Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

ΑΡΙΘΜΗΤΙΚΗ ΑΝΑΛΥΣΗ

Προγραμματιστική Εργασία

Κουκουλέτσου Αικατερίνη 10218

Μάιος 2024

Επέκταση της μεθόδου Newton για την επίλυση εξισώσεων μιας μεταβλητής

Η παρακάτω εργασία αποτελεί μια προσπάθεια επέκτασης της μεθόδου Newton για την εύρεση ριζών. Στηριζόμενοι στο γεγονός ότι για μια τιμή x_0 η συνάρτηση f(x) προσεγγίζεται επαρκώς καλά από τη σειρά Taylor σύμφωνα με την σχέση:

$$f(x) \approx y = f(x_0) + \sum_{n=1}^{N} \frac{f^n(x_0)}{n!}(x - x_0)^n$$

Για N = 2 επιλύεται ως προς x η παρακάτω εξίσωση

$$0 = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2$$

και έπειτα από πράξεις προκύπτει

$$x^2 \frac{f''(x_0)}{2} + x(f'(x_0) - x_0 f''(x_0)) + f(x_0) - x_0 f'(x_0) + \frac{x_0^2 f''(x_0)}{2} = 0$$

Επομένως προκύπτει μια κλασική δευτεροβάθμια εξίσωση ως προς x με τα α, β και γ να είναι αντίστοιχα τα τμήματα αυτά

$$x^2 \underbrace{\frac{f''(x_0)}{2}}_{\alpha} + x \underbrace{(f'(x_0) - x_0 f''(x_0))}_{\beta} + \underbrace{f(x_0) - x_0 f'(x_0) + \frac{x_0^2 f''(x_0)}{2}}_{\gamma} = 0$$

Η διακρίνουσα ισούται με

$$\Delta = \beta^2 - 4\alpha\gamma$$

όπου μετά από αντικατάσταση προκύπτει

$$\Delta = (f'(x_0) - x_0 f''(x_0))^2 - \frac{4f''(x_0)}{2}(\frac{f''(x_0)}{2}x_0^2 - f'(x_0)x_0 + f(x_0))$$

μετά από απλοποίηση η διακρίνουσα φαίνεται να είναι τελικά ίση με

$$\Delta = f'(x_0)^2 - 2f''(x_0)f(x_0)$$

από αυτό το σημείο και μετά, διακρίνονται τρεις περιπτώσεις

1. $\Delta ιακρινουσα < 0$, η εξίσωση είναι αδύνατη
2. $\Delta ιακρινουσα = 0$, υπάρχει μία ρίζα διπλή
3. $\Delta ιακρινουσα > 0$, υπάρχουν δύο ρίζες

Η λύση της παραπάνω εξίσωσης δίνεται από τον τύπο

$$x = \frac{-\beta \pm \sqrt{\Delta}}{2\alpha}$$

όπου με αντικατάσταση προκύπτει τελικά η αναδρομική σχέση

$$x_{k+1} = \frac{-(f'(x_k) - x_k f''(x_k)) \pm \sqrt{f'(x_k)^2 - 2f''(x_k)f(x_k)}}{f''(x_k)}$$

Ως συνθήκη τερματισμού επιλέγεται και πάλι

$$|x_{k+1} - x_k| = \varepsilon$$

όπου το ε ορίστηκε να είναι ίσο με 0.0001

Παρατηρήσεις για τον κώδικα:

Όπως φαίνεται και από την θεωρητική ανάλυση, προκύπτουν δύο αναδρομικοί τύποι, ένας με συν και ένας με πλην την ρίζα της διακρίνουσας.

$$x_{k+1} = \frac{-(f'(x_k) - x_k f''(x_k)) - \sqrt{\Delta}}{f''(x_k)}$$

$$x_{k+1} = \frac{-(f'(x_k) - x_k f''(x_k)) + \sqrt{\Delta}}{f''(x_k)}$$

Στον κώδικα, η συνάρτηση με τίτλο $NewtonMethod/version1$ λύνει τον πρώτο αναδρομικό τύπο ενώ η συνάρτηση με τίτλο $NewtonMethod/version2$ τον δεύτερο. Παρατηρήθηκε ότι, σε δευτεροβάθμιες εξισώσεις, όπου $\alpha \neq 0$ και άρα γίνεται λόγος για δύο διαφορετικές λύσεις, ο πρώτος αναδρομικός τύπος προσεγγίζει τη μία και ο δεύτερος την άλλη. Σε συναρτήσεις που έχουν μια μόνο λύση, ο ένας από τους δύο τύπους συγκλίνει και εντοπίζει την λύση ενώ ο άλλος αποκλίνει. Όλα αυτά υπό την προυπόθεση ότι έχει γίνει μια καλή επιλογή στο initial guess.

Διαγράμματα

Συνάρτηση f(x)	Πραγματική Λύση	Εκτιμώμενη Λύση	Initial Guess
$x^2 - 4$	± 2	± 2	0

Για όλες τις δευτεροβάθμιες με διακρίνουσα $\Delta \geq 0$, είναι εγγυημένη η σύγκλιση **στην πρώτη επανάληψη** ανεξαρτήτως αρχικής συνθήκης. Αυτό είναι αναμενόμενο καθώς

$$f(x) = \alpha x^2 + \beta x + \gamma$$

$$f'(x) = 2\alpha x + \beta$$

$$f''(x) = 2\alpha$$

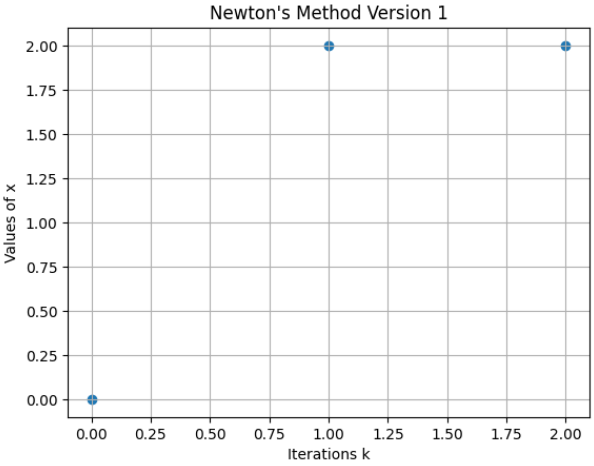
Με αντικατάσταση στην αναδρομική σχέση προκύπτει

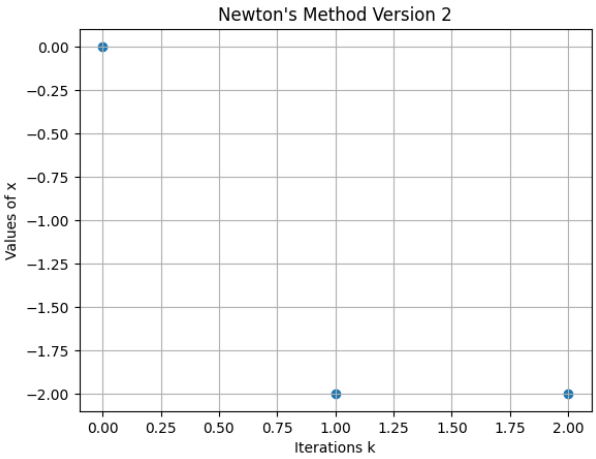
$$x_{k+1} = \frac{-(f'(x_k) - x_k f''(x_k)) + \sqrt{\Delta}}{f''(x_k)}$$

$$= \frac{-(2\alpha x + \beta - x2\alpha) \pm \sqrt{(2\alpha x + \beta)^2 - 2 \cdot 2\alpha(\alpha x^2 + \beta x + \gamma))}}{2} = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2}$$

Επομένως η επόμενη τιμή της αναδρομικής σχέσης κ + 1 είναι εντελώς ανεξάρτητη από την προηγούμενη τιμή κ. Ο αναδρομικός τύπος παίρνει την μορφή της λύσης της δευτεροβάθμιας, καθόλου τυχαίο αφού η μέθοδος Taylor δευτέρου βαθμού είναι μια ακριβής αναπαράσταση τετραγωνικής εξίσωσης.

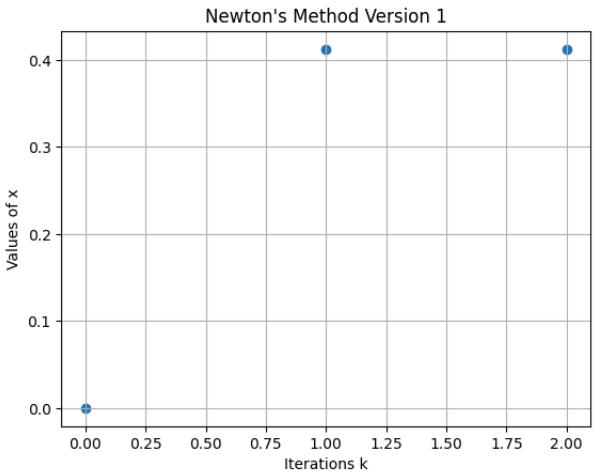
Καθώς εντοπίζεται η ακριβής λύση, $abs(error)$ είναι ίσο με το 0. Επομένως, το $log(abs(error))$ δεν ορίζεται. Για αυτό και το ζητούμενα διαγράμματα log - log απουσιάζουν.

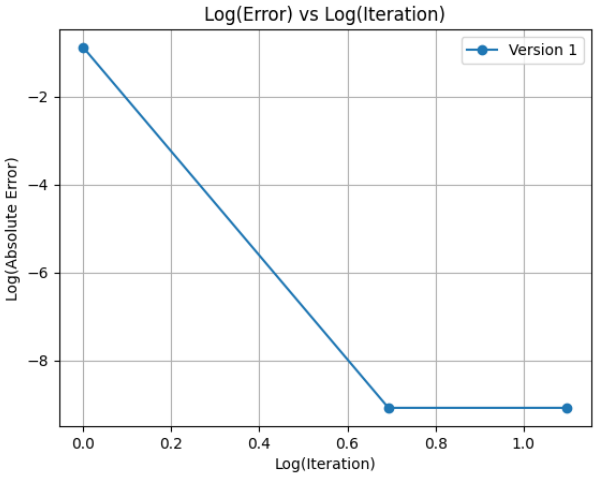


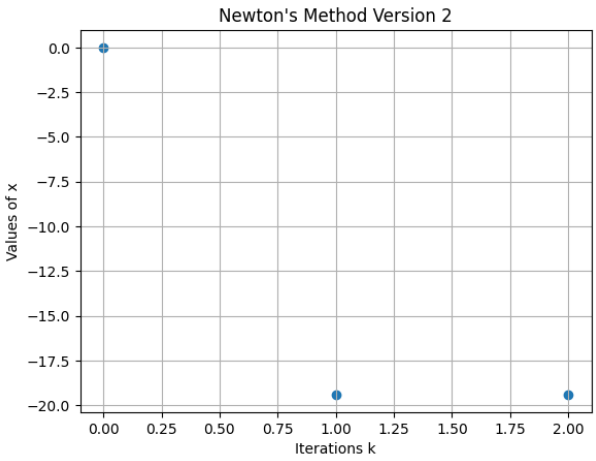


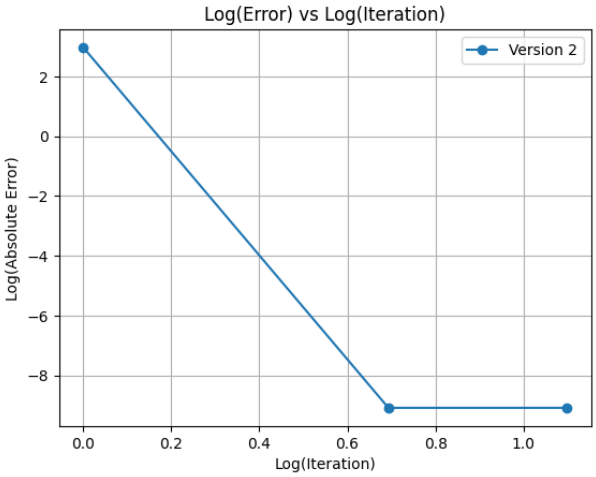
Συνάρτηση f(x)	Πραγματική Λύση	Εκτιμώμενη Λύση	Initial Guess
$x^2 + 19x - 8$	$-19.412, 0.412$	$-19.412, 0.412$	0

Δευτεροβάθμια με δύο διαφορετικές ρίζες, όπως αποδείχθηκε προηγουμένως θα συγκλίνει μετά την πρώτη επανάληψη.

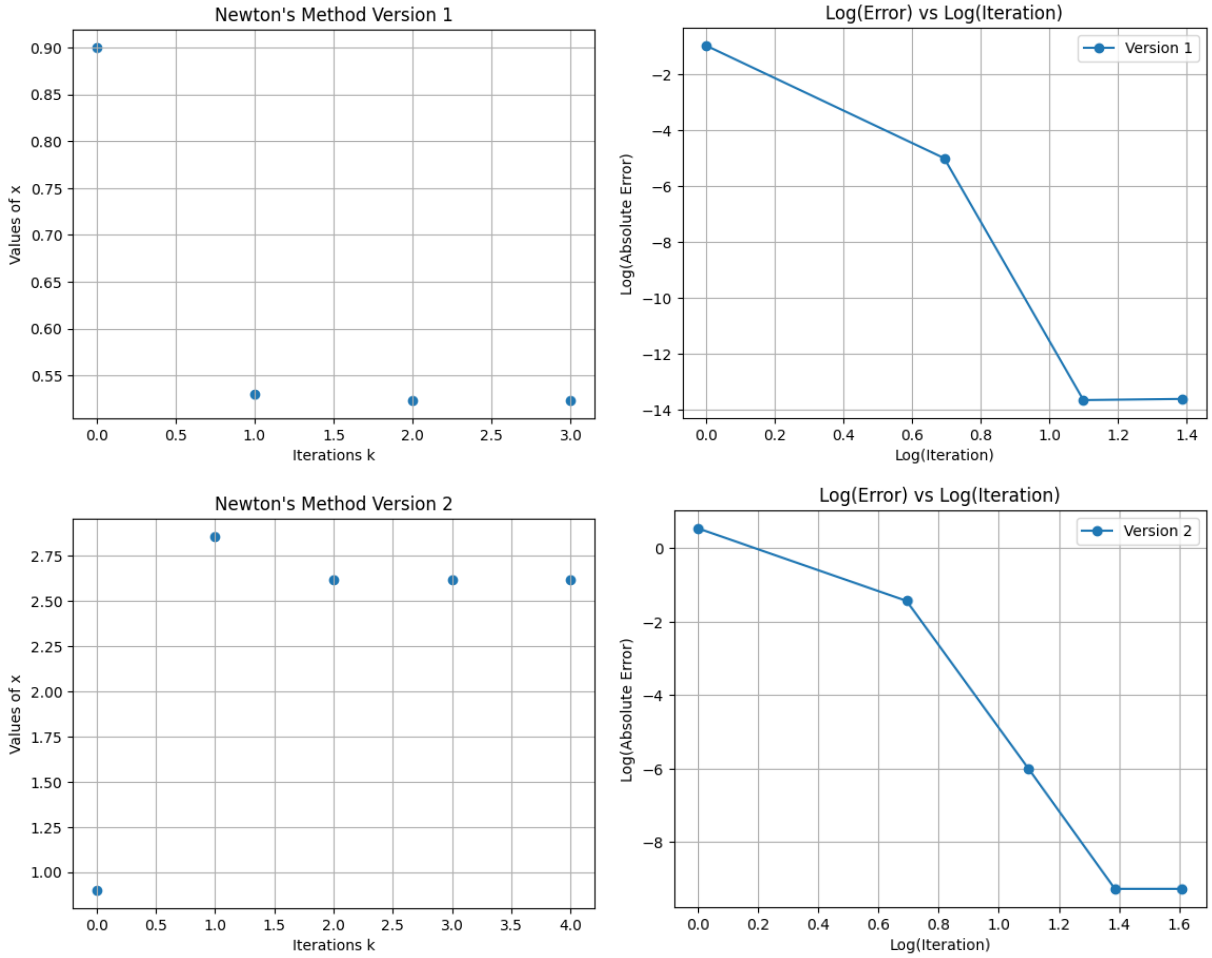






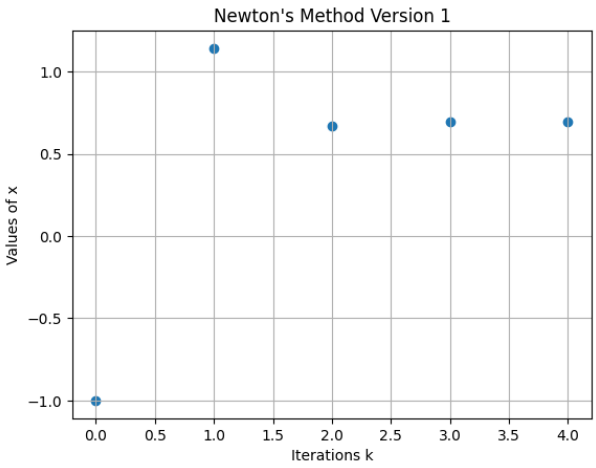


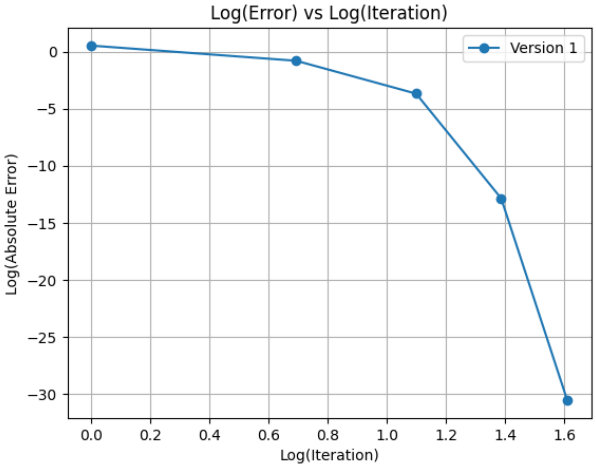
Συνάρτηση f(x)	Πραγματική Λύση	Εκτιμώμενη Λύση	Initial Guess
$sin(x) - 0.5$	$0.5236, 2.6179$	$0.5236, 2.6179$	0.9, 0.9



Σημειώνεται πως για την συγκεκριμένη συνάρτηση, κακή επιλογή στο initial guess προκάλεσε τερματισμό της διαδικασίας καθώς έβγαινε ο παρονομαστής της αναδρομικής σχέσης $f''(x)$ ίσος με το 0. Καθώς η δεύτερη παράγωγος $f''(x) = cos(x)$ υπάρχουν άπειρες τιμές που την μηδενίζουν και άρα χρειάζεται προσεκτική επιλογή στην αρχική εκτίμηση. Το ίδιο ισχύει και για συναρτήσεις όπως η $f(x) = cos(x) - 0.6$. Υπάρχει πιθανότητα να συγκλίνει αλλά θέλει προσοχή στο initial guess για να μην τερματιστει η διαδικασία κατευθείαν.

Συνάρτηση f(x)	Πραγματική Λύση	Εκτιμώμενη Λύση	Initial Guess
$e^x - 2$	0.69314718056	0.69314718056	-1





Καθώς η εξίσωση έχει μόνο μια ρίζα, ο ένας τύπος συγκλίνει ενώ ο δεύτερος αποκλίνει. Για τον δεύτερο τύπο, διαδικασία τερματίζεται αφού έχει ξεπεραστεί το όριο των μέγιστων επιτρεπόμενων επαναλήψεων.

Συνάρτηση f(x)	Πραγματική Λύση	Εκτιμώμενη Λύση	Initial Guess
$x^5 - 4x^3 - 3x - 1$	$-2.134, -0.299, 2.175, 2.6772$	-	0

Δοκιμάστηκαν πολλές διαφορετικές αρχικές εκτιμήσεις αλλά σε όλες η τιμή της ορίζουσας έβγαινε αρνητική στην πρώτη επανάληψη. Γενικά ο περιορισμός αυτός που θέτει η μέθοδος, για $\Delta \geq 0$ σε όλες τις επαναλήψεις είναι αρκετά αυστηρός. Σε περιπτώσεις όπως αυτην, που η μέθοδος εφαρμόζεται σε συναρτήσεις τάξεως $n \geq 3$, η διακρίνουσα και άρα η υπόρριζη ποσότητα θα είναι της τάξεως

$$\underbrace{f'(x)^2}_{2(n-1)} - 2\underbrace{f''(x)}_{n-2}\underbrace{f(x)}_{n}$$

$$\underbrace{f'(x)^2}_{2(n-1)} - 2\underbrace{f''(x)f(x)}_{2n-2}$$

Και άρα συνολικά πρόκειται για πολυώνυμο τάξης 2n-2 με $n \geq 3$ το οποίο πρέπει να είναι πάντα θετικό. Αυτό είναι μια πολύ αυστηρή προδιαγραφή και κάνει τον αλγόριθμο μη λειτουργικό για εύρεση ριζών σε πολυώνυμα τάξεως μεγαλύτερης του 2.

Συνάρτηση f(x)	Πραγματική Λύση	Εκτιμώμενη Λύση	Initial Guess
$x - 1$	1	-	0

Καθώς ο παρανομαστής της αναδρομικής σχέσης είναι μηδενικός για κάθε πρωτοβάθμια συνάρτηση, δεν μπορεί να χρησιμοποιηθεί ο συγκεκριμένος αλγόριθμος για εύρεση ρίζας σε καμία εξίσωση πρώτου βαθμού. Το οποίο είναι και το λογικό. Αφού ο τύπος έχει προέλθει από προσπάθεια εκτίμησης μιας συνάρτησης, δεν μπορει να χρησιμοποιηθεί δευτεροβάθμια συνάρτηση για την προσέγγιση πρωτοβάθμιας. Οι γραμμικές συναρτήσεις είναι πολυώνυμα πρώτου βαθμού χωρίς καμπυλότητα, καθιστώντας την προσέγγιση από συναρτήσεις ανώτερου βαθμού ακατάλληλες.

Σημειώνεται όμως και ότι, οι γραμμικές εξισώσεις μπορούν να λυθούν άμεσα χωρίς επαναλήψεις, καθώς έχουν μια απλή αλγεβρική λύση.

Συμπεράσματα

Πλεονεκτήματα και μειονεκτήματα που εντοπίστηκαν στην τροποποιημένη μέθοδο

1. Εύρεση Δύο Ριζών: σε δευτεροβάθμιες εξισώσεις ($\alpha \neq 0$) που έχουν δύο ρίζες ή μία διπλή, ο κάθε ένας από τους δύο αναδρομικούς τύπους θα βρει από μια λύση στην πρώτη επανάληψη ανεξαρτήτως αρχικής υπόθεσης.

2. Ταχύτερη Σύγκλιση: με την τροποποίηση αυτή μπορεί η μέθοδος τελικά να συγκλίνει πιο γρήγορα, αφού πλέον λαμβάνεται υπόψιν όχι μόνο η κλίση αλλά και η καμπυλότητα της συνάρτησης.

3. Υψηλότερη Ακρίβεια ανά Επανάληψη: σε συνέχεια του προηγουμένου, σε κάθε επανάληψη μπορεί να παρέχεται μια πιο ακριβής προσέγγιση της ρίζας, καθώς ο τετραγωνικός όρος βοηθά να προσεγγιστεί καλύτερα η συνάρτηση κοντά στη ρίζα.

4. Αυξημένη Υπολογιστική Πολυπλοκότητα: Η ανάγκη για τον υπολογισμό της δεύτερης παραγώγου προσθέτει ένα παραπάνω υπολογιστικό κόστος. Όπως και η ανάγκη για την λύση της τετραγωνικής εξίσωσης σε κάθε επανάληψη.

5. Μικρότερη Πιθανότητα Σύγκλισης: Είναι δυσκολότερο η μέθοδος να συγκλίνει. Σε κάθε επανάληψη υπάρχει πιθανότητα η διακρίνουσα να βγει αρνητική και άρα να τερματιστεί η διαδικασία. Αυτός ο περιορισμός δεν υπάρχει στην μέθοδο Newton που βασίζεται σε πολυώνυμο 1ης τάξης.

6. Απαίτηση για μια Ακριβέστερη Αρχική Εκτίμηση: σε συνέχεια του προηγουμένου, είναι απαιτητικότερη η ανάγκη για μια πιο εντοπισμένη αρχική εκτίμηση. Μια μη βέλτιστη αρχική εκτίμηση μπορεί να οδηγήσει όχι μόνο σε αργή σύγκλιση ή απόκλιση αλλά αυτήν την φορά και σε αρνητική διακρίνουσα ή μηδενική τιμή δεύτερης παραγώγου και άρα τερματισμό της διαδικασίας.

Συμπερασματικά, καθώς η τροποποιημένη αυτή μέθοδος λειτουργεί σωστά υπό αρκετά αυστηρότερες προυποθέσεις, ενώ η μέθοδος Newton έχει αρκετά πιο ευρεία εφαρμογή.

Μέθοδος Halley

Έπειτα από έρευνα βρέθηκε ότι μια τέτοια μέθοδος, κατά την οποία χρησιμοποιείται το πολυώνυμο Taylor 2ης τάξης για τον εντοπισμό ριζών, υπάρχει ήδη και λέγεται Halley's Method. Στην μέθοδο αυτήν, αντί να λυθεί η αρχική σχέση ως προς το x_{n+1}, λύνεται ως προς το $x_{n+1} - x_n$ όπως φαίνεται παρακάτω.

2nd Order Taylor Polynomial

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

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

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

Στην συνέχεια, για να απομακρυνθεί ο όρος x_{n+1} από το δεξί μέρος της εξίσωσης, χρησιμοποιείται η σχέση της μέθοδο Newton

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

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

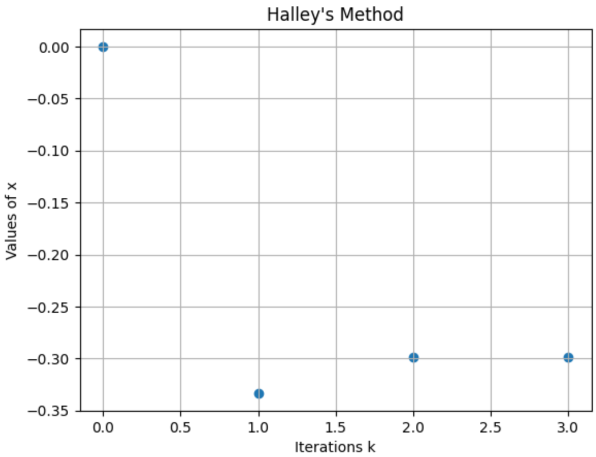
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n) - \frac{f''(x_n)f(x_n)}{2f'(x_n)}}\frac{2f'(x_n)}{2f'(x_n)}$$

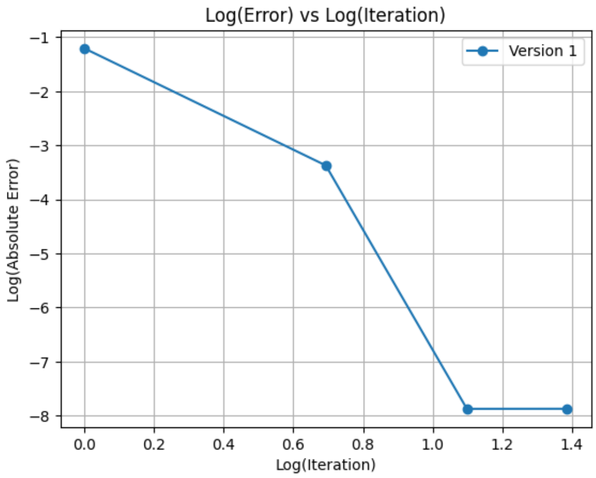
$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2f'(x_n)^2 - f''(x_n)f(x_n)}$$

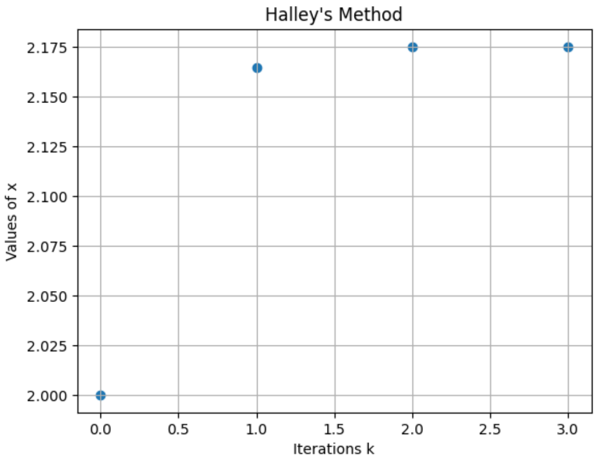
Κάνοντας τις πράξεις και λύνοντας ως προς x_{n+1} προκύπτει τελικά ο παραπάνω αναδρομικός τύπος. Με αυτήν την έξυπνη αντικατάσταση από την μέθοδο Taylor (1ης τάξης), παρακάμπτεται η ανάγκη για επίλυση δευτεροβάθμιας εξίσωσης για τον προσδιορισμό του x_{n+1} και κατ'επέκταση δεν υπάρχει πια η ανάγκη για χρήση δύο διαφορετικών τύπων ή η εξάρτηση από το πρόσημο της διακρίνουσας.

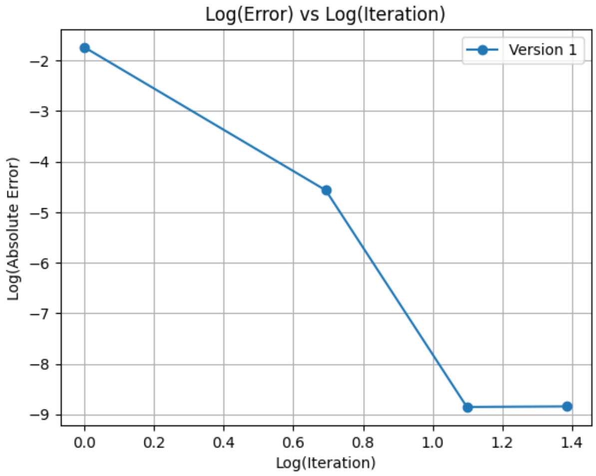
Διαγράμματα

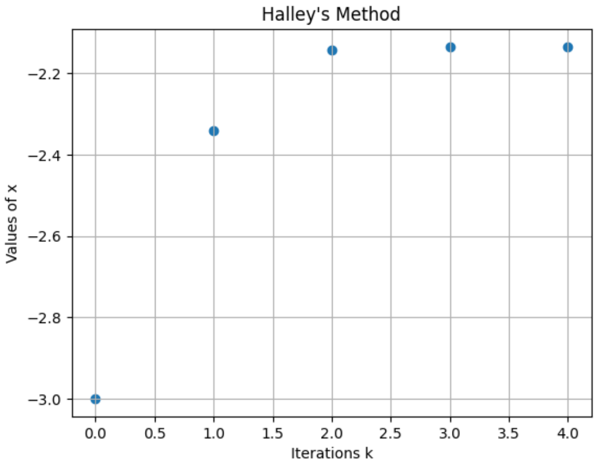
Συνάρτηση f(x)	Πραγματική Λύση	Εκτιμώμενη Λύση	Initial Guess
$x^5 - 4x^3 - 3x - 1$	$2.175, 0.299, -2.134$	2, 0, -3	2.1751 ,-0.298, -2.134

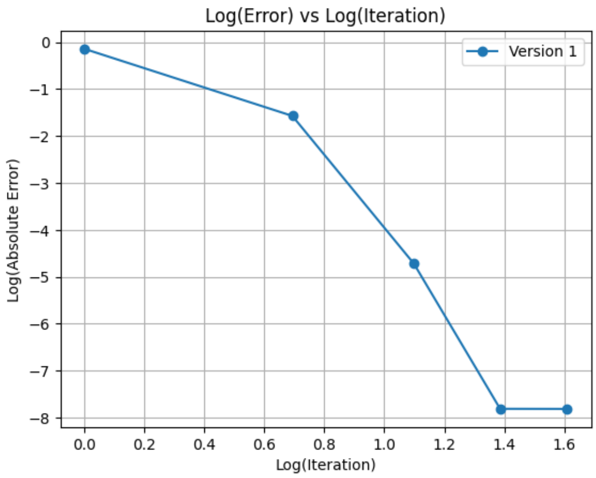




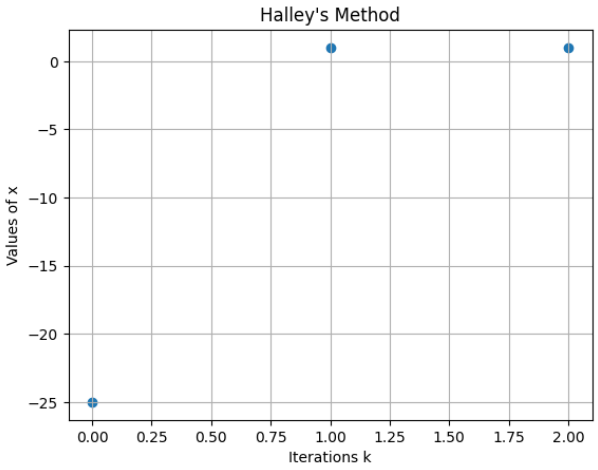








Συνάρτηση f(x)	Πραγματική Λύση	Εκτιμώμενη Λύση	Initial Guess
$x - 1$	1	1	-25



Φαίνεται ότι ακόμη και με μια κακή επιλογή αρχικού σημείου, ο αλγόριθμος Halley καταφέρνει να εντοπίσει την λύση στην πρωτοβάθμια εξίσωση, κάτι το οποίο η τροποποιημένη μέθοδος Newton δεν μπορούσε να κάνει. Καθώς γίνεται ακριβής εκτίμηση της λύσης, $abs(error) = 0$ και άρα $log(abs(error))$ δεν ορίζεται για αυτο και το διάγραμμα πάλι παραλείπεται.

Παρατηρήσεις

Φαίνεται πως η Halley, παρόλο που και αυτή βασίστηκε σε πολυώνυμο Taylor 2ης τάξης προσεγγίζει τις λύσεις και σε πωτοβάθμιες εξισώσεις και σε εξισώσεις βαθμού μεγαλύτερου του 2. Έχει πολύ λιγότερους περιορισμούς καθώς δεν έχει υπόρριζη ποσότητα στην αναδρομική σχέση και βρίσκει πολύ μεγαλύτερη εφαρμογή. Η μόνο απαίτηση για την σωστή λειτουργία της είναι μια καλή επιλογή για την αρχική υπόθεση το οποίο όμως είναι απαίτηση γενικά σε όλες τις μεθόδους εύρεσης ρίζας.

Επίλυση Συστήματος Διαφορικών Εξισώσεων με μέθοδο Runge-Kutta 4ης τάξης

Ας υποθέσουμε ότι υπάρχουν δύο χαρακτήρες, η Ελένη και ο Πάρης, των οποίων το συναίσθημα ποσοτικοποιείται σε μια κλίμακα από -5 έως 5 ως εξής:

-5	Μίσος
-2,5	Αποστροφή
0	Αδιαφορία
2,5	Συμπάθεια
5	Απέραντη Αγάπη

Μαθηματικά, τα συναισθήματα τους περιγράφονται από το ακόλουθο σύστημα διαφορικών εξισώσεων

$$\begin{cases} \frac{dx}{dt} = -\alpha y \\ \frac{dy}{dt} = \beta x - \gamma y^2 \end{cases} .$$

όπου α = 0.2, β = 0.8 και γ = 0.1.

Ως x(t) εκφράζεται το συναίσθημα που νιώθει ο Πάρης για την Ελένη και αντίστοιχα ως y(t) το συναίσθημα που νιώθει η Ελένη για τον Πάρη την χρονική στιγμή t.

Ερώτημα Α

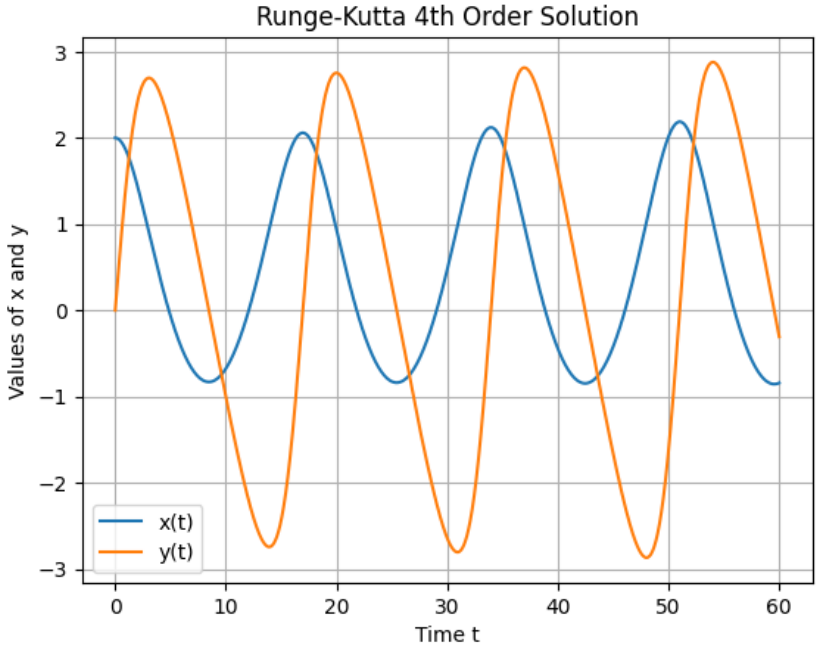
Με αρχικές τιμές x(0) = 2 και y(0) = 0, ζητείται η επίλυση του παραπάνω συστήματος διαφορικών εξισώσεων κάνοντας χρήση της κλάσσικής μεθόδου Runge-Kutta.

Μέθοδος Runge - Kutta 4ης τάξης (κλασσική)

$$\begin{aligned} k_0 &= hf(x_i, y_i) \\ k_1 &= hf(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_0) \\ k_2 &= hf(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1) \\ k_3 &= hf(x_i + h, y_i + k_2) \\ y_{i+1} &= y_i + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3) \end{aligned}$$

με h να είναι το βήμα το οποίο επιλέχθηκε ίσο με 0.1

Το διάγραμμα των x(t) και y(t) για το χρονικό διάστημα από 0 έως 60s φαίνεται στην παρακάτω εικόνα.



Ο Πάρης ξεκινάει από την τιμή 2 όμως τα συναισθήματά του δεν αυξάνονται με την πάροδο του χρόνου αλλά ταλαντώνονται μεταξύ δύο τιμών. Αυτό συμβαίνει γιατί καθώς περνάει ο χρόνος και τα συναισθήματα της Ελένης έχουν θετικό πρόσημο, από την διαφορική εξίσωση $\frac{dx}{dt} = -\alpha y$, η παράγωγος των συναισθημάτων του Πάρη αποκτά αρνητική τιμή και άρα η x(t) θα έχει φθίνουσα συμπεριφορά. Αυτό συνεχίζεται έως ότου τα συναισθήματα της Ελένης να γίνουν αρνητικά με αποτέλεσμα η παράγωγος να γίνει πλεόν θετική και άρα το x(t) να έχει αύξουσα συμπεριφορά. Αυτή η κατάσταση επαναλαμβάνεται.

Οι τιμές του πλάτους της ταλάντωσης για τον Πάρη είναι το 2.188 και το -0.856, το οποίο σημαίνει πως η συμπεριφορά του κυμαίνεται στα πλαίσια της αδιαφορίας με εξαίρεση μία φορά σε κάθε περίοδο T όπου και στιγμιαία πλησιάζει ίσως τα όρια της συμπάθειας, με το x(t) τότε να ισούται με το 2.18.

Όσο για την Ελένη, παρόλο που αρχικά το y(0) είναι ίσο με 0 και άρα ξεκινάει από την κατάσταση της αδιαφορίας, τα συναισθήματά της φαίνεται να αυξάνονται με την πάροδο του χρόνου εξαιτίας της παρουσίας του όρου βx στην διαφορική του y.
Λόγω αυξομειώσεων όμως στα συναισθήματα του Πάρη, προκαλούνται αυξομειώσεις και στα δικά της συναισθήματα και άρα παρουσιάζει και αυτή μια ταλαντωτική συμπεριφορά. Τα δικά της όρια ταλάντωσης είναι κατά πολύ μεγαλύτερα αφού έχει μέγιστη τιμή 2.8787 και ελάχιστη τιμή -2.869. Για την ελένη ισχύει ότι σε κάθε περίοδο T ξεκινάει από το 0 και φτάνει στην κατάσταση της συμπάθειας αλλά εξαιτίας του πάρη και κατα συνέπεια εξαιτίας της αυπνίας της (όρος $-\gamma y^2$) φτάνει στην αποστροφή. Το μοτίβο αυτό επίσης επαναλαμβάνεται.

Συμπερασματικά, και οι δύο φαίνεται να υπόκεινται σε μια ταλαντωτική συμπεριφορά. Καθώς οι διαφορικές των συναισθημάτων τους δεν είναι αποσυζευγμένες, επηρεάζει ο ένας τον άλλον (λόγω της εμπλοκής του όρου y στην διαφορική του x και της εμπλοκής του όρου x στην διαφορική του y), με αποτέλεσμα να έχουν εγκλωβιστεί σε μια ταλάντωση που φαίνεται

να είναι μη αποσβενύμενη.

Ερώτημα Β

Στο δεύτερο ερώτημα της εργασίας ζητείται να γραφτεί κώδικας που να προσεγγίζει την τιμή της σταθεράς γ, έτσι ώστε η λύση της διαφορικής την χρονική στιγμή t = 30s να είναι ίση με 0, δηλαδή y(30) = 0.

Το ζητούμενο αντιμετωπίστηκε ως ένα πρόβλημα έυρεσης ρίζας. Για τον εντοπισμό της ρίζας χρησιμοποιήθηκε η μέθοδος της διχοτόμου.

Μέθοδος Διχοτόμου

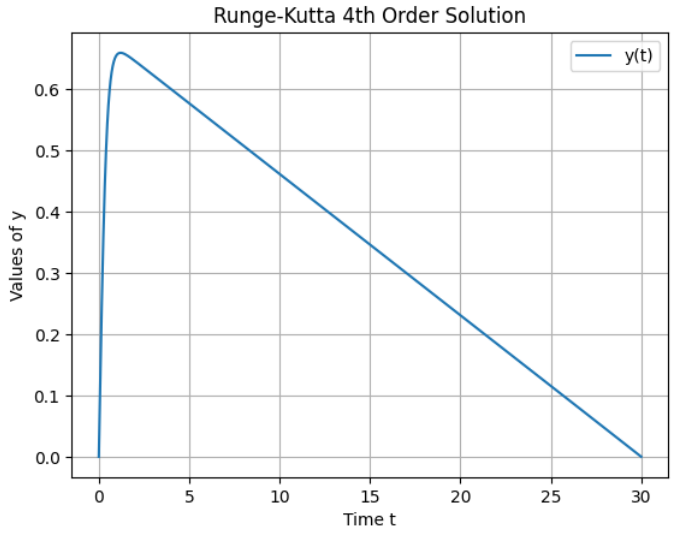
1. Επιλογή αρχικού διαστήματος: επιλέγονται τα άκρα του διαστήματος [a, b] τέτοια ώστε οι τιμές f(a) και f(b) να είναι ετερόσημες, $f(a)f(b) < 0$
2. Υπολογισμός του μέσου του διαστήματος $c = \frac{a+b}{2}$
3. Έλεγχος του σημείου c
   (a) Αν f(c) = 0, τότε το c είναι η ρίζα και επομένως τερματίζεται η διαδικασία.
   (b) Αν $f(a)f(c) < 0$, τότε η ρίζα βρίσκεται στο διάστημα [a, c], ορίζεται b = c και επαναλαμβάνεται η διαδικασία.
   (c) Αν $f(c)f(b) < 0$, τότε η ρίζα βρίσκεται στο διάστημα [c, b], ορίζεται a = c και επαναλαμβάνεται η διαδικασία.

Ως κριτήριο τερματισμού ορίζεται να είναι

$$|f(c)| < 0.0001$$

Ως αρχικό διάστημα ορίστηκε να είναι το [0, 10]. Ο αλγόριθμος φαίνεται να συγκλίνει έπειτα από 12 επαναλήψεις με την τιμή του γ να είναι τελικά ίση με 3.46923828125 και την τιμή του y(30) να είναι $7.000463394814336e-05 \approx 0$.

Στο ακόλουθο διάγραμμα φαίνεται η επίλυση των διαφορικών για γ = 3.46923828125 και για timespan = [0, 30s].



Κώδικες

Η υλοποίηση της μεθόδου Runge - Kutta 4 σε Python φαίνεται παρακάτω

```
# Runge - Kutta 4th order
def RungeKutta4thOrder(func1, func2, x0, y0, t0, tf, h):
    n = int((tf - t0) / h)

    t = np.linspace(t0, tf, n + 1)
    x = np.zeros(n + 1)
    y = np.zeros(n + 1)

    x[0] = x0
    y[0] = y0

    for i in range(n):
        k0_1 = h * func1(y[i])
        k1_1 = h * func1(y[i] + 1/2 * k0_1)
        k2_1 = h * func1(y[i] + 1/2 * k1_1)
        k3_1 = h * func1(y[i] + k2_1)
        x[i + 1] = x[i]+ 1/6 * (k0_1 + 2 * k1_1 + 2 * k2_1 + k3_1)


        k0_2 = h * func2(x[i], y[i])
        k1_2 = h * func2(x[i] + 1/2 * h, y[i] + 1/2 * k0_2)
        k2_2 = h * func2(x[i] + 1/2 * h, y[i] + 1/2 * k1_2)
        k3_2 = h * func2(x[i] + h, y[i] + k2_2)
        y[i + 1] = y[i] + 1/6 * (k0_2 + 2 * k1_2 + 2 * k2_2 + k3_2)

    return x, y, t
```

Η υλοποίηση της μεθόδου της διχοτόμησης η οποία χρησιμοποιεί μια παραλλαγή της συνάρτησης RungeKutta4thOrder του προηγούμενου ερωτήματος, η οποία δέχεται πλέον την τιμή του γ ως όρισμα κατά την κλήση της

```
def bisection_method():
    gamma_min = 0
    gamma_max = 10

    array1, ymin, t = RungeKutta4thOrder(func1, func2, x0, y0, t0, tf, h
    , gamma_min)
    array2, ymax, t = RungeKutta4thOrder(func1, func2, x0, y0, t0, tf, h
    , gamma_max)

    if ymin * ymax > 0:
        raise ValueError("The function must have different signs at the
    endpoints c_min and c_max.")

    for i in range(30):
        gamma_middle = (gamma_min + gamma_max) / 2
        array3, ymid, t = RungeKutta4thOrder(func1, func2, x0, y0, t0,
    tf, h, gamma_middle)

        if abs(ymid) < 0.0001:
            print("entered case 1")
            return gamma_middle

        if ymin * ymid < 0:
            print("entered case 2")
            gamma_max = gamma_middle
            ymax = ymid
        else:
            print("entered case 3")
            gamma_min = gamma_middle
            ymin = ymid

    return (gamma_min + gamma_max)/2
```

Επέκταση μεθόδου Newton για την εύρεση ρίζας σε κώδικα Python

```
def NewtonMethod2_version1(f, f_dot, f_double_dot, x0):

    # Define number of iterations
    x_values = [x0]

    i = 0
    max_iterations = 30
    flag1 = 0 # flag turns to 1 only if convergence is achieved

    while (i < max_iterations):
        delta = f_dot.subs(x, x_values[i])**2 - 2 * f_double_dot.subs(x,
    x_values[i]) * f.subs(x, x_values[i])

        if delta < 0:
            print("Version 1: Discriminant is negative, process is
   terminated")
            break
        elif f_double_dot.subs(x, x_values[i]) == 0:
            print("Version 1: Denominator is zero, process is terminated
   ")
            break
        else:
            var1 = -(f_dot.subs(x, x_values[i]) - x_values[i] *
   f_double_dot.subs(x, x_values[i]))
            var2 = math.sqrt(f_dot.subs(x, x_values[i])**2 - 2 *
   f_double_dot.subs(x, x_values[i]) * f.subs(x, x_values[i]))
            var3 = f_double_dot.subs(x, x_values[i])

            x_values.append((var1 + var2)/var3)

            i = i + 1

            # Check Convergence
            e = 0.0001
            if (abs(x_values[i] - x_values[i - 1]) <= e):
                print("Version 1: Convergence achieved.")
                flag1 = 1
                break
    return x_values, flag1
```

Μέθοδος Halley σε κώδικα Python

```
    # Halley's Method
    def HalleyMethod(f, f_dot, f_double_dot, x0):
    x_values = np.zeros(20)
    x_values[0] = x0
    i = 0
    while(1):
        if 2 * f_dot.subs(x, x_values[i])**2 - f_double_dot.subs(x,
    x_values[i]) * f.subs(x, x_values[i]) != 0:

            x_values[i + 1] = x_values[i] - (2 * f.subs(x, x_values[i])
    * f_dot.subs(x, x_values[i]))/(2 * f_dot.subs(x, x_values[i])**2 -
    f_double_dot.subs(x, x_values[i]) * f.subs(x, x_values[i]))
            # Check Convergence
            e = 0.1
            if (abs(x_values[i + 1] - x_values[i]) <= e):
                print("Convergence achieved")
                break

            if (i > 21):
                print("Failed to converge, choose new initial value and
    try again.")
                break

            i = i + 1
        else:
            print("Failed to converge, choose new initial value and try
    again.")

    return x_values
```