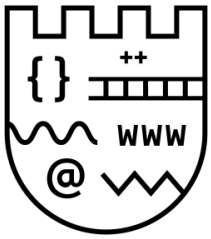


Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Τμήμα Πληροφορικής

Τεχνική αναφορά για NCO-03-02

Υλοποίηση αριθμητικών μεθόδων

Αλέξανδρος Κόρκος
alexkork@csd.auth.gr
3870

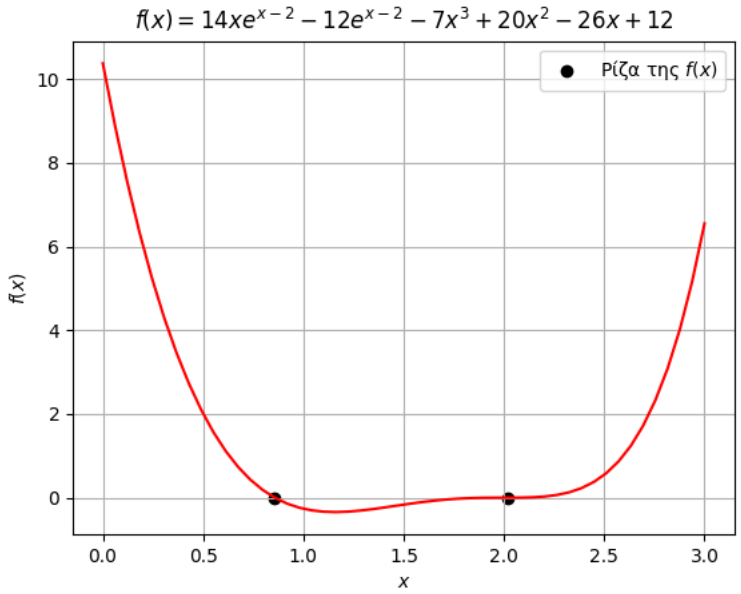
Θεσσαλονίκη, 8 Ιουνίου 2024

Περιεχόμενα

Μέρος I
Πρώτη εργασία

1 Αριθμητική επίλυση μη γραμμικών εξισώσεων

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 $r_2 \approx 2.00001$.

1.1 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.

```
def partition(f, a, b):
    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
```

Κώδικας 1: Μέθοδος της διχοτόμισης

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.

1.2 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.

```
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
```

Κώδικας 2: Newton-Raphson

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.*»

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

Κώδικας 3: Τετραγωνική σύγκλιση

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.

1.3 Μέθοδος της τέμνουσας

Για τον υπολογισμό στης συνάρτησης σύμφωνα με την μέθοδο της τέμνουσας, χρησιμοποιήθηκε ο παρακάτω κώδικας. Όπου f η συνάρτηση, x_0 και x_1 αρχικές προσεγγίσεις της ρίζας.

```python
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
```

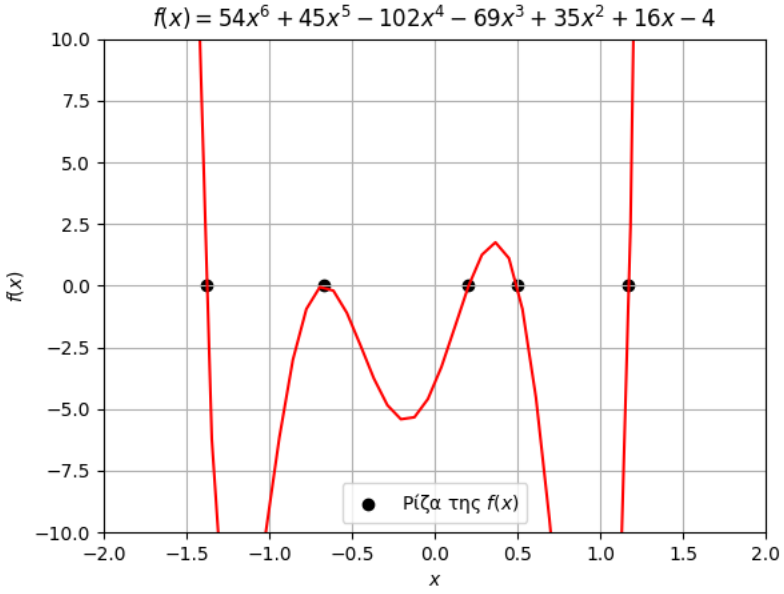
Κώδικας 4: Μέθοδος της τέμνουσας

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.

2 Αριθμητική επίλυση εξισώσεων με τροποποιημένες μεθόδους

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 $r_5 \approx 1.17611$.

2.1 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.

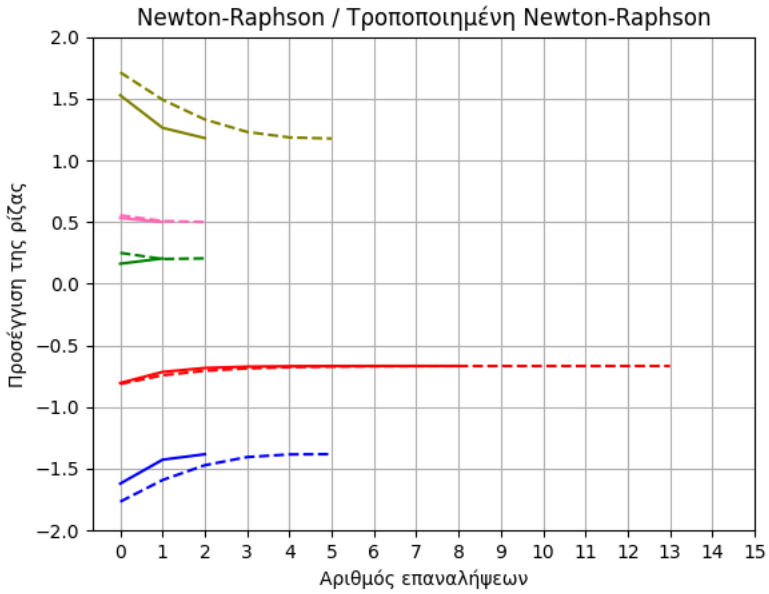
```
def newton(f, df, d2f, x0):
    n = 1
    xi = x0
    while True:
        xi_1 = xi - (1 / ((df(xi) / f(xi)) - (d2f(xi) / (2 * df(xi)))))
        if abs(f(xi_1) / df(xi_1)) < e:
            break
        n += 1
        xi = xi_1

    return xi
```

Κώδικας 5: Τροποποιημένη Newton-Raphson

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.

2.2 Τροποποιημένη μέθοδος διχοτόμησης

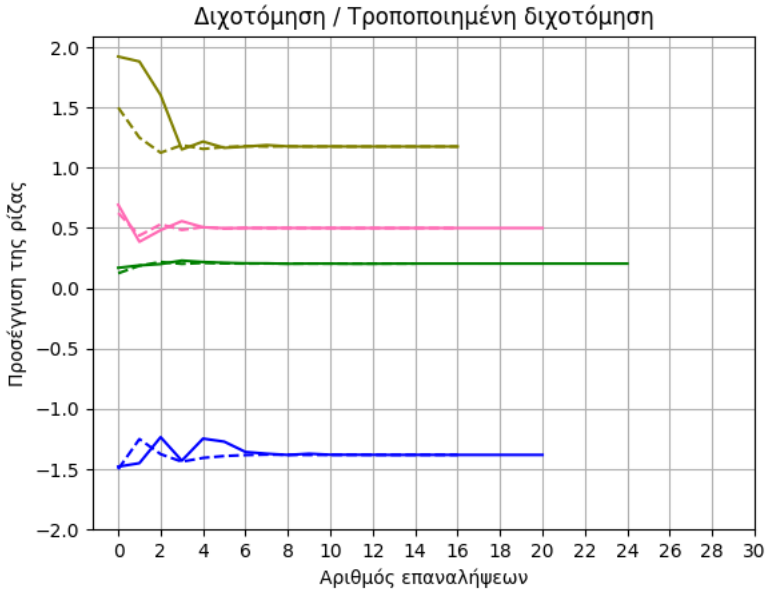
Για την υλοποίηση της τροποποιημένης μεθόδου διχοτόμησης ακολουθείτε η παρακάτω υλοποίηση.

```python
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
```

Κώδικας 6: Τροποποιημένη διχοτόμηση

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.

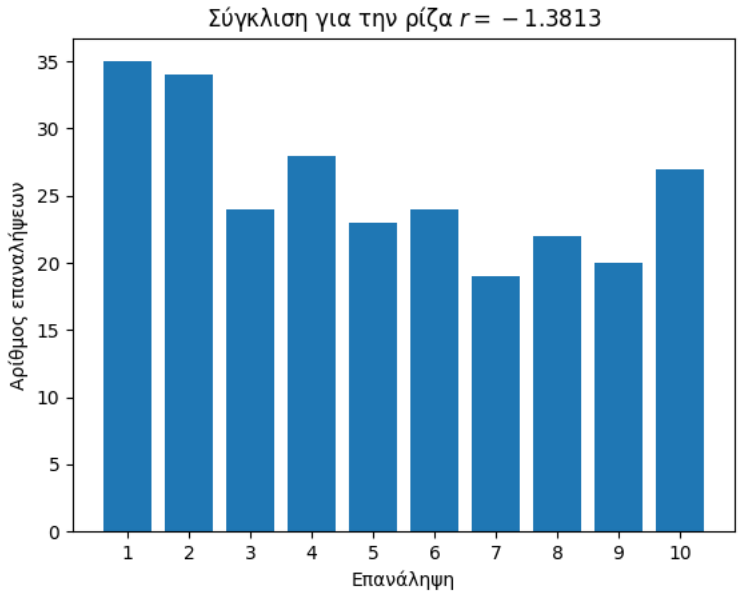


Σχήμα 4: Ταχύτητα σύγκλισης διχοτόμησης

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.

2.3 Τροποποιημένη μέθοδος τέμνουσας

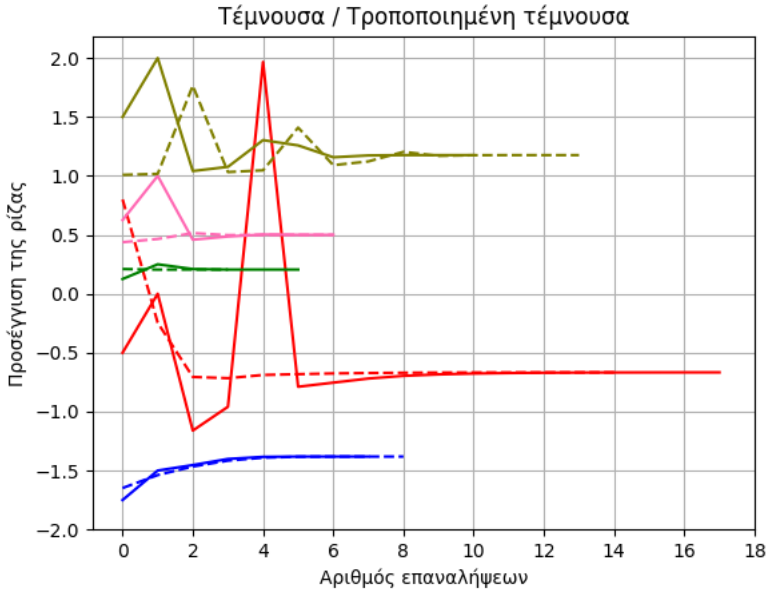
Το παρακάτω απόσωσμα κώδικα υπολογίζει την τροποποιημένη μέθοδο της τέμνουσας. Όπου f η συνάρτηση, x_0, x_1 και x_2 αρχικές προσεγγίσεις της ρίζας.

```
def secant(f, x0, x1, x2):
    xi = x0
    xi1 = x1
    xi2 = x2
    n = 1
    while abs(f(xi)) > e:
        q = f(xi) / f(xi1)
        r = f(xi2) / f(xi1)
        s = f(xi2) / f(xi)
        xi3 = xi2 - (r * (r - q) * (xi2 - xi1) + (1 - r) * s * (xi2 - xi)) / ((q - 1) *
        ↪  (r - 1) * (s - 1))
        xi = xi1
        xi1 = xi2
        xi2 = xi3
        n += 1
    return xi
```

Κώδικας 7: Τροποποιημένη τέμνουσα

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.

3 Σύστημα γραμμικών εξισώσεων

3.1 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 **P** καθορίζεται από τις εναλλαγές γραμμών που απαιτεί η διαδικασία της απαλοιφής με οδήγηση.
- Ο **L** έχει τους πολλαπλασιαστές της απαλοιφής κάτω από την διαγώνιο.
- Ο **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 **U**, **L**, **P**.

```
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
```

Κώδικας 8: Αποσύνθεση

Για ελαχιστοποίηση της ταχύτητας επίλυσης του γραμμικού συστήματος, στο πρόγραμμα που αναπτύχθηκε αποφεύγεται η εμπρός αντικατάσταση καθώς αντί να χρησιμοποιηθεί ο πίνακας **A** χρησιμοποιείται ο $[\mathbf{A}|\mathbf{b}]$. Έτσι το μόνο που χρειάζεται να γίνει είναι μια απλή πίσω αντικατάσταση για να υπολογιστεί το διάνυσμα **x**, όπως φαίνεται και παρακάτω.

```
def backSubstitution(U, b):
    n = b.size
    x = np.zeros(n)
    b = U[:, n]
    U = U[:, : n]
    n = U.shape[0]
    for i in range(n-1, -1, -1):
        temp = b[i]
        for j in range(n-1, i, -1):
            temp -= x[j] * U[i, j]
        x[i] = temp / U[i, i]
    return x
```

Κώδικας 9: Πίσω αντικατάσταση

3.2 Αποσύνθεση Cholesky

Η αποσύνθεσή Cholesky ορίζεται ως:

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

```
def cholesky(A):
    (n, m) = A.shape
    L = np.zeros((n, m))
    for k in range(n):
        for i in range(k + 1):
            sum = 0
            for j in range(i):
                sum += L[i, j] * L[k, j]
            if k == i:
                L[k, i] = np.sqrt(A[k, k] - sum)
            else:
                L[k, i] = (A[k, i] - sum) / L[i, i]
    return L
```

Κώδικας 10: Cholesky

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 $\frac{1}{l_{ii}}(a_{ki} - \sum_{j=1}^{i-1} l_{ij}l_{kj})$.

3.3 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.

```
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
```

Κώδικας 11: Gauss-Seidel

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.

4 Pagerank

4.1 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 **G** αποδεικνύεται πως είναι στοχαστικός.

4.2 Ζητούμενο 2

Για την κατασκευαστεί του πίνακα **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.

```
def createGoogleMatrix(A, q):
    n = A.shape[0]
    ni = []
    G = np.zeros((n, n))
    for i in range(n):
        ni.append(sum(A[i, :]))

    for i in range(n):
        for j in range(n):
            G[i, j] = (q / n) + ((A[j, i] * (1 - q)) / ni[j])

    return G
```

Ο πίνακας **G** που υπολογίστηκε έχει ως εξής:

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 **G**

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

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

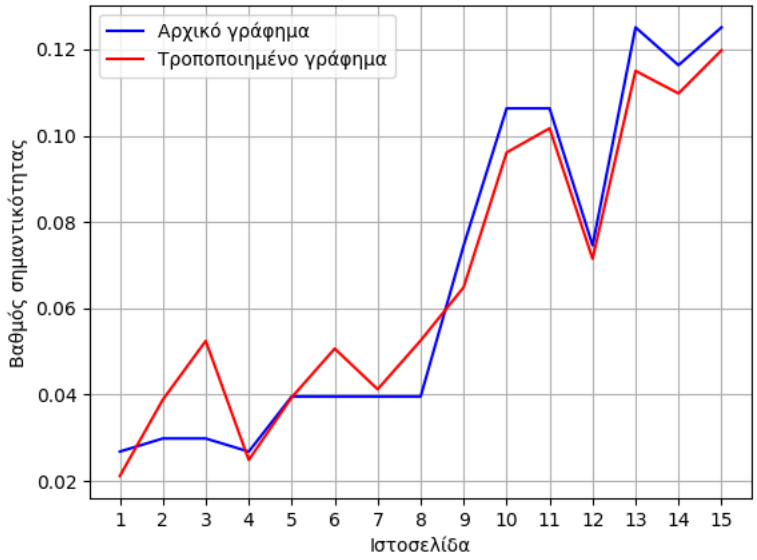
Ο πίνακας **p** που υπολογίστηκε:

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

Όπου ταυτίζεται με τον πίνακα **p** που δίνεται στην εκφώνηση.

4.3 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.

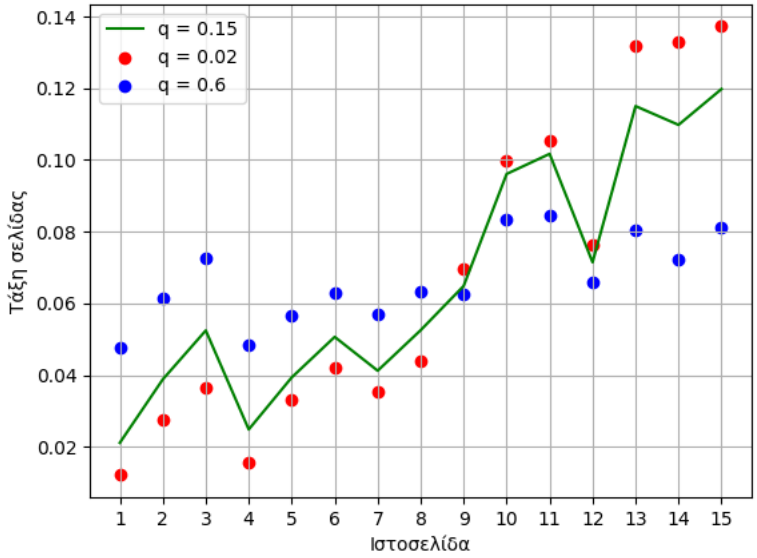


Σχήμα 7: Ζητούμενο 3

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

4.4 Ζητούμενο 4

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



Σχήμα 8: Ζητούμενο 4

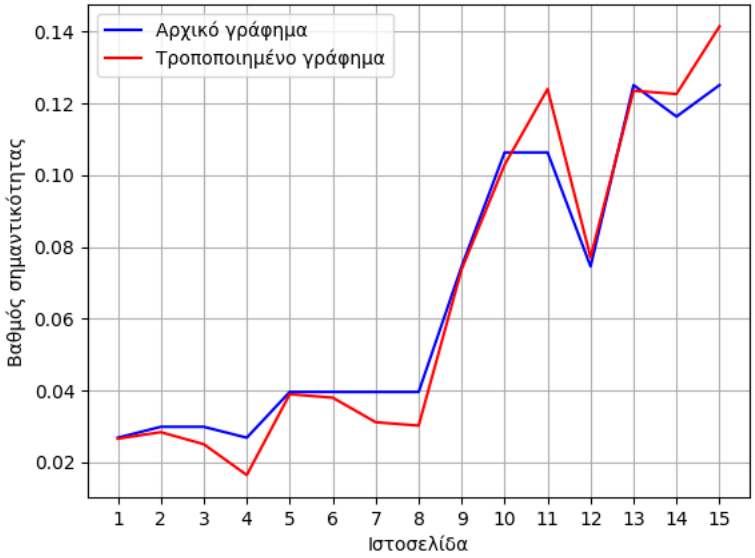
Εξετάζοντας λοιπόν τις μεταβολές, συμπεραίνετε:

- 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.

4.5 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.

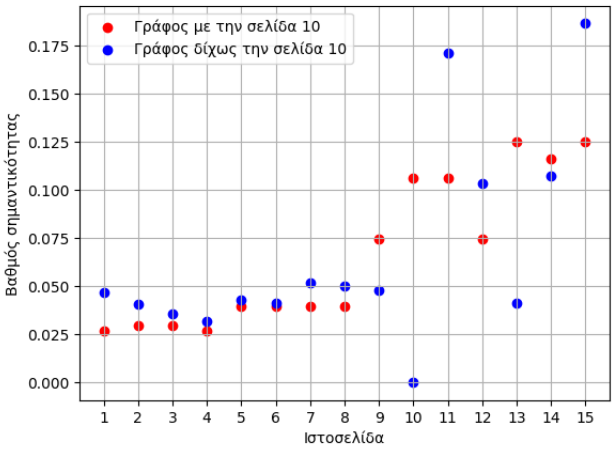


Σχήμα 9: Ζητούμενο 5

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

4.6 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 πορίσματα.

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.

Μέρος ΙΙ
Δεύτερη εργασία

5 Πολυωνυμική παρεμβολή

Για την υλοποίηση των παρακάτω μεθόδων χρησιμοποιήθηκαν οι παρακάτω τιμές των x στο ανομοιόμορφο κατανεμημένο διάστημα $[-\pi, \pi]$.

i	0	1	2	3	4	5	6	7	8	9
x_i	π	-2.80	-2.22	-1.31	-0.19	0.07	0.8	1.94	3.07	π
$sin(x_i)$	0	-0.33499	-0.79567	-0.96618	-0.18886	0.06994	0.71736	0.93262	0.07153	0

Πίνακας 2: Τιμές της $sin(x)$ για x

5.1 Μέθοδος Lagrange

Για την υλοποίησή της μεθόδου Lagrange χρησιμοποιήθηκαν οι παρακάτω τύποι:

$$p_n(x) = \sum_{i=0}^{n} y_i L_i(x)$$

όπου

$$L_i(x) = \frac{(x - x_0)...(x - x_{i-1})(x - x_{i+1})...(x - x_n)}{(x_i - x_0)...(x_i - x_{i-1})(x_i - x_{i+1})...(x_i - x_n)}$$

με $i = 0, 1, ..., n$. Η υλοποίηση φαίνεται παρακάτω:

```
def lagrangeCoefficients(xi, x, n):
    L = np.ones(n)

    for i in range(n):
        for j in range(n):
            if i != j:
                L[i] *= (x - xi[j]) / (xi[i] - xi[j])
    return L

def lagrange(xi, yi, x):
    n = xi.shape[0]
    pn = 0
    for i in range(n):
        pn += yi[i] * lagrangeCoefficients(xi, x, n)[i]

    return pn
```

Κώδικας 12: Μέθοδος Lagrange

Εκτελώντας των κώδικα 12 προκύπτει (όπως φαίνεται και από τον πίνακα 3) πως ταυτίζεται με την πραγματική τιμή του $sin(x_i)$ για $i = 0, 1, ..., 9$ που χρησιμοποιήθηκε.

i	0	1	2	3	4	5	6	7	8	9
x_i	π	-2.80	-2.22	-1.31	-0.19	0.07	0.8	1.94	3.07	π
$sin(x_i)$	0	-0.33499	-0.79567	-0.96618	-0.18886	0.06994	0.71736	0.93262	0.07153	0
$lagrange(x_i)$	-0	-0.33499	-0.79567	-0.96618	-0.18886	0.06994	0.71736	0.93262	0.07153	0

Πίνακας 3: Τιμές της $sin(x)$ και $lagrange(x)$.

5.2 Μέθοδος Splines

Το πολυώνυμο της φυσικής κυβικής Spline ορίζεται από τον εξής τύπο:

$$S_{n-1}(x) = y_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + d_{n-1}(x - x_{n-1})^3$$

για $[x_{n-1}, x_n]$.
Για να υπολογιστεί λοιπόν το πολυώνυμο, θα πρέπει να υπολογιστούν οι συντελεστές b_i, c_i, d_i. Για τον υπολογισμό λοιπόν, θα χρειαστεί να χρησιμοποιηθούν πίνακες, ως εξης:

- **d**, που αποθηκεύει τις τιμές d_i σε κάθε σειρά του πίνακα με τον τύπο:

  $$d_i = \frac{c_{i+1} - c_i}{3_i}$$

  για $i = 1, ..., n-1$

- **b**, που αποθηκεύει τις τιμές b_i σε κάθε σειρά του πίνακα με τον τύπο:

  $$b_i = \frac{i}{i} - \frac{i}{3}(2c_i + c_{i+1})$$

  για $i = 1, ..., n-1$

- **c**, που προκύπτει από την λύση του παρακάτω συστήματος:

$$\begin{bmatrix} 1 & 0 & 0 & & & \\ {\scriptstyle i} & 2_1 + 2_2 & {\scriptstyle 2} & \ddots & & \\ 0 & {\scriptstyle 2} & 2_2 + 2_3 & {\scriptstyle 3} & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & & {\scriptstyle n-2} & 2_{n-2} + 2_{n-1} & {\scriptstyle n-1} \\ & & & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 3(\frac{2}{2} - \frac{1}{1}) \\ \vdots \\ 3(\frac{n-1}{n-1} - \frac{n-2}{n-2}) \\ 0 \end{bmatrix}$$

Επιπλέον θα χρειαστούν και οι βοηθητικοί πίνακές:

- **δ**, που προκύπτει από τον τύπο:

  $$_i = x_{i+1} - x_i$$

  για $i = 1, ..., n-1$

- **Δ**, που προκύπτει από τον τύπο:

  $$_i = y_{i+1} - y_i$$

  για $i = 1, ..., n-1$

Αφού εκτελεσθεί η διαδικασία της φυσικής κυβικής Spline,υπολογίζεται η τιμή του $S(x)$ (όπου για x είναι η γωνία του $sin(x)$ που θέλουμε να υπολογίσουμε).

Η υλοποίηση των προηγουμένων, φαίνεται παρακάτω:

```python
def splines(xi, yi, x):
    n = xi.shape[0]
    A = np.zeros((n, n))
    r = np.zeros(n)
    dx = np.zeros(n - 1)
    dy = np.zeros(n - 1)
    a = yi.copy()
    b = np.zeros(n)
    d = np.zeros(n)

    for i in range(n - 1):
        dx[i] = xi[i + 1] - xi[i]
        dy[i] = yi[i + 1] - yi[i]

    A[0, 0] = 1
    A[-1, -1] = 1

    for i in range(1, n - 1):
        A[i, i] = 2 * (dx[i-1] + dx[i])
        A[i, i + 1] = dx[i]
        A[i, i - 1] = dx[i - 1]

        r[i] = 3 * (dy[i] / dx[i] - dy[i - 1] / dx[i - 1])

    c = np.linalg.inv(A).dot(r)

    for i in range(n - 1):
        b[i] = dy[i] / dx[i] - dx[i] * (2 * c[i] + c[i + 1]) / 3
        d[i] = (c[i + 1] - c[i]) / (3 * dx[i])

    return a, b, c, d
```

Κώδικας 13: Μέθοδος Splines

Εκτελώντας των κώδικα 13 προκύπτει πως ταυτίζεται με την πραγματική τιμή του $sin(x)$ για κάθε x που χρησιμοποιήθηκε.

5.3 Μέθοδος ελαχίστων τετραγώνων

Για να λειτουργήσει η μέθοδος των ελάχιστων τετραγώνων, πρέπει να προσδιορίσουμε το αρχικό πολυώνυμο με έναν συγκεκριμένο βαθμό, δηλαδή να έχει την μορφή:

$$P(x) = c_1 + c_2 x + ... + c_n x^n$$

όπου n ο βαθμός του πολυωνύμου. Μέσω λοιπόν της μεθόδου θέλουμε η γραφική παράσταση του $P(x)$ να πλησιάσει όσο τον δυνατό πιο κοντά στην γραφική συνάρτηση του $sin(x)$. Συνεπώς, πρέπει να υπολογιστούν οι τιμές των $c_i, i = 1, 2, ..., n$. Για να γίνει αυτό θα πρέπει να χρησιμοποιηθούν οι εξής πίνακες:

- **A**, που αποθηκεύει της τιμές των x υψωμένες στον βαθμό του πολυωνύμου (για κάθε σειρά διαφορετικό x), δηλαδή:

$$\mathbf{A} = \begin{bmatrix} 1 & x_1 & x_1^2 & ... & x_1^n \\ 1 & x_2 & x_2^2 & ... & x_2^n \\ 1 & x_3 & x_3^2 & ... & x_3^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & ... & x_m^n \end{bmatrix}$$

  όπου m το πλήθος των τιμών x που δίνονται την μέθοδο.

- **c**, που αποθηκεύει της τιμές c_i για $i = 1, 2, ..., n$, δηλαδή:

$$\mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

- **b**, που αποθηκεύει σε κάθε σειρά την τιμή $sin(x_i)$ για $i = 1, 2, ..., m$, δηλαδή:

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Τέλος, για να βρούμε τις τιμές των c, θα πρέπει να λυθεί το σύστημα:

$$\mathbf{A^T A c} = \mathbf{A^T b}$$

Αφού εκτελεσθεί η διαδικασία των ελάχιστων τετράγωνων, υπολογίζεται η τιμή του $P(x)$ (όπου για x είναι η γωνία του $sin(x)$ που θέλουμε να υπολογίσουμε). Η υλοποίηση των παραπάνω, φαίνεται παρακάτω:

```
def leastSquares(xi, yi, x, degree = 9):
    n = xi.shape[0]
    A = np.zeros((n, degree))
    b = yi

    for i in range(degree):
        A[:, i] = xi[:] ** i
    c = np.linalg.inv(np.dot(A.T, A)).dot(A.T).dot(b)

    return c
```

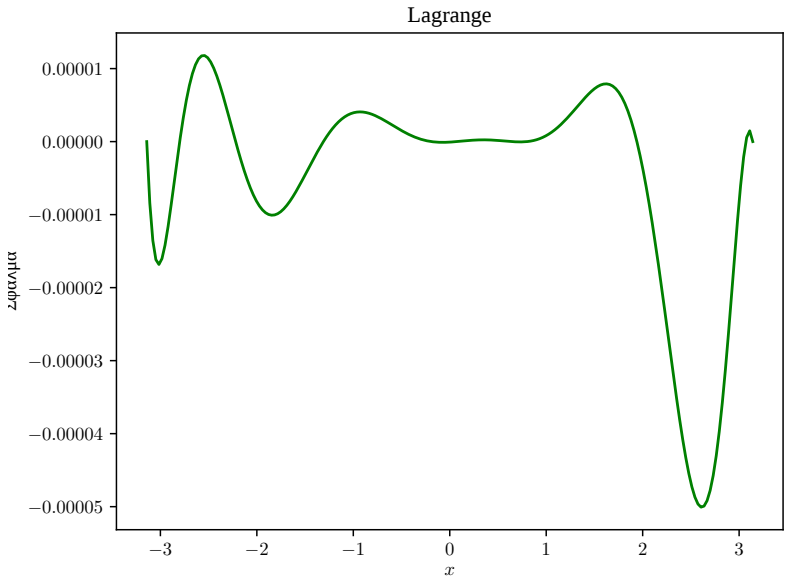
Κώδικας 14: Μέθοδος ελαχίστων τετραγώνων

Εκτελώντας των κώδικα 14 προκύπτει πως ταυτίζεται με την πραγματική τιμή του $sin(x)$ για κάθε x που χρησιμοποιήθηκε.

5.4 Υπολογισμός σφαλμάτων

Τα διαγράμματά σφαλμάτων που ακολουθούν, διαμορφώνονται εκτελώντας τα προγράμματα 12, 13, 14. Επιπλέον, εξετάζονται κάθε φορά 200 σημεία στο διάστημα $[-\pi, \pi]$ και χρησιμοποιούνται οι κατάλληλες μετρικές ($RMSE$) για την αξιολόγηση.

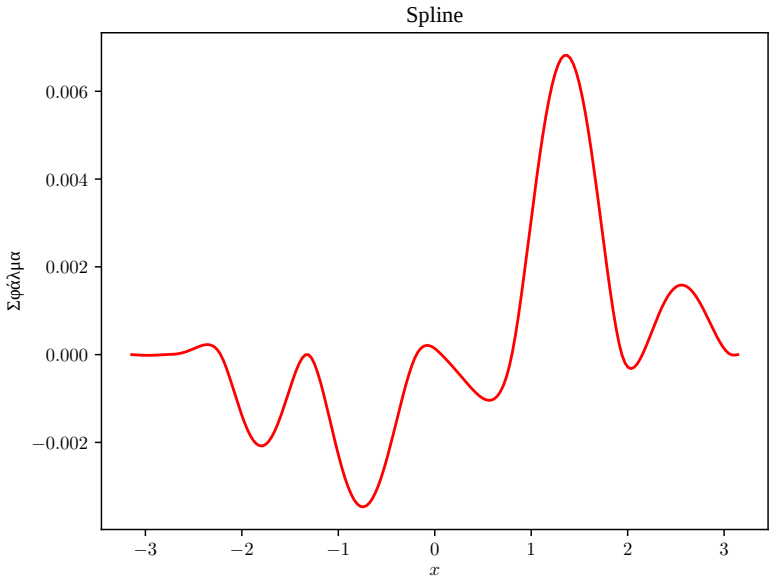
5.4.1 Σφάλμα Lagrange



Σχήμα 11: Σφάλμα 200 σημείων με Lagrange

Εξετάζοντάς την ελάχιστη και την μέγιστη τιμή σφάλματος, βρίσκουμε: $|\min| = |-0.0005| = 0.0005, \max = 0.00055$. Έτσι βλέπουμε ότι η ακρίβεια που επιτυγχάνετε είναι 3 δεκαδικά ψηφία. Επιπλέον το $RMSE$ ισούται με 0.000768. Έχοντας το μικρότερο $RMSE$ από όλες τις μεθόδους καθίσταται και η καλύτερή μέθοδος προσέγγισής από τις τρεις.

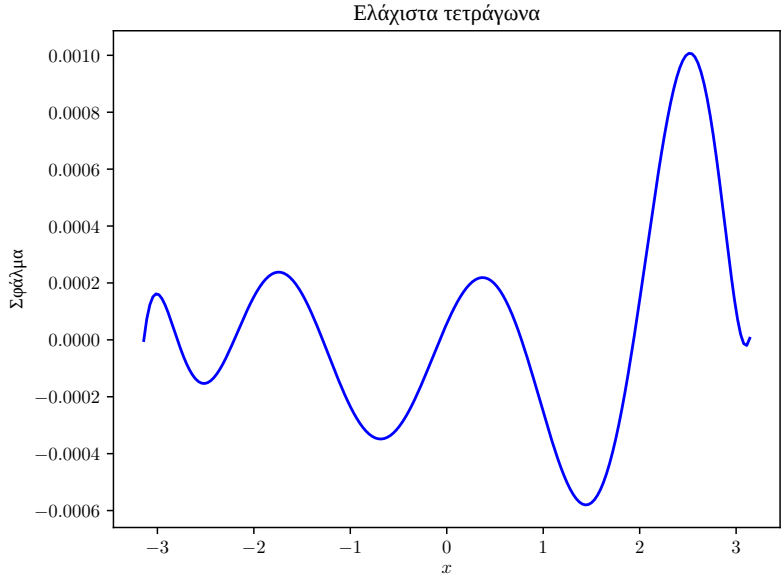
5.4.2 Σφάλμα Splines



Σχήμα 12: Σφάλμα 200 σημείων με Splines

Εξετάζοντάς την ελάχιστη και την μέγιστη τιμή σφάλματος, βρίσκουμε: $|\min| = |-0.003470| = 0.00347, \max = 0.006819$. Έτσι βλέπουμε ότι η ακρίβεια που επιτυγχάνετε είναι 2 δεκαδικά ψηφία. Το $RMSE$ για την παραπάνω μέθοδο ανέρχεται σε 0.002274. Το $RMSE$ της είναι το μεγαλύτερο από τις τρεις, κάνοντας την την χειρότερη μέθοδο προσέγγισης από τις τρεις.

5.4.3 Σφάλμα ελάχιστων τετράγωνων



Σχήμα 13: Σφάλμα 200 σημείων με ελάχιστα τετράγωνα

Εξετάζοντάς την ελάχιστη και την μέγιστη τιμή σφάλματος, βρίσκουμε: $|\min| = |-0.00058| = 0.00058, \max = 0.001007$. Έτσι βλέπουμε ότι η ακρίβεια που επιτυγχάνετε είναι 2 με 3 δεκαδικά ψηφία. Το $RMSE$ για τη παραπάνω μέθοδο είναι 0.00076, καθιστώντας την μέθοδο των ελαχίστων τετραγώνων την δεύτερη καλύτερη από τις τρεις.

6 Υπολογισμός ολοκληρωμάτων

Όπως γνωρίζουμε, για έναν αριθμό από n σημεία (εδώ $n = 11$) χρειάζονται $N = n - 1$ (εδώ $N = 10$) διαμερίσεις.

6.1 Μέθοδος Simpson

Για τον υπολογισμό του ολοκληρώματος με τον κανόνα του τραπεζίου χρησιμοποιήθηκε ο τύπος:

$$\int_a^b f(x)dx = \frac{b-a}{3n}(f(x_0) + f(x_n) + 2\sum_{i=1}^{n/2+1} f(x_{2i}) + 4\sum_{i=1}^{n/2} f(x_{2i-1}))$$

όπου n-ισομήκη υποδιαστήματα του $[a, b]$, $x_0 = a, x_n = b$ και $x_i = x_0 + i\frac{b-a}{n}$ με $i = 0, ..., n$. Η υλοποίηση φαίνεται παρακάτω:

```python
def simpson(f, a, b, n):
    I = f(a) + f(b)
    h = (b - a) / n

    firstSum = 0
    for i in range(1, n // 2 + 1):
        xi = a + (2 * i - 1) * h
        firstSum += f(xi)

    secondSum = 0
    for i in range(1, n // 2):
        xi = a + 2 * i * h
        secondSum += f(xi)

    I += 4 * firstSum + 2 * secondSum
    I *= h / 3

    return I
```

Κώδικας 15: Μέθοδος Simpson

Εκτελώντας τον κώδικα 15 για $f(x) = sin(x)$ για $[0, \pi/2]$, επιστρέφεται πως:

$$\int_0^{\pi/2} sin(x)dx = \mathbf{1.000003}$$

.

Για τον υπολογισμό του θεωρητικού σφάλματος χρησιμοποιήθηκε ο τύπος:

$$|e| \leq \frac{(b-a)^5}{180n^4}M$$

με $M = \max_{x\in[a,b]}\{|f^{(4)}(x)| : x \in [a, b]\}$. Η υλοποίηση φαίνεται παρακάτω:

```python
def simpsonError(d4f, a, b, n):
    M = abs(d4f(b))
    if abs(d4f(a)) > abs(d4f(b)):
        M = abs(d4f(a))
    return (b - a) ** 5 * M / (180 * n ** 4)
```

Κώδικας 16: Μέθοδος Simpson

Εκτελώντας τον κώδικα 16 το θεωρητικό σφάλμα $|e| \leq 0.000005$ και με αριθμητικό σφάλμα $|e| \approx 0.000003$.

6.2 Μέθοδος τραπεζίου

Για τον υπολογισμό του ολοκληρώματος με τον κανόνα του τραπεζίου χρησιμοποιήθηκε ο τύπος:

$$\int_a^b f(x)dx = \frac{b-a}{2n}(f(x_0) + f(x_n) + 2\sum_{i=1}^{n-1} f(x_i))$$

όπου n ισομήκη υποδιαστήματα του $[a, b]$, $x_0 = a, x_n = b$ και $x_i = x_0 + i\frac{b-a}{n}$ με $i = 0, ..., n$. Η υλοποίηση φαίνεται παρακάτω:

```python
def trapezoid(f, a, b, n):
    I = f(a) + f(b)
    h = (b - a) / n
    for i in range(1, n):
        xi = a + i * h
        I += 2 * f(xi)
    I *= h / 2
    return I
```

Κώδικας 17: Μέθοδος τραπεζίου

Εκτελώντας τον κώδικα 17 για $f(x) = sin(x)$ για $[0, \pi/2]$, επιστρέφεται πως:

$$\int_0^{\pi/2} sin(x)dx = \mathbf{0.997943}$$

.

Για τον υπολογισμό του θεωρητικού σφάλματος χρησιμοποιήθηκε ο τύπος:

$$|e| \leq \frac{(b-a)^3}{12n^2} M$$

με $M = \max_{x\in[a,b]}\{|f''(x)| : x \in [a, b]\}$. Η υλοποίηση φαίνεται παρακάτω:

```python
def trapezoidError(d2f, a, b, n):
    M = abs(d2f(b))
    if abs(d2f(a)) > abs(d2f(b)):
        M = abs(d2f(a))
    return (b - a) ** 3 * M / (12 * n ** 2)
```

Κώδικας 18: Μέθοδος τραπεζίου

Εκτελώντας τον κώδικα 18 το θεωρητικό σφάλμα $|e| \leq 0.003230$ και με αριθμητικό σφάλμα $|e| \approx 0.002057$.

7 Χρηματιστηριακές προβλέψεις

Η ημερομηνία γέννησης μου είναι 18/4, για αυτό το λόγο θα χρησιμοποιηθούν οι εξής συνεδρίες από το έτος 2021: 1/4, 6/4, 7/4, 8/4, 9/4, 12/4, 13/4, 14/4, 15/4 και 16/4. Η μετοχές που εξετάστηκαν είναι η **ΟΛΘ** και η **ΣΠΕΙΣ**. Παρακάτω θα ακολουθήσουν οι προβλέψεις για το διάστημα 19/4 έως 23/4 και με Α/Α συνεδρίασης στο διάστημα [11, 15].

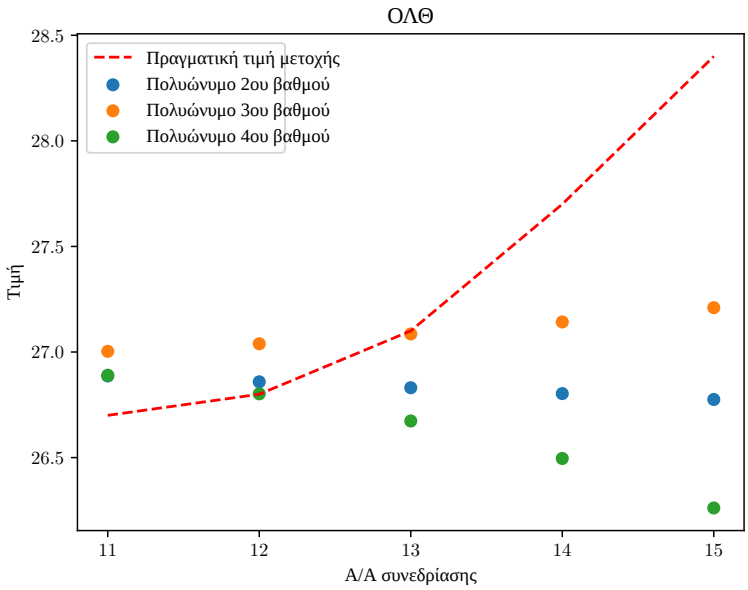
7.1 Προβλέψεις ΟΛΘ

Το ιστορικό της μετοχής για τις δέκα παραπάνω ημερομηνίες, διαμορφώνεται όπως δείχνει ο πίνακας 4.

Α/Α συνεδρίασης	Ημερ/νία	Τιμή κλεισίματος
1	1/4	27,20
2	6/4	27,30
3	7/4	27,00
4	8/4	27,00
5	9/4	27,00
6	12/4	26,90
7	13/4	27,10
8	14/4	27,10
9	15/4	26,80
10	16/4	27,00

Πίνακας 4: Ιστορικό ΟΛΘ

Συνεπώς, εκτελώντας τον κώδικα 14 δίνοντας του τα περιεχόμενα του πίνακα 4, παρατηρείται:



Σχήμα 14: Προσεγγίσεις ΟΛΘ

Για να αποτυπωθεί μια σαφέστερη εικόνα για κάθε προσέγγιση θα πρέπει να εξεταστούν και οι τιμές των προσεγγίσεων με την πραγματική τιμή της μετοχής από τον πίνακα 5:

Α/Α	Προσεγγίσεις			Πραγματική τιμή
	2ου βαθμού	3ου βαθμού	4ου βαθμού	
11	26.88	27.00	26.89	26.70
12	26.85	27.03	26.80	26.80
13	26.83	27.08	26.67	27.10
14	26.80	27.14	26.49	27.70
15	26.77	27.21	26.26	28.40

Πίνακας 5: Αναλυτικές προσεγγίσεις για ΟΛΘ

Αξιολογώντας τα δεδομένα του πίνακα 5 καταλήγουμε στο ότι η καλύτερη προσέγγιση προήλθε από

το πολυώνυμο 3ου βαθμού μιας και οι προβλέψεις είναι σχετικά κοντά με την πραγματική τιμή. Καλές προβλέψεις φαίνεται να έδωσε και το πολυώνυμο 2ου βαθμού σε αντίθεση με αυτό του 4ου βαθμού που πέρα από την πρόβλεψη για την 12 συνεδρίαση, οι υπόλοιπες απείχαν από την πραγματική τιμή της μετοχής.

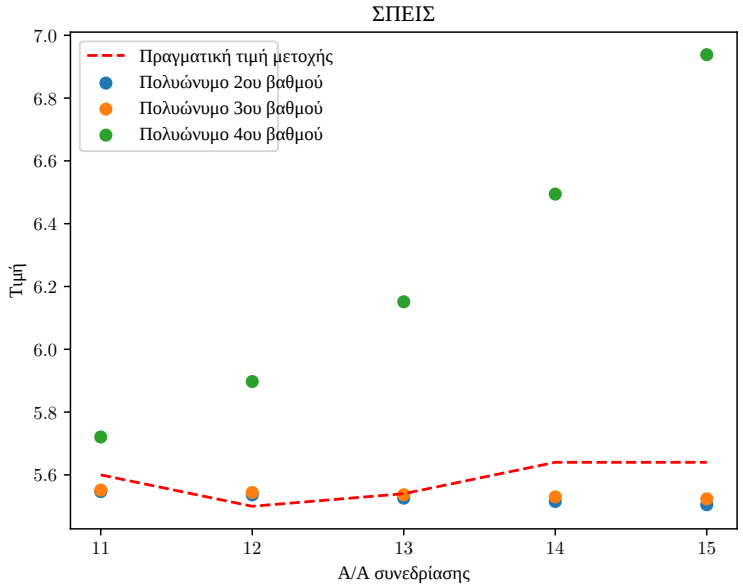
7.2 Προβλέψεις ΣΠΕΙΣ

Το ιστορικό της μετοχής για τις δέκα παραπάνω ημερομηνίες, διαμορφώνεται όπως δείχνει ο πίνακας 6.

Α/Α συνεδρίασης	Ημερ/νία	Τιμή κλεισίματος
1	1/4	5,64
2	6/4	5,60
3	7/4	5,64
4	8/4	5,70
5	9/4	5,66
6	12/4	5,60
7	13/4	5,50
8	14/4	5,50
9	15/4	5,60
10	16/4	5,60

Πίνακας 6: Ιστορικό ΣΠΕΙΣ

Συνεπώς, εκτελώντας τον κώδικα 14 δίνοντας του τα περιεχόμενα του πίνακα 6, παρατηρείται:



Σχήμα 15: Προσεγγίσεις ΣΠΕΙΣ

Για να αποτυπωθεί μια σαφέστερη εικόνα για κάθε προσέγγιση θα πρέπει να εξεταστούν και οι τιμές των προσεγγίσεων με την πραγματική τιμή της μετοχής από τον πίνακα 7:

Α/Α	Προσεγγίσεις			Πραγματική τιμή
	2ου βαθμού	3ου βαθμού	4ου βαθμού	
11	5.54	5.55	5.72	5.64
12	5.53	5.54	5.89	5.50
13	5.52	5.53	6.15	5.54
14	5.51	5.52	6.49	5.64
15	5.50	5.52	6.93	5.64

Πίνακας 7: Αναλυτικές προσεγγίσεις για ΣΠΕΙΣ

Κρίνοντας τα αποτελέσματα από τον πίνακα 7 μπορούμε να αποφανθούμε στο ότι η καλύτερη πρόβλεψη επιτεύχθηκε με το πολυώνυμο 3ου βαθμού. Επιπλέον και η πρόβλεψη του πολυωνύμου 2ου βαθμού είναι και αυτή αρκετά καλή. Τέλος η προσέγγιση του πολυώνυμο 4ου βαθμού είναι εντελώς ανακριβής μιας και δεν πετυχαίνει να βρεθεί κοντά σε καμία τιμή της μετοχής.

Αναφορές

[1] T. Sauer, *Numerical Analysis*. USA: Addison-Wesley Publishing Company, 2nd ed., 2011.

[2] J. Kiusalaas, *Numerical Methods in Engineering with Python 3*. Numerical Methods in Engineering with Python 3, Cambridge University Press, 2013.

[3] G. Papageorgiou, *Αριθμητική Ανάλυση με εφαρμογές σε Mathematica και Matlab*. Εκδόσεις Τσότρας, 2015.