MA3206: Statistics I

Estimating π by rejection sampling

Spring 2022

Satvik Saha
19MS154

Indian Institute of Science Education and Research, Kolkata,
Mohanpur, West Bengal, 741246, India.

Contents

1 Problem statement

Consider the unit disc,

$$D = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \leq 1\},$$

and the ellipse

$$D' = \{(x, y) \in \mathbb{R}^2 : 2(x-1)^2 + 5(y-2)^2 \leq 8\}.$$

The task is to sample some n points from D and D', and thereby estimate the value of π. For reference,

$$\pi = 3.141592653589793\ldots$$

2 Rejection sampling

Consider a region $A \subset \mathbb{R}^2$, from which we want to sample points uniformly. In other words, we want to simulate the probability density function

$$u_A \colon \mathbb{R}^2 \to \mathbb{R}, \qquad u_A(x, y) = \frac{1}{\mu(A)} \chi_A(x, y) = \begin{cases} 1/\mu(A), & \text{if } (x, y) \in A, \\ 0, & \text{otherwise.} \end{cases}$$

Here, for $E \subseteq \mathbb{R}^2$, we denote

$$\mu(E) = \int_E dx\, dy,$$

the 'area' or measure of E. Now suppose that A is enclosed within a rectangle $R = [a, b] \times [c, d]$. We can simulate a uniform distribution on R easily enough, by sampling individual coordinates $x \in [a, b]$, $y \in [c, d]$ uniformly. This means that we can simulate the probability density function

$$u_R \colon \mathbb{R}^2 \to \mathbb{R}, \qquad u_R(x, y) = \frac{1}{\mu(R)} \chi_R(x, y) = \begin{cases} 1/\mu(R), & \text{if } (x, y) \in R, \\ 0, & \text{otherwise.} \end{cases}$$

Note that we have normalized

$$\int_{\mathbb{R}^2} u_A(x, y)\, dx\, dy = \int_{\mathbb{R}^2} u_R(x, y)\, dx\, dy = 1.$$

Also, $\mu(R) = (b - a)(d - c)$.

In order to simulate the distribution u_A, we propose rejection sampling from u_R. In other words, draw a point $(x, y) \in R$ using the distribution u_R. If $(x, y) \in A$, accept this as a sample point; if not, reject it. Repeat this process of drawing points from R and checking until we have all n sample points.

Consider a point $(X, Y) \in R$ drawn using the distribution u_R. We compute the probability that it is accepted as follows.

$$\begin{aligned} P[(X, Y) \text{ accepted}] &= \int_R P[(X, Y) \text{ accepted} \mid X = x, Y = y]\, u_R(x, y)\, dx\, dy \\ &= \int_R \chi_A(x, y)\, u_R(x, y)\, dx\, dy \\ &= \int_A \frac{1}{\mu(R)}\, dx\, dy \\ &= \frac{\mu(A)}{\mu(R)}. \end{aligned}$$

The second line follows from the fact that (X, Y) is accepted if and only if $(X, Y) \in A$. This result aligns well with our intuition that the probability of acceptance ought to be the ratio of areas of A and R. If N points are drawn from R out of which n were accepted, we approximate

$$P[(X, Y) \text{ accepted}] = \frac{\mu(A)}{\mu(R)} \approx \frac{n}{N}.$$

Now, we show that points sampled in this way are indeed distributed uniformly over A, as in u_A. Let $E \subseteq A$ be an arbitrary (measurable) region. Consider an accepted point $(X, Y) \in A$ in our sample. Using Bayes' Theorem, we have

$$P[(X, Y) \in E \mid (X, Y) \text{ accepted}] = P[(X, Y) \text{ accepted} \mid (X, Y) \in E] \cdot \frac{P[(X, Y) \in E]}{P[(X, Y) \text{ accepted}]}.$$

Note that (X, Y) is always accepted if $(X, Y) \in E \subseteq$A. Also, (X, Y) was chosen uniformly from R using u_R, so we can compute much like before

$$\begin{aligned} P[(X, Y) \in E] &= \int_R P[(X, Y) \in E \mid X = x, Y = y]\, u_R(x, y)\, dx\, dy \\ &= \int_R \chi_E(x, y)\, u_R(x, y)\, dx\, dy \\ &= \int_E \frac{1}{\mu(R)}\, dx\, dy \\ &= \frac{\mu(E)}{\mu(R)}. \end{aligned}$$

Putting things together,

$$P[(X, Y) \in E \mid (X, Y) \text{ accepted}] = \frac{\mu(E)}{\mu(R)} \cdot \frac{\mu(R)}{\mu(A)} = \frac{\mu(E)}{\mu(A)}.$$

However, this is precisely the defining property of the uniform distribution on A described by u_A; the probability of a sample point (X, Y) lying within an arbitrary region $E \subseteq A$ being proportional to the area of E. Thus, our sample obtained in this way does indeed follow distribution u_A.

2.1 The unit disc

The unit disc D is enclosed within the square $S = [-1, 1] \times [-1, 1]$. Thus, we can calculate the areas $\mu(D) = \pi$, $\mu(S) = 4$. By setting $A = D$, $R = S$, we can use the outlined method to draw sample points uniformly from D. If n samples points are obtained over N draws, we approximate

$$\frac{\pi}{4} \approx \frac{n}{N}.$$

2.2 The ellipse

By rewriting the defining equation for D' as

$$\frac{(x-1)^2}{4} + \frac{(y-2)^2}{8/5} \leq 1,$$

it is clear that this is an elliptical region centred at $(1, 2)$, with major and minor axes parallel to the x and y axes respectively. Furthermore, the semi-major and semi-minor axes have lengths 2 and $\sqrt{8/5}$ respectively.

The ellipse D' is thus enclosed within the square $S' = [-1, 3] \times [0, 4]$. We can calculate the areas $\mu(D') = \pi \cdot 2 \cdot \sqrt{8/5}$, $\mu(S') = 16$. By setting $A = D'$, $R = S'$, we can use the outlined method to draw sample points uniformly from D'. If n samples points are obtained over N draws, we approximate

$$\frac{\pi \cdot 2 \cdot \sqrt{8/5}}{16} = \frac{\pi}{\sqrt{40}} \approx \frac{n}{N}.$$

3 Algorithm

In summary, here is the algorithm used to sample n points from a region $A \subset \mathbb{R}^2$ with uniform distribution u_A, where A is enclosed within the rectangle $R = [a, b] \times [c, d]$.

1. Select $x_i \in [a, b]$ and $y_i \in [c, d]$ uniformly. In other words, select (x_i, y_i) uniformly from the rectangle R.
2. Check whether $(x_i, y_i) \in A$. If so, add the point (x_i, y_i) to a pile of accepted points, otherwise add it to a pile of rejected points.
3. Repeat steps 1, 2 until the accepted pile has n points.
4. Let the total number of points in the accepted and rejected piles be N. Then, we have $\mu(A)/\mu(R) \approx n/N$.
   (a) For the unit disc, we use the approximation $\pi \approx 4n/N$.
   (b) For the ellipse, we use the approximation $\pi \approx \sqrt{40}n/N$.

4 Program execution

4.1 The unit disc

The above algorithm for the unit disc has been implemented in the python script `sample_disc.py`, listed in Sec. 5. The script also plots the data points for better visualisation.

Below is an execution of the script, with $n = 10$ samples drawn from the unit disc. Each line (apart from the summary at the end) shows the x and y coordinate of the data point, along with whether the point was accepted or rejected.

```
$ ./sample_disc.py 10
  0.73233093   -0.29713792   accepted
  0.34659601    0.05158366   accepted
  0.52026872   -0.75515856   accepted
 -0.22484447   -0.80110737   accepted
  0.32262891   -0.11055622   accepted
  0.52150728   -0.90408779   rejected
```

```
 -0.85531360    0.18878968    accepted
 -0.05718717   -0.43605130    accepted
  0.83832780   -0.81159708    rejected
 -0.04045486   -0.97506796    accepted
 -0.25494000    0.36426489    accepted
  0.33717710   -0.13154315    accepted
10 samples accepted out of 12 draws.
Estimated value of pi is 3.33333333
Error is 0.19174068 (6.103295%)
```

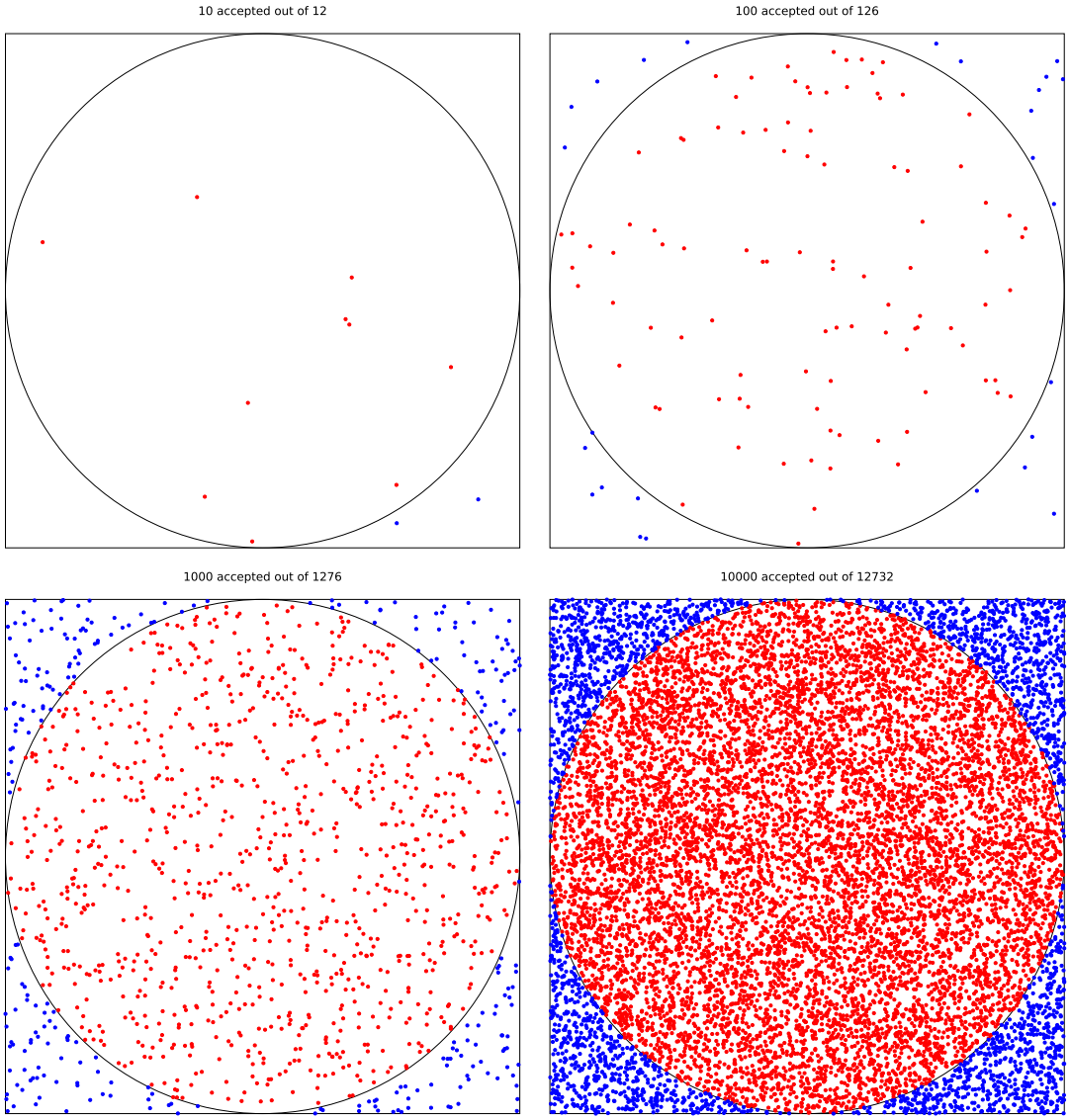

Figure 1: Samples drawn from the unit disc in red, rejected points in blue, $n = 10, 100, 1000, 10000$.

Following are executions of the script with $n = 100, 1000, 10^4, 10^5, 10^6$ samples drawn. The data points have been redirected to separate files for brevity.

```
$ ./sample_disc.py 100 2> samples_disc_100.dat
100 samples accepted out of 126 draws.
Estimated value of pi is 3.17460317
Error is 0.03301035 (1.050752%)
```

```
$ ./sample_disc.py 1000 2> samples_disc_1000.dat
1000 samples accepted out of 1276 draws.
Estimated value of pi is 3.13479624
Error is -0.00679641 (-0.216337%)

$ ./sample_disc.py 10000 2> samples_disc_10000.dat
10000 samples accepted out of 12732 draws.
Estimated value of pi is 3.14169023
Error is 0.00009758 (0.003106%)

$ ./sample_disc.py 100000 2> samples_disc_100000.dat
100000 samples accepted out of 127433 draws.
Estimated value of pi is 3.13890437
Error is -0.00268828 (-0.085571%)

$ ./sample_disc.py 1000000 2> samples_disc_1000000.dat
1000000 samples accepted out of 1272742 draws.
Estimated value of pi is 3.14282078
Error is 0.00122813 (0.039092%)
```

The `sample_disc_mean.py` script performs this sampling process for multiple runs, averaging the estimates of π from each run, and displaying the π estimates as a histogram.

```
$ ./sample_disc_mean.py 1000 1000 2> /dev/null
Mean of pi estimates from 1000 runs, each with 1000 samples, is
3.142724742209441
Error is 0.00113209 (0.036035%)
```

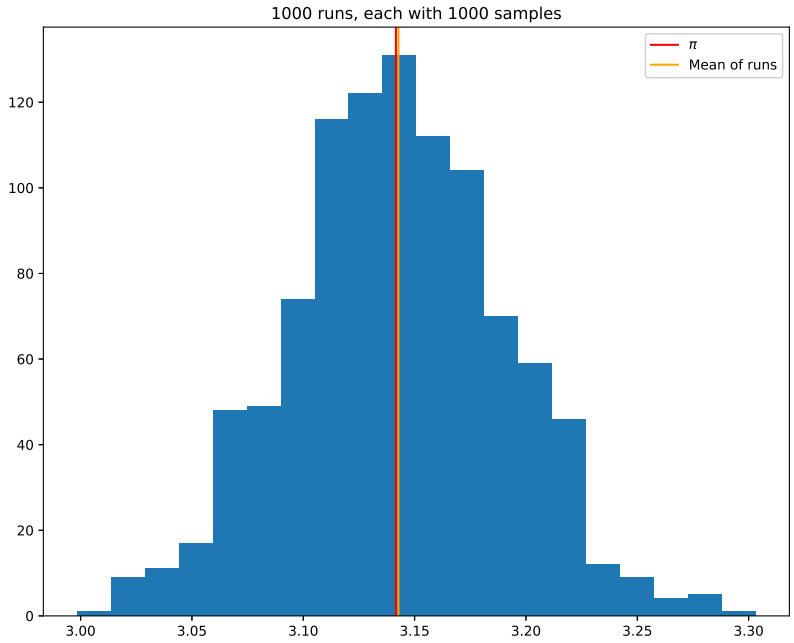

Figure 2: Histogram of estimates of π obtained from 1000 runs, each of which samples $n = 1000$ points.

In order to obtain much faster execution speeds, the algorithm for the unit disc has also been implemented in the C++ program `sample_disc.cpp`, listed in Sec. 5. Below are a few executions for $n = 10^5, 10^6, 10^7, 10^8$.

```
$ g++ -o sample_disc sample_disc.cpp
```

```
$ ./sample_disc 100000
100000 accepted, 127441 total, pi estimated as 3.138707
Error is -0.002885 (-0.091843%)

$ ./sample_disc 1000000
1000000 accepted, 1273707 total, pi estimated as 3.140440
Error is -0.001153 (-0.036700%)

$ ./sample_disc 10000000
10000000 accepted, 12733700 total, pi estimated as 3.141271
Error is -0.000322 (-0.010245%)

$ ./sample_disc 100000000
100000000 accepted, 127320126 total, pi estimated as 3.141687
Error is 0.000094 (0.003007%)
```

4.2 The ellipse

The algorithm for the ellipse has been implemented in the python script `sample_ellipse.py`, listed in Sec. 5. The script also plots the data points for better visualisation. Below is an execution of the script, with $n = 10$ samples drawn from the unit disc.

```
$ ./sample_ellipse.py 10
  2.17950199   3.73800320   rejected
  1.08590943   1.97214632   accepted
 -0.72460138   0.29850537   rejected
  0.14060985   3.13547978   accepted
  2.43026007   0.00517840   rejected
  1.69725367   0.37317408   rejected
  2.98606978   2.04202534   accepted
 -0.75396791   1.43706716   accepted
  2.02528250   1.24398366   accepted
 -0.57610408   0.60839471   rejected
  0.65326239   3.43285296   rejected
  2.02370550   2.07640832   accepted
  2.71615959   0.07882299   rejected
  2.75012591   3.90455867   rejected
  2.35430809   0.99566044   rejected
  1.04111028   0.61737784   rejected
 -0.25423444   2.30809614   accepted
 -0.52347198   3.03817098   rejected
  1.97866215   0.44457484   rejected
  0.27278245   1.30040501   accepted
  1.38116679   1.38268388   accepted
  2.71280690   1.65653127   accepted
10 samples accepted out of 22 draws.
Estimated value of pi is 2.87479787
Error is -0.26679478 (-8.492342%)
```

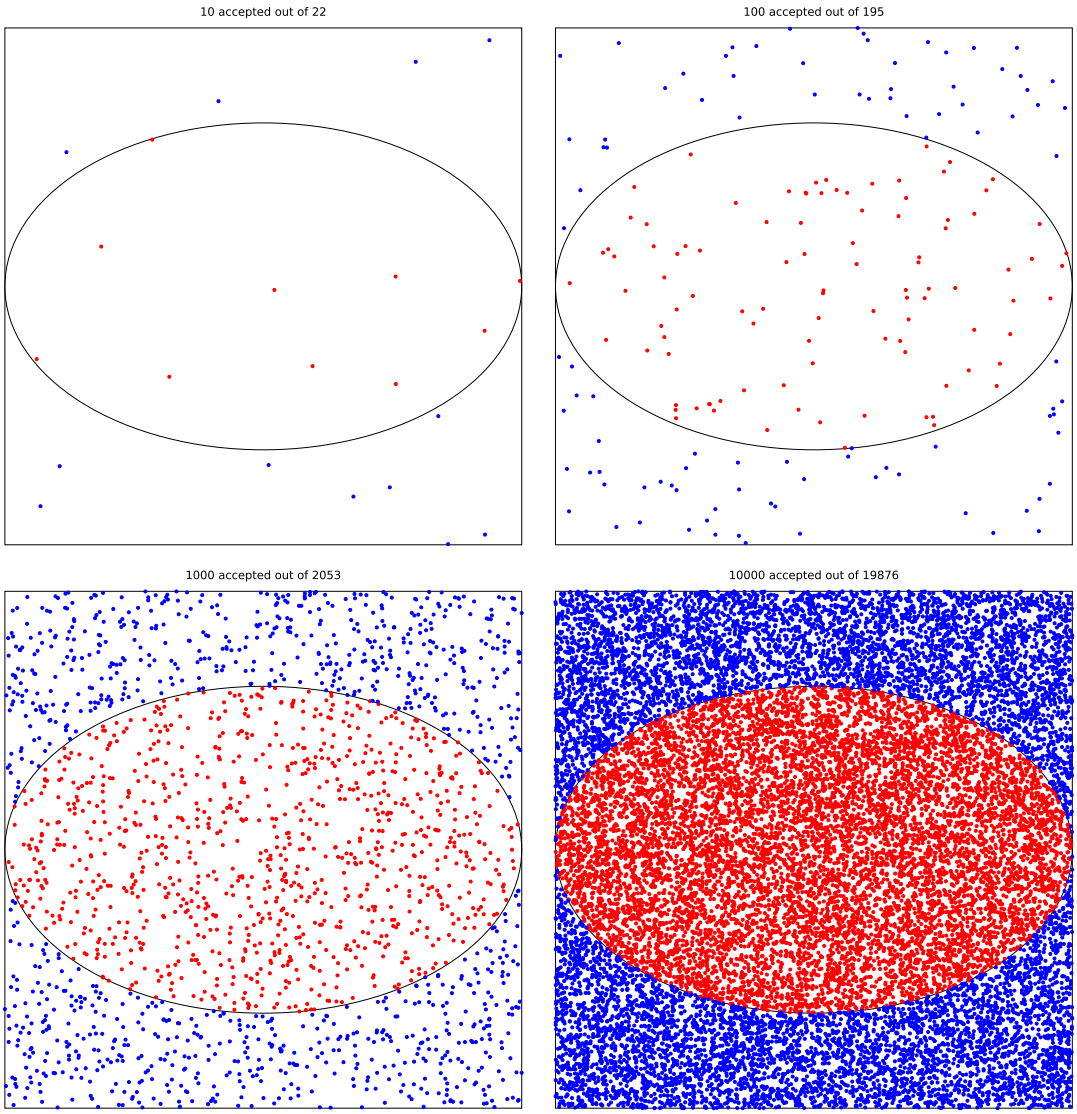

Figure 3: Samples drawn from the ellipse in red, rejected points in blue, $n = 10, 100, 1000, 10000$.

Following are executions of the script with $n = 100, 1000, 10^4, 10^5, 10^6$ samples drawn. The data points have been redirected to separate files for brevity.

```
$ ./sample_ellipse.py 100 2> samples_ellipse_100.dat
100 samples accepted out of 195 draws.
Estimated value of pi is 3.24336170
Error is 0.10176905 (3.239409%)

$ ./sample_ellipse.py 1000 2> samples_ellipse_1000.dat
1000 samples accepted out of 2053 draws.
Estimated value of pi is 3.08064068
Error is -0.06095197 (-1.940162%)

$ ./sample_ellipse.py 10000 2> samples_ellipse_10000.dat
10000 samples accepted out of 19876 draws.
Estimated value of pi is 3.18200610
Error is 0.04041344 (1.286400%)

$ ./sample_ellipse.py 100000 2> samples_ellipse_100000.dat
100000 samples accepted out of 201676 draws.
Estimated value of pi is 3.13599800
```

```
Error is -0.00559466 (-0.178083%)

$ ./sample_ellipse.py 1000000 2> samples_ellipse_1000000.dat
1000000 samples accepted out of 2012042 draws.
Estimated value of pi is 3.14335154
Error is 0.00175889 (0.055987%)
```

5 Code listings

Listing 1: sample_disc.py

```python
#!/usr/bin/env python3

import sys
import matplotlib.pyplot as plt
import numpy as np
from random import uniform


def sample_unit_disc(n):
    # Prepare lists of accepted and rejected points
    accepted = []
    rejected = []
    # Loop until 'n' points from within the unit disc have been accepted
    while len(accepted) < n:
        # Pick a random point uniformly from the square [-1, 1] x [-1, 1]
        x = uniform(-1, 1)
        y = uniform(-1, 1)
        # Check whether (x, y) falls in the unit disc
        if x**2 + y**2 <= 1:
            accepted.append((x, y))
            print(f"{x:12.8f} {y:12.8f}   accepted", file=sys.stderr)
        else:
            rejected.append((x, y))
            print(f"{x:12.8f} {y:12.8f}   rejected", file=sys.stderr)
    return accepted, rejected


def plot_points(accepted, rejected):
    fig, ax = plt.subplots()
    plt.axis("off")
    ax.axis("equal")
    fig.set_size_inches(10, 10)

    ax.add_patch(plt.Circle((0, 0), 1, fill=False))
    ax.add_patch(plt.Rectangle((-1, -1), 2, 2, fill=False))

    plt.scatter(*zip(*accepted), color="red", s=10)
    plt.scatter(*zip(*rejected), color="blue", s=10)

    plt.axis((-1.05, 1.05, -1.05, 1.05))

    a = len(accepted)
    total = len(accepted) + len(rejected)
    plt.title(f"{a} accepted out of {total}")
    plt.show()


def main():
    # First command line argument is the number of sample points to be drawn
    n = int(sys.argv[1])
    # Sample 'n' points from the unit disc
    accepted, rejected = sample_unit_disc(n)
    # Estimate 'pi' using the areas of the square and the disc
    total = len(accepted) + len(rejected)
    pi_estimate = 4 * (n / total)
    error = pi_estimate - np.pi
    error_rel = 100 * error / np.pi
```

```
58
59      print(f"{n} samples accepted out of {total} draws.")
60      print(f"Estimated value of pi is {pi_estimate:.8f}")
61      print(f"Error is {error:.8f} ({error_rel:.6f}%)")
62
63      plot_points(accepted, rejected)
64
65
66  if __name__ == "__main__":
67      main()
```

Listing 2: `sample_disc_mean.py`

```
1   #!/usr/bin/env python3
2
3   import sys
4   import matplotlib.pyplot as plt
5   import numpy as np
6   from sample_disc import sample_unit_disc
7
8
9   def pi_estimate(samples):
10      # Estimate the value of pi using 'samples' sample points
11      accepted, rejected = sample_unit_disc(samples)
12      total = len(accepted) + len(rejected)
13      return 4 * (samples / total)
14
15
16  def main():
17      # First command line argument is the number of runs to perform
18      # Second command line argument is the number of sample points to be drawn per run
19      runs, samples = int(sys.argv[1]), int(sys.argv[2])
20      # Calculate the estimates
21      estimates = [pi_estimate(samples) for i in range(runs)]
22      mean_pi_estimate = sum(estimates) / runs
23      error = mean_pi_estimate - np.pi
24      error_rel = 100 * error / np.pi
25
26      print(
27          f"Mean of pi estimates from {runs} runs, each with {samples} samples, is {
        mean_pi_estimate}"
28      )
29      print(f"Error is {error:.8f} ({error_rel:.6f}%)")
30
31      plt.hist(estimates, bins=20)
32      plt.axvline(x=np.pi, color="red", label="$\pi$")
33      plt.axvline(x=mean_pi_estimate, color="orange", label="Mean of runs")
34      plt.title(f"{runs} runs, each with {samples} samples")
35      plt.legend()
36      plt.show()
37
38
39  if __name__ == "__main__":
40      main()
```

Listing 3: `sample_ellipse.py`

```
1   #!/usr/bin/env python3
2
3   import sys
4   import matplotlib.pyplot as plt
5   from matplotlib.patches import Ellipse
6   import numpy as np
7   from random import uniform
8
9
10  def sample_ellipse(n):
11      accepted = []
12      rejected = []
13      while len(accepted) < n:
14          x = uniform(-1, 3)
15          y = uniform(0, 4)
16          if 2 * (x - 1) ** 2 + 5 * (y - 2) ** 2 <= 8:
```

```
            accepted.append((x, y))
            print(f"{x:12.8f} {y:12.8f}   accepted", file=sys.stderr)
        else:
            rejected.append((x, y))
            print(f"{x:12.8f} {y:12.8f}   rejected", file=sys.stderr)
    return accepted, rejected


def plot_points(accepted, rejected):
    fig, ax = plt.subplots()
    plt.axis("off")
    ax.axis("equal")
    fig.set_size_inches(10, 10)

    ax.add_patch(Ellipse(xy=(1, 2), width=4, height=2 * np.sqrt(8 / 5), fill=False))
    ax.add_patch(plt.Rectangle((-1, 0), 4, 4, fill=False))

    plt.scatter(*zip(*accepted), color="red", s=10)
    plt.scatter(*zip(*rejected), color="blue", s=10)

    plt.axis((-1.05, 3.05, -0.05, 4.05))

    a = len(accepted)
    total = len(accepted) + len(rejected)
    plt.title(f"{a} accepted out of {total}")
    plt.show()


def main():
    # First command line argument is the number of sample points to be drawn
    n = int(sys.argv[1])
    # Sample 'n' points from the ellipse
    accepted, rejected = sample_ellipse(n)
    # Estimate 'pi' using the areas of the square and the ellipse
    total = len(accepted) + len(rejected)
    pi_estimate = np.sqrt(40) * (n / total)
    error = pi_estimate - np.pi
    error_rel = 100 * error / np.pi

    print(f"{n} samples accepted out of {total} draws.")
    print(f"Estimated value of pi is {pi_estimate:.8f}")
    print(f"Error is {error:.8f} ({error_rel:.6f}%)")

    plot_points(accepted, rejected)


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

Listing 4: `sample_disc.cpp`

```
#include <random>
#include <cstdlib>

const double pi = 3.14159265358979323846;

int main(int argc, const char *argv[]) {
        long n = 100;
        if (argc > 1)
                n = std::stol(argv[1]);

        // Set up random number generator
        std::random_device rd;
        std::mt19937 generator(rd());
        std::uniform_real_distribution<> distribution(-1, 1);

        long accepted = 0;
        long rejected = 0;

        // Loop until 'n' points accepted
        while (accepted < n) {
                // Pick random coordinates from the square
                double x = distribution(generator);
```

```
23                  double y = distribution(generator);
24                  if (x*x + y*y <= 1) {
25                          accepted += 1;
26                  } else {
27                          rejected += 1;
28                  }
29          }
30
31          // Calculate pi estimates, errors
32          long total = accepted + rejected;
33          double pi_estimate = 4.0 * accepted / total;
34          double error = pi_estimate - pi;
35          double error_rel = 100 * error / pi;
36
37          printf("%d accepted, %d total, pi estimated as %f\n", accepted, total,
        pi_estimate);
38          printf("Error is %f (%f%%)\n", error, error_rel);
39
40          return 0;
41  }
```