπολογίσουμε και την δεύτερη και τρίτη παράγωγο της $\mathbf{f(x)}$ παρατηρούμε ότι κι αυτές έχουν ρίζα το $x_2 = \mathbf{0.0000}$, επομένως η ρίζα $x_2$ έχει πολλαπλότητα $m = \mathbf{4}$. Χρησιμοποιώντας το ακόλουθο θεώρημα

  **Θεώρημα :** Έστω μία συνάρτηση $\mathbf{f(x)}$ $(m+1)$ φορές συνεχώς παραγωγίσιμη στο διάστημα **[a,b]** κι έστω ότι η συνάρτηση έχει ως ρίζα το σημείο $x = \mathbf{r}$ με πολλαπλότητα $m$, τότε η μέθοδος **Newton-Raphson** συγκλίνει τοπικά στο σημείο $x = \mathbf{r}$ και το σφάλμα $e_i$ στην **i**-οστή επανάληψη ικανοποιεί το παρακάτω

  $$\lim_{n\to\infty} \frac{e_{i+1}}{x_i} = S,$$

  όπου

  $$S = \frac{m-1}{m}.$$

  Χρησιμοποιώντας το παραπάνω θεώρημα έχουμε $S = \frac{3}{4}$ οπότε η μέθοδος **Newton-Raphson** συγκλίνει γραμμικά με $e_{i+1} \approx \frac{3}{4} * e_i$. Αν η πολλαπλότητα μιας ρίζας $\mathbf{r}$ είναι γνωστή από πριν μπορούμε να βελτιώσουμε την απόδοση της κλασσικής μεθόδου **Newton-Raphson** με μία μικρή τροποποίηση :

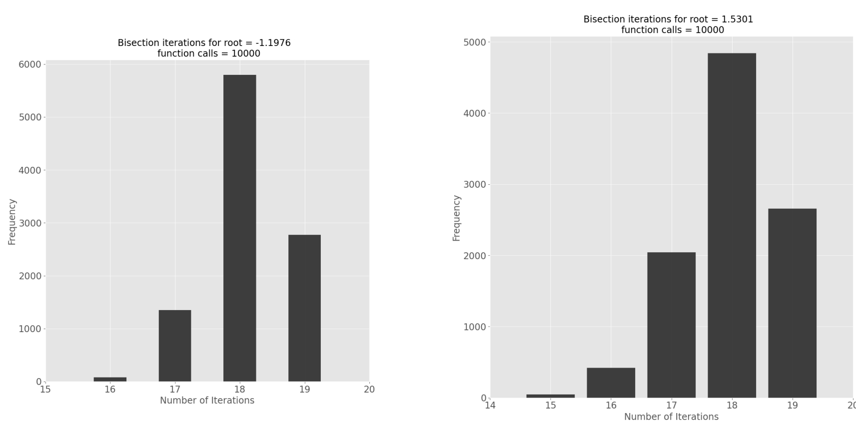
  $$x_{n+1} = x_n - \frac{m * f(x_n)}{f'(x_n)}$$

  όπου αποδεικνύεται ότι συγκλίνει τετραγωνικά στην ρίζα $\mathbf{r}$. Μία τέτοια συνάρτηση έχει υλοποιηθεί στο αρχείο ***mult_newton_raphson.py*** με όνομα ***mult_ newton_ raphson***, όπου έχει προστεθεί ένα ακόμα όρισμα **m** που δηλώνει την πολλαπλότητα της ρίζας που θέλουμε να υπολογιστεί από την συνάρτηση.

## 2.5 Σύγκριση επαναλήψεων για κάθε ρίζα

Σε αυτή την παράγραφο γίνεται μία σύγκριση των επαναλήψεων που χρειάστηκε κάθε μέθοδος για την εύρεση των τριών ριζών. Για να είναι τα αποτελέσματα της σύγκρισης αυτής πιο αντιπροσωπευτικά χρησιμοποιήθηκε δείγμα 10000 κλήσεων των συναρτήσεων που υλοποιούν την κάθε μέθοδο σε κατάλληλα διαστήματα και αρχικά σημεία και όχι τα μεμονωμένα ορίσματα που χρησιμοποιηθήκαν στις παραγράφους 2.1, 2.2 και 2.3. Εφόσον για την ρίζα $x=$ **0** δεν μπορούν να βρεθούν κατάλληλα διαστήματα και αρχικά σημεία έτσι ώστε να τηρούνται οι προϋποθέσεις για κάθε μέθοδο, η ρίζα αυτή δεν συμπεριλαμβάνεται σε αυτή την ανάλυση.

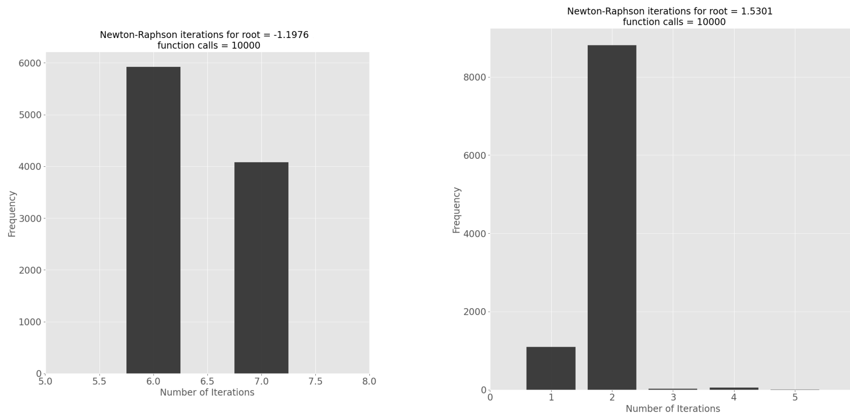
### 2.5.1 Μέθοδος της διχοτόμησης




Σχήμα 24: Ιστογράμματα συχνοτήτων του αριθμού των επαναλήψεων για την μέθοδο της διχοτόμησης.

Όπως παρατηρούμε από τα ιστογράμματα στο *Σχήμα 24* η μέθοδος της διχοτόμησης αποδίδει σε αριθμούς επαναλήψεων γύρω από το 18.

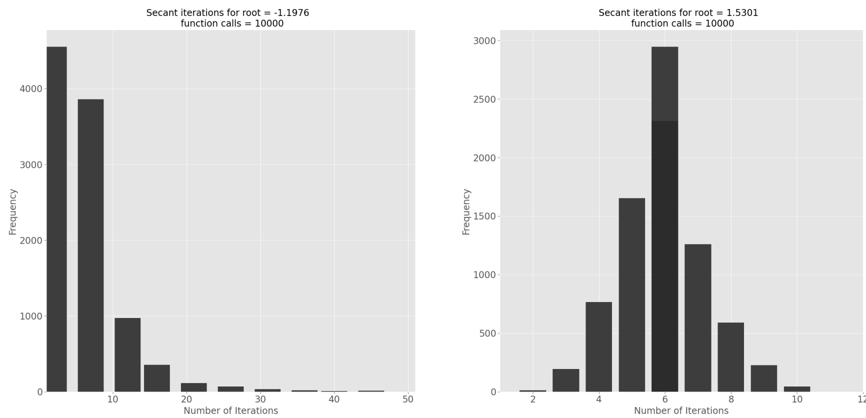
### 2.5.2 Μέθοδος Newton-Raphson




Σχήμα 25: Ιστογράμματα συχνοτήτων του αριθμού των επαναλήψεων για την μέθοδο Newton-Raphson.

Όπως παρατηρούμε από τα ιστογράμματα στο *Σχήμα 25* η μέθοδος Newton-Raphson αποδίδει κυρίως σε 2,6 και 7 αριθμούς επαναλήψεων.

### 2.5.3 Μέθοδος της τέμνουσας




Σχήμα 26: Ιστογράμματα συχνοτήτων του αριθμού των επαναλήψεων για την μέθοδο της τέμνουσας.

Όπως παρατηρούμε από τα ιστογράμματα στο *Σχήμα 26* η μέθοδος της τέμνουσας αποδίδει κυρίως σε αριθμούς επαναλήψεων κάτω από 10, με τις περισσότερες απ΄ αυτές να είναι γύρω από το 6, αλλά σε κάποιες περιπτώσεις ( περίπου **20%** του δείγματος ) ξεπερνάει τις 10.

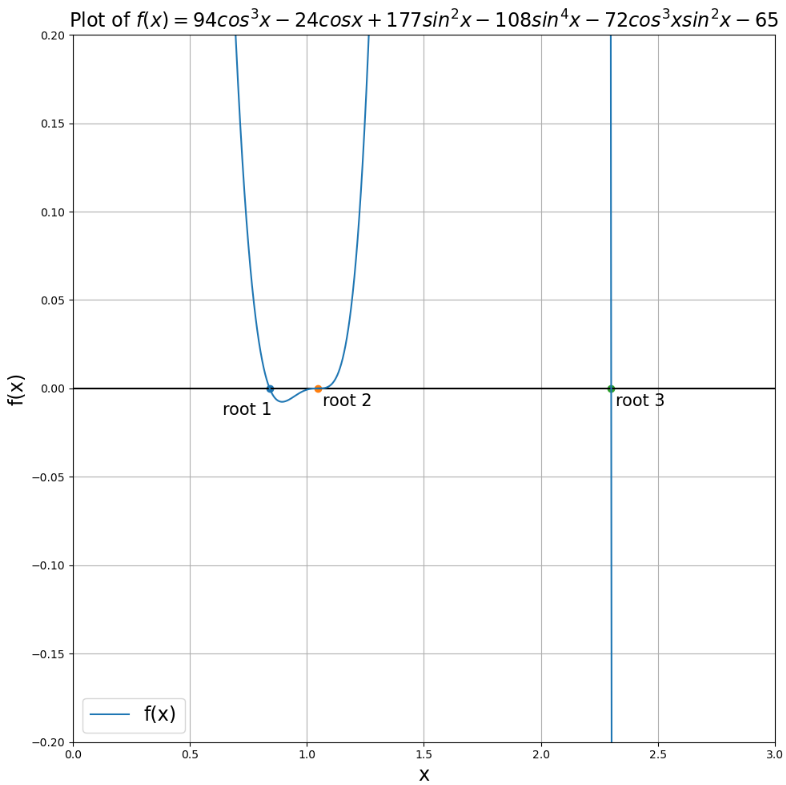
**Συμπερασματικά**, έχουμε σύμφωνα με το δείγμα ότι έφοσον πληρούνται οι αντίστοιχες προϋποθέσεις για όλες τις μεθόδους η αποδοτικότερη και πιο σταθερή στην απόδοση της από τις μεθόδους ως προς τον αριθμό των επαναλήψεων είναι η μέθοδος **Newton-Raphson**. Στην συνέχεια, λιγότερο αποδοτικότερη είναι η μέθοδος της **τέμνουσας** ενώ τέλος την χειρότερη απόδοση έχει η μέθοδος της **διχοτόμησης**. Εξάλλου, η τάξη σύγκλισης της μεθόδου **Newton-Raphson** είναι $\mathbf{p = 2}$, της μεθόδου της **τέμνουσας** $\mathbf{p \approx 1.62}$ ενώ της μεθόδου της **διχοτόμησης** $\mathbf{p = 1}$, οπότε τα συμπεράσματα από τα ιστογράμματα αιτιολογούνται από τις τάξεις σύγκλισης της κάθε μεθόδου.

# 3 Άσκηση 2

Στην **2η άσκηση** ζητείται να υλοποιηθούν κάποιες παραλλαγές των μεθόδων της **1ης άσκησης** ενώ στην συνέχεια μας δίνεται η συνάρτηση

$$f(x) = 94cos^3x - 24cosx + 177sin^2x - 108sin^4x - 72cos^3xsin^2x - 65$$

και ζητείται να βρεθούν όλες οι ρίζες της στο διάστημα $[\mathbf{0,3}]$. Αρχικά, σχηματίζουμε και πάλι την γραφική παράστασης της νέας συνάρτησης :

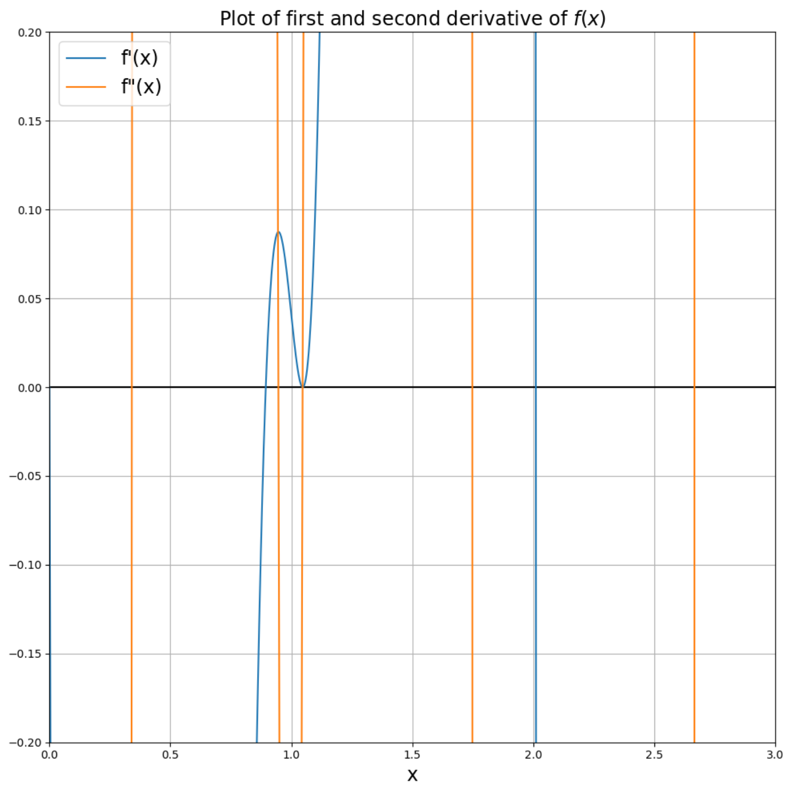



Σχήμα 27: Γραφική παράσταση της συνάρτησης $\mathbf{f(x)}$

Από την οποία παρατηρούμε στο *Σχήμα 27* ότι η συνάρτηση έχει 3 ρίζες στο διάστημα $[\mathbf{0,3}]$.

## 3.1 Τροποποιημένη μέθοδος Newton-Raphson

Στην τροποποιημένη μέθοδο Newton-Raphson αλλάζει ο αναδρομικός τύπος της ακολουθίας για την εκτίμηση της ρίζας, ενώ οι προϋποθέσεις της μεθόδου παραμένουν ίδιες. Η συνάρτηση ***modified_newton_raphson*** από το αρχείο ***modified_newton_raphson.py*** δέχεται τα ίδια ορίσματα με την κλασσική μέθοδο Newton-Raphson.




Σχήμα 28: Γραφική παράσταση της πρώτης και δεύτερης παραγώγου της συνάρτησης $\mathbf{f(x)}$

Για την πρώτη ρίζα της $\mathbf{f(x)}$ στο διάστημα επιλέγουμε $x_0 = \mathbf{0.5}$ όπου τηρούνται όλες οι προϋποθέσεις για την μέθοδο **Newton-Raphson**. Μετά την κλήση της συνάρτησης έχουμε τα παρακάτω αποτελέσματα :

```
root, iterations_num = modified_newton_raphson(f, fprime, f_second_der, 0.5)
```

Σχήμα 29: Παράδειγμα κλήσης της συνάρτησης ***modified_newton_raphson***.

```
Root = 0.84107
Iterations = 5
```

Σχήμα 30: Αποτελέσματα κλήσης της συνάρτησης
***modified_newton_raphson***
με αρχικό σημείο $x_0 = \mathbf{0.5}$
.

Όπου παρατηρούμε ότι η ρίζα της **f(x)** στο διάστημα **[0.5,1.0]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **0.84107** καθώς και ότι η τροποποιημένη μέθοδος **Newton-Raphson** χρειάστηκε **5** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Για την ρίζα της συνάρτησης στο διάστημα **[2.0,2.5]** επιλέγουμε $x_0 = \mathbf{2.5}$ όπου τηρούνται όλες οι προϋποθέσεις για την μέθοδο **Newton-Raphson**. Μετά την κλήση της συνάρτησης έχουμε τα παρακάτω αποτελέσματα :

```
root, iterations_num = modified_newton_raphson(f, fprime, f_second_der, 2.5)
```

Σχήμα 31: Παράδειγμα κλήσης της συνάρτησης ***modified_newton_raphson***.

```
Root = 2.3005
Iterations = 3
```

Σχήμα 32: Αποτελέσματα κλήσης της συνάρτησης
***modified_newton_raphson***
με αρχικό σημείο $x_0 = \mathbf{0.5}$
.

Όπου παρατηρούμε ότι η ρίζα της **f(x)** στο διάστημα **[2.0,2.5]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **2.3005** καθώς και ότι η τροποποιημένη μέθοδος **Newton-Raphson** χρειάστηκε **3** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Τέλος, για την ρίζα της **f(x)** στο διάστημα **[1.0,1.25]** αν υπολογίσουμε τις τιμές της πρώτης και δεύτερης παραγώγου θα δούμε ότι μηδενίζονται και αυτές. Επομένως, η ρίζα αυτή έχει πολλαπλότητα $\mathbf{m} = \mathbf{3}$ και δεν τηρούνται όλες οι προϋποθέσεις για την μέθοδο **Newton-Raphson** και είτε δεν θα έχουμε καθόλου σύγκλιση είτε σύγκλιση αλλά όχι με την επιθυμητή ακρίβεια. Για παράδειγμα, αν δοκίμασουμε να καλέσουμε την συνάρτηση με αρχικό σημείο το $x_0 = \mathbf{1.04}$ ( ένα σημείο αρκετά κοντά στην ρίζα ) :

```
Root = 1.0471
Iterations = 11
```

Σχήμα 33: Αποτελέσματα κλήσης της συνάρτησης
***modified_ newton_ raphson***
με αρχικό σημείο $x_0 = \mathbf{1.04}$
.

Όπου παρατηρούμε ότι δεν έχουμε την επιθυμητή ακρίβεια.

## 3.2 Τροποποιημένη μέθοδος της διχοτόμησης

Στην τροποποιημένη μέθοδο της διχοτόμησης αυτό που αλλάζει είναι ο τρόπος που γίνεται η εκτίμηση της ρίζας από μία γεννήτρια παραγωγής τυχαίων αριθμών ενώ οι προϋποθέσεις της μέθοδου παραμένουν ίδιες. Η συνάρτηση ***modified_ bisection*** από το αρχείο ***modified_ bisection.py*** δέχεται τα ίδια ορίσματα με την κλασσική μέθοδο της διχοτόμησης, ενώ η εκτίμηση της ρίζας γίνεται από μία γεννήτρια παραγωγής τυχαίων αριθμών από ομοιόμορφη κατανομή. Ακόμα, αλλάζει ο τρόπος υπολογισμού του σφάλματος σε κάθε βήμα και οι επαναλήψεις συνεχίζονται μέχρι το μήκος του διαστήματος **[a,b]** να γίνει μικρότερο από το σφάλμα **eps**, με αυτό τον τρόπο το μεγαλύτερο σφάλμα που μπορεί να γίνει από την γεννήτρια παραγωγής τυχαίων αριθμών είναι **eps**, που είναι το επιθυμητό.Επομένως, έχουμε για την ρίζα στο διάστημα **[0.5,1.0]** τα παρακάτω :

```
root, iterations_num = modified_bisection(f, 0.5, 1.0)
```

Σχήμα 34: Παράδειγμα κλήσης της συνάρτησης ***modified_ bisection***.

Μετά την κλήση της συνάρτησης ***modified_ bisection*** τα αποτελέσματα είναι τα εξής:

```
Root = 0.84107
Iterations = 24
```

Σχήμα 35: Αποτελέσματα κλήσης της συνάρτησης ***modified_ bisection*** στο διάστημα **[0.5,1.0]**.

Όπου παρατηρούμε ότι η ρίζα της $\mathbf{f(x)}$ στο διάστημα **[0.5,1.0]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **0.84107** καθώς και ότι η τροποποιημένη μέθοδος της διχοτόμησης χρειάστηκε **24** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Για την ρίζα της συνάρτησης στο διάστημα **[1.0,1.25]** έχουμε

```
root, iterations_num = modified_bisection(f, 1.0, 1.25)
```

Σχήμα 36: Παράδειγμα κλήσης της συνάρτησης ***modified_ bisection***.

Μετά την κλήση της συνάρτησης ***modified_ bisection*** τα αποτελέσματα είναι τα εξής:

```
Root = 1.0472
Iterations = 22
```

Σχήμα 37: Αποτελέσματα κλήσης της συνάρτησης ***modified_ bisection*** στο διάστημα **[1.0,1.25]**.

Όπου παρατηρούμε ότι η ρίζα της $\mathbf{f(x)}$ στο διάστημα **[1.0,1.25]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **1.0472** καθώς και ότι η τροποποιημένη μέθοδος της διχοτόμησης χρειάστηκε **22** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Στην συνέχεια παρατηρούμε ότι η τελευταία ρίζα της συνάρτησης βρίσκεται στο διάστημα [**2.0**,**2.5**], αν υπολογίσουμε τις τιμές της συνάρτησης στα άκρα του διαστήματος έχουμε

$$f(2.0) * f(2.5) = 15.018 * -31.305 = -470.13 < 0$$

οπότε σύμφωνα με το θεώρημα **Bolzano** η συνάρτηση όντως έχει μία ρίζα σε αυτό το διάστημα.

```
root, iterations_num = bisection(f, 2.0, 2.5)
```

Σχήμα 38: Παράδειγμα κλήσης της συνάρτησης ***modified_ bisection***.

Μετά την κλήση της συνάρτησης ***modified_ bisection*** τα αποτελέσματα είναι τα εξής:

```
Root = 2.3005
Iterations = 28
```

Σχήμα 39: Αποτελέσματα κλήσης της συνάρτησης ***modified_ bisection*** στο διάστημα [**2.0**,**2.5**].

Όπου παρατηρούμε ότι η ρίζα της $\mathbf{f(x)}$ στο διάστημα [**2.0**,**2.5**] με ακρίβεια 5 δεκαδικών ψηφίων είναι η **2.3005** καθώς και ότι η τροποποιημένη μέθοδος της διχοτόμησης χρειάστηκε **28** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια.

## 3.3 Τροποποιημένη μέθοδος της τέμνουσας

Στην τροποποιημένη μέθοδο της τέμνουσας αλλάζει ο αναδρομικός τύπος της ακολουθίας για την εκτίμηση της ρίζας όπου χρησιμοποιούνται τρία αρχικά σημεία αντί για δύο, στην πραγματικότητα η μέθοδος βρίσκει μία παραβολή που περνάει και από τα τρία σημεία και ακολούθως βρίσκεται η τομή της παραβολής με τον άξονα $x$ όπου αποτελεί η νέα εκτίμηση της ρίζας, ενώ οι προϋποθέσεις της μεθόδου παραμένουν ίδιες(;). Η συνάρτηση ***modified_ secant*** από το αρχείο ***modified_ secant.py*** δέχεται τα ίδια ορίσματα με την κλασσική μέθοδο της τέμνουσας εκτός από την προσθήκη ενός ακόμα ορίσματος για το τρίτο αρχικό σημείο $x_3$. Ιδιαίτερη προσοχή χρειάζεται στην επιλογή των αρχικών σημείων έτσι ώστε οι τιμές της $\mathbf{f(x)}$ να είναι διακριτές και δύο από αυτές να έχουν αντίθετο πρόσημο καθώς αν για παράδειγμα στο πρώτο βήμα ισχύει $f(x_0) = f(x_2)$ δεν ορίζεται διαίρεση με το μηδέν και ο αλγόριθμος δεν μπορεί να συνεχίσει. Για την ρίζα της συνάρτησης στο διάστημα **[0.5,1.0]** επιλέγουμε τα αρχικά σημεία $x_0 = \mathbf{0,}$ $x_1 = \mathbf{0.5}$ και $x_2 = \mathbf{0.75}$.

```
root, iterations_num = modified_secant(f, 0, 0.5, 0.75)
```

Σχήμα 40: Παράδειγμα κλήσης της συνάρτησης ***modified_ secant***.

Μετά την κλήση τα αποτελέσματα είναι τα εξής:

```
Root = 0.84107
Iterations = 6
```

Σχήμα 41: Αποτελέσματα κλήσης της συνάρτησης ***modified_ secant*** με αρχικά σημεία $x_0 = \mathbf{0}$ $x_1 = \mathbf{0.5}$ και $x_2 = \mathbf{0.75}$ .

Όπου παρατηρούμε ότι η ρίζα της $\mathbf{f(x)}$ στο διάστημα **[0.5,1.0]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **0.84107** καθώς και ότι η τροποποιήμενη μέθοδος της τέμνουσας χρειάστηκε **6** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Για την ρίζα της $\mathbf{f(x)}$ στο διάστημα **[2.0,2.5]** επιλέγουμε τα αρχικά σημεία $x_0 = \mathbf{1.5,}$ $x_1 = \mathbf{1.8}$ και $x_2 = \mathbf{2.3}$ κι έχουμε τα αποτελέσματα του *Σχήματος 42*.

Όπου παρατηρούμε ότι η ρίζα της $\mathbf{f(x)}$ στο διάστημα **[2.0,2.5]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **2.3005** καθώς και ότι η τροποποιήμενη μέθοδος

```
Root = 2.3005
Iterations = 3
```

Σχήμα 42: Αποτελέσματα κλήσης της συνάρτησης ***modified_ secant*** με αρχικά σημεία $x_0 = \mathbf{1.5}$ $x_1 = \mathbf{1.8}$ και $x_2 = \mathbf{2.3}$ .

της τέμνουσας χρειάστηκε **3** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Τέλος, για την ρίζα της $\mathbf{f(x)}$ στο διάστημα **[1.0,1.25]** επιλέγουμε τα αρχικά σημεία $x_0 = \mathbf{1.5},\ x_1 = \mathbf{1.8}$ και $x_2 = \mathbf{2.3}$ κι έχουμε τα παρακάτω αποτελέσματα :

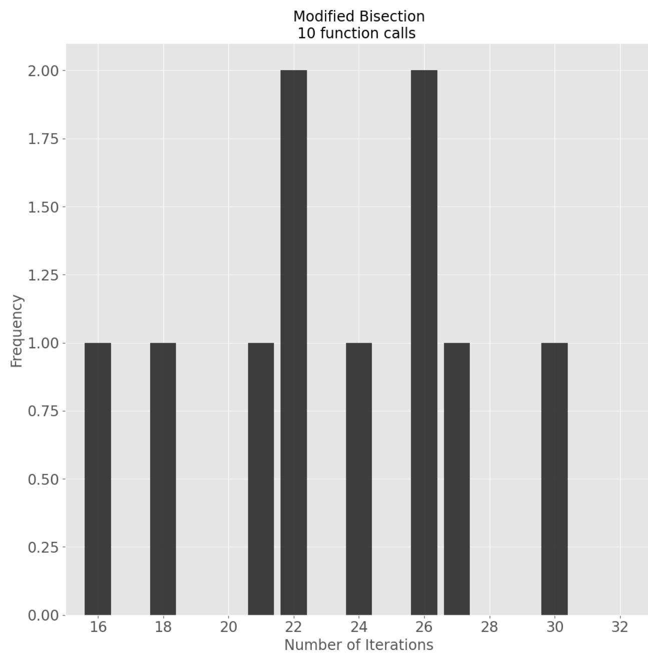
```
Root = 1.0472
Iterations = 26
```

Σχήμα 43: Αποτελέσματα κλήσης της συνάρτησης ***modified_ secant*** με αρχικά σημεία $x_0 = \mathbf{0.9}$ $x_1 = \mathbf{1.1}$ και $x_2 = \mathbf{1.2}$ .

Όπου παρατηρούμε ότι η ρίζα της $\mathbf{f(x)}$ στο διάστημα **[1.0,1.25]** με ακρίβεια 5 δεκαδικών ψηφίων είναι η **1.0472** καθώς και ότι η τροποποιήμενη μέθοδος της τέμνουσας χρειάστηκε **26** επαναλήψεις για να επιτύχει την επιθυμητή ακρίβεια. Παρατηρούμε, ότι σε αυτή την περίπτωση ο αλγόριθμος χρειάστηκε αρκετές παραπάνω επαναλήψεις από τις άλλες ρίζες και αυτό μάλλον οφείλεται στην πολλαπλότητα της συγκεκριμένης ρίζας και στην μορφή που έχει η συνάρτηση στην περιοχή γύρω από την ρίζα.

## 3.4 Σύγκλιση τροποποιήμενης μεθόδου της διχοτομήσης

Σε αυτή την παράγραφο εκτελούμε τον αλγόριθμο της τροποποιήμενης διχοτόμησης και αναλύουμε αν συγκλίνει πάντα σε ίδιο αριθμό επαναλήψεων. Υπενθυμίζεται ότι ο τροποποιήμενος αλγόριθμος της διχοτομήσης στην γενική περίπτωση εκτιμά την ρίζα μέσω μιας συνάρτησης παραγωγής τυχαίων αριθμών ενώ στην συγκεκριμένη υλοποίηση του αρχείου **modified_bisection.py** οι τυχαίοι αριθμοί προέρχονται από συνεχή ομοιόμορφη κατανομή. Εκτελούμε την συνάρτηση για την ρίζα $\mathbf{x = 2.3005}$ με ορίσματα $\mathbf{a = 2.0}$ και $\mathbf{b = 2.5}$ κι έχουμε τα παρακάτω αποτελέσματα :

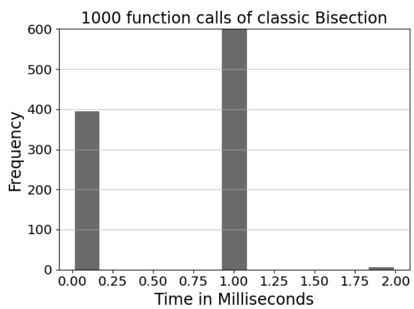


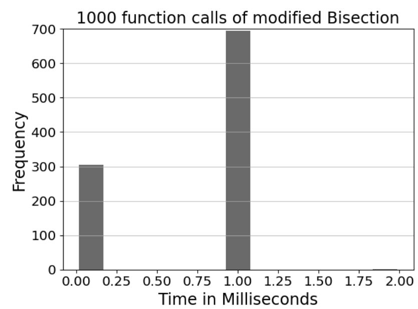

Σχήμα 44: Ιστογράμματα συχνοτήτων του αριθμού των επαναλήψεων για την τροποποιημένη μέθοδο της διχοτομήσης.

Από το ιστόγραμμα του *σχήματος 44* παρατηρούμε ότι ο αλγόριθμος κινείται σε αριθμούς επαναλήψεων 16 με 30 γι΄ αυτό το δείγμα κλήσεων της συνάρτησης και δεν συγκλίνει πάντα σε ίδιο αριθμό επαναλήψεων. Αυτό συμβαίνει προφανώς γιατί η εκτίμηση της ρίζας σε κάθε βήμα εξαρτάται από την συνάρτηση παραγωγής τυχαίων αριθμών.

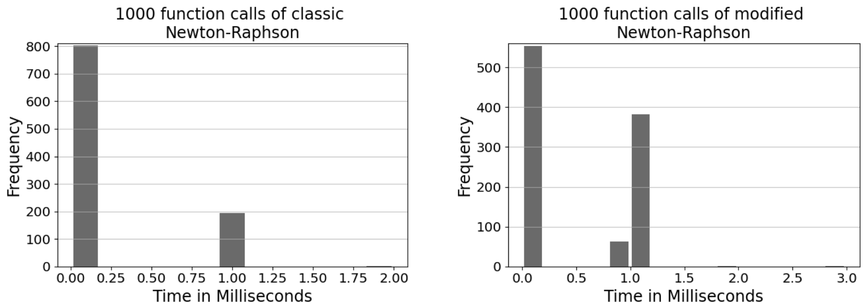
## 3.5 Πειραματική σύγκριση της ταχύτητας σύγκλισης των τροποιήμενων με τις κλασσικές μεθόδους

Σε αυτή την παράγραφο γίνεται μία πειραματική σύγκριση μεταξύ τροποποιήμενων και κλασσικών μεθόδων που έχουν υλοποιηθεί στην **1η άσκηση**. Για την σύγκριση αυτή χρησιμοποιούνται δεδομένα από 1000 κλήσεις της κάθε συνάρτησης όπου σε κάθε κλήση μετριέται ο χρόνος σε **ms** που χρειάστηκε για να τελειώσει ο αλγόριθμος.

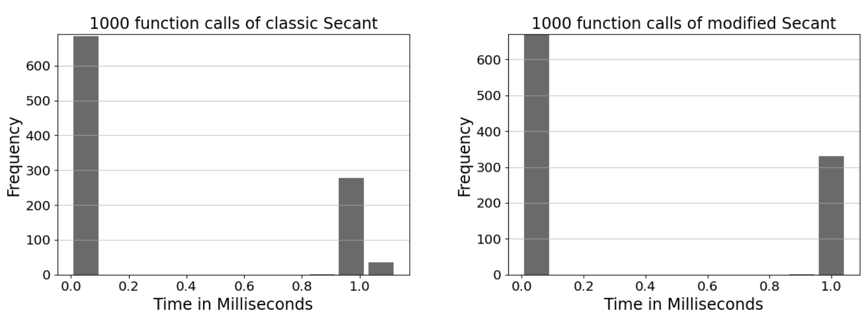



Από τα παραπάνω ιστογράμματα που αφορούν την μέθοδο της διχοτόμησης παρατηρούμε ότι περίπου 100 παραπάνω κλήσεις της τροποποιήμενης μεθόδου χρειάζονται 1 **ms** σε σχέση με την κλασσική μέθοδο της διχοτόμησης ενώ αντίστοιχα περίπου 100 παραπάνω κλήσεις της κλασσικής μεθόδου χρειάζονται μεταξύ 0 και 0.25 **ms** σε σχέση με την τροποποιήμενη. Επίσης, παρατηρούμε ότι υπάρχει ένα μικρό ποσοστό του δείγματος σε κάθε μορφή μεθόδου που χρειάστηκε μεταξύ 1.75 και 2 **ms** και αυτό εξάρτααι από το μήκος του διαστήματος με το οποίο κλήθηκε η συνάρτηση. Σε γενικές γραμμές η κλασσική μέθοδος πειραματικά είναι πιο γρήγορη κι αυτό είναι ως ένα βαθμό λογικό καθώς η τροποποιήμενη εξαρτάται από την ακολουθία των αριθμών που παράγει η συνάρτηση παραγωγής τυχαίων αριθμών.




Από τα παραπάνω ιστογράμματα που αφορούν την μέθοδο Newton-Raphson παρατηρούμε ότι περίπου 200 παραπάνω κλήσεις της κλασσικής μεθόδου χρειάστηκαν μεταξύ 0 και 0.25 **ms** σε σχέση με την τροποποιήμενη ενώ αντίστοιχα περίπου 200 παραπάνω κλήσεις της τροποποιήμενης μεθόδου χρειάζονται μεταξύ 1 και 1.5 **ms** σε σχέση με την κλασσική. Ακόμα, παρατηρούμε και πάλι ότι ένα μικρό ποσοστό του δείγματος σε κάθε μορφή μεθόδου χρειάστηκε πάνω από 2 **ms** για να ολοκληρωθεί όπου σε αυτή την περίπτωση ευθύνεται το αρχικό σημείο. Σε γενικές γραμμές η κλασσική μέθοδος πειραματικά είναι πιο γρήγορη, γεγονός και πάλι λογικό αφού στην τροποποιήμενη μέθοδο πραγματοποιούνται περισσότερες κλήσεις συναρτήσεων και υπολογίζονται δυνάμεις 2ου και 3ου βαθμού πραγματικών αριθμών.




Από τα παραπάνω ιστογράμματα που αφορούν την μέθοδο της τέμνουσας παρατηρούμε ότι οι δύο μέθοδοι αποδίδουν σχεδόν το ίδιο με την κλασσική μέθοδο να είναι λιγότερο αποδοτική σε περίπου 50 κλήσεις του δείγματος που χρειάστηκε πάνω από 1 **ms** σε σχέση με την τροποποιήμενη που ήταν πιο σταθερή στην απόδοση της, και πάλι η συμπεριφορά αυτή εξαρτάται από την επιλογή των αρχικών σημείων. Τέλος, **τονίζεται** ότι παράλληλα με την εκτέλεση των συναρτήσεων για κάθε μέθοδο το λειτουργικό σύστημα εκτελεί και άλλες διεργασίες που μπορεί να επηρρεάσουν την απόδοση του κώδικα, επομένως τα παραπάνω συμπεράσματα και στις τρεις μεθόδους είναι αντιπροσωπευτικά ως ένα βαθμό.

# 4 Άσκηση 3

Στην **3η άσκηση** υλοποιούνται μέθοδοι που αφορούν την επίλυση γραμμικών συστημάτων. Αρχικά, αναπτύσσεται η μέθοδος $\mathbf{PA} = \mathbf{LU}$, που χρησιμοποιείται για την επίλυση γραμμικών συστημάτων της μορφής Ax=b με τον A να είναι τετραγωνικό πίνακας διάστασης n. Στην συνέχεια αναπτύσσεται η μέθοδος της αποσύνθεσης **Cholesky** για συμμετρικούς και θετικά ορισμένους πίνακες. Τέλος, αναπτύσσεται η μέθοδος **Gauss-Seidel** που συνήθως χρησιμοποιείται για την επίλυση $n \times n$ γραμμικών συστήματων όπως παραπάνω, όταν ο πίνακας A είναι αραιός.

## 4.1 PA = LU αποσύνθεση

Η μέθοδος της $\mathbf{PA} = \mathbf{LU}$ αποσύνθεσης στηρίζεται στην μέθοδο του **Gauss**, την οποία βελτιώνει έτσι ώστε να αποφευχθούν σφάλματα αλλά και για να μπορούν να επιλυθούν πολλά συστήματα της μορφής $\mathbf{Ax} = \mathbf{b}$ με τον ίδιο πίνακα $\mathbf{A}$ των συντελεστών των αγνώστων και διαφορετικό πίνακα στήλη $\mathbf{b}$ των αγνώστων. Η συνάρτηση που έχει υλοποιηθεί στο αρχείο **lu_decomposition.py** υλοποιεί τον παρακάτω αλγόριθμο :

- Για κάθε γραμμή του πίνακα $A$ βρες το μεγαλύτερο στοιχείο που πρέπει να μπει στην κύρια διαγώνιο ως pivot.
- Αντιμετάθεσε αν απαιτείται την τωρινή γραμμή με την γραμμή που περιέχει το pivot στους πίνακες U,P και L.
- Κάνε απαλοιφή Gauss.
- Όταν οι γραμμές του πίνακα $A$ τελειώσουν λύσε το σύστημα $Lc = Pb$ ως προς $c$.
- Έπειτα λύσε το σύστημα $Ux = c$ ως προς $x$.

- Η λύση του συστήματος είναι το παραπάνω διάνυσμα $x$.

Πιο συγκεκριμένα, η συνάρτηση **lu** δέχεται ως παραμέτρους τον πίνακα A και το διάνυσμα b και επιστρέφει την λύση του συστήματος $x$. Αρχικά, η συνάρτηση πραγματοποιεί τις κατάλληλες αρχικοποιήσεις των πινάκων $P$,$L$ και $U$. Ο πίνακας $U$ στην αρχή είναι ίδιος με τον $A$ έτσι ώστε η συνάρτηση να μην πραγματοποιήσει καμία αλλαγή στον $A$. Στην συνέχεια, για κάθε γραμμή του πίνακα $U$ η συνάρτηση βρίσκει το μεγαλύτερο στοιχείο και το μεταφέρει στην κύρια διαγώνιο, οι αλλαγές γραμμών γίνονται παράλληλα στους πίνακες $P$,$L$ και $U$. Μετά την εύρεση του μεγαλύτερου στοιχείου η συνάρτηση εκτελεί την κλασσική απαλοιφή **Gauss** και συνέχιζει την παραπάνω διαδικάσια για όλες τις γραμμές του πίνακα $U$. Έπειτα, στην κύρια διαγώνιο του $L$ τοποθετούνται 1 και η συνάρτηση συνεχίζει με την επίλυση του συστήματος $Lc = Pb$ από πάνω προς τα κάτω ως προς $c$. Τέλος, η συνάρτηση επιλύει το σύστημα $Ux = c$ από κάτω προς τα πάνω ως προς $x$ όπου τελικά υπολογίζεται η λύση $x$ του συστήματος $Ax = b$ και επιστρέφεται στην καλούσα συνάρτηση. Τονίζεται ότι η διαδικάσια της $\mathbf{PA} = \mathbf{LU}$ αποσύνθεσης στα βήματα πριν την επίλυση του συστήματος $Lc = Pb$ δεν εξαρτάται από το διάνυσμα $\mathbf{b}$ και επομένως αν για κάποια προβλήματα ο πίνακας των συντελεστών $\mathbf{A}$ είναι ίδιος αλλά αλλάζει το διάνυσμα $\mathbf{b}$ μπορεί να πραγματοποιηθεί **μία** φορά η $\mathbf{PA} = \mathbf{LU}$ αποσύνθεση και στην συνέχεια να λυθούν τα συστήματα για τα αντίστοιχα $\mathbf{b}$. Έτσι, το κόστος είναι μία φορά της τάξης του $n^3$ για την διαδικάσια $\mathbf{PA} = \mathbf{LU}$ και στην συνέχεια $n^2$ για κάθε διαφορετικό $\mathbf{b}$. Στην συνέχεια, παρουσιάζεται ένα παράδειγμα επίλυσης ενός γραμμικού συστήματος με την χρήση της συνάρτησης

**lu** με το οποίο τελειώνει αυτή η παράγραφος. Το σύστημα είναι το παρακάτω

$$3x_1 + x_2 - 4x3 + x_4 = 1$$

$$-5x_1 + 2x_2 + x3 - 2x_4 = -3$$

$$-x_1 + 6x_2 - 3x3 - 4x_4 = 2$$

$$-2x_1 + x_2 - 4x3 + 2x_4 = 0$$

Για την λύση του παραπάνω συστήματος αρχικά δημιουργούμε τον πίνακα $A$ και το διάνυσμα $b$. Στην συνέχεια καλούμε την συνάρτηση **lu** με ορίσματα τον πίνακα $A$ και το διάνυσμα $b$ από την οποία παίρνουμε τα παρακάτω αποτελέσματα

```
a = [[3, 1, -4, 1], [-5, 2, 1, -2], [-1, 6, -3, -4], [-2, 1, -4, 2]]
b = [1, -3, 2, 0]
```

Σχήμα 45: Δημιουργία πινάκων για το παραπάνω σύστημα

```
p :
 0.0    1  0.0  0.0
 0.0  0.0    1  0.0
 0.0  0.0  0.0    1
   1  0.0  0.0  0.0

a :
   3    1   -4    1
  -5    2    1   -2
  -1    6   -3   -4
  -2    1   -4    2

l :
   1  0.0  0.0  0.0
 0.2    1  0.0  0.0
 0.4 0.035714285714285705    1  0.0
-0.6 0.3928571428571429 0.4999999999999998    1

u :
  -5    2    1   -2
 0.0  5.6 -3.2 -3.6
 0.0 2.7755575615628914e-17 -4.2857142857142865 2.9285714285714284
 0.0  0.0  0.0 -0.2499999999999989


x = [2.1000000000000063, 10.133333333333377, 6.233333333333336, 9.500000000000004]
```

Σχήμα 46: **PA** $=$ **LU** αποσύνθεση για τον πίνακα $A$ του συστήματος και η λύση του συστήματος

## 4.2 Αποσύνθεση Cholesky

Η μέθοδος της αποσύνθεσης **Cholesky** αφορά συμμετρικούς και θετικά ορισμένους $n \times n$ τετραγωνικούς πίνακες τους οποίους αποσυνθέτει στο παρακάτω γινόμενο

$$R^T R,$$

με τον $R$ να είναι άνω τριγωνικός. Η συνάρτηση που έχει υλοποιηθεί στο αρχείο **cholesky_decomposition.py** υλοποιεί τον παρακάτω αλγόριθμο :

- Για κάθε γραμμή $i$ του πίνακα $A$ αν το στοιχείο $A_{ii}$ είναι αρνητικό ο αλγόριθμος σταματάει.
- Ανάθεσε στο στοιχείο $R_{ii}$ την τετραγωνική ρίζα του $A_{ii}$.
- Βρες τον διάνυσμα $u^T$ σύμφωνα με το $u^T = \frac{1}{R_{ii}} * A_{i,i+1:n}$
- Τοποθέτησε το διάνυσμα $u^T$ στα στοιχεία δεξιά της κύριας διαγωνίου στην γραμμή $i$ σύμφωνα με το $R_{i,i+1:n} = u^T$
- Αφαίρεσε από τον διάστασης $i+1$ υποπίνακα του $A$ τον πίνακα $uu^T$ σύμφωνα με το $A_{i+1:n,i+1:n} = A_{i+1:n,i+1:n} - uu^T$

Πιο συγκεκριμένα, η συνάρτηση **chol** δέχεται ως παράμετρο τον πίνακα A και επιστρέφει τον κάτω τριγωνικό πίνακα $L$ ( ή τον $R^T$ ) που αποτελεί την αποσύνθεση **Cholesky** του πίνακα A. Αρχικά, η συνάρτηση πραγματοποιεί τις κατάλληλες αρχικοποιήσεις για τον πίνακα $R$ και δημιουργεί ένα αντίγραφο του πίνακα $A$ έτσι ώστε να μην πραγματοποιήσει καμία αλλαγή στον πίνακα που πήρε σαν όρισμα. Στην συνέχεια, για κάθε γραμμή $i$ του αντίγραφου του πίνακα $A$ ελέγχει αν $A_{ii} < 0$ και αν ναι, η συνάρτηση σταματάει και επιστρέφει τον ανάστροφο πίνακα του $R$. Αν όχι η συνάρτηση συνεχίζει και αναθέτει στο στοιχείο $R_{ii}$ το $\sqrt{A_{ii}}$. Στην συνέχεια, υπολογίζεται το διάνυσμα $u^T$ και αντιγράφονται τα στοιχεία του στα δεξιά του στοιχείου $R_{ii}$. Τέλος, υπολογίζεται ο πίνακας $uu^T$ και αφαιρείται από τον

$i+1$ διάστασης υποπίνακα του $A$. Στην συνέχεια, παρουσιάζεται ένα παράδειγμα εύρεσης της αποσύνθεσης **Cholesky** με την χρήση της συνάρτησης **chol** με το οποίο τελειώνει αυτή η παράγραφος. Στο παράδειγμα μας ο πίνακας $A$ είναι ο

$$A = \begin{pmatrix} 9 & 0 & -27 & 18 \\ 0 & 9 & -9 & -27 \\ -27 & -9 & 99 & -27 \\ 18 & -27 & -27 & 121 \end{pmatrix}$$

Για την εύρεση της αποσύνθεσης **Cholesky** του παραπάνω πίνακα αρχικά δημιουργούμε τον πίνακα $A$ με παρόμοιο τρόπο όπως στην παράγραφο **4.1** και καλούμε την συνάρτηση **chol** με όρισμα τον πίνακα $A$ από την οποία παίρνουμε τα εξής αποτελέσματα:

```
l :
 3.0  0.0  0.0  0.0
 0.0  3.0  0.0  0.0
-9.0 -3.0  3.0  0.0
 6.0 -9.0  0.0  2.0
```

Σχήμα 47: Αποσύνθεση **Cholesky** για τον πίνακα $A$

## 4.3 Μέθοδος Gauss-Seidel

Η μέθοδος **Gauss-Seidel** αποτελεί μία επαναληπτική μέθοδος επίλυσης γραμμικών συστημάτων της μορφής $\mathbf{Ax} = \mathbf{b}$, όπου $A$ τετραγωνικός πίνακας $n \times n$. Χρησιμοποιείται συνήθως σε αραιά συστήματα όπου και έχει καλύτερη απόδοση από την μέθοδο της $\mathbf{PA} = \mathbf{LU}$ αποσύνθεσης. Η συνάρτηση που έχει υλοποιηθεί στο αρχείο **gauss_seidel.py** υλοποιεί τον παρακάτω αλγόριθμο :

- Για κάθε γραμμή $i$ του πίνακα $A$ υπολόγισε το άθροισμα των τιμών των μεταβλητών που έχουν ανανεωθεί σε αυτό το βήμα, για τις μεταβλητές που δεν έχουν ανανεωθεί ακόμα χρησιμοποιήσε τις τιμές τους από την προηγούμενο βήμα.

- Ανάθεσε στο στοιχείο $x_i$ την τιμή

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

όπου $\sum_{j=1}^{i-1} a_{ij}x_j$ το άθροισμα των τιμών των μεταβλητών που έχουν ανανεωθεί σε αυτό το βήμα και $\sum_{j=i+1}^{n} a_{ij}x_j$ αυτών που δεν έχουν ανανεωθεί.

- Συνέχισε την παραπάνω διαδικασία μέχρι να υπολογιστεί η λύση με την επιθυμητή ακρίβεια.

Πιο συγκεκριμένα, η συνάρτηση **gauss_seidel** δέχεται ως παραμέτρους τον πίνακα A, το διάνυσμα b και την επιθυμητή ακρίβεια και επιστρέφει την λύση του συστήματος $x$. Το **default** όρισμα **eps** έχει την τιμή που χρειάζεται για ακρίβεια **4** δεκαδικών ψηφίων. Αρχικά, η συνάρτηση πραγματοποιεί τις κατάλληλες αρχικοποιήσεις των διανυσμάτων $x$ και previous_x για να μπορεί να υπολογισθεί η ακρίβεια της λύσης καθώς και δημιουργεί και ένα αντίγραφο του πίνακα $A$ έτσι ώστε να μην πραγματοποιήσει καμία αλλαγή στον πίνακα που πήρε σαν όρισμα. Στην συνέχεια, για κάθε γραμμή $i$ του πίνακα $A$ υπολογίζει τον αντίστροφο του συντελεστή της μεταβλητής $x_i$ και το άθροισμα των τιμών των μεταβλητών. Για τις μεταβλητές που έχουν ανανεωθεί σε αυτό το βήμα χρησιμοποιεί τις νέες τιμές τους, ενώ γι' αυτές που δεν έχουν ανανεωθεί τις τιμές τους από το προηγούμενο βήμα. Έπειτα, αλλάζει τις τιμές του διανύσματος previous_x και την τιμή του διανύσματος $x$ σύμφωνα με την τιμή που αναφέρθηκε παραπάνω και ελέγχει αν η εκτίμηση της λύσης έχει βρεθεί με την επιθυμητή ακρίβεια **eps**. Η ακρίβεια υπολογίζεται ως προς την άπειρη νόρμα του σφάλματος στην λύση. Τέλος, αναφέρεται ότι αν ο $A$ έχει κυριαρχική διαγώνιο η μέθοδος συγκλίνει με οποιοδήποτε αρχικό διάνυσμα $x_0$, η συνάρτηση ξεκινάει από το μηδενικό διάνυσμα μήκους $n$.

Στην συνέχεια παρουσιάζεται η λύση του συστήματος που ζητείται για $n = 10$ και $n = 10000$ με το οποίο τελειώνει αυτή η παράγραφος. Το σύστημα είναι το παρακάτω

$$\begin{pmatrix} 5 & -2 & & & \\ -2 & 5 & -2 & & \\ & \ddots & \ddots & \ddots & \\ & & -2 & 5 & -2 \\ & & & -2 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ \vdots \\ 1 \\ 3 \end{pmatrix}$$

Για την λύση του παραπάνω συστήματος αρχικά δημιουργούμε τον πίνακα $A$ με παρόμοιο τρόπο όπως στην παράγραφο **4.1** και καλούμε την συνάρτηση **gauss_seidel** με όρισμα τον πίνακα $A$ και το διάνυσμα $b$ από την οποία παίρνουμε τα εξής αποτελέσματα ( με ακρίβεια 4 δεκαδικών ψηφίων ):

```
x = [0.99996, 0.99994, 0.99993, 0.99994, 0.99995, 0.99996, 0.99997, 0.99998, 0.99999, 1.00000]
iterations = 20
```

Σχήμα 48: Λύση του συστήματος για $n = 10$

Όπου παρατηρούμε ότι τα $x_i$ με ακρίβεια 4 δεκαδικών είναι $\mathbf{0.9999}$ και ότι χρειάστηκαν 20 επαναλήψεις για να συγκλίνει ο αλγόριθμος με την επιθυμητή ακρίβεια. Για $n = 10000$ τα $x_i$ είναι και πάλι $\mathbf{0.9999}$ και χρειάστηκαν 23 επαναλήψεις για να συγκλίνει ο αλγόριθμος με την επιθυμητή ακρίβεια ( Η λύση δεν εμφανίζεται για ευνόητους λόγους ).