SDS386D: Monte Carlo Methods in Statistics **Cesar N. Yahia**
Homework 2 Prof. S.G. Walker

1.

A chain with a transition matrix P and a stationary density π is reversible if $\pi_i p_{ij} = \pi_j p_{ji}$. Show that this conditions is satisfied for given p_{ij} and α_{ij}.

Case 1:
For $i \neq j$
Consider that $\alpha_{ij} = 1$ such that $1 < \frac{\pi_j q_{ji}}{\pi_i q_{ij}}$
This implies that $\frac{\pi_i q_{ij}}{\pi_j q_{ji}} < 1$ such that $\alpha_{ji} = \frac{\pi_i q_{ij}}{\pi_j q_{ji}}$
Substituting in the equation for p with $i \neq j$, $p_{ij} = q_{ij}$. Therefore, $\pi_i p_{ij} = \pi_i(q_{ij}) \star$
Similarly, $p_{ji} = \alpha_{ji} q_{ji} = \frac{\pi_i q_{ij}}{\pi_j q_{ji}} q_{ji} = \frac{\pi_i q_{ij}}{\pi_j}$. Therefore, $\pi_j p_{ji} = \pi_i q_{ij} \star\star$
From $\star$ and $\star\star$ we have that $\pi_j p_{ji} = \pi_i p_{ij}$
The same argument follows if $\alpha_{ji} = 1$ and $\alpha_{ij} = \frac{\pi_j q_{ji}}{\pi_i q_{ij}}$

Case 2:
For $i = j$
$\alpha_{ij} = \alpha_{ji} = 1$
$p_{ii} = q_{ii} + (1 - r_i)$
$\pi_i p_{ii} = \pi_i(q_{ii} + (1 - r_i))$
$p_{jj} = q_{jj} + (1 - r_j)$
$\pi_j p_{jj} = \pi_j(q_{jj} + (1 - r_j))$
$r_i = r_j$, $q_{ii} = q_{ji}$, and $\pi_i = \pi_j$
So, $\pi_j p_{jj} = \pi_i p_{ii}$ should be trivially satisfied.

2.

Suppose that the joint density $f(x,q) = q^x(1-q)^{1-x}$, $x \in \{0,1\}$ and $0 < q < 1$
Consider the transition density $X_n, X_{n+1} \in \{0,1\}$
$p(X_{n+1}|X_n) = 2\int_0^1 q^{X_n+X_{n+1}}(1-q)^{2-X_n-X_{n+1}}dq \star$
Find the stationary density.
To find the stationary, we have

$p(X_{n+1}|X_n) = \int f(X_{n+1}|q)f(q|X_n)dq$
$f(X_{n+1}|q) = \frac{f(X_{n+1},q)}{f(q)} \; \alpha \; f(X_{n+1},q)$
$f(q|X_n) = \frac{f(X_n,q)}{f(X_n)}$
Therefore,
$p(X_{n+1}|X_n) = \int \frac{f(X_{n+1},q)}{f(q)} \frac{f(X_n,q)}{f(X_n)} dq \star\star$
But,
Considering that q is uniformly distributed on (0,1), we have that $f(q) = 1$
So,
$f(X_{n+1}) = \sum p(X_{n+1}|X_n)f(X_n)$
$f(X_{n+1}) = \sum \int f(X_{n+1},q)\frac{f(X_n,q)}{f(X_n)}dqf(X_n)$
$f(X_{n+1}) = \sum \int f(X_{n+1},q)\frac{f(X_n,q)}{f(X_n)}f(X_n)dq$
$f(X_{n+1}) = \sum \int f(X_{n+1},q)f(X_n,q)dq$ *summation is over possible values of X_n which are zero and one*
$f(X_{n+1},q)f(X_n,q) = q^{X_n+X_{n+1}}(1-q)^{2-X_n-X_{n+1}}$

By comparing $\star$ and $\star\star$ one possible value of the stationary distribution is $f(X_n) = \frac{1}{2}$
We can validate that using simulations.
For simulation:
(1) start by generating a bernoulli random variable X_0 with probability of success $p = 1/2$
(2) Then find $p(X_1 = 1|X_0)$ by setting X_{n+1} to 1 and X_n to X_0 in the transition probability equation
(3) This gives us $p^* = P(X_1 = 1|X_0)$. Then generate a Bernoulli random variable with parameter p^*, this is your X_1
(4) Repeat the process with X_1 instead of X_0 and X_2 instead of X_1
The code for this algorithm is shown below.

```
import numpy.random as rn
import scipy.integrate as intgrate



# define a bernoulli random variable
def bernoulli(p):
        """
        defines a bernoulli random variable
        :param p: probability of success
        :return: bernoulli sample
        """
        bern = rn.binomial(1,p)
        return bern
```

```
# define the transition probability
def ptrans(Xn):
        """
        for a value of Xn, returns the probability
        that Xn+1 = 1
        :param Xn: realization of R.V. Xn
        :return: P(Xn+1 | Xn)
        """
        p = intgrate.quad(lambda q: q**(Xn + 1) *
         (1-q)**(2-Xn-1), 0, 1)
        prob = 2.0 * p[0]
        return prob


if __name__ == '__main__':
        # start by generating a bernoulli X0 with parameter 1/2
        X = bernoulli(0.5)
        samples = []  # list of generated samples
        iter = 1
        numiter = 2000
        while iter <= numiter:
                p = ptrans(X)
                X = bernoulli(p)  # generate samples
                samples.append(X)
                iter += 1

        successes = samples.count(1)
        failures = samples.count(0)
        stationary_success = float(successes) / numiter
        stationary_failures = float(failures) / numiter
        print(failures, successes)
        print(stationary_failures, stationary_success)
```

This code prints out at the end the number of successes and failures among the generated samples, and the stationary probability of successes and failures.

For 2000 iterations, the number of failures was 997 and the number of successes was 1003, which confirms a stationary probability of 1/2 such that the probability of failure was 0.4985 and the probability of success was 0.5015 (after averaging over the total number of samples generated)

3.

Consider the integral $I = \int_{-\infty}^{+\infty} \frac{1}{1+x^4} e^{\frac{-x^2}{2}} dx$
Evaluate the integral using a Markov chain sample X_n given by $X_{n+1} \sim N(\rho X_n, 1-\rho^2)$
We have that $X_{n+1} = \rho X_n + \sqrt{1-\rho^2} Z$
Where Z is a standard normal R.V. $Z \sim N(0,1)$
The integral I can be written as $I = \int g(x) f(x)$ where $f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ is a standard normal, and $g(x) = \frac{\sqrt{2\pi}}{1+x^4}$. By sampling x_i from a standard normal, we can estimate I by $\hat{I} = \frac{1}{N} \sum g(x_i)$. Therefore, we have to show that the samples from our Markov chain come from a standard normal. Consider that X_1 is sampled from a standard normal.
$E[X_2] = \rho E[X_1] = 0$
$Var(X_2) = \rho^2 Var(X_1) + (1-\rho^2) = 1$
Since X_2 is the sum of normals, it has a normal distribution. Therefore, using the Markov chain, X_2 is also coming from standard normal. This generalizes for X_n and X_{n+1} by induction.

The following code was implemented to evaluate the integral. The procedure is as follows:
(1) sample X_1 from a standard normal. Find $S = g(X_1)$
(2) get X_2 from X_1 using the Markov chain, with Z sampled from a standard normal.
(3) add $g(X_2)$ to S.
(4) similarly, get X_3 from X_2 using the Markov chain, and add $g(X_3)$ to S.
(5) keep repeating the process.
(6) $\hat{I} = \frac{S}{N}$ where N is the number of samples.

```
"""
This code is used to evaluate the integral using
a Gaussian Markov process.

@cnyahia
"""
import numpy as np
import matplotlib.pyplot as plt


def gx(Xn):
        """
        This is the function g(x)
        :param x: sample from the Markov chain
        :return: g(x)
        """
        gofx = np.sqrt(2 * np.pi) / (1 + Xn**4)
        return gofx

if __name__ == '__main__':
        # start by generating a standard normal R.V.
        X = np.random.normal(0, 1)
        samples = []
        S = 0  # the sum in Ihat
        currentIhat = []
        iter = 1
```

```
rho = 0.1
numiter = 5000
while iter <= numiter:
        S = S + gx(X)
        samples.append(X)
        X = rho * X + np.sqrt(1 - rho**2) * np.random.normal(0, 1)
        currentIhat.append(S/iter)
        iter += 1

Ihat = S / numiter
print(Ihat)

# plot a graph showing convergence to value of integral I
Iterations = range(1, len(currentIhat) + 1)
plt.plot(Iterations, currentIhat, 'g')
plt.title("Convergence of Ihat to the Integral value")
plt.xlabel("Iterations")
plt.ylabel("Value of Ihat")
plt.tight_layout()
plt.savefig('Ihat.png')
plt.show()
```

For $\rho = 0.1$ the estimator $\hat{I}$ converges to 1.69. The rate at which this converges is shown in the Figure 1 below.

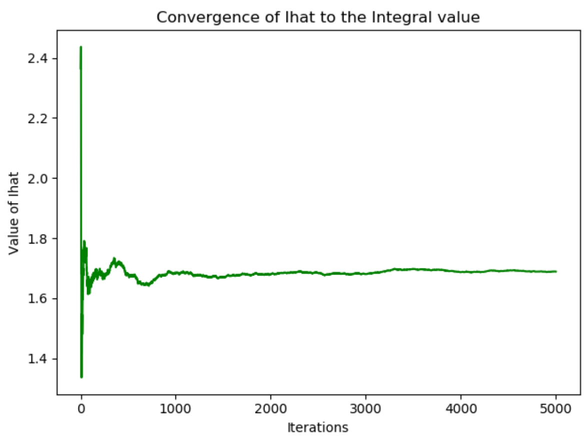

Figure 1: Convergence of the estimator

To find the best value of ρ, we first plot below convergence rate for some values of ρ. Note that the estimator diverges for $\rho = 1$ and $\rho = -1$

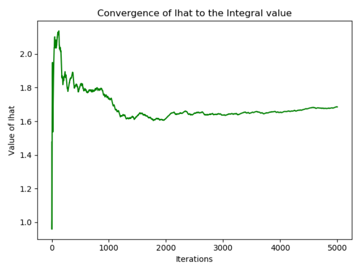

Figure 2: Convergence of the estimator for $\rho = -0.9$

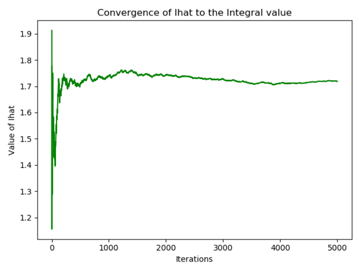

Figure 3: Convergence of the estimator for $\rho = -0.5$

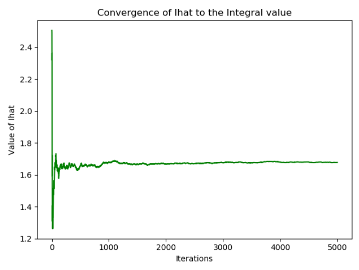

Figure 4: Convergence of the estimator for $\rho = -0.2$

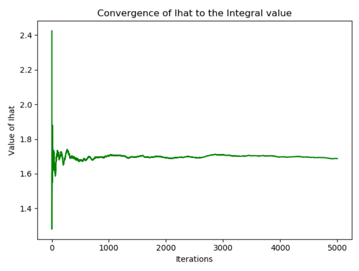

Figure 5: Convergence of the estimator for $\rho = 0$

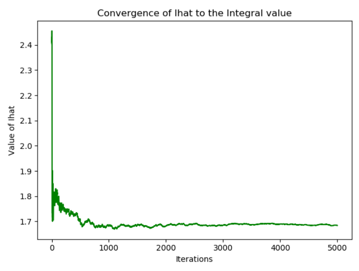

Figure 6: Convergence of the estimator for $\rho = 0.2$

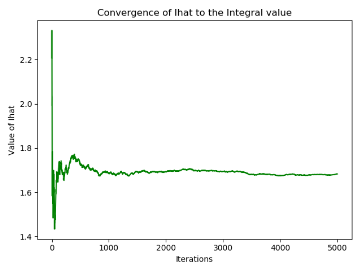

Figure 7: Convergence of the estimator for $\rho = 0.5$

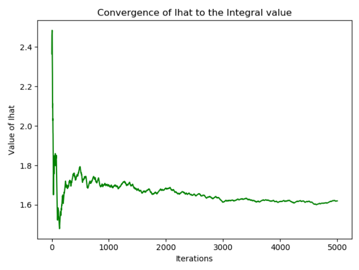

Figure 8: Convergence of the estimator for $\rho = 0.9$

By observing the figures above, it is clear that the estimator converges faster for values of ρ close to zero, which implies low dependence between the Markov chain random variables.

To find the best value of ρ we need to look at the variance of the estimator $\hat{I}$. The lower the variance, the better the value of ρ

We can modify the code as follows to obtain the variance for multiple values of ρ

```
rows = np.arange(-1, 1, 0.05).tolist()
dictvar = {}
for row in rows:
        # start by generating a standard normal R.V.
        X = np.random.normal(0, 1)
```

```
    samples = []
    S = 0  # the sum in Ihat
    currentIhat = []
    iter = 1
    rho = 0.0
    numiter = 50
    VarIhat = []
    while iter <= numiter:
        S = S + gx(X)
        samples.append(X)
        X = rho * X + np.sqrt(1 - rho**2) * np.random.normal(0, 1)
        currentIhat.append(S/iter)
        iter += 1
        if len(currentIhat) > 1:
            VarIhat.append(np.var(currentIhat, ddof=1))

    Ihat = S / numiter
    # print(Ihat)
    dictvar[row] = VarIhat[-1]

print(dictvar)
```

The following dictionary shows the values of the variance after 5000 iterations for ρ values tested by simulation. Dictionary structure is {rho: variance, rho: variance, ...}

{-1.0: 0.0015302660654209629, -0.95: 0.00045173049425096688, -0.9: 0.0017760816019123929, -0.85: 0.00099554111771543892, -0.8: 0.0013459041886798061, -0.75: 0.0010292672066535614, -0.7: 0.00043559368824044582, -0.65: 0.0013235558680598869, -0.6: 0.00091126927507314995, -0.55: 0.00081295561440543004, -0.5: 0.0015056328556494855, -0.45: 0.0012802440749101049, -0.4: 0.0011067060066602945, -0.35: 0.0017992677609650822, -0.3: 0.001005195647025117, -0.25: 0.00083811202834584565, -0.2: 0.00060469822117324538, -0.15: 0.00080614809669697131, -0.1: 0.0012105867629822124, -0.05: 0.0010571570613242755, 0.0: 0.00074250502250812586, 0.05: 0.0013752427903114741, 0.1: 0.00096576164048063498, 0.15: 0.00091824787323872793, 0.2: 0.00033014214092838676, 0.25: 0.0016002961994487285, 0.3: 0.0010327824683807945, 0.35: 0.0015611878954751011, 0.4: 0.00037857666047482792, 0.45: 0.00061953108889730541,

0.5: 0.00053550834274333954, 0.55: 0.00051908912463701 97, 0.6: 0.0007333129380426407, 0.65: 0.0014633497630652367, 0.7: 0.001638936032131002, 0.75: 0.0027633435895519616, 0.8: 0.00093260792218760916, 0.85: 0.00068507258206007722, 0.9: 0.0013877855845619723, 0.95: 0.0014369830543965445}

The lowest variance was at $\rho = 0.2$. However, the results are inconclusive since there are low variance values for other ρ values as well.

To find the ρ that minimizes the variance analytically, we need to minimize $var(\frac{1}{N}\sum \frac{\sqrt{2\pi}}{1+X_i^4}) = \frac{1}{N^2}var(\sum \frac{\sqrt{2\pi}}{1+X_i^4}) = \frac{1}{N^2}(E[(\sum \frac{\sqrt{2\pi}}{1+X_i^4})^2] - E[(\sum \frac{\sqrt{2\pi}}{1+X_i^4})]^2)$ where X_i's are dependent.