MC Integration: Rationale

Suppose that we wish to approximate some integral I using simulations. I has the form

$$I = \int g(x)f(x)dx$$

where $g(x)$ is some function of a continuous r.v. X and $f(x)$ is its density. We can draw a sample of size m from the density $f(x)$ and then use the empirical average to get a Monte Carlo estimate of I:

$$\hat{I} = \frac{1}{m}\sum_{i=1}^{m} g(x_i)$$

The standard error $\hat{se}(\hat{I})$ is given by

$$\hat{se}(\hat{I}) = \sqrt{\frac{1}{m^2}\sum_{i=1}^{m}\left(g(x_i) - \hat{I}\right)^2}$$

MC Confidence Intervals

Clearly, we see that the larger the number of realizations m, the lower the standard error. Indeed, by the Law of Large Numbers (LLN), we have that

$$\lim_{m\to\infty} \frac{1}{m} \sum_{i=1}^{m} g(x_i) = E_f[g(x)]$$

or equivalently $\lim_{m\to\infty} \hat{I} = I$. Then, by the Central Limit Theorem (CLT), we have that $\hat{I} \to N(I, se(\hat{I}))$ and the quantity $\frac{\hat{I}-I}{se(\hat{I})} \sim N(0,1)$.

We can then construct a $(1-\alpha)100\%$ Confidence Interval for I

$$\left[\hat{I} - z_{1-\alpha/2}\hat{se}(\hat{I}), \hat{I} + z_{1-\alpha/2}\hat{se}(\hat{I})\right]$$

where $z_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the standard Normal distribution.

Working example (1/2)

Suppose that we wish to evaluate the following integral

$$I = \int_0^{\pi} g(x)dx \qquad \text{where } g(x) = sin(x) + cos(x)$$

We can compute it analytically and we will find that $I = 2$. Using Monte Carlo integration, we will generate $m = 10,000$ realizations of $X \sim U[0, \pi]$ and then use the fact that the PDF of X is

$$f(x) = \begin{cases} 1/(\pi - 0) & \text{if } x \in [0, \pi], \\ 0 & \text{otherwise} \end{cases}$$

In our case, we have

$$I = \int_0^{\pi} g(x)dx = \pi \int_0^{\pi} g(x)\frac{1}{\pi}dx = \pi E_f[g(x)]$$

Working example (2/2)

We will implement a MC integration procedure in R and Python to approximate the integral of interest and its standard error, namely

$$\hat{I} = \frac{\pi}{10,000} \sum_{i=1}^{10,000} g(x_i)$$

$$\hat{se}(\hat{I}) = \frac{\pi}{10,000} \sqrt{\sum_{i=1}^{10,000} \left(g(x_i) - \hat{I} \right)^2}$$

95%-Confidence Intervals spoiler alert: The results are obviously about equivalent. These are the obtained output in R and Python respectively, rounded to 5 decimal places and using the same seed.

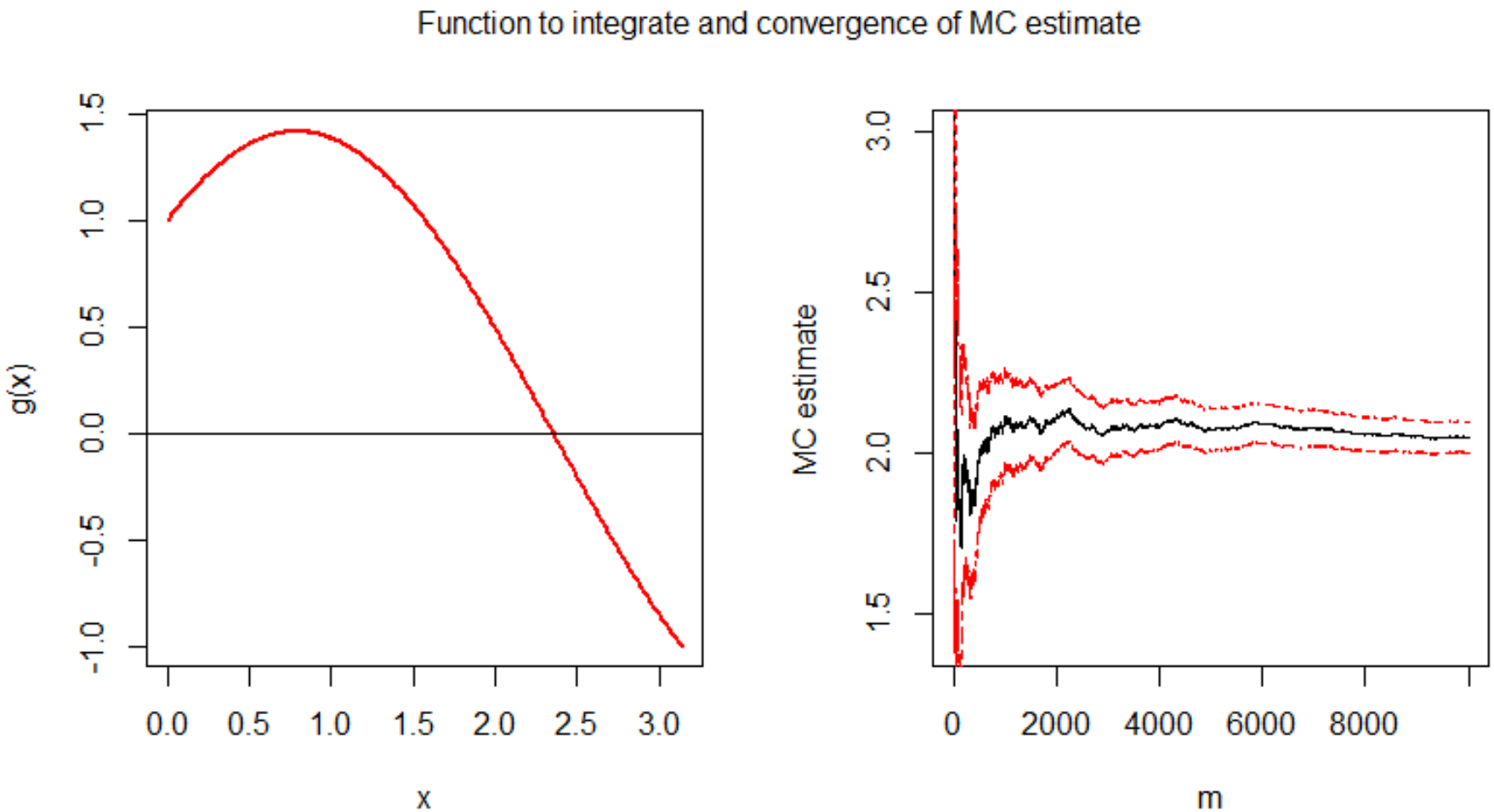
$$I \in [1.949382.145387]$$

$$I \in [1.91469, 2.10769]$$

MC integration with R

```r
# 1. Define the function g() that we wish to integrate
g = function(x) {sin(x) + cos(x)}

# 2. Perform the MC approximation using m = 10,000 simulations
set.seed(2023)

m = 10000
fx = runif(m, 0, pi)

# MC estimate of the integral
I_hat = ((pi - 0) / m) * sum(g(fx))
# [1] 2.047382

# approximation of the standard error of the MC estimate
se_I_hat = (pi / m) * sqrt(sum((g(fx) - I_hat)^2))
# [1] 0.05000365

# 95% Confidence Interval

c(I_hat - qnorm(1 - 0.05/2) * se_I_hat, I_hat + qnorm(1 - 0.05/2) * se_I_hat)
# [1] 1.949377 2.145387
```

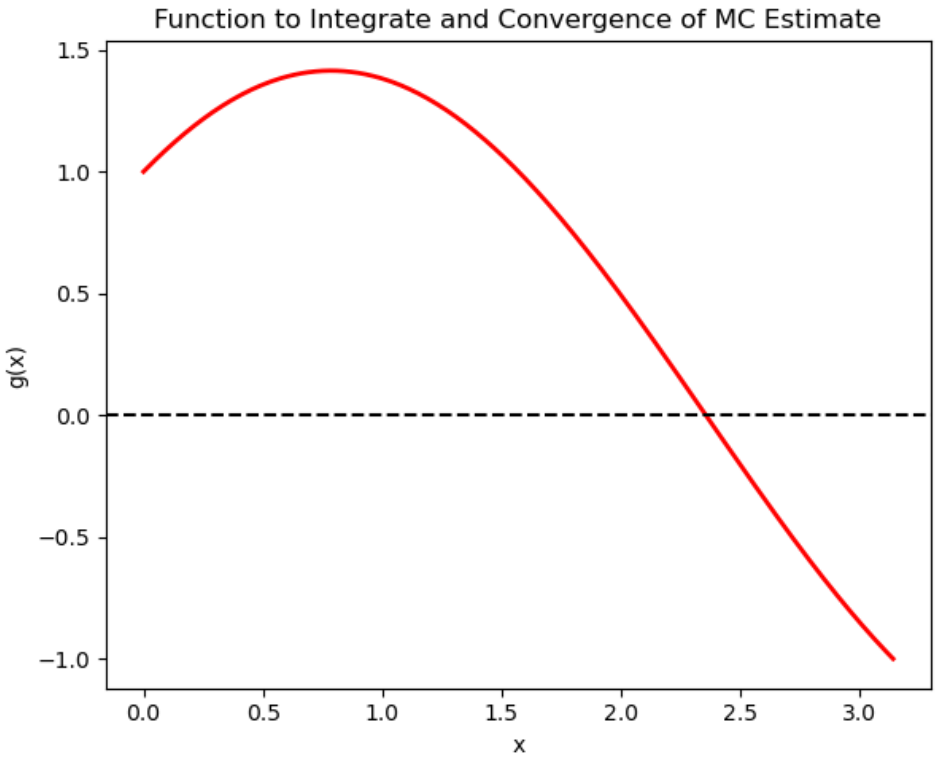
Plots in R

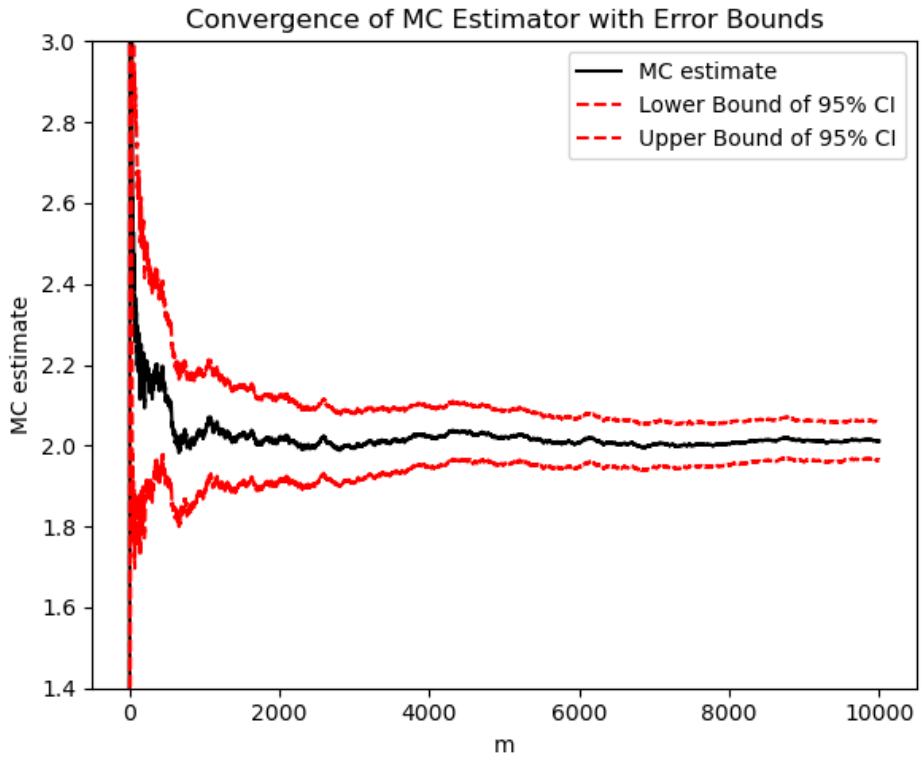

Function to integrate and convergence of MC estimate
g(x)
-1.0
-0.5
0.0
0.5
1.0
1.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x
MC estimate
1.5
2.0
2.5
3.0
0
2000
4000
6000
8000
m

MC integration with Python

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.stats import norm
4
5 # Define the function g(x) that we wish to integrate
6 def g(x):
7     return np.sin(x) + np.cos(x)
8
9 # Perform the MC approximation using m = 10,000 simulations
10 np.random.seed(2023)
11
12 m = 10000
13 fx = np.random.uniform(0, np.pi, m)
14
15 # MC estimate of the integral
16 I_hat = ((np.pi - 0) / m) * np.sum(g(fx))
17 print("MC estimate of the integral:", I_hat)
18 MC estimate of the integral: 2.0111916600488873
19
20 # Approximation of the standard error of the MC estimate
21 se_I_hat = (np.pi / m) * np.sqrt(np.sum((g(fx) - I_hat)**2))
22 print("Approximation of the standard error:", se_I_hat)
23 Approximation of the standard error: 0.049309740888497745
24
25 # 95% Confidence Interval
26 confidence_interval = norm.interval(0.95, loc=I_hat, scale=se_I_hat/np.sqrt(m))
27 print("Confidence Interval:", CI)
28 Confidence Interval: [1.9145463438204295, 2.107836976277345]
```

Plots in Python





MC simulations for empirical significance level

The power of a test is defined as the probability that we reject the null hypothesis when it is indeed false. Here we test the null hypothesis that the true mean is 0 using 100 Normal samples. The MC simulation shows that the empirical α-level of the nominal 5% t-test is between 1% and 3%.

```r
set.seed(2024)
testdata = function(data) {
  t.test(data)$p.value
}
X = matrix(rnorm(10*100), ncol=10)
pval = apply(X, 1, testdata)
mean(pval < 0.05)
# [1] 0.03
```

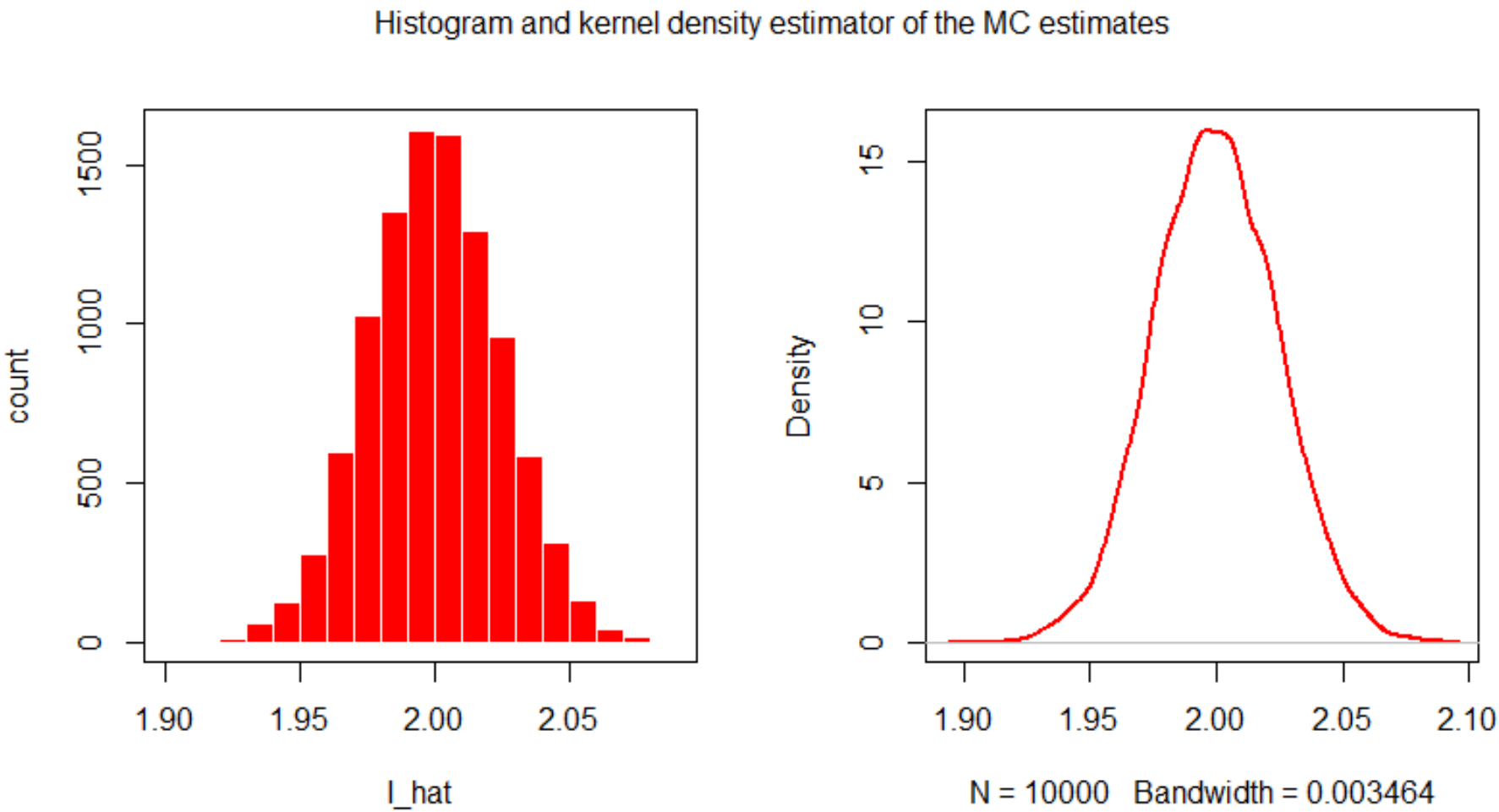
```python
import numpy as np
from scipy.stats import ttest_1samp
np.random.seed(2024)
def test_data(data):
    return ttest_1samp(data, popmean=0).pvalue
X = np.random.randn(10, 1000)
p_val = np.apply_along_axis(test_data, axis=1, arr=X)
print(np.mean(p_val < 0.05))
# 0.1
```

MC simulations with R

To tackle the same problem using what we call MC simulations, in our example that we could call 'MC simulations for MC integration', the idea is to introduce n, the number of simulations in addition to m, the number of replications or realizations. Obviously the process is slower but we do it to illustrate what could be an example of MC simulations.

```
# 1. MC simulations (using a for loop)
# Using MC simulation and MC integration

set.seed(12023)
n = 10000
m = 5000
est = numeric(n)

for(i in 1:n) {
  x = runif(m, 0, pi)
  g = sin(x) + cos(x)
  est[i] = (((pi - 0) / m) * sum(g))
}

I_hat = mean(est)
I_hat
# [1] 1.99988
```

Histogram and Density in R

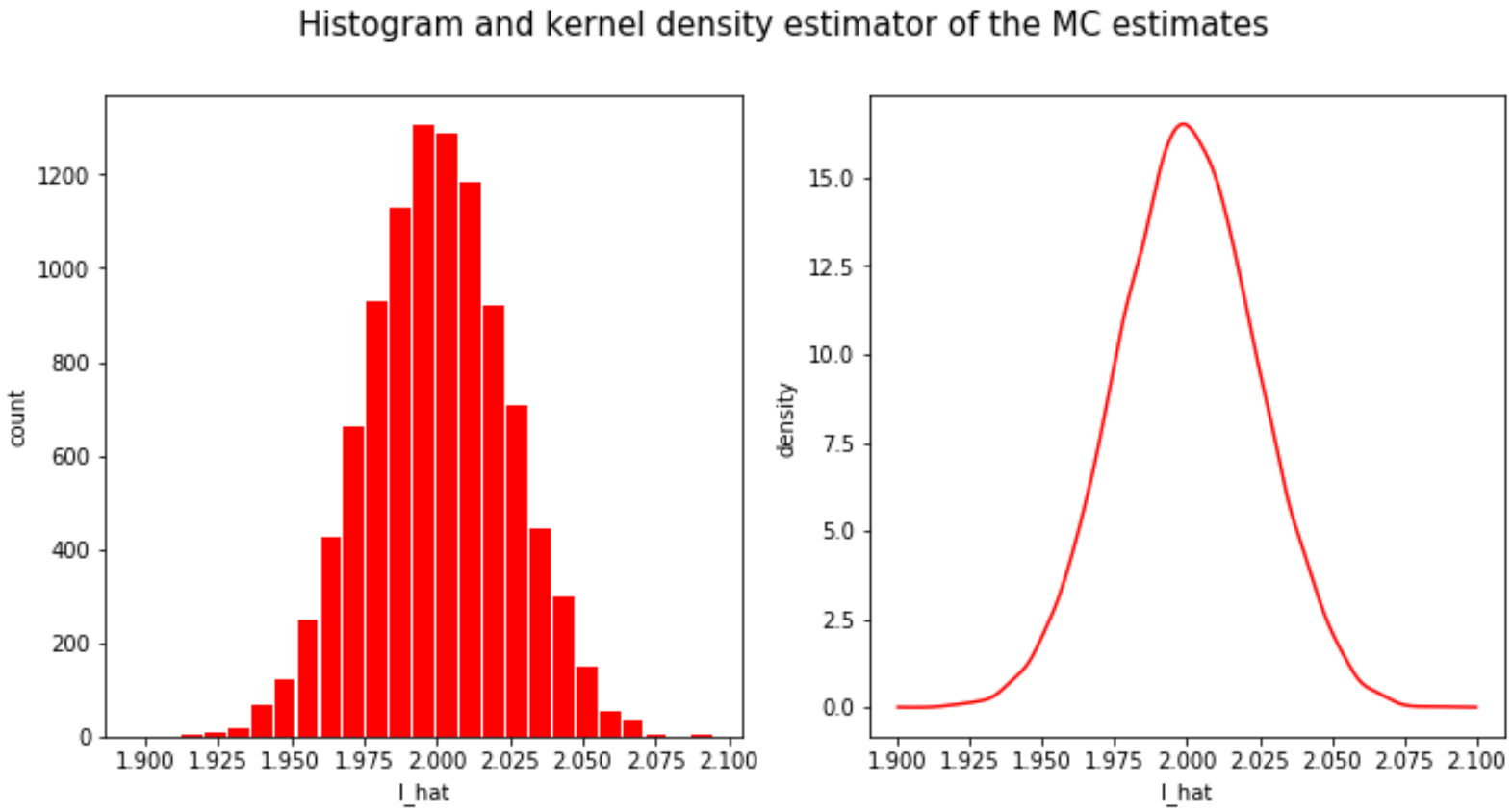


MC simulations with Python

We repeat the exact same procedure, this time in Python. Obviously the code can be different and optimized, the idea is to use a for loop.

```
# 1. MC simulations (simple for loop)
# Using MC simulation and MC integration

import statistics
np.random.seed(2023)

n = 10000
m = 5000
est = []
I_hat = []

for i in range(n):
    g = []
    x = np.random.uniform(low = 0, high = math.pi, size = m)
    g = (np.sin(x) + np.cos(x))
    # MC estimate of the integral
    est = (((math.pi - 0) / m) * np.sum(g))
    I_hat.append(est)

statistics.mean(I_hat)
# 2.0002932056229525
```

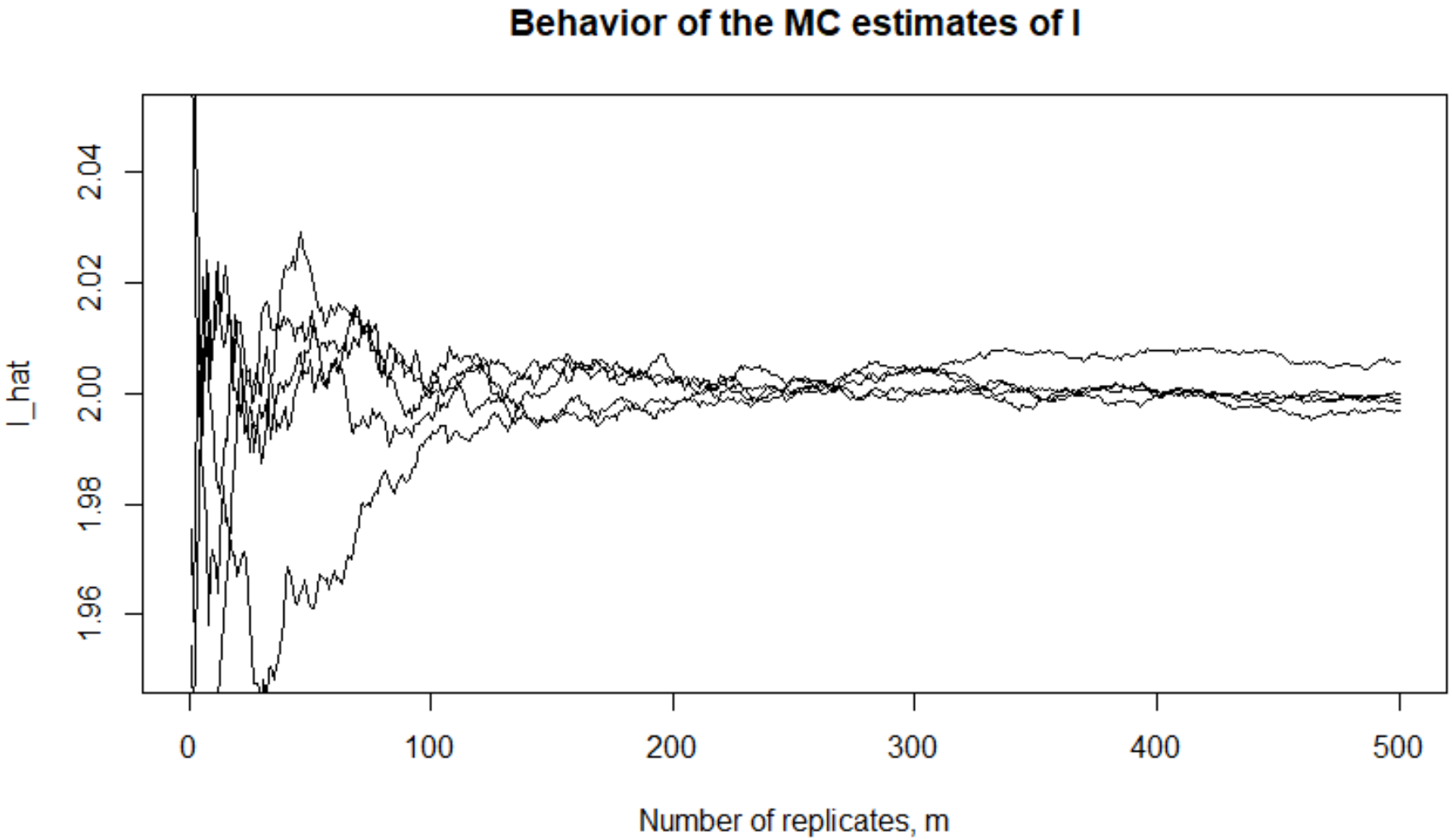
Histogram and Density in Python



MC simulations with R: advanced example for the same result

```r
# 3. Advanced example of MC simulations: nested for loop
# MC simulations (example taking about 3 min to run)
# for MC integration

# Define the function g() that we wish to integrate
g = function(x) {sin(x) + cos(x)}

# 4. Define storage objects and erform the MC approximation
# using m = 500 replications and n = 10,000 simulations.
set.seed(2023)

n = 10000
I_hat = numeric(n)
m = 500
est = matrix(m*n, nrow = n, ncol = m)

for(i in 1:n) {
  for(j in 1:m) {

    # Sum on all rows then we take the mean to get our final estimate
    est[i, j] = ((pi - 0) / m) * sum(g(runif(j, 0, pi))[j])
  }
  # MC estimate of the integral
  I_hat[i] = sum(est[i,])
}

mean(I_hat)
# [1] 2.000983
```

Behavior of the MC estimates in R



Conclusion from R and Python outputs

Setting: The integral to approximate is $I = \int_0^\pi g(x)dx$, with $g(x) = sin(x) + cos(x)$. Using $m = 10,000$ realizations and with $f(x) \sim U[0, \pi]$, the estimate $\hat{I}$ of I is about 2, which is the analytical answer. When computing 95%-Confidence Intervals, we get:

In R:

```r
# 95% Confidence Interval
c(I_hat - qnorm(1 - 0.05/2) * se_I_hat, I_hat + qnorm(1 - 0.05/2) * se_I_hat)
# [1] 1.949377 2.145387
```

In Python:

```python
# 95% Confidence Interval
[I_hat - scipy.stats.norm.ppf(1 - 0.05/2) * se_I_hat, I_hat + scipy.stats.norm.
    ppf(1 - 0.05/2) * se_I_hat]
# [1.9146915353267746, 2.107691784771]
```

For the same number of replications m and provided that the MC estimates $\hat{I}$ and $\hat{se}(\hat{I})$ are correctly computed, the results are obviously equivalent using both programming languages.

Further reading and code

Rizzo, M.L. (2019). Statistical Computing with R, Second Edition (2nd ed.). Chapman and Hall/CRC.
https://doi.org/10.1201/9780429192760

Kloke, J. and McKean, J.W. (2015), Nonparametric Statistical Methods using R, CRC Press
ISBN: 13: 978-1-4398-7344-1 (eBook - PDF)

The R Project for Statistical Computing:
https://www.r-project.org/

Python:
https://www.python.org/