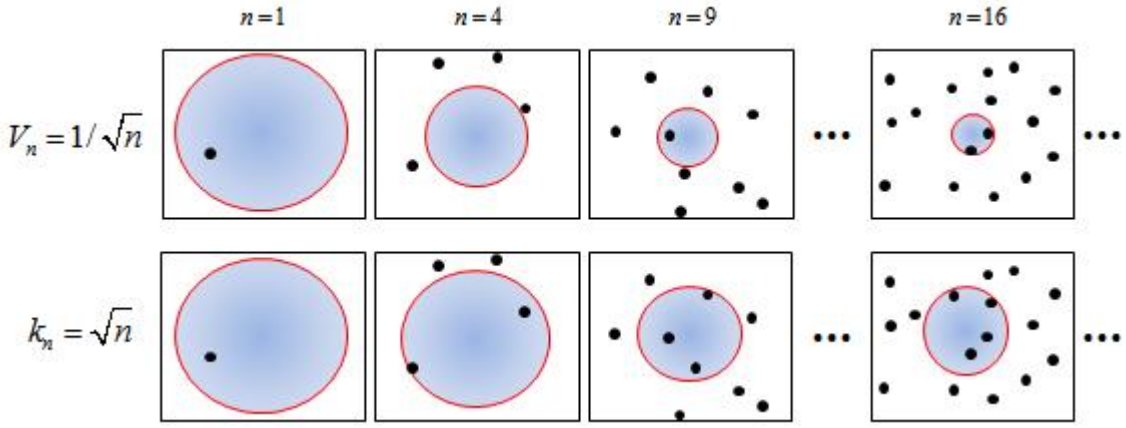
Αναγνώριση Προτύπων

2021-2022

Εργασία 4η

ΜΗ ΠΑΡΑΜΕΤΡΙΚΟΙ ΕΚΤΙΜΗΤΕΣ-PARZEN-KNN



Καρυπίδης Ευστάθιος 57556

5/12/21, Ξάνθη

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

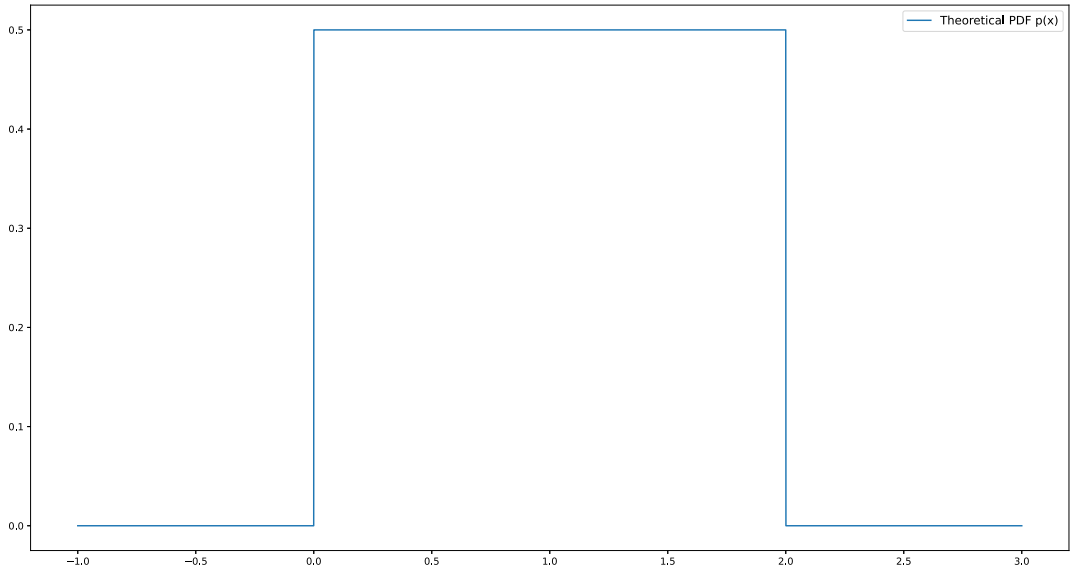
Άσκηση 1

Δίνεται ότι η σ.π.π μια τυχαίας μεταβλητής δίνεται από τη σχέση:

$$p(x) = \begin{cases} \frac{1}{2} & \gamma\iota\alpha\ 0 \leq x \leq 2 \\ 0 & \alpha\lambda\lambda\circ\acute{\upsilon} \end{cases}$$

Ο αντίστοιχος κώδικας σε python για παραγωγή της είναι:

```
x_theoretical = np.linspace(-1, 3, 401)
p_theoretical = np.where((0 < x_theoretical) & (x_theoretical < 2), 1/2, 0)
```



Ερώτημα α

Ζητείται να προσεγγιστεί η παραπάνω σ.π.π με τη μέθοδο παραθύρων Parzen χρησιμοποιώντας ως συνάρτηση παραθύρου τη Gaussian $N(0,1)$. Θα γίνει δοκιμή για $h = 0.05$ και $h = 0.2$ ενώ σε κάθε περίπτωση θα σχεδιαστή η εκτίμηση βασισμένη σε $N = 32, N = 256, N = 5000$ σημεία αντίστοιχα τα οποία παράγονται από μία γεννήτρια ψευδοτυχαίων αριθμών σύμφωνα μέ την p(x).

Γνωρίζουμε ότι ο γενικός τύπος για την εκτίμηση μιας σ.π.π με τη χρήση συνάρτησης παραθύρου(Parzen Window) είναι:

$$p_n(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{V_n} * \varphi\left(\frac{x - x_i}{h_n}\right)$$

όπου:

- h_n το πλάτος της συνάρτησης
- V_n ο όγκος της συνάρτησης
- φ η συνάρτηση παραθύρου
- ο αριθμός των παρατηρήσεων

Γνωρίζουμε ότι η $f(x)$σ.π.π μιας Gaussian με μέση τιμή μ και τυπική απόκλιση σ είναι:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Στη περίπτωση μας εφόσον θέλουμε συνάρτηση παραθύρου με Gaussian $N(0,1)$ θα έχουμε ότι:

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$$

Άρα η εκτίμηση μαις σ.π.π θα είναι

$$p_n(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h_n} * \varphi\left(\frac{x - x_i}{h_n}\right) \text{ όπου}$$

$$\varphi(u) = \frac{1}{\sqrt{2\pi}} e^{\frac{-u^2}{2}}$$

$$u = \frac{x - x_i}{h_n}$$

Τελικά:

$$p_n(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h_n\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-x_i}{h_n})^2}$$

Για τον υπολογισμό της εκτίμησης δημιουργήθηκε η παρακάτω συνάρτηση:

```
def parzen_gaussian_kernel(v, h):
    return np.mean(norm.pdf(v, loc=0, scale=1), axis=0)/h
```

Ειδικότερα παίρνει όρισματα τα:

- $v = \frac{x-x_i}{h_n}$ δηλαδή ένα διάνυσμα
- h_n το πλάτος της συνάρτησης.

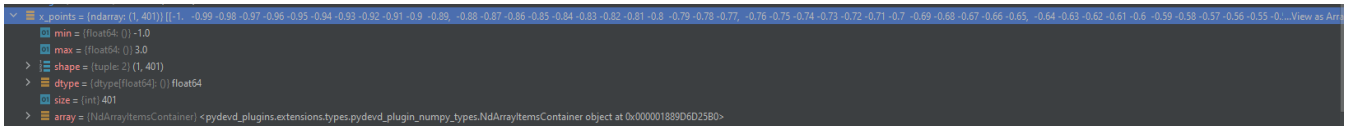
και χρησιμοποιεί την `norm.pdf` όπου loc είναι μέση τιμή και scale η τυπική απόκλιση.

Η λογική έχει ως εξής: Αρχικά,κτιμούμε την pdf για κάθε σύμφωνα με ένα πλήθος σημείων x_i και στη συνέχεια σε κάθε σημείο x προσεγγίζουμε και προσαρμόζουμε μία Gaussian pdf. Το σύνολο των προσεγγίσεων για όλα τα x μας δίνει την εκτίμηση της pdf. Ο κώδικας για τον υπολογισμό της εκτίμησης για ένα σύνολο σημείων με τις παραμέτρους που θέλουμε είναι:

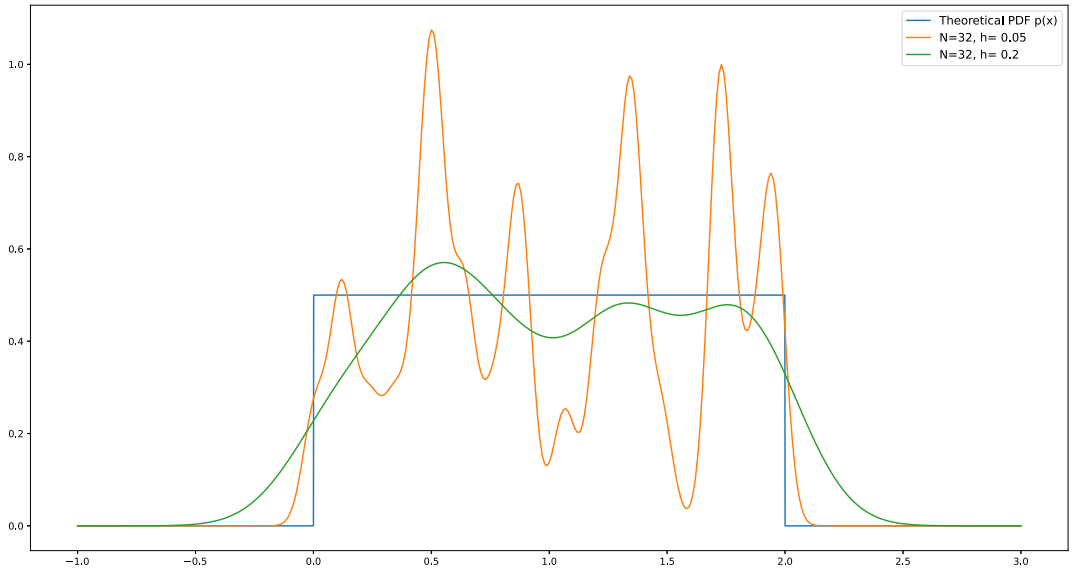
```
# Parzen Windows
N_values = [32, 256, 5000]
h_values = [0.05, 0.2]
for j in range(len(N_values)):
    X = rng.uniform(0, 2, N_values[j]).reshape((N_values[j], 1))
    plt.figure()
    plt.plot(x_theoretical, p_theoretical, label="Theoretical PDF p(x)")
    for i in range(len(h_values)):
        x_points = np.linspace(-1, 3, 401).reshape((1, 401))
        u = np.subtract(x_points, X)/h_values[i]
        # ph = np.mean((1/((np.sqrt(2*np.pi))*h_values[i]))*np.exp((-1/2)*(u**2)), axis=0)
        ph = parzen_gaussian_kernel(u, h_values[i])
```

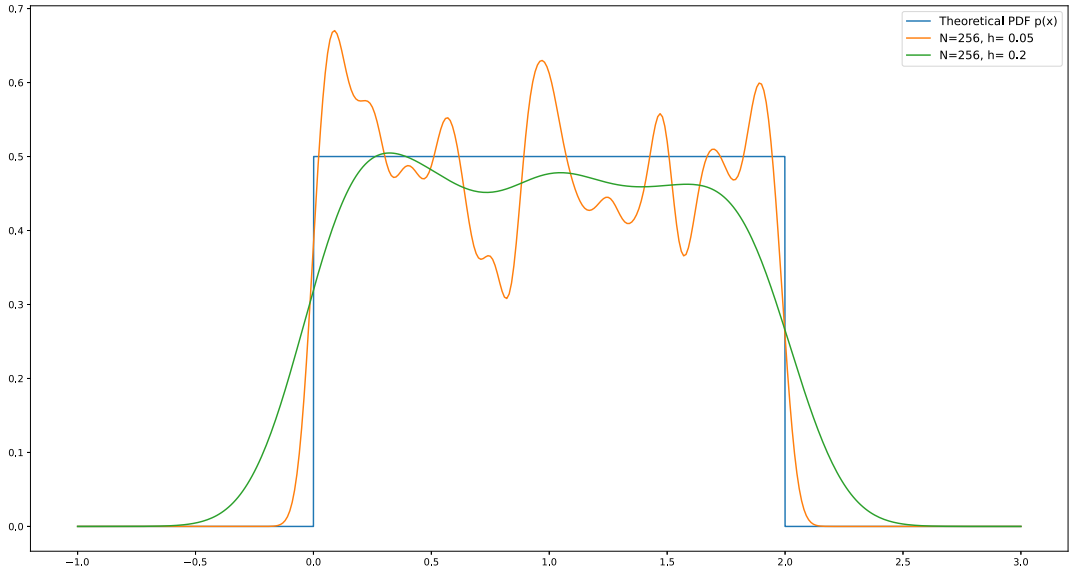
```
        plt.plot(x_points.reshape(401, 1), ph, label="N="+str(N_values[j])+", h=
"+str(h_values[i]))
        plt.legend(prop={'size': 12})
    plt.show()
```

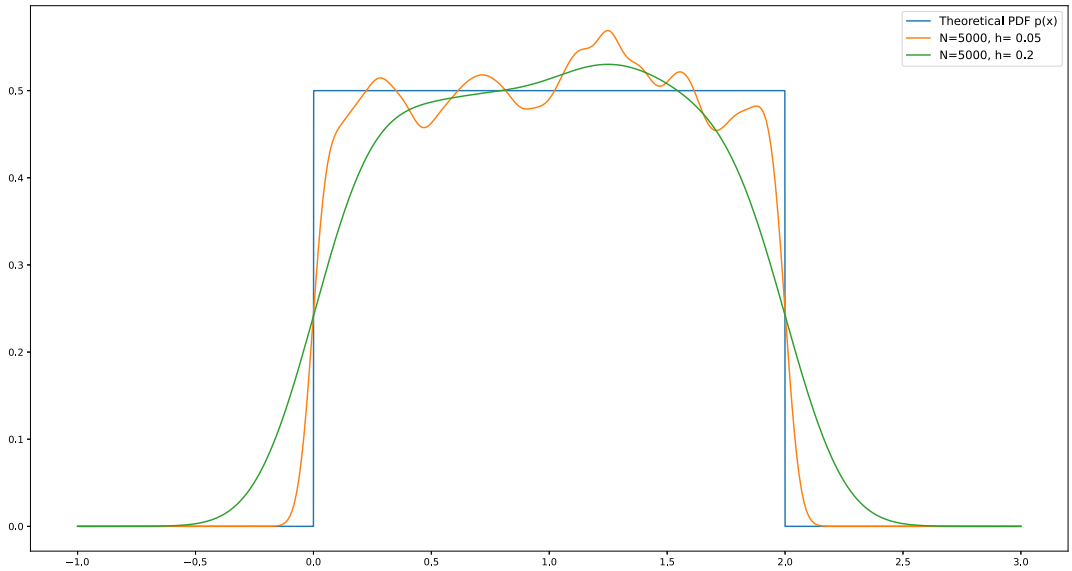
Αρχικά ορίζουμε τις τιμές N και h. Στη συνέχεια για κάθε περίπτωση παράγουμε ενα διάνυσμα 401 σημείων x_points με τιμές από -1 εως 3(εφόσον) με βήμα 0.01:



Σε αυτά τα σημεία θα προσπαθήσουμε να προσεγγίσουμε την pdf για κάθε σημείο X. Στη συνέχεια υπολογίζουμε το παράγοντα u ο οποίος θα έχει διάσταση Nx401 και τέλος εκτιμούμε την σ.π.π στα N σημεία είτε μέσω της συνάρτησης που αναφέρθηκε παραπάνω και κάνει χρήσητης συνάρτησης `norm.pdf` είτε γράφοντας απευθείας την εξίσωση(όπως φαίνεται στα σχόλια). Τέλος κανουμε τα αντίστοιχα plot. Τα αποτελέσματα είναι:



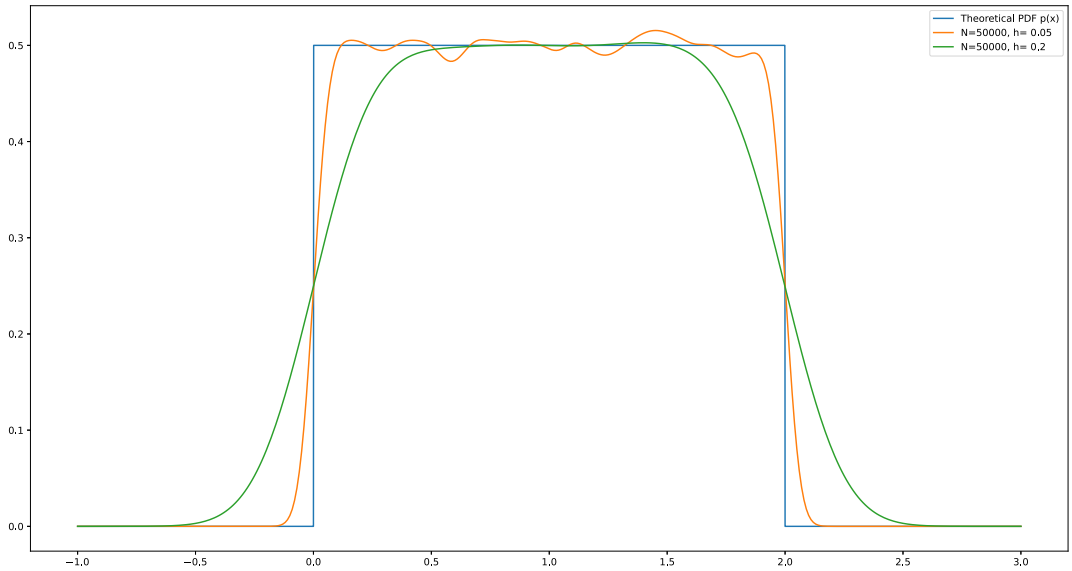




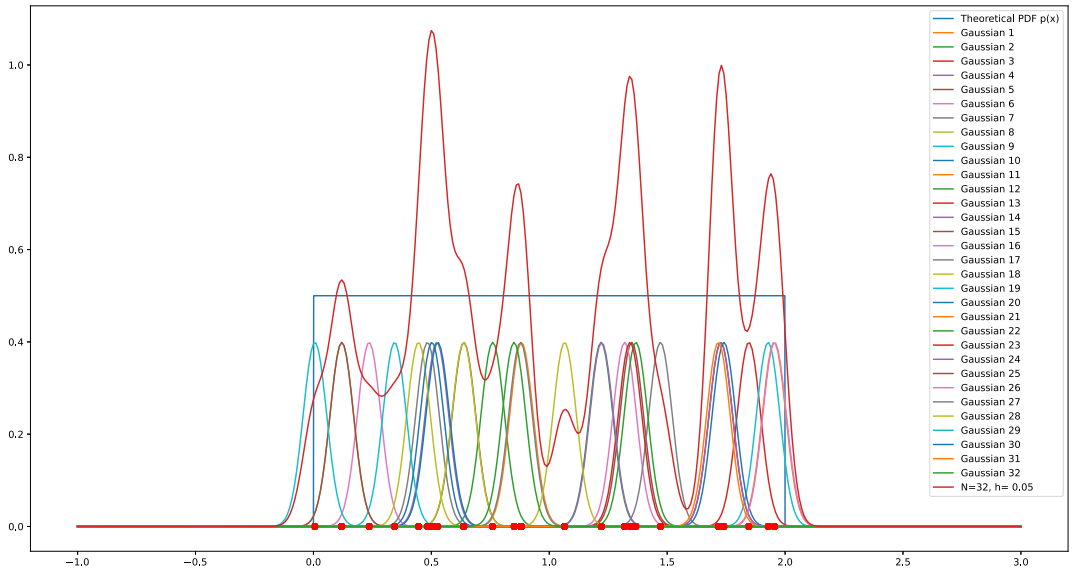
Αυτά ειναι τα διαγράμματα των προσεγγίσεων των pdf που προέκυψαν για $N = 32, N = 256$ και $N = 5000$ και με $h = 0.05$ και $h = 0.2$, Όπως φαίνεται, τα αποτελέσματα είναι πιο κοντά στα θεωρητικά(ιδανική ομοιόμορφη κατανομή) για μεγαλύτερα για $N = 5000$ και $h = 0.2$. Αυτό, είναι λογικό καθώς:

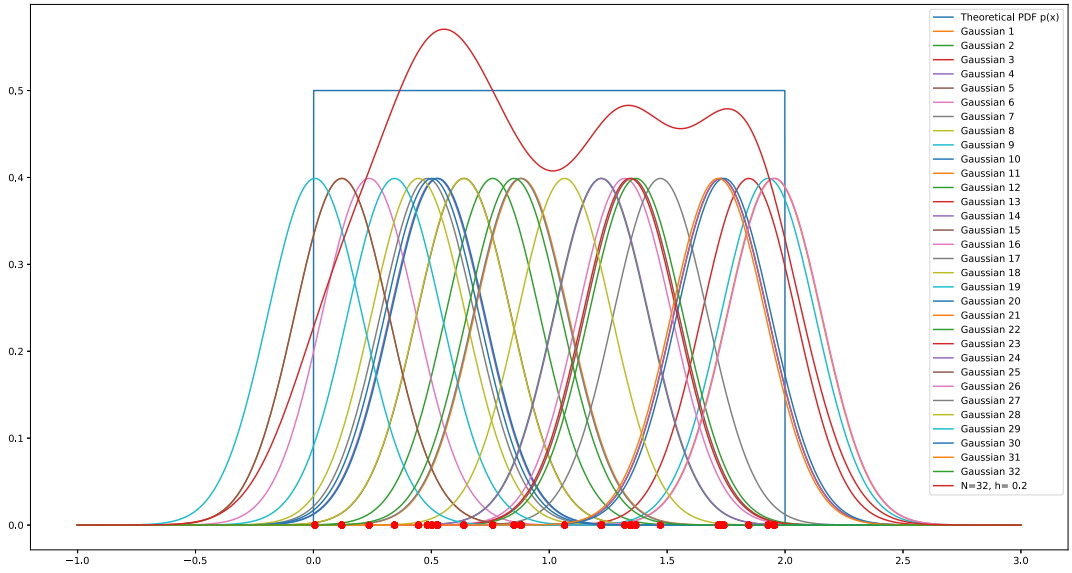
- Όσο αυξάνεται το N τόσο περισσότερα σημεία έχουμε για να προσεγγίσουμε την κατανομή και άρα καλύτερη εικόνα για αυτή.
- Οσο αυξάνεται το h, το οποίο όπως γνωρίζουμε αποτελέι παράμετρο εξομάλυνσης τόσο περισσότερο μειώνεται ο θόρυβος. Ωστόσο, πολυ μεγάλες τιμες του h μπορούσαν να προκαλέσουν υπερβολική εξομάλυνση. Ακόμη βλέπουμε ότι κοντα στις οριακές συνθήκες προσαρμόζεται καλύτερα(η εκτίμηση στη θεωρητική) όταν $h = 0.05$ καθώς υπάρχουν απότομες μεταβολές ενώ στο εσωτερικό προσαρμόζεται καλύτερα για $h = 0.2$.

Τέλος, έκανα και μια δοκιμή με N = 50000 όπου όπως βλέπουμε τα αποτελέσματα είναι πολύ κοντα στα θεωρητικά.



Τέλος, για καλύτερη κατανόηση του πως δουλεύει η εκτίμηση με μέθοδο παραθύρων Parzen έκανα οπτικοποίηση τις Gaussians στα για N=32 καθώς για περισσότερα ήταν πολύ δύσκολη η διάκριση μεταξύ τους. Ο κώδικας βρίσκεται στο αρχείο μεσα στη for αλλά σε σχόλια. Τα αποτελέσματα είναι τα εξής:





Όπως παρατηρούμε ο παράγοντας h υποδηλώνει και την τυπική απόκλιση των Gaussian όπως είδαμε και στη θεωρία. Ακόμη, εκεί που υπάρχουν πολλά σημεία εμφανίζονται πολλές Gaussian και συνεπώς αυξάνεται η τιμή για τη PDF τη συνολική.

Ερώτημα β

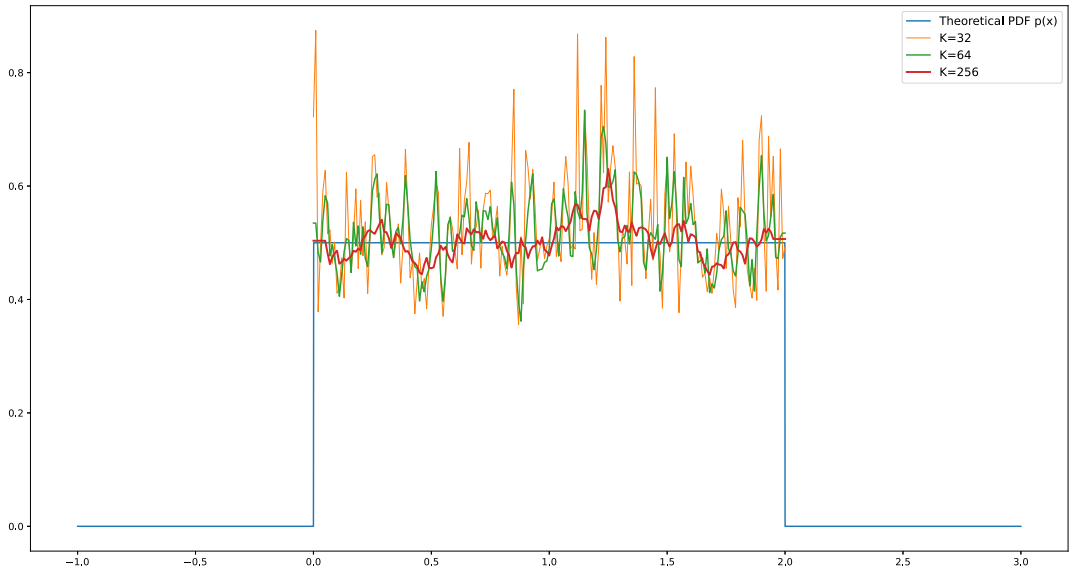
Στο ερώτημα αυτό ζητείται η εκτίμηση της pdf με χρήση της μεθόδου k-πλησιέστερερων γειτόνων με $k = 32$, $k = 64$, $k = 256$ έχοντας $N = 5000$ σημεία. Η εκτίμηση KNN βασίζεται στον εντοπισμό των K πλησιέστερων γειτόνων(K-NN). Με τη χρήση αυτής της μεθόδου η pdf προσεγγίζεται με χρήση της σχέσης:

$$p(x) = \frac{k}{N \cdot V}$$

όπου k ο αριθμός των κοντινότερων γειτόνων του σημείου x, N το πλήθως των δεδομένων , V = dist όπου dist η απόσταση του πιο μακρινού και πιο κοντινού γείτονα στη περίπτωση 1 διάστασης. Ο κώδικας είναι:

```
# K Nearest Neighbors
N = 5000
X = rng.uniform(0, 2, N).reshape((N, 1))
K_values = [32, 64, 256]
plt.figure()
plt.plot(x_theoretical, p_theoretical, label="Theoretical PDF p(x)")
for j in range(len(K_values)):
    x_points = np.linspace(0, 2, 201).reshape((1, 201))
    distances = np.abs(np.subtract(x_points, X))  # Manhattan Distance (L1) = Euclidian
Distance (L2) because dim = 1
    sorted_dist_ind = np.argsort(distances, axis=0)
    Nearest_Neighbors = X[sorted_dist_ind[0:K_values[j]]]
    V = np.max(Nearest_Neighbors, axis=0)-np.min(Nearest_Neighbors, axis=0)
    ph = K_values[j]/(N*V)
    plt.plot(x_points.reshape(201, 1), ph, label="K=" + str(K_values[j]), linewidth=j/2+1)
plt.legend(prop={'size': 12})
plt.show()
```

Το αποτέλεσμα είναι το εξής:



Όπως παρατηρούμε, καθώς η τιμή του K αυξάνεται η προσέγγιση βελτιώνεται. Αυτό είναι απόλυτα λογικό αφού αυξάνοντας το K αυξάνουμε το πλήθος των γειτόνων που εξετάζουμε για την εκτίμηση της pdf κάθε σημείου, άρα η εικόνα που έχουμε για την κατανομή γίνεται καλύτερη.

Άσκηση 2

Δίνεται ένα πρόβλημα με 3 κλάσεις:

$$P(x|\omega_1) = N(2, 0.5),\ P(x|\omega_2) = N(1, 1),\ P(x|\omega_3) = N(3, 1.2)$$
$$P(\omega_1) = 0.5,\ P(\omega_2) = 0.3,\ P(\omega_1) = 0.2$$

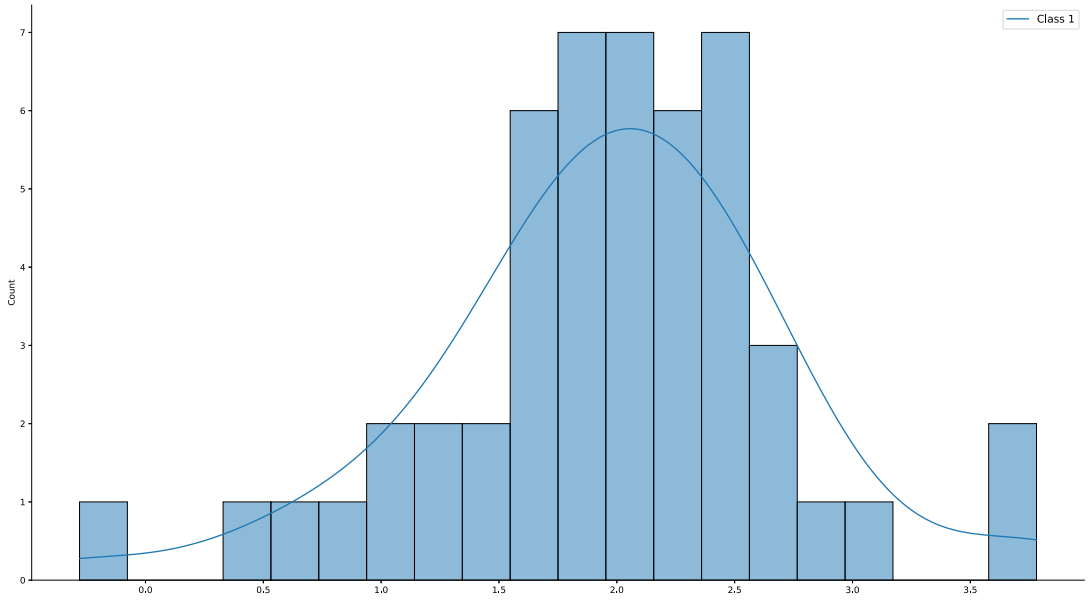
Ερώτημα α

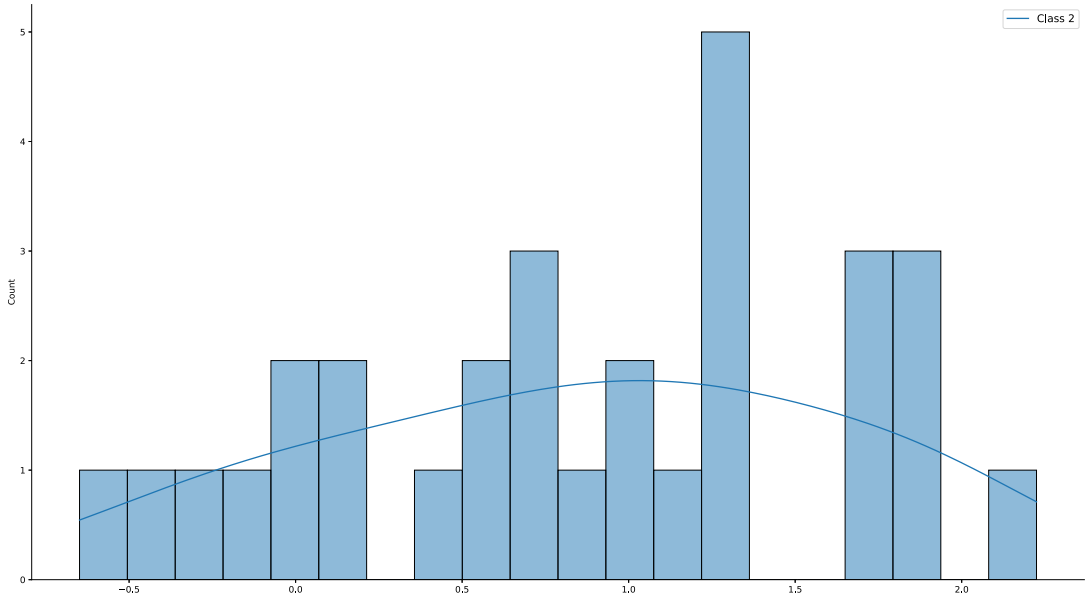
Αρχικά, ζητείται να δημιουργηθεί ένα τυχαίο δείγμα 100 δειγμάτων(προτύπων) που να ακολουθεί τις παραπάνω κατανομές, σύμφωνα με τις δεδομένες εκ των προτέρων πιθανότητες. Αυτό, θα χρησιμοποιηθεί ως σύνολο εκμάθησης. Στη συνέχεια θα δημιουργηθεί ενα τυχαίο δείγμα 1000 σύμφωνα με τις δεδομένες εκ των προτέρων πιθανότητες. Αυτό, θα χρησιμοποιηθεί ως σύνολο δοκιμής. Για το σκοπό αυτό όρισα την παρακάτω συνάρτηση. Η συνάρτηση αυτή επιστρέφει μία λίστα με πίνακες 3 πίνακες δεδομένων για κάθε κλάση και μία λίστα με 3 πίνακες δεδομένων με τον αριθμό της κλάσης(0, 1, 2).

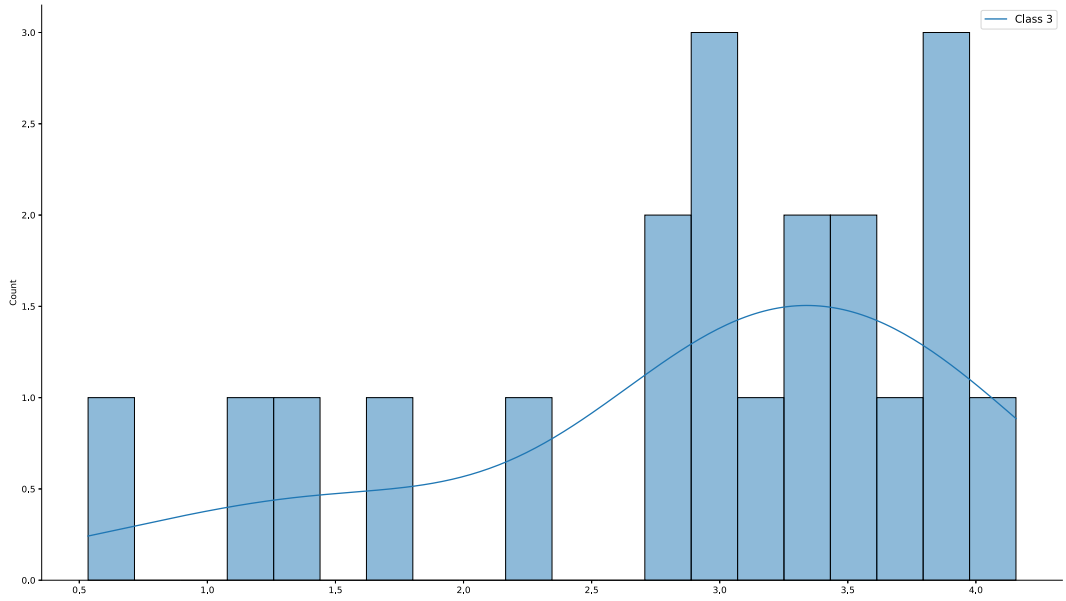
```
def create_data(num_of_classes, size, priors_list, list_of_means, list_of_std):
    data, labels = [], []
    available_num_of_samples = size
    for j in range(num_of_classes-1):
        classSize = math.floor(size*priors_list[j])
        data.append(rng.normal(list_of_means[j], math.sqrt(list_of_std[j]),
size=classSize))
        labels.append(np.ones(classSize) * j)
        available_num_of_samples -= classSize
    # for last class we need to take the remaining samples because rounding error of
size*priors()
    data.append(rng.normal(list_of_means[-1], list_of_std[-1],
size=available_num_of_samples))
    labels.append(np.ones(available_num_of_samples)*(num_of_classes-1))
    return data, labels
```

Τώρα μέσω του παρακάτω κώδικα κάνουμε χρήση της συνάρτησης αφού ορίσουμε τις μέσες τιμές, τις διακυμάνσεις και τις apriori πιθανότητες και στη συνέχεια ενοποιούμε τις λίστες σε ένα πίνακα. Τέλος μπορούμε να κάνουμε μία σχετική οπτικοποίηση των δεδομένων εκπαίδευσης.

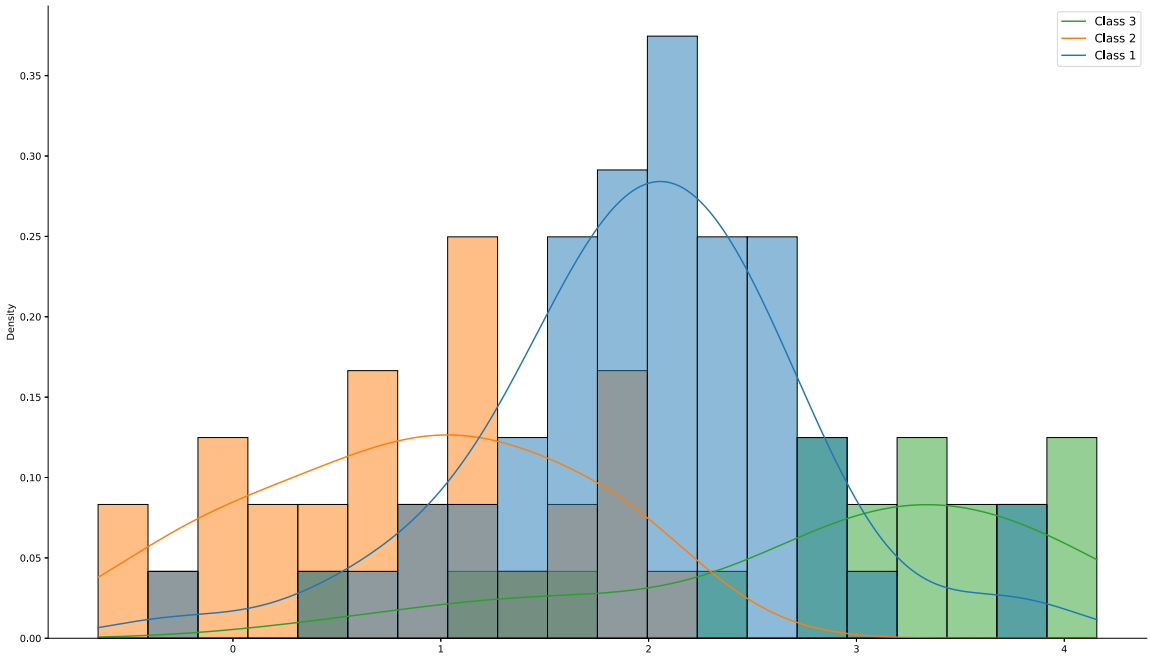
```
# Exercise 4.2 A
priors = np.array([0.5, 0.3, 0.2])
means = np.array([2, 1, 3])
covariances = np.array([0.5, 1, 1.2])
stds = np.sqrt(covariances)
train_data_list, train_labels_list = create_data(3, 100, priors, means, stds)
test_data_list, test_labels_list = create_data(3, 1000, priors, means, stds)
train_data = np.concatenate(train_data_list)
test_data = np.concatenate(test_data_list)
train_labels = np.concatenate(train_labels_list)
test_labels = np.concatenate(test_labels_list)
# Visualization
sns.displot(train_data_list, bins=25, kde=True, stat="density", legend=False)
plt.legend(["Class 3", "Class 2", "Class 1"])
sns.displot(train_data_list[0], bins=25, kde=True, legend=False)
sns.displot(train_data_list[1], bins=25, kde=True, legend=False)
sns.displot(train_data_list[2], bins=25, kde=True, legend=False)
plt.show()
```


Class 1
Count
0
1
2
3
4
5
6
7
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


Class 2
Count
0
1
2
3
4
5
−0.5
0.0
0.5
1.0
1.5
2.0


Class 3
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Τέλος, μπορούμε να τις δούμε συγκεντρωτικά. Όπως παρατηρούμε είναι σχετικά κοντα στις θεωρητικές.



Ερώτημα β

Στο ερώτημα αυτό ζητείται να υλοποιηθεί ο αλγόριθμος k-nearest-neighbor και να υπολογιστεί η πιθανότητα λάθους για k=1,2,3. Ακόμη, ζητείται να συγκριθούν τα αποτελέσματα με τον Bayesian Ταξινομητή. Όσον αφορά τον knn ταξινομητή υλοποίησα τη παρακάτω συνάρτηση:

```
def knn_classifier(number_of_classes, train, test, train_classes, k_neighbours):
    class_votes = np.zeros((number_of_classes, len(test)), dtype=int)
    distances = np.sqrt(np.power(np.subtract(test.reshape(-1, 1).T, train.reshape(-1, 1)),
2))  # Manhattan Distance == Euclidian distances as dimension = 1
    sorted_dist_ind = np.argsort(distances, axis=0)
    Nearest_Neighbors_class = train_classes[sorted_dist_ind[0:k_neighbours]]
    for i in range(Nearest_Neighbors_class.shape[1]):
        for j in range(number_of_classes):
            class_votes[j, i] += (Nearest_Neighbors_class[:, i] == j).sum()
    knn_prediction = np.argmax(class_votes, axis=0)
    return knn_prediction
```

Ειδικότερα, αρχικοποιώ ένα πινακα class_votes μεγέθους 3xN όπου N ο αριθμός των σημείων που θέλουμε να ταξινομήσουμε. Σε αυτή τη λίστα θα αποθηκεύουμε για κάθε σημείο το πόσους κοντινότερους γείτονες από κάθε κλάση η αλλιώς πόσες ψήφους έχει η κάθε κλάση. Στη συνέχεια υπολογίζονται οι αποστάσεις μεταξύ των σημείων που θέλουμε ταξινομήσουμε και των δειγμάτων εκπαίδευσης και στη συνέχεια παίρνουμε μέσω της `np.argsort` τις θέσεις των στοιχείων του πίνακα αποστάσεων αλλά αφού έχει ταξινομηθεί από τις μικρότετες προς τις μεγαλύτερες. Οι κοντινότεροι k γείτονες στο πίνακα των δεδομένων εκπαίδευσης βρίσκονται παίρνοντας τις k πρώτες τιμές από το πίνακα όπου έχει τους δείκτες για τις ταξινομημένες αποστάσεις. Έπειτα, για κάθε σημείο υπολογίζουμε τις ψήφους(κοντινότερους γείτονες) για κάθε κλάση. Τέλος, ταξινομούμε το σημείο στη κλάση με τις περισσότερες ψήφους. Στη συνέχεια παραθέτω το κώδικα όπου χρησιμοποιούμε τη συνάρτηση:

```
# Exercise 4.2 B # Knn Classifier
K_values = [1, 2, 3]
for K in K_values:
    knn_predictions = knn_classifier(3, train_data, test_data, train_labels, K)
    knn_wrong_predictions = calculate_wrong_predictions(knn_predictions, test_labels)
    print("With k = ", K, " Knn Classifier Misclassified ", knn_wrong_predictions, "points.
Total error is ", knn_wrong_predictions/len(test_labels))
```

Όπως φαίνεται γίνεται χρήση ακόμη της συνάρτησης `calculate_wrong_predictions` για υπολογισμό των λαθών η οποία παρατείθεται παρακάτω:

```
def calculate_wrong_predictions(prediction, real_classes):
    return (prediction != real_classes).sum()
```

Τα αποτελέσματα είναι τα εξής:

```
With k =  1  Knn Classifier Misclassified  436 points. Total error is  0.436
With k =  2  Knn Classifier Misclassified  419 points. Total error is  0.419
With k =  3  Knn Classifier Misclassified  387 points. Total error is  0.387
```

Στη συνέχεια ζητείται να συγκρίνουμε τα αποτελέσματα με τον ταξινομητή Bays. Η συνάρτηση όπως υλοποιήθηκε και σε προηγούμενη εργασία για τον Bayes Classifier είναι:

```
def bayes_classifier(test, list_of_means, list_of_covariances, priors_list):
    g1 = np.array([discriminant_function_1d(test[t], list_of_means[0],
list_of_covariances[0], priors_list[0]) for t in range(len(test))]).reshape(-1, 1)
    g2 = np.array([discriminant_function_1d(test[t], list_of_means[1],
list_of_covariances[1], priors_list[1]) for t in range(len(test))]).reshape(-1, 1)
    g3 = np.array([discriminant_function_1d(test[t], list_of_means[2],
list_of_covariances[2], priors_list[2]) for t in range(len(test))]).reshape(-1, 1)
    g = np.concatenate((g1, g2, g3), axis=1)
    bayes_prediction = np.argmax(g, axis=1)
    return bayes_prediction
```

Όπως γνωρίζουμε υπολογίζουμε 3 συναρτησεις διάκρισης για κάθε σημείο και το ταξινομούμε στη κλάση με τη μεγαλύτερη τιμή της αντίστοιχης συνάρτησης διάκρισης. Η τιμής της συνάρτηση διάκρισης για μια διάσταση υπολογίζεται από τη παρακάτω συνάρτηση:

```
def discriminant_function_1d(x, m, s, prior):
    c = x - m
    S_inv = s ** (-1)
    det_S = abs(s)
    return -0.5 * c ** 2 * S_inv - (1 / 2) * math.log(2*math.pi) - 0.5 * math.log(det_S) +
math.log(prior)
```

Τέλος, μένει να καλέσουμε κάθε συνάρτηση. Ωστόσο, πριν γίνει αυτό χρειάζεται να κάνουμε εκτίμηση παραμέτρων. Ειδικότερα με χρήση της μεθόδου Maximum Likelihood Estimation κάνουμε εκτίμηση των μέσων τιμών και των διακυμάνσεων.

```
# Bayes Classifier
estimated_means = [np.mean(td) for td in test_data_list]
estimated_cov = [np.cov(td) for td in test_data_list]
print("Estimated means from MLE: ", estimated_means)
print("Estimated covariances from MLE: ", estimated_cov)
bayes_predictions = bayes_classifier(test_data, estimated_means, estimated_cov, priors)
bayes_wrong_predictions = calculate_wrong_predictions(bayes_predictions, test_labels)
print("Bayes Classifier misclassified ", bayes_wrong_predictions, "points. Total error is
", bayes_wrong_predictions/len(test_labels))
```

Τα αποτελέσματα είναι τα εξής:

```
Estimated means from MLE:  [1.9681050764177128, 0.9298761958411303, 2.891149977477306]
Estimated covariances from MLE:  [array(0.66249898), array(1.1321936), array(1.21085285)]
Bayes Classifier misclassified  354 points. Total error is  0.354
```

Συγκρίνοντας τα σφάλματα των 2 ταξινομητών, παρατηρώ πως ο Bayesian ταξινομητής έχει μικρότερο σφάλμα σε σχέση με τον kNN ταξινομητή. Αυτό, συμβαίνει καθώς με το ταξινομητή Bayes μπορεί κανονικά να υπολογιστεί η pdf αλλά στον kNN ταξινομητή οι pdf προσεγγίζονται ως $p(\omega|x) \approx \frac{ki}{k}$ όπου ki ο αριθμός των γειτόνων της κλάσης i γύρω από το δείγμα x και k ο συνολικός αριθμός προτύπων όλων των κλάσεων στην περιοχή γύρω από το x. Ακόμη, η απόδοση του kNN εξαρτάται σε μεγάλο βαθμό από το k όπως θα δούμε και παρακάτω.

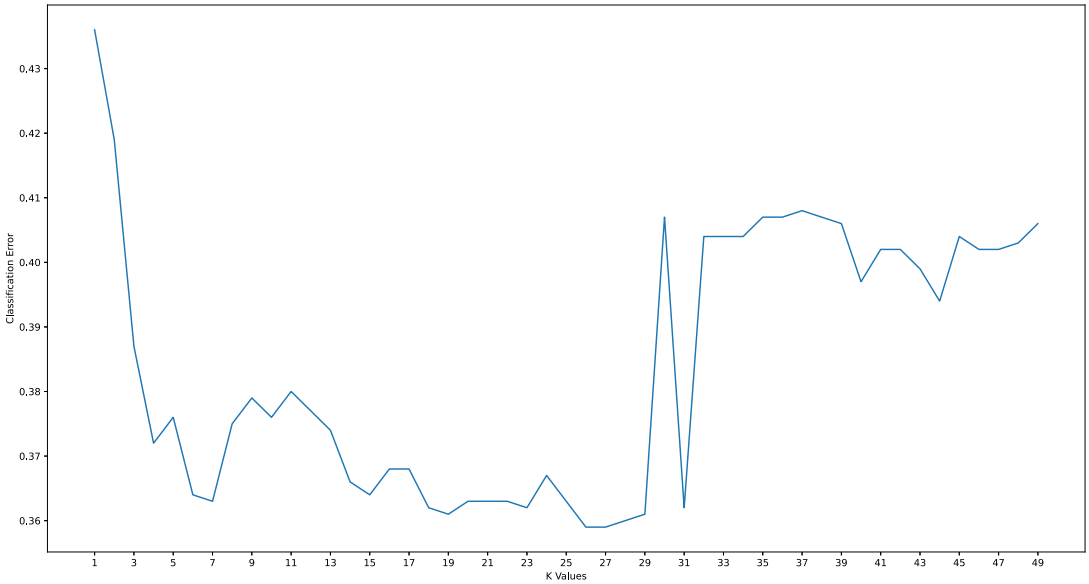
Προεραιτικό Ερώτημα

Στο ερώτημα αυτό με τη χρήση bruteforce για τιμές 1 εως 50 υπολογίζεται η βέλτιστη τιμή για το k, δηλαδή η τιμή του k για την οποία ελαχιστοποιείται το σφάλμα. Κανονικά σε τέτοιες περιπτώσεις είναι η χρήση η υπαρξη validation set καθώς η επιλογή παραμέτρων με βάση το σύνολο εκπαίδευσης μπορεί να οδηγεί σε ovefitting. Ωστόσο, στην άσκηση αυτή δεν δίνεται validation set καθώς επίσης ο αριθμός των δειγμάτων εκπαίδευσης είναι μικρός πράγμα που σημαίνει ότι η τιμή του k που θα βρεθεί είναι βέλτιστη για τα συγκεκριμένα σύνολα δοκιμής και ελέγχου και όχι απαραίτειτα βέλτιστη για νέα δείγματα. Ακόμη, πέραν του bruteforce αλγορίθμου δοκιμάζεται και ο κανόνας του $k = \sqrt{N}$ όπου N ο αριθμός των δειγμάτων εκπαίδευσης. Τέλος αξίζει να αναφερθεί ότι εμφανίζεται συχνα το πρόβλημα της ισοψηφίας γειτόνων, για το οποίο όμως δεν υπάρχει κάποια στανταρ τεχνική για την επίλυση του αλλά είτε μπορούν να ληφθούν υπ οψην και οι αποστάσεις των γειτόνων, είτε να αλλάξει ο αριμός των γειτόνων(πχ ± 1) είτε να γίνει τυχαία επιλογή.

```
K_rule = int(len(train_data) ** (1/2))  # Rule for K
K_rule_predictions = knn_classifier(3, train_data, test_data, train_labels, K_rule)
K_rule_mistakes = calculate_wrong_predictions(K_rule_predictions, test_labels)
print("Using rule k =sqrt(N)=  ", K_rule, " Knn Classifier Misclassified ",
K_rule_mistakes, "points. Total error is ", K_rule_mistakes/len(test_labels))
# Bruteforce Knn
knn_mistakes = []
K_test = [i for i in range(1, 50)]
for k in K_test:
    test_knn_predictions = knn_classifier(3, train_data, test_data, train_labels, k)
    knn_mistakes.append(calculate_wrong_predictions(test_knn_predictions, test_labels))
knn_best_idx, k_best = np.argmin(knn_mistakes) , K_test[knn_best_idx]
best_knn_mistakes = knn_mistakes[knn_best_idx]
print("With best k = ", k_best, " Knn Classifier Misclassified ", best_knn_mistakes,
"points. Total error is ", best_knn_mistakes/len(test_labels))
```

Τα αποτελέσματα είναι τα εξής:

```
Using rule k =sqrt(N)=  10  Knn Classifier Misclassified  376 points. Total error is  0.376
With best k =  26  Knn Classifier Misclassified  359 points. Total error is  0.359
```



Ερώτημα γ

Στο ερώτημα αυτό ζητείται η ταξινόμηση των δειγμάτων με χρήση των Parzen Windows και να γίνει δοκιμή για τουλάχιστον 4 τιμές της παραμέτρου h. Η συνάρτηση που υλοποιήθηκε είναι η εξής:

```
def parzen_gaussian_kernel(x, v):
    return np.mean(norm.pdf(x, loc=0, scale=1), axis=0)/v

def parzen_classifier(train_list, test, h, apriori):
    u = []
    pdf = []
    g = []
    for i in range(len(train_list)):
        u.append(np.subtract(test.reshape(-1, 1).T, train_list[i].reshape(-1, 1))/h)
        pdf.append(parzen_gaussian_kernel(u[i], h))
        g.append(pdf[i]*apriori[i])
    parzen_prediction = np.argmax(g, axis=0)
    return parzen_prediction
```

Ειδικότερα, η ταξινόμηση μέσω της μεθόδου Parzen Windows βασίζεται στην εκτίμηση του likelihood της κάθε κλάσης και στη συνέχεια αφού πολλαπλασιαστεί με την apriori πιθανότητα και προκύπτει ένα γινόμενο το οποίο είναι ανάλογο της posterior πιθανότητας με βάση το κανόνα του Bayes. Η κλάση λοιπόν, που ταξινομείται κάθε δείγμα είναι αυτή με το μεγαλύτερο γινόμενο likelihood * prior. Στη συνέχεια μένει να καλέσουμε τη συνάρτηση για διάφορες τιμές του h. Τέλος, αξίζει να αναφερθεί ότι ως συνάρτηση παραθύρου χρησιμοποιήθηκε η Gaussian $N(0,1)$ όπως στη προηγούμενη άσκηση.

```
# Parzen Classifier
h_values = [0.05, 0.1, 0.2, 0.4, 0.8]
for H in h_values:
    parzen_predictions = parzen_classifier(train_data_list, test_data, H, priors)
    parzen_wrong_predictions = calculate_wrong_predictions(parzen_predictions, test_labels)
    print("With h = ", H, " Parzen Classifier Misclassified ", parzen_wrong_predictions,
"points. Total error is ", parzen_wrong_predictions/len(test_labels))
```

Τα αποτελέσματα είναι τα εξής:

```
With h =  0.05  Parzen Classifier Misclassified  389 points. Total error is  0.389
With h =  0.1  Parzen Classifier Misclassified  372 points. Total error is  0.372
With h =  0.2  Parzen Classifier Misclassified  366 points. Total error is  0.366
With h =  0.4  Parzen Classifier Misclassified  364 points. Total error is  0.364
With h =  0.8  Parzen Classifier Misclassified  352 points. Total error is  0.352
With h =  1  Parzen Classifier Misclassified  356 points. Total error is  0.356
```

Όπως είναι αναμενόμενο τα αποτελέσματα καλύτερα ούτε για πολυ μεγάλη τιμή του h αλλά ούτε για πολύ μικρή. Όπως αναφέρθηκε προηγουμένως, το h είναι παράμετρος εξομάλυνσης και υπάρχει αντιστάθμιση μεταξύ της ευαισθησίας για θόρυβ στα δεδομένα εκπαίδευσης για πολύ μικρό h καθώς και υπερβολική εξομάλυνση για μεγάλο h. Η βελτιστοποίηση του h είναι ενα πρόβλημα που αφορά και την πολυπλοκότητα του μοντέλο.

Προεραιτικό Ερώτημα

Σε αυτό το ερώτημα αναζητείται η βέλτιστη τιμή του h για το πρόβλημα μας. Θα ακολουθηθεί και πάλι η μέθοδος του bruteforce για τιμές 0.05 εως 4 με βήμα 0.05.

```
# Bruteforce Parzen
parzen_mistakes = []
h_test = np.linspace(0.05, 4, 80)
for ht in h_test:
    test_parzen_predictions = parzen_classifier(train_data_list, test_data, ht, priors)
    parzen_mistakes.append(calculate_wrong_predictions(test_parzen_predictions,
test_labels))
h_best_idx = np.argmin(parzen_mistakes)
h_best = h_test[h_best_idx]
best_parzen_mistakes = parzen_mistakes[h_best_idx]
print("With best h = ", h_best, " Parzen Classified Misclassified ", best_parzen_mistakes,
"points. Total error is ", best_parzen_mistakes/len(test_labels))
```

Τα αποτελέσματα είναι τα εξής:

```
With best h =  0.9000000000000001  Parzen Classified Misclassified  346 points. Total error is  0.346
```

Προφανώς η τιμή που τυπώνεται είναι αποτέλεσμα αδυναμίας απεικόνισης του αριθμού 0.9 στο δυαδικό συστημα.