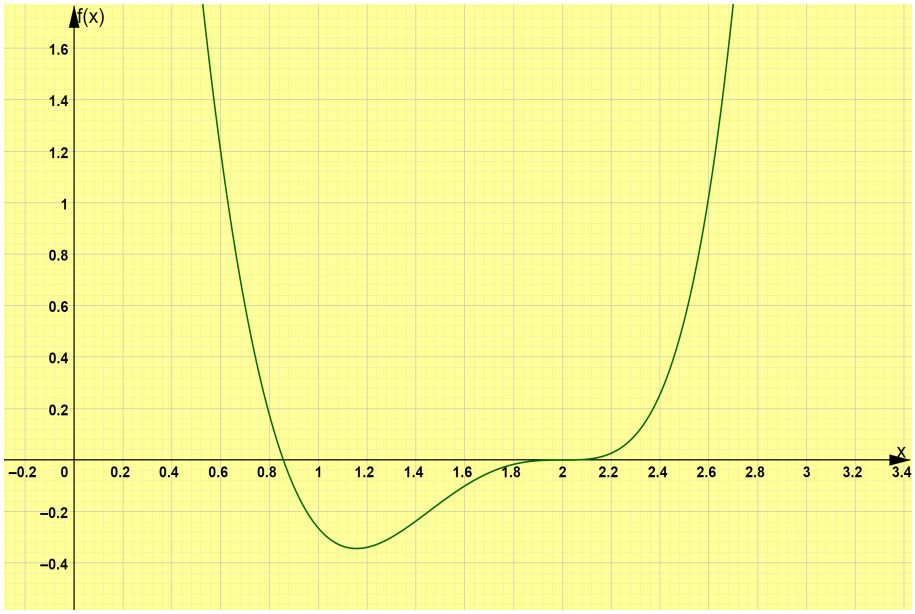
Πρώτη Εργασία Στο Μάθημα της Αριθμητικής Ανάλυσης

Ονοματεπώνυμο: Σταύρος Νικολαΐδης
ΑΕΜ: 3975

Δεκέμβριος 2021

1 Πρώτη Άσκηση



Γραφική παράσταση της συνάρτησης: $f(x) = 14xe^{x-2} - 12e^{x-2} - 7x^3 + 20x^2 - 26x + 12$

Μερικά γενικά για την συνάρτηση:
Όπως παρατηρούμε η συνάρτηση είναι τρίτου βαθμού επομένως περιμένουμε να έχει το πολύ τρεις ρίζες. Αρχικά στην γραφική παράσταση φαίνεται να έχουμε 2 ρίζες, μία κοντά στο 0,8 και μία κοντά στο 2. Θα χρειαστεί να επιβεβαιώσουμε αν κάποια από αυτές είναι διπλή ρίζα.

1.1 Μέθοδος της διχοτόμισης:

Αρχείο κώδικα: "ex 1a.py"

Η μέθοδος της διχοτόμισης ψάχνει την ρίζα αναδρομικά σε ένα διάστημα που όλο και μικραίνει (διχοτομείται) χρησιμοποιόντας το μέσο του διαστήματος για να ελέγξει αν αυτό μηδενίζει την συνάρτηση. Σε περίπτωση που δεν την μηδενίζει, δημιουργείται ένα νέο διάστημα I:

Αν $f(a)f(m) < 0$ τότε I = (a, m).

Αλλιώς αν $f(a)f(m) \geq 0$ τότε I = (m, b).

όπου m το μέσο του αρχικού διαστήματος (a, b).

Σημείωση: σε κάθε διάστημα I πρέπει να ισχύει το θεώρημα Bolzano ώστε να υπάρχει τουλάχιστον μία ρίζα για να βρει η μέθοδος.

Συνθήκη τερματισμού: Η μέθοδος τερματίζει όταν η διαφορά των άκρων του διαστήματος γίνει μικρότερη του σφάλματος ανοχής που στην προκειμένη περίπτωση είναι ίσο με $\varepsilon = \frac{10^{-5}}{2}$.

Η μέθοδος της διχοτόμησης έχει μερικά προβλήματα στον υπολογισμό της ριζάς. Αρχικά δεν μπορεί να βρει και τις 2 ρίζες που υπάρχουνε. Επομένως θα κάνουμε χρήση της μεθόδου σε δύο ξεχωριστά διαστήματα, [0, 1.5] και [1.5, 3] ώστε να βρούμε της ρίζες.

Με την μέθοδο της διχοτόμισης μπορούμε να ξέρουμε πριν καν τρέξουμε το πρόγραμμα τον μικρότερο αριθμό επαναλήψεων που θα χρειαστούμε για να προσεγγίσουμε την ρίζα εφαρμόζοντας τον εξής τύπο:

$$n > \frac{\ln(b-a) - \ln(\varepsilon)}{\ln(2)}$$

Όπου: n ο αριθμός των επαναλήψεων, a και b τα άκρα του διαστήματος και $\varepsilon = 0.5 * 10^k$, όπου k τα ψηφία για την ακρίβεια που θέλουμε.

Επομένως για να προσεγγίσουμε τις 2 ρίζες με ακρίβεια 5 δεκαδικών ψηφίων, που και οι δυο βρίσκονται σε ένα διάστημα με εύρος 1.5, θα

χρειαστούμε τουλάχιστον 19 επαναλήψεις.

Αποτελέσματα για την πρώτη ρίζα:

```
Δώσε διάστημα για εύρεση ρίζας:
a:
0
b:
1.5
n: 1, a: 0.00000, b: 1.50000, c: 0.75000, f(c): 0.36712
n: 2, a: 0.75000, b: 1.50000, c: 1.12500, f(c): -0.34106
n: 3, a: 0.75000, b: 1.12500, c: 0.93750, f(c): -0.17591
n: 4, a: 0.75000, b: 0.93750, c: 0.84375, f(c): 0.03704
n: 5, a: 0.84375, b: 0.93750, c: 0.89062, f(c): -0.08260
n: 6, a: 0.84375, b: 0.89062, c: 0.86719, f(c): -0.02625
n: 7, a: 0.84375, b: 0.86719, c: 0.85547, f(c): 0.00450
n: 8, a: 0.85547, b: 0.86719, c: 0.86133, f(c): -0.01110
n: 9, a: 0.85547, b: 0.86133, c: 0.85840, f(c): -0.00335
n: 10, a: 0.85547, b: 0.85840, c: 0.85693, f(c): 0.00056
n: 11, a: 0.85693, b: 0.85840, c: 0.85767, f(c): -0.00140
n: 12, a: 0.85693, b: 0.85767, c: 0.85730, f(c): -0.00042
n: 13, a: 0.85693, b: 0.85730, c: 0.85712, f(c): 0.00007
n: 14, a: 0.85712, b: 0.85730, c: 0.85721, f(c): -0.00018
n: 15, a: 0.85712, b: 0.85721, c: 0.85716, f(c): -0.00005
n: 16, a: 0.85712, b: 0.85716, c: 0.85714, f(c): 0.00001
n: 17, a: 0.85714, b: 0.85716, c: 0.85715, f(c): -0.00002
n: 18, a: 0.85714, b: 0.85715, c: 0.85715, f(c): -0.00001
n: 19, a: 0.85714, b: 0.85715, c: 0.85714, f(c): 0.00000
--------------
c: 0.85714
f(c): 0.00000
```

Όπως βλέπουμε η μέθοδος χρειάστηκε 19 επαναλήψεις ώστε να βρει προσεγγιστικά την πρώτη ρίζα, δηλαδή την 0.85714 στο διάστημα [0, 1.5].

Αποτελέσματα για την δεύτερη ρίζα:

```
a:
1.5
b:
3
n: 1, a: 1.50000, b: 3.00000, c: 2.25000, f(c): 0.05412
n: 2, a: 1.50000, b: 2.25000, c: 1.87500, f(c): -0.00450
n: 3, a: 1.87500, b: 2.25000, c: 2.06250, f(c): 0.00070
n: 4, a: 1.87500, b: 2.06250, c: 1.96875, f(c): -0.00008
n: 5, a: 1.96875, b: 2.06250, c: 2.01562, f(c): 0.00001
n: 6, a: 1.96875, b: 2.01562, c: 1.99219, f(c): -0.00000
n: 7, a: 1.99219, b: 2.01562, c: 2.00391, f(c): 0.00000
n: 8, a: 1.99219, b: 2.00391, c: 1.99805, f(c): -0.00000
n: 9, a: 1.99805, b: 2.00391, c: 2.00098, f(c): 0.00000
n: 10, a: 1.99805, b: 2.00098, c: 1.99951, f(c): -0.00000
n: 11, a: 1.99951, b: 2.00098, c: 2.00024, f(c): 0.00000
n: 12, a: 1.99951, b: 2.00024, c: 1.99988, f(c): -0.00000
n: 13, a: 1.99988, b: 2.00024, c: 2.00006, f(c): 0.00000
n: 14, a: 1.99988, b: 2.00006, c: 1.99997, f(c): -0.00000
n: 15, a: 1.99997, b: 2.00006, c: 2.00002, f(c): 0.00000
n: 16, a: 1.99997, b: 2.00002, c: 1.99999, f(c): -0.00000
n: 17, a: 1.99999, b: 2.00002, c: 2.00000, f(c): 0.00000
--------------
c: 2.00000
f(c): 0.00000
```

Όπως βλέπουμε η μέθοδος χρειάστηκε 17 επαναλήψεις ώστε να βρει την δεύτερη ρίζα, δηλαδή την 2.

1.2 Μέθοδος Newton-Raphson:

Αρχείο κώδικα: "ex 1b.py"

Η μέθοδος Newton-Raphson αξιοποιεί την πρώτη παράγωγο της

συνάρτησης και μέσω του αναδρομικού τύπου:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}, \quad k \geq 0$$

όπου x0 ένα αρχικό σημείο για το οποίο ισχύει: $f(x_0) * f''(x_0) > 0$, προσεγγίζουμε την ρίζα της εξίσωσης. Η μέθοδος τερματίζει αν η απόλυτη τιμή του $h = \frac{f(x^{(k)})}{f'(x^{(k)})}$ γίνει μικρότερη του σφάλματος ανοχής το οποίο είναι ίσο με $\varepsilon = 10^{(-5)}$ (γιατί θέλουμε ακρίβεια 5 ψηφείων).

Αποτελέσματα για την πρώτη ρίζα:

```
Δώσε το x0:
0
Ισχύει το f(x0) * f''(x0) > 0!
n: 1, x[1]: 0.40327, f(x): 3.02129
n: 2, x[2]: 0.66072, f(x): 0.81265
n: 3, x[3]: 0.80106, f(x): 0.17139
n: 4, x[4]: 0.85075, f(x): 0.01738
n: 5, x[5]: 0.85705, f(x): 0.00026
n: 6, x[6]: 0.85714, f(x): 0.00000
n: 7, x[7]: 0.85714, f(x): 0.00000
--------------------
c: 0.85714
f(c): 0.00000
Συγκλίνει τετραγωνικά!
```

Όπως βλέπουμε, με αρχικό σημείο $x_0 = 0$ (για το οποίο ισχύει ο παραπάνω τύπος), χρειάστηκαν πολύ λιγότερες επαναλήψεις για την προσέγγιση της πρώτης ρίζας από την μέθοδο της διχοτόμισης. Επίσης η πρώτη ρίζα συγκλίνει τετραγωνικά αφού ισχύει η συνθήκη:

$$f(c) = 0 \quad \text{και} \quad f'(c) \neq 0 \quad \text{όπου c η ρίζα της f}$$

Προσοχή!
Αυτό δεν σημαίνει ότι η μέθοδος Newton-Raphson είναι πιο γρήγορη από την μέθοδο της Διχοτόμισης. Δείτε παρακάτω για την επόμενη ρίζα.

Αποτελέσματα για την δεύτερη ρίζα:

```
Δώσε το x0:
1.5
Ισχύει το f(x0) * f''(x0) > 0!
n: 1, x[1]: 1.73739, f(x): -0.03488
n: 2, x[2]: 1.83692, f(x): -0.00952
n: 3, x[3]: 1.89529, f(x): -0.00271
n: 4, x[4]: 1.93173, f(x): -0.00078
n: 5, x[5]: 1.95512, f(x): -0.00023
n: 6, x[6]: 1.97034, f(x): -0.00007
n: 7, x[7]: 1.98034, f(x): -0.00002
n: 8, x[8]: 1.98694, f(x): -0.00001
n: 9, x[9]: 1.99132, f(x): -0.00000
n: 10, x[10]: 1.99422, f(x): -0.00000
n: 11, x[11]: 1.99615, f(x): -0.00000
n: 12, x[12]: 1.99744, f(x): -0.00000
n: 13, x[13]: 1.99829, f(x): -0.00000
n: 14, x[14]: 1.99886, f(x): -0.00000
n: 15, x[15]: 1.99924, f(x): -0.00000
n: 16, x[16]: 1.99949, f(x): -0.00000
n: 17, x[17]: 1.99966, f(x): -0.00000
n: 18, x[18]: 1.99978, f(x): -0.00000
n: 19, x[19]: 1.99985, f(x): -0.00000
n: 20, x[20]: 1.99990, f(x): -0.00000
n: 21, x[21]: 1.99993, f(x): -0.00000
n: 22, x[22]: 1.99996, f(x): -0.00000
n: 23, x[23]: 1.99997, f(x): -0.00000
n: 24, x[24]: 1.99998, f(x): -0.00000
n: 25, x[25]: 1.99999, f(x): -0.00000
-------------------
c: 1.99999
f(c): -0.00000
Δεν συγκλίνει τετραγωνικά!
```

Όπως βλέπουμε η μέθοδος Newton-Raphson χρειάστηκε 25 επαναλήψεις για την προσσεγιστική εύρεση της 2ης ρίζας με $x_0 = 1.5$. Αυτό συμ-

βαίνει γιατί η δεύτερη ρίζα μηδενίζει την πρώτη παράγωγο της f για αυτό και η ρίζα δεν συγκλίνει τετραγωνικά και για αυτό θέλει περισσότερες επαναλήψεις με την μέθοδο Newton-Raphson.

Επομένως το χαρακτηριστικό των ριζών που δεν συγκλίνουν τετραγωνικά για την μέθοδο Newton-Raphson είναι ότι μηδενίζουν την πρώτη παράγωγο της συνάρτησης. Πράγμα που κάνει πιο δύσκολη την αναζήτηση της ρίζας αφού η μέθοδος αξιοποιεί την πρώτη παράγωγο ως παρανομαστή κλάσματος.

1.3 Μέθοδος τέμνουσας:

Αρχείο κώδικα: "ex 1c.py"

Σε περίπτωση που η παράγωγος της συνάρτησης δεν υπάρχει ή δεν μπορεί να υπολογιστεί εύκολα μπορούμε να κάνουμε χρήση της μεθόδου της τέμνουσας αλλάζοντας λίγο τον αναδρομικό τύπο της Newton-Raphson σε:

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

Επομένως χρειαζόμαστε δύο αρχικά σημεία x_0, x_1 για να κάνουμε χρήση του τύπου και με κατάλληλες αρχικοποιήσεις να προσεγγίσουμε τις δύο ρίζες. Η μέθοδος τερματίζει μόλις η απόλυτη τιμή της διαφοράς $x_{n+1} - x_n$ γίνει μικρότερη του σφάλματος ανοχής το οποίο είναι ίσο με $\varepsilon = 10^{(-5)}$

Αποτελέσματα για την πρώτη ρίζα:

```
Δώσε το x0:
0
Δώσε το x1:
1
n: 1, x0: 0.00000, x1: 1.00000, x2: 0.97517, f(x2): -0.23372
n: 2, x1: 1.00000, x2: 0.97517, x3: 0.78496, f(x3): 0.22881
n: 3, x2: 0.97517, x3: 0.78496, x4: 0.87905, f(x4): -0.05563
n: 4, x3: 0.78496, x4: 0.87905, x5: 0.86065, f(x5): -0.00932
n: 5, x4: 0.87905, x5: 0.86065, x6: 0.85695, f(x6): 0.00052
n: 6, x5: 0.86065, x6: 0.85695, x7: 0.85714, f(x7): -0.00000
n: 7, x6: 0.85695, x7: 0.85714, x8: 0.85714, f(x8): -0.00000
--------------------
c: 0.85714
f(c): -0.00000
```

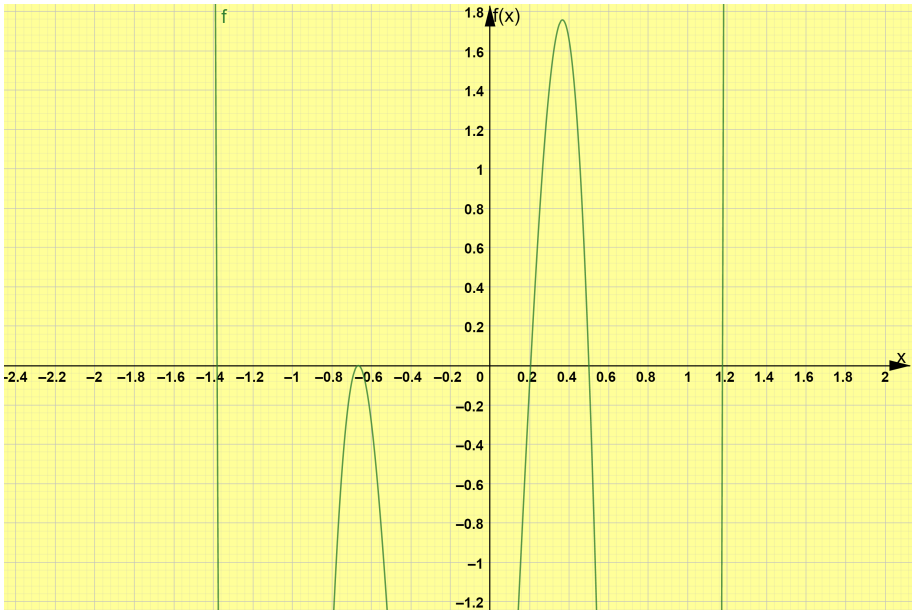
Παρατηρούμε ότι με αρχικά σημεία $x_0 = 0, x_1 = 1$ η μέθοδος χρειάστηκε 7 επαναλήψεις για να προσεγγίσει την πρώτη ρίζα.

Αποτελέσματα για την δεύτερη ρίζα:

```
n: 1, x0: 1.60000, x1: 2.20000, x2: 2.07545, f(x2): 0.00124
n: 2, x1: 2.20000, x2: 2.07545, x3: 2.06928, f(x3): 0.00096
n: 3, x2: 2.07545, x3: 2.06928, x4: 2.04873, f(x4): 0.00033
n: 4, x3: 2.06928, x4: 2.04873, x5: 2.03813, f(x5): 0.00015
n: 5, x4: 2.04873, x5: 2.03813, x6: 2.02860, f(x6): 0.00006
n: 6, x5: 2.03813, x6: 2.02860, x7: 2.02177, f(x7): 0.00003
n: 7, x6: 2.02860, x7: 2.02177, x8: 2.01645, f(x8): 0.00001
n: 8, x7: 2.02177, x8: 2.01645, x9: 2.01245, f(x9): 0.00001
n: 9, x8: 2.01645, x9: 2.01245, x10: 2.00941, f(x10): 0.00000
n: 10, x9: 2.01245, x10: 2.00941, x11: 2.00712, f(x11): 0.00000
n: 11, x10: 2.00941, x11: 2.00712, x12: 2.00538, f(x12): 0.00000
n: 12, x11: 2.00712, x12: 2.00538, x13: 2.00406, f(x13): 0.00000
n: 13, x12: 2.00538, x13: 2.00406, x14: 2.00307, f(x14): 0.00000
n: 14, x13: 2.00406, x14: 2.00307, x15: 2.00232, f(x15): 0.00000
n: 15, x14: 2.00307, x15: 2.00232, x16: 2.00175, f(x16): 0.00000
n: 16, x15: 2.00232, x16: 2.00175, x17: 2.00132, f(x17): 0.00000
n: 17, x16: 2.00175, x17: 2.00132, x18: 2.00100, f(x18): 0.00000
n: 18, x17: 2.00132, x18: 2.00100, x19: 2.00075, f(x19): 0.00000
n: 19, x18: 2.00100, x19: 2.00075, x20: 2.00057, f(x20): 0.00000
n: 20, x19: 2.00075, x20: 2.00057, x21: 2.00043, f(x21): 0.00000
n: 21, x20: 2.00057, x21: 2.00043, x22: 2.00032, f(x22): 0.00000
n: 22, x21: 2.00043, x22: 2.00032, x23: 2.00024, f(x23): 0.00000
n: 23, x22: 2.00032, x23: 2.00024, x24: 2.00018, f(x24): 0.00000
n: 24, x23: 2.00024, x24: 2.00018, x25: 2.00014, f(x25): 0.00000
n: 25, x24: 2.00018, x25: 2.00014, x26: 2.00011, f(x26): 0.00000
n: 26, x25: 2.00014, x26: 2.00011, x27: 2.00008, f(x27): 0.00000
n: 27, x26: 2.00011, x27: 2.00008, x28: 2.00006, f(x28): 0.00000
n: 28, x27: 2.00008, x28: 2.00006, x29: 2.00005, f(x29): 0.00000
n: 29, x28: 2.00006, x29: 2.00005, x30: 2.00003, f(x30): 0.00000
n: 30, x29: 2.00005, x30: 2.00003, x31: 2.00003, f(x31): 0.00000
------------------
c: 2.00003
f(c): 0.00000
```

Παρατηρούμε ότι με αρχικά σημεία $x_0 = 1.6, x_1 = 2.2$ η μέθοδος χρειάστηκε 30 επαναλήψεις για να προσεγγίσει την δεύτερη ρίζα.

2 Δεύτερη Άσκηση



Γραφική παράσταση της συνάρτησης: $f(x) = 54x^6 + 45x^5 - 102x^4 - 69x^3 + 35x^2 + 16x - 4$

Μερικά γενικά για την συνάρτηση:
Όπως παρατηρούμε η συνάρτηση είναι έκτου βαθμού επομένως περιμένουμε να έχει το πολύ έξι ρίζες. Αρχικά στην γραφική παράσταση φαίνεται να έχουμε 5 ρίζες.

2.1 Ερώτημα 1:

2.1.1 Τροποποιημένη μέθοδος Newton-Raphson:

Αρχείο κώδικα: "ex 2a.py"
Αυτό που χρειάζεται να αλλάξουμε στην Newton-Raphson για να έχουμε την νέα τροποποιημένη μέθοδο είναι το όριο h από τον τύπο $x_{n+1} = x_n = h$ σε:

$$h = \frac{1}{\frac{f'(x_n)}{f_(x_n)} - \frac{1}{2}\frac{f''(x_n)}{f'(x_n)}}$$

Η νέα μέθοδος τερματίζει όταν η απόλυτη τιμή του h γίνει μικρότερη

του σφάλματος ανοχής $\varepsilon = 10^{(-5)}$.

Αποτελέσματα πρώτης ρίζας:

```
Δώσε το x0:
-2
Ισχύει το f(x0) * f''(x0) > 0!
n: 1, x[1]: -1.62016, f(x): 126.90168
n: 2, x[2]: -1.42638, f(x): 11.49297
n: 3, x[3]: -1.38211, f(x): 0.17189
n: 4, x[4]: -1.38130, f(x): 0.00000
----------------------
c: -1.38130
f(c): 0.00000
```

Παρατηρούμε ότι με $x_0 = -2$ η μέθοδος χρειάστηκε μόνο 4 επαναλήψεις για να βρει την πρώτη ρίζα, δηλαδή την $c_1 = -1.38130$.

Αποτελέσματα δεύτερης ρίζας:

```
Δώσε το x0:
-1
Ισχύει το f(x0) * f''(x0) > 0!
n: 1, x[1]: -0.80488, f(x): -1.55246
n: 2, x[2]: -0.71519, f(x): -0.17918
n: 3, x[3]: -0.68314, f(x): -0.02000
n: 4, x[4]: -0.67219, f(x): -0.00222
n: 5, x[5]: -0.66851, f(x): -0.00025
n: 6, x[6]: -0.66728, f(x): -0.00003
n: 7, x[7]: -0.66687, f(x): -0.00000
n: 8, x[8]: -0.66674, f(x): -0.00000
n: 9, x[9]: -0.66669, f(x): -0.00000
n: 10, x[10]: -0.66667, f(x): -0.00000
----------------------
c: -0.66667
f(c): -0.00000
```

Παρατηρούμε ότι με $x_0 = -1$ η μέθοδος χρειάστηκε 10 επαναλήψεις για να βρει την δεύτερη ρίζα, δηλαδή την $c_2 = -0.66667$.

Αποτελέσματα τρίτης ρίζας:

```
Δώσε το x0:
0.2
Ισχύει το f(x0) * f''(x0) > 0!
n: 1, x[1]: 0.20518, f(x): -0.00002
----------------------
c: 0.20518
f(c): -0.00002
```

Παρατηρούμε ότι με $x_0 = 0.2$ η μέθοδος χρειάστηκε μόνο 1 επανάληψη για να βρει την τρίτη ρίζα, δηλαδή την $c_3 = 0.20518$.

Αποτελέσματα τέταρτης ρίζας:

```
Δώσε το x0:
0.66
Ισχύει το f(x0) * f''(x0) > 0!
n: 1, x[1]: 0.51985, f(x): -0.59252
n: 2, x[2]: 0.50011, f(x): -0.00290
n: 3, x[3]: 0.50000, f(x): -0.00000
----------------------
c: 0.50000
f(c): -0.00000
```

Παρατηρούμε ότι με $x_0 = 0.66$ η μέθοδος χρειάστηκε μόνο 3 επαναλήψεις για να βρει την τέταρτη ρίζα, δηλαδή την $c_4 = 0.5$.

Αποτελέσματα πέμπτη ρίζας:

```
Δώσε το x0:
3
Ισχύει το f(x0) * f''(x0) > 0!
n: 1, x[1]: 2.18679, f(x): 5299.80754
n: 2, x[2]: 1.64484, f(x): 674.52491
n: 3, x[3]: 1.32120, f(x): 76.67953
n: 4, x[4]: 1.18997, f(x): 4.50523
n: 5, x[5]: 1.17614, f(x): 0.00748
n: 6, x[6]: 1.17612, f(x): 0.00000
----------------------
c: 1.17612
f(c): 0.00000
```

Παρατηρούμε ότι με $x_0 = 0.3$ η μέθοδος χρειάστηκε 6 επαναλήψεις για να βρει την πέμπτη ρίζα, δηλαδή την $c_5 = 1.17612$.

2.1.2 Τροποποιημένη μέθοδος Διχοτόμισης:

Αρχείο κώδικα: "ex 2b.py"
Το μόνο που χρειάζεται να αλλάξουμε στον κώδικα της μεθόδου της διχοτόμισης για την τροποποίηση είναι την επιλογή του σημείου c από $c = \frac{a+b}{2}$ όπου πλέον το σημείο επιλέγεται τυχαία μέσω της συνάρτησης uniform(a, b) (από την βιβλιοθήκη random) η οποία επιστρέφει μια τυχαία τιμή από το διάστημα [a, b]. Θα ψάξουμε λοιπόν τις ρίζες ανα διαστήματα.

Αποτελέσματα πρώτης ρίζας:

```
Δώσε διάστημα για εύρεση ρίζας:
a:
-2
b:
-1
n: 1.00000 a: -2.00000, b: -1.00000, c: -1.59048, f(c): 100.11279
n: 2.00000 a: -1.59048, b: -1.00000, c: -1.03962, f(c): -10.89817
n: 3.00000 a: -1.59048, b: -1.03962, c: -1.34050, f(c): -7.16120
```

ΟΙ ΥΠΟΛΟΙΠΕΣ 20 ΕΠΑΝΑΛΗΨΕΙΣ

```
n: 23.00000 a: -1.38132, b: -1.38128, c: -1.38130, f(c): -0.00049
n: 24.00000 a: -1.38132, b: -1.38130, c: -1.38130, f(c): 0.00071
n: 25.00000 a: -1.38130, b: -1.38130, c: -1.38130, f(c): 0.00047
----------------------
c: -1.38130
f(c): 0.00047
```

Παρατηρούμε ότι με $a = -2$και $b = -1$ η μέθοδος χρειάστηκε 25 επαναλήψεις για να βρει την πρώτη ρίζα, δηλαδή την $c_1 = -1.38130$.

Αποτελέσματα δεύτερης ρίζας:
Επειδή δεν ισχύει το θεώρημα Bolzano στο διάστημα εύρεσης της δεύτερης ρίζας η μέθοδος της διχοτόμισης δεν μπορεί να βρει την δεύτερη ρίζα, δηλαδή την $c_2 = -0.66667$.

Αποτελέσματα τρίτης ρίζας:

```
Δώσε διάστημα για εύρεση ρίζας:
a:
0
b:
0.4
n: 1.00000 a: 0.00000, b: 0.40000, c: 0.35097, f(c): 1.73668
n: 2.00000 a: 0.00000, b: 0.35097, c: 0.06569, f(c): -2.81926
n: 3.00000 a: 0.06569, b: 0.35097, c: 0.14487, f(c): -1.19884
                ΟΙ ΥΠΟΛΟΙΠΕΣ ΕΠΑΝΑΛΗΨΕΙΣ
n: 24.00000 a: 0.20518, b: 0.20523, c: 0.20522, f(c): 0.00074
n: 25.00000 a: 0.20518, b: 0.20522, c: 0.20521, f(c): 0.00049
n: 26.00000 a: 0.20518, b: 0.20521, c: 0.20518, f(c): 0.00002
----------------------
c: 0.20518
f(c): 0.00002
```

Παρατηρούμε ότι με $a = 0$και $b = 0.4$ η μέθοδος χρειάστηκε 26 επαναλήψεις για να βρει την τρίτη ρίζα, δηλαδή την $c_3 = 0.20518$.

Αποτελέσματα τέταρτης ρίζας:

```
Δώσε διάστημα για εύρεση ρίζας:
a:
0.4
b:
0.6
n: 1.00000 a: 0.40000, b: 0.60000, c: 0.56074, f(c): -2.09964
n: 2.00000 a: 0.40000, b: 0.56074, c: 0.49637, f(c): 0.09842
n: 3.00000 a: 0.49637, b: 0.56074, c: 0.53898, f(c): -1.24973

n: 17.00000 a: 0.49999, b: 0.50000, c: 0.50000, f(c): 0.00011
n: 18.00000 a: 0.50000, b: 0.50000, c: 0.50000, f(c): 0.00009
----------------------
c: 0.50000
f(c): 0.00009
```

Παρατηρούμε ότι με $a = 0.4$και $b = 0.6$ η μέθοδος χρειάστηκε 18 επαναλήψεις για να βρει την τέταρτη ρίζα, δηλαδή την $c_4 = 0.5$.

Αποτελέσματα πέμπτη ρίζας:

```
Δώσε διάστημα για εύρεση ρίζας:
a:
1
b:
1.4
n: 1.00000 a: 1.00000, b: 1.40000, c: 1.30447, f(c): 63.95944
n: 2.00000 a: 1.00000, b: 1.30447, c: 1.25227, f(c): 31.41409

n: 14.00000 a: 1.17546, b: 1.17612, c: 1.17608, f(c): -0.00942
n: 15.00000 a: 1.17608, b: 1.17612, c: 1.17611, f(c): -0.00024
----------------------
c: 1.17611
f(c): -0.00024
```

Παρατηρούμε ότι με $a = 1$και $b = 1.4$ η μέθοδος χρειάστηκε 15 επαναλήψεις για να βρει την πέμπτη ρίζα, δηλαδή την $c_5 = 1.17612$.

2.1.3 Τροποποιημένη μέθοδος Τέμνουσας:

Αρχείο κώδικα: "ex 2c.py"

Στην τροποποιημένη μέθοδο της Τέμνουσας απλά αλλάζουμε τον τύπο υπολογισμού του x_{n+1} αξιοποιώντας τρία αρχικά σημεία. Εδώ θέλει λίγη προσοχή καθώς δεν μπορούμε να δώσουμε το 0 σε αρχικό σημείο γιατί αξιοποιείται στους παρονομαστές των: q, r και s. Η συνθήκη τερματισμού παραμένει ίδια με την κλασσική μέθοδο.

Αποτελέσματα πρώτης ρίζας:

```
Δώσε x0:
-4
Δώσε x1:
-3
Δώσε x2:
-2
n: 1 x: -3.00000, f(x): 22295.00000

n: 9 x: -1.38202, f(x): 0.15171
n: 10 x: -1.38131, f(x): 0.00145
----------------------
xn - x: 0.000007
c: -1.38131
f(c): 0.00145
```

Παρατηρούμε ότι με $x_0 = -4$ και $x_1 = -3$ και $x_2 = -2$ η μέθοδος χρειάστηκε 10 επαναλήψεις για να βρει (προσεγγιστικά) την πρώτη ρίζα, δηλαδή την $c_1 = -1.38130$.

Αποτελέσματα δεύτερης ρίζας:

```
Δώσε x0:
-3
Δώσε x1:
-2
Δώσε x2:
-1
n: 1 x: -2.00000, f(x): 1040.00000

n: 20 x: -0.66669, f(x): -0.00000
n: 21 x: -0.66668, f(x): -0.00000
----------------------
xn - x: 0.000006
c: -0.66668
f(c): -0.00000
```

Παρατηρούμε ότι με $x_0 = -3$ και $x_1 = -2$ και $x_2 = -1$ η μέθοδος χρειάστηκε 21 επαναλήψεις για να βρει (προσεγγιστικά) την πρώτη ρίζα, δηλαδή την $c_2 = -0.66667$.

Αποτελέσματα τρίτης ρίζας:

```
Δώσε x0:
-0.25
Δώσε x1:
0
Δώσε x2:
0.25
n: 1 x: 0.00000, f(x): -4.00000

n: 6 x: 0.20519, f(x): 0.00022
n: 7 x: 0.20518, f(x): 0.00000
-----------------------
xn - x: 0.000000
c: 0.20518
f(c): 0.00000
```

Παρατηρούμε ότι με $x_0 = -0.25$ και $x_1 = 0$ και $x_2 = 0.25$ η μέθοδος χρειάστηκε 7 επαναλήψεις για να βρει την πρώτη ρίζα, δηλαδή την $c_3 = 0.20518$.

Αποτελέσματα τέταρτης ρίζας:

```
Δώσε x0:
0.3
Δώσε x1:
0.6
Δώσε x2:
0.9
n: 1 x: 0.60000, f(x): -3.90458

n: 8 x: 0.49999, f(x): 0.00033
n: 9 x: 0.50000, f(x): -0.00000
-----------------------
xn - x: 0.000000
c: 0.50000
f(c): -0.00000
```

Παρατηρούμε ότι με $x_0 = 0.3$ και $x_1 = 0.6$ και $x_2 = 0.9$ η μέθοδος χρειάστηκε 9 επαναλήψεις για να βρει την πρώτη ρίζα, δηλαδή την $c_4 = 0.5$.

Αποτελέσματα πέμπτης ρίζας:

```
Δώσε x0:
1
Δώσε x1:
2
Δώσε x2:
3
n: 1 x: 2.00000, f(x): 2880.00000

n: 13 x: 1.17623, f(x): 0.03437
n: 14 x: 1.17612, f(x): 0.00006
-----------------------
xn - x: 0.000000
c: 1.17612
```

Παρατηρούμε ότι με $x_0 = 1$ και $x_1 = 2$ και $x_2 = 3$ η μέθοδος χρειάστηκε 14 επαναλήψεις για να βρει την πρώτη ρίζα, δηλαδή την $c_5 = 1.17612$.

2.2 Ερώτημα 2:

Τρέχοντας τον αλγόριθμο 10 φορές στο ίδιο διάστημα παρατηρούμε ότι δεν συγκλίνει πάντα στον ίδιο αριθμό επαναλήψεων, που είναι λογικό αφού το σημείο επιλέγεται τυχαία και αυτό σημαίνει ότι μπορεί άλλες φορές να είναι πιο κοντά στην ρίζα, άρα χρειάζονται λιγότερες επαναλήψεις και άλλες φορές πιο μακριά, άρα απαιτούνται περισσότερες επαναλήψεις.

Συγκεκριμένα για το διάστημα [0.6, 1.6] της f χρειάστηκαν: 29, 29, 25, 26, 28, 23, 36, 24, 17 και 18 επαναλήψεις για την εύρεση της ρίζας $x = 1.17611$.

2.3 Ερώτημα 3:

Για να συγκρίνουμε τους νέους αλγορίθμους με τους κλασσικούς χρειάζεται να τους χρησιμοποιήσουμε για την ίδια συνάρτηση. Επομένως θα χρησιμοποιήσουμε την συνάρτηση f για τους κλασσικούς αλγορίθμους και θα δούμε πόσο γρήγορα ή όχι συγκλίνουν στην πρώτη ρίζα, δηλαδή την $c_1 = -1.38130$ με ίδιες αρχικοποιήσεις.

2.3.1 Κλασσική μέθοδος Newton-Raphson:

Αρχείο κώδικα: "ex 2aa.py"

```
Δώσε το x0:
-2
Ισχύει το f(x0) * f''(x0) > 0!
n: 1, x[1]: -1.76661, f(x): 331.07499

n: 7, x[7]: -1.38130, f(x): 0.00000
n: 8, x[8]: -1.38130, f(x): -0.00000
--------------------
c: -1.38130
f(c): -0.00000
```

Η κλασσική μέθοδος της Newton-Raphson χρειάστηκε 8 επαναλήψεις για να βρει την ρίζα στο διάστημα με $x_0 = -2$. Επομένως μπορούμε να συμπεράνουμε ότι η τροποποιημένη μέθοδος (4 επαναλήψεις με ίδια αρχικοποίηση) είναι πιο γρήγορη από την κλασσική.

2.3.2 Κλασσική μέθοδος της Διχοτόμισης:

Αρχείο κώδικα: "ex 2bb.py"

```
Δώσε διάστημα για εύρεση ρίζας:
a:
-2
b:
-1
n: 1, a: -2.00000, b: -1.00000, c: -1.50000, f(c): 40.62500

n: 17, a: -1.38130, b: -1.38129, c: -1.38129, f(c): -0.00089
n: 18, a: -1.38130, b: -1.38129, c: -1.38130, f(c): -0.00009
--------------
c: -1.38130
f(c): -0.00009
```

Η κλασσική μέθοδος της διχοτόμισης χρειάστηκε 18 επαναλήψεις για να βρει την ρίζα στο διάστημα [-2, -1]. Αυτό δεν σημαίνει απαραίτητα

ότι είναι πιο γρήγορη από την τροποποιημένη μέθοδο καθώς όπως είπανε στο Ερώτημα 2 η νέα μέθοδος χρησιμοποιεί τυχαίο σημείο το οποίο μπορεί να είναι είτε κοντά στην ρίζα, είτε πολύ μακριά από αυτήν. Μπορεί ακόμα και να είναι το ίδιο το σημείο η ρίζα, έτσι μπορεί να αργεί στην σύγκλιση ή να είναι πιο γρήγορη από την κλασσική μέθοδο.

2.3.3 Κλασσική μέθοδος Τέμνουσας:

Αρχείο κώδικα: "ex 2cc.py"
Με την μέθοδο της Τέμνουσας υπάρχει ένα πρόβλημα. Στην τροποποιημένη χρησιμοποιούμε 3 αρχικά σημεία ενώ στην κλασσική 2. Επομένως θα πάρουμε ως αρχικά της κλασσικής τα δύο τελευταία σημεία που πήραμε στην τροποποιημένη, δηλαδή τα $x_1 = -3$ και $x_2 = -2$.

```
Δώσε το x0:
-3
Δώσε το x1:
-2
n: 1, x0: -3.00000, x1: -2.00000, x2: -1.95107, f(x2): 838.89396

n: 10, x9: -1.38235, x10: -1.38134, x11: -1.38130, f(x11): 0.00004
n: 11, x10: -1.38134, x11: -1.38130, x12: -1.38130, f(x12): 0.00000
--------------------
c: -1.38130
f(c): 0.00004
```

Όπως παρατηρούμε η κλασσική μέθοδος χρειάστηκε 11 επαναλήψεις για να προσεγγίσει την ρίζα. Επομένως μπορούμε να πούμε ότι είναι περίπου όσο γρήγορη είναι η τροποποιημένη.

3 Τρίτη Άσκηση

3.1 Ερώτημα 1:

Αρχείο κώδικα: "ex 3a.py"

Θα χρησιμοποίησουμε τον παρακάτω 4x4 πίνακα για την $\mathbf{PA} = \mathbf{LU}$ παραγοντοποίηση. Ο λόγος που χρησιμοποιούμε τετραγωνικό πίνακα είναι είναι γιατί η $\mathbf{LU}$ παραγοντοποίηση παράγει αδύνατο σύστημα για πίνακες nxm με $n < m$ ενώ για $n > m$ δεν ορίζεται καν. Ο πίνακας μας λοιπόν είναι ο:

$$\mathbf{A} = \begin{pmatrix} 0 & 2 & -1 & 1 \\ -1 & 1 & 2 & -1 \\ 2 & -1 & 2 & 2 \\ 1 & 1 & -1 & 2 \end{pmatrix}$$

Ο πίνακας των σταθερών όρων είναι ο:

$$\mathbf{b} = \begin{pmatrix} 6 \\ 3 \\ 14 \\ 8 \end{pmatrix}$$

Η συνάρτηση που προγραμματίσαμε επιστρέφει την λύση του της εξίσωσης: $\mathbf{Ax} = \mathbf{b}$ κάνοντας χρήση της $\mathbf{PA} = \mathbf{LU}$ παραγοντοποίησης. Συγκεκρίμενα, αρχικά ελέγχει αν υπάρχουν μηδενικά στους οδηγούς κάθε σειράς και κάνει αντιμεταθέσεις στον πίνκα $\mathbf{A}$ για να δημιουργήσει τον πίνακα $\mathbf{P}$ ο οποίος είναι αρχικά ένας μοναδιαίος πίνακας και αλλάζει με τις ίδιες αντιμεταθέσεις σειρών που θα γίνουν στον πίνακα $\mathbf{A}$. Έπειτα χρησιμοποιούμε την μέθοδο Gauss για να λύσουμε την εξίσωση: $\mathbf{LY} = \mathbf{Pb}$ όπου $Y = Ux$. Όπως βλέπουμε αξιοποιούμε τον πίνακα $\mathbf{P}$ για να αντιμεταθέσουμε τον πίνακα των σταθερών όρων σε αυτό το σημείο. Στην συνέχεια κάνουμε πάλι Gauss μέθοδο για την εξίσωση $Ux = Y$ αλλά μόνο κάνοντας αντικαταστάσεις των όρων ανάποδα αφού ο πίνακας $\mathbf{U}$ είναι άνω τριγωνικός και επομένως η πρώτη εξάλειψη των κάτω όρων του πίνακα δεν έχει καμία αξία εδώ. Τελικά λαμβάνουμε το διάνυσμα $\mathbf{x}$ με τις λύσεις του συστήματος.

Τα αποτελέσματα:

```
p: [[0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
------------------------
L: [[ 1.          0.          0.          0.        ]
 [-0.          1.          0.          0.        ]
 [-2.          0.5         1.          0.        ]
 [-1.          1.          0.30769231  1.        ]]
------------------------
U: [[-1.          1.          2.         -1.        ]
 [ 0.          2.         -1.          1.        ]
 [ 0.          0.          6.5        -0.5       ]
 [ 0.          0.          0.          0.15384615]]
------------------------
x:[ 8.5  5.   2.5 -1.5]
------------------------
```

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -2 & 0.5 & 1 & 0 \\ -1 & 1 & 0.30769231 & 1 \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} -1 & 1 & 2 & -1 \\ 0 & 2 & -1 & 1 \\ 0 & 0 & 6.5 & -0.5 \\ 0 & 0 & 0 & 0.15384615 \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} 8.5 \\ 5 \\ 2.5 \\ -1.5 \end{bmatrix}$$

3.2 Ερώτημα 2:

Αρχείο κώδικα: "ex 3b.py"

Για το ερώτημα αυτό θα χρησιμοποιήσουμε τον συμμετρικό θετικά ορισμένο πίνακα:

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

Προφανώς είναι συμμετρικός αφού είναι 3x3. Θα αποδείξουμε ότι είναι θετικά ορισμένος. Για να είναι ένας πίνακας θετικά ορισμένος πρέπει όλες οι ιδιοτιμές του είναι θετικές.

$$det(\mathbf{A} - \lambda\mathbf{x}) = \begin{vmatrix} 2-\lambda & 1 & 1 \\ 1 & 2-\lambda & 1 \\ 1 & 1 & 2-\lambda \end{vmatrix} = -(\lambda-1)^2(\lambda-4) \qquad (1)$$

Από την σχέση (1) παρατηρούμε ότι οι ιδιοτιμές του πίνακα $\mathbf{A}$ είναι $\lambda = 1$ (διπλή) και $\lambda = 4$, οι οποίες είναι θετικές. Επομένως απεδείχθη ότι ο πίνακας $\mathbf{A}$ είναι συμμετρικός και θετικά ορισμένος.

Η μέθοδος Cholesky είναι μια επαναληπτική που πάραγει τον κάτω τριγωνικό $\mathbf{L}$ πίνακα του A. Θα χρησιμοποιήσουμε τους παρακάτω τύπους για να υπολογίσουμε τον $\mathbf{L}$:

$$L_{i,j} = (\pm)\sqrt{A_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2} \qquad \text{αν } i = j$$

$$L_{i,j} = \frac{1}{L_{j,j}}(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k}L_{j,k}) \qquad \text{αν } i > j$$

$$L_{i,j} = 0 \qquad \text{αν } i < j$$

Επομένως για τον πίνακα $\mathbf{A}$ έχουμε:

```
A matrix:
[[2. 1. 1.]
 [1. 2. 1.]
 [1. 1. 2.]]
L of A matrix:
[[1.41421356 0.         0.        ]
 [0.70710678 1.22474487 0.        ]
 [0.70710678 0.40824829 1.15470054]]
```

$$\mathbf{L} = \begin{pmatrix} 1.41 & 0 & 0 \\ 0.7 & 1.22 & 0 \\ 0.7 & 0.4 & 1.15 \end{pmatrix}$$

3.3 Ερώτημα 3:

Αρχείο κώδικα: "ex 3c.py"

Αρχικά πρέπει να ελέγξουμε αν ισχύει το θεώρημα της σύγκλισης για το σύστημα της εκφώνησης. Παρατηρούμε ότι για κάθε γραμμή εκτός από την πρώτη και την τελευταία ισχύει:

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}| \Rightarrow |5| > |-4| \quad \text{i = 2, ..., n-1 και j = 1, ..., n}$$

και για τις γραμμές 1 και n

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}| \Rightarrow |5| > |-2| \quad \text{i = 1 και i = n και j = 1, ..., n}$$

Επομένως μπορούμε να αξιοποιήσουμε την μέθοδο Gauss-Seidel για το σύστημα. Η συνάρτηση στο πρόγραμμα δέχεται 3 πίνακες, τον **A**, τον **b** (τους έχουμε αρχικοποιήσει στην αρχή με τις συναρτήσεις filla και fillb) και τον **x**, ο οποίος στην αρχή είναι ένα μηδενικό διάνυσμα. Έχουμε δυο μεταβλητές στο κύριο πρόγραμμα ώστε να κρατάμε τους δύο τελευταίους πίνακες **x** και να ελέγχουμε κάθε φορά αν η διαφορά τους ως προς την άπειρη νορμα είναι μικρότερη ή ίση του σφάλματος το οποίο είναι ίσο με 0.00005. Η μέθοδος Gauss-Seidel προσεγγίζει επαναληπτικά την λύση του συστήματος αξιοποιώντας

τον παρακάτω τύπο τον οποίο έχουμε υλοποιήσει στην συνάρτηση gauss_seidel του προγράμματος:

$$x_i^{(m+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(m)}) \quad \text{i = 1, ..., n και m = 1, ...}$$

Τα αποτελέσματα για n = 10:

```
Δώσε το n:
10
Η λύση του συστήματος: [0.99943503 0.99915176 0.99907157 0.99912607 0.99925817 0.99942322
 0.99958935 0.9997366  0.99985498 0.99994199]
```

Τα αποτελέσματα για n = 10000:

```
Δώσε το n:
10000
Η λύση του συστήματος: [0.99919344 0.99866318 0.99831756 ... 0.99968849 0.99982996 0.99993198]
```

4 Άσκηση 4:

Αρχείο κώδικα: "ex 4.py" (υπάρχουν σχόλια μέσα στον κώδικα για το πότε αρχίζει κάθε υποερώτημα)

Κατασκευάζουμε τον πίνακα G σύμφωνα με την εκφώνηση στις γραμμές 24-29 του αρχείου του κώδικα.

4.1 Ερώτημα 1:

Θα χρειαστεί να τρέξουμε μια δομή επανάληψης ώστε να αθροίσουμε κάθε στήλη του πίνακα **G**, τον οποίο έχουμε ήδη κατασκευάσει, ώστε να δείξουμε ότι όλες οι στήλες έχουν άθροισμα 1, άρα ο πίνακας **G** θα είναι στοχαστικός. Η δομή βρίσκεται στις γραμμές 31-34 όπου αποθηκεύουμε τα αθροίσματα των στηλών στον πίνακα ni και παρακάτω τον εμφανίζουμε:

```
ni: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Επομένως ο **G** είναι στοχαστικός.

4.2 Ερώτημα 2:

Κατασκευάζουμε με την μέθοδο της δυνάμεως των πίνακα p, οποίος στην αρχή είναι ένα τυχαίο διάνυσμα, χρησιμοποιώντας μια δομή επανάληψης, η οποία θα τρέξει 15 φορές, για να ενημερώνουμε τον **p** κάθε φορά που τον πολλαπλασιάζουμε με τον **G** και να τον διαιρούμε με το πρώτο στοιχείο του. Έπειτα τον κανονικοποιούμε διαιρόντας κάθε στοιχείο του με το άθροισμα όλων των στοιχείων του. Ο νέος πίνακας **p** είναι πλέον το ιδιοδιάνυσμα της εκφώνησης.

```
p (for for question 3):
p[1]: 0.023980537746679007
p[2]: 0.02577857575859879
p[3]: 0.02629273441149864
p[4]: 0.02544661409283812
p[5]: 0.03289507609192779
p[6]: 0.033003920636472485
p[7]: 0.036235558480841944
p[8]: 0.036344403025386635
p[9]: 0.06147232756371907
p[10]: 0.07082447724091692
p[11]: 0.13265736305477965
p[12]: 0.07326235647244793
p[13]: 0.09712607609182604
p[14]: 0.2012738089851505
p[15]: 0.12340617034691648
```

4.3 Ερώτημα 3:

Επιλέγουμε την σελίδα 14 και αφαιρούμε την σύνδεση της με την σελίδα 10 καθώς δεν θέλουμε να μεταφέρεται η σημαντικότητα της σε αυτήν και επισης προσθέτουμε 4 νέες συνδέσεις προς την 14 και θα αφαιρέσουμε μια από την 14 προς μια άλλη ώστε να αυξήσουμε τον βαθμό σημαντικότητας της 14. Αρχικά η σημαντικότητα της σελίδας 14 είναι ίση με: $p_{14} = 0.11628928$. Αφού συνδέσουμε τις σελίδες 2, 3, 10 και 11 και αφού αφαιρέσουμε την σύνδεση από την 14 προς την 10, τρέχουμε ξανά τον αλγόριθμο και παρατηρούμε ότι η σημαντικότητα

της σελίδας 14 έχει αυξηθεί σε: $p_{14} = 0.20127381$.

Τα αποτελέσματα:

```
p (for for question 3):
p[1]: 0.023980537746679007
p[2]: 0.02577857575859879
p[3]: 0.02629273441149864
p[4]: 0.02544661409283812
p[5]: 0.03289507609192779
p[6]: 0.033003920636472485
p[7]: 0.036235558480841944
p[8]: 0.036344403025386635
p[9]: 0.06147232756371907
p[10]: 0.07082447724091692
p[11]: 0.13265736305477965
p[12]: 0.07326235647244793
p[13]: 0.09712607609182604
p[14]: 0.2012738089851505
p[15]: 0.12340617034691648
```

4.4 Ερώτημα 4:

Σε αυτό το ερώτημα αλλάζουμε απλά την μεταβλητή q και δημιουργούμε εκ νεού δύο νέους πίνακες **G** και τα ιδιοδιανύσματα **p** τους (τα αποτελέσματα βρίσκονται παρακάτω).

Τα αποτελέσματα:

```
p (with q = 0.02):
p[1]: 0.012845595733971902
p[2]: 0.010590214491750271
p[3]: 0.012102067261192504
p[4]: 0.01668381867144234
p[5]: 0.023489839091679836
p[6]: 0.023855772018430983
p[7]: 0.030959575585295288
p[8]: 0.03132550851204644
p[9]: 0.05987990984707442
p[10]: 0.05926671903460616
p[11]: 0.14679693604738636
p[12]: 0.08275434707882168
p[13]: 0.10659232440206787
p[14]: 0.23336910497093735
p[15]: 0.1494882672532966
```

```
p (with q = 0.6):
p[1]: 0.05083497605901765
p[2]: 0.0557414443006451 4
p[3]: 0.0557444909610471 4
p[4]: 0.0508517326378272
p[5]: 0.05417488031572483
p[6]: 0.05417518498155184
p[7]: 0.05425835848154485
p[8]: 0.05425866314737188
p[9]: 0.06450551919784316
p[10]: 0.08112242064057001
p[11]: 0.0966907912036962
p[12]: 0.06513160535474073
p[13]: 0.07169261993067776
p[14]: 0.11601101871402053
p[15]: 0.07480629407372112
```

4.4.1 q = 0.02:

Για $q = 0.02$ το οποίο είναι μικρότερο από το αρχικό $q = 0.15$ η τάξη των σελίδων πλέον δεν εξαρτάται τόσο από την πιθανότητα μεταπήδησης (τυχαιότητα) αλλά από την είσοδο των σελίδων επομένως είναι πιο πιθανό να μεταφερθούμε σε μια πιο σημαντική σελίδα από μια πιο ασήμαντη, για αυτό και παρατηρείται μια αύξηση στις σημαντικές σελίδες και μια μείωση στις όχι τόσο σημαντικές σελίδες.

4.4.2 q = 0.6:

Όπως παρατηρούμε για $q = 0.6$ το οποίο είναι μεγαλύτερο από το αρχικό $q = 0.15$ η τάξη των σελίδων έχει γίνει σχεδόν ίδια σε όλες, εκτός της 14 την οποία επηρεάσαμε εμείς αρκετά στο προηγούμενο υποερώτημα. Πλέον η πιθανότητα μεταπήδησης q παίζει πιο σημαντικό ρόλο από την είσοδος των σελίδων, επομένως είναι λογικό να έχουμε σχεδόν ίδια τάξη σε κάθε σελίδα.

4.5 Ερώτημα 5:

Αρχικά στο Σχήμα 1 η σελίδα 11 έχει τάξη/βαθμό σημαντικότητας ίση με: $p_{11} = 0.10634572$. Αφού φέρουμε τον πίνακα $\mathbf{A}$ στην αρχική του μορφή και αλλάξουμε τα στοιχεία $A_{(8,11)}$ και $A_{(12,11)}$ σε 3, υπολογίσουμε εκ νεού τον πίνακα $\mathbf{G}$ και το ιδιοδιάνυσμα $\mathbf{p}$ και εκεί βρίσκουμε την νέα τάξη η οποία είναι ίση με: $p_{11} = 0.12398625$, η οποία έχει αυξηθεί σε σχέση με την πρώτη.

```
p (for question 5):
p[1]: 0.02655175840173208
p[2]: 0.028371675459836376
p[3]: 0.025014272018672935
p[4]: 0.01641963351464448
p[5]: 0.03892357584008604
p[6]: 0.037974713085981285
p[7]: 0.031134538991248066
p[8]: 0.030185676237143307
p[9]: 0.07377106803035319
p[10]: 0.1028695867477067
p[11]: 0.12398624612463058
p[12]: 0.07711761978338273
p[13]: 0.12353265721796623
p[14]: 0.12262440148005746
p[15]: 0.14152257706655874
```

Επομένως μπορούμε να συμπεράνουμε ότι αυτή η στρατηγική δουλεύει και αυξάνει κατά λίγο τον βαθμό σημαντικότητας της σελίδας 11.

4.6 Ερώτημα 6:

Από την αφαίρεση της σελίδας 10 του γραφήματος προφανώς η σημαντικότητα της 10 μηδενίζεται και επηρεάζονται πολύ όσες σελίδες η σελίδα 10 έδειχνε, δηλαδή η 13, που ειναι λογικό καθώς η σελίδα 10 είχε μεγάλη σημαντικότητα λόγω των πολλών εισόδων που είχε για αυτό και η τάξη της σελίδας 13 μειώθηκε κατά περίπου 0.08. Από την άλλη μεριά, οι σελίδες που δείχνουν στην σελίδα 10 δηλαδή οι: 5, 6, 7, 9, 14 δεν έδειξαν σημαντική αλλαγή στην τάξη τους, πέρα από

την 9 που είχε έμμεση είσοδο της 10 από την 13. Αυτό είναι λογικό καθώς αν δεν έχουν καμιά άλλη σχέση αυτές οι σελίδες με την 10 τότε είναι λογικό να μην έχει καμία σημαντική επίδραση στην τάξη τους η αφαίρεση της. Το παράδοξο είναι ότι από τις σελίδες που δεν είχαν κάποια άμεσση σχέση με την 10 άλλες είτε αυξησαν την τάξη τους (π.χ. 15), σε άλλες η τάξη μειώθηκε(π.χ. 4) και σε άλλες έμεινε σχεδόν ίδια (π.χ. 8). Πράγμα που μας φέρνει σε θέση να συμπεράνουμε ότι δεν είναι τόσο απλό να μετρήσουμε την σημαντικότητα μιας σελίδας μόνο από τις εισόδους της και την τάξη της δηλαδή. Παρακάτω θα βρείτε τα συγκριτικά αποτελέσματα.

Τα αποτελέσματα:

```
p:
p[1]: 0.02683059243575669
p[2]: 0.029857240899868496
p[3]: 0.029860551708434383
p[4]: 0.026838065890207104
p[5]: 0.039573940990177206
p[6]: 0.03957531259980056
p[7]: 0.03958289008002238
p[8]: 0.039584261689645725
p[9]: 0.07453523618267337
p[10]: 0.1063293020640803
p[11]: 0.10634572460837588
p[12]: 0.07455412427358678
p[13]: 0.1251106829665866
p[14]: 0.11628928027972558
p[15]: 0.12513279333105903
```

```
p (for question 6):
p[1]: 0.04601242271120563
p[2]: 0.03808756237166892
p[3]: 0.028737005471132428
p[4]: 0.017959741348652757
p[5]: 0.0419467825496933
p[6]: 0.0392353030827162
p[7]: 0.03926638416124056
p[8]: 0.03655490469426348
p[9]: 0.04865240259783012
p[10]: 0.0
p[11]: 0.18933129282290223
p[12]: 0.10648172701310489
p[13]: 0.04368063748274988
p[14]: 0.11754442452498877
p[15]: 0.20650940916785085
```