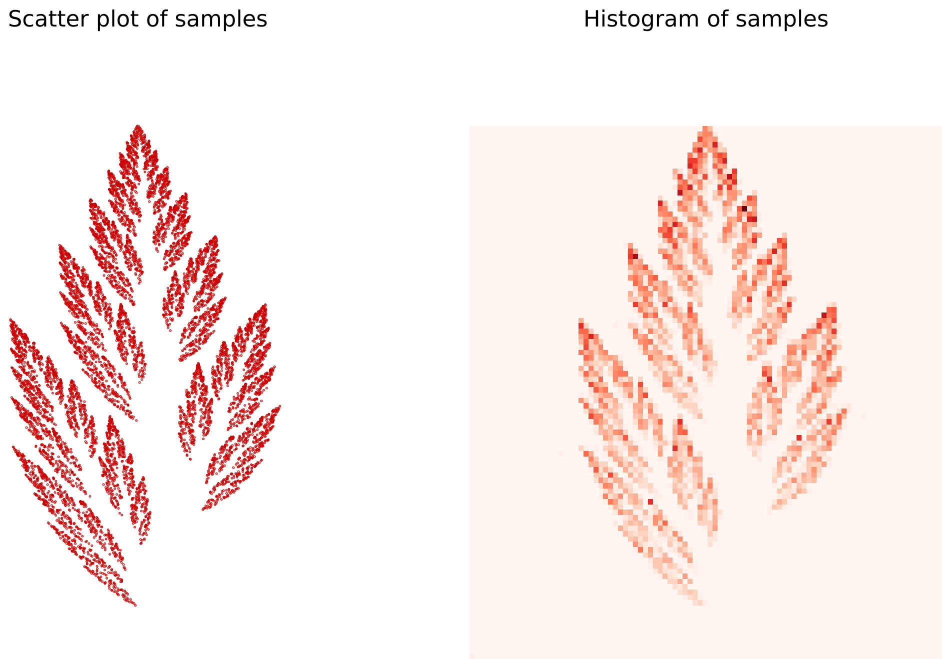
SOLUTIONS 7

Solution 7.1. The code is given below.

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

# simulate a markov chain to generate a fern

w_1 = 0.2993
w_2 = 0.7007

a_1 = np.array([[0.4, -0.3733], [0.06, 0.6]])
b_1 = np.array([[0.3533], [0.0]])
a_2 = np.array([[-0.8, -0.18], [0.1371, 0.8]])
b_2 = np.array([[1.1], [0.1]])

print(b_1.shape)

printIndex = 1

n = 10000
x = np.zeros((2, n))
x[:, 0] = np.array([0.0, 0.0])

fig = plt.figure(figsize=(10, 6))

for k in range(1, n):
    r = np.random.uniform(0, 1)
    if r < w_1:
        x[:, k] = a_1 @ x[:, k - 1] + b_1[:, 0]
    else:
        x[:, k] = a_2 @ x[:, k - 1] + b_2[:, 0]

# scatter samples and plot the histogram in a 1 x 2 plot
plt.clf()
plt.subplot(1, 2, 1)
plt.scatter(x[0, :], x[1, :], s=0.1, color=[0.8, 0, 0])
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['bottom'].set_visible(False)
plt.gca().spines['left'].set_visible(False)
plt.gca().set_xticks([])
plt.gca().set_yticks([])
plt.gca().set_xlim(0, 1.05)
plt.gca().set_ylim(0, 1)
plt.title("Scatter plot of samples")

plt.subplot(1, 2, 2)
plt.hist2d(x[0, :], x[1, :], bins=100, cmap='Reds')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['bottom'].set_visible(False)
plt.gca().spines['left'].set_visible(False)
plt.gca().set_xticks([])
plt.gca().set_yticks([])
plt.gca().set_xlim(0, 1.05)
plt.gca().set_ylim(0, 1)
plt.title("Histogram of samples")
```



The fern generated by the Markov chain and the histogram of the samples.

```
plt.show()
```

If you have implemented it, then you should have seen the Fern plot above.

Solution 7.2. We would like to check

$$K(x|y)p_\star(y) = K(y|x)p_\star(x).$$

Let us write both parts and check

$$\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(x-ay)^2}{2}\right)\frac{1}{\sqrt{2\pi(1/1-a^2)}}\exp\left(-\frac{y^2(1-a^2)}{2}\right) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(y-ax)^2}{2}\right)\frac{1}{\sqrt{2\pi(1/1-a^2)}}\exp\left(-\frac{x^2(1-a^2)}{2}\right)$$

The constants cancel and we end up with

$$\exp\left(-\frac{(x-ay)^2}{2}\right)\exp\left(-\frac{y^2(1-a^2)}{2}\right) = \exp\left(-\frac{(y-ax)^2}{2}\right)\exp\left(-\frac{x^2(1-a^2)}{2}\right).$$

Let us look at what is inside exponentials to make our lives easier:

$$x^2 - 2axy + a^2y^2 + y^2 - a^2y^2 = y^2 - 2axy + a^2x^2 + x^2 - a^2x^2.$$

Both sides cancel, hence we obtain equality.

Solution 7.3. Let us denote starting point $x' = x_0$. We know that one iteration of the method goes

$$x_1 = ax_0 + W_1,$$

where $W_1 \sim \mathcal{N}(0,1)$. We can expand this recursion

$$\begin{aligned}
x_1 &= a x_0 + W_1, \\
x_2 &= a x_1 + W_2 = a^2 x_0 + a W_1 + W_2 \\
x_3 &= a x_2 + W_3 = a^3 x_0 + a^2 W_1 + a W_2 + W_3, \\
&\vdots \\
x_n &= a^n x_0 + \sum_{i=0}^{n-1} a^i W_{i+1}
\end{aligned}$$

This gives us $K^{(n)}(x_n|x_0)$. In particular, we see that this is a distribution with mean $a^n x_0$ (as W_i are zero mean). The variance can be computed as

$$\mathsf{var}(X_n) = \sum_{i=0}^{n-1} a^{2i} \mathsf{var}(W_{i+1}),$$

which results in the geometric sum and we obtain

$$\mathsf{var}(X_n) = \frac{1 - a^{2n}}{1 - a^2}.$$

As a result, we have analytically worked out the kernel as

$$K^{(n)}(x_n|x_0) = \mathcal{N}\left(x_n; a^n x_0, \frac{1 - a^{2n}}{1 - a^2}\right).$$

Taking the limit $n \to \infty$ gives us the conclusion we wanted.

Solution 7.4. The code is given below. This code implements both parts of the exercise, namely, Random walk Metropolis and MALA.

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

rng = np.random.default_rng(35)

# banana function for testing MCMC
def banana(x, y):
    return -x**2 / 10 - y**4 / 100 - 2 * (y - x**2)**2

# for surf plot banana 2d
x_bb = np.linspace(-4, 4, 100)
y_bb = np.linspace(-2, 6, 100)
X_bb, Y_bb = np.meshgrid(x_bb, y_bb)
Z_bb = np.exp(banana(X_bb, Y_bb))

def grad_banana(x, y):
    return np.array([-x/5 + 8 * y * x - 8 * x**3, -(2 * y**3)/5 - 4 *
                                        y + 4 * x**2])

def log_MALA_kernel(x_s, x, gamma, grad_banana):
    return - (1/(4 * gamma)) * np.linalg.norm(x_s - x - gamma *
                                        grad_banana(x[0], x[1]))**2

N = 100000
```

```python
samples_RW = np.zeros((2, N))
samples_Langevin = np.zeros((2, N))

# initial values
x = 0
y = 0
samples_RW[:, 0] = np.array([x, y])
samples_Langevin[:, 0] = np.array([x, y])

# parameters
gamma = 0.01

sigma_rw = 0.5
sigma_langevin = np.sqrt(2 * gamma) # since metropolis step corrects this, we can have larger variance: smaller variance works worse.
fig = plt.figure(figsize=(20, 5))

burnin = 20

for n in range(1, N):
    # random walk
    x_s = samples_RW[:, n-1] + sigma_rw * rng.normal(0, 1, 2)
    # metropolis
    u = rng.uniform(0, 1)

    if np.log(u) < banana(x_s[0], x_s[1]) - banana(samples_RW[0, n-1], samples_RW[1, n-1]):
        samples_RW[:, n] = x_s
    else:
        samples_RW[:, n] = samples_RW[:, n-1]

    # langevin
    x_s_l = samples_Langevin[:, n-1] + gamma * grad_banana(samples_Langevin[0, n-1], samples_Langevin[1, n-1]) + sigma_langevin * rng.normal(0, 1, 2)
    # metropolis
    u = rng.uniform(0, 1)

    if np.log(u) < banana(x_s_l[0], x_s_l[1]) - banana(samples_Langevin[0, n-1], samples_Langevin[1, n-1]) + log_MALA_kernel(samples_Langevin[:, n-1], x_s_l, gamma, grad_banana) - log_MALA_kernel(x_s_l, samples_Langevin[:, n-1], gamma, grad_banana):
        samples_Langevin[:, n] = x_s_l
    else:
        samples_Langevin[:, n] = samples_Langevin[:, n-1]

plt.clf()
# make fonts bigger
plt.rcParams.update({'font.size': 15})
plt.subplot(1, 3, 1)
plt.title('Target Distribution')
```

```
plt.contourf(X_bb, Y_bb, Z_bb, 100, cmap='RdBu')
plt.subplot(1, 3, 2)
plt.hist2d(samples_RW[0, burnin:n], samples_RW[1, burnin:n], 100, cmap='RdBu', range=[[-4, 4], [-2, 6]], density=True)
plt.title('Random Walk Metropolis')

plt.subplot(1, 3, 3)
plt.hist2d(samples_Langevin[0, burnin:n], samples_Langevin[1, burnin:n], 100, cmap='RdBu', range=[[-4, 4], [-2, 6]], density=True)
plt.title('Metropolis Adjusted Langevin Algorithm')
plt.show()
```