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 $x_i \sim U(0,1)$

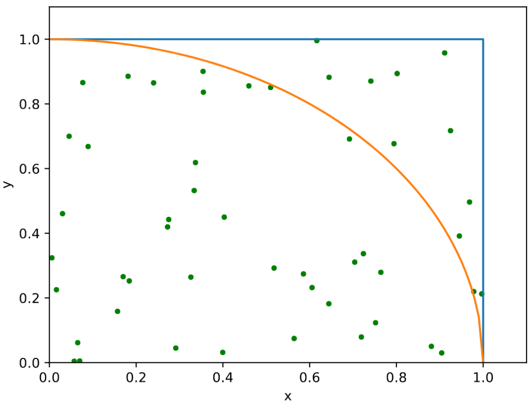


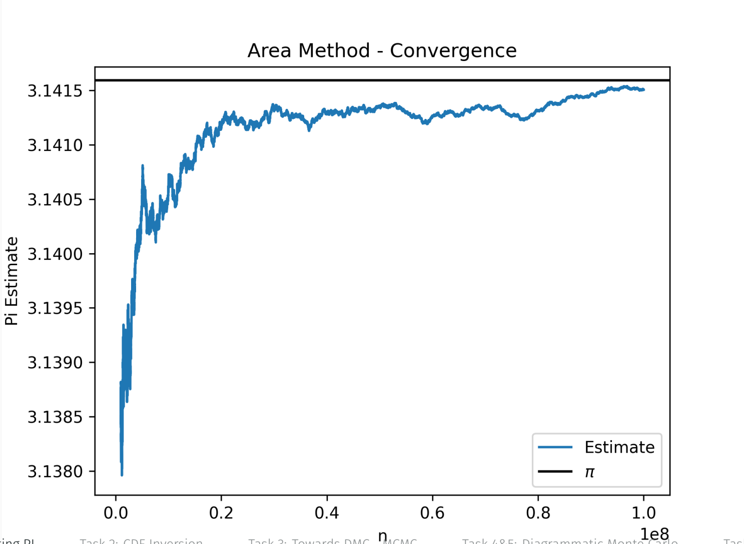
Figure 1: Visualization of the area method.[Ces22]

Estimating PI: Area Method - Implementation

```
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

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

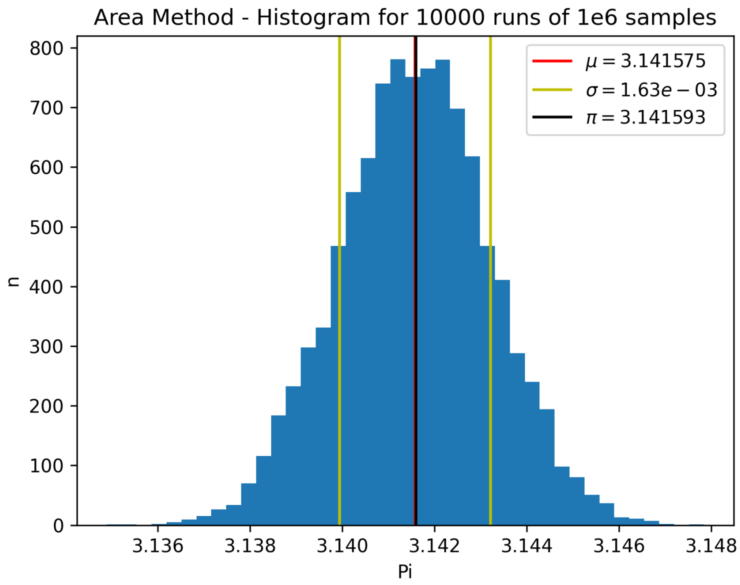


Estimating PI: Area Method - We can do better

```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

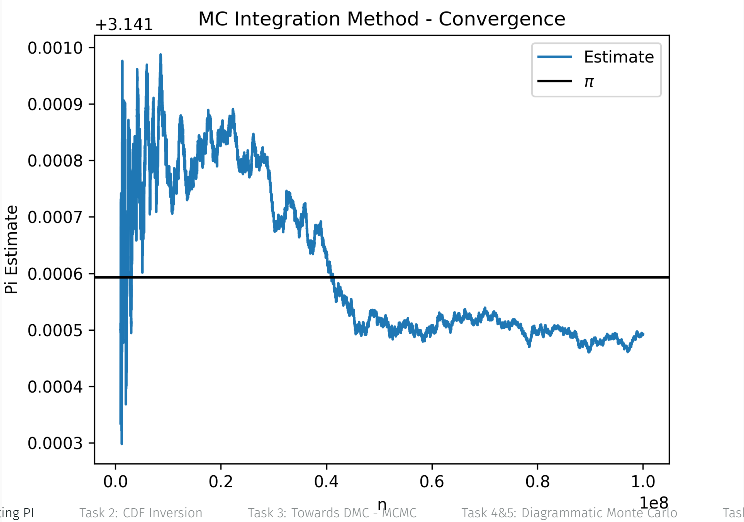


Estimating PI: MC Integration Method - Implementation

```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)
```

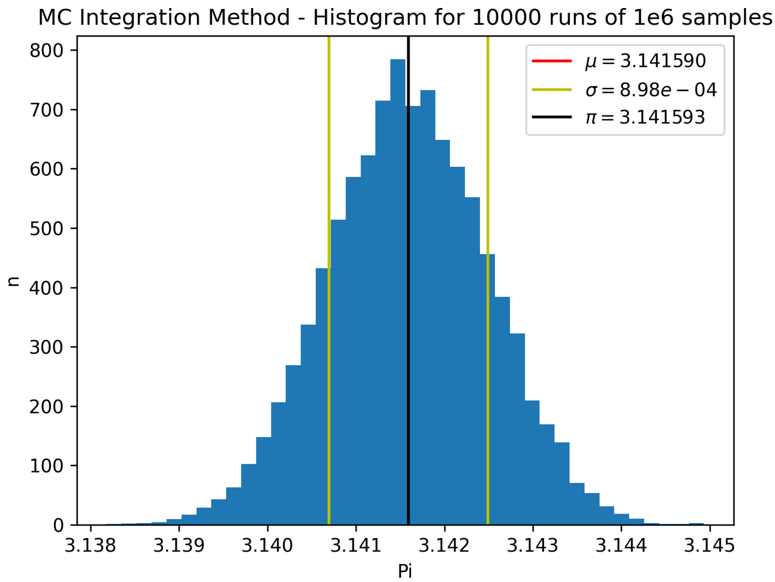


Estimating PI: MC Integration Method - We can do better

```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:

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

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

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

We can now

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

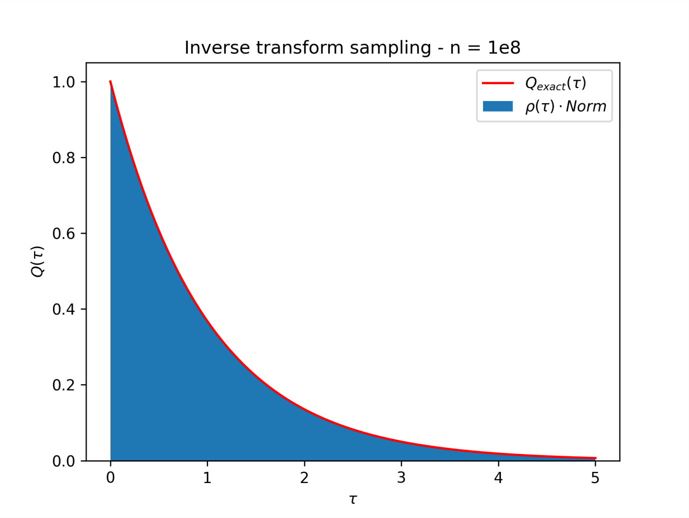
CDF Inversion - Implementation

```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

```python
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.

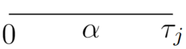
Task 4&5: Diagrammatic Monte Carlo

Diagrammatic Monte Carlo - Task

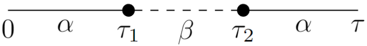
We are interested in the following function:

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

which can be represented as zero 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?

Diagrammatic Monte Carlo - Task

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)

Mind the context factors of the updates when changing the order of the diagram[FSW08, ch. 12.3.1.2]:

$$A_{0\to 2} = \min\left\{1, \frac{p_{\text{rem}}}{p_{\text{add}}} \frac{D_2^{\xi_2}\left(\tau, \alpha, V; \tau_1, \tau_2, \beta\right) d\tau_1 d\tau_2}{D_0^{\xi_0}\left(\tau_0, \alpha_0\right) p_1\left(\tau_1\right) p_2\left(\tau_2\right) p_3(\beta) d\tau_1 d\tau_2}\right\}$$

We implement a Python class to sample, analyse and plot the results of the DMC algorithm. We will first take a look at some of the methods of this class, then have a look at the whole class **DMC.py** and finally look at the results.

Diagrammatic Monte Carlo Implementation - Initialization

```python
def __init__(...) -> None:
  # check parameters
  if tau_max <= tau_min:
      raise ValueError("tau_max must be greater than tau_min")
  # And so on...
  self.rng = np.random.default_rng() # set up random number generator
  self.uni = lambda a, b: self.rng.random() * (b - a) + a
  self.n_equ, n_sam, tau_min; tau_max, V, use_change_alpha = ... # Set up config parameters
  self.tau, alpha, beta, tau1, tau2 = ... # Set up state variables
  self.n_tau_acc, n_tau_rej, n_alpha_acc, n_alpha_rej... = 0 # Counters
  self.samples = np.zeros((n_sam, 2) if use_change_alpha else n_sam) # Set up sample array
  # Set up updater methods
  self.updaters = [self.change_tau]
  if use_change_alpha:
      self.updaters.append(self.change_alpha)
  # And so on...

```

Diagrammatic Monte Carlo Implementation - Weight Calculation

```python
def weight(self, tau=None, alpha=None, beta=None, tau1=None, tau2=None) -> float:
    if tau is None:
        tau = self.tau
    # Same for alpha, beta, tau1 and tau2
    # ...
    # If the diagram is of zero order, the weight is simple
    if beta == 0:
        return np.exp(-alpha * tau)
    # If the diagram is of second order, we need to include
    # two interaction vertices and the beta propagator
    return (
        np.exp(-alpha * tau1)
        * self.V
        * np.exp(-beta * (tau2 - tau1))
        * self.V
        * np.exp(-alpha * (tau - tau2))
    )
```

As we will see, this method is used in every update to calculate the acceptance ratio.

Diagrammatic Monte Carlo Implementation - Change-τ&α

```python
def change_tau(self) -> bool:
    prop_tau = self.uni(self.tau2, self.tau_max)
    M = self.weight(tau=prop_tau) / self.weight()
    if M > self.rng.random():
        self.tau = prop_tau
        self.n_tau_acc += 1
        return True
    else:
        self.n_tau_rej += 1
        return False

def change_alpha(self) -> bool:
    prop_alpha = [0.5, 1][int(self.rng.random() * 2)]
    M = self.weight(alpha=prop_alpha) / self.weight()
    if M > self.rng.random():
        self.alpha = prop_alpha
        self.n_alpha_acc += 1
        return True
    else:
        self.n_alpha_rej += 1
        return False
```

Diagrammatic Monte Carlo Implementation - Change-Order

```python
def add_beta(self) -> bool:
    if self.beta != 0:
        self.n_beta_rej += 1
        return False
    prop_tau2 = self.uni(self.tau_min, self.tau)
    prop_tau1 = self.uni(self.tau_min, prop_tau2)
    prop_beta = [0.25, 0.75][int(self.rng.random() * 2)]
    M = self.weight(beta=prop_beta, tau1=prop_tau1, tau2=prop_tau2) / (
        self.weight() / (self.tau - self.tau_min) / (prop_tau2 - self.tau_min) * 0.5
    )
    if M > self.rng.random():
        self.beta = prop_beta
        self.tau1 = prop_tau1
        self.tau2 = prop_tau2
        self.n_beta_acc += 1
        return True
    else:
        self.n_beta_rej += 1
        return False
```

The remove-β update works the same way. The Metropolis ratio is the inverse of the add-β ratio.

Diagrammatic Monte Carlo Implementation - Updating, Sampling

```python
def update(self) -> bool:
    updater = self.updaters[int(self.rng.random() * len(self.updaters))]
    return updater()

def sample(self) -> None:
    for i in trange(self.n_sam):
        self.update()
        if self.use_change_alpha:
            self.samples[i, 0] = self.tau
            self.samples[i, 1] = self.alpha
            if self.beta == 0:
                self.n_zero_order[self.alpha] = self.n_zero_order.get(self.alpha, 0) + 1
        else:
            self.samples[i] = self.tau
            if self.beta == 0:
                self.n_zero_order += 1
    # Logging ...
```

Equilibration works the same way, but without sampling.

Diagrammatic Monte Carlo Implementation - Normalization

The normalization constant could still be calculated analytically, but normally this is not possible.
Solution: Use normalization constant C_0 of zero order diagram and number of zero order diagrams N_0 to estimate C [RFT20, ch. 3.3]:

$$C \approx C_0 \frac{N}{N_0}$$

This is implemented as an alternative to the analytical normalization. Furthermore, helper methods for plotting and analysis are implemented.

Let's take a look at the whole class **DMC.py**, before we come back to the results.

Switching to VS Code

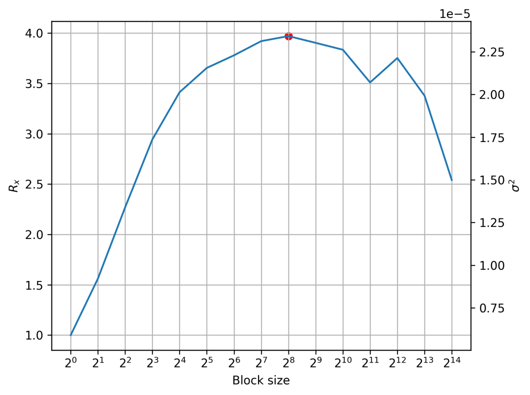
Diagrammatic Monte Carlo - Results

```python
from DMC import DMC
dmc = DMC(
    n_equ=int(1e4),
    n_sam=int(5e6),
    tau_min=0,
    tau_max=5,
    alpha_0=1,
    tau_0=1,
    use_change_alpha=True,
    use_change_beta=False,
    use_analytical=True,
)
dmc.equilibrate()
dmc.sample()
dmc.print_I1(0.5)
dmc.print_I1(1)
dmc.print_I2(0.5)
dmc.print_I2(1)
dmc.plot_hist(1)
dmc.plot_hist(0.5)
```

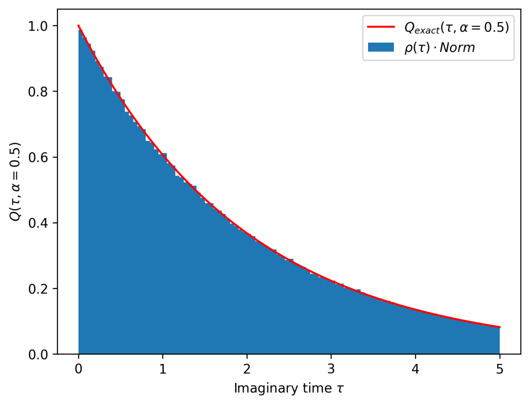
```
Warming up: 10000 steps
Warmup complete: τ acc. ratio of 0.54, α acc. ratio of 0.86
Sampling: 5000000 steps
100%|██████████| 5000000/5000000 [00:25<00:00, 194393.91it/s]
Sampling complete: τ acc. ratio of 0.54, α acc. ratio of 0.85
I1(α=0.5) mean: 2.85430, var: 9.53e-06
I1(α=1) mean: 0.96035, var: 2.40e-06
I2(α=0.5) mean: 7.31758, var: 1.63e-04
I2(α=1) mean: 1.75439, var: 2.34e-05
```

Plots are shown on the next slide.

Diagrammatic Monte Carlo - Results



One of the blocking analysis plots

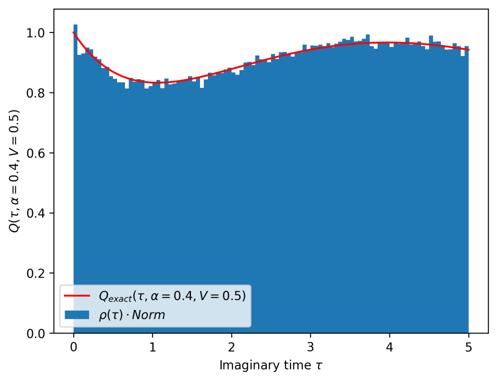


Histogram for $\alpha = 0.5$

Diagrammatic Monte Carlo - Results

```python
from DMC import DMC

dmc = DMC(
    n_equ=int(1e4),
    n_sam=int(3e6),
    tau_min=0,
    tau_max=5,
    alpha_0=0.4,
    tau_0=1,
    V=0.5,
    use_change_alpha=False,
    use_change_beta=True,
    use_analytical=True,
)
dmc.equilibrate()
dmc.sample()
dmc.plot_hist()
```

```
Warming up: 10000 steps
Warmup complete: τ acc. ratio of 0.76, β acc. ratio of 0.22
Sampling: 3000000 steps
100%|██████████| 3000000/3000000 [00:14<00:00, 205720.34it/s]
Sampling complete: τ acc. ratio of 0.76, β acc. ratio of 0.22
```



Task 6: DMC - Green Function Estimator

Diagrammatic Monte Carlo - Greens Function Estimator

An exact MC estimator for the Greens function can be derived [MPSS00, ch. III.C]:

$$\langle q_{\tau_0} \rangle_{MC} = C * Q\left(\tau_0, \alpha, V\right)$$

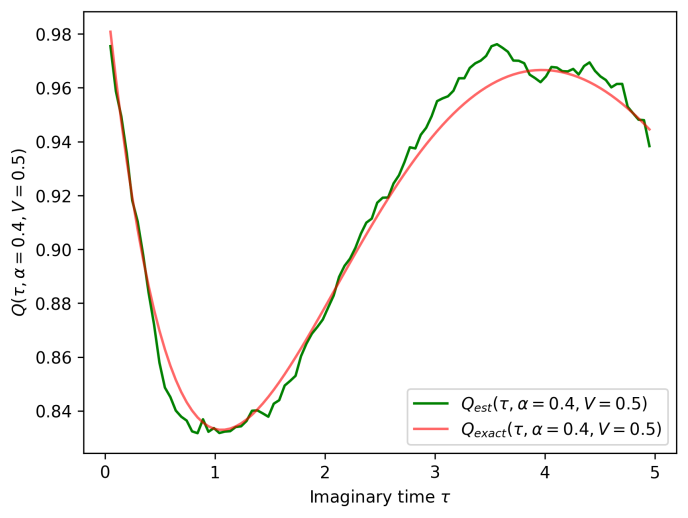
$$q_{\tau_0}(\nu, \tau) = \begin{cases} q(\nu)\mathcal{D}_\nu\left(\tau_0\right)/\mathcal{D}_\nu(\tau) = |\Gamma_0|^{-1} e^{-\alpha(\tau_0 - \tau)}, & \text{if } \tau \in \Gamma_0 \text{ and } \mathcal{D}_\nu(\tau) \neq 0 \\ 0, & \text{otherwise.} \end{cases}$$

Where i choose $\Gamma_0 = [\tau_0 - a, \tau_0 + a]$ with $a = \min(0.2, \tau_0, \tau_{\max} - \tau_0)$
This eliminates the discretization error of the histogram method.

```python
def green_estimator(self, tau0: float, alpha: float | None = None) -> float:
    if alpha is None:
        alpha = self.alpha
    tau = self.get_samples(alpha)
    diff = np.subtract.outer(tau0, tau)
    tau0_stacked = np.repeat(tau0, tau.size).reshape(diff.shape)
    mins = np.minimum(np.minimum(tau0_stacked, self.tau_max - tau0_stacked), 0.2)
    res = np.exp(-alpha * (diff)) / (2 * mins)
    res[np.abs(diff) > mins] = 0
    meaned = np.mean(res, axis=1) *self.norm(alpha, analytical=self.use_analytical)
    return meaned
```

Diagrammatic Monte Carlo - Greens Function Estimator - Results

Plot of the Greens function estimator for $n = 3 \cdot 10^6$ samples:



[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.

[MPSS00] A. Mishchenko, N. Prokof'ev, A. Sakamoto, and B. Svistunov.
Diagrammatic quantum monte carlo study of the fröhlich polaron.
Physical Review B, 62(10):6317–6336, September 2000.

[RFT20] S. Ragni, C. Franchini, and Hahn T.
Diagrammatic monte carlo study of the holstein polaron.
Master's thesis, Alma Mater Studiorum - University of Bologna, 2020.
[Online; accessed June 24, 2023].