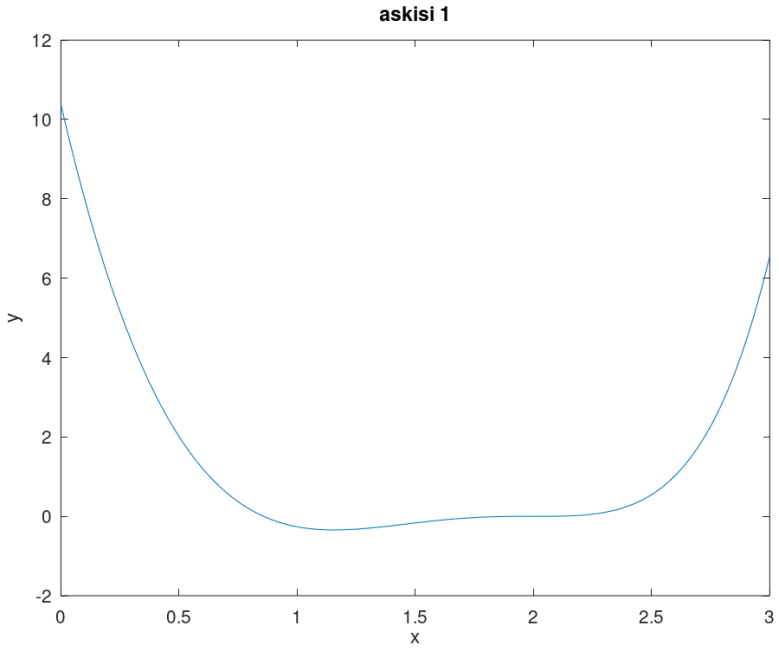
1η Υποχρεωτική Εργασία
Αριθμητική Ανάλυση
Εαρινό Εξάμηνο 2021-2022

Αλεξία Νταντουρή - 3871

29 Δεκεμβρίου 2021

1 Άσκηση 1

Αυτή είναι η γραφική παράσταση της συνάρτησης $f(x) = 14xe^{x-2} - 12e^{x-2} - 7x^3 + 20x^2 - 26x + 12$ στο διάστημα [0,3]:



Χωρίζω το διάστημα [0, 3] και καλώ τις μεθόδους για τα διαστήματα [0, 1.1] και [1.1, 3], καθώς ισχύει $f'(x), f''(x) \neq 0$ για $x < 1.1$.

1.1 Μέθοδος διχοτόμησης

[Αρχείο: **ask1Bisection.py**]
1η ρίζα = 0.85716 με 14 επαναλήψεις
2η ρίζα = 1.99996 με 15 επαναλήψεις

1.2 Newton-Raphson Method

[Αρχείο: **ask1NewtonRaphson.py**]
1η ρίζα = 0.85714 με 7 επαναλήψεις
2η ρίζα = 2.00006 με 24 επαναλήψεις

1.3 Μέθοδος τέμνουσας

[Αρχείο: **ask1Secant.py**]
1η ρίζα = 0.85714 με 10 επαναλήψεις
2η ρίζα = 2.00012 με 29 επαναλήψεις

1.4 Σύγκριση επαναλήψεων

Παρατηρούμε πως για την εύρεση της 1ης ρίζας, η μέθοδος Newton-Raphson θέλει τις λιγότερες επαναλήψεις, ενώ η μέθοδος της διχοτόμησης τις περισσότερες επαναλήψεις.
Για την εύρεση της 2ης ρίζας, η μέθοδος διχοτόμησης θέλει τις λιγότερες επαναλήψεις, ενώ η μέθοδος της τέμνουσας τις περισσότερες επαναλήψεις.

1.5 Τετραγωνική σύγκλιση

Για τη μέθοδο Newton-Raphson, η 1η ρίζα συγκλίνει τετραγωνικά, καθώς ισχύει $f(0.857) = 0$ και $f'(0.857) \neq 0$, ενώ η 2η ρίζα δεν συγκλίνει τετραγωνικά, καθώς ισχύει $f(2) = 0$ και $f'(2) = 0$.

2 Άσκηση 2

Τροποποιημένη μέθοδος Newton-Raphson: **ask2NewtonRaphsonModified.py**
Τροποποιημένη μέθοδος διχοτόμησης: **ask2BisectionModified.py**
Τροποποιημένη μέθοδος τέμνουσας: **ask2SecantModified.py**

Κλασική μέθοδος Newton-Raphson: **ask2NewtonRaphsonClassic.py**
Κλασική μέθοδος διχοτόμησης: **ask2BisectionClassic.py**
Κλασική μέθοδος τέμνουσας: **ask2SecantClassic.py**

2.1 Ρίζες συνάρτησης

Στον παρακάτω πίνακα φαίνονται οι ρίζες της εξίσωσης $f(x) = 54x^6 + 45x^5 - 102x^4 - 69x^3 + 35x^2 + 16x - 4$ με ακρίβεια 5ου δεκαδικού ψηφίου χρησιμοποιώντας τις τροποποιημένες μεθόδους, που είναι υλοποιημένες σε python.

	1η ρίζα	2η ρίζα	3η ρίζα	4η ρίζα	5η ρίζα
Τροπ μεθ Newton-Raphson	-1.38130	-0.66669	0.20518	0.50000	1.17612
Τροπ μεθ διχοτόμησης	-1.38128	no root	0.20515	0.49996	1.17617
Τροπ μεθ τέμνουσας	-1.38130	-0.66671	0.20518	0.50000	1.17612

Δεν μπορούμε να βρούμε την δεύτερη ρίζα με την τροποιημένη μέθοδο διχοτόμησης, καθώς ισχύει $f(x) \leq 0$ κοντά στη 2η ρίζα, με αποτέλεσμα να μην ισχύει το θεώρημα του Bolzano.

2.2 Αλγόριθμος (β)

Ο αλγόριθμος (β) δεν συγκλίνει πάντα σε ίδιο αριθμό επαναλήψεων.
Μέσος όρος επαναλήψεων για την εύρεση κάθε ρίζας:
1η ρίζα: 22.3 επαναλήψεις
2η ρίζα: -
3η ρίζα: 15.7 επαναλήψεις
4η ρίζα: 17.3 επαναλήψεις
5η ρίζα: 20 επαναλήψεις

2.3 Σύγκριση τροποιημένων μεθόδων σε σχέση με τις κλασικές ως προς την ταχύτητα σύγκλισης

Στους παρακάτω πίνακες παρουσιάζεται ο αριθμός των επαναλήψεων για την εύρεση της κάθε ρίζας με τις κλασικές και τις τροποποιημενες μεθόδους.

Μέθοδος Newton-Raphson	1η ρίζα	2η ρίζα	3η ρίζα	4η ρίζα	5η ρίζα
Τροποποιημένη	5	9	4	4	5
Κλασική	7	13	6	5	8

Μέθοδος διχοτόμησης	1η ρίζα	2η ρίζα	3η ρίζα	4η ρίζα	5η ρίζα
Τροποποιημένη	22.3	-	15.7	17.3	20
Κλασική	14	-	12	12	14

Μέθοδος τέμνουσας	1η ρίζα	2η ρίζα	3η ρίζα	4η ρίζα	5η ρίζα
Τροποποιημένη	11	2	4	6	9
Κλασική	18	17	6	7	13

Φαίνεται, παραπάνω, πως οι τροποιημένες μέθοδοι Newton-Raphson και τέμνουσας είναι πιο γρήγορες σε σχέση με τις κλασικές, σε αντίθεση με την τροποιημένη μέθοδο διχοτόμησης που είναι κατά μέσο όρο πιο αργή από την κλασική.

3 Άσκηση 3

3.1 PA=LU

[Αρχείο: **ask3PALU.py**]
Προγραμμάτισα σε python μία συνάρτηση **plu_solve(A, b)**, η οποία δέχεται ως ορίσματα έναν πίνακα **A** και ένα διάνυσμα **b** και επιστρέφει ως έξοδο το διάνυσμα των αγνώστων **x**. Η συνάρτηση **plu_solve(A, b)** καλεί την συνάρτηση **palu(A)**, η οποία δέχεται έναν πίνακα **A** κι επιστρέφει τους πίνακες P, L, U. Έπειτα, η συνάρτηση **plu_solve(A, b)** παράγει το διάνυσμα **x** καλώντας τη συνάρτηση **backward_substitution(U, y)** , αφότου έχει παράξει το διάνυσμα **y** καλώντας τη συνάρτηση **forward_substitution(L, b)**.
Ως είσοδο χρησιμοποίησα τα εξής:

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 5 \\ 0 \\ 6 \end{bmatrix} \tag{1}$$

Κι ως έξοδο, ο κώδικας παράγει τα εξής:

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad \mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ 0.25 & 1 & 0 \\ 0.5 & -0.5 & 1 \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} 4 & 4 & -4 \\ 0 & 2 & 2 \\ 0 & 0 & 8 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix} \tag{2}$$

3.2 Cholesky

[Αρχείο: **ask3Cholesky.py**]
Προγραμμάτισα σε python μία συνάρτηση **cholesky_decomposition(M)**, η οποία δέχεται ως όρισμα έναν συμμετρικό και θετικά ορισμένο πίνακα **M** και επιστρέφει έναν κάτω τριγωνικό πίνακα **L** που αποτελεί την αποσύνθεση Cholesky του πίνακα **A**.
Ως είσοδο χρησιμοποίησα τον πίνακα:

$$\mathbf{A} = \begin{bmatrix} 4 & -2 & 2 \\ -2 & 2 & -4 \\ 2 & -4 & 11 \end{bmatrix} \tag{3}$$

Κι ως έξοδο, ο κώδικας παράγει τον πίνακα:

$$\mathbf{L} = \begin{bmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -3 & 1 \end{bmatrix} \tag{4}$$

3.3 Gauss-Seidel

[Αρχείο: **ask3GaussSeidel.py**]
Προγραμμάτισα σε python μία συνάρτηση **gauss_seidel(A, b, x, xpre, n)**, η οποία επιλύει με ακρίβεια 4 δεκαδικών ψηφίων το $n \times n$ αραιό σύστημα $\mathbf{Ax} = \mathbf{b}$ της εκφώνησης για $n = 10$ και $n = 10000$. Ο κώδικας περιλαμβάνει και μία συνάρτηση **initialize_arrays(n)**, η οποία αρχικοποιεί τους πίνακες που χρειάζομαι για να λύσω το σύστημα.
Ως αποτέλεσμα ο κώδικας βρίσκει για το 1ο σύστημα ($n = 10$):
$[0.9996, 0.9994, 0.9994, 0.9994, 0.9995, 0.9996, 0.9998, 0.9999, 1.0, 1.0]$ σε 17 επαναλήψεις.
Και για το 2ο σύστημα ($n = 10000$):
$[0.9997, 0.9995, 0.9993, 0.9992, 0.9991, 0.9991, ..., 0.9999, 1.0, 1.0]$ σε 18 επαναλήψεις.

4 Άσκηση 4

4.1 Απόδειξη πίνακας στοχαστικός

Θα δείξω ότι κάθε στήλη του πίνακα Google έχει άθροισμα 1, δηλαδή για κάθε i ισχύει:

$$\sum_{j=1}^{n} G_{i,j} = 1$$

Αναλυτικά:

$$\sum_{j=1}^{n} G_{i,j} =$$

$$\sum_{j=1}^{n} \frac{q}{n} + \frac{A_{(j,i)}(1-q)}{N_j} =$$

$$n\frac{q}{n} + (1-q)\sum_{j=1}^{n} \frac{A_{(j,i)}}{N_j} =$$

$$q + (1-q) = 1$$

αφού προφανώς

$$\sum_{j=1}^{n} \frac{A_{(j,i)}}{N_j} = 1$$

4.2 Ιδιοδιάνυσμα μέγιστης ιδιοτιμής

[Αρχείο: **ask4_2.py**]
Υλοποίησα σε python τον αλγόριθμο για την κατασκευή του πίνακα **G** και την μέθοδο των δυνάμεων για τη δημιουργία του ιδιοδιανύσματος **p** της μέγιστης ιδιοτιμής. Στην παρακάτω εικόνα φαίνεται ο πίνακας **G** και το ιδιοδιάνυσμα **p**. Οι συνιστώσες του ιδιοδιανύσματος συμπίπτουν με αυτές τις εκφώνησης με ακρίβεια 4 δεκαδικών ψηφίων.

```
C:\Users\alexi\PycharmProjects\power_method\venv\Scripts\python.exe C:/Users/alexi/PycharmProjects/power_method/main.py
N =
2 3 3 2 2 2 2 2 3 1 1 3 2 4 2
G =
0.010000 0.010000 0.010000 0.010000 0.435000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000
0.435000 0.010000 0.293333 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000
0.010000 0.293333 0.010000 0.435000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000
0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.435000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000
0.010000 0.293333 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.293333 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000
0.010000 0.010000 0.293333 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.293333 0.010000 0.010000 0.010000 0.010000 0.010000
0.010000 0.293333 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.293333 0.010000 0.010000 0.010000
0.010000 0.010000 0.293333 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.293333 0.010000 0.010000 0.010000
0.435000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.435000 0.010000 0.010000
0.010000 0.010000 0.010000 0.010000 0.435000 0.435000 0.435000 0.010000 0.293333 0.010000 0.010000 0.010000 0.010000 0.222500 0.010000
0.010000 0.010000 0.010000 0.010000 0.010000 0.435000 0.435000 0.435000 0.010000 0.010000 0.010000 0.293333 0.010000 0.222500 0.010000
0.010000 0.010000 0.010000 0.435000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.435000
0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.860000 0.010000 0.010000 0.010000 0.222500 0.010000
0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.435000 0.010000 0.435000
0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.010000 0.860000 0.010000 0.010000 0.222500 0.010000

sum of columns =
1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000
p =
0.0268309 0.0298596 0.0298596 0.0268309 0.0395832 0.0395832 0.0395832 0.0395832 0.0745504 0.1063334 0.1063334 0.0745504 0.1251087 0.1163013 0.1251087

Process finished with exit code 0
```

4.3 Βελτίωση βαθμού σημαντικότητας μιας σελίδας

[Αρχείο: **ask4_3.py**]
Επέλεξα να βελτιώσω τον βαθμό σημαντικότητας της σελίδας 5. Για να το πετύχω αυτό πρόσθεσα τις εξής συνδέσεις:
$1 \to 5$
$6 \to 5$
$10 \to 5$
$13 \to 5$
και αφαίρεσα την σύνδεση:
$2 \to 5$
Το αποτέλεσμα αυτών των αλλαγών είναι να αυξηθεί η τάξη της σελίδας 5 από 0.0395832 σε 0.1269914. Μάλιστα, η τάξη αυτής της σελίδας είναι πλέον η μέγιστη μεταξύ όλων των τάξεων των σελίδων.

4.4 Αλλαγή πιθανότητας μεταπήδησης στο νέο γράφο

Για το νέο γράφο που δημιουργείται, η τάξη της σελίδας 5 είναι 0.1269914 με πιθανότητα μεταπήδησης $q = 0.15$. Αν αλλάξουμε την πιθανότητα μεταπήδησης σε $q = 0.02$ η τάξη της σελίδας 5 γίνεται 0.1498330, ενώ αν αλλάξουμε την πιθανότητα μεταπήδησης σε $q = 0.6$ η τάξη της σελίδας 5 γίνεται 0.0895668.

q	τάξη σελίδας
0.02	0.1498330
0.15	0.1269914
0.60	0.0895668

Φαίνεται, λοιπόν, πως όταν αυξάνεται η πιθανότητα μεταπηδησης (αύξηση q) μειώνεται η τάξη της σελίδας. Δηλαδή, η αύξηση της πιθανότητας μεταπήδησης μειώνει την επιρροή των διασυνδέσεων που υπάρχουν μεταξύ των κόμβων.

4.5 Βελτίωση τάξης της ιστοσελίδας 11

[Αρχείο: **ask4_5.py**]
Αν αντικαταστήσουμε το $A_{(8,11)}$ και το $A_{(12,11)}$ με 3, η τάξη της σελίδας 11 αυξάνεται από 0.1063334 σε 0.1239842, ενώ η τάξη της σελίδας 10 μειώνεται από 0.1063334 σε 0.1028875. Οπότε, αυτή η στρατηγική δουλεύει.

4.6 Επίδραση της διαγραφής της σελίδας 10 από το γράφημα

[Αρχείο: **ask4_6.py**]
Στον παρακάτω φαίνεται η τάξη των σελίδων πριν και μετά τη διαγραφή της σελίδας 10.

q	τάξη σελίδας πριν τη διαγραφή της 10	τάξη σελίδας μετά τη διαγραφή της 10	μεταβολή τάξης
1	0.0268309	0.0471132	αύξηση
2	0.0298596	0.0409539	αύξηση
3	0.0298596	0.0359364	αύξηση
4	0.0268309	0.0320622	αύξηση
5	0.0395832	0.0428133	αύξηση
6	0.0395832	0.0413978	αύξηση
7	0.0395832	0.0516734	αύξηση
8	0.0395832	0.0502580	αύξηση
9	0.0745504	0.0482447	μείωση
10	0.1063334	-	-
11	0.1063334	0.1709568	αύξηση
12	0.0745504	0.1035727	αύξηση
13	0.1251087	0.0411576	μείωση
14	0.1163013	0.1074311	μείωση
15	0.1251087	0.1864288	αύξηση

Συνολικά, παρατηρούμε πως μειώθηκε η τάξη των σελίδων 9, 13 και 14, ενώ η τάξη των υπόλοιπων σελίδων αυξήθηκαν.
Η διαγραφή της σελίδας 10 μειώνει σημαντικά την τάξη της σελίδας 13 (από 0.1251087 σε 0.0411576), καθώς ο κόμβος 13 δεν είχε μεγάλο αριθμό εισόδου, αλλά ο κόμβος 10, που είχε τον μεγαλύτερο αριθμό εισόδου μετέφερε τη σημαντικότητά του στον κόμβο 13, καθώς έδειχνε σε αυτόν. Η μείωση της τάξης της σελίδας 13 μειώνει και την τάξη των σελίδων 9 και 14 προς τις οποίες δείχνει.