ECE4271 - Evolutionary Processes, Algorithms, and Games

Monte Carlo Coding Project

Robin (Zihao) Lin (zl755)

April 7, 2022

1 SLLN

The sample code for the SLLN method for estimating $E_p(f)$ is ran for the following combinations of spike parameter and number of iterations: (0.2, 10000), (2, 10000), (11, 10000), (0.2, 50000), (2, 50000), (11, 50000), (0.2, 100000), (2, 100000), and (11, 100000). Each pair above is run ten times, and the average values for the estimation of $E_p(f)$ are recorded. Below is a summary of the results:

Configuration	Mean	Standard Deviation
(0.2, 10000)	99.18945321869438	0.28011693507514485
(2, 10000)	101.55449238007664	7.349285850650872
(11, 10000)	93.19985612192312	33.170083951174064
(0.2, 50000)	99.06226448100894	0.23812457389126926
(2, 50000)	100.06894082350163	4.352721105146157
(11, 50000)	96.973057650798	20.274873591652653
(0.2, 100000)	99.15940987494434	0.14425490318255824
(2, 100000)	99.54446585889069	2.3750553074405802
(11, 100000)	97.09652322709066	10.549470638196974

Table 1: Mean and Standard Deviation of Estimate of $E_p(f)$ at various configurations of spike and number of iterations.

As the number of iterations increased, the mean value for the estimate of $E_p(f)$ is getting closer to the actual value, which is 100. In addition, as the number of iterations increases for a given spike value, the standard deviation for the estimate of $E_p(f)$ seems to be decreasing, which is expected. However, for a given number of iterations, as the spike value is increased, the standard deviation of the estimate increases significantly. In addition, in the examples shown, the spike value of 11 seems to yield to worst performance for each and every number of iterations tested. Therefore, regarding overall performance, it looks like the configuration (2, 50000) is the best. For the code for this section, please see Appendix A.

2 Metropolis

2.1 Simple Proposal Process

In this section, the results for the simple proposal process of proposing uniformly among the nine nearest neighbors of a given state are shown. The code for this section is shown in Appendix B.

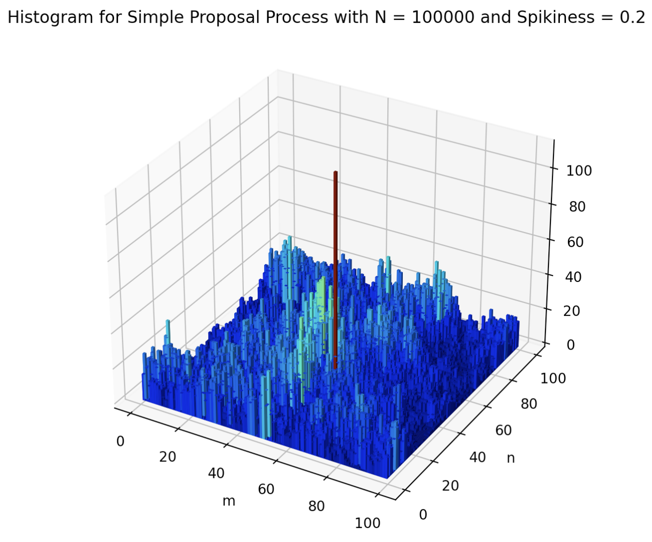
First, the performance at various combinations of spike and number of iterations are presented. The configurations (0.2, 10000), (2, 10000), (11, 10000), (0.2, 50000), (2, 50000), (11, 50000), (0.2, 100000), (2, 100000), and (11, 100000) are each ran for 10 times.

Configuration	Mean	Standard Deviation
(0.2, 10000)	101.35354464553545	11.56730363524549
(2, 10000)	100.05530446955305	10.19687113909237
(11, 10000)	100.0001299870013	0.9351818709066497
(0.2, 50000)	99.38366232675347	5.407594364850596
(2, 50000)	100.65882482350354	3.9775690222866102
(11, 50000)	99.92607947841043	0.14573084276282605
(0.2, 100000)	99.31057789422105	3.9177845629927304
(2, 100000)	99.46462035379646	1.975480073345437
(11, 100000)	100.03083969160309	0.10473358304018181

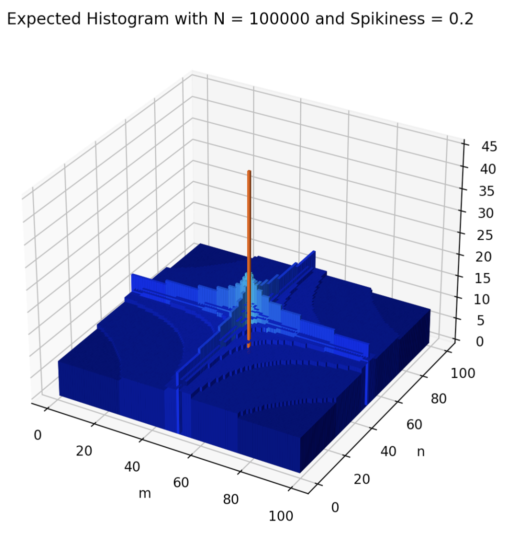
Table 2: Mean and Standard Deviation of Estimate of $E_p(f)$ at various configurations of spike and number of iterations for the Simple Proposal Process.

For a given number of iterations, as the spike value increases, the mean of the estimate of $E_p(f)$ gets closer to the expected value of $E_p(f)$. In addition, the standard deviation of the estimate of $E_p(f)$ decreases as the spike value is increased. On the other hand, the effect of the number of iterations on the mean of the estimate of $E_p(f)$ is more random. However, the standard deviation does decrease with an increased number of iterations.

Another interesting factor is the comparison between the P-histogram (histogram of generated points in P) and the expected histogram according to the distribution $p(m, n)$. Shown below are the comparisons between the experimental and theoretical

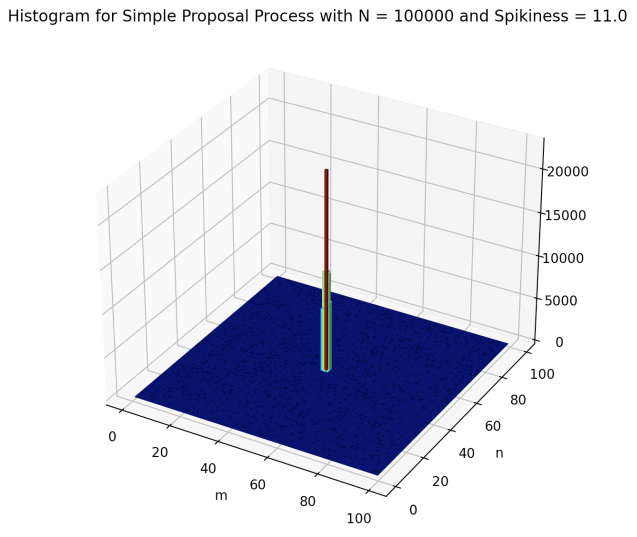


(a) Experimental Histogram for Spike = 0.2 and N = 100000

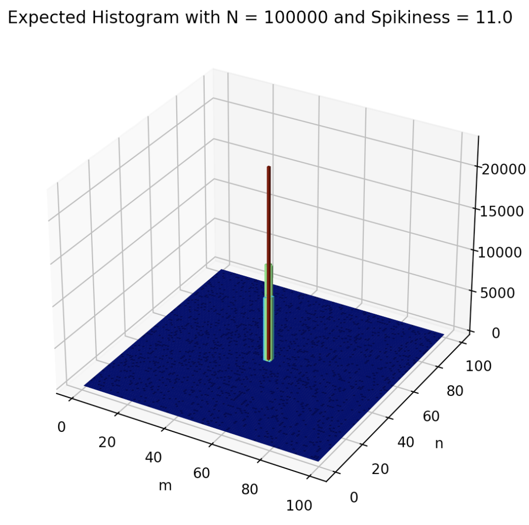


(b) Theoretical Histogram for Spike = 0.2 and N = 100000

Figure 1: Comparison between Experimental and Theoretical Histograms of Spike = 0.2 and N = 100000



(a) Experimental Histogram for Spike = 11 and N = 100000



(b) Theoretical Histogram for Spike = 11 and N = 100000

Figure 2: Comparison between Experimental and Theoretical Histograms of Spike = 11 and N = 100000

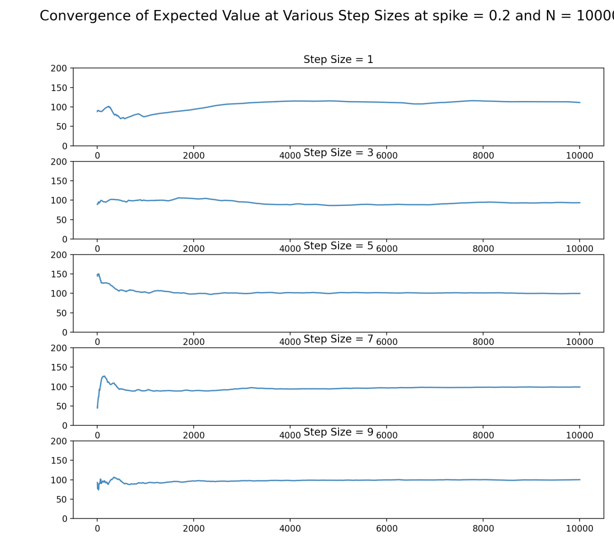
histograms for two different configurations of spike value and number of iterations: (0.2, 100000) and (11, 100000).

For the (0.2, 100000) configuration, the experimental P-histogram differs significantly from the theoretical histogram for $p(m, n)$. The point (50, 50) has the highest number of occurrences. However, in the experimental P-histogram, the overall distribution is highly random. It somewhat follows the general shape of the theoretical histogram but possesses a lot of noise. When the spike value is increased to 11, the experimental P-histogram matches very closely to the theoretical histogram. They are almost identical. This shows that the spike value affects the randomness in the Monte Carlo simulation.

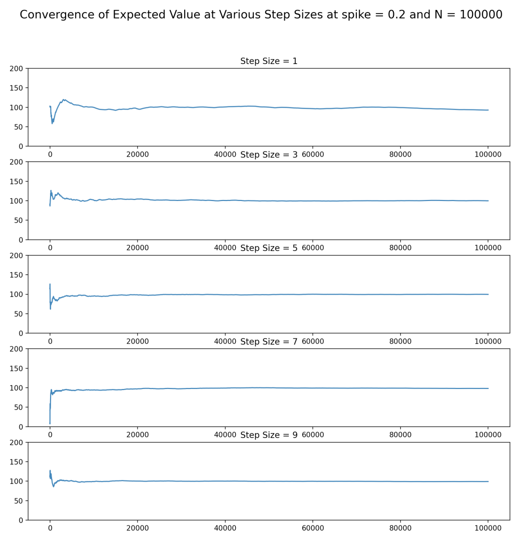
2.2 Complex Proposal Process

This section presents the results of a more complex proposal process. In the complex proposal process, instead of proposing uniformly among the nine nearest neighbors, it reaches out farther to all $(m + j, n + k)$ where $|j|$ and $|k|$ are less than or equal to k, where k is a user input to the proposal process. The code for this section is shown in Appendix C.

An interesting factor to look at the effect of step size on the convergence of the estimate of $E_p(f)$. A graph illustrating this behavior is shown below for the configurations (0.2, 10000), (0.2, 100000), (11, 10000), and (11, 100000).

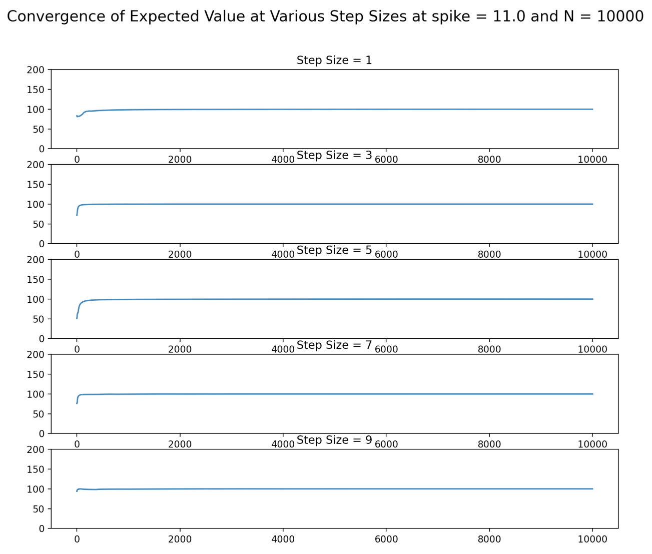


(a) Convergence Behavior for Spike = 0.2 and N = 10000

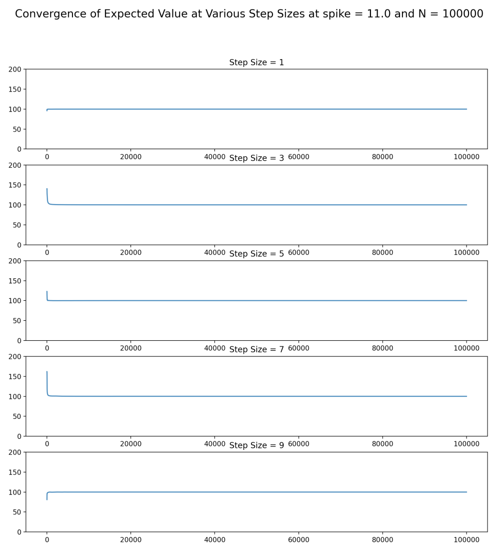


(b) Convergence Behavior for Spike = 0.2 and N = 100000

Figure 3: Convergence Behavior for Spike = 0.2 and N = 10000 and 100000



(a) Convergence Behavior for Spike = 11 and N = 10000



(b) Convergence Behavior for Spike = 11 and N = 100000

Figure 4: Convergence Behavior for Spike = 11 and N = 10000 and 100000

As shown in Figure 4, for a spike value of 0.2, the effect of the step size in the proposal process is quite significant. As the step size is increased from 1 to 9, the estimate of $E_p(f)$ converges much more quickly. The convergence also seems to be more stable. The effects of the step size are also noticeable for a spike value of 11, though not as significant. The spike value also plays an important role in leading to quicker and stable convergence, perhaps even more so than the step size. In all cases, it looks like the convergence occurs within the first 10000 iterations.

Appendix

A SSLN Script

```
import random
import statistics


def q(m, n, spike):
    return 1 / (1 + (abs(m - 50)) ** spike + (abs(n - 50)) ** spike)


def simulate(spike, N):
    normalization = 0
    for i in range(100):
        for j in range(100):
            normalization += q(i, j, spike)

    time_avg = 0
    for k in range(N):
        m = random.randint(0, 99)
        n = random.randint(0, 99)
        time_avg = (k * time_avg + 10000 * (m + n) * q(m, n, spike) / normalization) / (
            k + 1
        )

    return time_avg


def automated_experiment():
    combinations = [
        (0.2, 10000),
        (2, 10000),
        (11, 10000),
        (0.2, 50000),
        (2, 50000),
        (11, 50000),
        (0.2, 100000),
        (2, 100000),
        (11, 100000),
    ]

    for combination in combinations:
        spike = combination[0]
        N = combination[1]
        hist = []
        for _ in range(10):
            hist.append(simulate(spike, N))

        print(
            f"Average E_p(f) for spike = {spike} and N = {N} is {statistics.mean(hist)} with standard
                                                                 deviation {statistics.stdev(hist)}"
        )


def manual_experiment():
    spike = float(input("Enter spikiness value: "))
    N = int(input("Enter number of iterations: "))
    time_avg = simulate(spike, N)
    print(f"Computed mean for spike = {spike} and N = {N} is: " + str(time_avg))


def main():
    automated_experiment()


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

B Metropolis with Simple Proposal Script

```
import numpy as np
import matplotlib.pyplot as plt
```

```python
from matplotlib import cm
import random
import statistics


def q(m, n, spike):
    return 1 / (1 + (abs(m - 50)) ** spike + (abs(n - 50)) ** spike)


def f(m, n):
    return m + n


def propose_naive(curr):
    m = curr[0]
    n = curr[1]
    # Generate candidate neighbours
    candidates = []
    for i in [-1, 0, 1]:
        for j in [-1, 0, 1]:
            candidates.append(((m + i) % 100, (n + j) % 100))

    return random.choice(candidates)


def accept_reject(curr, proposed, spike):
    proposed_q = q(proposed[0], proposed[1], spike)
    curr_q = q(curr[0], curr[1], spike)

    if proposed_q >= curr_q:
        return proposed

    if random.random() <= proposed_q / curr_q:
        return proposed

    return curr


def compute_average(p):
    f_sum = 0
    for state in p:
        f_sum += f(state[0], state[1])
    return f_sum / len(p)


def plot_3d_histogram(points, title):
    m = np.array([i[0] for i in points])
    n = np.array([i[1] for i in points])
    hist, medges, nedges = np.histogram2d(
        m, n, bins=(100, 100), range=[[0, 99], [0, 99]]
    )
    mpos, npos = np.meshgrid(medges[:-1] + medges[1:], nedges[:-1] + nedges[1:])

    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")
    mpos = mpos.flatten() / 2.0
    npos = npos.flatten() / 2.0
    zpos = np.zeros_like(mpos)

    dm = medges[1] - medges[0]
    dn = nedges[1] - nedges[0]
    dz = hist.flatten()

    cmap = cm.get_cmap("jet")
    max_height = np.max(dz)
    min_height = np.min(dz)
    rgba = [cmap((k - min_height) / max_height) for k in dz]

    ax.bar3d(mpos, npos, zpos, dm, dn, dz, color=rgba, zsort="average")
    plt.title(title)
    plt.xlabel("m")
    plt.ylabel("n")
    plt.show()


def plot_expected_histogram(spike, N):
    normalization = 0
    for i in range(100):
        for j in range(100):
            normalization += q(i, j, spike)

    points = []
    for i in range(100):
        for j in range(100):
            points.append((i, j))

    p = []
    for point in points:
        mult = int(q(point[0], point[1], spike) / normalization * N)
        p += [point] * mult
    plot_3d_histogram(p, f"Expected Histogram with N = {N} and Spikiness = {spike}")


def histogram_analysis():
    spike = float(input("Enter spikiness value: "))
    N = int(input("Enter number of iterations: "))

    # Initialization
    p = [(random.randint(0, 99), random.randint(0, 99))]

    for iteration in range(N):
        p.append(accept_reject(p[-1], propose_naive(p[-1]), spike))

    print("Computed mean = " + str(compute_average(p)))
    plot_3d_histogram(
        p,
        f"Histogram for Simple Proposal Process with N = {N} and Spikiness = {spike}",
    )
    plot_expected_histogram(spike, N)


def statistical_analysis():
    combinations = [
        (0.2, 10000),
        (2, 10000),
        (11, 10000),
        (0.2, 50000),
        (2, 50000),
        (11, 50000),
        (0.2, 100000),
        (2, 100000),
        (11, 100000),
    ]

    for combination in combinations:
        spike = combination[0]
        N = combination[1]
        hist = []
        for _ in range(10):
            # Initialization
            p = [(random.randint(0, 99), random.randint(0, 99))]
            for iteration in range(N):
                p.append(accept_reject(p[-1], propose_naive(p[-1]), spike))

            hist.append(compute_average(p))

        print(
            f"Average E_p(f) for spike = {spike} and N = {N} is {statistics.mean(hist)} with standard
                                                              deviation {statistics.stdev(hist)}"
        )


def main():
    statistical_analysis()


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

C Metropolis with Complex Proposal Script

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import random


def q(m, n, spike):
    return 1 / (1 + (abs(m - 50)) ** spike + (abs(n - 50)) ** spike)


def f(m, n):
    return m + n


def propose_advanced(curr, step_size):
    m = curr[0]
    n = curr[1]

    # Generate candidate neighbours based on the step size
    candidates = []
    for i in range(-step_size, step_size + 1):
        for j in range(-step_size, step_size + 1):
            candidates.append(((m + i) % 100, (n + j) % 100))

    return random.choice(candidates)


def accept_reject(curr, proposed, spike):
    proposed_q = q(proposed[0], proposed[1], spike)
    curr_q = q(curr[0], curr[1], spike)

    if proposed_q >= curr_q:
        return proposed

    if random.random() <= proposed_q / curr_q:
        return proposed

    return curr


def compute_average(p):
    f_sum = 0
    for state in p:
        f_sum += f(state[0], state[1])
    return f_sum / len(p)


def plot_3d_histogram(points, title):
    m = np.array([i[0] for i in points])
    n = np.array([i[1] for i in points])
    hist, medges, nedges = np.histogram2d(
        m, n, bins=(100, 100), range=[[0, 99], [0, 99]]
    )
    mpos, npos = np.meshgrid(medges[:-1] + medges[1:], nedges[:-1] + nedges[1:])

    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")
    mpos = mpos.flatten() / 2.0
    npos = npos.flatten() / 2.0
    zpos = np.zeros_like(mpos)

    dm = medges[1] - medges[0]
    dn = nedges[1] - nedges[0]
    dz = hist.flatten()

    cmap = cm.get_cmap("jet")
    max_height = np.max(dz)
    min_height = np.min(dz)
    rgba = [cmap((k - min_height) / max_height) for k in dz]

    ax.bar3d(mpos, npos, zpos, dm, dn, dz, color=rgba, zsort="average")
    plt.title(title)
    plt.xlabel("m")
    plt.ylabel("n")
    plt.show()
```

```python
def plot_expected_histogram(spike, N):
    normalization = 0
    for i in range(100):
        for j in range(100):
            normalization += q(i, j, spike)

    points = []
    for i in range(100):
        for j in range(100):
            points.append((i, j))

    p = []
    for point in points:
        mult = int(q(point[0], point[1], spike) / normalization * N)
        p += [point] * mult
    plot_3d_histogram(p, f"Expected Histogram with N = {N} and Spikiness = {spike}")


def single_experiment():
    spike = float(input("Enter spikiness value: "))
    N = int(input("Enter number of iterations: "))
    step_size = int(input("Enter the step size for the proposal process: "))

    # Initialization
    p = [(random.randint(0, 99), random.randint(0, 99))]

    for iteration in range(N):
        p.append(accept_reject(p[-1], propose_advanced(p[-1], step_size), spike))

    print("Computed mean = " + str(compute_average(p)))
    plot_3d_histogram(
        p,
        f"Histogram for Simple Proposal Process with N = {N} and Spikiness = {spike}",
    )
    plot_expected_histogram(spike, N)


def plot_convergence(step_size, spike, N):
    for plot_id, step in enumerate(step_size):
        print(f"Plotting plot ID {plot_id}")
        points = step_size[step]
        averages = []
        total_sum = 0
        for i, point in enumerate(points):
            total_sum += f(point[0], point[1])
            averages.append(total_sum / (i + 1))
        iters = list(range(1, len(points) + 1))

        plt.subplot(len(step_size), 1, plot_id + 1)
        plt.plot(iters, averages)
        plt.title(f"Step Size = {step}", fontsize=12)
        plt.ylim(0, 200)
    plt.suptitle(
        f"Convergence of Expected Value at Various Step Sizes at spike = {spike} and N = {N}",
        fontsize=16,
    )
    plt.show()


def step_size_experiment():
    spike = float(input("Enter spikiness value: "))
    N = int(input("Enter number of iterations: "))
    step_size = dict.fromkeys(list(range(1, 10, 2)))

    for step in step_size:
        print(f"Processing step size of {step}")

        p = [(random.randint(0, 99), random.randint(0, 99))]

        for iteration in range(N):
            p.append(accept_reject(p[-1], propose_advanced(p[-1], step), spike))

        step_size[step] = p

        print(f"Computed mean for step_size = {step} is: " + str(compute_average(p)))

    plot_convergence(step_size, spike, N)
```

```python
def main():
    step_size_experiment()


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