Υπολογιστική Φυσική - 3η Σειρά Ασκήσεων

Αναστάσιος Τζαβέλλας / ΑΜ:1110 2016 00255

21/11/2017

1η Άσκηση

Ζητείται το πολυώνυμο Lagrange τετάρτου βαθμού που παρεμβάλει την $f(x) = e^{2x} - 1$ στα σημεία $\{1, 1.1, 1.2, 1.3, 1.4\}$. Η επίλυση αυτού το προβλήματος είναι απλή. Γίνεται εφαρμογή των εξισώσεων (1) και (2), με το αποτέλεσμα να δίνεται στο Σχήμα 1.

$$L_i(x) = \prod_{j=0, j\neq i}^{4} \frac{x - x_j}{x_i - x_j} \tag{1}$$

$$p_4(x) = \sum_{i=0}^{4} L_i(x) f(x_i) \tag{2}$$

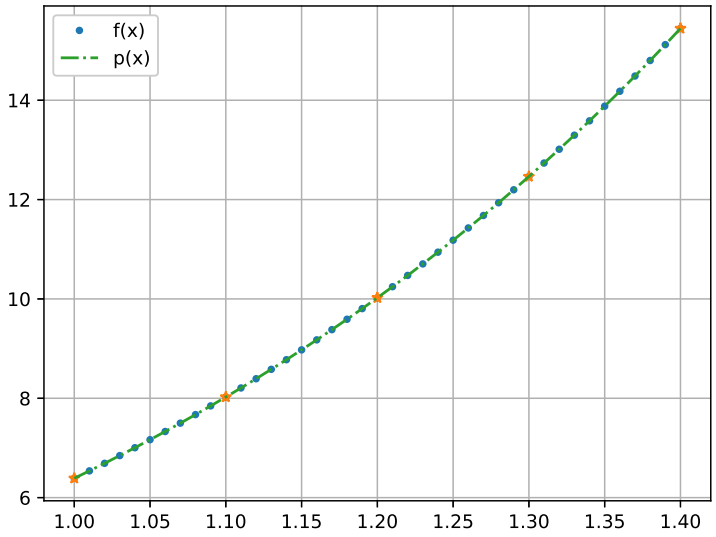
Η τιμή της συνάρτησης στο σημείο $x = 1.25$ είναι $f(1.25) = 11.1824939607$ ενώ η τιμή του πολυωνύμου είναι $p_4(1.25) = 11.1824517251$. Παρατηρείται ότι η διαφορά των δύο τιμών είναι της τάξης του 10^{-5}.

Παρακάτω ακολουθεί ο κώδικας που γράφτηκε σε Python και έγινε χρήση της βιβλιοθήκης Numpy.

ex1.py

```
1 import numpy as np
  import matplotlib.pyplot as plt


  def f(x):
6     return np.exp(2 * x) - 1.
```

```
def L(x, i, xValues):
    """Evaluates ith degree
    Lagrange Polynomial"""
    xi = xValues[i]
    product = 1.
    for j, xj in enumerate(xValues):
        if j != i:
            product = product * (x - xj) / (xi - xj)
    return product


def Lagrange(x, xValues, fValues):
    """Evaluates Lagrange Polynomial"""
    val = 0.
    for i, fi in enumerate(fValues):
        val = val + L(x, i, xValues) * fi
    return val


xValues = np.array([1, 1.1, 1.2, 1.3, 1.4])
fValues = f(xValues)
p = Lagrange(1.25, xValues, fValues)
print('Function Value f(1.25)=', f(1.25))
print('Lagrange polyn p(1.25)=', p)

plt.close('all')
x = np.arange(1.0, 1.41, 0.01)
ps = Lagrange(x, xValues, fValues)
plt.plot(x, f(x), '.', label='f(x)')
plt.plot(xValues, fValues, '*')
plt.plot(x, ps, '-.', label='p(x)')
plt.legend()
plt.grid()
```



Σχήμα 1: Παρεμβολή με πολυώνυμο Lagrange

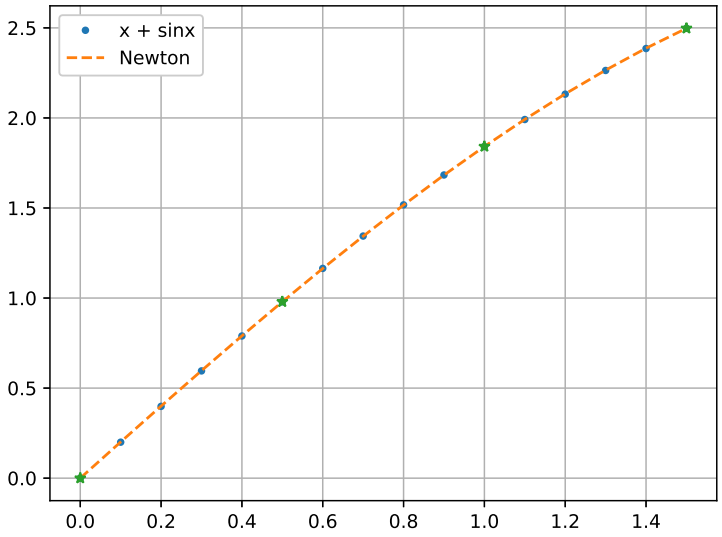
2η Άσκηση

Το πολυώνυμο παρεμβολής Newton μπορεί να βρεθεί με χρήση του αλγορίθμου διηρημένων διαφορών. Συγκεκριμένα το πολυώνυμο που προκύπτει είναι της μορφής (3), όπου οι συντελεστές a_i δίνονται από τις διηρημένες διαφορές (4) και $x_i, i = 0, 1, \ldots n$ τα σημεία παρεμβολής.

$$P_n(x) = a_0 + a_1(x - x_0) + \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_n) \quad (3)$$

$$a_i = f[x_0, x_1, \ldots, x_i] \quad (4)$$

Το αποτέλεσμα της παρεμβολής δίνεται στο Σχήμα 2. Σχετικά με το σφάλμα



Σχήμα 2: Παρεμβολή με πολυώνυμο Newton

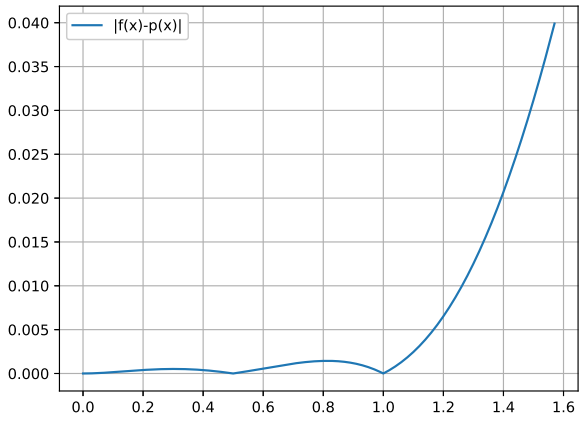
προσέγγισης $f(x) - P(x)$ στο $x \in [0, \frac{\pi}{2}]$, αποδεικνύεται ότι δίνεται από την

εξίσωση (5), με $\xi \in (a, b)$, όπου $a = \min[x_0, \dots x_n]$ και $b = \max[x_0, \dots x_n]$.

$$f(x) - p_n(x) = \frac{(x - x_0) \cdots (x - x_n)}{(n+1)!} f^{(n+1)}(\xi) \tag{5}$$

Λαμβάνοντας την απόλυτη τιμή της (5) και για την δοσμένη $f(x)$ προκύπτει η (6). Στο Σχήμα 3, δίνεται η γραφική παράσταση του σφάλματος στο υπό εξέταση διάστημα.

$$\begin{aligned} |f(x) - p_n(x)| &= \left| \frac{x(x - \frac{1}{2})(x-1)}{4!} \sin\xi \right| \to \\ &\leq \frac{|x||x - \frac{1}{2}||x-1|}{4!} \to \\ &\leq \frac{\frac{\pi}{2}(\frac{\pi}{2} - \frac{1}{2})(\frac{\pi}{2} - 1)}{4!} \to \\ &= \frac{\pi^3 - 3\pi^2 + 2\pi}{192} \approx 0.04 \end{aligned} \tag{6}$$



Σχήμα 3: Παρεμβολή με πολυώνυμο Lagrange

Παρακάτω ακολουθεί ο κώδικας που γράφτηκε σε Python και έγινε χρήση της βιβλιοθήκης Numpy. Ο αλγόριθμος διηρημένων διαφορών και φωλιασμένου υπολογισμού της τιμής του πολυωνύμου δίνεται στο Παράρτημα.

ex2.py

```
import numpy as np
import matplotlib.pyplot as plt
import interpolation


def f(x):
    return x + np.sin(x)


def er(x):
    return x * (x - .5) * (x - 1) * np.sin(x) / np.math.
    factorial(4)


xValues = np.array([0, 0.5, 1, 1.5])
yValues = f(xValues)

coefficients = interpolation.dividedDifferenceTable(xValues,
    yValues)
p = interpolation.NestedMultiplication(0, xValues, coefficients)

plt.close('all')
x = np.arange(0, 1.6, 0.1)
y = interpolation.NestedMultiplication(x, xValues, coefficients)
plt.figure(1)
plt.plot(x, f(x), '.', label='x + sinx')
plt.plot(x, y, '--', label='Newton')
plt.plot(xValues, yValues, '*')
plt.legend()
plt.grid()


x = np.arange(0, np.pi/2, 0.01)
err = np.abs(er(x))
plt.figure(2)
```

```
plt.plot(x, err, label='|f(x)-p(x)|')
plt.legend()
plt.grid()
error = (np.power(np.pi, 3) - 3 * np.power(np.pi, 2) + 2 * np.pi
    ) / 192
print('Theoretical Error:', error)
print('Graphical Error:', max(err))
```

3η Άσκηση

Παρατηρείται ότι το πολυώνυμο δίνεται σε φωλιασμένη μορφή άρα είναι πολυώνυμο Newton. Επομένως, μπορεί πολύ εύκολα να υπολογιστεί ένα νέο πολυώνυμο που να παρεμβάλει και το πέμπτο σημείο.

Συγκεκριμένα, αν $p_3 = 2 - (x+1) + x(x+1) - 2x(x+1)(x-1)$ τρίτου βαθμού πολυώνυμο, τότε το νέο πολυώνυμο $p_4(x)$ θα είναι της μορφής (7) και θα είναι τέταρτου βαθμού.

$$p_4(x) = p_3(x) + cx(x+1)(x-1)(x-2) \tag{7}$$

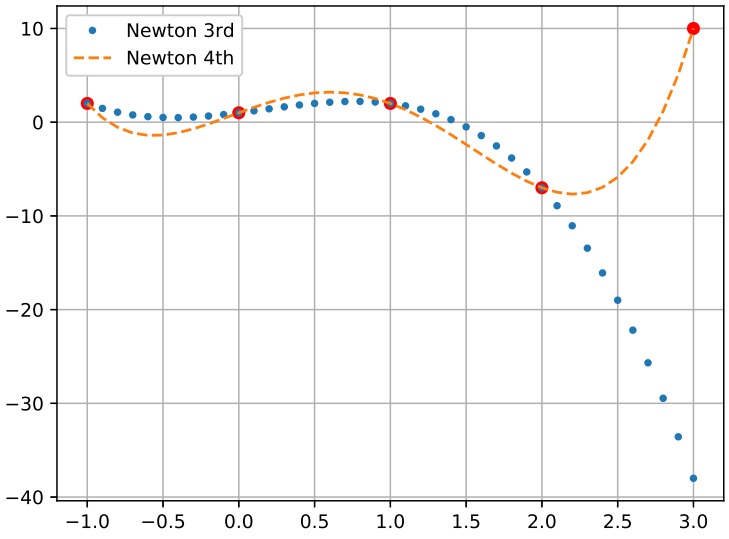
Ο προσδιορισμός της σταθεράς c γίνεται μέσω της τιμής του νέου πολυωνύμου στο επιπλέον σημείο (8).

Το υπολογιστικό φορτίο που απαιτείται για να προστεθεί ένας επιπλέον όρος είναι μικρό: 8 πράξεις κινητής υποδιαστολής (4 προσθαφαιρέσεις, 3 πολλαπλασιασμοί και 1 διαίρεση). Εδώ φαίνεται η υπεροχή της χρήσης των πολυωνύμων Newton σε σχέση με τα πολυώνυμα Lagrange, για τα οποία θα χρειαζόταν ο υπολογισμός όλου του πολυωνύμου εξαρχής.

$$\begin{aligned} p_4(x_4) = 10 \rightarrow \\ y_4 = p_3(x_4) + cx_4(x_4+1)(x_4-1)(x_4-2) \rightarrow \\ c = \frac{y_4 - p_3(x_4)}{x_4(x_4+1)(x_4-1)(x_4-2)} \end{aligned} \tag{8}$$

Στο Σχήμα 4 δίνονται τα $p_3(x)$ και $p_4(x)$. Όπως είναι αναμενόμενο, τα δύο πολυώνυμα είναι σχετικά κοντά μέχρι το $(2, -7)$ και αποκλίνουν σημαντικά

μετά από αυτό.



Σχήμα 4: Παρεμβολή με πολυώνυμα Newton 3ου και 4ου βαθμού

Παρακάτω ακολουθεί ο κώδικας που γράφτηκε σε Python και έγινε χρήση της βιβλιοθήκης Numpy. Η υλοποίηση των αλγορίθμων προσθήκης νέου όρου στο πολυώνυμο Newton και φωλιασμένου υπολογισμού της τιμής του πολυωνύμου δίνονται στο Παράρτημα.

ex3.py

```
import numpy as np
import matplotlib.pyplot as plt
import interpolation

xValues = np.array([-1, 0., 1, 2, 3])
yValues = np.array([2., 1, 2, -7, 10])
coefficients = np.array([2., -1, 1, -2])

plt.close('all')
x = np.arange(-1, 3.1, 0.1)
y = interpolation.NestedMultiplication(x,
```

```
                                        xValues,
                                        coefficients)

plt.plot(xValues, yValues, 'ro')  # Plot points
plt.plot(x, y, '.', label='Newton 3rd')  # Plot 3rd degree
    Newton

newCoefficients = interpolation.addCoefficient(xValues,
                                               yValues,
                                               coefficients)
print('Order', newCoefficients.size - 1,
      'coefficent is', newCoefficients[-1])

y = interpolation.NestedMultiplication(x,
                                        xValues,
                                        newCoefficients)
plt.plot(x, y, '--', label='Newton 4th')  # Plot 4th degree
    Newton
plt.legend()
plt.grid()
```

4η Άσκηση

Ζητούμενο είναι να προεγγιστούν οι συναρτήσεις $f_1(x) = x^2 - 2x + 3$, $f_2(x) = \cos(\pi x)$ και $f_3(x) = e^{-x}$ με πολυώνυμο δευτέρου βαθμού, οι συντελεστές των όρων του οποίου υπολογίζονται με χρήση της μεθόδου ελαχίστων τετραγώνων.

Έστω ότι το πολυώνυμο που προκύπτει από την μέθοδο ελαχίστων τετραγώνων δίνεται από την εξίσωση 9, όπου $\{\phi_i(x)\}$ οικογένεια πολυωνύμων που θα χρησιμοποιηθούν σαν βάση για την προσέγγιση. Εν προκειμένω, ζητείται ο βαθμός του πολυωνύμου να είναι $n = 2$, άρα ο βαθμός του $\phi_n(x)$ θα είναι το πολύ 2.

$$p_n(x) = a_0 + a_1\phi_1(x) + \cdots + a_n\phi_n(x) \tag{9}$$

Υπάρχουν διάφορες επιλογές για την οικογένεια των πολυωνύμων δευτέρου βαθμού. Η πιο απλή επιλογή είναι τα μονώνυμα $\{1, x, x^2\}$ αλλά υπάρχουν και

ορθογώνια πολυώνυμα όπως τα πολυώνυμα Legendre $\{1, x, x^2 - \frac{1}{3}\}$.

Η μέθοδος ελαχίστων τετραγώνων ελαχιστοποιεί το σφάλμα (10).

$$E = \int_a^b [f(x) - P_2(x)]^2 \, dx \tag{10}$$

Η προσέγγιση με μονώνυμα για το διάστημα $[a, b]$ απαιτεί επίλυση του συστήματος (11), όπου $s_i = \int_a^b x^i dx$ και $b_i = \int_a^b x^i f(x) dx$.

$$\begin{bmatrix} s_0 & s_1 & s_2 \\ s_1 & s_2 & s_3 \\ s_2 & s_3 & s_4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} \tag{11}$$

Για $[a, b] = [0, 1]$ ο πίνακας είναι ο (12), ο πίνακας Hilbert, ο οποίος είναι αντιστρέψιμος για $n = 3$, όμως για μεγαλύτερα n (πχ 5) οδηγεί σε ασταθή συστήματα.

$$S = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix} \tag{12}$$

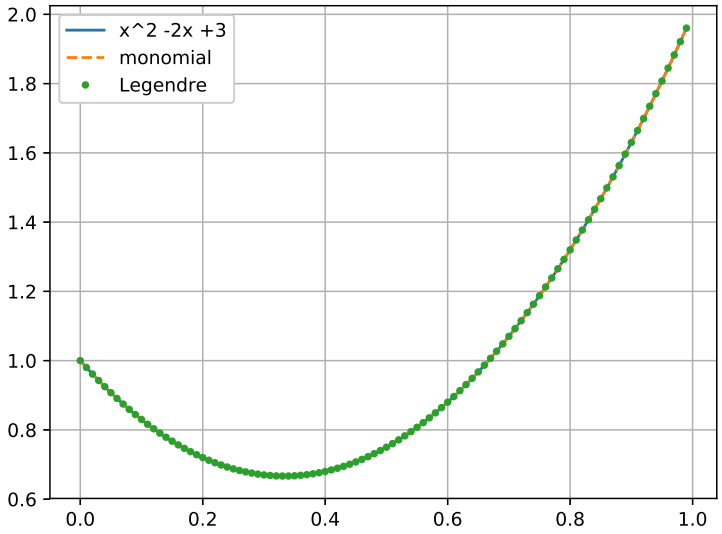
Για το λόγο αυτό, σε μεγαλύτερα n χρησιμοποιούνται ορθογώνια πολυώνυμα, όπως τα Legendre. Τα ορθογώνια πολυώνυμα μηδενίζουν όλα τα μη διαγώνια στοιχεία του Πίνακα 11 και έτσι η επίλυσή του είναι άμεση. Συγκεκριμένα, είναι $s_i = \int_a^b w(x)\phi_i^2(x) dx$ και $b_i = \int_a^b w(x)\phi_i(x) f(x) dx$, όπου $w(x)$ κατάληλλη συνάρτηση βάρους και ίση με $w(x) = 1$ για τα πολυώνυμα Legendre. Σε αντιδιαστολή με την περίπτωση των μονωνύμων, η ολοκλήρωση με πολυώνυμα Legendre πρέπει να γίνει στο διάστημα $[-1, 1]$, για να ισχύει η σχέση ορθογωνιότητας (13). Γενικότερα, η ολοκλήρωση πρέπει να γίνει σε διάστημα στο οποίο θα ικανοποιείται η σχέση ορθογωνιότητας. Το ζητούμενο διάστημα προσέγγισης εξακολουθεί να είναι το $[0, 1]$ καθώς είναι υποσύνολο του $[-1, 1]$.

$$\int_{-1}^{1} \phi_i(x)\phi_j(x) dx = 0 \tag{13}$$

Για σύγκριση, το πρόβλημα λύνεται υπολογιστικά και με τις δύο προσεγγίσεις. Και στις δύο περιπτώσεις, αναμένεται για κάθε συνάρτηση $f_k(x)$ να

παραχθεί ένα πολυώνυμο δευτέρου βαθμού. Ειδικότερα, η $f_1(x)$ είναι ήδη ένα πολυώνυμο δευτέρου βαθμού και επομένως αναμένεται η μέθοδος ελαχίστων τετραγώνων να παράγει σαν πολυώνυμο την ίδια την $f_1(x)$.

Στα Σχήματα 5, 6 και 7 φαίνονται τα αποτελέσματα της προσέγγισης. Όπως ήταν αναμενόμενο, η μέθοδος ελαχίστων τετραγώνων παράγει την ίδια την $f_1(x)$ στην πρώτη περίπτωση. Επίσης, στην δεύτερη περίπτωση, η λύση με μονώνυμα είναι μια ευθεία σε αντίθεση με την λύση με πολυώνυμα Legendre, που είναι δευτέρου βαθμού.

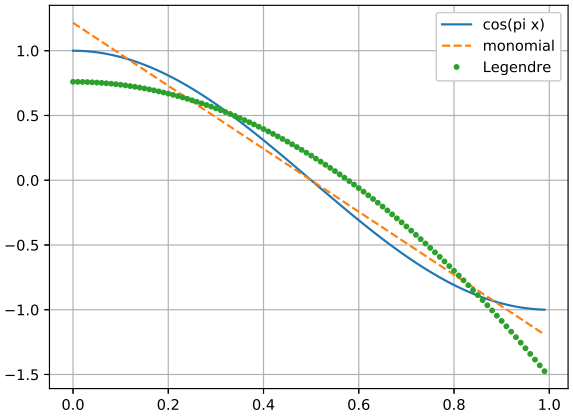


Σχήμα 5: Ελάχιστα Τετράγωνα για $x^2 - 2x + 3$

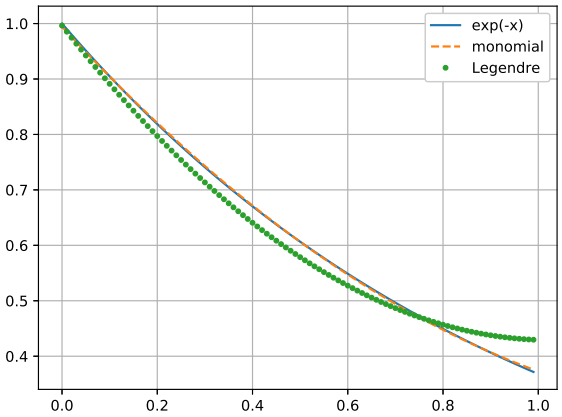
Παρακάτω ακολουθεί ο κώδικας που γράφτηκε σε Python και έγινε χρήση της βιβλιοθήκης Numpy. Οι ρουτίνες ελαχίστων τετραγώνων με μονώνυμα και με πολυώνυμα Legendre δίνονται στο Παράρτημα.

ex4.py

```
import numpy as np
import matplotlib.pyplot as plt
import interpolation
```



Σχήμα 6: Ελάχιστα Τετράγωνα για $\cos(\pi x)$



Σχήμα 7: Ελάχιστα Τετράγωνα για e^{-x}

```
4
def f1(x):
    """Evaluates x^2-2x+3"""
    return np.polyval([3, -2, 1], x)
9

def f2(x):
    """Evaluates cos(pi * x)"""
    return np.cos(np.pi*x)
14

def f3(x):
    """Evaluates e^(-x)"""
    return np.exp(-x)
19

def Legendre(x, n):
    """Evaluates Legendre polynomial
    of n degree"""
24  c = np.zeros(n + 1)
    c[n] = 1.
    return np.polynomial.legendre.legval(x, c)

29 def wLegendre(x):
    """Weight function for Legendre
    polynomials used in Least Squares"""
    return np.ones(x)

34

a = 0
b = 1
order = 2

39 x = np.arange(a, b, 0.01)

plt.close('all')

p1 = interpolation.lSquaresMonomial(f1, order, a, b)
```

```
a1 = interpolation.lSquaresOrthogonal(f1,
                                      wLegendre,
                                      Legendre,
                                      3,
                                      -1,
                                      1)  # Base Legendre
    polynomials
print("Monomial:", p1)
print("Legendre:", a1)
plt.figure(1)
plt.plot(x, f1(x), label='x^2 -2x +3')
plt.plot(x, p1(x), '--', label='monomial')
plt.plot(x, a1(x), '.', label='Legendre')
plt.legend()
plt.grid()

p2 = interpolation.lSquaresMonomial(f2, order, a, b)
a2 = interpolation.lSquaresOrthogonal(f2,
                                      wLegendre,
                                      Legendre,
                                      3,
                                      -1,
                                      1)  # Base Legendre
    polynomials
print("\nMonomial:", p2)
print("Legendre:", a2)
plt.figure(2)
plt.plot(x, f2(x), label='cos(pi x)')
plt.plot(x, p2(x), '--', label='monomial')
plt.plot(x, a2(x), '.', label='Legendre')
plt.legend()
plt.grid()

p3 = interpolation.lSquaresMonomial(f3, order, a, b)
a3 = interpolation.lSquaresOrthogonal(f3,
                                      wLegendre,
                                      Legendre,
                                      3,
                                      -1,
                                      1)  # Base Legendre
```

```
    polynomials
print("\nMonomial:", p3)
print("Legendre:", a3)
plt.figure(3)
plt.plot(x, f3(x), label='exp(-x)')
plt.plot(x, p3(x), '--', label='monomial')
plt.plot(x, a3(x), '.', label='Legendre')
plt.legend()
plt.grid()
```

5η Άσκηση

Τα σημεία της συνάρτησης είναι $\{0.25, 0.5, 0.75\}$, δηλαδή ισαπέχοντα. Για την προσέγγιση των παραγώγων πρώτης και δεύτερης τάξης, μπορεί να εφαρμοστούν οι τύποι διαφορών (14) και (15) για $n = 2$. Εν προκειμένω, είναι $h = 0.25$ το βήμα και $x = x_1$ το σημείο υπολογισμού των παραγώγων.

$$f'(x) \approx \frac{1}{h}\left[\Delta f_0 + \frac{1}{2}(2\theta - 1)\Delta^2 f_0 + \cdots + \frac{d}{d\theta}\binom{\theta}{n}\Delta^n f_0\right] \tag{14}$$

$$f''(x) \approx \frac{1}{h^2}\left[\Delta^2 f_0 + (\theta - 1)\Delta^3 f_0 + \cdots + \frac{d^2}{d\theta^2}\binom{\theta}{n}\Delta^n f_0\right] \tag{15}$$

Τελικά οι τύποι της πρώτης και δεύτερης παραγώγου δίνονται από τις εξισώσεις (16) και μπορούν εύκολα να γραφεί πρόγραμμα σε υπολογιστή.

$$\begin{aligned} f'(x) &\approx \frac{1}{h}\left(\Delta f_0 + \frac{1}{2}(2\theta - 1)\Delta^2 f_0\right) \\ f''(x) &\approx \frac{1}{h^2}\Delta^2 f_0 \end{aligned} \tag{16}$$

Μετά από εκτέλεση του προγράμματος προκύπτουν τα νούμερα του Πίνακα 1. Καταρχάς, οι πραγματικές τιμές της πρώτης και της δεύτερης παραγώγου διαφέρουν μόνο ως προς το πρόσημο, λόγω του ότι η $f(x)$ είναι η εκθετική συνάρτηση. Το σφάλμα είναι γενικά μικρό αλλά μπορεί να εκτιμηθεί και το άνω φράγμα του.

	Πραγματική	Αριθμητική
$f'(0.5)$	-0.6065	-0.6128
$f''(0.5)$	0.6065	0.6096

Πίνακας 1: Σύγκριση τιμών παραγώγων

Από θεωρία, είναι γνωστή η σχέση (17), που προέρχεται από την αντίστοιχη σχέση σφάλματος στην παρεμβολή πολυωνύμων και μπορεί να ληφθεί η απόλυτη τιμή της.

$$E_n(x_k) = \prod_{i=0, i\neq k}^{n} (x_k - x_i) \frac{f^{(n+1)\xi}}{(n+1)!} \tag{17}$$

Ο τελικός υπολογισμός για το φράγμα της πρώτης παραγώγου δίνεται από την (18). Το άνω φράγμα της $|e^{-\xi}|$ προέκυψε $e^{-0.25}$ γιατί η συνάρτηση είναι γνησίως φθίνουσα και στο διάστημα $(0.25, 0.75)$ θα είναι πάντα $|e^{-\xi}| < e^{-0.25}$.

$$\begin{aligned} |E_2(0.25)| &= |x_1 - x_0||x_1 - x_2| \frac{|e^{-\xi}|}{3!} \rightarrow \\ &< \frac{e^{-0.25}|x_1 - x_0||x_1 - x_2|}{6} \rightarrow \\ &\approx 0.008 \end{aligned} \tag{18}$$

Για την δεύτερη παράγωγο, ο αντίστοιχος τύπος του (17) είναι πολύπλοκος. Ωστόσο, μπορεί να παρατηρηθεί ότι, η $f''(x) = f(x)$ και επομένως σε αυτή την περίπτωση θα ισχύει ο τύπος σφάλματος παρεμβολής (19). Λαμβάνοντας την απόλυτη τιμή του τελευταίου υπολογίζεται η (20).

$$f(x) - p_2(x) = \frac{(x - x_0)(x - x_1)(x - x_2)}{(3)!} f^{(3)}(\xi) \tag{19}$$

$$
\begin{aligned}
|f(x) - p_2(x)| &= \left| \frac{(x-0.25)(x-0.5)(x-0.75)}{3!} e^{-\xi} \right| \rightarrow \\
&< \frac{|x-0.25||x-0.5||x-0.75|}{6} e^{-0.25} \rightarrow \\
&< \frac{\frac{1}{2}\frac{1}{2}\frac{1}{4}}{6} e^{-0.25} \rightarrow \\
&= \frac{e^{-0.25}}{96} \approx 0.008
\end{aligned}
\tag{20}
$$

Μπορεί το φράγμα του σφάλματος και στις δύο περιπτώσεις να είναι περίπου ίδιο, αλλά υπάρχει ένα λεπτό σημείο. Στην περίπτωση της πρώτης παραγώγου, το φράγμα για το σφάλμα δίνεται για το $x = x_1$ μόνο, ενώ στην δεύτερη παράγωγο για όλο το διάστημα $[0.25, 0.75]$.

Παρακάτω ακολουθεί ο κώδικας που γράφτηκε σε Python και έγινε χρήση της βιβλιοθήκης Numpy. Οι ρουτίνες υπολογισμού εμπρός διαφορών και αριθμητικών παραγώγων δίνονται στο Παράρτημα.

ex5.py

```
import numpy as np
import matplotlib.pyplot as plt
import interpolation


def f(x):
    return np.exp(-x)


def df(x):
    return -np.exp(-x)


def d2f(x):
    return f(x)


def err2(x):
```

```
    return np.exp(-x) * (x-0.25) *(x-0.5) *(x-0.75) /6


def err1(x):
    return np.exp(-x) * (0.5-0.25) *(0.5-0.75) /6

xValues = np.array([0.25, 0.5, 0.75])
fValues = f(xValues)
dValues = interpolation.differenceTable(fValues)


print('Actual df(0.5)=', df(0.5))
print('Numeric df(0.5)', interpolation.df(0.5, xValues, dValues)
    )
print('Actual d2f(0.5)=', d2f(0.5))
print('Numeric d2f(0.5)', interpolation.d2f(0.5, xValues,
    dValues))

error = f(xValues[0]) / 6 * \
        np.abs((xValues[1]-xValues[0]) * (xValues[1] - xValues
    [2]))
print('Max error is', error)

x = np.arange(0.25, 0.76, 0.01)
```

Παραρτήματα

Κώδικας Υπολογισμών Παρεμβολής-Ελαχίστων Τετραγώνων

interpolation.py

```
import numpy as np
import scipy.integrate as integrate


def df(x, xValues, dValues):
    """Derivative using partial differences
    second order"""
    h = xValues[1] - xValues[0]
```

```
    theta = (x - xValues[0]) / h
    return (dValues[1] + 0.5 * (2*theta - 1)*dValues[2]) / h


def d2f(x, xValues, dValues):
    """Second Derivative using partial differences
    second order"""
    h = xValues[1] - xValues[0]
    return dValues[2] / np.power(h, 2)


def differenceTable(fxValues):
    """Calculates difference table and returns its
    top diagonal"""
    n = fxValues.size
    dValues = np.copy(fxValues)
    for i in range(1, n):
        for j in reversed(range(i, n)):
            dValues[j] = dValues[j] - dValues[j-1]
    return dValues


def dividedDifferenceTable(xValues, fxValues):
    """Calculates divided difference table and returs its
    top diagonal"""
    n = fxValues.size
    dValues = np.copy(fxValues)
    for i in range(1, n):
        for j in reversed(range(i, n)):
            dValues[j] = (dValues[j] - dValues[j-1]) / \
                         (xValues[j] - xValues[j-i])
    return dValues


def addCoefficient(xValues, fxValues, coefficients):
    """Adds extra Newton Polynomial Coefficient"""
    n = coefficients.size
    x = xValues[n]  # Get x of new point to interpolate
    y_x = fxValues[n]  # Get y of new point to interpolate
```

```
    p_x = NestedMultiplication(x, xValues, coefficients)  #
    evaluate p(x)
    product = 1.
    for i in range(n):
        product = product * (x - xValues[i])  # (x-x0)...(x-x_n
    -1)
    newCoefficient = (y_x - p_x) / product  # new coefficient
    return np.append(coefficients, newCoefficient)


def NestedMultiplication(x, xValues, coeff):
    """Evaluates Newton Polynomial at x in nested form
    given the interpolating points and its coefficents"""
    n = coeff.size
    y = coeff[n-1]
    for i in reversed(range(n - 1)):
        y = coeff[i] + (x - xValues[i]) * y
    return y


def lSquaresMonomial(f, order, a, b):
    """Solves least squares with monomial polynomials"""
    n = order + 1
    S = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            S[i, j] = integrate.quad(lambda x: np.power(x, i+j),
             a,
             b)[0]
    y = np.zeros(n)
    for i in range(n):
        y[i] = integrate.quad(lambda x: f(x) * np.power(x, i),
         a,
         b)[0]
        # Round to 0 if less than 1e-8
        y[i] = 0. if np.isclose(y[i], 0., atol=1e-8) else y[i]
    p = np.linalg.solve(S, y)
    return np.poly1d(np.flip(p, 0))
```

```
def lSquaresOrthogonal(f, w, phi, order, a, b):
    """Solves least squares system using orthogonal
    polynomials with weight w(x) and base phi(x)"""
    c = np.zeros(order)
    for i in range(order):
        c[i] = integrate.quad(lambda x: np.power(phi(x, i), 2),
         a,
         b)[0]
    A = np.zeros(order)
    for i in range(order):
        A[i] = integrate.quad(lambda x: f(x) * phi(x, i),
         a,
         b)[0] / c[i]
    legpoly = np.polynomial.legendre.leg2poly(A)
    return np.poly1d(np.flip(legpoly, 0))
```

Εκτέλεση Προγραμμάτων

```
Python 3.6.3 |Anaconda custom (64-bit)| (default, Nov  8 2017, 15:10:56) [MSC v.1900 64
bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

Restarting kernel...
```

```
In [1]: runfile('C:/Users/tzave/ComputationalPhysics/Assignment3/ex1.py', wdir='C:/Users/
tzave/ComputationalPhysics/Assignment3')
Function Value f(1.25)= 11.1824939607
Lagrange polyn p(1.25)= 11.1824517251

In [2]: runfile('C:/Users/tzave/ComputationalPhysics/Assignment3/ex2.py', wdir='C:/Users/
tzave/ComputationalPhysics/Assignment3')
Theoretical Error: 0.0400033790844
Graphical Error: 0.0398976123497

In [3]: runfile('C:/Users/tzave/ComputationalPhysics/Assignment3/ex3.py', wdir='C:/Users/
tzave/ComputationalPhysics/Assignment3')
Reloaded modules: interpolation
Order 4 coefficent is 2.0

In [4]: runfile('C:/Users/tzave/ComputationalPhysics/Assignment3/ex4.py', wdir='C:/Users/
tzave/ComputationalPhysics/Assignment3')
Reloaded modules: interpolation
Monomial:    2
3 x - 2 x + 1
Legendre:    2
3 x - 2 x + 1

Monomial:             2
-2.929e-14 x - 2.432 x + 1.216
Legendre:        2
-2.28 x + 0.7599

Monomial:         2
0.3087 x - 0.9305 x + 0.9945
Legendre:         2
0.5367 x - 1.104 x + 0.9963

In [5]: runfile('C:/Users/tzave/ComputationalPhysics/Assignment3/ex5.py', wdir='C:/Users/
tzave/ComputationalPhysics/Assignment3')
Reloaded modules: interpolation
Actual df(0.5)= -0.606530659713
Numeric df(0.5) -0.612868460661
Actual d2f(0.5)= 0.606530659713
Numeric d2f(0.5) 0.609696262194
Max error is 0.00811250815699

In [6]:
```