Solution Notes for Data Science Example Sheet 4

Question 1

We will use the property that $\Pr(A, B \mid C) = \Pr(A \mid B, C) \cdot \Pr(B \mid C)(*)$ for $\Pr(C) > 0$ and $\Pr(B, C) > 0$.

$$\begin{aligned}
\Pr(X_3 = r \mid X_0 = g) &= \sum_{x_1}\sum_{x_2} \Pr(X_3 = r, X_2 = x_2, X_3 = x_3 \mid X_0 = g) \text{ (by marginalisation)} \\
&= \sum_{x_1}\sum_{x_2} \Pr(X_3 = r \mid X_2 = x_2, X_1 = x_1, X_0 = g) \cdot \Pr(X_2 = x_2, X_3 = x_3 \mid X_0 = g) \text{(by *)} \\
&= \sum_{x_1}\sum_{x_2} \Pr(X_3 = r \mid X_2 = x_2) \cdot \Pr(X_2 = x_2, X_1 = x_1 \mid X_0 = g) \text{ (by memoryless property)} \\
&= \sum_{x_1}\sum_{x_2} P_{x_2 r} \cdot \Pr(X_2 = x_2, X_1 = x_1 \mid X_0 = g) \\
&= \sum_{x_1}\sum_{x_2} P_{x_2 r} \cdot \Pr(X_2 = x_2 \mid X_1 = x_1, X_0 = g) \cdot \Pr(X_1 = x_1 \mid X_0 = g) \\
&= \sum_{x_1}\sum_{x_2} P_{x_2 r} \cdot P_{x_1 x_2} \cdot P_{g x_1}
\end{aligned}$$

Note: In matrix notation this can be written as $(P^3)_{gr}$.

Question 2

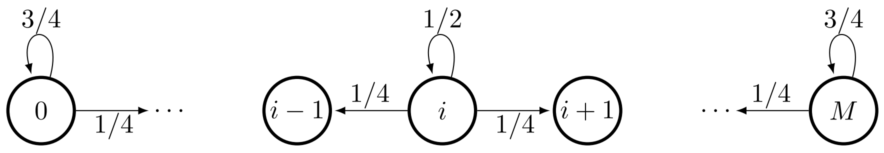
Assuming $a > 0$ and $b > 0$, then there is a non-zero probability path from each state to the other.

The stationary equations give,

$$\begin{pmatrix}\pi_a \\ \pi_b\end{pmatrix} = \begin{bmatrix}1-\alpha & b \\ \alpha & 1-b\end{bmatrix} \cdot \begin{pmatrix}\pi_a \\ \pi_b\end{pmatrix} \Rightarrow \begin{matrix}\pi_a = (1-a)\pi_a + b\pi_b \\ \pi_b = a\pi_a + (1-b)\pi_b\end{matrix} \Rightarrow a\pi_a = b\pi_b.$$

Since π is a distribution we also have the constraint $\pi_a + \pi_b = 1$. Hence, $a(1-\pi_b) = b\pi_b \Rightarrow \pi_b = \frac{a}{a+b}$ and $\pi_a = 1 - \frac{a}{a+b} = \frac{b}{a+b}$.

Question 3



The stationary distribution satisfies $\pi = \pi P$, and $\sum_{n=0}^{M} \pi_n = 1$ or equivalently $\pi^T \cdot \mathbf{1} = 1$.

The code below finds the stationary distribution:

```python
import numpy as np

P = np.zeros((10, 10))

# Transition probabilities for state 0.
P[0, 0] = 3 / 4
P[0, 1] = 1 / 4
# Transition probabilities for state 9
P[9, 9] = 3 / 4
```

```
P[9, 8] = 1 / 4
# Transition probabilities for state 0<i<9.
for i in range(1, 9):
    P[i, i - 1] = 1 / 4
    P[i, i] = 1 / 2
    P[i, i + 1] = 1 / 4

# Sanity check for rows to sum to 1.
assert np.allclose(P @ np.ones(10), np.ones(10))

# Solve these equations simultaneously
A = np.concatenate([(P - np.eye(10)).T, np.ones((1, 10))])
pi = np.linalg.lstsq(A, np.concatenate([np.zeros(10), [1]]))[0]
print(f'The stationary distribution is : {pi}')

# Expected output: [0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1]
```

Note: This is expected since the uniform distribution satisfies $\pi_x = \sum_y \pi_y \cdot P_{yx}$.

Question 4

1. In order to generate the matrix we need to compute the transition probability from state x to state y. Let $A \sim \text{Po}(r \cdot x/d)$ and $B \sim \text{Bin}(x, 1/d)$ and let $D = A - B$. Then we want to compute $\Pr(D = y - x)$ and we do this by marginalisation,

$$\begin{aligned}\Pr(D = y - x) = \Pr(A - B = y - x) &= \sum_{k=\max(0,x-y)}^{x} \Pr(B = k, A - B = y - x)\\ &= \sum_{k=\max(0,x-y)}^{x} \Pr(B = k, A = y - x + k)\\ &= \sum_{k=\max(0,x-y)}^{x} \Pr(B = k) \cdot \Pr(A = y - x + k).\end{aligned}$$

   where $\Pr(B = k)$ and $\Pr(B = y - x + k)$ we can compute using the pmfs of the Poisson and the Binomial distributions. The code below does this:

```
def generate_matrix(N, r, d):
    P = np.zeros((N + 1, N + 1))
    for x in range(N):
        binom_param = 1 / d
        poisson_param = r * x / d
        for y in range(N):
            # Evaluate P(X_{n+1} = y | X_n = x).
            prob = 0
            for k in range(max(y - x, 0), x + 1):
                prob += scipy.stats.binom.pmf(k=k, n=x, p=binom_param) * \
                        scipy.stats.poisson.pmf(k=y - (x - k), mu=poisson_param)
            P[x, y] = prob
    # Set the absorbing state.
    P[:N, N] = 1 - np.sum(P[:N, :N], axis=1)
    P[N, N] = 1
    return P
```

2. We now need to solve the hitting equations, i.e. find the vector π such that $\pi_x = \sum_y P_{xy}\pi_y$ and $\pi_0 = 1$ (since it is the only item of the hitting set). We also need to set $\pi_N = 0$ since N is absorbing (and the previous equations leave it unconstrained).

The complete code is shown below:

```python
import numpy as np
import scipy.stats


def generate_matrix(N, r, d):
    P = np.zeros((N + 1, N + 1))
    for x in range(N):
        binom_param = 1 / d
        poisson_param = r * x / d
        for y in range(N):
            # Evaluate P(X_{n+1} = y | X_n = x).
            prob = 0
            for k in range(max(x - y, 0), x + 1):
                prob += scipy.stats.binom.pmf(k=k, n=x, p=binom_param) * \
                        scipy.stats.poisson.pmf(k=y - (x - k), mu=poisson_param)
            P[x, y] = prob
    # Set the absorbing state.
    P[:N, N] = 1 - np.sum(P[:N, :N], axis=1)
    P[N, N] = 1
    return P


r, d = 1.1, 14
N = 100

P = generate_matrix(N, r, d)
# Solve for the hitting probability vector as shown in Section 10.3.
Q = np.copy(P)
Q[0, :] = 0  # The elements of the hitting set have a probability 1.
Q[N, :] = 0  # Absorbing states have 0 probability of reaching the hitting set.
hit_probabilities = np.linalg.solve(np.eye(N + 1) - Q, np.concatenate([np.ones(1),
↪   np.zeros(N)]))
print(f"Probability of hitting 0 starting at X50: {hit_probabilities[50]}")
# Expected output: Probability of hitting 0 starting at X50: 0.007035837489220857
```

Question 5

To show that it is the stationary distribution we need to show that it is a distribution ($\pi_n \geq 0$ and $\sum_n \pi_n = 1$) and that it satisfies $\pi_x = \sum_y \pi_y P_{yx}$.

We begin by verifying that π_n is a distribution. One could argue directly that this is a Geometric distribution with $p = \frac{a}{b} < 1$. But we can also prove this from first principles.

$$\pi_n = (1 - \frac{a}{b}) \left(\frac{a}{b}\right)^n > 0 \quad (\text{since } b > a > 0 \text{ so } 0 < \frac{a}{b} < 1)$$

$$\sum_{n=0}^{\infty} \pi_n = \sum_{n=0}^{\infty} (1 - \frac{a}{b}) \left(\frac{a}{b}\right)^n = (1 - \frac{a}{b}) \cdot \frac{1}{1 - \frac{a}{b}} = 1$$

Now for the remaining condition,

$$\pi_0 = (1 - a)\pi_0 + b\pi_1 = (1 - \frac{a}{b})(1 - a) + b(1 - \frac{a}{b})\frac{a}{b} = (1 - \frac{a}{b})(1 - a + a) = (1 - \frac{a}{b}) = \pi_0$$

$$\begin{aligned}\pi_{n+1} &= a\cdot\pi_{n-1} + (1-a-b)\cdot\pi_n + b\cdot\pi_{n+1}\\ &= a(1-a/b)(a/b)^{n-1} + (1-a-b)(1-a/b)(a/b)^n + b(1-a/b)(a/b)^{n+1}\\ &= (1-a/b)(a/b)^{n-1}(a + (1-a-b)(a/b) + b(a/b)^2)\\ &= (1-a/b)(a/b)^{n-1}(ab + (1-a-b)a + a^2)/b\\ &= (1-a/b)(a/b)^{n-1}a/b\\ &= (1-a/b)(a/b)^n\end{aligned}$$

Question 6

(a) In order for π to be a distribution we need to show that it is a distribution (holds by assumption) and show that it satisfies $\pi_x = \sum_y \pi_y P_{yx}$ for all x.

$$\pi_x P_{xy} = \pi_y P_{yx} \Rightarrow \sum_y \pi_x P_{xy} = \sum_y \pi_y P_{yx} \Rightarrow \pi_x \sum_y P_{xy} = \sum_y \pi_y P_{yx} \Rightarrow \pi_x = \sum_y \pi_y P_{yx}$$

(b) Note that the stationary distribution exists because the graph is connected. We will try to find a distribution that satisfies the detailed balanced equations, which is stronger than satisfying the equations for the stationary distributions. In order for the distribution to satisfy the detailed balanced equations, we need to have for any two directly connected nodes u and v,

$$\pi_u \cdot \frac{1}{\deg(u)} = \pi_v \cdot \frac{1}{\deg(v)}$$

However, note that if $u \leftrightarrow v$ and $v \leftrightarrow w$, we have that

$$\pi_u \cdot \frac{1}{\deg(u)} = \pi_v \cdot \frac{1}{\deg(v)} \quad \text{and} \quad \pi_v \cdot \frac{1}{\deg(v)} = \pi_w \cdot \frac{1}{\deg(w)}$$

By the transitivity property of connectedness, we get that the ratio $\pi_u \cdot \frac{1}{\deg(u)}$ is constant (say k), independent of the node u. So $\pi_u = \deg(u)\cdot k$. By summing on both sides of the equation,

$$\sum_u \pi_u = \sum_u \deg(u)\cdot k \Rightarrow 1 = k\cdot\sum_u \deg(u) \Rightarrow k = \frac{1}{\sum_u \deg(u)}.$$

Hence, for every node v, we have $\pi_v = \frac{\deg(v)}{\sum_u \deg(u)}$. Since this distribution satisfies the detailed balanced equations, it also satisfies the second requirement for the stationary distribution by (a).

Note: An alternative approach is to guess the stationary distribution $\pi_v = \frac{\deg(v)}{\sum_u \deg(u)}$ and prove that it is indeed a stationary distribution.

Question 7

(a)

$$\begin{tikzcd} X_0 \arrow[r] \arrow[d] & X_1 \arrow[r] \arrow[d] & X_2 \arrow[r] \arrow[d] & \cdots \\ Y_0 & Y_1 & Y_2 & \end{tikzcd}$$

(b) The base case is just an application of Bayes' rule.

$$\Pr(x_0 \mid y_0) = \frac{\Pr(y_0 \mid x_0)\Pr(x_0)}{\Pr(y_0)} = (\text{const}) \cdot \Pr(y_0 \mid x_0)\Pr(x_0)$$

$$\Pr(x_n \mid h) = \sum_{x_{n-1}} \Pr(x_n, x_{n-1} \mid h) = \sum_{x_{n-1}} \Pr(x_n \mid x_{n-1})\cdot\Pr(x_{n-1} \mid h)$$

$$\Pr(x_n \mid h, y_n) = \frac{\Pr(y_n \mid x_n, h)\Pr(x_n \mid h)}{\Pr(y_n \mid h)} = (\text{const}) \ \Pr(y_n \mid x_n)\Pr(x_n \mid h)$$

Note that any probability involving only y_n and h can be regarded constant.

(c) By combining the two terms above we get

$$\begin{aligned}\Pr(x_n \mid y_n, h) &= (\text{const}) \ \cdot \Pr(y_n \mid x_n) \sum_{x_{n-1}} \Pr(x_{n-1} \mid h) \cdot \Pr(x_n \mid x_{n-1}) \\ &= (\text{const}) \ \cdot E_{x_n y_n} \sum_{x_{n-1}} \Pr(x_{n-1} \mid h) \cdot P_{x_n x_{n-1}}\end{aligned}$$

where E is the emission matrix and P is the transition matrix. By denoting $\pi_x^{(n)} = \Pr(x \mid y_n, h)$, we get the following recursive equation $\pi^{(n+1)} \propto E \odot (\pi^{(n)} \cdot P)$ (where $\odot$ is element-wise multiplication). This leads to the following filter implementation.

```python
def filter(P, E, ys):
    pi = np.ones(MAX_STATE + 1) / (MAX_STATE + 1)
    for yn in ys:
        # Compute const * P(yn | xn) * Sum_{x_{n-1}} P(x_{n-1} | h) P(x_n | x_{n-1})
        pi = np.multiply(E[:, yn], pi @ P)
        # Normalise the probability vector.
        pi = pi / np.sum(pi)
    return pi
```

The entire code is shown below:

```python
import numpy as np

MAX_STATE = 9


def generate_emission_matrix():
    # Emission matrix. E[x, y] = Pr(y | x)
    E = np.zeros((MAX_STATE + 1, MAX_STATE + 1))
    for x in range(MAX_STATE + 1):
        E[x, max(x - 1, 0)] += 1 / 3
        E[x, x] += 1 / 3
        E[x, min(x + 1, MAX_STATE)] += 1 / 3
    assert np.allclose(np.sum(E, axis=1), np.ones(MAX_STATE + 1))
    return E


def generate_transition_matrix():
    # Transition matrix. P[x,y] = Pr( y | x)
    P = np.zeros((MAX_STATE + 1, MAX_STATE + 1))
    for x in range(MAX_STATE + 1):
        P[x, max(x-1, 0)] += 1 / 4
        P[x, x] += 1 / 2
        P[x, min(x + 1, MAX_STATE)] += 1 / 4
    assert np.allclose(np.sum(P, axis=1), np.ones(MAX_STATE + 1))
    return P


def filter(P, E, ys):
    pi = np.ones(MAX_STATE + 1) / (MAX_STATE + 1)
    for yn in ys:
        # Compute const * P(yn | xn) * Sum_{x_{n-1}} P(x_{n-1} | h) P(x_n | x_{n-1})
        pi = np.multiply(E[:, yn], pi @ P)
        # Normalise the probability vector.
        pi = pi / np.sum(pi)
    return pi
E = generate_emission_matrix()
P = generate_transition_matrix()

print(filter(P, E, [3, 3, 4, 5, 4, 3, 3, 2, 2, 1, 2, 1, 2, 2]))
# Expected output: [0. 0.3749904  0.41518442 0.20982518 0. 0. 0. 0. 0. 0.]

print(filter(P, E, [3, 3, 4, 9]))
# Expected output: [nan nan nan nan nan nan nan nan nan nan]
```

(d) It fails by divide-by-zero because the event $\{X_n = 9 \mid X_{n-1} = 4\}$ has zero probability, so Bayes' rule does not apply.

Note: It is possible to avoid the matrix multiplication since the matrix is very sparse, i.e. it has 3 entries on every row.

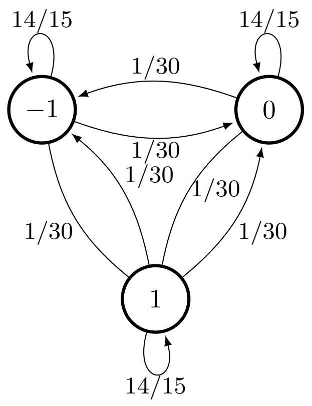
Question 8

We only need to change the transition matrix, so that we can reach any state from any other state with probability at least ϵ.

```
eps = 0.01

def generate_transition_matrix():
    # Transition matrix. P[x,y] = Pr( y | x)
    P = (eps / (MAX_STATE + 1)) * np.ones((MAX_STATE + 1, MAX_STATE + 1))
    for x in range(MAX_STATE + 1):
        P[x, max(x-1, 0)] += 1 / 4 * (1 - eps)
        P[x, x] += 1 / 2 * (1 - eps)
        P[x, min(x + 1, MAX_STATE)] += 1 / 4 * (1 - eps)
    assert np.allclose(np.sum(P, axis=1), np.ones(MAX_STATE + 1))
    return P


print(filter(P, E, [3, 3, 4, 5, 4, 3, 3, 2, 2, 1, 2, 1, 2, 2]))
# Expected output: [0. 0.3749904  0.41518442 0.20982518 0. 0. 0. 0. 0. 0.]
# Contrast with  : [0. 0.37478324 0.41479646 0.2104203  0. 0. 0. 0. 0. 0.]

print(filter(P, E, [3, 3, 4, 9]))
# Expected output: [0. 0. 0. 0. 0. 0. 0. 0. 0.33333333 0.66666667]
# Contrast with  : [nan nan nan nan nan nan nan nan nan nan]
```

For the alternative scheme, we can deterministically achieve this by discarding entries y_n whose probability is 0. (Note: we might discard many entries..)

Question 9



```
import numpy as np

N = 10  # Number of states for X
total_states = 3 * N


def convert_pair(x, v):
    return v * N + x


def convert_single(a):
    return a % N, a // N


P = np.zeros((total_states, total_states))
for a in range(0, total_states):
    x, v = convert_single(a)
    for dv in range(0, 3):
        pv = 14/15 if v == dv else 1/30
        xx = max(0, min(N - 1, x + v - 1))
        # Different velocities might lead to the same states
        # so we need to accumulate the transition probabilities.
        P[a, convert_pair(xx, dv)] += pv

# Sanity check for rows to sum to 1.
assert np.allclose(P @ np.ones(total_states), np.ones(total_states))

# Solve these equations simultaneously.
A = np.concatenate([(P - np.eye(total_states)).T, np.ones((1, total_states))])
pi = np.linalg.lstsq(A, np.concatenate([np.zeros(total_states), [1]]))[0]
print(f'The stationary distribution is : {pi}')
```

Question 10

The chain is irreducible because the teleport functionality for $d < 1$ guarantees that every node can be visited from any other node.

We begin by writing out the transition matrix

$$P_{uv} = \begin{cases} \frac{1-d}{|V|} + \frac{d}{|\Gamma_u|} & \text{for } v \in \Gamma_u \\ \frac{1-d}{|V|} & \text{otherwise.} \end{cases}$$

To verify that the given π is a distribution we notice that:

1. $\pi_u \geq 0$ for every $u \in V$.

2. $\sum_{v\in V} = 1$ for $d < 1$ since

$$\begin{aligned}
\sum_{v\in V} \pi_v &= \sum_v \frac{1-d}{|V|} + d \sum_{u:u\to v} \frac{\pi_v}{|\Gamma_u|} \\
&= \frac{1-d}{|V|} \cdot |V| + d \sum_v \sum_{u:u\to v} \frac{\pi_v}{|\Gamma_u|} \\
&= (1-d) + d \sum_{v\in V} \sum_{u:u\to v} \frac{\pi_u}{|\Gamma_u|} \\
&= (1-d) + d \sum_{u\in V} \sum_{v:u\to v} \frac{\pi_u}{|\Gamma_u|} \text{ (By changing summation order)} \\
&= (1-d) + d \sum_{u\in V} |\Gamma_u| \cdot \frac{\pi_u}{|\Gamma_u|} \text{ (Since the inner term depends only on } u\text{)} \\
&= (1-d) + d \cdot \sum_{u\in V} \pi_u \Rightarrow (1-d) \sum_{u\in V} = 1-d \Rightarrow \sum_{u\in V} = 1.
\end{aligned}$$

3. It satisfies $\pi_v = \sum_{u\in V} P_{vu}\pi_u$, since

$$\begin{aligned}
\pi_v &= \sum_{u\in V} P_{uv}\pi_u \Leftrightarrow \\
\pi_v &= \sum_{u\in V} \pi_u \frac{1-d}{|V|} + \sum_{u:u\to v} (1-d)\pi_u \frac{1}{|\Gamma_u|} \Leftrightarrow \\
\pi_v &= \frac{1-d}{|V|} \sum_{u\in V} \pi_u + \sum_{u:u\to v} (1-d)\pi_u \frac{1}{|\Gamma_u|} \Leftrightarrow \\
\pi_v &= \frac{1-d}{|V|} + (1-d) \sum_{u:u\to v} \frac{\pi_u}{|\Gamma_u|}
\end{aligned}$$

where the last statement holds.

The code below implements the PageRank algorithm for the graph of 10.3.2.

```python
import numpy as np


neighbours = [
    [1, 2], [3, 4], [0, 5],
    [1, 5], [0], [4, 2]
]
V = len(neighbours)


def generate_transition_matrix(d):
    P = np.zeros((V, V))
    for v in range(V):
        neigh_count = len(neighbours[v])
        for neigh in neighbours[v]:
            P[v, neigh] = d / neigh_count
        factor = d if neigh_count > 0 else 0
        for u in range(V):
            P[v, u] += (1 - factor) / V
    return P


def get_stationary_distribution(P):
    # Sanity check for rows to sum to 1.
    assert np.allclose(P @ np.ones(V), np.ones(V))

    # Solve these equations simultaneously.
    A = np.concatenate([(P - np.eye(V)).T, np.ones((1, V))])

    return np.linalg.lstsq(A, np.concatenate([np.zeros(V), [1]]))[0]


def run_for(d):
    P = generate_transition_matrix(d)
    pi = get_stationary_distribution(P)
    print(f'The stationary distribution for d={d} is :\n {pi}')


run_for(0.85)
run_for(0.05)
run_for(0.01)

# Expected output;
# The stationary distribution for d=0.85 is :
#  [0.2404889  0.16821698 0.18948795 0.09649221 0.15877238 0.14654157]
# The stationary distribution for d=0.05 is :
#  [0.17083575 0.16666673 0.16676835 0.1625     0.16666413 0.16656504]
# The stationary distribution for d=0.01 is :
#  [0.16750002 0.16666667 0.16667081 0.16583333 0.16666665 0.16666252]
```

As $d \to 0$, the stationary distribution becomes more and more uniform, this is expected because the underlying graph structure matters less and less. On the other hand when $d = 1$, the graph structure only affects the transition probabilities.

Question 11

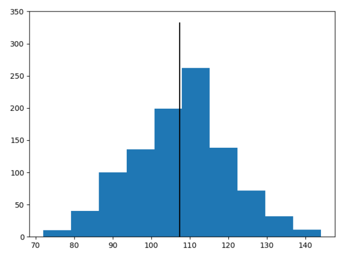
We start by deriving the recurrence relation, noticing that the expectation for the Poisson distribution is just its parameter and the expectation for the Binomial it is the product of its two parameters,

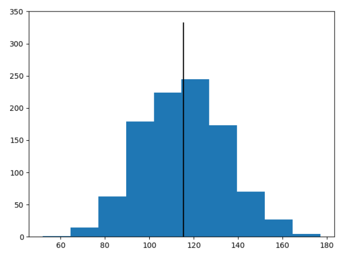
$$\begin{aligned} x_{n+1} = E[X_{n+1} \mid X_n = x_n] &= x_n + \frac{r \cdot x_n}{d} - \frac{x_n}{d} = x_n \left(1 + \frac{r}{d} - \frac{1}{d}\right) \\ &= x_0 \cdot \left(1 + \frac{r}{d} - \frac{1}{d}\right)^n \end{aligned}$$

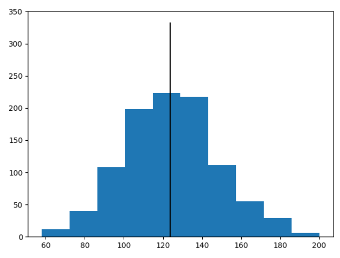
The following code collects samples and plots a histogram of the values obtained. Notice that the values are concentrated around the expectation. (Note: the code will be slow for $N = 200$, you can start with smaller parameters)

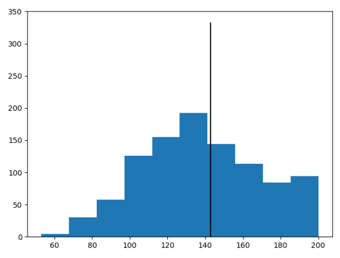
```
import numpy as np
import scipy.stats
import matplotlib.pyplot as plt


def generate_matrix(N, r, d):
    P = np.zeros((N + 1, N + 1))
    for x in range(N):
        binom_param = 1 / d
        poisson_param = r * x / d
        for y in range(N):
            # Evaluate P(X_{n+1} = y | X_n = x).
            prob = 0
            for k in range(-min(y - x, 0), x + 1):
```

```
            prob += scipy.stats.binom.pmf(k=k, n=x, p=binom_param) * \
                    scipy.stats.poisson.pmf(k=y - (x - k), mu=poisson_param)
        P[x, y] = prob
    # Set the absorbing state.
    P[:N, N] = 1 - np.sum(P[:N, :N], axis=1)
    P[N, N] = 1
    return P


def run_simulation(x0, P, n):
    x = x0
    while n > 0:
        p = P[x]
        x = np.random.choice(a=len(p), p=p)
        n = n - 1
    return x


def calculate_expected(x0, r, d, n):
    return x0 * ((1 + r/d - 1/d) ** n)


r, d = 1.1, 14
N = 200
samples = 1000
x0 = 100

P = generate_matrix(N, r, d)


def plot_for(steps):
    v_samples = [run_simulation(x0, P, steps) for _ in range(samples)]

    plt.hist(v_samples)
    plt.vlines(x=calculate_expected(x0, r, d, steps), ymin=0, ymax=samples/3)
    plt.show()


plot_for(steps=10)
plot_for(steps=20)
plot_for(steps=30)
plot_for(steps=50)
```


0
50
100
150
200
250
300
350
70
80
90
100
110
120
130
140


0
50
100
150
200
250
300
350
60
80
100
120
140
160
180


0
50
100
150
200
250
300
350
60
80
100
120
140
160
180
200


0
50
100
150
200
250
300
350
60
80
100
120
140
160
180
200