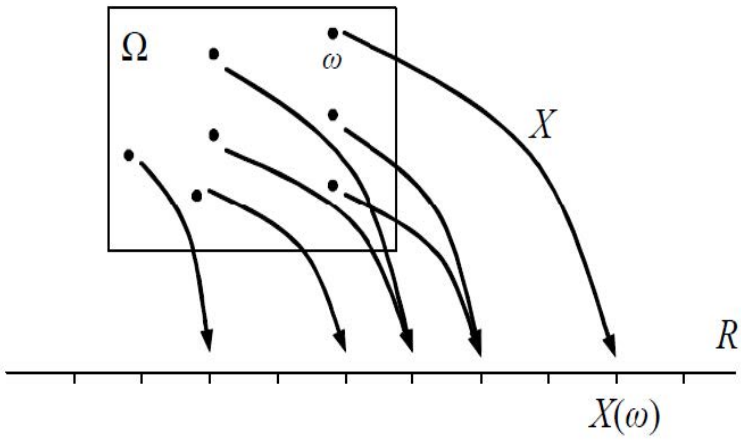
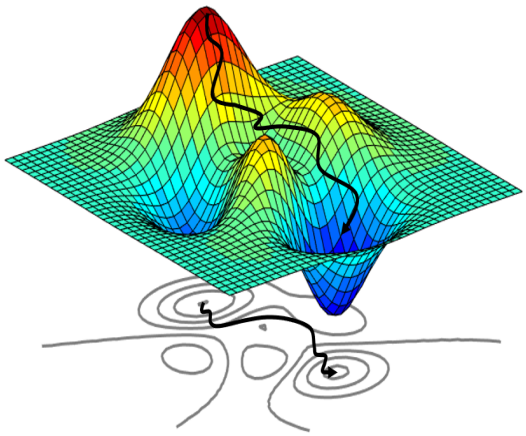
Αναγνώριση Προτύπων

2021-2022

Εργασία 1η

ΕΙΣΑΓΩΓΗ ΣΤΙΣ ΤΥΧΑΙΕΣ ΜΕΤΑΒΛΗΤΕΣ ΚΑΙ ΤΗΝ ΒΕΛΤΙΟΣΤΟΠΟΙΗΣΗ





Καρυπίδης Ευστάθιος 57556

6/11/21, Ξάνθη

Για την υλοποίηση των ερωτημάτων της εργασίας έγινε χρήση γλώσσας python και των βιβλιοθηκών numpy,scipy και matplotlib.

Άσκηση 1η

Ερώτημα α

Για ένα απλό φυσιολογικό ζάρι $\mathrm{Z1}$ με 6 πλευρές και ως $z1$ μια διακριτή τυχαία μεταβλητή που συμβολίζει το αποτέλεσμα της ρίψης του $\mathrm{Z1}$ έχουμε ότι ο δειγματοχώρος Ω είναι:

$$\Omega = \{1, 2, 3, 4, 5, 6\}$$

Τα στοιχεία του δειγματοχώρου Ω έχουν ίσες πιθανότητες, οπότε οι τιμές της τυχαίας μεταβλητής $z1$ είναι 1,2,3,4,5,6 με ίσες πιθανότητες δηλαδή

$z1$	1	2	3	4	5	6
P($Z = z1$)	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

Άρα η συνάρτηση πυκνότητας πιθανότητας(PDF) ή πιο σωστά η συνάρτηση μάζας πιθανότητας(PMF) εφόσον αναφερόμαστε σε διακριτή τυχαία μεταβλητή είναι

$$f(z1) = \frac{1}{6} \; \forall \; z1 = 1, 2, 3, 4, 5, 6$$

Ερώτημα β

Για την μέση τιμή γνωρίζουμε ότι:

$$E[Z] = \sum_k z_k P(z_k) = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = \frac{21}{6} = 3.5$$

Για την μεταβλητότητα γνωρίζουμε ότι:

$$Var[Z] = \sigma^2 = E[(Z - E[Z])^2] = E[Z^2] - E[Z]^2$$

$$E[Z^2] = \sum_k z_k^2 P(z_k) = 1^2 \cdot \frac{1}{6} + 2^2 \cdot \frac{1}{6} + 3^2 \cdot \frac{1}{6} + 4^2 \cdot \frac{1}{6} + 5^2 \cdot \frac{1}{6} + 6^2 \cdot \frac{1}{6} = \frac{91}{6}$$

Άρα τελικά

$$Var[Z] = E[Z^2] - E[Z]^2 = \frac{91}{6} - (\frac{21}{6})^2 = \frac{91}{6} - \frac{441}{36} = \frac{105}{36} = 2.91666667$$

Ερώτημα γ

Για το skewness γνωρίζουμε ότι:

$$\text{skewness} = \frac{c_3}{c_2^{\frac{3}{2}}} = E[(\frac{z - \mu}{\sigma})^3]$$

Γενικά για μία ροπή n τάξης έχουμε ότι:

$$c_n = \sum_k (z_k - \mu_1)^n P(z_k)$$

Η ροπή 2ης τάξης είναι η μεταβλητότητα που υπολογίστηκε προηγουμένως. Για την ροπή τρίτης τάξης έχουμε:

$$\sum_k z_k^3 P(z_k) = (1-3.5)^3 \cdot \frac{1}{6} + (2-3.5)^3 \cdot \frac{1}{6} + (3-3.5)^3 \cdot \frac{1}{6} +$$

$$+(4-3.5)^3 \cdot \frac{1}{6} + (5-3.5)^3 \cdot \frac{1}{6} + (6-3.5)^3 \cdot \frac{1}{6} = 0$$

Άρα τελικά:

$$\text{skewness} = \frac{c_4}{c_2^2} = \frac{0}{2.91666667^{\frac{3}{2}}} = 0$$

αντίστοιχα για το kurtosis γνωρίζουμε ότι:

$$\text{kurtosis} = \frac{c_3}{c_2^{\frac{3}{2}}} = E[(\frac{z-\mu}{\sigma})^4]$$

$$\sum_k z_k^4 P(z_k) = (1-3.5)^4 \cdot \frac{1}{6} + (2-3.5)^4 \cdot \frac{1}{6} + (3-3.5)^4 \cdot \frac{1}{6} +$$

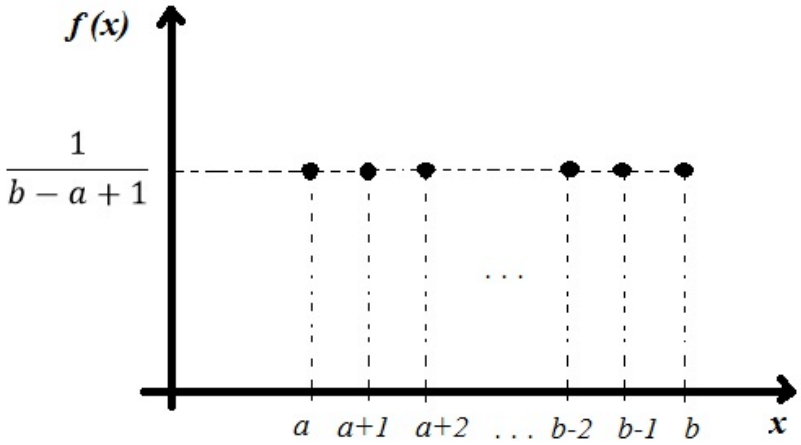
$$+(4-3.5)^4 \cdot \frac{1}{6} + (5-3.5)^4 \cdot \frac{1}{6} + (6-3.5)^4 \cdot \frac{1}{6} = 14.7291666667$$

Άρα τελικά:

$$\text{kurtosis} = \frac{c_3}{c_2^2} = \frac{14.7291666667}{2.91666667^2} = 1.7314285\ldots$$

Εναλλακτική λύση

Τα αποτελέσματα αυτά μπορούν να βρεθούν και να επαληθευτούν εαν εκμεταλλευτούμε το γεγονός πως στην περίπτωση του απλού φυσιολογικού ζαριού $\mathrm{Z1}$ με 6 πλευρές και της τυχαίας μεταβλητής $z1$ έχουμε ομοιόμορφη διακριτή κατανομή.



Στη δική μας περίπτωση ισχυεί πως $a = 1, b = 6$ και άρα $n = b - a + 1 = 6$

- Για την PMF έχουμε:

$$f(z1) = \frac{1}{n} = \frac{1}{6}$$

- Για τη μέση τιμή και την μεταβλητότητα έχουμε:

$$E[z] = \frac{\alpha + b}{2} = \frac{1 + 6}{2} = 3.5$$
$$\mathrm{Var}[z] = \frac{n^2 - 1}{12} = \frac{35}{12} = 2.91667$$

- Για skewness και kurtosis έχουμε:

$$\mathrm{skewness} \ = 0$$
$$\mathrm{kurtosis} \ = -\frac{6 \cdot \left(n^2 + 1\right)}{5 \cdot \left(n^2 - 1\right)} + 3 = -\frac{6 \cdot 37}{5 \cdot 35} + 3 = -1.2685714 + 3 = 1.7314285\ldots$$

Άσκηση 2

Ερώτημα α

Για την παραγωγή των τυχαίων ρίψεων από το ζάρι χρησιμοποιήθηκε η ακόλουθη συνάρτηση της python

numpy.random.Generator.integers

method

random.Generator.integers(*low, high=None, size=None, dtype=np.int64, endpoint=False*)

Return random integers from *low* (inclusive) to *high* (exclusive), or if endpoint=True, *low* (inclusive) to *high* (inclusive). Replaces *RandomState.randint* (with endpoint=False) and *RandomState.random_integers* (with endpoint=True)

Return random integers from the "discrete uniform" distribution of the specified dtype. If *high* is None (the default), then results are from 0 to *low*.

Δημιούργησα τη συνάρτηση die_rolls όπου σαν όρισμα δέχεται το πόσες ρίψεις θέλουμε και επιστρέφει έναν numpy array με τις ρίψεις του ζαριού.

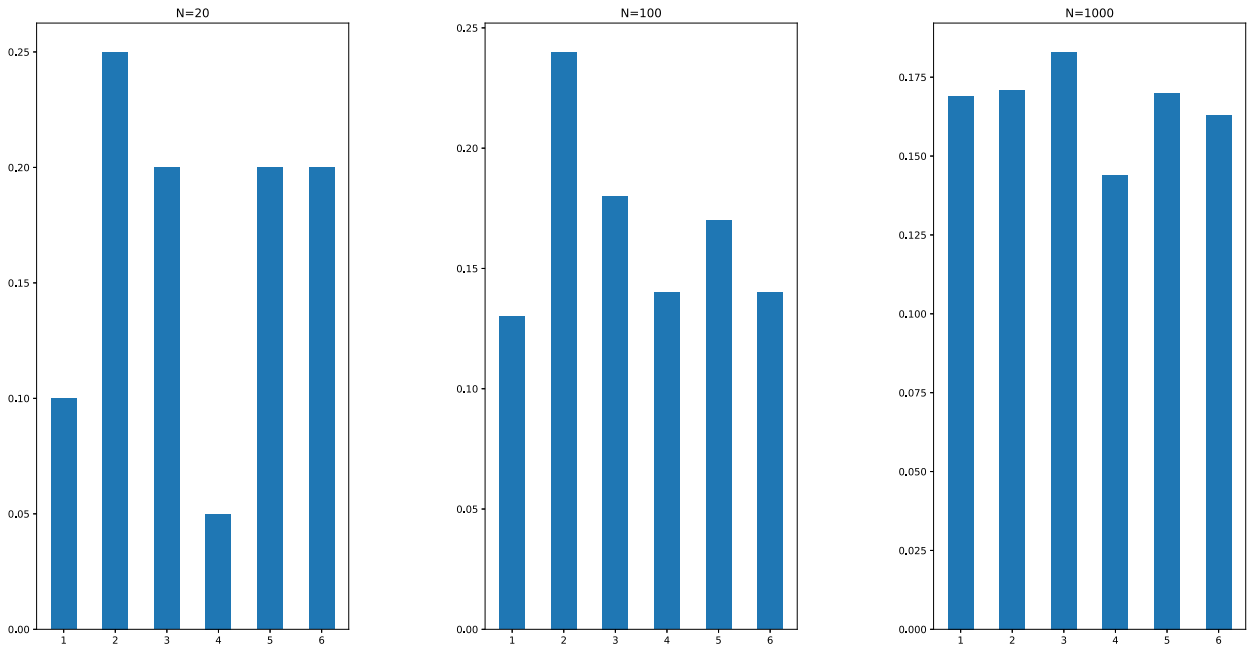
```
def die_rolls(num_of_throws):
rng = np.random.default_rng(seed=123)
total_throws = rng.integers(low=1, high=6, endpoint=True, size=num_of_throws)
return total_throws
```

Το seed ορίστηκε τυχαία στο 123 με στόχο την επαναληψιμότητα των πειραμάτων.

Ερώτημα β

Με τις παρακάτω εντολές παράγουμε και απεικονίζουμε τη προσεγγιστική συνάρτηση (πυκνότητας) μάζας πιθανότητας των ρίψεων του ζαριού.

```
# Ερώτημα β Άσκηση 2
n_rolls_a = [20, 100, 1000]
throws_list_a = [die_rolls(n) for n in n_rolls_a]
print(throws_list_a)
fig1, axes1 = plt.subplots(nrows=1, ncols=len(n_rolls_a))
for ax in range(len(axes1)):
axes1[ax].hist(throws_list_a[ax], bins=[i+0.5 for i in range(0, 7)],
rwidth=0.5)
axes1[ax].set_title("N="+str(n_rolls_a[ax]))
fig1.tight_layout()
plt.show()
```



Όπως παρατηρούμε για 20 ρίψεις δε φαίνεται να ακολουθείται κάποια κατανομή. Για 100 ρίψεις και πάλι δεν ειναι πολύ ευδιάκριτη η κατανομή ωστόσο υπάρχουν πιο μικρές διαφορές στις πιθανότητες. Για 1000 ρίψεις φαίνεται να σχηματίζεται καλά η ομοιόμορφη διακριτή κατανομή με τις πιθανότητες για κάθε ρίψη να είναι αρκετά κοντά στις θεωρητικές.

Ερώτημα γ

Μέσω των συναρτήσεων mean, var της numpy και skew,kurtosis της scipy υπολογίζονται τα αντίστοιχα στατιστικά. Ιδιαίτερη προσοχή θέλει η συνάρτηση kurtosis η οποία για να βγάλει επιθυμητά αποτελέσματα πρέπει να δώσουμε την τιμή False στο όρισμα fisher καθώς διαφορετικά έχει default παράμετρο να υπολογίζει την Excess Kurtosis δηλαδή την kurtosis που υπολογίζουμε εμείς -3(δηλαδή για να έχει η κανονική κατανομή kurtosis 0 και όχι 3). Συνεπώς για επιθυμητά αποτελέσματα χρησιμοποιείται fisher=False και χρησιμοποιείται ο ορισμός του Pearson.

scipy.stats.kurtosis

`scipy.stats.kurtosis(a, axis=0, fisher=True, bias=True, nan_policy='propagate')` [source]

Compute the kurtosis (Fisher or Pearson) of a dataset.

Kurtosis is the fourth central moment divided by the square of the variance. If Fisher's definition is used, then 3.0 is subtracted from the result to give 0.0 for a normal distribution.

```
# Ερώτημα γ Άσκηση 2
n_rolls_c = [10, 20, 50, 100, 500, 1000]
throws_list_c = [die_rolls(n) for n in n_rolls_c]
mean_values = [np.mean(thl) for thl in throws_list_c]
variance_values = [np.var(thl) for thl in throws_list_c]
skewness_values = [skew(thl) for thl in throws_list_c]
kurtosis_values = [kurtosis(thl, fisher=False) for thl in throws_list_c]
for i in range(len(n_rolls_c)):
    print(30*"==")
    print(str(n_rolls_c[i])+" die rolls results:")
    print("Mean value: ", format(mean_values[i], '.4f'), "Difference from
theoretical value: ", format(abs(3.5-mean_values[i]), '.4f'))
    print("Variance: ", format(variance_values[i], '.4f'), "Difference from
theoretical value: ", format(abs(2.9166-variance_values[i]), '.4f'))
    print("Skewness is", format(skewness_values[i], '.4f'), "Difference from
theoretical value: ", format(abs(skewness_values[i]), '.4f'))
    print("Kurtosis is", format(kurtosis_values[i], '.4f'), "Difference from
theoretical value: ", format(abs(1.731428-kurtosis_values[i]), '.4f'))
    print(30*"==")
fig, axes = plt.subplots(nrows=2, ncols=3)
for ax0 in range(len(axes)):
    for ax1 in range(len(axes[0])):
        axes[ax0][ax1].hist(throws_list_c[3*ax0+ax1], bins=[i+0.5 for i in range(0,
7)], rwidth=0.5, density=True)
        axes[ax0][ax1].set_title("N="+str(n_rolls_c[3*ax0+ax1]))
fig.tight_layout()
plt.show()
```

Τα αποτελέσματα είναι τα εξής:

```
============================================================ ============================================================
10 die rolls results:                                       100 die rolls results:
Mean value:  2.8000 Difference:  0.7000                     Mean value:  3.4000 Difference:  0.1000
Variance:  2.5600 Difference:  0.3566                       Variance:  2.6800 Difference:  0.2366
Skewness is 0.7676 Difference:  0.7676                      Skewness is 0.1641 Difference:  0.1641
Kurtosis is 2.3345 Difference:  0.6030                      Kurtosis is 1.7779 Difference:  0.0465
============================================================ ============================================================
============================================================ ============================================================
20 die rolls results:                                       500 die rolls results:
Mean value:  3.6000 Difference:  0.1000                     Mean value:  3.4060 Difference:  0.0940
Variance:  2.9400 Difference:  0.0234                       Variance:  2.8172 Difference:  0.0994
Skewness is 0.0976 Difference:  0.0976                      Skewness is 0.0998 Difference:  0.0998
Kurtosis is 1.5780 Difference:  0.1535                      Kurtosis is 1.7543 Difference:  0.0228
============================================================ ============================================================
============================================================ ============================================================
50 die rolls results:                                       1000 die rolls results:
Mean value:  3.3000 Difference:  0.2000                     Mean value:  3.4640 Difference:  0.0360
Variance:  2.7300 Difference:  0.1866                       Variance:  2.9227 Difference:  0.0061
Skewness is 0.2341 Difference:  0.2341                      Skewness is 0.0428 Difference:  0.0428
Kurtosis is 1.7162 Difference:  0.0153                      Kurtosis is 1.7236 Difference:  0.0078
============================================================ ============================================================
```

Ερώτημα δ

Παρατηρώντας τα αποτελέσματα, οι πειραματικές τιμές τιμές προσεγγίζουν τις θεωρητικές τιμές σε ακρίβεια 2 δεκαδικών για τιμές πλήθους 500 και πάνω(500,1000).

Ερώτημα ε

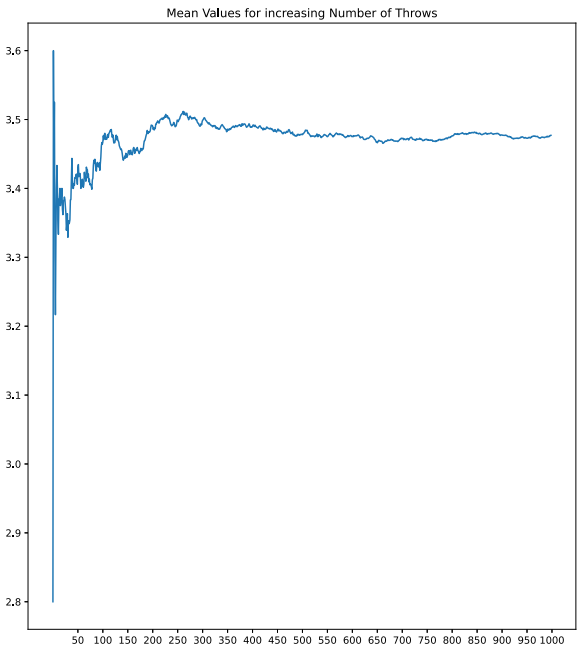
Με βάση τη θεωρία μια διεργασία $X(n)$ είναι στάσιμη υπό την ευρεία έννοια (Wide Sense Stationary) όταν η μέση τιμή είναι σταθερή(και ανεξάρτητη από το χρόνο) καθώς επίσης η αυτοσυσχέτιση εξαρτάται μόνο απο την χρονικη διαφορά $\tau = t1 - t2$ και όχι από τα $t1, t2$ ξεχωριστά. Δηλαδή:

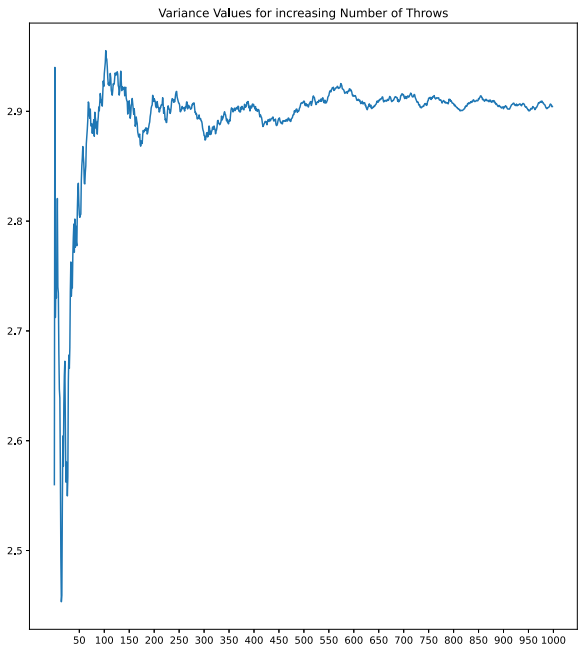
$$m_X(t) = E[X(t)] = \text{Constant}$$
$$R_X(t_1, t_2) = E[X(t_1)X(t_2)] = R_X(t_1 - t_2) = R_X(\tau)$$

Θέλουμε, λοιπόν να δούμε από ποιό αριθμό ρίψεων και πάνω έχουμε σταθερό μέση τιμή, για παράδειγμα με απόκλειση μικρότερη των 2 δεκαδικών. Μέσω του παρακάτω κώδικα, αρχικά εκτελούμε 10000 ρίψεις και στη συνέχεια υπολογίζουμε τα στατιστικά μέση τιμή και διακύμανση για ολοένα και μεγαλύτερα σύνολα ρίψεων με βήμα 10(δηλαδή 10,20,30 ..) καθώς επίσης κάνουμε το διάγραμμα συνάρτησης αυτοσυσχέτισης (autocorrelation function-acf) η αλλιώς correlogram.

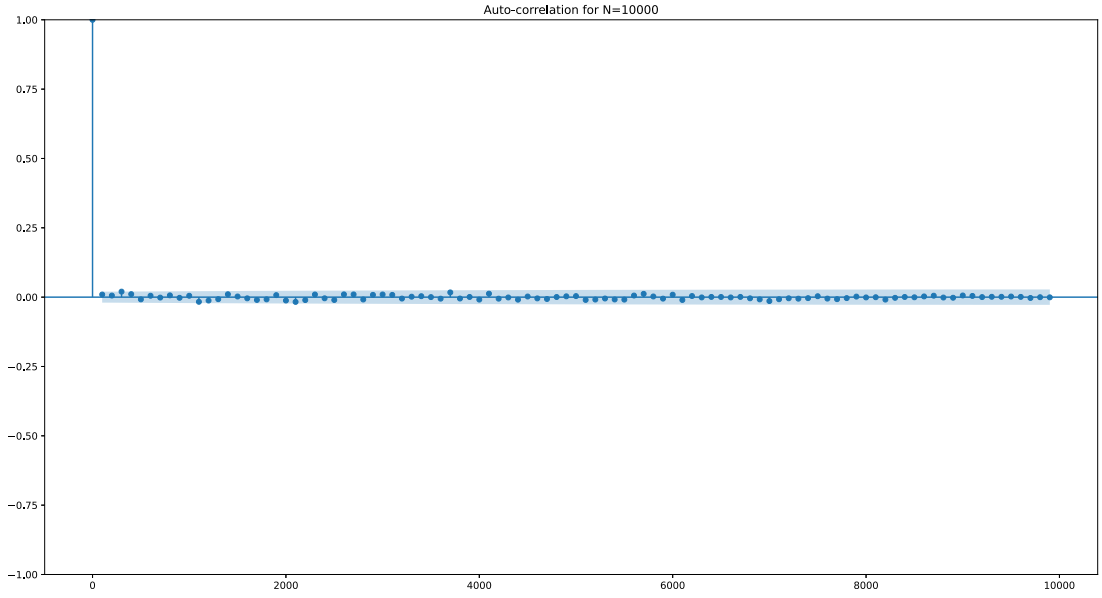
```
# Ερώτημα ε
s = 10000
throws_list_e = die_rolls(s)
mean_values = [np.mean(throws_list_e[0:i]) for i in range(10, s, 10)]
var_values = [np.var(throws_list_e[0:i]) for i in range(10, s, 10)]
fig, axes = plt.subplots(nrows=1, ncols=2)
axes[0].plot(mean_values)
axes[0].set_title("Mean Values for increasing Number of Throws")
axes[0].set_xticks(range(50, 1050, 50))
axes[1].plot(var_values)
axes[1].set_title("Variance Values for increasing Number of Throws")
axes[1].set_xticks(range(50, 1050, 50))
fig.tight_layout()
plt.show()
#Plot autocorrelation function - correlogram
plot_acf(throws_list_e, lags=range(0, s, 100), title="Auto-correlation for
N="+str(s))
plt.show()
```

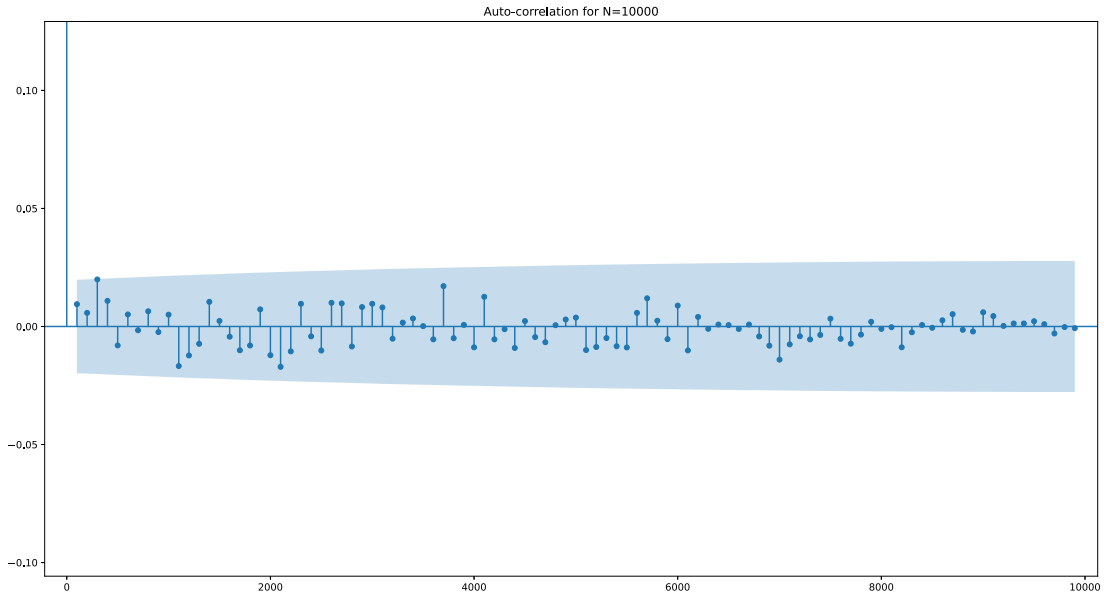
Τα αποτελέσματα είναι τα εξής:





Χρειάζεται λιγη προσοχή στην ερμηνία όσον αφορά τον άξονα χ καθώς οι τιμές 50,100,150,200 κλπ πρέπει να πολ\στούν με το βήμα 10 που αναφέρθηκε νωρίτερα για να δούμε τον αριθμό ρίψεων τον οποίο εξετάζουμε. Όπως βλέπουμε στο διάγραμμα της μέσης τιμής, μετά τις 1500 με 2000 ρίψεις(δλδ 150,200) παρατηρείται πως η μέση τιμή μένει σχεδόν σταθερή με πολύ καλή ακρίβεια. Στη συνέχεια εξετάζουμε το correlogram με αριθμό βήματος στα lags = 100.





Το αρχικό διάγραμμα δεν ειναι πολύ ευδιάκριτο, οπότε κάνουμε ζουμ στη περιοχή που μας ενδιαφέρει. Γενικά δεν παρατηρούμε κάποια ισχυρή εξάρτηση μεταξύ κάποιων χρονικών στιγμών του σήματος. Και εδώ βλέπουμε οτι μετά τις 2000 ριψεις πλην ελάχιστων εξεραίσεων ο ρυθμός με τον οποίο αλλάζει η στοχαστική διεργασία είναι εξαιρετικά μικρός.

Συνεπώς, καταλήγουμε πως οι 2000 ρίψεις είναι ένας επαρκής αριθμός ώστε να υπάρχει Wide Sense Stationarity.

Άσκηση 3

Ερώτημα α

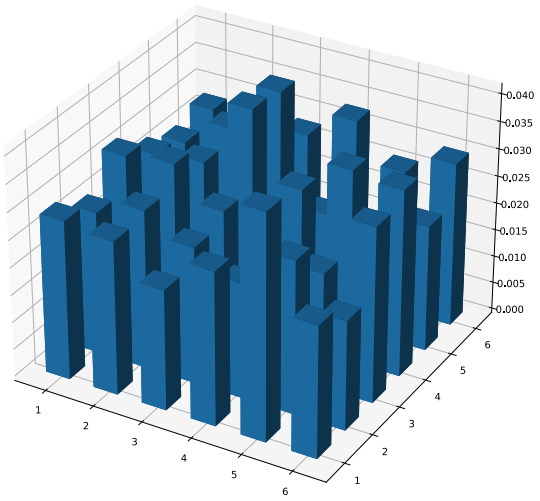
Στην περίπτωση 2 ζαριών, όπου $z1$ μια διακριτή τυχαία μεταβλητή που συμβολίζει το αποτέλεσμα της ρίψης του $\mathrm{Z1}$ και $z2$ μια διακριτή τυχαία μεταβλητή που συμβολίζει το αποτέλεσμα της ρίψης του $\mathrm{Z2}$. Το αποτέλεσμα του ενός ζαριού είναι ανεξάρτητο από το αποτέλεσμα του άλλου. Συνεπώς ισχυεί ότι για τα 36 στοιχεία του δειγματοχώρου(πχ {1,1},{1,2},{1,3},...{2,1},{2,2},..{6,6}):

$$f(z1, z2) = f(z1) \cdot f(z2) = \frac{1}{6} \cdot \frac{1}{6} = 0.027777\ldots$$

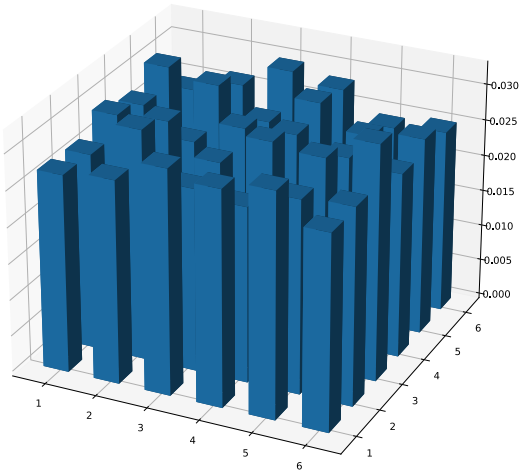
Μέσω του παρακάτω κώδικα απεικονιζουε τη κοινή συνάρτηση μάζας (πυκνότητας) πιθανότητας.

```
# Ερώτημα α
num_of_throws = 1000
rng = np.random.default_rng(seed=123)
z1 = rng.integers(low=1, high=6, endpoint=True, size=num_of_throws)
z2 = rng.integers(low=1, high=6, endpoint=True, size=num_of_throws)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
hist, x_edges, y_edges = np.histogram2d(z1, z2, bins=[i+0.5 for i in range(0,
7)],density=True)
```

```
x_pos, y_pos = np.meshgrid(x_edges[:-1] + 0.25, y_edges[:-1] + 0.25, indexing="ij")
x_pos = x_pos.ravel()
y_pos = y_pos.ravel()
z_pos = 0
dx = dy = 0.5 * np.ones_like(z_pos)
dz = hist.ravel()
ax.bar3d(x_pos, y_pos, z_pos, dx, dy, dz, zsort='average')
plt.show()
```



Παρατηρώ ότι για στη περίπτωση των 1000 ρίψεων έχουμε μια σχετική απόκλιση από τη θεωρητική καθώς τα πιθανά στοιχεία του δειγματοχώρου είναι 36 αντί για 6. Εαν για παράδειγμα αυξήσουμε(τετραπλασιάσουμε) σε 4000 ρίψεις το αποτέλεσμα είναι πολύ πιο κοντά στο θεωρητικό.



Ερώτημα β

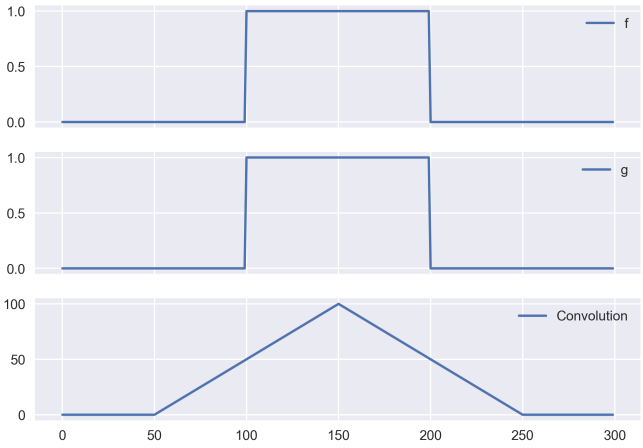
Ορίζουμε μία νεα διακριτή τυχαία μεταβλητή ως το άθροισμα των τιμών των 2 ζαριών $y = z1 + z2$. Ο δειγματοχώρος στη περίπτωση αυτή είναι:

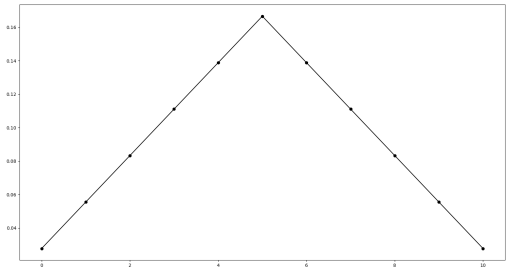
$$\Omega = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$$

Όπως είδαμε και στη θεωρία(S. 36 Lecture 3) η κατανομή αθροίσματος δύο ανεξάρτητων τυχαίων μεταβλητών έχει την ιδιότητα η κοινή συνάρτηση μάζας(πυκνότητας) πιθανότητας προκύπτει ως συνέλιξη των 2 επιμέρους συναρτήσεων, δηλαδή:

$$f(y) = f(z1) * f(z2)$$

Αν δούμε τις επιμέρους συναρτήσεις ως διανύσματα 6 στοιχείων η κάθε μία με όλα τα στοιχεία ίσα με 1/6 και πάρουμε τη συνέλιξη των δύο τότε αναμένουμε να έχουμε ενα αποτέλεσμα σαν ενας τριγωνικός παλμός καθώς οι τα δύο αρχικά σήματα ήταν ορθογώνιοι παλμοί(με όρους Ψηφιακής Επεξεργασίας σημάτων). Στο αριστερό παρακάτω σχήμα βλέπουμε ενα παράδειγμα ποιοτικό ενώ στα δεξιά βλέπουμε με χρήση του παρακάτω κώδικα τα ακριβή αποτελέσματα.

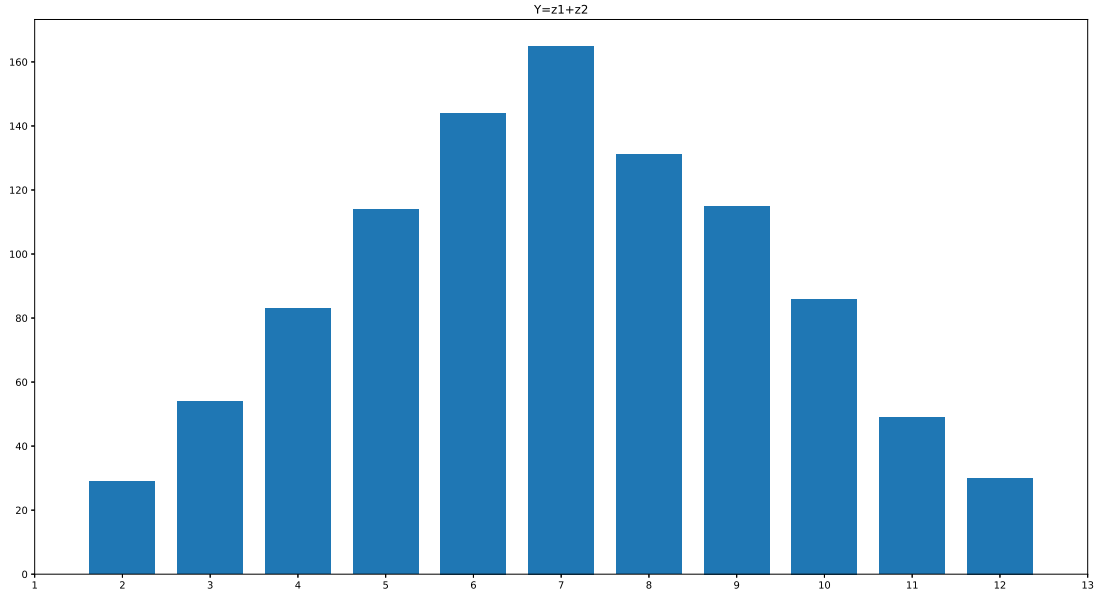




Αυτό το σχήμα μοιάζει με την διακριτή ομοιόμορφη κατανομή την οποία και αναμένουμε να δούμε στο σχήμα με την κοινή συνάρτηση μάζας(πυκνότητας) πιθανότητας.

```
# Ερώτημα β
z_temp = 1/6 * np.ones(6)
con = np.convolve(z_temp, z_temp, "full")
plt.plot(con, "-ok")
plt.show()

y = z1+z2
plt.hist(y, bins=[i for i in range(2, 14)], align="left", rwidth=0.75)
plt.xticks([i for i in range(1, 14)])
plt.title("Y=z1+z2")
plt.show()
```



Όπως επιβεβαιώνουμε η κοινή συνάρτηση μάζας(πυκνότητας) πιθανότητας ακολουθεί την κανονική διακριτή κατανομή (Gauss).

Άσκηση 4

Ερώτημα α

Έχουμε τη συνάρτηση $f(x) = (x-5)^4 + 3x$. Για να βρούμε τη πραγματική λύση της παίρνουμε την παράγωγο και την εξισώνουμε με το 0.

$$f'(x) = 4(x-5)^3 + 3 \Rightarrow$$

$$f'(x) = 0 \Rightarrow 4(x-5)^3 = -3 \Rightarrow (x-5)^3 = -\frac{3}{4}$$

$$\text{Για πραγματικά x: } x = -\sqrt[3]{\frac{3}{4}} + 5$$

Απο python:

```
# Ερώτημα α
analytic_sol = -pow(3/4, 1/3) + 5
print("Analytic Solution with Derivative Method: ", analytic_sol)
```

```
Analytic Solution with Derivative Method:  4.09143970358393
```

Ερώτημα β

Η υλοποίηση σε python της gradient descend είναι:

```
def gradient_descent(df, x_init, max_iterations, min_step, learning_rate=0.01):
    iterations = 0
    x = x_init
    step = 10
    while iterations < max_iterations and abs(step) > min_step:
        prev_x = x  # Store current x value in prev_x
        x = prev_x - learning_rate * df(prev_x)  # Gradient descent rule
        iterations += 1  # Iteration count
        step = prev_x - x
    print("The local minimum using Gradient Descent is ", x, 'after', iterations,
'iterations')
    return x, iterations
```

Ως ορίσματα δέχεται το df που είναι η συνάρτηση πρώτη παράγωγος της συνάρτησης που θέλουμε να βρούμε το ελάχιστο, x_init η αρχική προσέγγιση/θέση, max_iterations οι μέγιστες επαναλήψεις που επιτρέπουμε να τρέξει, min_step η ελάχιστη διαφορά/βήμα που πρέπει να υπάρχει μεταξύ δυο επαναλήψεων του αλγορίθμου για να συνεχίση η εκτέλεση του(εαν δηλαδή δύο εκτελέσεις έχουν διαφορά μικρότερη του min_step τότε το πρόγραμμα τερματίζει) και τέλος learning_rate είναι ο ρυθμός εκμάθησης στον οποίο δίνεται και μία default τιμή 0.01. Ωστόσο για να τρέξει ο αλγόριθμος πρέπει να μπορεί να υπολογιστεί η πρώτη παράγωγος. Η υλοιποίηση της σε python είναι:

```
def first_der(x):
    return 4 * pow((x - 5), 3) + 3
```

Τέλος καλούμε τη συνάρτηση με τα ορίσματα που θέλουμε. Η συνάρτηση επιστρέφει και εκτυπώνει τόσο την τελική τιμή που υπολίγισε όσο και τον αριθμό των επαναλήψεων που χρειάστηκαν.

```
# Ερώτημα β
[x_g, it_g] = gradient_descent(first_der, 10, 2000, 1e-9, 0.017)
print("Difference from analytic Solution :", abs(x_g-analytic_sol))
```

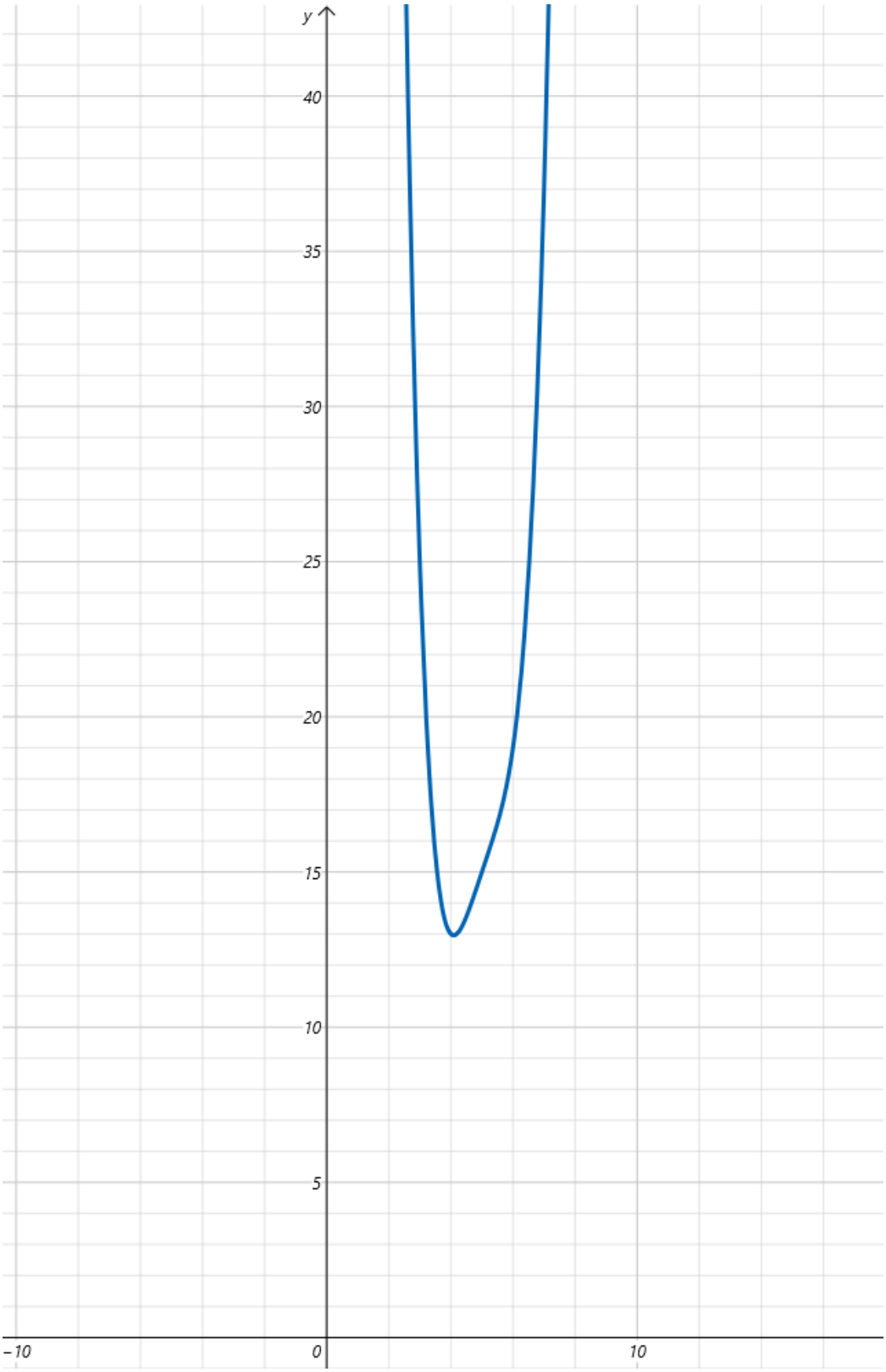
Μετά από πολλές δοκιμές η μεγαλύτερη δυνατή learning rate για αρχική προσέγγιση 10 που κατάφερε να τρέξει το πρόγραμμα ήταν αυτή του 0.017.

```
The local minimum using Gradient Descent is  4.091439708444329 after 103 iterations
Difference from analytic Solution : 4.860399194228648e-09
```

Σε περίπτωση που αυξήσουμε το learning rate περισσότερο θα εμφανιστεί το παρακάτω μήνυμα.

```
Traceback (most recent call last):
  File "C:\Users\stath\Documents\Pycharm\Pattern_Recognition\HW1\HW1_Exercise4.py", line 37, in <module>
    [x_g, it_g] = gradient_descent(first_der, 10, 1000, 1e-9, 0.02)
  File "C:\Users\stath\Documents\Pycharm\Pattern_Recognition\HW1\HW1_Exercise4.py", line 7, in gradient_descent
    x = prev_x - learning_rate * df(prev_x)  # Gradient descent rule
  File "C:\Users\stath\Documents\Pycharm\Pattern_Recognition\HW1\HW1_Exercise4.py", line 28, in first_der
    return 4 * pow((x - 5), 3) + 3
OverflowError: (34, 'Result too large')
```

Αναφέρεται ότι προέκυψε overflow πράγμα που συνέβει λόγω της γεωμετρίας της συνάρτησης. Όπως φαίνεται στο παρακάτω σχήμα η συνάρτηση είναι πολύ απότομη γύρω απο το ελάχιστο. Συνεπώς για μια αρχική τιμή 10 ακόμη και ένα learning rate της τάξης 0.02 αποτυγχάνει να βρεί το ελάχιστο και οδηγείται σε αστάθεια και κάποια στιγμή λόγω άυξησης της τιμής της παραγώγου να δημιουργείται overflow. Περισσότερα αποτελέσματα και δοκιμές παρουσιάζονται στο ερώτημα δ.



Ερώτημα γ

Η υλοποίηση σε python της Newton Method είναι:

```
def newton_method(fir_derivative, sec_derivative, x_init, max_iterations,
min_step):
    iterations = 0
    x = x_init
    step = 10
    while iterations < max_iterations and abs(step) > min_step:
        prev_x = x  # Store current x value in prev_x
        x = prev_x - fir_derivative(x)/sec_derivative(x)  # Newton-Raphson rule
        iterations += 1  # Iteration count
        step = prev_x - x
    print("The local minimum using Newton Method is ", x, 'after', iterations,
'iterations')
    return x, iterations
```

Στην περιπτωση αυτή τα ορίσματα είναι ίδια με με την εξαίρεση εδώ ότι απαιτείται η δεύτερη παράγωγος της συνάρτησης και όχι learning rate. Η υλοποίηση της πρώτης παραγώγου είναι ίδια με πρίν ενώ η υλοποίηση της 2ης παραγώγου σε python είναι:

```
def second_der(x):
    return 12*pow((x-5), 2)
```

Τέλος καλούμε τη συνάρτηση.

```
# Ερώτημα γ
[x_n, it_n] = newton_method(first_der, second_der, 10, 2000, 1e-9)
print("Difference from analytic Solution :", abs(x_n-analytic_sol))
```

Τα αποτελέσματα είναι τα εξής

```
The local minimum using Newton Method is  4.09143970358393 after 17 iterations
Difference from analytic Solution : 0.0
```

Εκ πρώτης όψεως φαίνεται πως η μέθοδος Newton συγκλίνει πολύ πιο γρήγορα από την Gradient Descent ενώ παράλληλα η διαφορά με τη πραγματική ρίζα γίνεται 0.

Ερώτημα δ

- Για αρχική προσέγγιση x = 10 έχουμε

Learning Rate	0.001	0.005	0.01	0.017
Gradient Descent Result	4.091440019617119 after 2000 iterations	4.091439722322443 after 447 iterations	4.0914397125409385 after 192 iterations	4.091439708444329 after 103 iterations
Difference from Analytic Solution	3.16033188951792e-07	1.8738512963523135e-08	8.957008468257754e-09	4.860399194228648e-09

ενώ τα αποτελέσματα της Newton method είναι όπως είδαμε πριν:

```
The local minimum using Newton Method is  4.09143970358393 after 17 iterations
Difference from analytic Solution : 0.0
```

Αρχικά, επιβεβαιώνεται η ταχύτερη σύγκλιση της Newton method σε αντίθεση με τη Gradient Descent όπου για learning rate=0.001 δεν κατάφερε καν να συγκλίνει στην ακρίβεια που θέλαμε και έφτασε το όριο των 2000 επαναλήψεων. Προφανώς όσο αυξάνεται το learning rate στη μέθοδο Gradient Descent η σύγκλιση έρχεται νωρίτερα αλλά αυτό όπως φάνηκε και νωρίτερα μπορεί να γίνει έως ένα όριο καθώς μπορεί να εμφανιστούν προβλήματα αστάθειας. Παρά το γεγονός όμως ότι η Newton Raphson φαίνεται να έχει πολύ πιο γρήγορα και ακριβή αποτέλεσματα(καθώς χρησιμοποιεί τη πληροφορία της καμπυλότητας δηλαδή της 2ης παραγώγου) ενώ παράλληλα δεν απαιτεί και ρύθμιση της υπερπαραμέτρου learning_rate συνήθως είναι δύσκολο έως αδύνατο να χρησιμοποιηθεί σε προβλήματα της αναγνώρισης προτύπων καθώς απαιτεί γνώση της 2ης παραγώγου ενώ στα προβλήματα αυτά υπάρχουν περιορισμοί και δυσκολίες ήδη κατα την έυρεση και ύπαρξη της πρώτης παραγώγου.

- Για αρχική προσέγγιση x = 1 (δηλαδή πιο κοντά στην ρίζα):

Learning Rate	0.001	0.01	0.02	0.03
Gradient Descent Result	4.0914396038060525 after 1564 iterations	4.0914396949801635 after 168 iterations	4.091439707051318 after 110 iterations	4.091439701314007 after 54 iterations
Difference from Analytic Solution	9.977787751580536e-08	8.603766588066719e-09	3.4673881543767493e-09	2.2699229162981283e-09

ενώ τα αποτελέσματα της Newton method είναι:

```
The local minimum using Newton Method is  4.09143970358393 after 9 iterations
Difference from analytic Solution : 0.0
```

Και εδώ βλέπουμε την Newton method να υπερέχει. Ωστόσο, επειδή η αρχική τιμή είναι πιο κοντά στη λύση και η κλίση είναι πιο μικρή είναι δυνατή και επιλογή learning_rate = 0.3. Προφανώς τα αποτελέσματα αυτά είναι λίγο "αυστηρά" υπο την έννοια οτι τέθηκε σφάλμα/ελάχιστο βήμα = 1e-9. Εαν οι απαιτήσεις ακρίβειας δεν είναι τόσο μεγάλες τότε ακόμη και το gradient descent έρχεται σε πιο γρήγορη σύγκλιση.