2η Υποχρεωτική Εργασία
Στο Μάθημα της Αριθμητικής Ανάλυσης:
Άσκηση 5

Ονοματεπώνυμο: Μπαρακλιλής Ιωάννης
ΑΕΜ: 3685

4 Ιανουαρίου 2021

1 Πολυωνυμική προσέγγιση

Ζητείται να γίνει προσέγγιση του ημίτονου με πολυώνυμο χρησιμοποιώντας 10 τιμές του. Έχοντας αυτές τις τιμές δεδομένες, υπάρχει μοναδικό πολυώνυμο (το πολύ 9ου βαθμού) που επαληθεύει αυτά τα σημεία και μπορεί να βρεθεί με διάφορες μεθόδους. Επιλέγω να χρησιμοποιήσω την μέθοδο εύρεσης πολυωνύμου Newton.

Για την επιλογή των 10 σημείων, επιλέγω 10 σημεία, με την ακρίβεια τιμής του ημίτονου στα 6 δεκαδικά ψηφία, ομοιόμορφα κατανεμημένα στο διάστημα $[-\pi, \pi]$ τα οποία είναι:

1. $x = -\pi$ με $sin(x) = 0$, δηλαδή το σημείο $(-\pi, 0)$
2. $x = -\frac{7}{9}\pi$ με $sin(x) = -0.642788$, δηλαδή το σημείο $(-\frac{7}{9}\pi, -0.642788)$
3. $x = -\frac{5}{9}\pi$ με $sin(x) = -0.984808$, δηλαδή το σημείο $(-\frac{5}{9}\pi, -0.984808)$
4. $x = -\frac{\pi}{3}$ με $sin(x) = -\frac{\sqrt{3}}{2} \simeq -0.866025$, δηλαδή το σημείο $(-\frac{\pi}{3}, -0.866025)$
5. $x = -\frac{\pi}{9}$ με $sin(x) = -0.342020$, δηλαδή το σημείο $(-\frac{\pi}{9}, -0.342020)$
6. $x = \frac{\pi}{9}$ με $sin(x) = 0.342020$, δηλαδή το σημείο $(\frac{\pi}{9}, 0.342020)$
7. $x = \frac{\pi}{3}$ με $sin(x) = \frac{\sqrt{3}}{2} \simeq 0.866025$, δηλαδή το σημείο $(\frac{\pi}{3}, 0.866025)$
8. $x = \frac{5}{9}\pi$ με $sin(x) = 0.984808$, δηλαδή το σημείο $(\frac{5}{9}\pi, 0.984808)$

9. $x = \frac{7}{9}\pi$ με $sin(x) = 0.642788$, δηλαδή το σημείο $(\frac{7}{9}\pi, 0.642788)$

10. $x = \pi$ με $sin(x) = 0$, δηλαδή το σημείο $(\pi, 0)$

Το ζητούμενο υλοποιείται προγραμματιστικά στην γλώσσα python (3.7) στο αρχείο με όνομα a_polynomial_approximation.py το οποίο φαίνεται παρακάτω:

```
from math import sin, pi

def polynomial_approximate(function_points):
    x = []
    y = []

    for x_y in function_points:
        x.append(x_y[0])
        y.append(x_y[1])

    sort_xy_pairs(x, y)

    dd = []

    temp_dd = []
    for i in range(1, len(function_points)):
        temp_dd.append((y[i] - y[i - 1]) / (x[i] - x[i - 1]))

    dd.append(temp_dd)

    for i in range(2, len(function_points)):
        temp_dd = []

        previous_dd = dd[i - 2]

        for j in range(len(function_points) - i):
            temp_dd.append((previous_dd[j + 1] - previous_dd[j
                ]) / (x[i + j] - x[j]))

        dd.append(temp_dd)

    x_coefficients = [0 for i in range(len(function_points))]

    x_coefficients[0] = y[0]

    x_minus_xi_coefficients = [0 for i in range(len(
        function_points))]
```

```
    x_minus_xi_coefficients[0] = 1

    for i in range(1, len(x)):
        x_minus_xi_coefficients_new = [0 for i in range(len(
            x_minus_xi_coefficients))]

        for j in range(1, len(x_minus_xi_coefficients)):
            x_minus_xi_coefficients_new[j] =
                x_minus_xi_coefficients[j - 1]

        for j in range(len(x_coefficients)):
            x_minus_xi_coefficients_new[j] -=
                x_minus_xi_coefficients[j] * x[i - 1]

        x_minus_xi_coefficients = x_minus_xi_coefficients_new

        for j in range(len(x_coefficients)):
            x_coefficients[j] += x_minus_xi_coefficients[j] *
                dd[i - 1][0]

    return x_coefficients


def sort_xy_pairs(x, y):
    for i in range(len(x)):
        minimum = i
        for j in range(i, len(x)):
            if x[j] < x[minimum]:
                minimum = j

        temp = x[minimum]
        x[minimum] = x[i]
        x[i] = temp

        temp = y[minimum]
        y[minimum] = y[i]
        y[i] = temp


def calculate_polynomial(coefficients, x):
    x_power = 1

    result = 0
```

```
    for c in coefficients:
        result += c * x_power
        x_power *= x

    return result


def sin_approximation(x):
    sin_training_points = [
        [-pi, 0],
        [-(7 / 9) * pi, -0.642788],
        [-(5 / 9) * pi, -0.984808],
        [-pi / 3, -0.866025], # -sqrt(3) / 2
        [-pi / 9, -0.342020],
        [pi / 9, 0.342020],
        [pi / 3, 0.866025], # sqrt(3) / 2
        [(5 / 9) * pi, 0.984808],
        [(7 / 9) * pi, 0.642788],
        [pi, 0]
    ]

    return calculate_polynomial(polynomial_approximate(
        sin_training_points), x)


def main():
    given_x = [
        -pi,
        -(7 / 9) * pi,
        - (5 / 9) * pi,
        -pi / 3,
        -pi / 9,
        pi / 9,
        pi / 3,
        (5 / 9) * pi,
        (7 / 9) * pi,
        pi
    ]

    print("Approximation␣of␣given␣points:")
    for i in range(10):
        approximation = sin_approximation(given_x[i])

        print("Approximation␣of␣sin({:f})␣is␣{:f},␣with␣
```

```
            absolute␣error:␣{:e}".format(
            given_x[i], approximation, abs(sin(given_x[i]) -
                approximation))
        )


if __name__ == '__main__':
    main()
```

Στον παραπάνω κώδικα:

Αρχικά, εισάγονται (γίνονται import) τα sin, pi απο την βιβλιοθήκη math που θα χρειαστούν στην συνέχεια.

Μετά, ορίζεται η συνάρτηση polynomial_approximate η οποία δέχεται ως όρισμα έναν δισδιάστο πίνακα όπου η κάθε γραμμή του αποτελεί ένα σημείο όπου στην πρώτη στήλη υπάρχει το x και στην άλλη το y και επιστρέφει έναν μονοδιάστατο πίνακα με τους συντελεστές του πολυωνύμου που περνάει από αυτά τα σημεία και το κάθε στοιχείο που περιέχει αποτελεί τον συντελεστή στην αντίστοιχή θέση (δηλαδή το πρώτο στοιχείο (θέση 0) του πίνακα αντιστοιχεί στην σταθερά, το δεύτερο (θέση 1) αντιστοιχεί στον συντελεστή του x, το τρίτο (θέση 2), εφόσον το πολυώνυμο είναι δευτέρου βαθμού στον συντελεστή του x^2, κ.ο.κ.).
Ανάλυση της λειτουργίας της:
Αρχικά, αντιγράφω τα x, y των σημείων σε ξεχωριστούς (νέους) πίνακες x, y αντίστοιχα.
Στην συνέχεια, ταξινομώ τους πίνακες x, y ως ζεύγη (εφόσον είναι προϋπόθεση στην κατασκευή διαρεμένων διαφορών η ταξινόμηση των x) με βάση το x (ουσιαστικά ταξινομώ τα x και τα y αλλάζουν θέσεις σε αντιστοιχία με τα x) χρησιμοποιώντας την συνάρτηση sort_xy_pairs που ορίζεται πιο κάτω.
Μετά, υπολογίζω τον πίνακα διαιρεμένων διαφορών (που αποθηκεύεται στον πίνακα με όνομα temp_dd) υπολογίζοντας αρχικά τις διαιρεμένες διαφορές πρώτης τάξης (που εξαρτάται απο το y) και μετά των επομένων τάξεων (που εξαρτώνται από τις προηγούμενης τάξης διαιρεμένες διάφορές) σύμφωνα με την θεωρία.
Στην συνέχεια, ορίζω τον πίνακα x_coefficients που θα αποθηκεύει τους συντελεστές του πολυωνύμου, και αρχικά έχει 0 σε κάθε θέση του εφόσον θα προστίθενται στην συνέχεια οι συντελεστές κάθε βήματος.
Επίσης ορίζω τον βοηθητικό πίνακα x_minus_xi_coefficients που αποθηκεύει τους συντελεστές του πολυωνύμου (η θέση του κάθε στοιχείου αντιστοιχεί στον συντελεστή της αντίστοιχης δύναμης του x) που σχηματίζεται από το γινόμενο $\prod_{i=1}^{j}(x-x_{i-1})$ όπου x_i είναι το x του i-ου σημείου και το j είναι το βήμα του αλγορίθμου στο οποίο βρισκόμαστε (αρχικά (βήμα 0) ο πίνακας έχει τιμή 1 στο πρώτο στοιχείο και 0 στα υπόλοιπα του και στην συνέχεια τους συντε-

λεστές του παραπάνω γινομένου). Για την υπολογισμό των περιεχομένων του (σε κάθε βήμα) αρκεί να θέσουμε ως νέα στοιχεία του τα μετατοπισμένα κατά μία θέση δεξιά στοιχεία του πίνακα του προηγουμένου βήματος (το πρώτο στοιχείο του νέου πίνακα θεωρούμε 0 και το τελευταίο του μετατοπισμένου πίνακα αγνοείται) και να αφαιρέσουμε τα στοιχεία της ίδιας θέσης πολλαπλασιασμένα κατά το x_i (i = βήμα - 1) (δηλαδή στο βήμα 0 έχουμε ότι ο πίνακας έχει τιμή 1 στο πρώτο στοιχείο και 0 στα υπόλοιπα, στο βήμα 1 θα έχει $-x_0$ στο πρώτο στοιχείο, 1 στο δεύτερο και 0 στα υπόλοιπα, στο βήμα 2 θα έχει $(-x_0) \cdot (-x_1)$ στο πρώτο στοιχείο, $-x_0 - (x_1)$ στο δεύτερο, 1 στο τρίτο και 0 στα υπόλοιπα, κ.ο.κ. για τα υπόλοιπα βήματα). Αυτό εξηγείται ως εξής: αρχικά έχουμε 1 στο πρώτο στοιχείο και 0 στα υπόλοιπα δηλαδή το πολυώνυμο 1, την συνέχεια αν πολλαπλασιάσουμε το προηγούμενο με $(x - x_0)$ τότε έχουμε το πολυώνυμο $1 \cdot (x - x_0) = -x_0 + x$ δηλαδή το πολυώνυμο με συντελεστές τα $-x_0$ και 1, στην συνέχεια αν πολλαπλασιάσουμε το προηγούμενο με $(x - x_1)$ έχουμε το πολυώνυμο $(x - x_0) \cdot (x - x_1) = x^2 + x \cdot (-x_1) + (-x_0) \cdot x + (-x_0) \cdot (-x_1) = (-x_0) \cdot (-x_1) + (-x_1 - x_0)x + x^2$ δηλαδή το πολυώνυμο με συντελεστές τα $(-x_0) \cdot (-x_1)$, $-x_0 - (x_1)$ και 1, κ.ο.κ. για τα επόμενα βήματα. Επομένως, βλέπουμε ότι μπορεί με αυτόν τον τρόπο να υπολογιστούν οι συντελεστές του γινομένου αυτού.
Ακολούθως, αρχίζει ένας βρόχος στον οποίο: Αρχικά, ορίζεται ο πίνακας x_minus_xi_coefficients_new που έχει αρχικά σε κάθε θέση 0 και την συνέχεια υπολογίζονται σύμφωνα με την παραπάνω διαδικασία και αποθηκεύονται σε αυτόν οι συντελεστές του πολυωνύμου του γινομένου $\prod_{i=1}^{j}(x - x_{i-1})$ του επομένου βήματος. Τέλος, προστίθεται στον πίνακα x_coefficients ο πίνακας x_minus_xi_coefficients_new πολλαπλασιασμένος με την πρώτη διαιρεμένη διαφορά της τάξης ίδιας με του βήματος.
Σύμφωνα με τα παραπάνω, μπορεί κάποιος να διαπιστώσει ότι αφού τερματίσει ο βρόχος στον πίνακα x_coefficients θα βρίσκονται οι συντελεστές του πολυωνύμου που διέρχεται από τα σημεία που δίνονται ως όρισμα. Τέλος, επιστρέφεται ο πίνακας x_coefficients.

Στην συνέχεια, ορίζεται η συνάρτηση sort_xy_pairs η οποία δέχεται δύο πίνακες και ταξινομεί επί τόπου (in place) τα στοιχεία τους ως ζεύγη βάσει των στοιχείων του πρώτου πίνακα με την μέθοδο της ταξινόμησης με επιλογή. Δηλαδή, ταξινομεί τον πρώτο πίνακα και κάθε ανταλλαγή στοιχείων που γίνεται στον πρώτο γίνεται η αντίστοιχη και στον δεύτερο.

Ακολούθως, ορίζεται η συνάρτηση calculate_polynomial η οποία δέχεται έναν πίνακα συντελεστών πολυωνύμου (όπου η θέση κάθε στοιχείου αντιστοιχεί στην δύναμη του αγνώστου) και έναν (πραγματικό) αριθμό x και υπολογίζει και επιστρέφει την τιμή του πολυωνύμου για το δοθέν σημείο.

Μετά, ορίζεται η συνάρτηση sin_approximation η οποία δέχεται έναν (πραγματικό) αριθμό x και υπολογίζει και επιστρέφει το ημίτονο της δοθείσας γωνίας

x χρησιμοποιώντας πολυωνυμική προσέγγιση με βάση τα παραπάνω σημεία. Αυτή η συνάρτηση για τον υπολογισμό της προσέγγισης του ημίτονου χρησιμοποιεί τις παραπάνω συναρτήσεις. Για τον υπολογισμό του ημίτονου αρκεί κάποιος να εκτελέσει αυτή την συνάρτηση για κάποια γωνία.

Τέλος, ορίζεται η συνάρτηση main, η οποία δεν δέχεται ορίσματα, η οποία θα εκτελεστεί όταν εκτελεστεί το αρχείο στο οποίο αναφερόμαστε και για τα σημεία που επέλεξα παραπάνω υπολογίζει μέσω την πολυωνυμικής προσέγγισης και εμφανίζει (εμφάνιση με ακρίβεια 6 δεκαδικών ψηφίων) το ημίτονο στα παραπάνω σημεία και εμφανίζει το απόλυτο σφάλμα από το πραγματικό ημίτονο, χρησιμοποιώντας την συνάρτηση sin από την βιβλιοθήκη math.

Αν εκτελέσουμε το αρχείο θα τυπωθεί στην οθόνη το ακόλουθο:

```
Approximation of given points:
Approximation of sin(-3.141593) is 0.000000, with absolute error: 1.454732e-15
Approximation of sin(-2.443461) is -0.642788, with absolute error: 3.903135e-07
Approximation of sin(-1.745329) is -0.984808, with absolute error: 2.469878e-07
Approximation of sin(-1.047198) is -0.866025, with absolute error: 4.037844e-07
Approximation of sin(-0.349066) is -0.342020, with absolute error: 1.433257e-07
Approximation of sin(0.349066) is 0.342020, with absolute error: 1.433257e-07
Approximation of sin(1.047198) is 0.866025, with absolute error: 4.037844e-07
Approximation of sin(1.745329) is 0.984808, with absolute error: 2.469878e-07
Approximation of sin(2.443461) is 0.642788, with absolute error: 3.903135e-07
Approximation of sin(3.141593) is 0.000000, with absolute error: 2.958404e-15
```

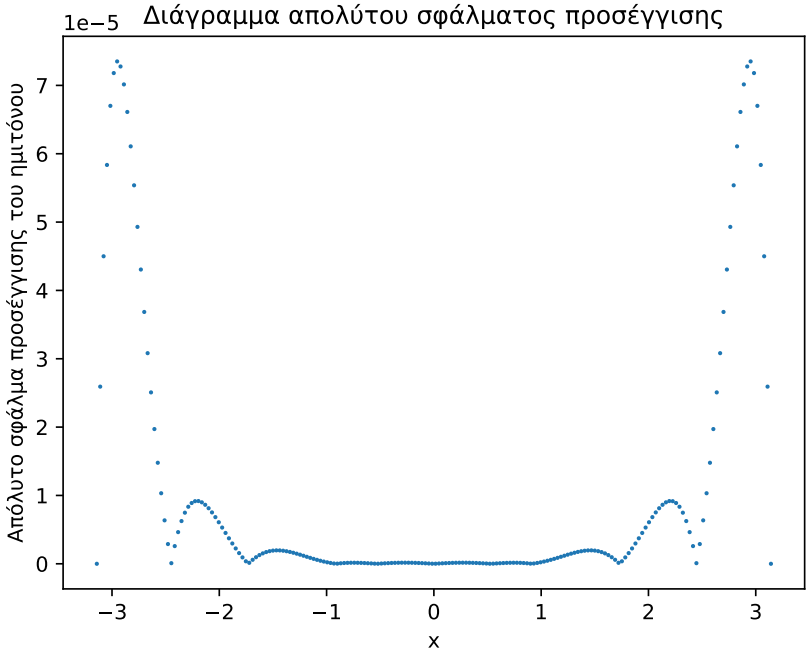
Παραπάνω, βλέπουμε ότι για:

- $x = -\pi$ έχω προσέγγιση έχω προσέγγιση 0.000000 και απόλυτο σφάλμα $1.454732 \cdot 10^{-15}$
- $x = -\frac{7}{9}\pi$ έχω προσέγγιση -0.642788 και απόλυτο σφάλμα $3.903135 \cdot 10^{-7}$
- $x = -\frac{5}{9}\pi$ έχω προσέγγιση -0.984808 και απόλυτο σφάλμα $2.469878 \cdot 10^{-7}$
- $x = -\frac{\pi}{3}$ έχω προσέγγιση -0.866025 και απόλυτο σφάλμα $4.037844 \cdot 10^{-7}$
- $x = -\frac{\pi}{9}$ έχω προσέγγιση -0.342020 και απόλυτο σφάλμα $1.433257 \cdot 10^{-7}$
- $x = \frac{\pi}{9}$ έχω προσέγγιση 0.342020 και απόλυτο σφάλμα $1.433257 \cdot 10^{-7}$
- $x = \frac{\pi}{3}$ έχω προσέγγιση 0.866025 και απόλυτο σφάλμα $4.037844 \cdot 10^{-7}$
- $x = \frac{5}{9}\pi$ έχω προσέγγιση 0.984808 και απόλυτο σφάλμα $2.469878 \cdot 10^{-7}$

- $x = \frac{7}{9}\pi$ έχω προσέγγιση 0.642788 και απόλυτο σφάλμα $3.903135 \cdot 10^{-7}$
- $x = \pi$ έχω προσέγγιση 0.000000 και απόλυτο σφάλμα $2.958404 \cdot 10^{-15}$

Επομένως, στα "γνωστά" σημεία, έχουμε μεγαλύτερο απόλυτο σφάλμα για $x = -\frac{\pi}{3}, x = \frac{\pi}{3}$ με σφάλμα $4.037844 \cdot 10^{-7} = 0.4037844 \cdot 10^{-8} < \frac{1}{2}10^{-8}$, επομένως στα "γνωστά" σημεία έχουμε ακρίβεια τουλάχιστον 8 δεκαδικών ψηφίων.

Αν στην συνέχεια πάρουμε 200 σημεία ομοιόμορφα κατανεμημένα στο $[-\pi, \pi]$, υπολογίσουμε το απόλυτο σφάλμα, και εμφανίσουμε το σφάλμα αυτό κάθε σημείου σε διάγραμμα, έχουμε το παρακάτω (υπολογίστηκε χρησιμοποιώντας την συνάρτηση scatter του pyplot της βιβλιοθήκης matplotlib στο αρχείο a_polynomial_approximation_errors_graph.py που περιλαμβάνει τον ίδιο κώδικα με το αρχείο που περιγράφεται παραπάνω με διαφορετική όμως main που αντί να χρησιμοποιεί τα δεδομένα σημεία, "επιλέγει" ομοιόμορφα κατανεμημένα 200 σημεία στο $[-\pi, \pi]$, υπολογίζει τις προσεγγίσεις κάθε ενός από τα αυτά και κατασκευάζει το διάγραμμα):



Από το παραπάνω διάγραμμα μπορούμε να δούμε ότι στα 200 σημεία το απόλυτο σφάλμα δεν ξεπερνάει το $8 \cdot 10^{-5} = 0.8 \cdot 10^{-6} < \frac{1}{2}10^{-5}$. Επομένως, σε αυτά τα 200 σημεία έχουμε τουλάχιστον 5 δεκαδικά ψηφία ακρίβειας.

2 Προσέγγιση με Splines

Ζητείται να γίνει προσέγγιση του ημίτονου με splines χρησιμοποιώντας 10 τιμές του. Όπως και στην περίπτωση της πολυωνυμικής προσέγγισης για την επιλογή των 10 σημείων, επιλέγω 10 σημεία, με την ακρίβεια τιμής του ημίτονου στα 6 δεκαδικά ψηφία, ομοιόμορφα κατανεμημένα στο διάστημα $[-\pi, \pi]$ τα οποία είναι:

1. $x = -\pi$ με $sin(x) = 0$, δηλαδή το σημείο $(-\pi, 0)$
2. $x = -\frac{7}{9}\pi$ με $sin(x) = -0.642788$, δηλαδή το σημείο $(-\frac{7}{9}\pi, -0.642788)$
3. $x = -\frac{5}{9}\pi$ με $sin(x) = -0.984808$, δηλαδή το σημείο $(-\frac{5}{9}\pi, -0.984808)$
4. $x = -\frac{\pi}{3}$ με $sin(x) = -\frac{\sqrt{3}}{2} \simeq -0.866025$, δηλαδή το σημείο $(-\frac{\pi}{3}, -0.866025)$
5. $x = -\frac{\pi}{9}$ με $sin(x) = -0.342020$, δηλαδή το σημείο $(-\frac{\pi}{9}, -0.342020)$
6. $x = \frac{\pi}{9}$ με $sin(x) = 0.342020$, δηλαδή το σημείο $(\frac{\pi}{9}, 0.342020)$
7. $x = \frac{\pi}{3}$ με $sin(x) = \frac{\sqrt{3}}{2} \simeq 0.866025$, δηλαδή το σημείο $(\frac{\pi}{3}, 0.866025)$
8. $x = \frac{5}{9}\pi$ με $sin(x) = 0.984808$, δηλαδή το σημείο $(\frac{5}{9}\pi, 0.984808)$
9. $x = \frac{7}{9}\pi$ με $sin(x) = 0.642788$, δηλαδή το σημείο $(\frac{7}{9}\pi, 0.642788)$
10. $x = \pi$ με $sin(x) = 0$, δηλαδή το σημείο $(\pi, 0)$

Επίσης, επιλέγω να υπολογιστεί ως spline η φυσική κυβική spline

Το ζητούμενο υλοποιείται προγραμματιστικά στην γλώσσα python (3.7) στο αρχείο με όνομα b_splines_approximation.py το οποίο φαίνεται παρακάτω:

```
from math import sin, pi


def splines_approximate(function_points):
    x = [xy[0] for xy in function_points]
    y = [xy[1] for xy in function_points]
    sort_xy_pairs(x, y)

    x_y = []
    for i in range(len(x)):
        x_y.append([x[i], y[i]])
```

```
coefficients_matrix = [[0 for j in range(4 * (len(x_y) - 1)
    )] for i in range(4 * (len(x_y) - 1))]
constants_matrix = [0 for i in range(4 * (len(x_y) - 1))]

equation_row = 0

for given_point_number in range(1, len(x_y)):
    for x_power in range(3, 0, -1):
        coefficients_matrix[equation_row][3 - x_power + (
            given_point_number - 1) * 4] = x_y[
            given_point_number - 1][0] ** x_power
        coefficients_matrix[equation_row + 1][3 - x_power +
             (given_point_number - 1) * 4] = x_y[
            given_point_number][0] ** x_power
    constants_matrix[equation_row] = x_y[given_point_number
         - 1][1]
    constants_matrix[equation_row + 1] = x_y[
        given_point_number][1]
    coefficients_matrix[equation_row][3 + (
        given_point_number - 1) * 4] = 1
    coefficients_matrix[equation_row + 1][3 + (
        given_point_number - 1) * 4] = 1

    equation_row += 2

for given_point_number in range(1, len(x_y) - 1):
    for x_power in range(2, 0, -1):
        coefficients_matrix[equation_row][2 - x_power + (
            given_point_number - 1) * 4] = (x_y[
            given_point_number][0] ** x_power) * (x_power +
            1)
        coefficients_matrix[equation_row][2 - x_power +
            given_point_number * 4] = -1 * (x_y[
            given_point_number][0] ** x_power) * (x_power +
            1)
    coefficients_matrix[equation_row][2 + (
        given_point_number - 1) * 4] = 1
    coefficients_matrix[equation_row][2 +
        given_point_number * 4] = -1
    constants_matrix[equation_row] = 0

    equation_row += 1
```

```
    for given_point_number in range(1, len(x_y) - 1):
        coefficients_matrix[equation_row][(given_point_number -
             1) * 4] = x_y[given_point_number][0] * 6
        coefficients_matrix[equation_row][given_point_number *
            4] = -1 * x_y[given_point_number][0] * 6
        coefficients_matrix[equation_row][1 + (
            given_point_number - 1) * 4] = 2
        coefficients_matrix[equation_row][1 +
            given_point_number * 4] = -2
        constants_matrix[equation_row] = 0

        equation_row += 1

    coefficients_matrix[equation_row][0] = x_y[0][0] * 6
    coefficients_matrix[equation_row][1] = 2
    constants_matrix[equation_row] = 0
    equation_row += 1

    coefficients_matrix[equation_row][(len(x_y) - 2) * 4] = x_y
        [len(x_y) - 1][0] * 6
    coefficients_matrix[equation_row][1 + (len(x_y) - 2) * 4] =
         2
    constants_matrix[equation_row] = 0

    splines_coefficients_array = solve_system(
        coefficients_matrix, constants_matrix)
    splines_coefficients_matrix = []

    for i in range(0, len(x_y) - 1):
        x_coefficients = []
        x_range = [x_y[i][0], x_y[i + 1][0]]
        for j in range(4):
            x_coefficients.append(splines_coefficients_array[4
                * i + j])
        splines_coefficients_matrix.append([x_range,
            x_coefficients])

    return splines_coefficients_matrix


def calculate_spline(spline_coefficients, x):
    for i in range(len(spline_coefficients)):
        if spline_coefficients[i][0][0] <= x <=
            spline_coefficients[i][0][1]:
```

```
            return spline_coefficients[i][1][0] * x ** 3 +
                spline_coefficients[i][1][1] * x ** 2 +
                spline_coefficients[i][1][2] * x +
                spline_coefficients[i][1][3]

    return None


def swap_rows(matrix, row1, row2):
    temp = matrix[row1]
    matrix[row1] = matrix[row2]
    matrix[row2] = temp


def pivot_matrix(matrix, pivot, P):
    max_row_element_pos = pivot
    for i in range(pivot + 1, len(matrix)):
        if abs(matrix[i][pivot]) > abs(matrix[
            max_row_element_pos][pivot]):
            max_row_element_pos = i

    swap_rows(matrix, pivot, max_row_element_pos)
    swap_rows(P, pivot, max_row_element_pos)


def matrix_vector_multiplication(matrix, vector):
    result = [0 for i in range(len(vector))]

    for i in range(len(matrix)):
        for j in range(len(matrix)):
            result[i] += matrix[i][j] * vector[j]

    return result


def PLU(A):
    P = []
    U = []
    for rowNumber, row in enumerate(A):
        U.append([])
        P.append([])
        for element in row:
            U[rowNumber].append(element)
            P[rowNumber].append(0)
```

```
    for r in range(len(U)):
        P[r][r] = 1

    for row_number in range(len(U) - 1):
        pivot_matrix(U, row_number, P)
        for other_row_num in range(row_number + 1, len(U)):
            U[other_row_num][row_number] = U[other_row_num][
                row_number] / U[row_number][row_number]
            for col_number in range(row_number + 1, len(U)):
                U[other_row_num][col_number] = U[other_row_num][
                    col_number] - (U[other_row_num][row_number])
                     * U[row_number][col_number]

    L = [[0 for i in range(len(U))] for j in range(len(U))]
    for i in range(len(U)):
        L[i][i] = 1
    for i in range(len(U)):
        for j in range(len(U)):
            if i > j:
                L[i][j] = U[i][j]
                U[i][j] = 0

    return P, L, U


def solve_system(A, b):
    P, L, U = PLU(A)

    Pb = matrix_vector_multiplication(P, b)

    y = [0.0 for i in range(len(Pb))]

    for i in range(len(L)):
        y[i] = Pb[i]
        for j in range(len(L)):
            if j == i:
                continue

            y[i] -= L[i][j] * y[j]
        y[i] = y[i] / L[i][i]

    x = [0.0 for i in range(len(y))]
```

```
    for i in range(len(U) - 1, -1, -1):
        x[i] = y[i]
        for j in range(len(U)):
            if j == i:
                continue

            x[i] -= U[i][j] * x[j]
        x[i] = x[i] / U[i][i]

    return x


def sort_xy_pairs(x, y):
    for i in range(len(x)):
        minimum = i
        for j in range(i, len(x)):
            if x[j] < x[minimum]:
                minimum = j

        temp = x[minimum]
        x[minimum] = x[i]
        x[i] = temp

        temp = y[minimum]
        y[minimum] = y[i]
        y[i] = temp


def sin_approximation(x):
    sin_training_points = [
        [-pi, 0],
        [-(7 / 9) * pi, -0.642788],
        [-(5 / 9) * pi, -0.984808],
        [-pi / 3, -0.866025], # -sqrt(3) / 2
        [-pi / 9, -0.342020],
        [pi / 9, 0.342020],
        [pi / 3, 0.866025], # sqrt(3) / 2
        [(5 / 9) * pi, 0.984808],
        [(7 / 9) * pi, 0.642788],
        [pi, 0]
    ]

    return calculate_spline(splines_approximate(
        sin_training_points), x)
```

```
def main():
    given_x = [
        -pi,
        -(7 / 9) * pi,
        -(5 / 9) * pi,
        -pi / 3,
        -pi / 9,
        pi / 9,
        pi / 3,
        (5 / 9) * pi,
        (7 / 9) * pi,
        pi
    ]

    print("Approximation␣of␣given␣points:")
    for i in range(10):
        approximation = sin_approximation(given_x[i])

        print("Approximation␣of␣sin({:f})␣is␣{:f},␣with␣
            absolute␣error:␣{:e}".format(
            given_x[i], approximation, abs(sin(given_x[i]) -
                approximation))
        )


if __name__ == '__main__':
    main()
```

Στον παραπάνω κώδικα:

Αρχικά, εισάγονται (γίνονται import) τα sin, pi απο την βιβλιοθήκη math που θα χρειαστούν στην συνέχεια.

Μετά, ορίζεται η συνάρτηση splines_approximate που δέχεται ως όρισμα έναν δισδιάστο πίνακα όπου η κάθε γραμμή του αποτελεί ένα σημείο όπου στην πρώτη στήλη υπάρχει το x και στην άλλη το y και επιστρέφει έναν δισδιάστατο πίνακα όπου η κάθε γραμμή του αντιστοιχεί σε φυσική κυβική spline ενός διαστήματος. Σε αυτόν τον πίνακα που επιστρέφει, στο πρώτο στοιχείο της κάθε γραμμής υπάρχει ένας πίνακας δύο στοιχείων που θέτουν της αρχή και το τέλος του διαστήματος αντίστοιχα στο οποίο ορίζεται η spline και στο δεύτερο στοιχείο κάθε γραμμής υπάρχει ένας πίνακας 4 θέσεων όπου τα στοιχεία του είναι οι συντελεστές του πολυωνύμου τρίτης τάξης της spline στο αντίστοιχο

διάστημα.
Αναλυτικά:
Αρχικά, ταξινομώ τα σημεία που δίνονται (απαιτείται για τον υπολογισμό της spline) από το όρισμα με βάση το x (ουσιαστικά ταξινομώ τα x και τα y αλλάζουν θέσεις σε αντιστοιχία με τα x) χρησιμοποιώντας την συνάρτηση sort_xy_pairs που ορίζεται πιο κάτω και αποθηκεύω τα ταξινομημένα σημεία στον πίνακα x_y.
Στην συνέχεια ορίζω ένα σύστημα 4n × 4n όπου n είναι ο αριθμός σημείων - 1 αρχικοποιώντας του πίνακες coefficients_matrix που αποθηκεύει τους συντελεστές των αγνώστων του συστήματος και constants_matrix που αποθηκεύει τις σταθερές του συστήματος, όπου οι δύο πίνακες αρχικά έχουν 0 σε κάθε τους θέση.
Το σύστημα αυτό έχει ως αγνώστους τους συντελεστές του πολυωνύμου τρίτης τάξης κάθε κλάδου της spline. Έχουμε ότι για κάθε διάστημα $[x_i, x_{i+1}]$, $i = 1, ..., n$ (με n+1 αριθμό σημείων ορίσματος όπου n ο αριθμός των σημείων) έχουμε την φυσική κυβική spline $s^{(i)}(x) = a^{(i)}x^3 + b^{(i)}x^2 + c^{(i)}x + d^{(i)}$. Επομένως, εφόσον είναι δύο φορές παραγωγίσιμη, έχουμε $(s^{(i)})'(x) = 3a^{(i)}x^2 + 2b^{(i)}x + c^{(i)}$ και $(s^{(i)})''(x) = 6a^{(i)}x + 2b^{(i)}$. Κατασκευάζω αυτό το σύστημα ώστε να ισχύουν οι ακόλουθες συνθήκες της φυσικής κυβικής spline:

- $s^{(i)}(x_i) = y_i \Longrightarrow a^{(i)} \cdot x_i^3 + b^{(i)} \cdot x_i^2 + c^{(i)} \cdot x_i + d^{(i)} = y_i$, $\forall i = 0, ..., n$ με n+1 αριθμό σημείων ορίσματος,
- $s^{(i-1)}(x_i) = s^{(i)}(x_i) = y_i \Longrightarrow a^{(i-1)} \cdot x_i^3 + b^{(i-1)} \cdot x_i^2 + c^{(i-1)} \cdot x_i + d^{(i-1)} = y_i$, $\forall i = 1, ..., n$ όπου n+1 αριθμός σημείων ορίσματος,
- $(s^{(i-1)})'(x_i) = (s^{(i)})'(x_i) \Longrightarrow 3a^{(i-1)}x_i^2 + 2b^{(i-1)}x_i + c^{(i-1)} - 3a^{(i)}x_i^2 - 2b^{(i)}x_i - c^{(i)} = 0$, $\forall i = 1, ..., n$ όπου n+1 αριθμός σημείων ορίσματος,
- $(s^{(i-1)})''(x_i) = (s^{(i)})''(x_i) \Longrightarrow 6a^{(i-1)}x_i + 2b^{(i-1)} - 6a^{(i)}x_i - 2b^{(i)} = 0$, $\forall i = 1, ..., n$ όπου n+1 αριθμός σημείων ορίσματος,
- $(s^{(0)})''(x_0) = 0 \Longrightarrow 6a^{(i)}x_0 + 2b^{(i)} = 0$ και
- $(s^{(0)})''(x_n) = 0 \Longrightarrow 6a^{(i)}x_n + 2b^{(i)} = 0$.

Ακολούθως, λύνω το παραπάνω σύστημα με την συνάρτηση solve_system και χωρίζω το αποτέλεσμα σε πίνακες τεσσάρων στοιχείων (εφόσον έχω κυβική spline κάθε τμήμα της είναι πολυώνυμο τρίτου βαθμού επομένως έχει τέσσερις συντελεστές), εισάγω αυτούς του πίνακες σε νέο δισδιάστατο πίνακα πίνακα που έχει αριθμό γραμμών ίσο με τον αριθμό των δοθέντων σημείων - 1 και στην πρώτη στήλη της κάθε γραμμής έχει έναν πίνακα δύο στοιχείων που περιέχει την αρχή και το τέλος του διαστήματος αντίστοιχα στο οποίο ορίζεται η spline και στο δεύτερο στοιχείο κάθε γραμμής υπάρχει ένας ο πίνακας 4 θέσεων όπου τα στοιχεία του είναι οι συντελεστές του πολυωνύμου τρίτης τάξης της spline

στο αντίστοιχο διάστημα.

Στην συνέχεια, ορίζω την συνάρτηση calculate_spline η οποία δέχεται έναν διασδιάστατο πίνακα με ίδια μορφή με αυτή του πίνακα που επιστρέφει η splines_approximate και έναν (πραγματικό) αριθμό x και υπολογίζει και επιστρέφει την τιμή της spline (του αντίστοιχου διαστήματος) για το δοθέν σημείο (αν δεν υπάρχει διάστημα το οποίο να περιέχει το δοθέν σημείο και να έχει υπολογιστεί Spline σε αυτό, η συνάρτηση επιστέφει None).

Μετά, ορίζονται οι συναρτήσεις swap_rows, pivot_matrix, matrix_vector_multiplication, PLU, solve_system οι οποίες συνολικά έχουν την λειτουργία του να λύνουν ένα γραμμικό σύστημα με την μέθοδο PA = LU χρησιμοποιώντας την συνάρτηση solve_system που δέχεται έναν πίνακα συντελεστών και ένα διάνυσμα (πίνακα στήλη) του συστήματος και επιστρέφει την λύση του συστήματος. Η υλοποίηση τους δεν ζητείται από την εκφώνηση και είναι (εκτός κάποιας διόρθωσης) πανομοιότυπες με εκείνες που ζητήθηκαν στο πρώτο υποερώτημα της τρίτης άσκησης της πρώτης υποχρεωτικής εργασίας, συνεπώς η ανάλυση τους παραλείπεται.

Κατόπιν, ορίζεται η συνάρτηση sort_xy_pairs η οποία δέχεται δύο πίνακες και ταξινομεί επί τόπου (in place) τα στοιχεία τους ως ζεύγη βάσει των στοιχείων του πρώτου πίνακα με την μέθοδο της ταξινόμησης με επιλογή. Δηλαδή, ταξινομεί τον πρώτο πίνακα και κάθε ανταλλαγή στοιχείων που γίνεται στον πρώτο γίνεται η αντίστοιχη και στον δεύτερο. Είναι πανομοιότυπη με εκείνη του πρώτου ζητουμένου.

Μετά, ορίζεται η συνάρτηση sin_approximation η οποία δέχεται έναν (πραγματικό) αριθμό x και υπολογίζει και επιστρέφει το ημίτονο της δοθείσας γωνίας x χρησιμοποιώντας προσέγγιση με splines με βάση τα παραπάνω σημεία. Αυτή η συνάρτηση για τον υπολογισμό της προσέγγισης του ημίτονου χρησιμοποιεί τις παραπάνω συναρτήσεις (αν δεν υπάρχει διάστημα το οποίο να περιέχει το δοθέν σημείο και να έχει υπολογιστεί Spline σε αυτό, η συνάρτηση επιστέφει None). Για τον υπολογισμό του ημίτονου αρκεί κάποιος να εκτελέσει αυτή την συνάρτηση για κάποια γωνία.

Τέλος, ορίζεται η συνάρτηση main, η οποία δεν δέχεται ορίσματα, η οποία θα εκτελεστεί όταν εκτελεστεί το αρχείο στο οποίο αναφερόμαστε και για τα σημεία που επέλεξα παραπάνω υπολογίζει μέσω την προσέγγισης των (φυσικών κυβικών) splines και εμφανίζει (εμφάνιση με ακρίβεια 6 δεκαδικών ψηφίων) το ημίτονο στα παραπάνω σημεία και εμφανίζει το απόλυτο σφάλμα από το πραγματικό ημίτονο, χρησιμοποιώντας την συνάρτηση sin από την βιβλιοθήκη math.

Αν εκτελέσουμε το αρχείο θα τυπωθεί στην οθόνη το ακόλουθο:

```
Approximation of given points:
Approximation of sin(-3.141593) is 0.000000, with absolute error: 3.445093e-16
Approximation of sin(-2.443461) is -0.642788, with absolute error: 3.903135e-07
Approximation of sin(-1.745329) is -0.984808, with absolute error: 2.469878e-07
Approximation of sin(-1.047198) is -0.866025, with absolute error: 4.037844e-07
Approximation of sin(-0.349066) is -0.342020, with absolute error: 1.433257e-07
Approximation of sin(0.349066) is 0.342020, with absolute error: 1.433257e-07
Approximation of sin(1.047198) is 0.866025, with absolute error: 4.037844e-07
Approximation of sin(1.745329) is 0.984808, with absolute error: 2.469878e-07
Approximation of sin(2.443461) is 0.642788, with absolute error: 3.903135e-07
Approximation of sin(3.141593) is -0.000000, with absolute error: 5.665539e-16
```

Παραπάνω βλέπουμε ότι για:

- $x = -\pi$ έχω προσέγγιση 0.000000 και απόλυτο σφάλμα $3.445093 \cdot 10^{-16}$
- $x = -\frac{7}{9}\pi$ έχω προσέγγιση -0.642788 και απόλυτο σφάλμα $3.903135 \cdot 10^{-7}$
- $x = -\frac{5}{9}\pi$ έχω προσέγγιση -0.984808 και απόλυτο σφάλμα $2.469878 \cdot 10^{-7}$
- $x = -\frac{\pi}{3}$ έχω προσέγγιση -0.866025 και απόλυτο σφάλμα $4.037844 \cdot 10^{-7}$
- $x = -\frac{\pi}{9}$ έχω προσέγγιση -0.342020 και απόλυτο σφάλμα $1.433257 \cdot 10^{-7}$
- $x = \frac{\pi}{9}$ έχω προσέγγιση 0.342020 και απόλυτο σφάλμα $1.433257 \cdot 10^{-7}$
- $x = \frac{\pi}{3}$ έχω προσέγγιση 0.866025 και απόλυτο σφάλμα $4.037844 \cdot 10^{-7}$
- $x = \frac{5}{9}\pi$ έχω προσέγγιση 0.984808 και απόλυτο σφάλμα $2.469878 \cdot 10^{-7}$
- $x = \frac{7}{9}\pi$ έχω προσέγγιση 0.642788 και απόλυτο σφάλμα $3.903135 \cdot 10^{-7}$
- $x = \pi$ έχω προσέγγιση 0.000000 και απόλυτο σφάλμα $5.665539 \cdot 10^{-16}$

Επομένως, στα "γνωστά" σημεία, έχουμε μεγαλύτερο απόλυτο σφάλμα για $x = -\frac{\pi}{3}, x = \frac{\pi}{3}$ με (απόλυτο) σφάλμα $4.037844 \cdot 10^{-7} = 0.4037844 \cdot 10^{-8} < \frac{1}{2}10^{-}8$, επομένως στα "γνωστά" σημεία έχουμε ακρίβεια τουλάχιστον 8 δεκαδικών ψηφίων.

Αν στην συνέχεια πάρουμε 200 σημεία ομοιόμορφα κατανεμημένα στο $[-\pi, \pi]$, υπολογίσουμε το απόλυτο σφάλμα, και εμφανίσουμε το σφάλμα αυτό κάθε σημείου σε διάγραμμα έχουμε το παρακάτω (υπολογίστηκε χρησιμοποιώντας την συνάρτηση scatter του pyplot της βιβλιοθήκης matplotlib στο αρχείο

b_splines_approximation_errors_graph.py που περιλαμβάνει τον ίδιο κώδικα με το αρχείο που περιγράφεται παραπάνω με διαφορετική όμως main που αντί να χρησιμοποιεί τα δεδομένα σημεία, "επιλέγει" ομοιόμορφα κατανεμημένα 200 σημεία στο $[-\pi, \pi]$, υπολογίζει τις προσεγγίσεις κάθε ενός από τα αυτά και κατασκευάζει το διάγραμμα):

Από το παραπάνω διάγραμμα μπορούμε να δούμε ότι στα 200 σημεία το απόλυτο σφάλμα δεν ξεπερνάει το $0.00008 < \frac{1}{2}10^{-3}$. Επομένως, σε αυτά τα 200 σημεία έχουμε τουλάχιστον 3 δεκαδικά ψηφία ακρίβειας.

3 Προσέγγιση με ελάχιστα τετράγωνα

Ζητείται να γίνει προσέγγιση του ημίτονου με ελάχιστα τετράγωνα χρησιμοποιώντας 10 τιμές του. Όπως και στην περίπτωση της πολυωνυμικής προσέγγισης για την επιλογή των 10 σημείων, επιλέγω 10 σημεία, με την ακρίβεια τιμής του ημίτονου στα 6 δεκαδικά ψηφία, ομοιόμορφα κατανεμημένα στο διάστημα $[-\pi, \pi]$ τα οποία είναι:

1. $x = -\pi$ με $sin(x) = 0$, δηλαδή το σημείο $(-\pi, 0)$
2. $x = -\frac{7}{9}\pi$ με $sin(x) = -0.642788$, δηλαδή το σημείο $(-\frac{7}{9}\pi, -0.642788)$
3. $x = -\frac{5}{9}\pi$ με $sin(x) = -0.984808$, δηλαδή το σημείο $(-\frac{5}{9}\pi, -0.984808)$

4. $x = -\frac{\pi}{3}$ με $sin(x) = -\frac{\sqrt{3}}{2} \simeq -0.866025$, δηλαδή το σημείο $(-\frac{\pi}{3}, -0.866025)$
5. $x = -\frac{\pi}{9}$ με $sin(x) = -0.342020$, δηλαδή το σημείο $(-\frac{\pi}{9}, -0.342020)$
6. $x = \frac{\pi}{9}$ με $sin(x) = 0.342020$, δηλαδή το σημείο $(\frac{\pi}{9}, 0.342020)$
7. $x = \frac{\pi}{3}$ με $sin(x) = \frac{\sqrt{3}}{2} \simeq 0.866025$, δηλαδή το σημείο $(\frac{\pi}{3}, 0.866025)$
8. $x = \frac{5}{9}\pi$ με $sin(x) = 0.984808$, δηλαδή το σημείο $(\frac{5}{9}\pi, 0.984808)$
9. $x = \frac{7}{9}\pi$ με $sin(x) = 0.642788$, δηλαδή το σημείο $(\frac{7}{9}\pi, 0.642788)$
10. $x = \pi$ με $sin(x) = 0$, δηλαδή το σημείο $(\pi, 0)$

Επίσης, επιλέγω η προσέγγιση αυτή να γίνει με πολυώνυμο δευτέρου βαθμού.

Το ζητούμενο υλοποιείται προγραμματιστικά στην γλώσσα python (3.7) στο αρχείο με όνομα c_least_squares_approximation.py το οποίο φαίνεται παρακάτω:

```
from math import sin, pi


def least_squares(A, b):
    AT = matrix_transposition(A)

    AT_A = matrix_multiplication(AT, A)
    AT_b = matrix_vector_multiplication(AT, b)

    return solve_system(AT_A, AT_b)


def approximate_function_with_least_squares(function_points,
    polynomial_degree):
    A = [[0.0 for j in range(polynomial_degree + 1)] for i in
        range(len(function_points))]
    b = [0.0 for i in range(len(function_points))]

    for i in range(len(function_points)):
        for j in range(polynomial_degree + 1):
            A[i][j] = function_points[i][0] ** j
        b[i] = function_points[i][1]

    return least_squares(A, b)
```

```
def calculate_polynomial(coefficients, x):
    x_power = 1

    result = 0

    for c in coefficients:
        result += c * x_power
        x_power *= x

    return result


def matrix_multiplication(lhs_matrix, rhs_matrix):
    result = [[0.0 for j in range(len(rhs_matrix[0]))] for i in
         range(len(lhs_matrix))]

    for i in range(len(lhs_matrix)):
        for k in range(len(rhs_matrix[0])):
            for j in range(len(rhs_matrix)):
                result[i][k] += lhs_matrix[i][j] * rhs_matrix[j
                    ][k]

    return result


def matrix_transposition(matrix):
    transposed = [[0.0 for j in range(len(matrix))] for i in
        range(len(matrix[0]))]

    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            transposed[j][i] = matrix[i][j]

    return transposed


def swap_rows(matrix, row1, row2):
    temp = matrix[row1]
    matrix[row1] = matrix[row2]
    matrix[row2] = temp
```

```
def pivot_matrix(matrix, pivot, P):
    max_row_element_pos = pivot
    for i in range(pivot + 1, len(matrix)):
        if abs(matrix[i][pivot]) > abs(matrix[
            max_row_element_pos][pivot]):
            max_row_element_pos = i

    swap_rows(matrix, pivot, max_row_element_pos)
    swap_rows(P, pivot, max_row_element_pos)


def matrix_vector_multiplication(matrix, vector):
    result = [0 for i in range(len(matrix))]

    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            result[i] += matrix[i][j] * vector[j]

    return result


def PLU(A):
    P = []
    U = []
    for rowNumber, row in enumerate(A):
        U.append([])
        P.append([])
        for element in row:
            U[rowNumber].append(element)
            P[rowNumber].append(0)

    for r in range(len(U)):
        P[r][r] = 1

    for row_number in range(len(U) - 1):
        pivot_matrix(U, row_number, P)
        for other_row_num in range(row_number + 1, len(U)):
            U[other_row_num][row_number] = U[other_row_num][
                row_number] / U[row_number][row_number]
            for col_number in range(row_number + 1, len(U)):
                U[other_row_num][col_number] = U[other_row_num][
                    col_number] - (U[other_row_num][row_number])
                     * U[row_number][col_number]
```

```
    L = [[0 for i in range(len(U))] for j in range(len(U))]
    for i in range(len(U)):
        L[i][i] = 1
    for i in range(len(U)):
        for j in range(len(U)):
            if i > j:
                L[i][j] = U[i][j]
                U[i][j] = 0

    return P, L, U


def solve_system(A, b):
    P, L, U = PLU(A)

    Pb = matrix_vector_multiplication(P, b)

    y = [0.0 for i in range(len(Pb))]

    for i in range(len(L)):
        y[i] = Pb[i]
        for j in range(len(L)):
            if j == i:
                continue

            y[i] -= L[i][j] * y[j]
        y[i] = y[i] / L[i][i]

    x = [0.0 for i in range(len(y))]

    for i in range(len(U) - 1, -1, -1):
        x[i] = y[i]
        for j in range(len(U)):
            if j == i:
                continue

            x[i] -= U[i][j] * x[j]
        x[i] = x[i] / U[i][i]

    return x


def sin_approximation(x):
    sin_training_points = [
```

```
        [-pi, 0],
        [-(7 / 9) * pi, -0.642788],
        [-(5 / 9) * pi, -0.984808],
        [-pi / 3, -0.866025], # -sqrt(3) / 2
        [-pi / 9, -0.342020],
        [pi / 9, 0.342020],
        [pi / 3, 0.866025], # sqrt(3) / 2
        [(5 / 9) * pi, 0.984808],
        [(7 / 9) * pi, 0.642788],
        [pi, 0]
    ]

    return calculate_polynomial(
        approximate_function_with_least_squares(
        sin_training_points, 2), x)


def main():
    given_x = [
        -pi,
        -(7 / 9) * pi,
        -(5 / 9) * pi,
        -pi / 3,
        -pi / 9,
        pi / 9,
        pi / 3,
        (5 / 9) * pi,
        (7 / 9) * pi,
        pi
    ]

    print("Approximation␣of␣given␣points:")
    for i in range(10):
        approximation = sin_approximation(given_x[i])

        print("Approximation␣of␣sin({:f})␣is␣{:f},␣with␣
            absolute␣error:␣{:f}".format(
            given_x[i], approximation, abs(sin(given_x[i]) -
                approximation))
        )


if __name__ == '__main__':
    main()
```

Στον παραπάνω κώδικα:

Αρχικά, εισάγονται (γίνονται import) τα sin, pi απο την βιβλιοθήκη math που θα χρειαστούν στην συνέχεια.

Στην συνέχεια, ορίζω την συνάρτηση least_squares η οποία δέχεται έναν πίνακα συντελεστών A και ένα διάνυσμα (πίνακα στήλη) σταθερών b και επιστρέφει την λύση του συστήματος με ελάχιστα τετράγωνα.
Αναλυτικά:
Αρχικά, αναστρέφεται ο πίνακας A (με την συνάρτηση matrix_transposition που ορίζεται αργότερα) και ο ανάστροφος αυτός πολλαπλασιάζει απο τα αριστερά τον πίνακα A και το διάνυσμα b του ορίσματος (με τις συναρτήσεις matrix_multiplication matrix_vector_multiplication που εκτελούν πολλαπλασιασμό πινάκων και πολλαπλασιασμό πίνακα με διάνυσμα (πίνακα στήλη) αντίστοιχα και επιστρέφουν το αποτέλεσμα και οι οποίες ορίζονται παρακάτω) και χρησιμοποιώ τα αποτελέσματα των γινομένων ως πίνακα συντελεστών και διάνυσμα σταθερών ενός συστήματος το οποίο λύνω (με την συνάρτηση solve_system που ορίζεται παρακάτω) και επιστρέφω το αποτέλεσμα το οποίο είναι η λύση ελαχίστων τετραγώνων του συστήματος.

Ακολούθως, ορίζω την συνάρτηση approximate_function_with_least_squares η οποία δέχεται ως όρισμα έναν δισδιάστο πίνακα όπου η κάθε γραμμή του αποτελεί ένα σημείο όπου στην πρώτη στήλη υπάρχει το x και στην άλλη το y, και την τάξη του πολυωνύμου και επιστρέφει έναν μονοδιάστατο πίνακα με τους συντελεστές του πολυωνύμου ίδιας τάξης με αυτή που δόθηκε στο όρισμα που αποτελεί την προσέγγιση με ελάχιστα τετράγωνα. Το κάθε στοιχείο που περιέχει ο πίνακας (που επιστρέφεται) αποτελεί τον συντελεστή στην αντίστοιχή θέση (δηλαδή το πρώτο στοιχείο (θέση 0) του πίνακα αντιστοιχεί στην σταθερά, το δεύτερο (θέση 1) αντιστοιχεί στον συντελεστή του x, το τρίτο (θέση 2), εφόσον το πολυώνυμο είναι δευτέρου βαθμού στον συντελεστή του x^2, κ.ο.κ.).
Αναλυτικά: πρώτα αρχικοποιώ τους πίνακες A, b ώστε ο A να έχει αριθμό γραμμών και ίσο με τον αριθμό των δοθέντων σημείων και αριθμό στηλών ίσο με την τάξη του πολυωνύμου + 1 (γιατί πολυώνυμου ν βαθμού έχει ν+1 συντελεστές) που δόθηκε ως παράμετρος και το διάνυσμα (πίνακας στήλη) b να έχει αριθμό γραμμών ίσο με τον αριθμό των δοθέντων σημείων. Στην συνέχεια στους παραπάνω πίνακες για το πολυώνυμο για ελαχίστων τετραγώνων $p(x) = a_0 + a_1x + ... + a_ix^i$ όπου i η τάξη του πολυωνύμου, για κάθε ένα από τα δοθέντα σημεία "προσθέτω" στο σύστημα την εξίσωση $p(x_j) = y_j \Longrightarrow a_0 + a_1x_j + ... + a_ix_j^i = y_j$, όπου $j = 0, ..., n$ με n να είναι ο αριθμός των σημείων που δόθηκαν και i η τάξη του πολυωνύμου. Τέλος επιστρέφω την λύση του συστήματος με ελάχιστα τετράγωνα (χρησιμοποιώντας

την συνάρτηση least_squares) που αποτελεί την προσέγγιση ελαχίστων τετραγώνων των σημείων.

Κατόπιν, ορίζω τις συναρτήσεις calculate_polynomial η οποία δέχεται έναν πίνακα συντελεστών πολυωνύμου (όπου η θέση κάθε στοιχείου αντιστοιχεί στην δύναμη του αγνώστου) και έναν (πραγματικό) αριθμό x και υπολογίζει και επιστρέφει την τιμή του πολυωνύμου για το δοθέν σημείο και την matrix_multiplication που δέχεται δύο πίνακες και επιστρέφει το γινόμενο τους και matrix_transposition που δέχεται έναν πίνακα και επιστρέφει τον ανάστροφο του.

Μετά, ορίζονται οι συναρτήσεις swap_rows, pivot_matrix, matrix_vector_multiplication, PLU, solve_system οι οποίες συνολικά έχουν την λειτουργία του να λύνουν ένα γραμμικό σύστημα με την μέθοδο PA = LU χρησιμοποιώντας την συνάρτηση solve_system που δέχεται έναν πίνακα συντελεστών και ένα διάνυσμα (πίνακα στήλη) του συστήματος και επιστρέφει την λύση του συστήματος. Η υλοποίηση τους δεν ζητείται από την εκφώνηση και είναι (εκτός κάποιας διόρθωσης) πανομοιότυπες με εκείνες που ζητήθηκαν στο πρώτο υποερώτημα της τρίτης άσκησης της πρώτης υποχρεωτικής εργασίας, συνεπώς η ανάλυση τους παραλείπεται.

Στην συνέχεια, ορίζεται η συνάρτηση sin_approximation η οποία δέχεται έναν (πραγματικό) αριθμό x και υπολογίζει και επιστρέφει το ημίτονο της δοθείσας γωνίας x χρησιμοποιώντας προσέγγιση με πολυώνυμο 2ου βαθμού με ελάχιστα τετράγωνα με βάση τα παραπάνω σημεία. Αυτή η συνάρτηση για τον υπολογισμό της προσέγγισης του ημίτονου χρησιμοποιεί τις παραπάνω συναρτήσεις. Για τον υπολογισμό του ημίτονου αρκεί κάποιος να εκτελέσει αυτή την συνάρτηση για κάποια γωνία.

Τέλος, ορίζεται η συνάρτηση main, η οποία δεν δέχεται ορίσματα, η οποία θα εκτελεστεί όταν εκτελεστεί το αρχείο στο οποίο αναφερόμαστε και για τα σημεία που επέλεξα παραπάνω υπολογίζει μέσω την προσέγγισης ελαχίστων τετραγώνων (με πολυώνυμο δευτέρου βαθμού) και εμφανίζει (εμφάνιση με ακρίβεια 6 δεκαδικών ψηφίων) το ημίτονο στα παραπάνω σημεία και εμφανίζει το απόλυτο σφάλμα από το πραγματικό ημίτονο, χρησιμοποιώντας την συνάρτηση sin από την βιβλιοθήκη math.

Αν εκτελέσουμε το αρχείο θα τυπωθεί στην οθόνη το ακόλουθο:

```
Approximation of given points:
Approximation of sin(-3.141593) is -0.674381, with absolute error: 0.674381
Approximation of sin(-2.443461) is -0.524519, with absolute error: 0.118269
Approximation of sin(-1.745329) is -0.374656, with absolute error: 0.610152
Approximation of sin(-1.047198) is -0.224794, with absolute error: 0.641232
Approximation of sin(-0.349066) is -0.074931, with absolute error: 0.267089
Approximation of sin(0.349066) is 0.074931, with absolute error: 0.267089
Approximation of sin(1.047198) is 0.224794, with absolute error: 0.641232
Approximation of sin(1.745329) is 0.374656, with absolute error: 0.610152
Approximation of sin(2.443461) is 0.524519, with absolute error: 0.118269
Approximation of sin(3.141593) is 0.674381, with absolute error: 0.674381
```

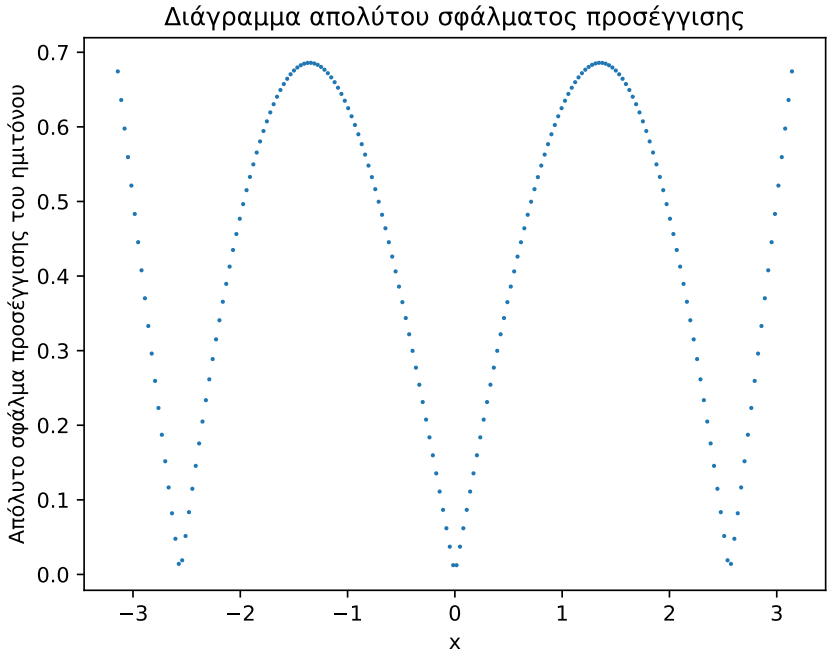
Παραπάνω βλέπουμε ότι για:

- $x = -\pi$ έχω προσέγγιση -0.674381 και απόλυτο σφάλμα 0.674381
- $x = -\frac{7}{9}\pi$ έχω προσέγγιση -0.524519 και απόλυτο σφάλμα 0.118269
- $x = -\frac{5}{9}\pi$ έχω προσέγγιση -0.374656 και απόλυτο σφάλμα 0.610152
- $x = -\frac{\pi}{3}$ έχω προσέγγιση -0.224794 και απόλυτο σφάλμα 0.641232
- $x = -\frac{\pi}{9}$ έχω προσέγγιση -0.074931 και απόλυτο σφάλμα 0.267089
- $x = \frac{\pi}{9}$ έχω προσέγγιση 0.074931 και απόλυτο σφάλμα 0.267089
- $x = \frac{\pi}{3}$ έχω προσέγγιση 0.224794 και απόλυτο σφάλμα 0.641232
- $x = \frac{5}{9}\pi$ έχω προσέγγιση 0.374656 και απόλυτο σφάλμα 0.610152
- $x = \frac{7}{9}\pi$ έχω προσέγγιση 0.524519 και απόλυτο σφάλμα 0.118269
- $x = \pi$ έχω προσέγγιση 0.674381 και απόλυτο σφάλμα 0.674381

Επομένως, στα "γνωστά" σημεία, έχουμε μεγαλύτερο απόλυτο σφάλμα για $x = -\pi, \pi$ με σφάλμα $0.674381 > 0$, επομένως στα "γνωστά" σημεία δεν έχουμε απαραίτητα κάποια δεκαδικά ψηφία ακρίβειας.

Αν στην συνέχεια πάρουμε 200 σημεία ομοιόμορφα κατανεμημένα στο $[-\pi, \pi]$, υπολογίσουμε το απόλυτο σφάλμα, και εμφανίσουμε το σφάλμα αυτό κάθε σημείου σε διάγραμμα έχουμε το παρακάτω (υπολογίστηκε χρησιμοποιώντας την συνάρτηση scatter του pyplot της βιβλιοθήκης matplotlib στο αρχείο c_least_squares_approximation_errors_graph.py που περιλαμβάνει τον ίδιο κώδικα με το αρχείο που περιγράφεται παραπάνω με διαφορετική όμως main που

αντί να χρησιμοποιεί τα δεδομένα σημεία, "επιλέγει" ομοιόμορφα κατανεμημένα 200 σημεία στο $[-\pi, \pi]$, υπολογίζει τις προσεγγίσεις κάθε ενός από τα αυτά και κατασκευάζει το διάγραμμα):



Από το παραπάνω διάγραμμα μπορούμε να δούμε ότι στα 200 σημεία το μέγιστο απόλυτο σφάλμα ξεπερνάει το 0. Επομένως, δεν έχουμε απαραίτητα κάποια δεκαδικά ψηφία ακρίβειας σε κάθε ένα από αυτά τα 200 σημεία .

4 Σύγκριση προσεγγίσεων

Από την παραπάνω ανάλυση, μπορούμε να διαπιστώσουμε ότι, εν γένει στο $[-\pi, \pi]$, η μέθοδος πολυωνυμικής προσέγγισης είναι η πιο ακριβής, ακολουθεί η μέθοδος με splines και η λιγότερο ακριβής είναι η μέθοδος προσέγγισης ελαχίστων τετραγώνων.