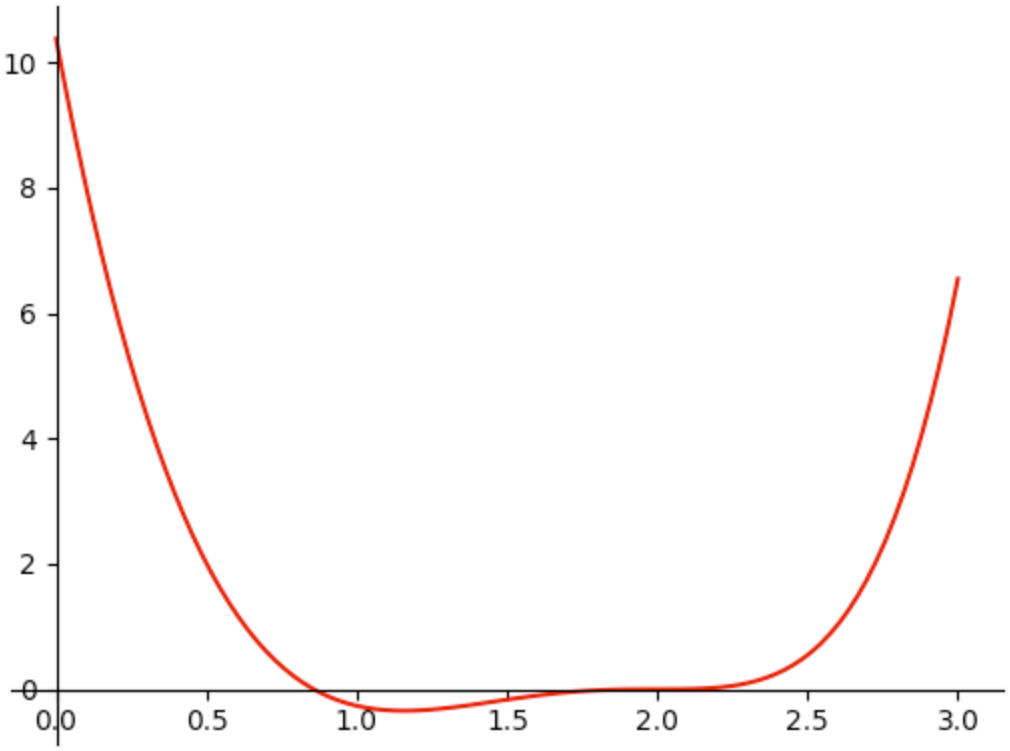
Αριθμητική Ανάλυση - 1η Υποχρεωτική Εργασία

Δημήτριος Αθανασιάδης - 3724

Άσκηση 1

$f(x) = 14xe^{x-2}-12e^{x-2}-7x^3+20x^2-26x+12$ στο διάστημα [0,3]

Γραφική παράσταση της συνάρτησης:



Τρέχοντας το script “askisi1.py” (python3) μπορούμε να υπολογίσουμε τις ρίζες της εξίσωσης f(x)=0 διαδοχικά με τις μεθόδους: α) διχοτόμησης, β) Newton-Raphson, γ) τέμνουσας. Παρατηρούμε πως για την ζητούμενη ακρίβεια 5ου δεκαδικού ψηφίου χρειάζονται οι παρακάτω επαναλήψεις:

	1η Ρίζα	**2η Ρίζα**
Μέθοδος Διχοτόμησης	18	1
Μέθοδος Newton-Raphson	5	13
Μέθοδος Τέμνουσας	47	8030

Παρακάτω βλέπουμε το output από το script “askisi1.py”:

```
$ python3 askisi1.py

f(x) = 14xe^(x-2)-12e^(x-2)-7x^3+20x^2-26x+12

Bisection method:
Number of iterations: 18
First root: 0.85714
Number of iterations: 1
Second root: 2.0

Newton-Raphson method:
Number of iterations: 5
First root: 0.85714
Number of iterations: 13
Second root: 2.00782

Secant method:
Number of iterations: 47
First root: 0.85714
Number of iterations: 8030
Second root: 1.98761
```

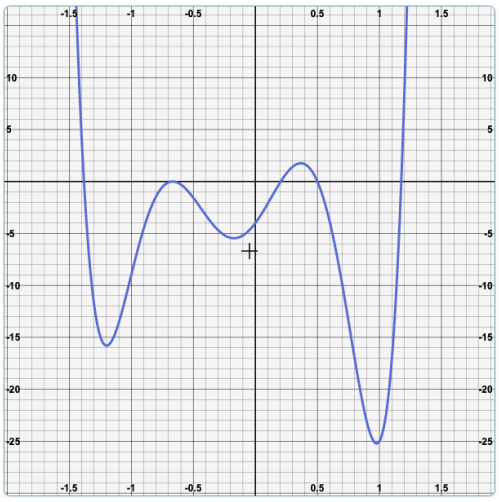
Για να συγκλίνει μια ρίζα τετραγωνικά στη μέθοδο Newton-Raphson θα πρέπει να ισχύουν οι δύο συνθήκες: **φ(ρ) == 0** και **φ'(ρ) != 0**. Επομένως έχουμε:

Για την πρώτη ρίζα (0.85714): **φ(0.85714) = 0** και **φ'(0.85714) = -2.67817**, επομένως η ρίζα 0.85714 συγκλίνει τετραγωνικά.

Για τη δεύτερη ρίζα (2.0): **φ(2) = 0** και **φ'(2) = 0**, άρα η ρίζα 2 δεν συγκλίνει τετραγωνικά καθώς δεν ισχύει η συνθήκη φ'(ρ) != 0

Άσκηση 2

Αφού υλοποιήσουμε τις τροποποιημένες μεθόδους Newton-Raphson, διχοτόμησης και τέμνουσας, ψάχνουμε να βρούμε τις ρίζες της συνάρτησης **$f(x) = 54x^6+45x^5-102x^4-69x^3+35x^2+16x-4$** στο διάστημα [-2,2], το γράφημα της οποίας βλέπουμε δεξιά.



Παρακάτω βλέπουμε το output από το πρώτο κομμάτι του script “askisi2.py”:

```
$ python3 askisi2.py

f(x) = 54x^6+45x^5-102x^4-69x^3+35x^2+16x-4

Newton-Raphson modified method:
Number of iterations: 3
First root: -1.3813
Number of iterations: 6
Second root: -0.66645
Number of iterations: 3
Third root: 0.20518
Number of iterations: 0
Fourth root: 0.5
Number of iterations: 5
Fifth root: 1.17612

Bisection modified method:
Number of iterations: 37
First root: -1.3813
Number of iterations: 26
Second root: 0.20518
Number of iterations: 26
Third root: 0.5
Number of iterations: 26
Fourth root: 1.17612

Secant modified method:
Number of iterations: 3
First root: -1.3813
Number of iterations: 3
Second root: 0.20518
Number of iterations: 4
Third root: 0.5
Number of iterations: 4
Fourth root: 1.17612
```

Παρατηρούμε πως στις τροποποιημένες μεθόδους διχοτόμησης και τέμνουσας (όπως επίσης και στις συμβατικές) χάνουμε την ρίζα $x \approx 0.66645$ καθώς η τιμή της $f(x)$ είναι μικρότερη του μηδενός στα αριστερά και στα δεξιά του $x \approx 0.66645$, επομένως θα έχουμε $f(\alpha)*f(\beta)>0$ για $\alpha<x$ και $\beta>x$ (για παράδειγμα στο διάστημα [-1,-0.5] στο οποίο περιλαμβάνεται η ρίζα $x \approx 0.66645$ έχουμε $f(-1)<0$ και $f(-0.5)<0$), άρα <u>δεν ισχύει</u> η συνθήκη $f(-1)*(-0.5)<0$).

Σχετικά με το ερώτημα 2 της άσκησης 2, εκτελώντας την τροποποιημένη μέθοδο διχοτόμησης 10 φορές για το ίδιο διάστημα παρατηρούμε πως συγκλίνει σε διαφορετικό αριθμό επαναλήψεων κάθε φορά. Αυτό οφείλεται στο ότι η ρίζα δεν είναι το μέσο του διαστήματος αναζήτησης, αλλά ένα τυχαίο σημείο μέσα σε αυτό.

Παρακάτω βλέπουμε το output από το δεύτερο κομμάτι του script “askisi2.py”:

```
Executing modified bisection method 10 times for the same root:
Number of iterations: 38
Number of iterations: 30
Number of iterations: 34
Number of iterations: 34
Number of iterations: 37
Number of iterations: 28
Number of iterations: 26
Number of iterations: 28
Number of iterations: 35
Number of iterations: 38
```

Τα αποτελέσματα περιγράφονται στον παρακάτω πίνακα:

Αριθμός εκτέλεσης	**Αριθμός επαναλήψεων**
1η εκτέλεση	38
2η εκτέλεση	30
3η εκτέλεση	34
4η εκτέλεση	34
5η εκτέλεση	37
6η εκτέλεση	28
7η εκτέλεση	26
8η εκτέλεση	28
9η εκτέλεση	35
10η εκτέλεση	38

Στο ερώτημα 3 της άσκησης 2, για να συγκρίνουμε την ταχύτητα σύγκλισης των τροποποιημένων μεθόδων σε σχέση με τις κλασικές μετράμε την διάρκεια εκτέλεσης της κάθε συνάρτησης. Πειραματικά διαπιστώνουμε πως:

- Η τροποποιημένη μέθοδος Newton-Raphson συγκλίνει (κατά μέσο όρο) γρηγορότερα σε σχέση με την κλασική
- Η κλασική μέθοδος διχοτόμησης συγκλίνει (κατά μέσο όρο) γρηγορότερα σε σχέση με την τροποποιημένη
- Η τροποποιημένη μέθοδος τέμνουσας συγκλίνει (κατά μέσο όρο) γρηγορότερα σε σχέση με την κλασική

Παρακάτω βλέπουμε το output από το τρίτο κομμάτι του script “askisi2.py”, το οποίο περιέχει τον αριθμό επαναλήψεων κάθε μεθόδου (κλασικής και τροποποιημένης) για μια συγκεκριμένη ρίζα και διάρκεια εκτέλεσης της:

```
Comparing the time elapsed for a specific root in classic and modified methods:
Number of iterations: 6
Time elapsed in classic Newton-Raphson method: 0.00012803077697753906
Number of iterations: 3
Time elapsed in modified Newton-Raphson method: 0.00003600120544433594

Number of iterations: 24
Time elapsed in classic bisection method: 0.00013279914855957031
Number of iterations: 34
Time elapsed in modified bisection method: 0.00019407272338867188

Number of iterations: 143
Time elapsed in classic secant method: 0.00145697593688964844
Number of iterations: 3
Time elapsed in modified secant method: 0.00006103515625000000
```

Άσκηση 3

Για το ερώτημα 1 της άσκησης 3, η συνάρτηση “luDecomposition()” του script “askisi3_1.py” δέχεται σαν είσοδο τον πίνακα A και το διάνυσμα B και επιστρέφει σαν έξοδο το διάνυσμα των αγνώστων x. Χρησιμοποιούμε μια “print()” για να εμφανίσουμε τα αποτελέσματα. Παρακάτω βλέπουμε το output από το script “askisi3_1.py” χρησιμοποιώντας ένα πίνακα A (περιγράφεται παρακάτω) ως παράδειγμα:

```
$ python3 askisi3_1.py
LU Decomposition method

A Matrix:
1       4       2
1       2       3
2       1       3

B Vector:
11      11      13

L Triangular:
1       0       0
1.0     1       0
2.0     3.5     1

U Triangular:
1       4       2
0       -2.0    1.0
0       0       -4.5

y Vector:
11.0    0.0     -9.0

Solution:
x0 = 3.0
x1 = 1.0
x2 = 2.0
```

Στο ερώτημα 2 της άσκησης 3, η ζητούμενη συνάρτηση “cholesky()” δέχεται έναν συμμετρικό και θετικά ορισμένο πίνακα A και επιστρέφει έναν κάτω τριγωνικό πίνακα L που αποτελεί την αποσύνθεση Cholesky του πίνακα A. Παρακάτω βλέπουμε το output από το script “askisi3_2.py”:

```
$ python3 askisi3_2.py
Cholesky Decomposition
A:
[[4, 12, -16], [12, 37, -43], [-16, -43, 98]]
L:
[[2.0, 0, 0], [6.0, 1.0, 0], [-8.0, 5.0, 3.0]]
```

Για το ερώτημα 3 της άσκησης 3, χρησιμοποιήθηκε η μέθοδος Gauss-Seidel για να επιλυθεί με ακρίβεια 4 δεκαδικών ψηφίων το αραιό σύστημα nxn, α) για n=10 και β) για n=10000. Παρατηρούμε πως για το πρώτο σύστημα με n=10 χρειάστηκαν 21 επαναλήψεις, ενώ για το δεύτερο σύστημα με n=10000 χρειάστηκαν 24 επαναλήψεις. Παρακάτω βλέπουμε το output από το script “askisi3_3.py”:

```
$ python3 askisi3_3.py
Gauss-Seidel method

Solution for 10x10 (n=10):
[0.9999574213657183, 0.9999369719607752, 0.9999320200289947, 0.9999368848502822, 0.
9999470557782718, 0.9999592201984544, 0.9999711615793162, 0.9999815797697306, 0.999
9898779630939, 0.9999959511852377]
Total iterations for 10x10 (n=10): 21

Solution for 10000x10000 (n=10000):
[0.9999694686699636, 0.9999492276998676, 0.9999358587477248, 0.9999270666527021, 0.
999921312402455, 0.999917566238629, 0.9999151412122895, 0.9999135808058044, 0.99991
2583015432, 0.999911949092305, 0.9999115489916142, 0.9999112981505689, 0.9999111419
42178, 0.999911045318755, 0.9999109859521657, 0.9999109497191977, 0.999910927750843
1, 0.9999109145177905, 0.9999109065976226, 0.9999109018871113, 0.9999108991028368,
0.9999108974670762, 0.999910896511761, 0.999910895957073, 0.9999108956368274, 0.999
9108954529591, 0.9999108953479621, 0.9999108952883204, 0.9999108952546166, 0.999910
8952356659, 0.9999108952250628, 0.9999108952191585, 0.9999108952158862, 0.999910895
```

…

```
0.9999108952119048, 0.9999108952119048, 0.9999108952119048, 0.9999108952119048, 0.9
999108959155922, 0.9999109025771666, 0.9999109353908938, 0.9999110473387045, 0.9999
113444483999, 0.9999119977814093, 0.9999132359430544, 0.9999153131802039, 0.9999184
583431695, 0.9999228187830089, 0.9999284162698847, 0.9999351282038444, 0.9999426990
021508, 0.9999507774219542, 0.9999589690796764, 0.9999668911897961, 0.9999742183069
253, 0.9999807121146347, 0.9999862332182605, 0.9999907369912817, 0.9999942580201097
, 0.9999968884974827, 0.9999987553989931]
Total iterations for 10000x10000: 24
```