DS303 - Statistical Foundations for Data Science
Feb-June 2021
Report for Assignment 7

Gajraj Singh Chouhan (B19130)

June 16, 2021

1 Questions

1.1 Question 1

We have to estimate π using Monte Carlo Simulation.

Because the area of a circle is πr^2, we can calculate the value of π using given parameters.

I chose a square of side length 2 units, and the circle would be of radius 1 unit. In the x-y plane, the square lies in (-1, 1) and the center of circle was at (0, 0).

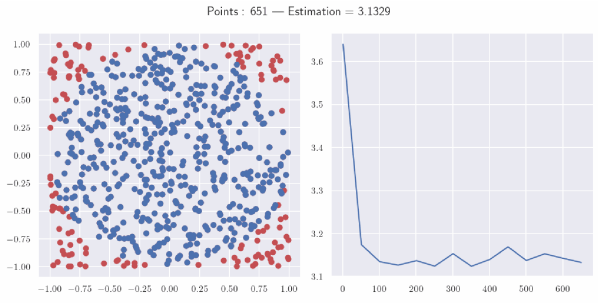
$$\frac{\pi * 1^2}{2^2} = \text{fraction of points lying in circle} \quad (1)$$

To check if a point (x, y) lied in the circle, we can use the equation of a circle.

$$x^2 + y^2 \leq 1 \quad (2)$$

Thus we get the fraction of points, so π can be calculated.

The animation was saved to a **.mp4** file.



This image shows blue points lying in the circle and red ones outside. The line plot on the right shows the value of pi we get.

As seen, the value of pi gets more accurate as points increase.

Up to 3000 points were considered in this simulation.

For the confidence interval, samples were generated 100 times for each number of points from 1-3000.

```
estimates = []
for _ in range(100):
    # generate 100 times to make sure
        we calculate the Confidence
        Interval
    area = self.area_func(self.low,
        self.high, points)
    mask = self.mask_func(area)
    masked = area[mask]
    estimate = self.estimate_func(self.
        high, self.low, masked, points)
    estimates.append(estimate)
```

```
self.estimates.append(estimates)

# mean of 100 estimates is used
estimate = np.mean(estimates)
```

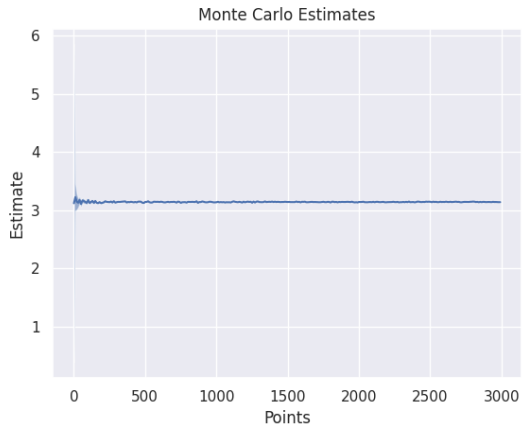
We can calculate the confidence interval using,

$$CI = \bar{x} \pm z\frac{\sigma}{\sqrt{n}} \tag{3}$$

Where z is the z-value for our CI (90%), σ is the standard deviation, n is number of points.

In the code,

```
mu = x.mean(axis=1)
sd = np.sqrt(x.var(axis=1))
ci = (1.645 * sd) / points  # calculate
    the confidence intervals
plt.plot(self.l, mu)
plt.xlabel("Points")
plt.ylabel("Estimate")
plt.fill_between(self.l, (mu - ci), (mu
    + ci), alpha=0.5)
```

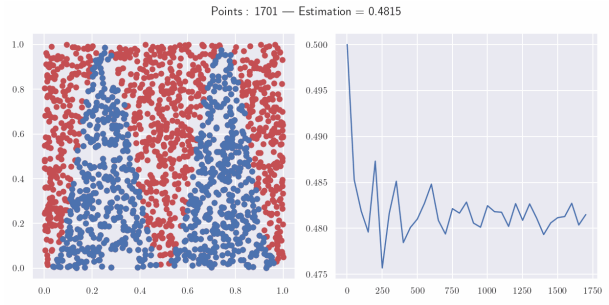


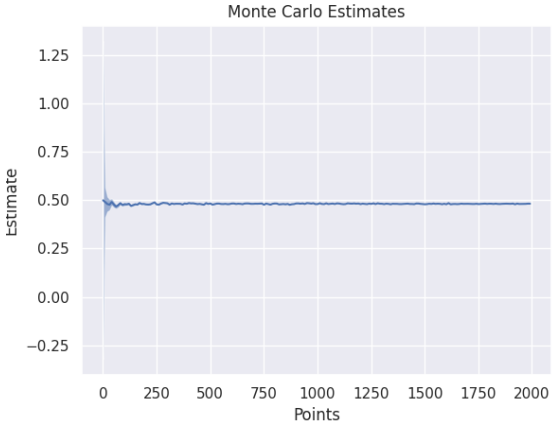
As expected the confidence interval denoted in the blue area, decreases as points increases.

1.2 Question 2

Here, we also estimate the area under the function $f(x)$ from 0 to 1 in a square of side 1 unit.

To check whether a point (x, y) lies under the function, use $f(x) \leq y$. Since the height of curve would always be greater than the points under it.





1.3 Question 3

Here, we use Rejection Sampling method which samples from proposed distribution, then accepts the sample for target distribution based on likelihood ratio.

Take,

$$M \geq \frac{f(x)}{g(x)} = \sqrt{\frac{\pi}{2}} \times e^{\frac{-x^2}{2}} \times (1 + x^2) \tag{4}$$

Minimum value of right size of inequality occurs at $x = 1$.

Therefore,

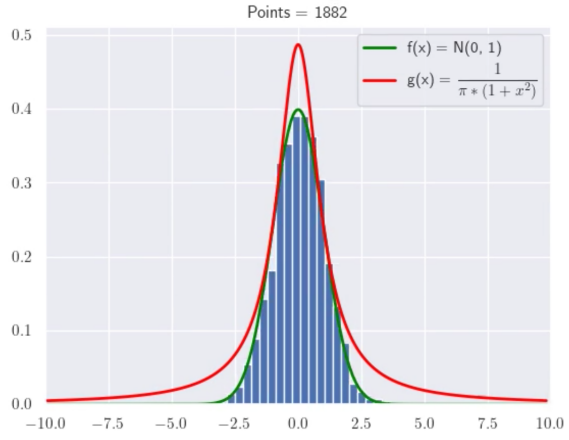
$$M \geq \sqrt{\frac{\pi}{2}} \times e^{-\frac{1}{2}} \times 2 \approx 1.5203 \tag{5}$$

```
u = np.random.default_rng().uniform
    (0.0, 1.0, (points, 1)).reshape(-1,
     1)
x = np.array([])

while len(x) < points:
    X = np.random.default_rng().
        standard_cauchy()
    if 0 <= abs(X) <= 10:
        x = np.append(x, X)
x = x.reshape(-1, 1)

r = (f(x)/g(x))/M
msk = np.ma.masked_where(u <= r, u)  #
    Points which will not be rejected

ax.plot(plot_x, f(plot_x), color="green
    ", label="f(x) = N(0, 1)",)
ax.plot(plot_x, M * g(plot_x), color="
    green", label="g(x)")
ax.hist(x[msk.mask], bins=20, density=
    True) # plot histogram
```



As the above graph shows, the histogram is aligning with our target distribution after sampling.

1.4 Question 4

Here Importance Sampling is used to estimate the different expectations.

In Importance Sampling, we sample from the proposed distribution.

Let $f(x)$ be the function whose expectation we wanna estimate where $x \sim p$.

$$E[f] = \int f(x)p(x)dx = \int \frac{f(x)p(x)}{g(x)}g(x)dx \tag{6}$$

where g is a distribution which is easy to sample from.

$$E_p[f] = E_g[\frac{f(x)p(x)}{g(x)}] \approx \frac{1}{N}\sum_{x_i=1}^{x_i=N}\frac{f(x_i)p(x_i)}{g(x_i)} \tag{7}$$

We sample from a $g(x) = N(0, 4)$ normal distribution.

The variance of importance sampling tells us how good our proposed distribution is.

Given by,

$$Var(\hat{I}_n) = \frac{1}{N}(\int \frac{f^2(z)p^2(z)}{g(z)}dz - I^2) \tag{8}$$

where is I is the true value. As seen in the table below $E[x^p]$ where p is increasing.

The increase in p also causing the variance to increase, which tell us the given $g(x)$ is not a good proposed distribution for larger exponents like x^5.

Points	Result	Error	Variance
100	0.0289	0.0289	1.101
500	-0.0168	0.0168	1.070
1000	0.0017	0.0017	1.151
2000	0.0023	0.0023	1.157
3000	-0.0376	0.0376	1.146
5000	-0.0001	0.0001	1.153
10000	-0.0050	0.0050	1.139

Table 1: $E[x^1]$, True Value: 0

Points	Result	Error	Variance
100	2.031	0.0315	3.108
500	1.881	0.1185	4.365
1000	1.993	0.0066	6.550
2000	1.942	0.0574	7.535
3000	1.951	0.0486	5.965
5000	1.981	0.0188	6.020
10000	1.940	0.0599	6.157

Table 2: $E[x^2]$, True Value: 2

Points	Result	Error	Variance
100	10.427	10.427	32333.363
500	-0.019	0.019	94099.982
1000	0.422	0.422	316813.366
2000	-15.980	15.980	532537.996
3000	-9.013	9.013	318931.616
5000	8.165	8.165	305142.641
10000	-7.361	7.361	343700.943

Table 3: $E[x^5]$, True Value: 0

1.5 Question 5

In inverse transform sampling method, use the CDF of a distribution for sampling from the distribution.

It follows the equation $X = F^{-1}(U)$ which is using the inverse CDF, hence the name.

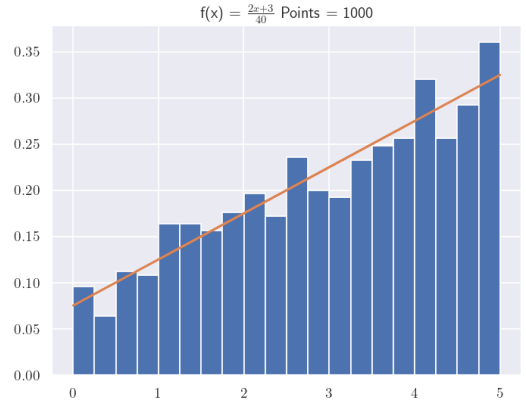
U is the standard uniform distribution.

The inverse CDF of distribution will be,

$$F^{-1}(x) = (40 \times u + \frac{9}{4})^{0.5} - \frac{3}{2} \tag{9}$$

To sample the uniform distribution, used numpy *random.uniform* function to generate 1000 points between 0 and 1.

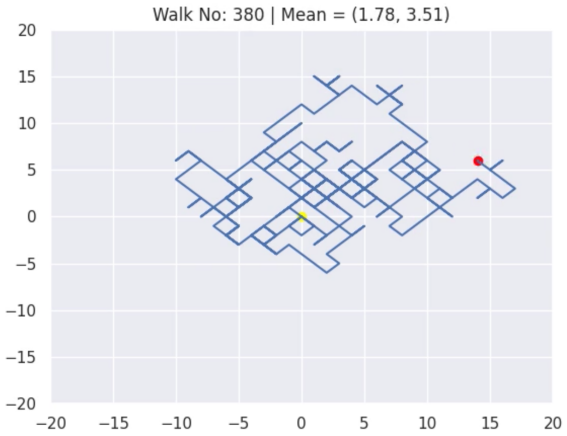
```
inv_cdf = np.vectorize(lambda u: (40 *
    u + 9 / 4) ** 0.5 - 3 / 2)
u = np.random.default_rng().uniform
    (0.0, 1.0, (points, 1))
samples = inv_cdf(u)
plt.hist(samples, density=True, bins
    =20)
plt.show()
```



We use the histogram of generated points to confirm it is matching pdf of distribution.

1.6 Question 6

Here for 2D random walk, let us assume there are four directions. We choose any one of these four directions in 2D grid, and take a step in that direction.



1.7 Question 7

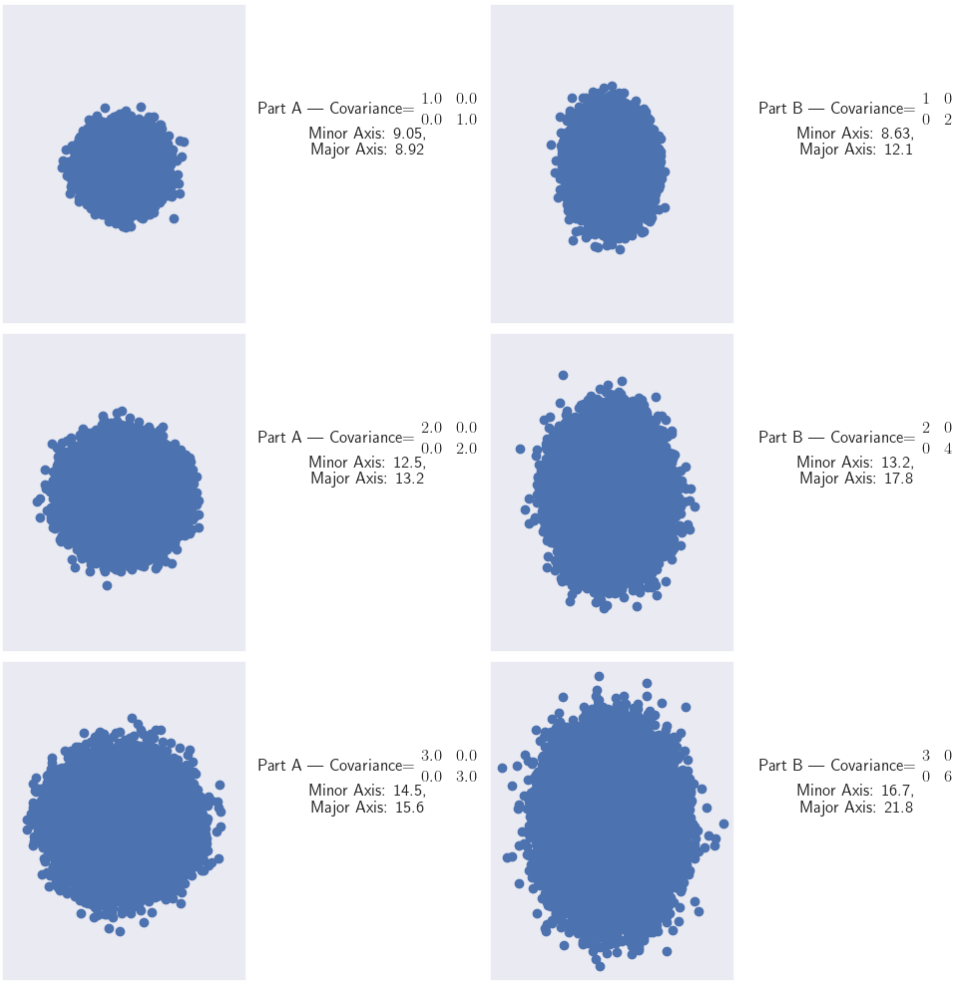
In this question, we analyze samples of a bivariate normal parameters. 100k samples were generated using numpy's *random.multivariate_normal* function.

A bivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ takes μ (mean) and σ (covariance matrix) as parameters. We only use a zero matrix as the mean here. In Part A and B, we used a diagonal matrix as covariance.

Part A,

$$\Sigma = c \times I \text{ where c } \in R^{+} \text{ and } I \text{ is } 2 \times 2 \text{ identity matrix.} \tag{10}$$

c varies from 1 to 3, as we can see this is producing a circle and radius of circle is increasing with value of c. Major and Minor axis are approximately equal (i.e. radius) with some error due to few outliers.



$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim N\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix}\right] \tag{11}$$

$$|\Sigma| = \sigma_1^2\sigma_2^2, \Sigma^{-1} = \frac{1}{\sigma_1^2\sigma_2^2}\begin{pmatrix} \sigma_2^2 & 0 \\ 0 & \sigma_1^2 \end{pmatrix} \tag{12}$$

$$\phi(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2}\exp\{-\frac{1}{2}\left[\left(\frac{x_1}{\sigma_1}\right)^2 + \left(\frac{x_2}{\sigma_2}\right)^2\right]\} \tag{13}$$

We can see this is a perfectly symmetric curve in all the dimensions (also seen in the plots above). Now for a constant 'z' in 3D, sampled points (x_1, x_2) lies on the below equation.

$$c = \left(\frac{x_1}{\sigma_1}\right)^2 + \left(\frac{x_2}{\sigma_2}\right)^2 \tag{14}$$

This equation is a locus of an ellipse centered at our mean, if variance are equal it will give us the locus of a circle.

Width of axes of ellipse is $2a$ and $2b$ where,

$$a = \sqrt{c}\sigma_1 \ \& \ b = \sqrt{c}\sigma_2 \tag{15}$$

Part A,

$$\sigma_1 = \sigma_2 = \sigma \tag{16}$$

Radius of circle is $\sqrt{c}\sigma$.

In Part B,

$$\Sigma = diag(a_1,\ a_2) \text{ where } a_1 \text{ and } a_2 \in R^+ \tag{17}$$

Now this will not be producing the samples inside a circle rather in an ellipse, whose length of the axis along y axis will be greater than x axis.

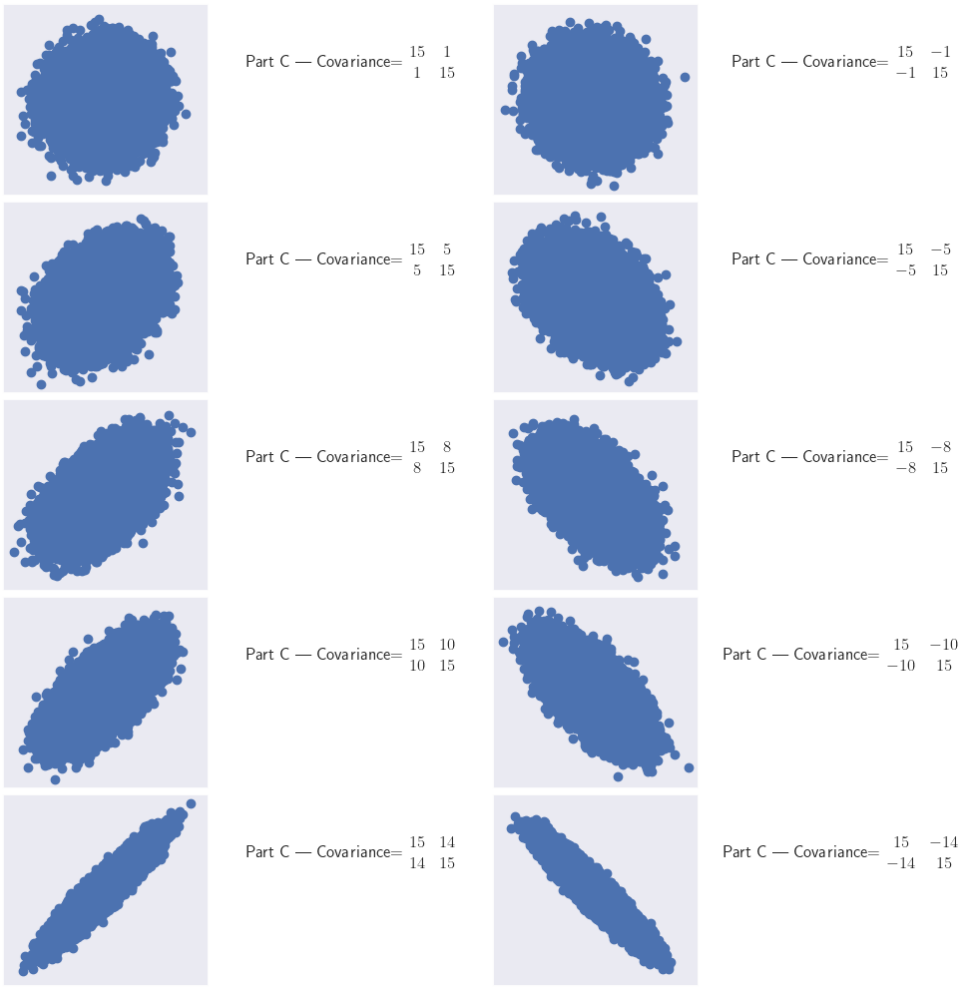
In Part C,

Σ is a symmetric matrix with fixed values of diagonal elements. We took them as 15 and the off diaognal elements varies as $1, 4, 8, 10, 15$.

Now as the off diagonal element increases, the ellipse will start to rotate and variance along that axis will increase while decreasing variance in the other axis.

The diagonal matrix's elements on main diagonal specify the "variance" in the x and y direction.

For negative elements, the direction is also opposite (as correlation will be negative).



Note that the eigen vectors of the diagonal matrix will decide the direction of axes of ellipse.

This will also hold true for more dimensions.

This property of variance & direction of ellipse made by multivariate normal distribution does not appear to be for other distribution, although for a large number of samples distributions will approximate to be like normal distribution (due to CLT) hence it is applicable there.

For a general covariance matrix, proof of directions can be found in the links below.

2 References

All the animations, data generation and manipulation were done in Python using matplotlib, pandas, numpy and standard library modules.

LaTeXis required to compile the text using matplotlib in Python scripts. Please see the documentation `https://matplotlib.org/stable/tutorials/text/usetex.html`

1. Animation, https://stackoverflow.com/a/63454331/12350727

2. Random Walk, Weisstein, Eric W. "Random Walk–2-Dimensional." From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/RandomWalk2-Dimensional.html

3. Multivariate Normal Distribution https://online.stat.psu.edu/stat505/book/export/html/636

4. Rotation of Ellipse in matrix form https://en.wikipedia.org/wiki/Matrix_representation_of_conic_sections

5. Central Limit Theorem https://en.wikipedia.org/wiki/Central_limit_theorem