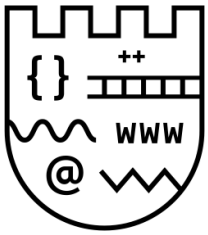


Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Τμήμα Πληροφορικής

Αριθμητική ανάλυσή

1η υποχρεωτική εργασία

Συντάκτης:
Αλέξανδρος Κόρκος

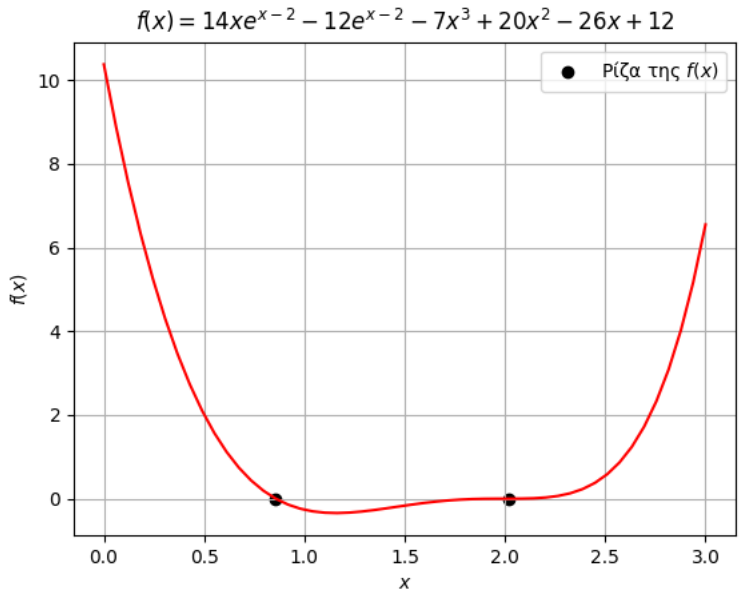
Καθηγητής:
Τέφας Αναστάσιος

ΑΚΑΔΗΜΑΪΚΌ ΈΤΟΣ 2021/22

Περιεχόμενα

Άσκηση 1

Η γραφική παράσταση της συνάρτησης $f(x)$ στο διάστημα $[0, 3]$ έχει την παρακάτω μορφή.



Σχήμα 1: Η γραφική παράσταση της $f(x)$

Όπως φανερώνει και το παραπάνω γράφημα, η συνάρτηση φαίνεται να έχει δυο ρίζες: μια για $r_1 \approx 0.85714$ και μια για $r_2 \approx 2.00001$.

Μέθοδος διχοτόμισης

Για τον υπολογισμό στης συνάρτησης σύμφωνα με την μέθοδο της διχοτόμησης, χρησιμοποιήθηκε ο παρακάτω κώδικας. Όπου f η συνάρτηση, a και b τα άκρα του διαστήματος.

Μέθοδος της διχοτόμισης:

```
1 def partition(f, a, b):
2     n = 1
```

```
    if np.sign(f(a)) * np.sign(f(b)) >= 0:
        return
    while (b - a) * .5 > e:
        m = (b + a) * .5
        if abs(f(m)) < e:
            break
        if np.sign(f(a)) * np.sign(f(m)) < 0:
            b = m
        else:
            a = m
        n += 1
    return m
```

Για να επιτευχθεί η εύρεση και των δυο ριζών της συνάρτησης, θα χωριστεί το αρχικό διάστημα σε $[0, 1]$ και $[1.5, 3]$. Στην πρώτη περίπτωση υπολογίζεται η ρίζα $r_1 = 0.85713$ μετά από 19 επαναλήψεις και στην δεύτερη η $r_2 = 1.99218$ με 6 επαναλήψεις. Η δεύτερη ρίζα απέχει αρκετά από την προσέγγιση με την μέθοδο αυτή.

Μέθοδος Newton-Raphson

Για τον υπολογισμό στης συνάρτησης σύμφωνα με την μέθοδο Newton-Raphson, χρησιμοποιήθηκε ο παρακάτω κώδικας. Όπου f η συνάρτηση, df η πρώτη παράγωγος της συνάρτησης και x0 μια αρχική προσέγγιση της ρίζας.

Newton-Raphson:

```
def newton(f, df, x0):
    n = 1
    xi = x0
    while df(xi) != 0:
        xi_1 = xi - f(xi) / df(xi)
        if abs(f(xi_1) / df(xi_1)) < e:
            break
        n += 1
        xi = xi_1
    return xi
```

Ο παραπάνω κώδικας, θα εκτελεστεί δυο φορές. Έτσι για $x_0 = 0$ ο αλγόριθμος θα επιστρέψει $r_1 = 0.85704$ με τον αριθμό των επαναλήψεων να ανέρχεται σε 6. Για $x_1 = 3$ ο αλγόριθμος επιστρέφει $r_2 = 2.00002$ με 28 επαναλήψεις.

Για να ελεγχθεί εάν η ρίζα συγκλίνει τετραγωνικά, θα χρησιμοποιηθεί ο παρακάτω αλγόριθμος, που κάνει εφαρμογή του εξής θεωρήματος: «*... αν $f'(\xi) \neq 0$ η τάξη σύγκλισής της μεθόδου Newton-Raphson είναι τετραγωνική.*»

Τετραγωνική σύγκλιση:

```
def converge(x):
    x = round(x, 2)
    if df(x) != 0:
        return True
    return False
```

Εκτελώντας τον παραπάνω αλγόριθμο για r_1 και r_2 συμπεραίνετε πως, για r_1 υπάρχει τετραγωνική σύγκλιση ενώ για r_2 δεν υπάρχει.

Το χαρακτηριστικό των ριζών που δεν συγκλίνουν τετραγωνικά είναι πως μηδενίζουν την πρώτη παράγωγο της συνάρτησης στην ρίζα που βρίσκει η μέθοδος.

Μέθοδος της τέμνουσας

Για τον υπολογισμό στης συνάρτησης σύμφωνα με την μέθοδο της τέμνουσας, χρησιμοποιήθηκε ο παρακάτω κώδικας. ΄Οπου f η συνάρτηση, x0 και x1 αρχικές προσεγγίσεις της ρίζας.

Μέθοδος της τέμνουσας:

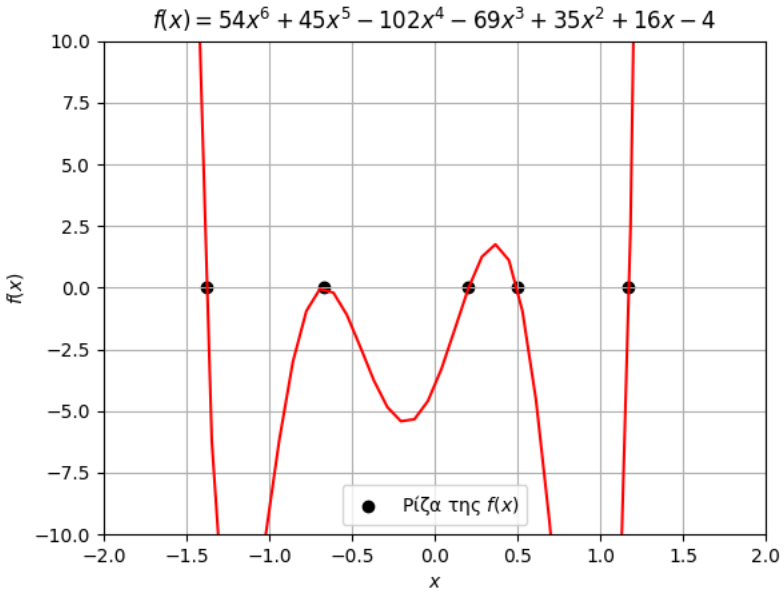
```
def secant(f, x0, x1):
    n = 1
    xi = x1
    xi_1 = x0
    while f(xi) - f(xi_1) != 0 and abs(f(xi)) >= e:
        xi1 = xi - (f(xi) * (xi - xi_1)) / (f(xi) - f(xi_1))
        xi_1 = xi
        xi = xi1
        n += 1
    return x
```

Για να βρεθούν και οι δυο ρίζες της συνάρτησης, θα χρειαστεί ο αλγόριθμος να τρέξει δυο φορές για διαφορετικά x_0 και x_1. Στην πρώτη εκτέλεση, με $x_0 = 0$ και $x_1 = 1$ θα επιστραφεί η $r_1 = 0.85714$ μετά από 7 επαναλήψεις. Για την δεύτερη, με $x_0 = 0$ και $x_1 = 2.5$ θα επιστραφεί η $r_2 = 2.01001$ μετά από 17 επαναλήψεις.

Για τον υπολογισμό της r_2 μπορούμε να κάνουμε χρήση και κάποιον πιο προσεγγιστικών αρχικών τιμών (π.χ. $x_0 = 1.5, x_1 = 2.5$) χάνοντας έτσι αρκετά ψηφία στην πραγματική ρίζα.

Άσκηση 2

Πριν περιγράφουν οι τροποποιημένες μορφές των προηγουμένων μεθόδων, είναι απαραίτητο να υπάρξει μια οπτικοποίηση για το που βρίσκονται οι ρίζες της συνάρτησης.



Σχήμα 2: Η γραφική παράσταση της $f(x)$

Από το σχήμα 2 εντοπίζονται πέντε ρίζες. Συγκεκριμένα $r_1 \approx -1.3813, r_2 \approx -0.66667, r_3 \approx 0.2052, r_4 \approx 0.5$ και $r_5 \approx 1.17611$.

Τροποποιημένη μέθοδος Newton-Raphson

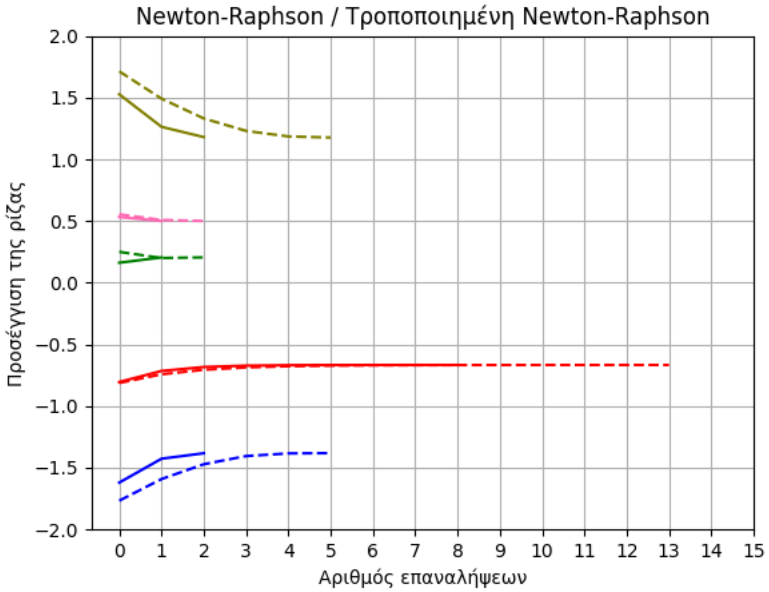
Για την υλοποίηση της τροποποιημένης μεθόδου Newton-Raphson χρησιμοποιήθηκε ο εξής αλγόριθμος. Όπου f η συνάρτηση, df η πρώτη παράγωγος της συνάρτησης, d2f η δεύτερη παράγωγος της συνάρτησης και x0 μια αρχική προσέγγιση της ρίζας.

Τροποποιημένη Newton-Raphson:

```
1  def newton(f, df, d2f, x0):
2      n = 1
3      xi = x0
4      while True:
5          xi_1 = xi- (1 / ((df(xi) / f(xi)) - (d2f(xi) / (2 *
               df(xi)))))
6          if abs(f(xi_1) / df(xi_1)) < e:
7              break
8          n += 1
9          xi = xi_1
10
11     return xi
```

Για $x_0 = -2$ προκύπτει $r_1 = -1.38129$, $x_0 = -1$ προκύπτει $r_2 = -0.66688$, $x_0 = 0$ προκύπτει $r_3 = 0.20518$, $x_0 = 0.7$ προκύπτει $r_4 = 0.5$ και για $x_0 = -2$ επιστρέφεται $r_5 = 1.17611$.

Εάν συγκρίνουμε την κλασσική μέθοδο με την τροποποιημένη, προκύπτει το εξής σχήμα.



Σχήμα 3: Ταχύτητα σύγκλισης Newton-Raphson

Όπως φαίνεται από το σχήμα 3, η κλασσική μέθοδος (διακεκομμένη γραμμή) φαίνεται σε όλες τις περιπτώσεις να χρειάζεται περισσότερες επαναλήψεις για να φτάσει προσεγγίσει την ρίζα με την κατάλληλη ακρίβεια έναντι της τροποποιημένης (συνεχόμενη γραμμή). Ενώ σε τρεις περιπτώσεις οι δυο μέθοδοι φαίνεται να συγκλίνουν σχεδόν στους ίδιους αριθμούς.

Τροποποιημένη μέθοδος διχοτόμησης

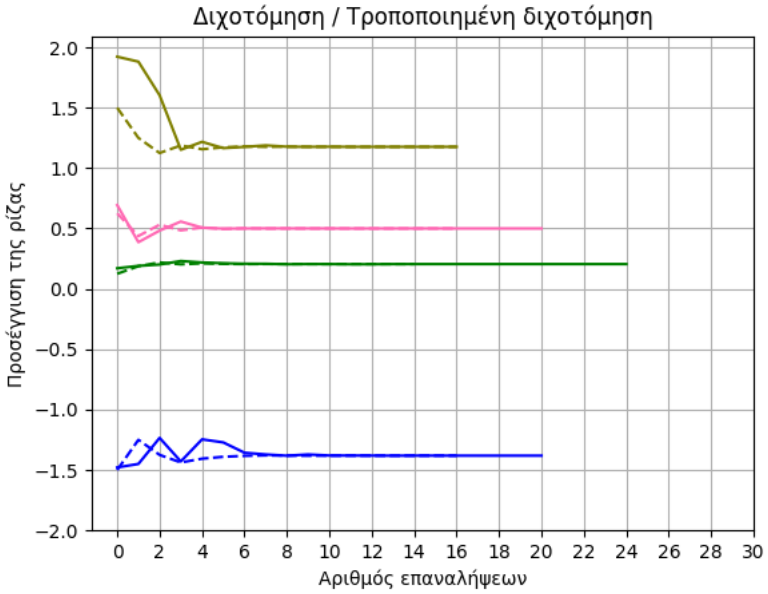
Για την υλοποίηση της τροποποιημένης μεθόδου διχοτόμησης ακολουθείτε η παρακάτω υλοποίηση.

Τροποποιημένη διχοτόμηση:

```
def partition(f, a, b):
    n = 1
    if np.sign(f(a)) * np.sign(f(b)) >= 0:
        return
    while (b - a) * .5 > e:
        m = uniform(a, b)
        if abs(f(m)) < e:
            break
        if np.sign(f(a)) * np.sign(f(m)) < 0:
            b = m
        else:
            a = m
        n += 1
    return m
```

Για $[-2, -1]$ προσδιορίζεται $r_1 = -1.38130$, $[0, 0.25]$ προσδιορίζεται $r_2 = 0.20518$, $[0.25, 0.75]$ προσδιορίζεται $r_3 = 0.5$ και τέλος για $[-2, -1]$ προσδιορίζεται $r_4 = 1.17611$.

Εάν συγκρίνουμε την κλασσική μέθοδο με την τροποποιημένη, προκύπτει το εξής σχήμα. Όπου f η συνάρτηση, a και b τα άκρα του διαστήματος.

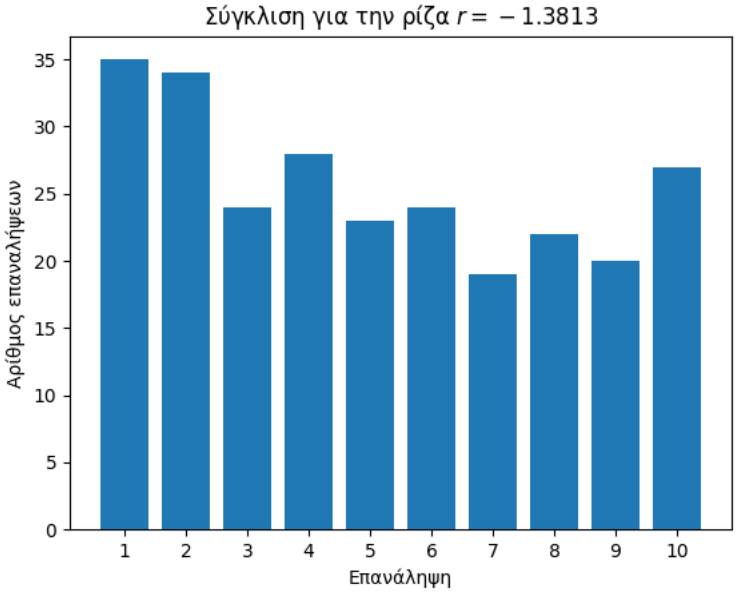


Σχήμα 4: Ταχύτητα σύγκλισης διχοτόμησης

Σε όλες τι περιπτώσεις η τροποποιημένη μέθοδος (συνεχόμενη γραμμή) χρειάζεται παραπάνω επαναλήψεις από την κλασσική, το οποίο ήταν αναμενόμενο μιας και κάθε φορά δεν χρησιμοποιείται το μέσο άλλα κάποιος τυχαίος αριθμός ανάμεσα στα δυο άκρα. Επιπλέον και οι δυο μέθοδοι συγκλίνουν περίπου στους ίδιους αριθμούς.

Δεν μπορεί με καμία από τις δυο μεθόδους να υπολογιστεί η ρίζα $r_3 = -0.6667$ διότι δεν ισχύει το θεώρημα Bolzano για αντίστοιχο διάστημα.

Μέσω του επόμενου γραφήματος, μπορούμε να δούμε τον αριθμό των επαναλήψεων που χρειάζεται η μέθοδος για να πετύχει την σύγκλιση στην ρίζα $r_1 = -1.3813$ εξετάζοντας δέκα διαφορετικές φορές.



Σχήμα 5: Σύγκλιση / επανάληψη

Όπως γίνετε αντιληπτό, επειδή η τροποποιημένη διχοτόμηση βασίζεται στην τυχαιότητα, ο αριθμός των επαναλήψεων για την προσέγγιση της ρίζας ποικίλει κάθε φορά.

Τροποποιημένη μέθοδος τέμνουσας

Το παρακάτω απόσωσμα κώδικα υπολογίζει την τροποποιημένη μέθοδο της τέμνουσας. Όπου f η συνάρτηση, x0, x1 και x2 αρχικές προσεγγίσεις της ρίζας.

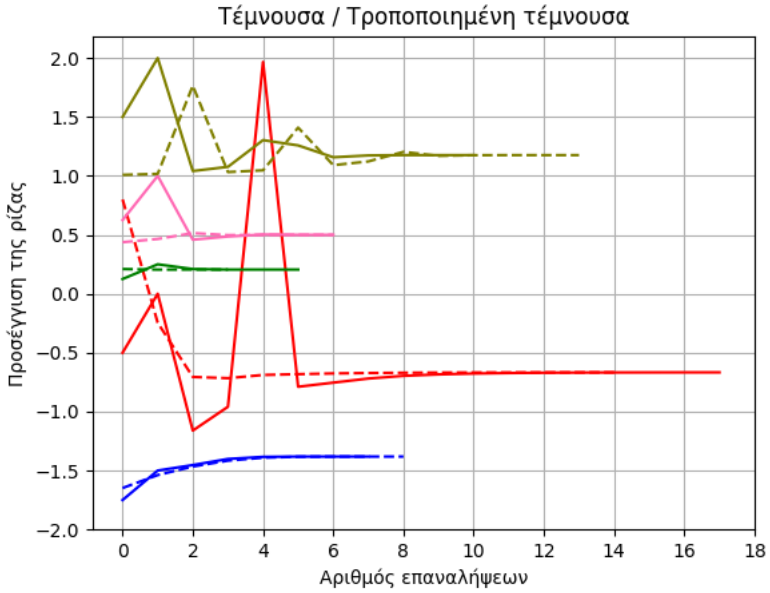
Τροποποιημένη τέμνουσα:

```
1  def secant(f, x0, x1, x2):
2      xi = x0
3      xi1 = x1
```

```
4       xi2 = x2
5       n = 1
6       while abs(f(xi)) > e:
7           q = f(xi) / f(xi1)
8           r = f(xi2) / f(xi1)
9           s = f(xi2) / f(xi)
10          xi3 = xi2 - (r * (r - q) * (xi2 - xi1) + (1 - r) * s *
                (xi2 - xi)) / ((q - 1) * (r - 1) * (s - 1))
11          xi = xi1
12          xi1 = xi2
13          xi2 = xi3
14          n += 1
15      return xi
```

Μετά το τέλος εκτέλεσης του κώδικά προκύπτουν τα εξής αποτελέσματά. Για $x_0 = -2, x_1 = -1.75, x_2 = -1.5$ προσδιορίζεται $r_1 = -1.38129$, $x_0 = -1, x_1 = -0.5, x_2 = 0$ προσδιορίζεται $r_2 = -0.66688$, $x_0 = 0, x_1 = 0.125, x_2 = 0.25$ προσδιορίζεται $r_3 = 0.20518$, $x_0 = 0.4, x_1 = -.625, x_2 = 1$ προσδιορίζεται $r_3 = 0.5$ και τέλος για $x_0 = 1, x_1 = 1.5, x_2 = 2$ προσδιορίζεται $r_3 = 1.17611$.

Εάν συγκρίνουμε την κλασσική μέθοδο με την τροποποιημένη, προκύπτει το εξής σχήμα.



Σχήμα 6: Ταχύτητα σύγκλισης τέμνουσας

Από το σχήμα 6 απορρέει πως οι δυο μέθοδοι συγκλίνουν σχεδόν το ίδιο γρήγορα σε κάθε περίπτωση. Το μόνο διαφορετικό που παρατηρείται είναι πως στις αρχικές επαναλήψεις οι προσεγγίσεις διαφέρουν αρκετά.

Άσκηση 3

Μέθοδος PA = LU

Κάθε τετραγωνικός πίνακας A μπορεί να αναλυθεί σε γινόμενο ενός πίνακα αντιμετάθεσης P με τον πίνακά A που ισούται με το γινόμενο ενός κάτω τριγωνικού πίνακα L με μονάδες στην διαγώνιο και ενός άνω τριγωνικού πίνακα U.

- Ο $\mathbf{P}$ καθορίζεται από τις εναλλαγές γραμμών που απαιτεί η διαδικασία της απαλοιφής με οδήγηση.
- Ο $\mathbf{L}$ έχει τους πολλαπλασιαστές της απαλοιφής κάτω από την διαγώνιο.
- Ο $\mathbf{U}$ τα στοιχεία του A όπως αυτά προκύπτουν μετά την απαλοιφή.

Σύμφωνα με τα παραπάνω θα χρησιμοποιηθεί ο παρακάτω κώδικας για τον υπολογισμό των πινάκων $\mathbf{U}, \mathbf{L}, \mathbf{P}$.

Αποσύνθεση:

```
def decomposition(A, b):
    U = np.concatenate((A,b), axis = 1)
    n = A.shape[0]
    L = np.identity(n)
    P = np.identity(n)
    for i in range(n):
        pivotRow = i + np.argmax(abs(U[i:, i]))
        U[[i, pivotRow]] = U[[pivotRow, i]]
        P[[i, pivotRow]] = P[[pivotRow, i]]
        for j in range(i + 1, n):
            m = U[j, i] / U[i, i]
            U[j] = U[j] - m * U[i]
            L[j, i] = m
    return U
```

Για ελαχιστοποίηση της ταχύτητας επίλυσης του γραμμικού συστήματος, στο πρόγραμμα που αναπτύχθηκε αποφεύγεται η εμπρός αντικατάσταση καθώς αντί να χρησιμοποιηθεί ο πίνακας $\mathbf{A}$ χρησιμοποιείται ο $[\mathbf{A}|\mathbf{b}]$. Έτσι το μόνο που

χρειάζεται να γίνει είναι μια απλή πίσω αντικατάσταση για να υπολογιστεί το διάνυσμα $\mathbf{x}$, όπως φαίνεται και παρακάτω.

Πίσω αντικατάσταση:

```
1  def backSubstitution(U, b):
2      n = b.size
3      x = np.zeros(n)
4      b = U[:, n]
5      U = U[:, : n]
6      n = U.shape[0]
7      for i in range(n-1, -1, -1):
8          temp = b[i]
9          for j in range(n-1, i, -1):
10             temp -= x[j] * U[i, j]
11         x[i] = temp / U[i, i]
12     return x
```

Αποσύνθεση Cholesky

Η αποσύνθεσή Cholesky ορίζεται ως:

$$l_{ki} = \begin{cases} \frac{1}{l_{ii}}(a_{ki} - \sum_{j=1}^{i-1} l_{ij}l_{kj}) & \text{για } k \neq i \\ \sqrt{a_{ki} - \sum_{j=1}^{k-1} l_{ij}l_{kj}} & \text{για } k = i. \end{cases} \quad k = 1, 2, ..., n$$

Cholesky:

```
1  def cholesky(A):
2      (n, m) = A.shape
3      L = np.zeros((n, m))
4      for k in range(n):
5          for i in range(k + 1):
6              sum = 0
7              for j in range(i):
8                  sum += L[i, j] * L[k, j]
9              if k == i:
10                 L[k, i] = np.sqrt(A[k, k] - sum)
11             else:
12                 L[k, i] = (A[k, i] - sum) / L[i, i]
13     return L
```

Στην σειρά 8 υπολογίζεται το άθροισμα $\sum_{j=1}^{i-1} l_{ij}l_{kj}$, ενώ στην σειρά 10 εφαρμόζεται ο τύπος $\sqrt{a_{ki} - \sum_{j=1}^{k-1} l_{ij}l_{kj}}$ και στην σειρά 12 ο τύπος $\frac{1}{l_{ii}}(a_{ki} - \sum_{j=1}^{i-1} l_{ij}l_{kj})$.

Μέθοδος Gauss-Seidel

Η μέθοδος Gauss-Seidel ορίζεται ως: $x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}),\ i = 1, 2, ..., n, k = 0, 1, 2, ...$ Επιπλέον χρειάζεται να οριστεί και ένα διάνυσμα $x_i^{(0)}, i = 1, 2, ..., n$ που περιέχει κάποιες αρχικές τιμές και αποτελεί κάποιες αρχικές προσεγγίσεις των τιμών του x_i.

Gauss-Seidel:

```
1  def gaussSeidel(A, x, b):
2      while True:
3          X0 = np.copy(x)
4          for i in range(n):
5              sum = 0.0
6              for j in range(n):
7                  if i != j:
8                      sum = sum + A[i, j] * x[j]
9              x[i] = (b[i] - sum) / A[i, i]
10         norm = 0.0
11         for i in range(n):
12             norm += abs(x[i] - X0[i])
13         if norm < e:
14             break
15     return x
```

Στην σειρά 8 υπολογίζεται $\sum_{j=i+1}^{n} a_{ij}x_j^{(k)}$ χρησιμοποιώντας την τρέχουσα προσέγγιση του αντίστοιχου x_j.

Αφού γίνει αυτό ενημερώνεται η τιμή του x_j για να χρησιμοποιηθεί στην επομένη επανάληψη (έτσι χρησιμοποιείται κάθε φορά η καλύτερη προσέγγιση).

Στις σειρές 11-12 υπολογίζεται η άπειρη νόρμα από τον τύπο $||x^{(k)} - x^{(k-1)}||$ όπου το $x^{(k)}$ αποθηκεύεται στον πίνακα x και $x^{(k-1)}$ στον πίνακα X0.

Άσκηση 4

Ζητούμενο 1

Στοχαστικός πίνακας ονομάζεται ένας τετραγωνικός πίνακας με μη αρνητικά στοιχεία τα οποία αναπαριστούν πιθανότητες. Οι στήλες (ή οι γραμμές) έχουν άθροισμα 1. Σύμφωνα με αυτόν τον ορισμό λειτουργεί και ο παρακάτω αλγόριθμος.

```
1  def isStochastic(A):
2      stochastic = True
3      sumOfCols = A.sum(axis=0)
4      for i in sumOfCols:
5          if round(i, 2) != 1:
6              stochastic = False
7      return stochastic
```

Εκτελώντας λοιπόν τον αλγόριθμο στον $\mathbf{G}$ αποδεικνύεται πως είναι στοχαστικός.

Ζητούμενο 2

Για την κατασκευαστεί του πίνακα $\mathbf{G}$ χρησιμοποιήθηκε ο εξής τύπος: $\mathbf{G}_{(i,j)} = \frac{q}{n} + \frac{\mathbf{A}_{(i,j)}(1-q)}{n_j}$, που υλοποιείται από τον παρακάτω κώδικα.

```
1  def createGoogleMatrix(A, q):
2      n = A.shape[0]
3      ni = []
4      G = np.zeros((n, n))
5      for i in range(n):
6          ni.append(sum(A[i, :]))
7
8      for i in range(n):
9          for j in range(n):
10             G[i, j] = (q / n) + ((A[j, i] * (1 - q)) / ni[j])
```

```
11
12      return G
```

Ο πίνακας $\mathbf{G}$ που υπολογίστηκε έχει ως εξής:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.0100	0.0100	0.0100	0.0100	0.4350	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100
0.4350	0.0100	0.2933	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100
0.0100	0.2933	0.0100	0.4350	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100
0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.4350	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100
0.0100	0.2933	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.2933	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100
0.0100	0.0100	0.2933	0.0100	0.0100	0.0100	0.0100	0.0100	0.2933	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100
0.0100	0.2933	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.2933	0.0100	0.0100	0.0100
0.0100	0.0100	0.2933	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.2933	0.0100	0.0100	0.0100
0.4350	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.4350	0.0100	0.0100
0.0100	0.0100	0.0100	0.0100	0.4350	0.4350	0.4350	0.0100	0.2933	0.0100	0.0100	0.0100	0.0100	0.2225	0.0100
0.0100	0.0100	0.0100	0.0100	0.0100	0.4350	0.4350	0.4350	0.0100	0.0100	0.0100	0.2933	0.0100	0.2225	0.0100
0.0100	0.0100	0.0100	0.4350	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.4350
0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.8600	0.0100	0.0100	0.0100	0.2225	0.0100
0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.4350	0.0100	0.4350
0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.8600	0.0100	0.0100	0.2225	0.0100

Πίνακας 1: Ο πίνακας $\mathbf{G}$

Στην συνέχεια για να βρεθεί το ιδιοδίανυσμα της μέγιστης ιδιοτιμής, υλοποιήθηκε η μέθοδος των δυνάμεων, σύμφωνα με το απόσπασμα που ακολουθεί.

```
1  def powerMethod(A):
2      x = np.empty(A.shape[0])
3      x.fill(1)
4      for _ in range(10000):
5          x = np.dot(A, x)
6          x = x / np.linalg.norm(x, 1)
7      return x
```

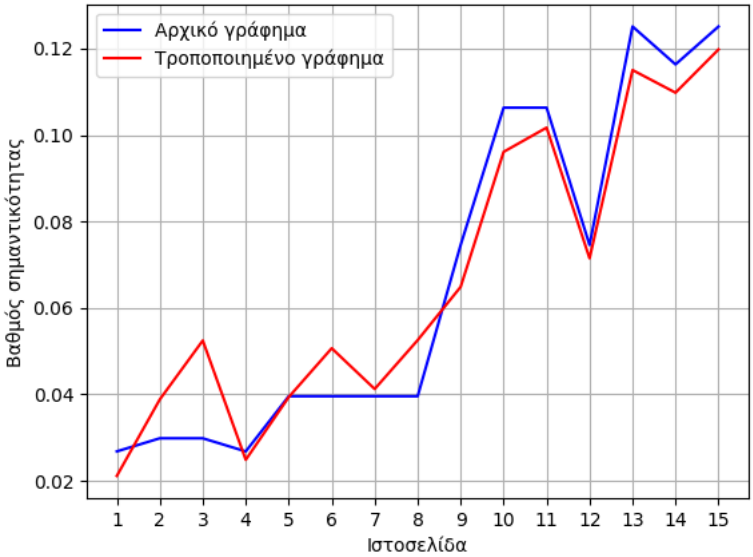
Ο πίνακας $\mathbf{p}$ που υπολογίστηκε:

$$\begin{pmatrix} 0.02682 \\ 0.02986 \\ 0.02986 \\ 0.02682 \\ 0.03959 \\ 0.03959 \\ 0.03959 \\ 0.03959 \\ 0.07456 \\ 0.10632 \\ 0.10632 \\ 0.07456 \\ 0.12509 \\ 0.11633 \\ 0.12509 \end{pmatrix}$$

Όπου ταυτίζεται με τον πίνακα $\mathbf{p}$ που δίνεται στην εκφώνηση.

Ζητούμενο 3

Στοχεύετε η αύξηση σημαντικότητάς του κόμβου 3. Διαγράφετε η σύνδεση του κόμβου 3 προς 2. Προσθέτεται η σύνδεση από το 5 στο 2, από 7 στο 2, από 8 προς 3 και 1 προς 3.

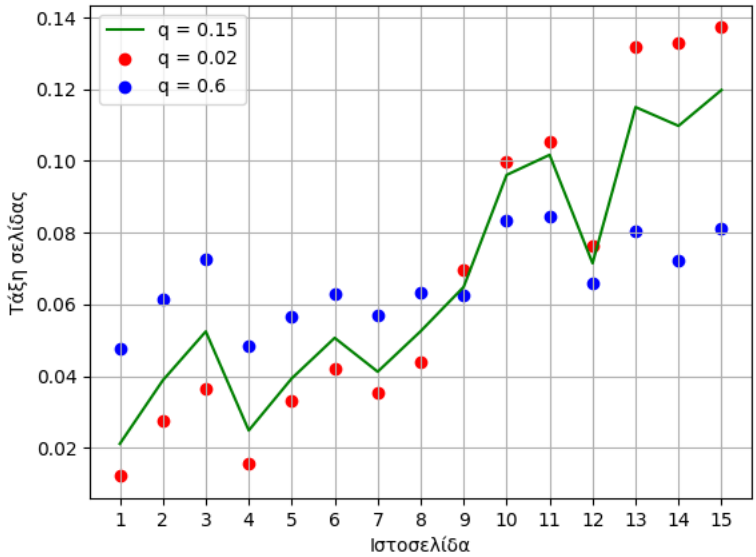


Σχήμα 7: Ζητούμενο 3

Όπως φαίνεται από το σχήμα 7 η τάξη σημαντικότητας του κόμβου που θέλαμε να αυξήσουμε πέτυχε. Επιπλέον αυξήθηκε και η τάξη του κόμβου 2, μιας και απέκτησε δυο νέες σελίδες

Ζητούμενο 4

Αλλάζοντάς τις τιμές του q για τον γράφου του παραπάνω ζητούμενου, παρατηρούνται οι εξής αλλαγές.



Σχήμα 8: Ζητούμενο 4

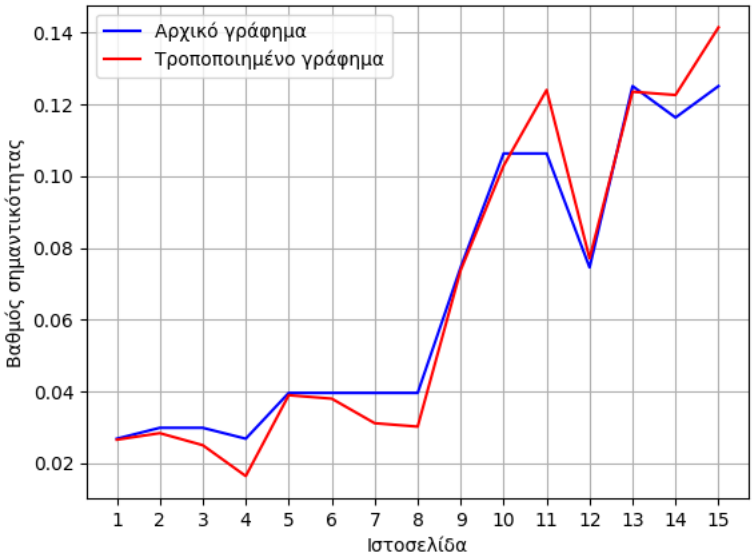
Εξετάζοντας λοιπόν τις μεταβολές, συμπεραίνετε:

- Για $q = 0.02$: Οι σελίδες που είχαν σχετικά χαμηλή τάξη (κάτω από 0.06) φαίνεται να μειώνεται η τάξη τους, ενώ οι υψηλές σελίδες (πάνω από 0.06) κερδίζουν από την αλλαγή αυτή.
- Για $q = 0.6$: Εδώ παρατηρείται το αντίθετο φαινόμενο. Οι σελίδες με χαμηλή τάξη (κάτω από 0.06) κερδίζουν σε άξια, αντίθετα οι υψηλές (πάνω από 0.06) χάνουν σε τάξη.

Η πιθανότητα αναπήδησης είναι ένας αριθμός που καθορίζει την πιθανότητα κάποιος χρήστης να μεταβεί από μια σελίδα σε κάποια άλλη.

Ζητούμενο 5

Όπως μπορούμε να δούμε από τον πίνακα **??** οι σελίδες 11 και 12 έχουνε την ίδια σημαντικότητα. Αλλάζοντας λοιπόν τις τιμές στο πίνακα γειτνίασης 1, προκύπτουν οι μεταβολές παρακάτω.

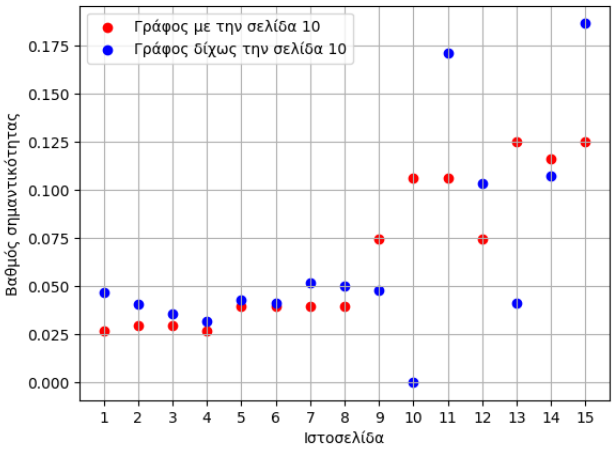


Σχήμα 9: Ζητούμενο 5

Από το παραπάνω σχήμα 9 συμπεραίνουμε πως η στρατηγική πέτυχε, δίνοντας της σελίδας 11 μια αύξηση της τάξεως $\approx 17\%$.

Ζητούμενο 6

Με την αφαίρεση της σελίδας 10 προκύπτουν οι εξής αλλαγές.



Σχήμα 10: Ζητούμενο 6

Με τα αποτελέσματα του σχήματος 10 μπορούνε τα προκύψουν τα εξής πορίσ-

ματα.

Οι κόμβοι στους οποίους δείχνει ο 10 (εδώ 13) δείχνουν να χάνουν σε σημαντικότητα, μιας και ο 10 είχε αυξημένη σημαντικότητα καθώς έδειχναν πάνω του αρκετοί κόμβοι.

Για τους κόμβους που δείχνουν στον 10, θα χρειαστεί να γίνει ένας διαχωρισμός.

Οι κομβόι στους οποίους δείχνει ο 10 (έμμεσα) έχοντας μεταξύ τους 2 οι παραπάνω κόμβους. Αυτοί είναι οι κόμβοι 5, 6, 7 και παρουσιάζουν μια αύξηση στην σημαντικότητα τους.

Αντίθετα, οι κομβόι στους οποίους δείχνει ο 10 (έμμεσα) έχοντας μεταξύ τους λιγότερο από 2 κόμβους φαίνεται να χάνουν σε σημαντικότητα.