Statistics for Data Science
Wintersemester 2024/25
Solutions to the 13th exercise

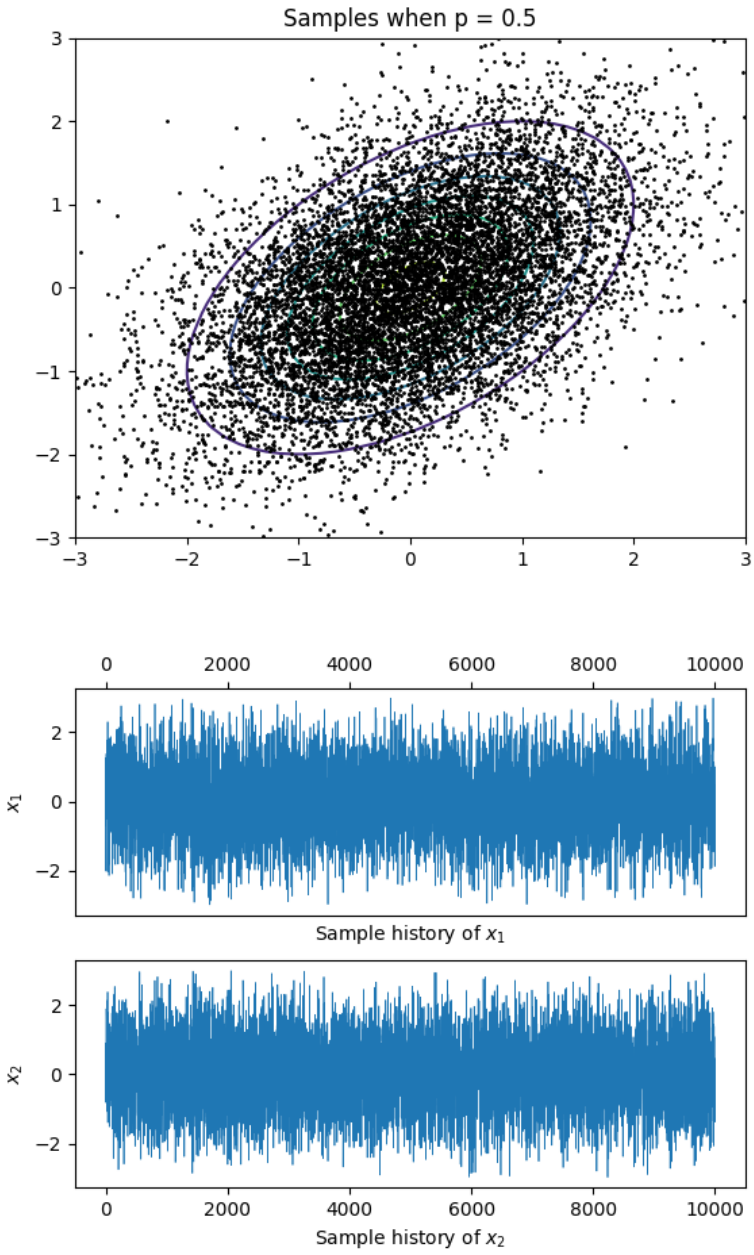
1.

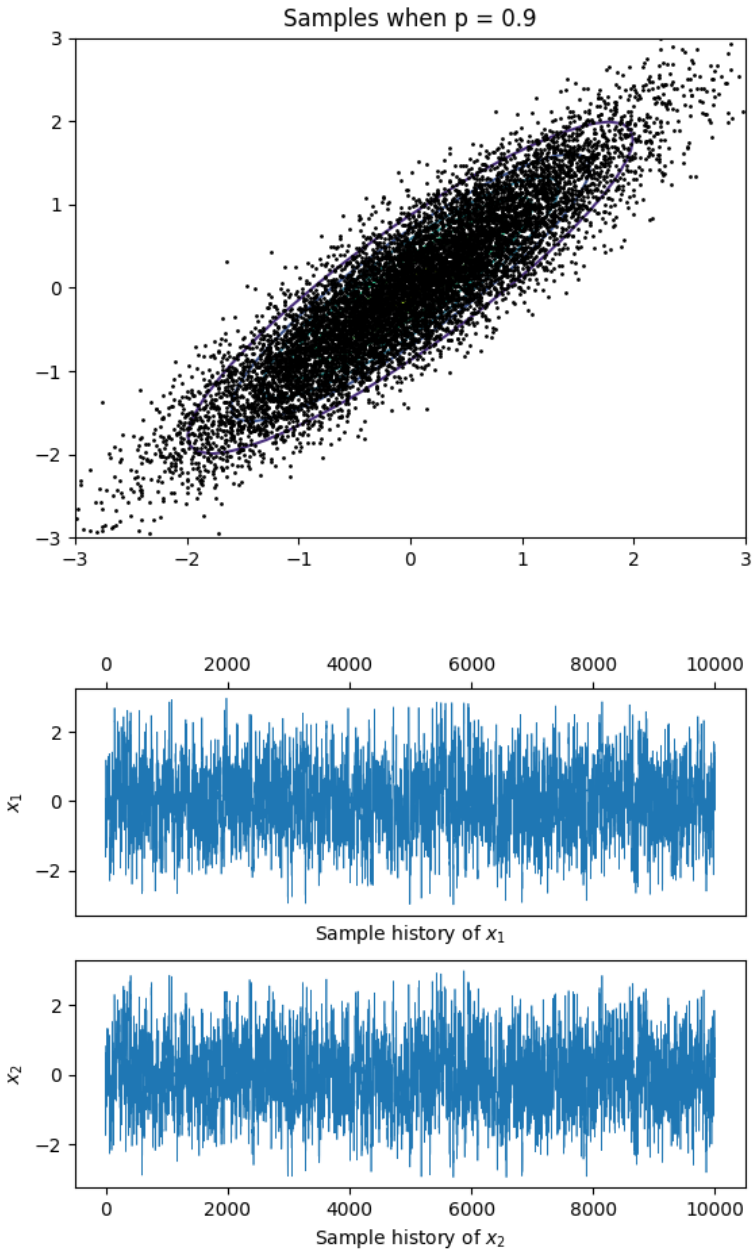
Solution using Python

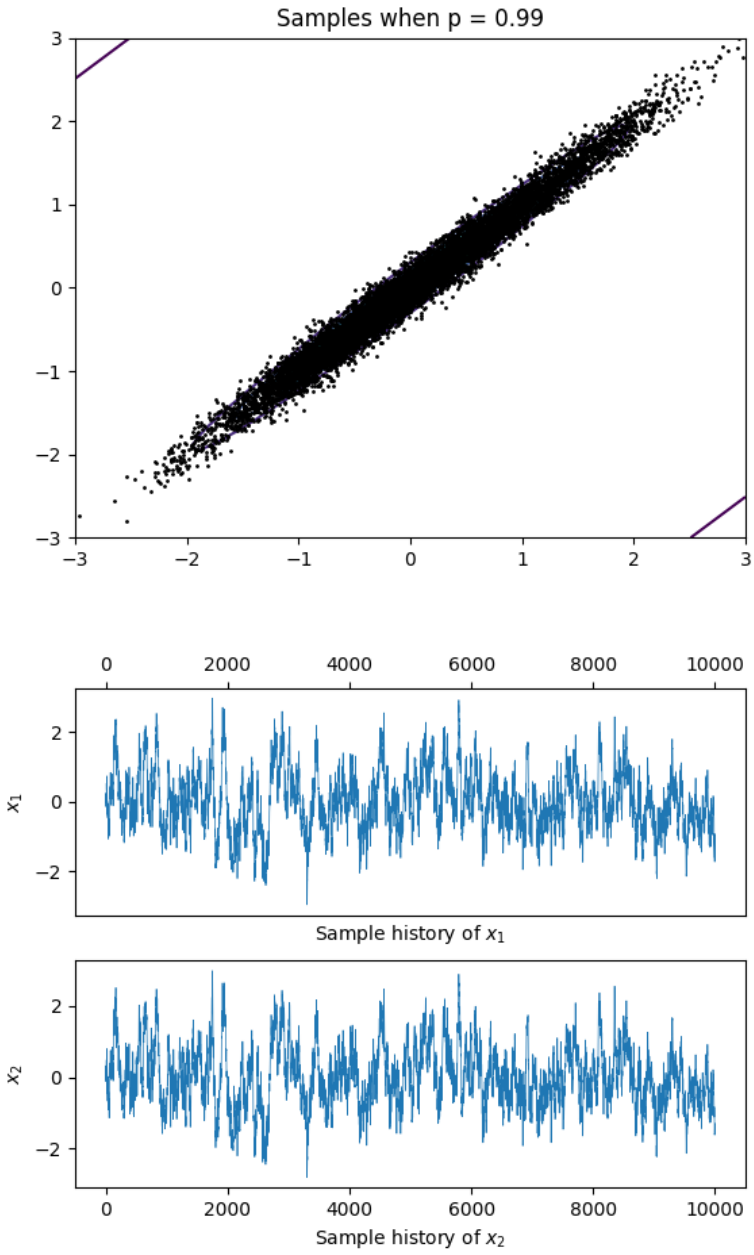
```
import numpy as np
from numpy.matlib import repmat
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

for p in [.5,.9,.99,.999]:
    print('Case p = ' + str(p))
    mu = np.array([0,0])
    G = np.array([[1,p],[p,1]])
    xx = np.linspace(-3,3,1000)
    x = np.array([0,0])
    # "Vectorize" the conditional multinormal distributions
    # w.r.t. the first and second argument, respectively
    pdf1 = lambda x1,x2: multivariate_normal.pdf(
                         np.vstack((x1,repmat(x2,1,x1.size))).T,mu,G)
    pdf2 = lambda x1,x2: multivariate_normal.pdf(
                         np.vstack((repmat(x1,1,x2.size),x2)).T,mu,G)
    samples = [x]
    nsize = 10000
    for _ in range(nsize):
        u = np.random.uniform()
        P = np.cumsum(pdf1(xx,x[1]))
        P = P/P[-1]
        ind = np.where(u<=P)[0][0]
        x1 = xx[ind]
        u = np.random.uniform()
        P = np.cumsum(pdf2(x1,xx))
        P = P/P[-1]
        ind = np.where(u<=P)[0][0]
        x2 = xx[ind]
        x = np.array([x1,x2])
        samples.append(x)
    samples = list(zip(*samples)) # transpose list
    cm = np.mean(samples,axis=1)
    st = np.std(samples,axis=1)
    print('Conditional mean estimate ' + str(cm))
    print('Marginal standard deviations: ' + str(st))
    X,Y = np.meshgrid(np.linspace(-3,3),np.linspace(-3,3))
```

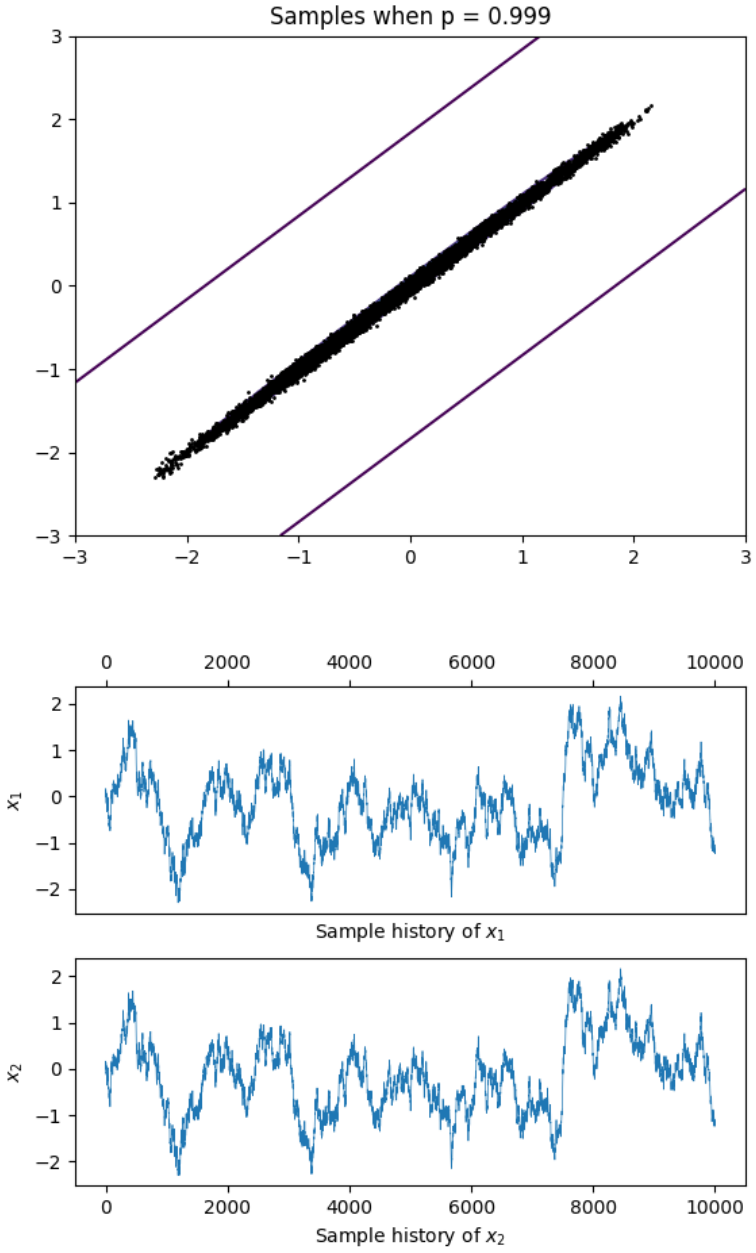
```
    rv = multivariate_normal(mu,G)
    Z = rv.pdf(np.dstack((X,Y)))

    plt.contour(X,Y,Z)
    plt.plot(samples[0],samples[1],'.',color='black',markersize=2)
    plt.title('Samples when p = ' + str(p))
    plt.show()

    fig,ax = plt.subplots(2,1)
    ax[0].plot(samples[0],linewidth=.5)
    ax[0].set_xlabel('Sample history of $x_1$')
    ax[0].xaxis.set_ticks_position('top')
    ax[0].set_ylabel('$x_1$')
    ax[1].plot(samples[1],linewidth=.5)
    ax[1].set_xlabel('Sample history of $x_2$')
    ax[1].set_ylabel('$x_2$')
    plt.show()
```

```
Case p = 0.5
Conditional mean estimate [0.0072173  0.00858172]
Marginal standard deviations: [0.9798433  0.98317472]
Case p = 0.9
Conditional mean estimate [-0.00563968 -0.0054493 ]
Marginal standard deviations: [0.96447317 0.96551602]
Case p = 0.99
Conditional mean estimate [0.02944871 0.02974477]
Marginal standard deviations: [0.86487827 0.86549707]
Case p = 0.999
Conditional mean estimate [-0.14924934 -0.14949796]
Marginal standard deviations: [0.81811924 0.81843321]
```


Samples when p = 0.5
Sample history of x_1
Sample history of x_2
x_1
x_2


Samples when p = 0.9
Sample history of x_1
Sample history of x_2
x_1
x_2


Samples when p = 0.99
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
0
2000
4000
6000
8000
10000
x_1
Sample history of x_1
x_2
Sample history of x_2



2. (a) Bayes' formula:

$$f(x_1, x_2|y) \propto \exp\left(-\frac{1}{2 \cdot 0.1^2}(y - (x_1^2 + x_2^2)^{1/2})^2 - \frac{1}{2}\big(|x_1 - 1| + (x_2 - 1)^2\big)\right).$$

(b) and (c)

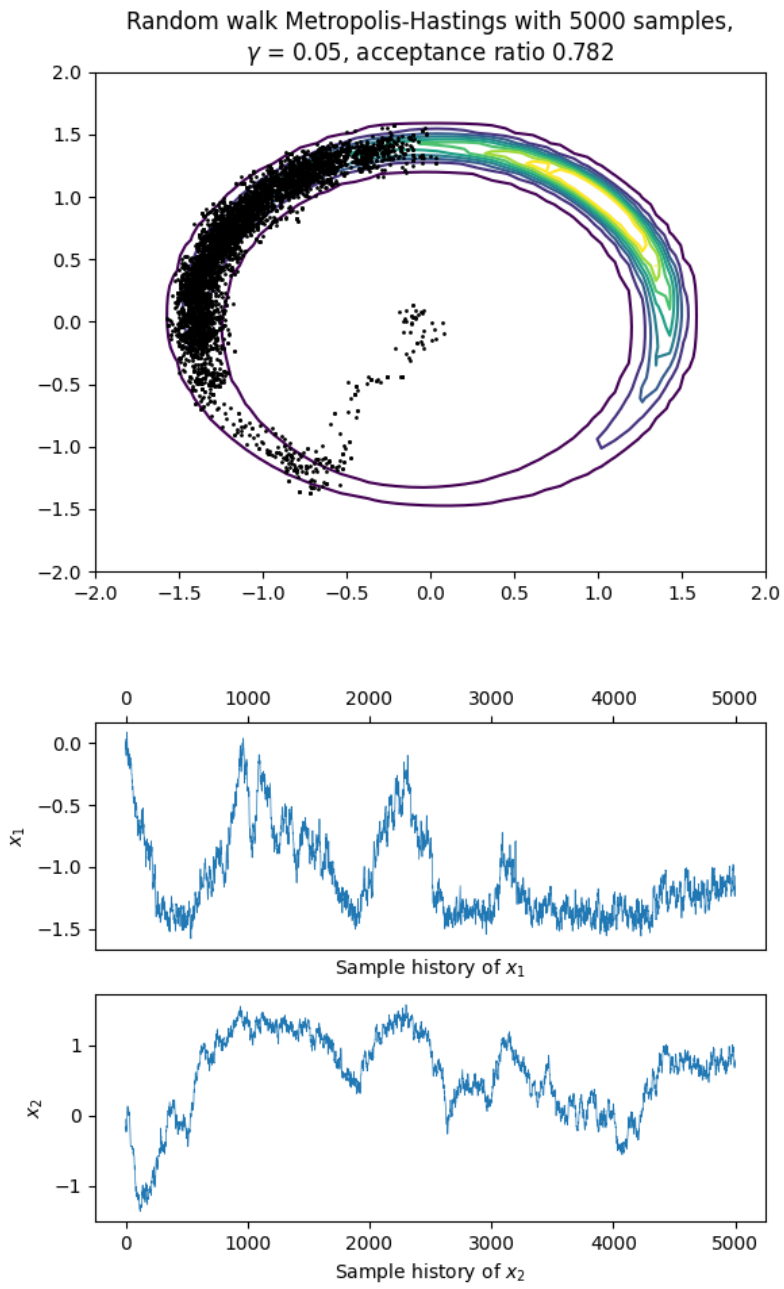
Solution using Python

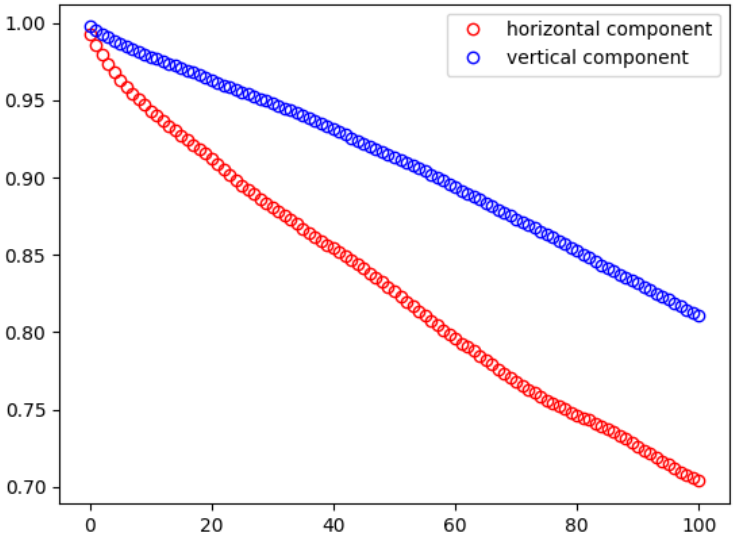
```
import numpy as np
import matplotlib.pyplot as plt

# In general, it is usually preferable to work with
```

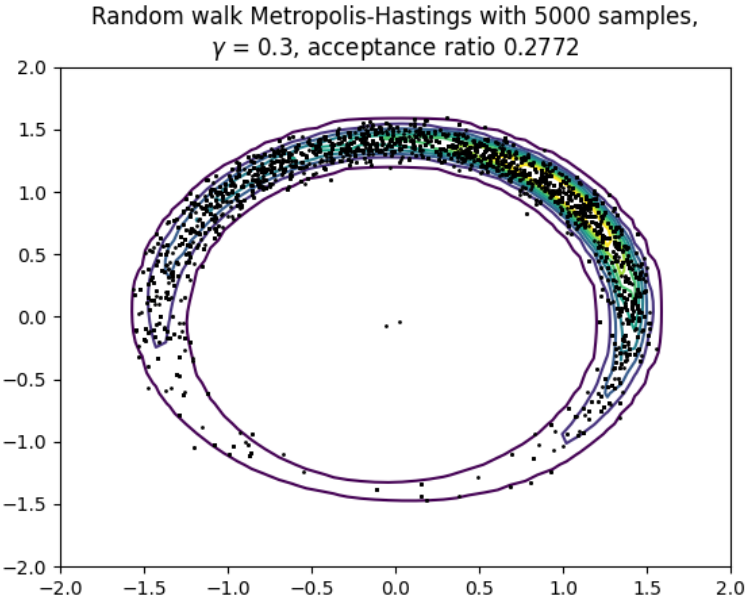
```
# the logarithm of the target density:
logp = lambda x,y: -1/2/.1**2*((x**2+y**2)**1/2 - 1)**2-1/2*(np.abs(x-1) + (y-1)**2)
x = np.array([0,0])
nsamples = 5000
samples = []
for gamma in [.05,0.3,1.5]:
    N_accepted = 0
    for _ in range(nsamples):
        step = gamma*np.random.normal(size=2)
        y = x + step
        # Logarithm of the acceptance probability alpha
        logalpha = logp(y[0],y[1]) - logp(x[0],x[1])
        t = np.random.uniform()
        if logalpha > np.log(t): # remember to account for the logarithm!
            samples.append(y)
            N_accepted = N_accepted+1
            x = y
        else:
            samples.append(x)
    # At this point, one would usually discard some number of
    # the initial samples (the burn-in period). Here, we omit
    # this step.
    samples = list(zip(*samples)) # transpose list
    print('Step size: ' + str(gamma) + ', CM estimator: '  \
    + str(np.mean(samples,axis=1)))
    # The true value of the CM estimator is approximately [0.392481, 0.676814].
    # For sample size 5000, the CM approximations are generally quite poor,
    # but gamma=0.3 and 1.5 seem to be generally the best. When the sample size
    # is increased, e.g., to 50000, then gamma=0.3 and 1.5 appear to yield somewhat
    # good approximations of the CM estimator.
    X,Y = np.meshgrid(np.linspace(-2,2),np.linspace(-2,2))
    plt.contour(X,Y,np.exp(logp(X,Y)),levels=np.arange(0.02,.82,0.1))
    plt.plot(samples[0],samples[1],'.',color='black',markersize=2)
    plt.title('Random walk Metropolis-Hastings with ' + str(nsamples) + \
    ' samples,\n$\gamma$ = ' + str(gamma) + ', acceptance ratio ' + \
    str(N_accepted/nsamples))
    plt.show()

    # Optional: assess the componentwise sample histories
    # to determine the quality of the sample
    fig,ax = plt.subplots(2,1)
    ax[0].plot(samples[0],linewidth=.5)
    ax[0].set_xlabel('Sample history of $x_1$')
    ax[0].xaxis.set_ticks_position('top')
    ax[0].set_ylabel('$x_1$')
    ax[1].plot(samples[1],linewidth=.5)
    ax[1].set_xlabel('Sample history of $x_2$')
```

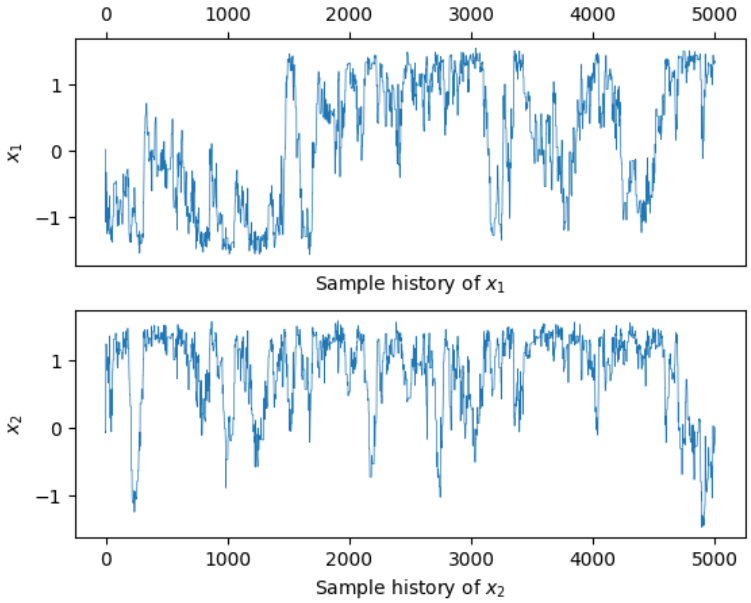
```
    ax[1].set_ylabel('$x_2$')
    plt.show()

    # Optional: assess the normalized autocovariance sequence
    # to determine the quality of the sample
    # Function to compute the autocovariance
    def autocovariance(f):
        N = len(f)
        gamma2 = np.zeros(N-1)
        f_c = f-np.mean(f)
        gamma2_0 = np.mean(f_c**2)
        for k in np.arange(N-1,0,-1):
            jj = np.arange(1,N-k+1)
            gamma2[k-1] = 1/(gamma2_0*(N-k)) * np.sum(f_c[jj-1]*f_c[jj+k-1])
        return gamma2

    ac1 = autocovariance(samples[0])
    ac2 = autocovariance(samples[1])
    N_ac = 100
    plt.plot(range(0,N_ac+1),ac1[0:N_ac+1],'o',markerfacecolor='none',
    markeredgecolor='red',label='horizontal component')
    plt.plot(range(0,N_ac+1),ac2[0:N_ac+1],'o',markerfacecolor='none',
    markeredgecolor='blue',label='vertical component')
    plt.legend()
    plt.show()
```

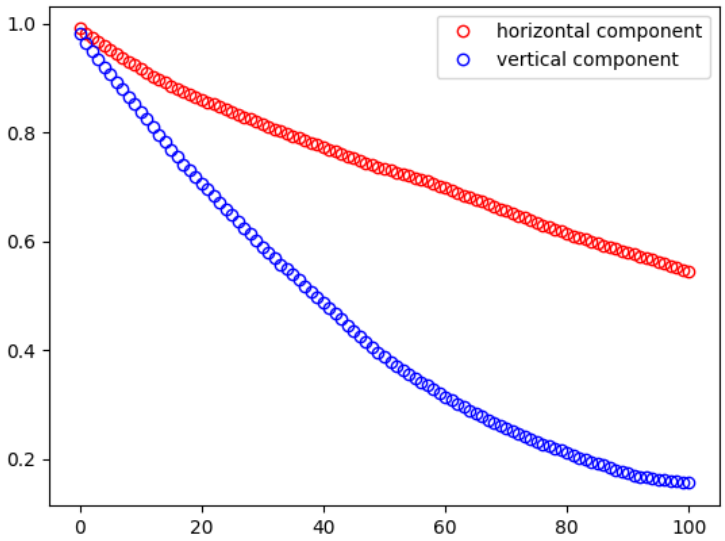
```
Step size: 0.05, CM estimator: [-0.00303354 -0.69142799]
Step size: 0.3, CM estimator: [0.36885628 0.75157444]
Step size: 1.5, CM estimator: [0.2893287  0.61195342]
```

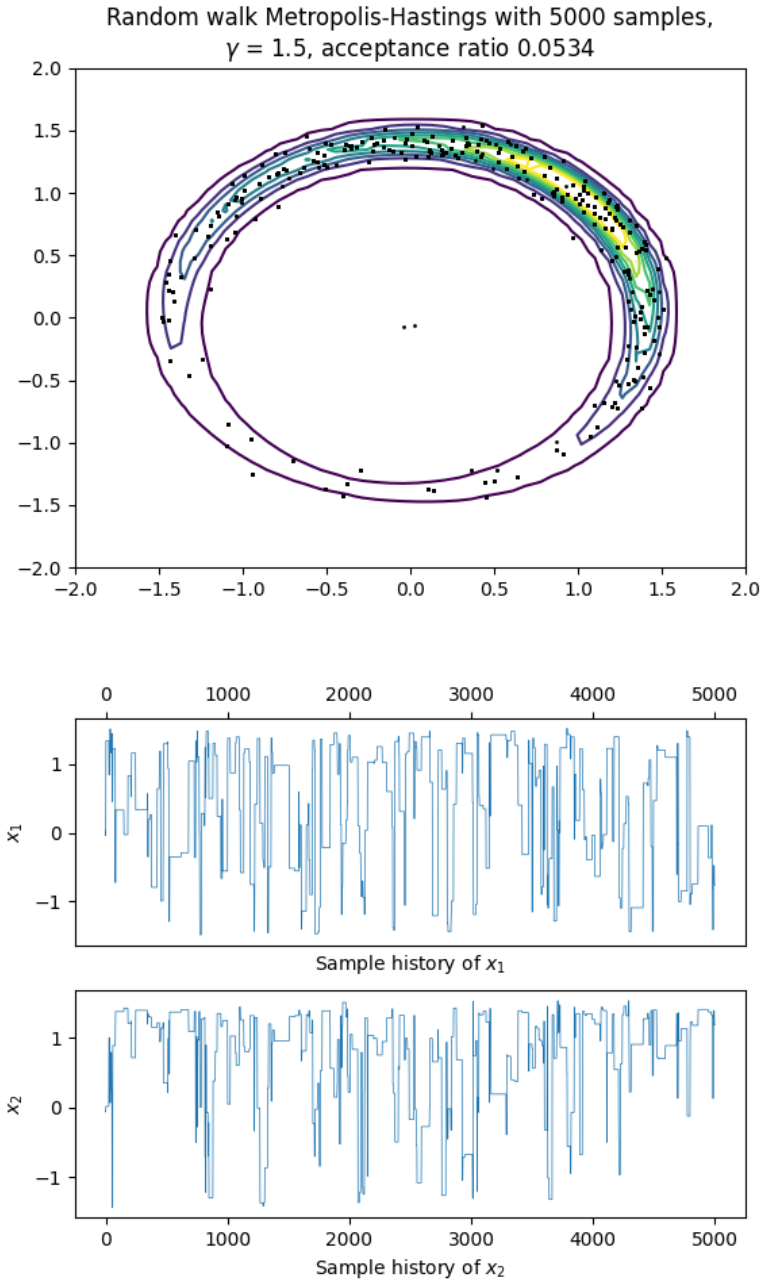

Random walk Metropolis-Hastings with 5000 samples,
γ = 0.05, acceptance ratio 0.782
Sample history of x_1
Sample history of x_2

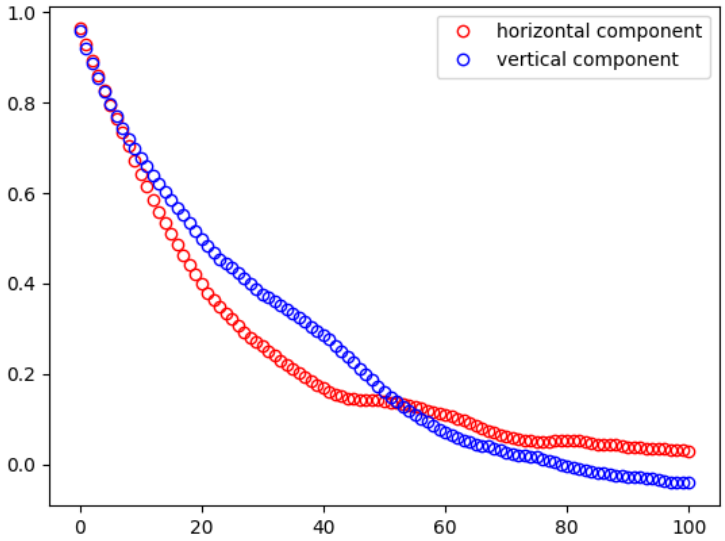

horizontal component
vertical component
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
20
40
60
80
100


Random walk Metropolis-Hastings with 5000 samples,
γ = 0.3, acceptance ratio 0.2772
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


0
1000
2000
3000
4000
5000
1
0
−1
x_1
Sample history of x_1
x_2
Sample history of x_2


horizontal component
vertical component
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100

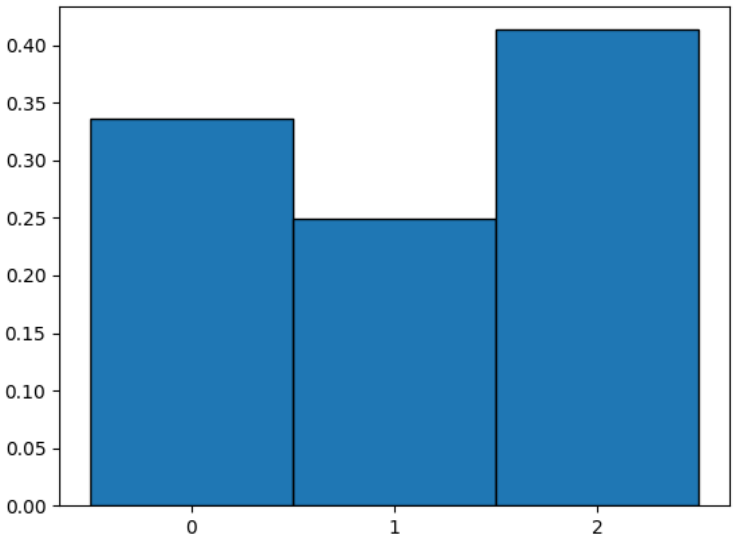

Random walk Metropolis-Hastings with 5000 samples,
γ = 1.5, acceptance ratio 0.0534
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
0
1000
2000
3000
4000
5000
x_1
1
0
−1
Sample history of x_1
x_2
1
0
−1
0
1000
2000
3000
4000
5000
Sample history of x_2



3.

Solution using Python

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

prob = [4/12,3/12,5/12]
sample = [0]

for _ in range(10000):
    current = sample[-1]
    proposal = np.random.choice(3)
    alpha = prob[proposal]/prob[current]
    t = np.random.uniform()
    if alpha > t:
        sample.append(proposal)
    else:
        sample.append(current)

plt.hist(sample,bins=[-0.5,.5,1.5,2.5],density=True,edgecolor='black',label='sample')
plt.xticks([0,1,2])
plt.show()
```



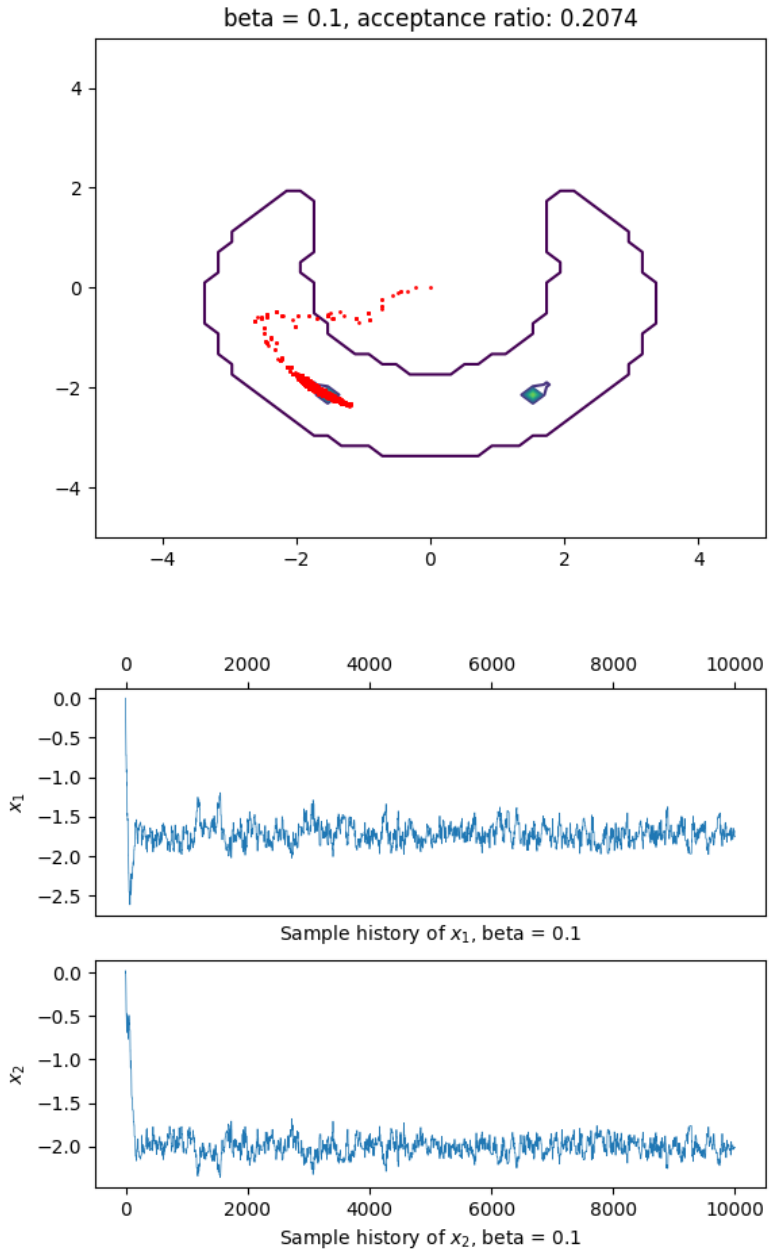
4.

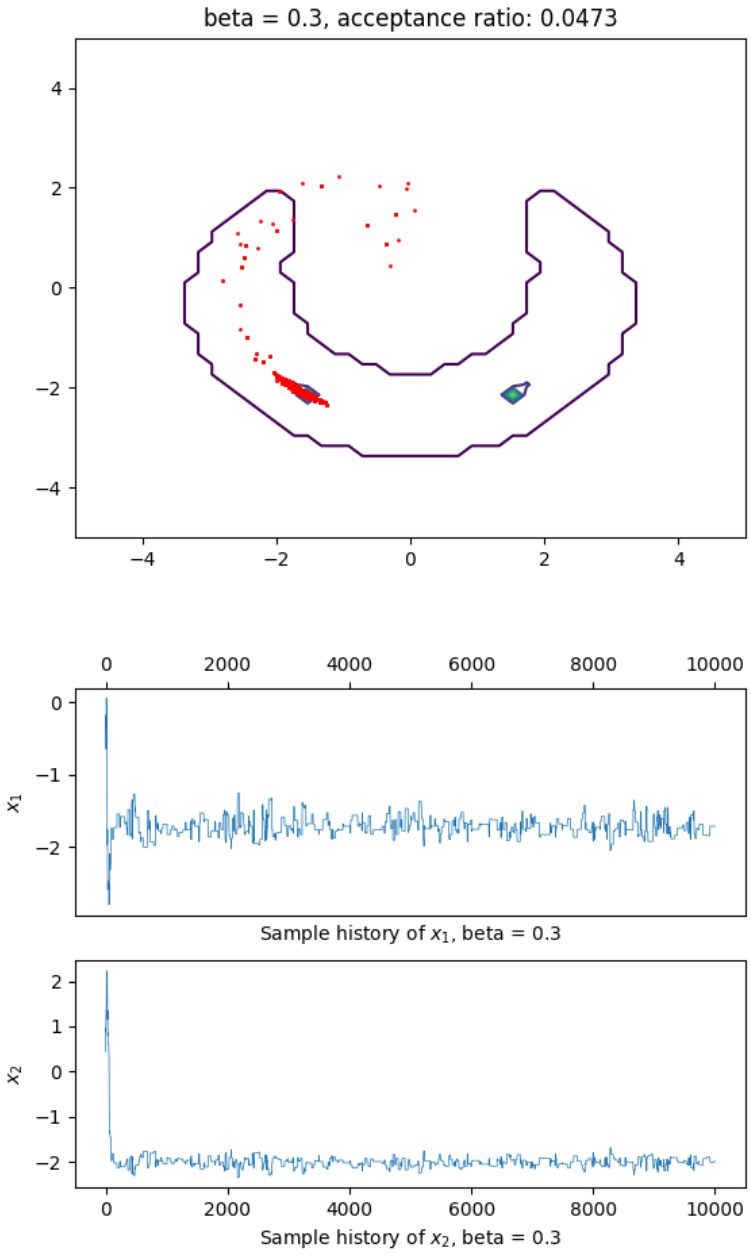
Solution using Python

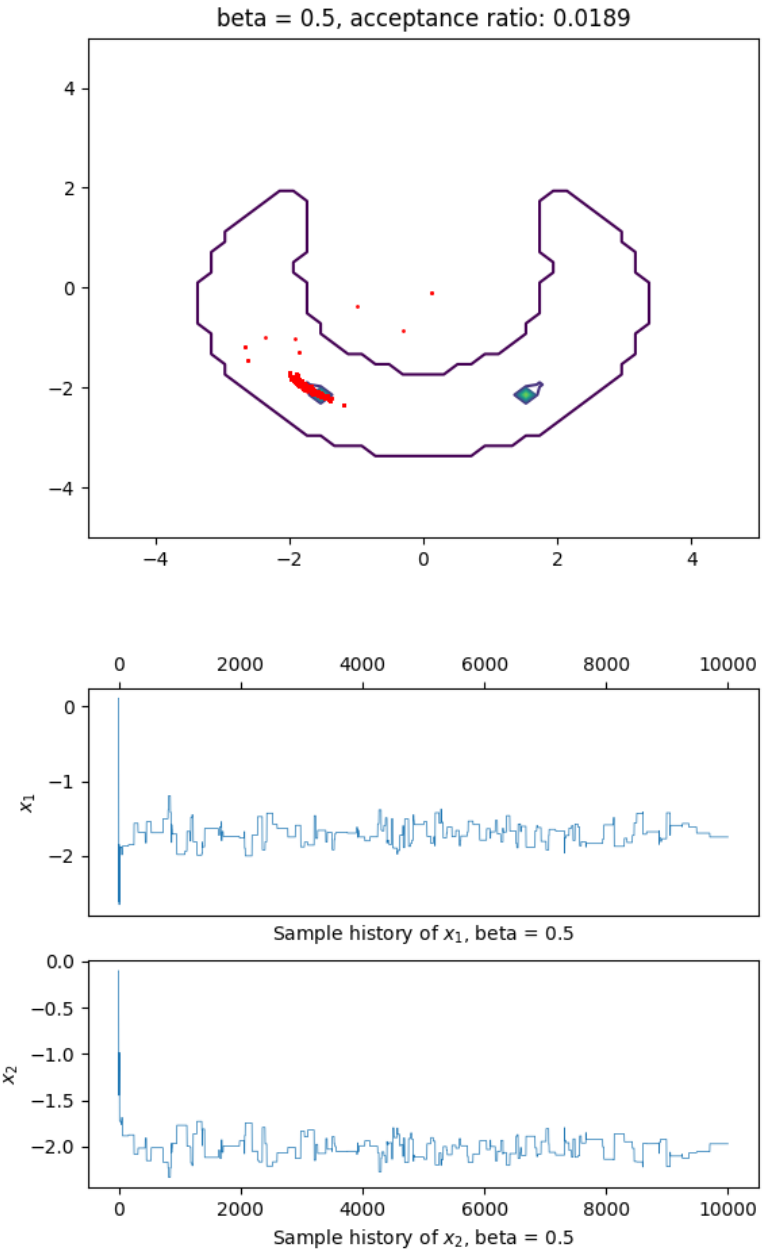
```
import numpy as np
import math
import matplotlib.pyplot as plt

y = np.array([7,-2])
delta = .1
logpost = lambda x1,x2: -1/2/delta**2 * \
                        ((y[0]-x1**2-x2**2)**2 + (y[1]-x2)**2) - (x1**2+x2**2)/2
nsamples = 10000
np.random.seed(321);
for beta in [.1,.3,.5,.7,.85,.9]:
    n_accept = 0
    x = np.array([0,0])
    samples = []
    logq = lambda old,new: -1/2/beta**2 * np.linalg.norm(new-np.sqrt(1-beta**2)*old)**2
    for _ in range(nsamples):
        new = np.sqrt(1-beta**2)*x + beta*np.random.normal(size=x.shape)
        if np.prod(np.abs(new)<=4):
            logalpha = logpost(new[0],new[1]) + logq(new,x) \
                       - logpost(x[0],x[1]) - logq(x,new)
        else:
            logalpha = -math.inf
        t = np.random.uniform()
        if logalpha > np.log(t):
            samples.append(new)
```

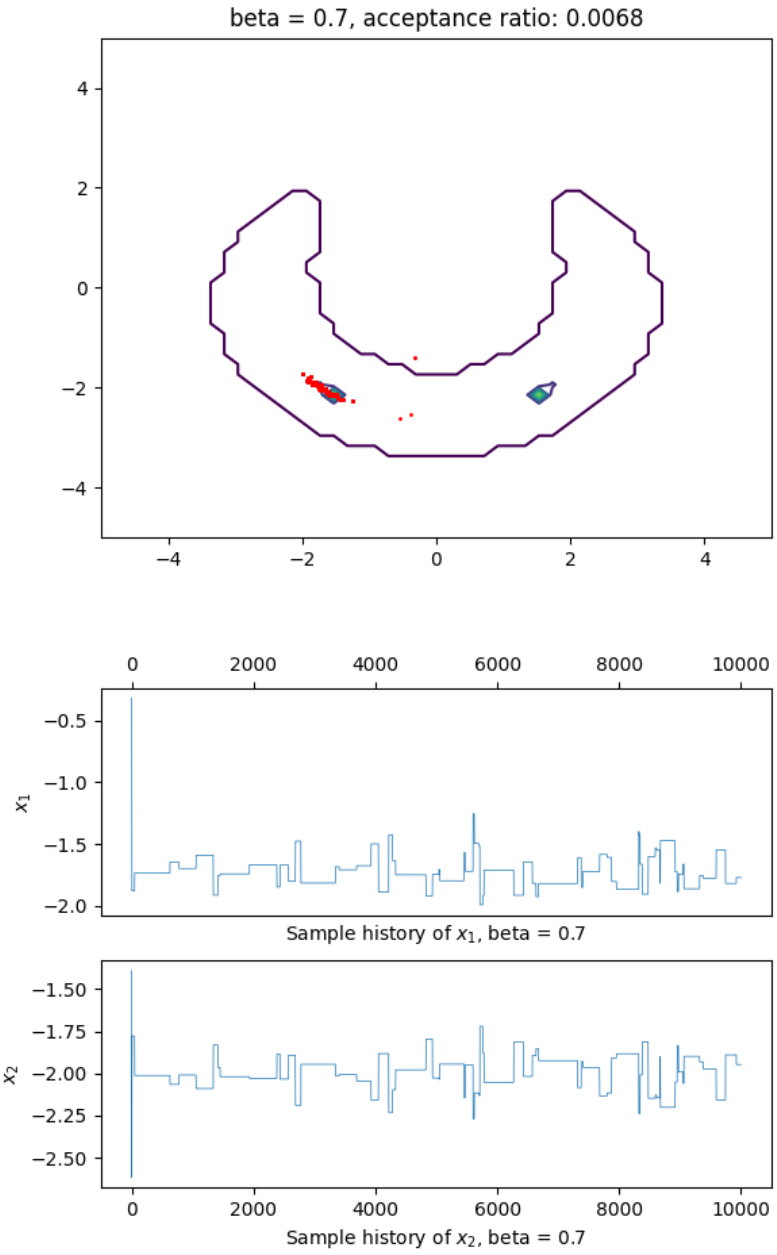
```
            x = new
            n_accept = n_accept+1
        else:
            samples.append(x)
    samples = list(zip(*samples)) # transpose list
    print('beta = ' + str(beta))
    print('Acceptance ratio: ' + str(n_accept/nsamples))
    print('CM estimate: ' + str(np.mean(samples,axis=1)))
    X,Y = np.meshgrid(np.linspace(-5,5),np.linspace(-5,5))
    post = (np.abs(X)<=4) * (np.abs(Y)<=4) * np.exp(logpost(X,Y))
    plt.contour(X,Y,post)
    plt.plot(samples[0],samples[1],'.',color='red',markersize=2)
    plt.title('beta = ' + str(beta) + ', acceptance ratio: ' + str(n_accept/nsamples))
    plt.show()
    fig,ax = plt.subplots(2,1)
    ax[0].plot(samples[0],linewidth=.5)
    ax[0].set_xlabel('Sample history of $x_1$, beta = ' + str(beta))
    ax[0].xaxis.set_ticks_position('top')
    ax[0].set_ylabel('$x_1$')
    ax[1].plot(samples[1],linewidth=.5)
    ax[1].set_xlabel('Sample history of $x_2$, beta = ' + str(beta))
    ax[1].set_ylabel('$x_2$')
    plt.show()
```

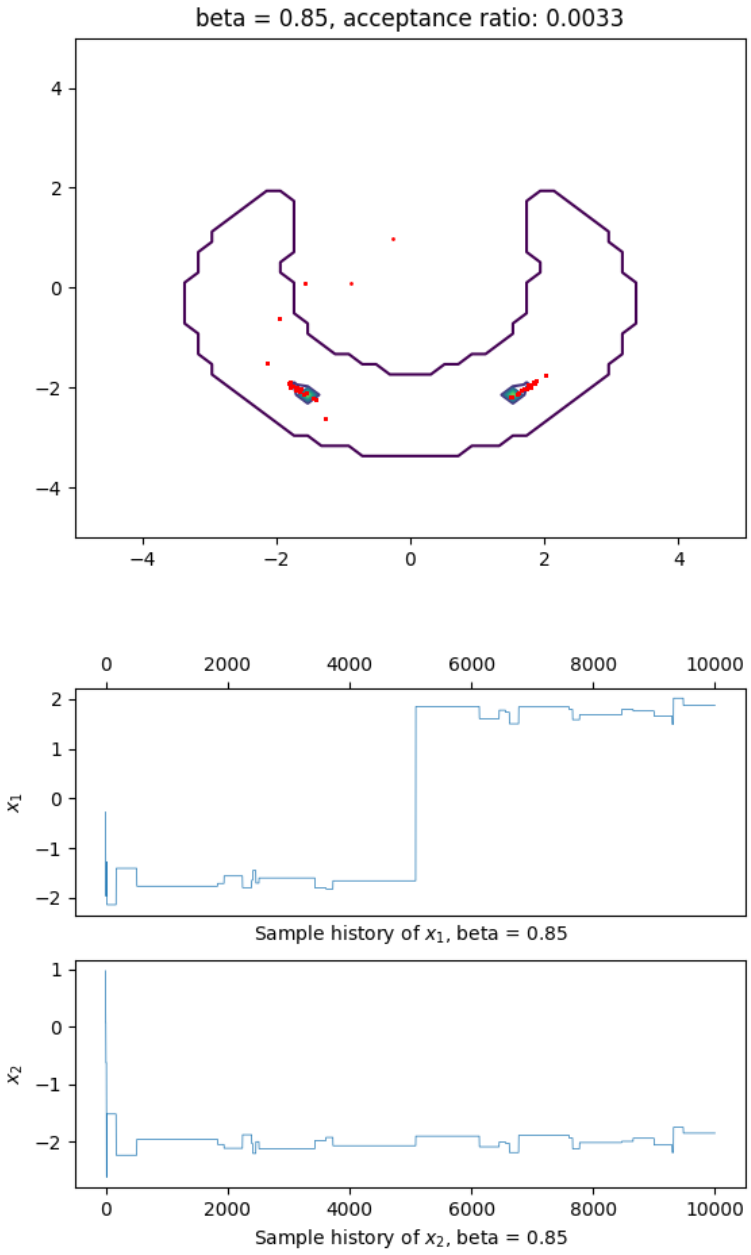
```
beta = 0.1
Acceptance ratio: 0.2074
CM estimate: [-1.71665233 -1.99309361]
beta = 0.3
Acceptance ratio: 0.0473
CM estimate: [-1.71372562 -1.98947477]
beta = 0.5
Acceptance ratio: 0.0189
CM estimate: [-1.70386687 -2.01274949]
beta = 0.7
Acceptance ratio: 0.0068
CM estimate: [-1.73266153 -1.99244125]
beta = 0.85
Acceptance ratio: 0.0033
CM estimate: [ 0.01952665 -1.99059837]
beta = 0.9
Acceptance ratio: 0.0024
CM estimate: [ 1.77597782 -1.94599581]
```

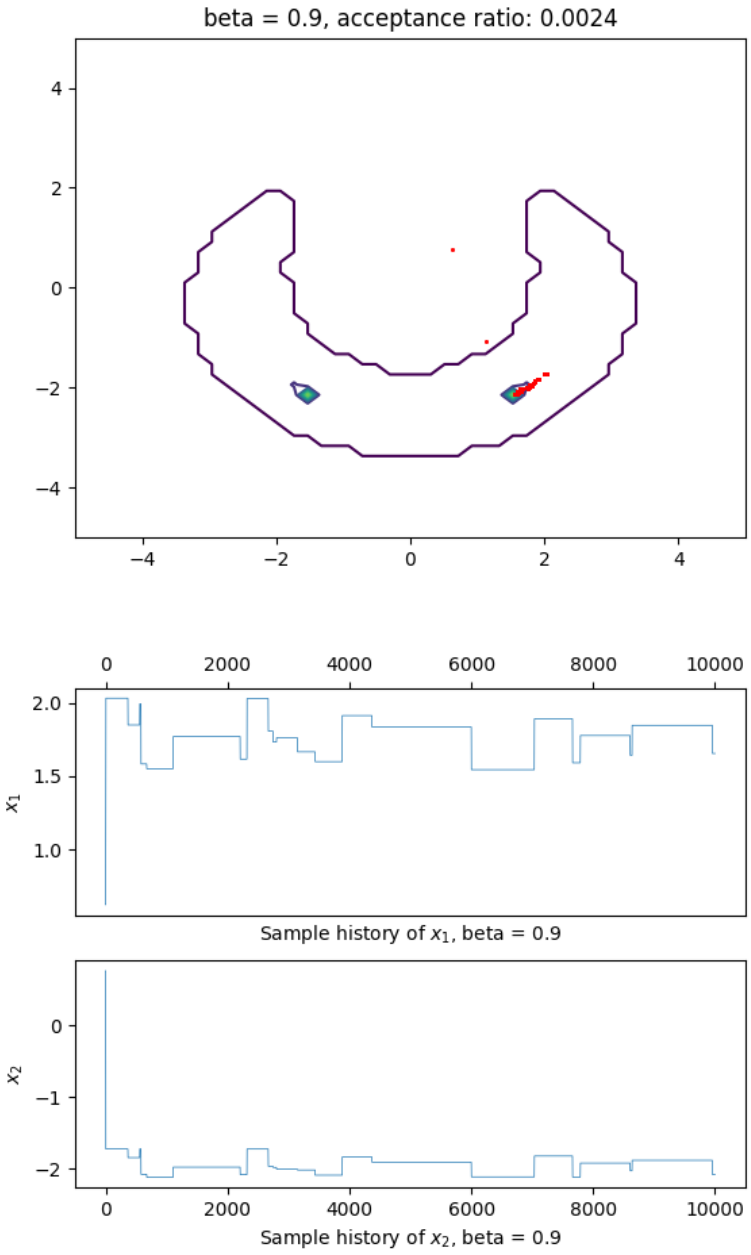

beta = 0.1, acceptance ratio: 0.2074
Sample history of x_1, beta = 0.1
Sample history of x_2, beta = 0.1


beta = 0.3, acceptance ratio: 0.0473
Sample history of x_1, beta = 0.3
Sample history of x_2, beta = 0.3


beta = 0.5, acceptance ratio: 0.0189
Sample history of x_1, beta = 0.5
Sample history of x_2, beta = 0.5


beta = 0.7, acceptance ratio: 0.0068
4
2
0
−2
−4
−4
−2
0
2
4
0
2000
4000
6000
8000
10000
−0.5
−1.0
−1.5
−2.0
x_1
Sample history of x_1, beta = 0.7
−1.50
−1.75
−2.00
−2.25
−2.50
x_2
0
2000
4000
6000
8000
10000
Sample history of x_2, beta = 0.7


beta = 0.85, acceptance ratio: 0.0033
Sample history of x_1, beta = 0.85
Sample history of x_2, beta = 0.85


beta = 0.9, acceptance ratio: 0.0024
4
2
0
−2
−4
−4
−2
0
2
4
0
2000
4000
6000
8000
10000
2.0
1.5
1.0
x_1
Sample history of x_1, beta = 0.9
0
−1
−2
x_2
0
2000
4000
6000
8000
10000
Sample history of x_2, beta = 0.9