APPM 5515: High Dimensional Probability– Fall 2020 — Homework 2

Brandon Finley

1. Let x and y be indepdent, uniformly distributed random variables on the interval [-1,1]. Compute

   (a) $E[x^2]$

   (b) $E[x-y]$

   (c) $E[xy]$

   (d) $E[(x-y)^2]$

   Solution:

   a)

$$\begin{aligned} E[x^2] &= \int_{-1}^{1} x^2 f_x(x)\,dx \\ &= \int_{-1}^{1} x^2 \cdot \frac{1}{2}\,dx \\ &= \frac{1}{6}\cdot 2 \\ &= \frac{1}{3} \end{aligned}$$

   b)

$$\begin{aligned} E[x-y] &= \int_{-1}^{1}\int_{-1}^{1} (x-y) f_{x,y}(x,y)\,dx\,dy \\ &= E[x]-E[y] \\ &= 0-0 \\ &= 0 \end{aligned}$$

   c)

$$\begin{aligned} E[xy] &= \int_{-1}^{1}\int_{-1}^{1} xy f_{x,y}(x,y)\,dx\,dy \\ &= E[x]E[y] \\ &= 0\cdot 0 \\ &= 0 \end{aligned}$$

d)

$$\begin{aligned} E[(x-y)^2] &= \int_{-1}^{1}\int_{-1}^{1}(x-y)^2 f_{x,y}(x,y)\,dx\,dy \\ &= E[x^2]-2E[xy]+E[y^2] \\ &= \frac{1}{3}-2\cdot 0+\frac{1}{3} \\ &= \frac{2}{3} \end{aligned}$$

2. Let $X=(x_1,x_2,\cdots,x_n)^T$ and $Y=(y_1,y_2,\cdots,y_n)^T$. X and Y are indepdent, random vectors in R^n.

   (a) Compute the expected squared distance between X and Y, that is $E[||X-Y||^2]$.

   (b) Computed the expected angle between X and Y, that is $E[\angle X,Y]$.

   Solution:

   a) Let $Z=X-Y$. Then $Z=(x_1-y_1,x_2-y_2,\cdots,x_n-y_n)^T$. This implies that $||X-Y||^2=||Z||^2$. Using the properties of a norm, we know this to be $(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2$, or more succintly, $\sum_{i=1}^n(x_i-y_i)^2$. We can then plug this into our expectation value to find what we are looking for

$$\begin{aligned} E[||X-Y||^2] &= E[||Z||^2] \\ &= E[\sum_{i=1}^n (x_i-y_i)^2] \\ &= \sum_{i=1}^n E[(x_i-y_i)^2] \\ &= \sum_{i=1}^n \frac{2}{3} \\ &= \frac{2n}{3} \end{aligned}$$

   b) Let us define the L-2 inner product. Then, we know $\langle X,Y\rangle = X\cdot Y = x_1y_1+x_2y_2+\cdots+x_ny_n$. Now using the expectation value as a linear operator, we get $E[X\cdot Y]=E[x_1y_1+x_2y_2+\cdots+x_ny_n]$. Recall from above that $E[x_iy_i]\ \forall i\in N$ is 0. Then we know that $E[X\cdot Y]=0$. Since the inner product between these two random variables is defined to be zero, then we know *orthogonality* must exist. As such, the angle between the two random variables must be $90°$ or $\frac{\pi}{2}$ radians.

3. Use Hoeffding inequality to show that

$$P\left(|\langle x,u\rangle|>\epsilon\right)\le 2e^{-2\epsilon^2} \tag{1}$$

   on the interval [-0.5, 0.5].

Solution:
Recall that Hoeffding's inequality states that

$$P\left(|S_n - E[S_n]| > t\right) \leq 2e^{\frac{-2t^2}{\sum_{i=1}^n (b_i - a_i)^2}}$$

Now back to the problem, we see $\langle x, \mathbf{u} \rangle$ is in the place of the absolute value terms. Writing this out we get

$$\begin{aligned}\langle x, u \rangle &= \langle x, \left[\tfrac{1}{\sqrt{n}}, \tfrac{1}{\sqrt{n}}, \cdots, \tfrac{1}{\sqrt{n}}\right]^T \rangle \\ &= \sum_{i=1}^n \frac{1}{\sqrt{n}} x_i \\ &= \frac{1}{\sqrt{n}} \sum_{i=1}^n x_i \\ &= \frac{1}{\sqrt{n}} S_n\end{aligned}$$

Also note that $E[S_n] = E[\sum_{i=1}^n x_i] = \sum_{i=1}^n E[x_i] = n \cdot 0 = 0$. Now let $t = \epsilon \cdot \sqrt{n}$. Then using Hoeffding's inequality, we can say

$$\begin{aligned}P\left(|S_n - E[S_n]| > t\right) &= P\left(|S_n| > \epsilon \cdot \sqrt{n}\right) \\ &\leq 2e^{\frac{-2n\epsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}}\end{aligned}$$

Now note that $b_i = 0.5$ and $a_i = -0.5$. Thus,

$$\begin{aligned}2e^{\frac{-2n\epsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}} &= 2e^{\frac{-2n\epsilon^2}{n}} \\ &= 2e^{-2\epsilon^2}\end{aligned}$$

4. Take $\epsilon = \sqrt{\log 20/2} \approx 1.2$ to conclude that the points in K_n concentrate in the thin slab, of thickness ϵ, centered around the hyperplane $\mathbf{u}_\perp$.

Solution:
Here, we use Hoeffding's inequality to explain why this is the case. In question 3, we calculated the probability of the projection of x onto $\mathbf{u}$ is greater than some ϵ. Doing this, we found that probability is an incredibly tight bound that gets tight exponentionally. The significiance behind this is that if the probability of the projection of x onto the diagonal vector of the cube outside a given ϵ is extremely small, then it must be incredibly near to the vector. Thus, we see that the points concentrate onto the thin slab of thickness ϵ.

5. Randomly generate 400 points inside K_{100} and plot the histogram of the distances between points and the angle between the vectors from the origin to the points, for all pairs of points. Explain your findings in terms of the theoretical analysis that you performed in question 2.

Solution:
Here we will show how we can use the theory done in question 2 in real applications or simulations. Running the simulation the question gives us, with 400 points inside K_{100} on

the interval [-0.5, 0.5], we get the following histograms for the expected distance and angle between any two points.

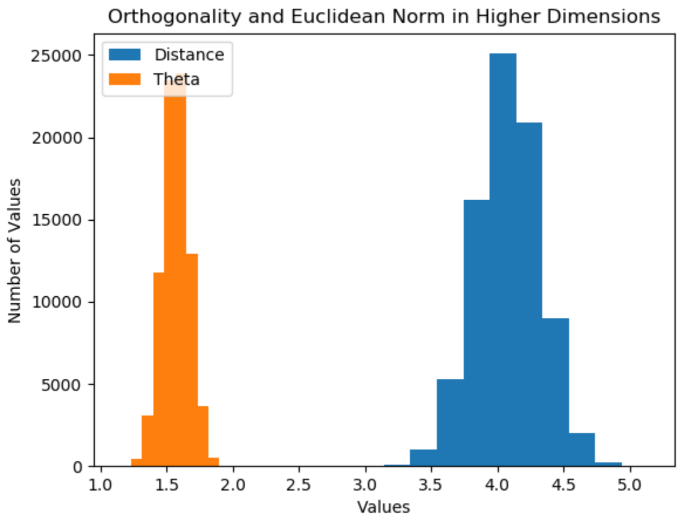

Figure 1: Histogram of Distance and Angle

We can see that the expected angle is 1.5 radians, which when plugged into cosine equals zero. This makes sense as it illustrates that in this dimensional space, most points are orthogonal. Moving onto the expected distance, we can apply the same procedure we used in question 2 but on the interval [-0.5, 0.5]. Doing this, we get $E[x^2] = \frac{1}{12}$. Thus, $E[(x_i - y_i)^2] = \frac{1}{6}$. This then implies that the expected squared distance is $n \cdot \frac{1}{6}$, or that the expected distance is $\sqrt{\frac{n}{6}}$. Finally, pluggin in $n = 100$, we get the value to be ≈ 4.0743. This is consistent with the experiment done. Yay for math!

6. Derive an expression for the volume V(Q) of

$$Q = [-1, 1] \times \cdots \times [-1, 1] \tag{2}$$

Solution:
We know the volume of a cube is the product of the length of its sides. Now, for 3 dimensions, that is l^3, for 4 dimensions, l^4, and so on. This impiles that for an n-dimensional cube, the volume is l^n. From the cube given above, we see that the side length, l, is 1 - (-1) = 2.
Thus,

$$\boxed{V(Q) = 2^n}$$

7. Prove that

$$\frac{V(B^n(1))}{V(Q)} \to \frac{1}{\sqrt{n}^n} \text{ as n } \to \infty \tag{3}$$

Solution:
Given from the problem statement, we know the following

$$V(B^n(1)) = \frac{(\sqrt{\pi})^n}{\Gamma(n/2 + 1)}$$

and

$$V(Q) = 2^n$$

We also know by Stirling's formula that as $n \to \infty$ we have the following:

$$\Gamma(z+1) \sim \sqrt{2\pi z}\left(\frac{z}{e}\right)^z$$

Therefore combing all equations as $n \to \infty$, and setting $z = n/2$, we can arrive at our conclusion.

$$\begin{aligned}
\frac{V(B^n(1))}{V(Q)} &= \frac{(\sqrt{\pi})^n}{2^n\Gamma(n/2+1)} \\
&\sim \frac{(\sqrt{\pi})^n}{2^n\sqrt{\pi n}\left(\frac{n}{2e}\right)^{n/2}} \\
&= \frac{1}{2^n\sqrt{\pi n}}\left[\sqrt{\frac{2\pi e}{n}}\right]^n \\
&\leq \frac{1}{(2\sqrt{n})^n} \\
&\leq \frac{1}{\sqrt{n}^n} \\
&\to 0 \text{ as } n \to \infty
\end{aligned}$$

8. Generate 10,000 samples distributed uniformly in $[-1,1]^n$ with algorithm 1, for $n = 1, \cdots, 400$. Plot as a function of n. Comment.

Solution:
As you can see from the figure below, as n approaches higher dimensions, the number of rejected points dramatically increases. Specifically, it seems to converge to all points being rejected around dimension 25.

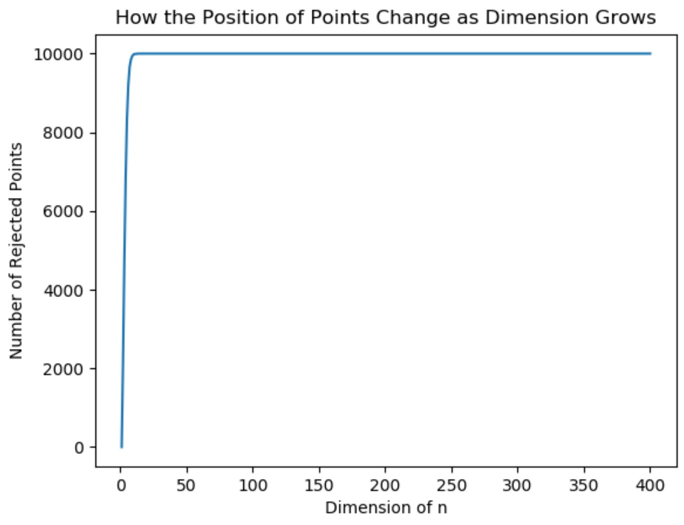

Figure 2: Simulation of Rejected Points Outside Ball

Looking at question 7 explains why this is the case. In question 7, we calculated the ratio of the volume to the ball compared to that of the cube. As we see, it quickly went to zero. This corroborates our simulation as many of the points inside the cube lied outside the unit ball, meaning that most of the points lied on the surface of the ball. This is an interesting finding that I am sure we will cover more of in future lectures.

— END —

Appendix

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

# ==================================================================
def calcDistance(x1, x2):

        # Euclidean norm
        d = np.sqrt(np.sum((x1 - x2)**2))

        return d

# ==================================================================
def comb(n, k):

        # Combination equation with failsafe
        if n > k:
                return int(np.math.factorial(n)/(np.math.factorial(n - k)*np.mat
        return -1

# ==================================================================
def calcAngle(x1, x2):

        # Using x1.x2 = |x1||x2|*cos(theta)
        x1_mag = np.linalg.norm(x1)
        x2_mag = np.linalg.norm(x2)
        x1_x2_dot = np.dot(x1, x2)

        # Finding final angle
        theta = np.arccos(x1_x2_dot/(x1_mag*x2_mag))

        return theta

# ==================================================================
def main():

        # Samples, dimension, cube length
        N = 400
```

```
    n = 100
    l = 0.5

    # Generating and transfomring array from [0,1] to [-0.5, 0.5]
    array = np.random.rand(N, n)
    array = l*(1 - 2*array)

    # Finding number of operations to initilize arrays
    num_operations = comb(N, 2)
    theta_arr = np.zeros(num_operations)
    d_arr = np.zeros(num_operations)
    count = 0
    for i in range(0, N):
        x1 = array[i,:]
        for j in range(i, N - 1):
            x2 = array[j + 1, :]
            d_arr[count] = calcDistance(x1, x2)
            theta_arr[count] = calcAngle(x1, x2)
            count += 1

    # Plotting Results
    plt.hist(d_arr, label='Distance')
    plt.hist(theta_arr, label='Theta')
    plt.ylabel('Number of Values')
    plt.xlabel('Values')
    plt.legend(loc="upper left")
    plt.title('Orthogonality and Euclidean Norm in Higher Dimensions')
    plt.show()

# ==========================================================================
if __name__ == "__main__":
    main()

import numpy as np
import matplotlib.pyplot as plt

# ==========================================================================
def plotRejectionPoints(n, r):
    # x-axis and plotting
    n_arr = np.arange(1, n + 1)
    plt.plot(n_arr, r)

    # Graph customization
    plt.xlabel('Dimension of n')
    plt.ylabel('Number of Rejected Points')
    plt.title('How the Position of Points Change as Dimension Grows')
    plt.show()
```

```
# ==========================================================================
def createRejectionPoints(N, n, r):

        # Create rejcetions arry for each dimension
        rejections = np.zeros(n)

        # Looping over the dimensions for each experiment (sampling N times for
        for k in range(1, n + 1):
                x = np.zeros((N, k))
                # Creating N samples per dimension
                for i in range(0, N):
                        # Creating points in n-dimensions
                        for j in range(0, k):
                                u = np.random.uniform(0, 1)
                                x[i][j] = r*(1 - 2*u)
                        # Rejecting points if the L-2 norm falls outside the sph
                        x_norm = np.linalg.norm(x[i, :], 2)
                        if x_norm > r:
                                rejections[k - 1] += 1

        return rejections

# ==========================================================================
def main():
        # Constants
        N = 10000
        n = 400
        r = 1

        # Generating and plotting
        rejection_arr = createRejectionPoints(N, n, r)
        plotRejectionPoints(n, rejection_arr)

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