Diagrammatic Monte Carlo

Exercises

Jonas Märtens

June 24, 2023

University of Bologna

Contents

Task 1: Monte Carlo Basics - Estimating PI

Estimating PI - Task

Two different methods to estimate π:

- Area Method (as seen in the picture)
- MC Integration Method
  $\pi/4 = \int_0^1 \sqrt{1-x^2}dx = \int_0^1 \frac{\sqrt{1-x^2}}{p(x)} p(x)dx = \langle \frac{\sqrt{1-x^2}}{p(x)} \rangle_p \approx \frac{1}{N} \sum_{i=1}^N \frac{\sqrt{1-x_i^2}}{p(x_i)}$ for $p(x) = 1$ and $x_i \sim U(0,1)$

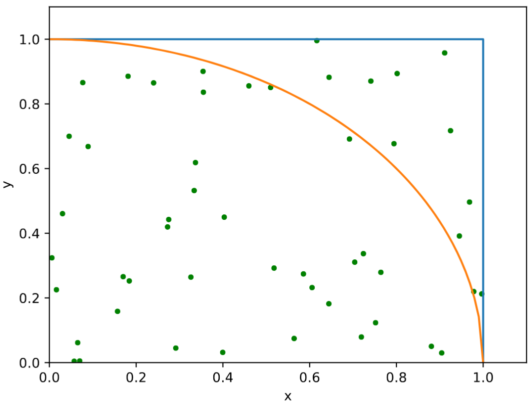


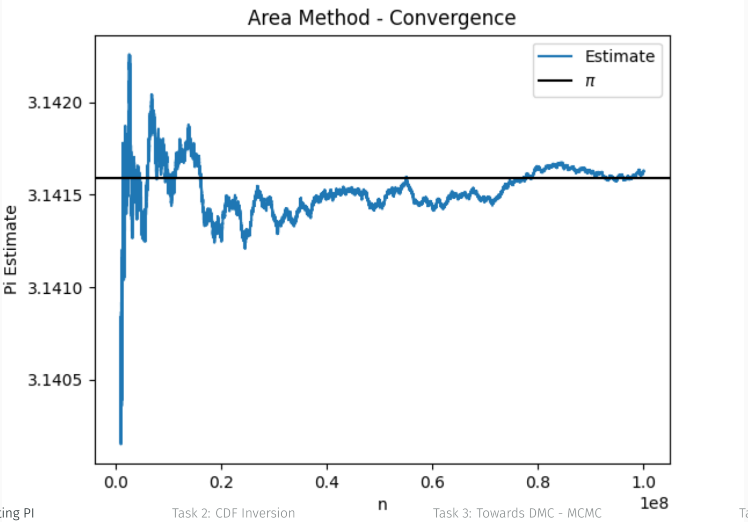
Figure 1: Visualization of the area method.[Ces22]

Estimating PI: Area Method - Implementation

```python
def area_method(n, plot=False):
    n_in = 0
    estimates_n = np.array([], dtype=int)
    estimates_area = np.array([])
    for i in range(n):
        # generate random numbers between 0 and 1
        x = np.random.random()
        y = np.random.random()
        # check if the point is in the circle
        if x**2 + y**2 <= 1:
            n_in += 1
        # every 10000 points, calculate the area
        if plot:
            if i % (int(n / 100000)) == 0 and i > 1000000:
                estimates_n = np.append(estimates_n, i + 1)
                estimates_area = np.append(estimates_area, 4 * n_in / (i + 1))
    if plot:
        # Do the plotting here
        # ...
    return 4 * n_in / n
```

Estimating PI: Area Method - Convergence

```
area_method(int(1e8), plot=True)
```

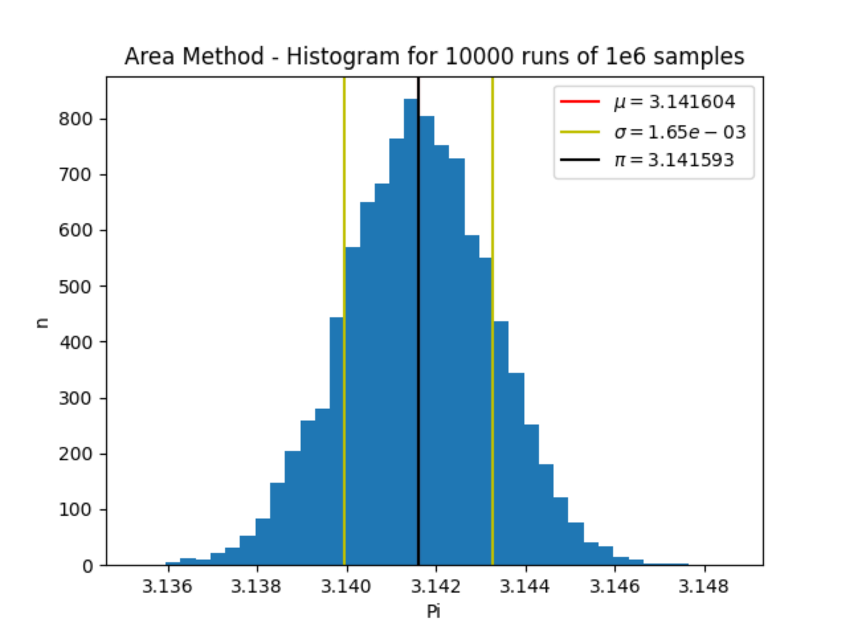


Estimating PI: Area Method - We can do better

```python
def area_method_fast(n):
    x = np.random.random(n)
    y = np.random.random(n)
    n_in = np.sum(x**2 + y**2 <= 1)
    return 4 * n_in / n
```

```python
pis = np.zeros(10000)
for i in range(10000):
    pis[i] = area_method_fast(int(1e6))
std = np.std(pis)
mean = np.mean(pis)
# Do the plotting here
# ...
print("Area Method:")
print(f"{mean = }")
print(f"{std = }")
```

Estimating PI: Area Method - Histogram

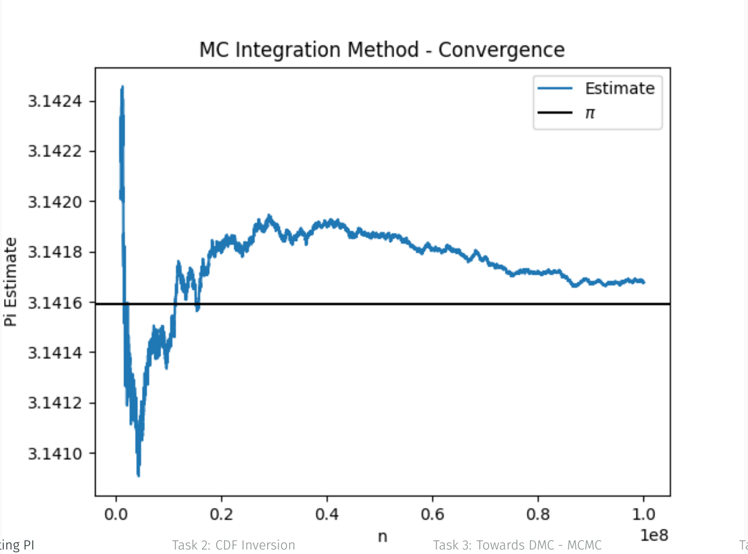


Estimating PI: MC Integration Method - Implementation

```python
def p(x):
    return 1

def f(x):
    return np.sqrt(1 - x**2)

def mc_integrate(n, plot=False):
    sum = 0
    estimates_n = np.array([], dtype=int)
    estimates_area = np.array([], dtype=np.float32)
    for i in range(n):
        x = np.random.random()
            sum += f(x) / p(x)
            if plot:
                if i % (int(n / 100000)) == 0 and i > 1000000:
                    estimates_n = np.append(estimates_n, i+1)
                    estimates_area = np.append(estimates_area, 4 * sum / (i+1))
    if plot:
        # Do the plotting here
        # ...
    return sum / n * 4
```

Estimating PI: MC Integration Method - Convergence

```
mc_integrate(int(1e8), plot=True)
```

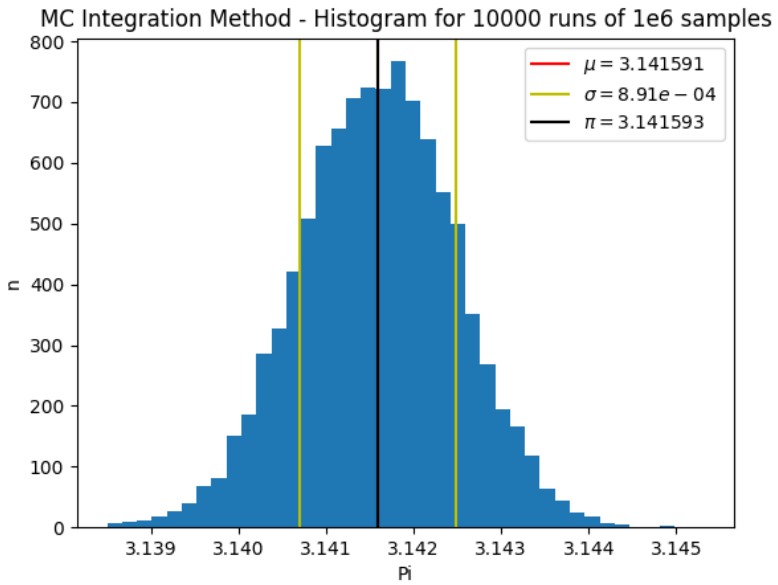


Estimating PI: MC Integration Method - We can do better

```python
def mc_integrate_fast(n):
    x = np.random.random(n)
    return np.sum(f(x)) / n * 4
```

```python
pis = np.zeros(10000)
for i in range(10000):
    pis[i] = mc_integrate_fast(int(1e6))
std = np.std(pis)
mean = np.mean(pis)
# Do the plotting here
# ...
print("MC Integration Method:")
print(f"{mean = }")
print(f"{std = }")
```

Estimating PI: Area Method - Histogram



Task 2: CDF Inversion

CDF Inversion - Task

Let $Q(\tau, \alpha) = \exp(-\alpha\tau)$. and $F(\tau) = \frac{1-\exp(-\alpha\tau)}{\alpha}$ the CDF. Sample from Q by inverting:

$$r = \frac{\int_{\tau_{\min}}^{\tau} Q\left(\tau'\right) d\tau'}{\int_{\tau_{\min}}^{\tau_{\max}} Q\left(\tau'\right) d\tau'} = \frac{F(\tau) - F\left(\tau_{\min}\right)}{F\left(\tau_{\max}\right) - F\left(\tau_{\min}\right)}$$

with $r \sim U(0,1)$. Setting $\tau_{\min} = 0$ and $\tau_{\max} = 5$ and solving:

$$\tau = -\frac{\log(1 - \alpha \cdot r \cdot F(\tau_{\max}))}{\alpha}$$

We can now

- Sample from Q and plot the histogram
- Calculate $I_1 = \int_0^5 \tau Q(\tau, \alpha) d\tau = \langle \tau \rangle \cdot \text{Norm} = \langle \tau \rangle \cdot F(\tau_{\max})$
- Calculate $I_2 = \int_0^5 \tau^2 Q(\tau, \alpha) d\tau = \langle \tau^2 \rangle \cdot F(\tau_{\max})$
- Calculate $\sigma(I_1)$ and $\sigma(I_2)$: $\sigma(\overline{\tau}) = \frac{\sigma_{\text{Samples}}}{\sqrt{n}}$ [GKW16, p.47]

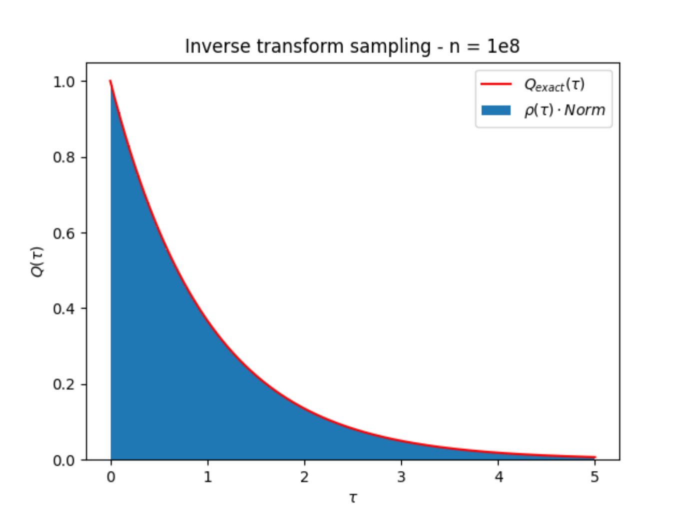
CDF Inversion - Implementation

```python
alpha = 1
x_max = 5
n = int(1e8)

def F(x):
    return (1 - np.exp(-alpha*x))/alpha

F_max = F(x_max)

def F_inv(r):
    return -np.log(1 - alpha*r*F_max)/alpha

#generate all samples at once using numpy
r = np.random.random(n)
x = F_inv(r)
# Do the plotting here
# Use np.histogram or plt.hist
print("I1 exact: 1-6*exp(-5) = 0.9595723")
print(f"I1 mean: { np.mean(x)*F_max:.6f}")
print(f"I1 std: {np.std(x)*F_max/np.sqrt(n):.3e}")
print("I2 exact: 2-37*exp(-5) = 1.750696")
print(f"I2 mean: {np.mean(x**2)*F_max:.6f}")
print(f"I2 std: {np.std(x**2)*F_max/np.sqrt(n):.3e}")
```

CDF Inversion - Histogram

Console Output:

```
I1 exact: 0.9595723
I1 mean:  0.959501
I1 std:   9.045e-05
I2 exact: 1.750696
I2 mean:  1.750502
I2 std:   3.205e-04
```



Task 3: Towards DMC - MCMC

Markov Chain Monte Carlo - Task

Basic principle behind DMC is Markov Chain Monte Carlo (MCMC). We again sample from a distribution f, this time using the Metropolis-Hastings algorithm:

- Propose a random configuration x' from a proposal distribution p
- Accept the configuration with probability $A(x, x') = \min\left(1, \frac{f(x')p(x)}{f(x)p(x')}\right)$
- p can be optimized to increase acceptance rate (similarly to CDF inversion, see also [FSW08, Chapter 12.3.1.1])
- This Markov Chain converges to desired distribution f, guarantees ergodicity
- Samples now correlated $\rightarrow$ Blocking Analysis to estimate σ

Markov Chain Monte Carlo - Implementation

```
alpha = 1
n_equ = int(1e4)
n_sam = int(1e6)
n_acc = 0
n_rej = 0
rng = np.random.default_rng()

#function to sample from
def f(x):
    return np.exp(-alpha*x)

#uniform distribution
uni = rng.uniform

#acceptance ratio for Metropolis-Hastings
def acc_ratio(x, x_new):
    return f(x_new)/f(x)

#initial sample
curr = uni(0,5)

#array to store samples
samples = np.zeros(n_sam)
```

Markov Chain Monte Carlo - Implementation

```python
#equilibration
for i in range(n_equ):
    prop = uni(0,5)
    if acc_ratio(curr, prop) > np.random.random():
        curr = prop
        n_acc += 1
    else:
        n_rej += 1

#sampling
for i in range(n_sam):
    prop = uni(0,5)
    if acc_ratio(curr, prop) > np.random.random():
        curr = prop
        n_acc += 1
    else:
        n_rej += 1
    samples[i] = curr

# Plotting, printing, etc.
# ...
```

Markov Chain Monte Carlo - Blocking Analysis

```python
def blocking(self, samples, min_blocks: int = 32) -> tuple[float, float]:
    means = np.copy(samples) # Copy the samples to avoid changing the original array
    mean = np.mean(means).astype(float) # Calculate the mean of the samples
    n = np.log2(len(means) // min_blocks).astype(int) # Calculate the iteration steps
    block_sizes = np.logspace(0, n, n + 1, base=2) # Minumum block size is 1, max is 2**n
    vars = np.zeros(n + 1, dtype=float) # Initialize the array for the variances
    vars[0] = 1 / (len(means) - 1) * (np.mean(means**2) - mean**2) # Naive variance
    Rx = np.zeros(n + 1) # Initialize Rx array
    Rx[0] = 1
    for i in range(1, n + 1): # Perform blocking
        if means.size % 2 != 0: # Make sure the number of blocks is even in order to divide by 2
            means = means[:-1]
        means = 0.5 * (means[::2] + means[1::2]) # Double the block size
        n_blocks = means.size
        varXBlock = n_blocks / (n_blocks - 1) * (np.mean(means**2) - mean**2)
        vars[i] = varXBlock / n_blocks # Calculate the variance
        Rx[i] = block_sizes[i] * varXBlock / vars[0] / samples.size
    plateau_index = np.argmin(np.abs(Rx[1:] - Rx[:-1])[3:]) + 4 # Find plateau in Rx
    # Plotting...
    return mean, vars[plateau_index]
```

For the maths behind the blocking analysis, see [GKW16, Chapter 3.4].

The results are very similar to the CDF inversion method, as we sample from the same distribution in a different way.

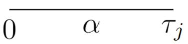
We will take a closer look at the blocking analysis after the next task.

Task 4&5: Diagrammatic Monte Carlo

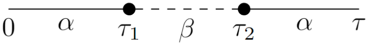
We are interested in the following function:

$$Q(\tau, \alpha, V) = \exp(-\alpha\tau) + \sum_{\beta} \int_0^{\tau} d\tau_2 \int_0^{\tau_2} d\tau_1 e^{-\alpha\tau_1} V e^{-\beta(\tau_2-\tau_1)} V e^{-\alpha(\tau-\tau_2)}$$

which can be represented as first and second order Feynman Diagrams of the following form:



$$= D_0^{\xi_0}(\tau_j, \alpha)$$



$$= D_2^{\xi_2}(\tau, \alpha, V; \tau_1, \tau_2, \beta) = e^{-\alpha\tau_1} V e^{-\beta(\tau_2-\tau_1)} V e^{-\alpha(\tau-\tau_2)}$$

How can we sample from this function?

We use the Metropolis-Hastings algorithm again, using the diagrams as weights. For ergodicity, we need to implement the following updates:

- Change of τ
- Increasing the order of the diagram
- Decreasing the order of the diagram
- (Change of α)

We will now look at a class that implements these updates, the sampling, the blocking analysis and the plotting.

[Ces22] Cesare Franchini.
Basic diagrammatic monte carlo - exercise sheet.
https://virtuale.unibo.it/pluginfile.php/1542120/mod_resource/content/1/project_basic_dmc.pdf, 2022.
[Online, private; accessed June 24, 2023].

[FSW08] H. Fehske, R. Schneider, and A. Weiße, editors.
Computational Many-Particle Physics.
Springer Berlin Heidelberg, 2008.

[GKW16] James Gubernatis, Naoki Kawashima, and Philipp Werner.
Quantum Monte Carlo Methods: Algorithms for Lattice Models.
Cambridge University Press, 2016.