

# Stat 202C - Project 2

## Problem 1

1.

Proof:

For  $t = 0$ :  $X_s^1 = 1 > X_s^2 = 0$  for any  $s$ ,  $\therefore X_\theta^1 s > X_\theta^2 s$

For  $t \geq 0$ : Suppose  $X_\theta^1 s > X_\theta^2 s$  at time  $t$ .

$$\therefore \sum_{i \in \partial s} I(1 = X_i^1) > \sum_{i \in \partial s} I(1 = X_i^2), \quad i.e., \pi(X_s^1 = 1 | X_\theta^1 s) > \pi(X_s^2 = 1 | X_\theta^2 s)$$

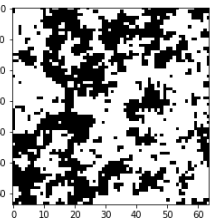
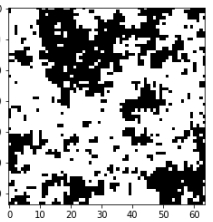
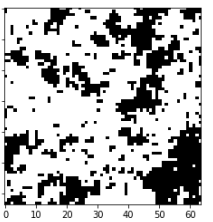
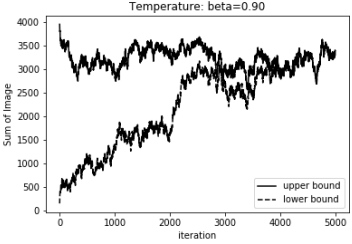
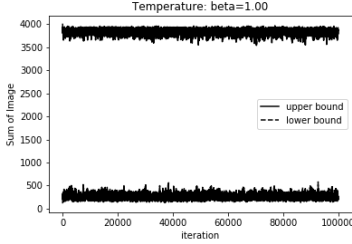
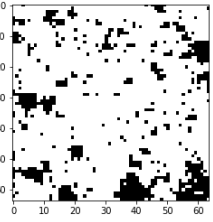
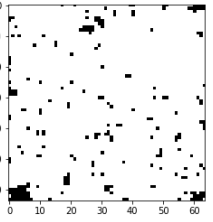
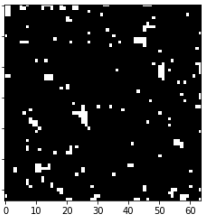
$\therefore$  the update share the same random number  $r$ , if  $r > \pi(X_s^1 = 1 | X_\theta^1 s)$ ,  $r > \pi(X_s^2 = 1 | X_\theta^2 s)$

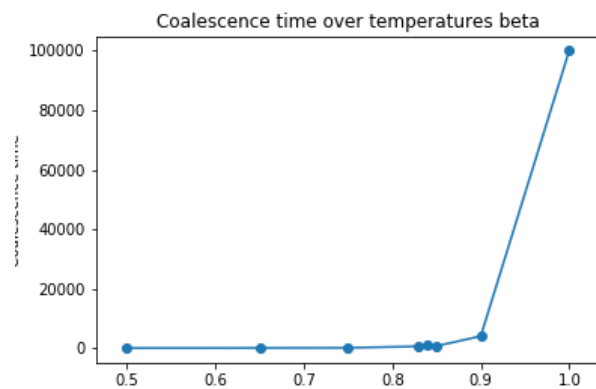
$\therefore$  if  $X_s^1 = 0$ ,  $X_s^2$  must also  $= 0$  at time  $(t+1)$ .

By induction the statement is proved.

2.

$\beta = 0.5, \tau = 18$	$\beta = 0.65, \tau = 42$	$\beta = 0.75, \tau = 90$
-	-	-
$\beta = 0.83, \tau = 572$	$\beta = 0.84, \tau = 896$	$\beta = 0.85, \tau = 687$
-	-	-

		
$\beta = 0.90, \tau = 4012$	$\beta = 1.00$	(not coalesce in $10^5$ iterations)
		
-	-	-
		

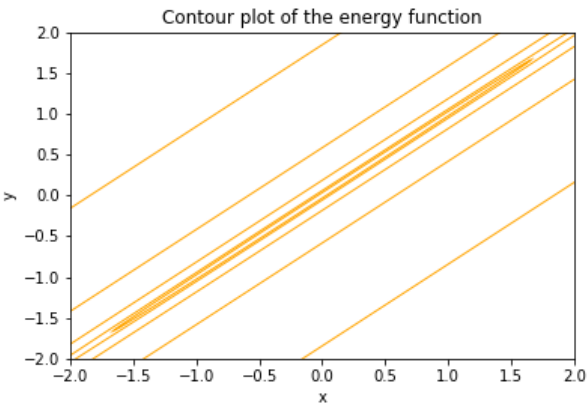


Problem 2

1.

a)

$$U(x,y) = \frac{1}{2}(x,y)^T \Phi^{-1}(x,y)$$



b)

$$\epsilon^* = \sqrt{\lambda(\Phi)_{min}} = 0.014142$$

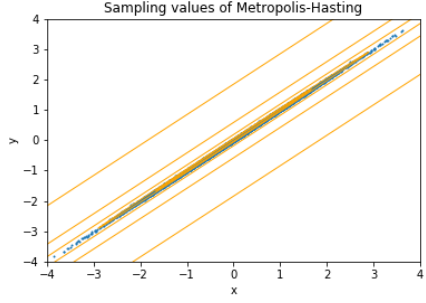
$$L^* = \frac{\sqrt{\lambda(\Phi)_{max}}}{\epsilon^*} = 100$$

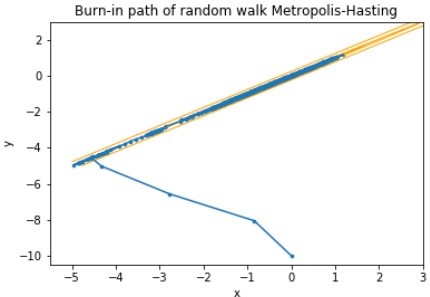
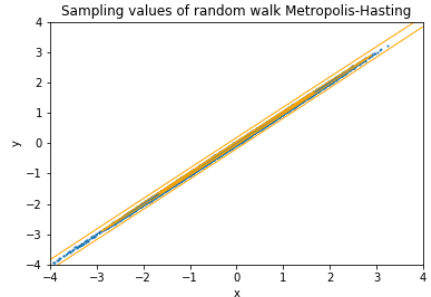
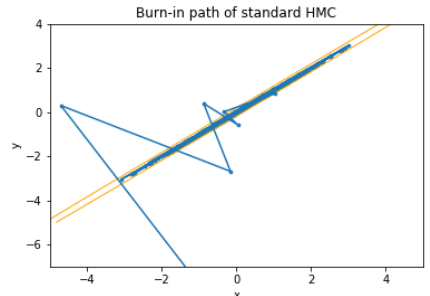
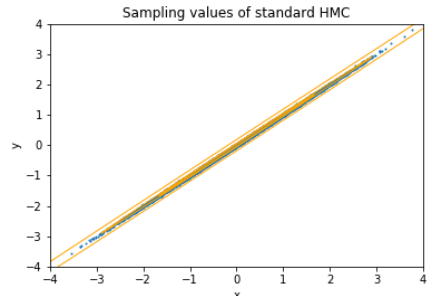
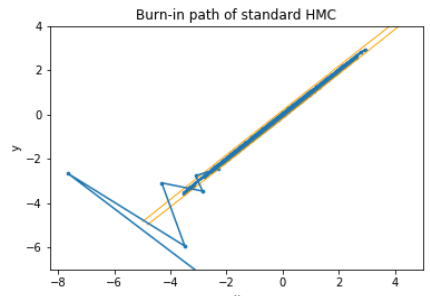
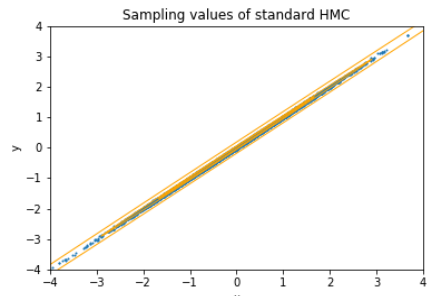
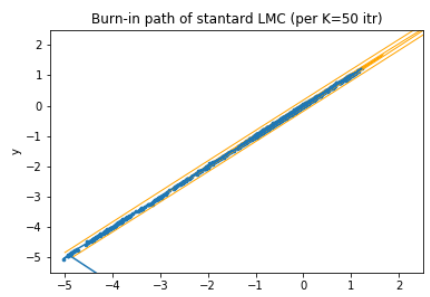
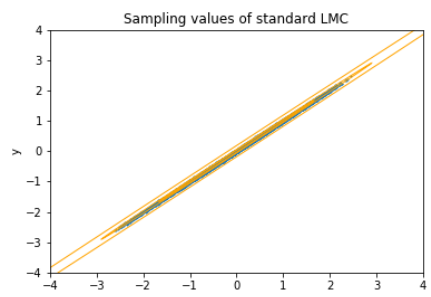
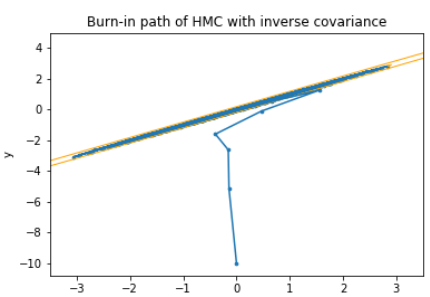
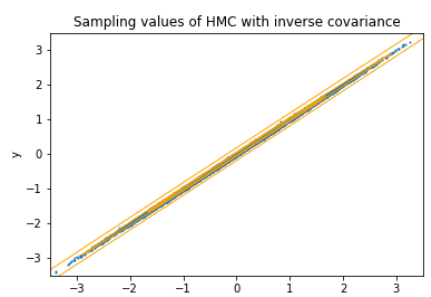
c)

$$\Sigma_{ideal} = \Sigma_{U(q)}^{-1} = \Phi^{-1} = \begin{bmatrix} 2500.25 & -2499.75 \\ -2499.75 & 2500.25 \end{bmatrix}$$

$$\epsilon^* = 1, L^* = 1$$

d)

Burn-in path	Sampling values	ESS
Direct Sampling		
	-	
Metropolis-hasting with Random Walk		

Burn-in path	Sampling values	ESS
		<p>ESS(x)=980, ESS(y)=981</p>
Standard HMC (L=100, ideal)		
		<p>ESS(x)=4334, ESS(y)=4338</p>
Standard HMC (L=50)		
		<p>ESS(x)=2850, ESS(y)=2863</p>
Standard LMC		
		<p>ESS(x)=392, ESS(y)=396</p>
RMHMC (equivalent to LMC with L=1)		
		<p>ESS(x)=1954, ESS(y)=1986</p>

Metropolis-Hasting has a higher dependency on the initial value, the burn-in path first reaches the lower left part of the distribution due to a too small initialization of  $y$  (-10) and fail to explore the whole distribution with in 1000 iterations. It can also be observed that the sample is biased to the lower left and the ess is relatively smaller.

Standard HMC with ideal leapfrog steps (100) has the highest ESS, and the sample seem to ditribute evenly over the target contour. The drawback would be that 100 steps is relatively more time consuming than other methods.

With a smaller leapfrog steps (50), the result is also satifactory, with smaller jumps at the first several iterations and higher efficiency, although the ESS shows that the sample is not as good as with the ideal leapfrog steps.

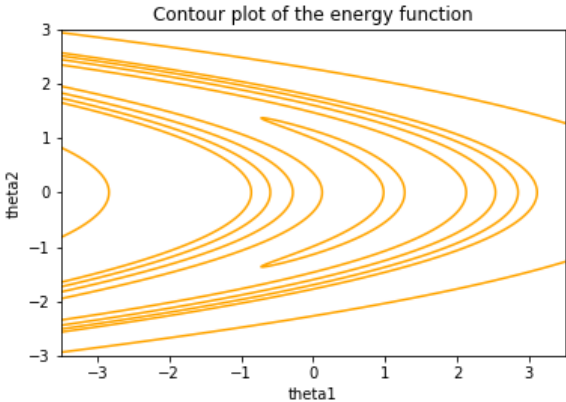
LMC converge slower in comparison, although faster than MH. The resulted sample is more concentrated with the lowest ESS. The reason may be that it can not go far enough with only one leapfrog step while the distribution is in a narrow shape so the stepsize must be small.

With the ideal  $\Sigma$ , we are able to get a good result with fewer sampling steps ( $L=1$ ), because in this method is analogous to sampling from standard normal to a target with identity variance-covariance.

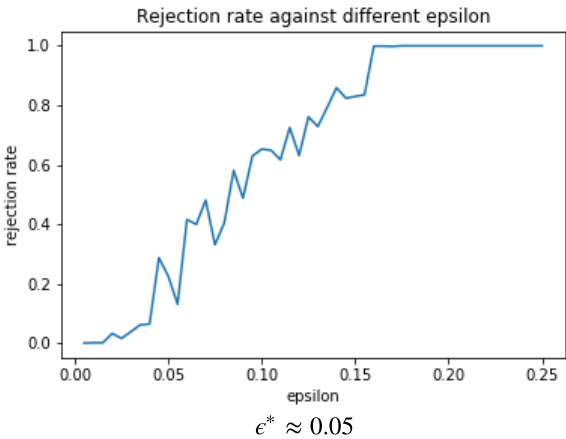
2.

a)

$$U(\theta|Y) = -\log \frac{P(Y|\theta)P(\theta)}{P(Y)} + const. = \sum_{i=1}^{100} \frac{(Y-(\theta_1+\theta_2^2))^2}{4} + \frac{\theta_1^2+\theta_2^2}{2}$$

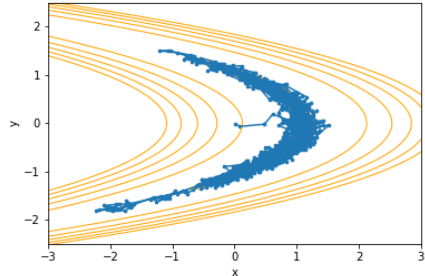
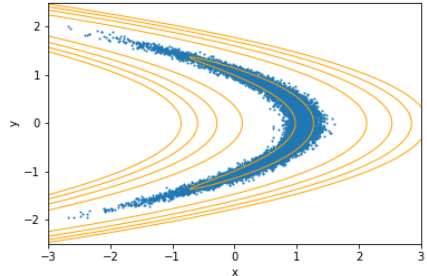
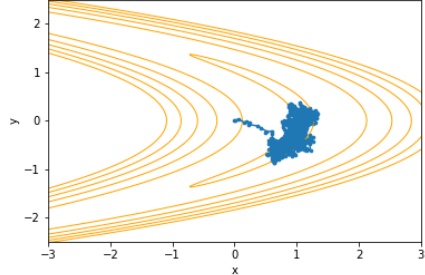
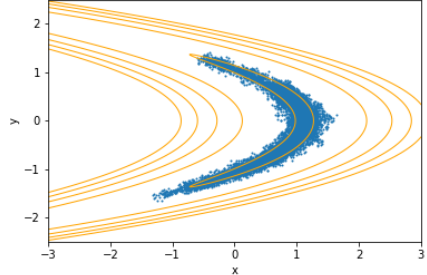
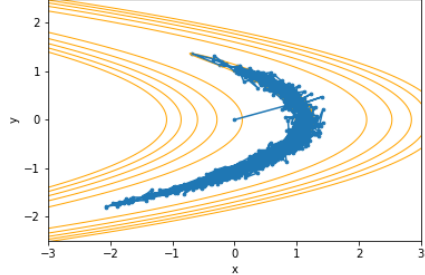
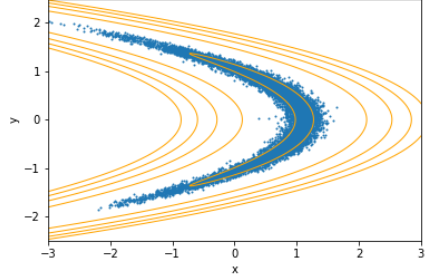
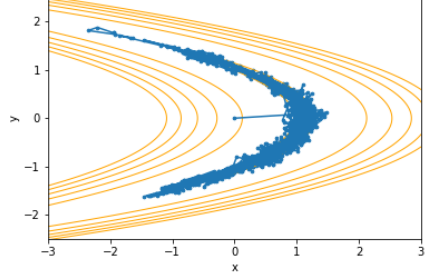
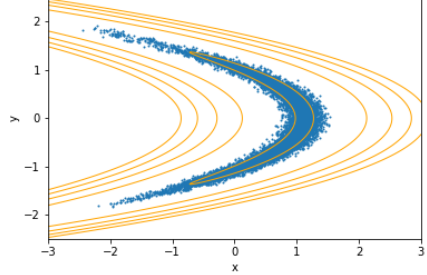
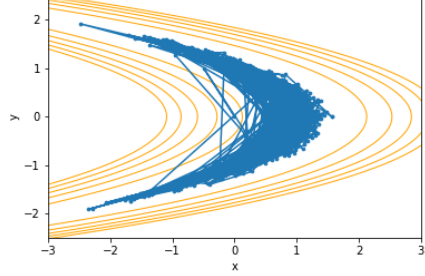
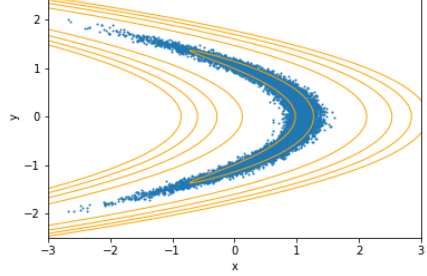


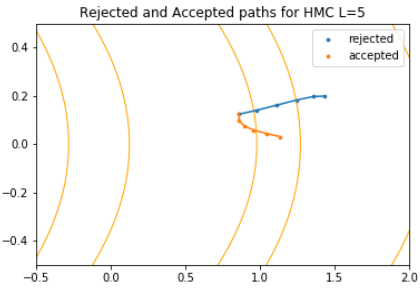
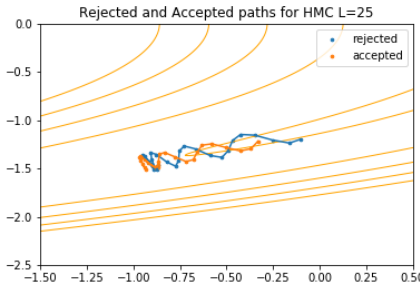
b)



c)

Burn-in path	Sampling values	ESS
Metropolis-hasting with Random Walk		

Burn-in path	Sampling values	ESS
<p>Burn-in path of random walk Metropolis-Hasting for banana distribution</p> 	<p>Sampling values of random walk Metropolis-Hasting for banana distribut</p> 	<p>ESS(x)=17, ESS(y)=49</p>
<p>Standard LMC (K=1)</p>		
<p>Burn-in path of LMC (K=1) for banana distribution</p> 	<p>Sampling values of LMC (K=1) for banana distribution</p> 	<p>ESS(x)=21, ESS(y)=13</p>
<p>Standard LMC (K=25)</p>		
<p>Burn-in path of LMC (K=25) for banana distribution</p> 	<p>Sampling values of LMC (K=25) for banana distribution</p> 	<p>ESS(x)=83, ESS(y)=36</p>
<p>Standard HMC (L=5)</p>		
<p>Burn-in path of HMC (L=5) for banana distribution</p> 	<p>Sampling values of HMC (L=5) for banana distribution</p> 	<p>ESS(x)=128, ESS(y)=98</p>
<p>Standard HMC (L=25)</p>		
<p>Burn-in path of HMC (L=25) for banana distribution</p> 	<p>Sampling values of HMC (L=25) for banana distribution</p> 	<p>ESS(x)=5043, ESS(y)=295</p>
<p>Accepted and rejected paths</p>		

Burn-in path	Sampling values	ESS
		

Metropolis-hasting for banana distribution get a low ESS and the resulted sample is a bit biased, as it tends to stuck in a local region.

LMC with fewer sampling steps also gives a relatively poor result. The convergence is slow, as it fails to explor the whole distribution in 1000 burn-in steps.

With more sampling steps (K=25), the result is significantly better. However, it still has the problem of sticking in a local region.

HMC with more than 1 leapfrog steps gives significantly better result. With L=5, the sample is closer to the original and 1000 burn-in steps manages to explore the whole high-energy region.

A longer leapfrog step (L=25) gives the highest ESS. It is able to make big jumps in this distribution with irregular energy landscape as shown in the burn-in plot, because it can walk longer in the same direction from the current point.

The plots of rejected and accepted paths also shows that, longer leapfrog step walks further from the current point, and the accepted path has a lower enegy (higher probability) in the target distribution, so it is more likely to be accepted.

## Problem 1

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
In [ ]: def margin(x, y, mat, beta):
    one = 0
    zero = 0
    if x > 0:
        one += mat[x-1, y]
        zero += 1 - mat[x-1, y]
    if x < n-1:
        one += mat[x+1, y]
        zero += 1 - mat[x+1, y]
    if y > 0:
        one += mat[x, y-1]
        zero += 1 - mat[x, y-1]
    if y < n-1:
        one += mat[x, y+1]
        zero += 1 - mat[x, y+1]
    c = np.exp(beta*one)
    prob = c/(c+np.exp(beta*zero))
    return prob
def swp(white, black, beta):
    for x in range(n):
        for y in range(n):
            rdn = np.random.uniform(0,1)
            if rdn < margin(x, y, white, beta):
                white[x, y] = 1
            else:
                white[x, y] = 0
            if rdn < margin(x, y, black, beta):
                black[x, y] = 1
            else:
                black[x, y] = 0
```

```
In [ ]: n = 64
betas = [0.5, 0.65, 0.75, 0.83, 0.84, 0.85, 0.9, 1.0]
# tau = []
# cdict = np.linspace(0,1,num=8)
for b in range(7,8):
    beta = betas[b]
    wh = []
    bl = []
    itr = 0
    white = np.ones((n,n))
    black = np.zeros((n,n))
    np.random.seed(7)
    for t in range(100000):
        swp(white, black, beta)
        ww, bb = np.sum(white), np.sum(black)
        wh.append(ww)
        bl.append(bb)
        if ww != bb:
            itr += 1
    plt.figure(2)
    plt.title("Temperature: beta=%.2f"%beta)
    plt.xlabel('iteration')
    plt.ylabel('Sum of Image')
    plt.plot(wh, c='k', label='upper bound')
    plt.plot(bl, c='k', linestyle='dashed', label='lower bound')
    plt.legend()
    plt.savefig("1-1(%s).png"%beta)
    plt.close()
    plt.imshow(white, cmap='gray')
    plt.savefig("1-1-%s.png"%beta)
    print(itr)
    tau.append(itr)
```

```
In [ ]: print(tau)
plt.imshow(black, cmap='gray')
plt.savefig("1-1-b.png")
```

```
In [ ]: plt.title('Coalescence time over temperatures beta')
plt.xlabel('Temperature: beta')
plt.ylabel('Coalescence time')
plt.scatter(betas, tau)
plt.plot(betas, tau)
plt.savefig("1-2.png")
```

## Problem 2

1.

```
In [ ]: ## a)
def energy_target(x, inv):
    x = np.array(x)
    return 0.5 * np.dot(np.dot(x.T, inv), x)
x1 = np.linspace(-5, 5, 1000)
x2 = np.linspace(-5, 5, 1000)
x11, x22 = np.meshgrid(x1, x2)
x = np.concatenate((x11.reshape(1000**2,1), x22.reshape(1000**2,1)), axis=1)
phi = np.array([[1,0.9998],[0.9998,1]])
inv = np.linalg.inv(phi)
e = []
for i in range(1000**2):
    xi = x[i, ]
    e.append(energy_target(xi, inv))
```



```
In [ ]: plt.title('Contour plot of the energy function')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-2,2)
plt.ylim(-2,2)
eig = np.sqrt(np.linalg.eigvals(phi)[0])
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig, 300*eig, 3000*eig], colors='orange', linewidths=1)
plt.savefig("2-1.png")
```

```
In [ ]: ## b)
epsI = np.sqrt(np.linalg.eigvals(phi)[1])
LI = np.sqrt(np.linalg.eigvals(phi)[0])/epsI
print(epsI,LI)
```

### **Direct sampling**

```
In [ ]: mean = [0, 0]
x,y = np.random.multivariate_normal(mean, phi, int(1e4)).T
plt.title('Sampling values of Metropolis-Hasting')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-4,4)
plt.ylim(-4,4)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig, 300*eig, 3000*eig], colors='orange', linewidths=1)
plt.scatter(x, y, s=1)
plt.savefig("2-1-0.png")
```

```
In [ ]: mu_x = 0
mu_y = 0
var_x = 1
var_y = 1
```

### **Metropolis Hasting**

```

In [ ]: def proposal():
        y = np.random.normal(0, eps)
        return y
    def enegy_prop(prop, eps):
        p = np.copy(prop)
        inv = np.array([[1/eps**2, 0], [0, 1/eps**2]])
        return 0.5 * np.dot(np.dot(p.T, inv), p)
    def MH_step(x):
        prop = [proposal() for xt in x]
        alpha = np.exp(energy_target(x, inv)-energy_target(prop, inv)+enegy_prop(prop, eps)-enegy_prop(x,
eps))
        # print(alpha)
        # print(energy_target(x, inv),energy_target(prop, inv),enegy_prop(prop, eps),enegy_prop(x, eps))
        if alpha >= 1:
            new = prop
        else:
            u = np.random.uniform(0,1)
            if u <= alpha:
                new = prop
            else:
                new = x
        return new
    def MH_sampling(init, num_burn=int(1e3), num_sample=int(1e4)):
        x = init
        burns = [init]
        for tb in range(num_burn):
            new = MH_step(x)
            burns.append(new)
            x = new
        samples = []
        for ts in range(num_sample):
            new = MH_step(x)
            samples.append(new)
            x = new
        return burns, samples

```

```

In [ ]: K = int(round(LI))
        eps = epsI
        init = [0, -10]
        bs, ss = MH_sampling(init, num_burn=int(1e3*K), num_sample=int(1e4*K))
        burns = []
        samples = []
        for i in range(int(1e3)):
            burns.append(bs[int(i*K)])
        for i in range(int(1e4)):
            samples.append(ss[int(i*K)])
        xb, yb = [c[0] for c in burns], [c[1] for c in burns]
        xs, ys = [c[0] for c in samples], [c[1] for c in samples]

```

```

In [ ]: plt.title('Burn-in path of Metropolis-Hasting')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig, 300*eig, 3000*eig], colors='o
range', linewidths=1)
        plt.scatter(xb, yb, s=7)
        plt.plot(xb, yb)
        plt.savefig("2-1-1.png")

```

```

In [ ]: plt.title('Sampling values of Metropolis-Hasting')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig, 300*eig, 3000*eig], colors='o
range', linewidths=1)
        plt.scatter(xs, ys, s=1)
        plt.savefig("2-1-2.png")

```

```

In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[: (k+1)])
    # mu_y = np.mean(y[: (k+1)])
    # var_x = np.var(x[: (k+1)])
    # var_y = np.var(y[: (k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)

```

### Metropolis Hasting - random walk

```

In [ ]: def energy_target(x, inv):
    x = np.array(x)
    return 0.5 * np.dot(np.dot(x.T, inv), x)
def MH_step_rw(x):
    prop = [xt+proposal() for xt in x]
    alpha = np.exp(energy_target(x, inv)-energy_target(prop, inv))
    # print(alpha)
    if alpha >= 1:
        new = prop
    else:
        u = np.random.uniform(0,1)
        if u <= alpha:
            new = prop
        else:
            new = x
    return new
def MH_sampling_rw(init, num_burn=int(1e3), num_sample=int(1e4)):
    x = init
    burns = []
    for tb in range(num_burn):
        new = MH_step_rw(x)
        burns.append(new)
        x = new
    samples = []
    for ts in range(num_sample):
        new = MH_step_rw(x)
        samples.append(new)
        x = new
    return burns, samples

```

```

In [ ]: init = [0, -10]
K = int(round(LI))
bs, ss = MH_sampling_rw(init, num_burn=int(1e3*K), num_sample=int(1e4*K))
burns = []
samples = []
for i in range(int(1e3)):
    burns.append(bs[int(i*K)])
for i in range(int(1e4)):
    samples.append(ss[int(i*K)])
xb, yb = [c[0] for c in burns], [c[1] for c in burns]
xs, ys = [c[0] for c in samples], [c[1] for c in samples]

```

```
In [ ]: plt.title('Burn-in path of random walk Metropolis-Hasting')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-5.5, 3)
plt.ylim(-10.5, 3)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 50*eig], colors='orange', linewidths=1)
plt.scatter(xb, yb, s=7)
plt.plot(xb, yb)
plt.savefig("2-2-1.png")
```

```
In [ ]: plt.title('Sampling values of random walk Metropolis-Hasting')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-4, 4)
plt.ylim(-4, 4)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig], colors='orange', linewidths=1)
plt.scatter(xs, ys, s=1)
plt.savefig("2-2-2.png")
```

```
In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[: (k+1)])
    # mu_y = np.mean(y[: (k+1)])
    # var_x = np.var(x[: (k+1)])
    # var_y = np.var(y[: (k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)
```

**Standard HMC**

```

In [ ]: def energy_target(x, inv):
        x = np.array(x)
        return 0.5 * np.dot(np.dot(x.T, inv), x)
    def deriv_energy(q, inv):
        q = np.array(q)
        return np.dot(q.T, inv)
    def energy_prop(p, sig_inv):
        p = np.array(p)
        return 0.5 * np.dot(np.dot(p.T, sig_inv), p)
    def momentum(mean, sig):
        p = np.random.multivariate_normal(mean, sig)
        return p
    def HMC_step(p, q, eps, L, sig_inv):
        # proposal
        p_half = p - 0.5*eps*deriv_energy(q, inv)
        ql = np.copy(q)
        for l in range(L-1):
            #print(type(ql[0]),type(eps*np.dot(sig, p_half)))
            ql += eps*np.dot(sig_inv, p_half)
            p_half -= eps*deriv_energy(ql, inv)
            qL = ql + eps*np.dot(sig_inv, p_half)
            pL = p_half - 0.5*eps*deriv_energy(qL, inv)
            ene = energy_target(q,inv)-energy_target(qL,inv) \
                  +energy_prop(p, sig_inv)-energy_prop(pL, sig_inv)
            alpha = np.exp(ene)
            # print(alpha)
            if alpha >= 1:
                new = qL
            else:
                u = np.random.uniform(0,1)
                if u <= alpha:
                    new = qL
                else:
                    new = q
            return new
    def HMC_sampling(init, eps, L, mean, sig, inv, num_burn=int(1e3), num_sample=int(1e4)):
        q = np.array(init, dtype=float)
        sig_inv = np.linalg.inv(sig)
        burns = [init]
        for tb in range(num_burn):
            p = momentum(mean, sig)
            new = HMC_step(p, q, eps, L, sig_inv)
            burns.append(new)
            q = new
        samples = []
        for ts in range(num_sample):
            p = momentum(mean, sig)
            new = HMC_step(p, q, eps, L, sig_inv)
            samples.append(new)
            q = new
        return burns, samples

```

```

In [ ]: init = [0, -10]
        mean = [0, 0]
        sig = np.array([[1,0],[0,1]])
        L = int(round(LI))
        eps = epsI
        burns, samples = HMC_sampling(init, eps, L, mean, sig, inv)

```

```

In [ ]: xb, yb = [c[0] for c in burns], [c[1] for c in burns]
        xs, ys = [c[0] for c in samples], [c[1] for c in samples]

```

```

In [ ]: plt.title('Burn-in path of standard HMC')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.scatter(xb, yb, s=7)
        plt.ylim(-7,4)
        plt.plot(xb, yb)
        plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig], colors='orange', linewidths=
1)
        plt.savefig("2-1-1.png")

```

```
In [ ]: plt.title('Sampling values of standard HMC')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-4,4)
plt.ylim(-4,4)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig], colors='orange', linewidths=1)
plt.scatter(xs, ys, s=1)
plt.savefig("2-1-2.png")
```

```
In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[: (k+1)])
    # mu_y = np.mean(y[: (k+1)])
    # var_x = np.var(x[: (k+1)])
    # var_y = np.var(y[: (k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)
```

#### **smaller L**

```
In [ ]: init = [0, -10]
mean = [0, 0]
sig = np.array([[1,0],[0,1]])
L = int(round(LI)/2)
eps = epsI
burns, samples = HMC_sampling(init, eps, L, mean, sig, inv)
```

```
In [ ]: xb, yb = [c[0] for c in burns], [c[1] for c in burns]
xs, ys = [c[0] for c in samples], [c[1] for c in samples]
```

```
In [ ]: plt.title('Burn-in path of standard HMC')
plt.xlabel('x')
plt.ylabel('y')
plt.scatter(xb, yb, s=7)
plt.ylim(-7,4)
plt.plot(xb, yb)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig], colors='orange', linewidths=1)
plt.savefig("2-3-1.png")
```

```
In [ ]: plt.title('Sampling values of standard HMC')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-4,4)
plt.ylim(-4,4)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig], colors='orange', linewidths=1)
plt.scatter(xs, ys, s=1)
plt.savefig("2-3-2.png")
```

```
In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[: (k+1)])
    # mu_y = np.mean(y[: (k+1)])
    # var_x = np.var(x[: (k+1)])
    # var_y = np.var(y[: (k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)
```

### Standard LMC

```
In [ ]: mean = [0, 0]
sig = np.array([[1,0],[0,1]])
L = 1
K = int(round(LI))
eps = epsI
bs, ss = HMC_sampling(init, eps, L, mean, sig, inv, num_burn=int(1e3*K), num_sample=int(1e4*K))
```

```
In [ ]: burns = []
samples = []
for i in range(int(1e3)):
    burns.append(bs[int(i*K)])
for i in range(int(1e4)):
    samples.append(ss[int(i*K)])
xb, yb = [c[0] for c in burns], [c[1] for c in burns]
xs, ys = [c[0] for c in samples], [c[1] for c in samples]
```

```
In [ ]: plt.title('Burn-in path of standart LMC (per K=50 itr)')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-5.5,2.5)
plt.xlim(-5.3,2.5)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig], colors='orange', linewidths=1)
plt.scatter(xb, yb, s=7)
plt.plot(xb, yb)
plt.savefig("2-4-1.png")
```

```
In [ ]: xbb, ybb = [c[0] for c in bs], [c[1] for c in bs]
plt.title('Burn-in path of standart LMC (per itr)')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-7,2.5)
plt.xlim(-5.3,2.5)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig], colors='orange', linewidths=1)
plt.scatter(xbb, ybb, s=7)
plt.plot(xbb, ybb)
plt.savefig("2-4-0.png")
```

```
In [ ]: plt.title('Sampling values of standard LMC')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-4,4)
plt.xlim(-4,4)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig], colors='orange', linewidths=1)
plt.scatter(xs, ys, s=1)
plt.savefig("2-4-2.png")
```

```
In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[: (k+1)])
    # mu_y = np.mean(y[: (k+1)])
    # var_x = np.var(x[: (k+1)])
    # var_y = np.var(y[: (k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)
```

## RMHMC/LMC

```
In [ ]: init = [0, -10]
mean = [0, 0]
sig = inv # 2nd derivative always inverse phi
L = 1
eps = 1
burns, samples = HMC_sampling(init, eps, L, mean, sig, inv)
xb, yb = [c[0] for c in burns], [c[1] for c in burns]
xs, ys = [c[0] for c in samples], [c[1] for c in samples]
```

```
In [ ]: plt.title('Burn-in path of HMC with inverse covariance')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-3.5,3.5)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig], colors='orange', linewidths=1)
plt.scatter(xb, yb, s=7)
plt.plot(xb, yb)
plt.savefig("2-5-1.png")
```

```
In [ ]: plt.title('Sampling values of HMC with inverse covariance')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-3.5,3.5)
plt.ylim(-3.5,3.5)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [eig, 3*eig, 30*eig], colors='orange', linewidths=1)
plt.scatter(xs, ys, s=1)
plt.savefig("2-5-2.png")
```

```
In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[: (k+1)])
    # mu_y = np.mean(y[: (k+1)])
    # var_x = np.var(x[: (k+1)])
    # var_y = np.var(y[: (k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)
```



2.

### Contour

```
In [ ]: text_file = open("project2_data.txt", "r")
content = text_file.read().split('\n')
data = [float(i) for i in content[:100]]
```

```
In [ ]: ## a)
def energy_post(t, y):
    y = np.array(y)
    ene = (t[0]**2)/2+(t[1]**2)/2+sum((y-t[0]-t[1]**2)**2)/4
    return ene
x1 = np.linspace(-5, 5, 1000)
x2 = np.linspace(-5, 5, 1000)
x11, x22 = np.meshgrid(x1, x2)
x = np.concatenate((x11.reshape(1000**2,1), x22.reshape(1000**2,1)), axis=1)
e = []
for i in range(1000**2):
    xi = x[i, ]
    ene = energy_post(xi, data)
    e.append(ene)
```

```
In [ ]: plt.title('Contour plot of the energy function')
plt.xlabel('theta1')
plt.ylabel('theta2')
plt.ylim(-3,3)
plt.xlim(-3.5,3.5)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200, 500], colors='orange')
plt.savefig("3-1.png")
```

```

In [ ]: def energy_target(q, data):
        y = np.copy(data)
        ene = (q[0]**2)/2+(q[1]**2)/2+sum((y-q[0]-q[1]**2)**2)/4
        return ene

    def deriv_energy(q, data):
        y = np.copy(data)
        q = np.copy(q)
        deriv = -(np.sum(y - q[0] - q[1]**2))/2*np.array([1,2*q[1]])+q
        return deriv

    def energy_prop(p):
        p = np.copy(p)
        # sig_inv = np.linalg.inv(sig)
        return 0.5 * np.dot(p.T, p)

    def momentum(mean, sig):
        p = np.random.multivariate_normal(mean, sig)
        return p

    def HMC_step(p, q, eps, L, sig_inv):
        # proposal
        p_half = p - 0.5*eps*deriv_energy(q, data)
        qL = np.copy(q)
        for l in range(L-1):
            #print(type(qL[0]),type(eps*np.dot(sig, p_half)))
            qL += eps*np.dot(sig_inv, p_half)
            p_half -= eps*deriv_energy(qL, data)
        qL = qL + eps*np.dot(sig_inv, p_half)
        pL = p_half - 0.5*eps*deriv_energy(qL, data)
        ene = energy_target(q, data)-energy_target(qL, data) \
            +energy_prop(p)-energy_prop(pL)

        alpha = np.exp(ene)
        # print(alpha)
        acc = 1
        if alpha >= 1:
            new = qL
        else:
            u = np.random.uniform(0,1)
            if u <= alpha:
                new = qL
            else:
                new = q
                acc = 0
        return new, acc

    def HMC_sampling(init, eps, L, mean, sig, num_itr=2000):
        q = np.array(init, dtype=float)
        sig_inv = np.linalg.inv(sig)
        samples = []
        accs = 0
        for ts in range(num_itr):
            p = momentum(mean, sig)
            new, acc = HMC_step(p, q, eps, L, sig_inv)
            samples.append(new)
            accs += acc
            q = new
        return samples, accs/num_itr

```

```

In [ ]: init = [0, 0]
        mean = [0, 0]
        sig = np.array([[1,0],[0,1]])
        L = 1
        #grid = range(101)/100
        rates = []
        for eps in range(1, 51):
            _, rate = HMC_sampling(init, eps/100, L, mean, sig)
            rates.append([eps/200, 1-rate])

```

```

In [ ]: print(rates)
        plt.title('Rejection rate against different epsilon')
        plt.xlabel('epsilon')
        plt.ylabel('rejection rate')
        plt.plot([x[0] for x in rates],[x[1] for x in rates])
        plt.savefig("3-2.png")
        eps = 0.05

```

```

In [ ]: def energy_target(q, data):
        y = np.copy(data)
        ene = (q[0]**2)/2+(q[1]**2)/2+sum((y-q[0]-q[1]**2)**2)/4
        return ene
    def deriv_energy(q, data):
        y = np.copy(data)
        q = np.copy(q)
        deriv = -(np.sum(y - q[0] - q[1]**2))/2*np.array([1,2*q[1]])+q
        return deriv
    def energy_prop(p):
        p = np.copy(p)
        return 0.5 * np.dot(p.T, p)
    def momentum(mean, sig):
        p = np.random.multivariate_normal(mean, sig)
        return p
    def HMC_step(p, q, eps, L):
        # proposal
        p_half = p - 0.5*eps*deriv_energy(q, data)
        ql = np.copy(q)
        for l in range(L-1):
            #print(type(ql[0]),type(eps*np.dot(sig, p_half)))
            ql += eps*np.dot(sig, p_half)
            p_half -= eps*deriv_energy(ql, data)
            qL = ql + eps*np.dot(sig, p_half)
            pL = p_half - 0.5*eps*deriv_energy(qL, data)
            ene = energy_target(q, data)-energy_target(qL, data) \
                +energy_prop(p)-energy_prop(pL)
            alpha = np.exp(ene)
            # print(alpha)
            if alpha >= 1:
                new = qL
            else:
                u = np.random.uniform(0,1)
                if u <= alpha:
                    new = qL
                else:
                    new = q
        return new
    def HMC_sampling(init, eps, L, mean, sig, num_burn=int(1e3), num_sample=int(1e4)):
        q = np.array(init, dtype=float)
        # sig_inv = np.linalg.inv(sig)
        burns = [init]
        for tb in range(num_burn):
            p = momentum(mean, sig)
            new = HMC_step(p, q, eps, L)
            burns.append(new)
            q = new
        samples = []
        for ts in range(num_sample):
            p = momentum(mean, sig)
            new = HMC_step(p, q, eps, L)
            samples.append(new)
            q = new
        return burns, samples

```

```

In [ ]: init = [0, 0]
        mean = [0, 0]
        eps = 0.05
        sig = np.array([[1,0],[0,1]])
        L = 25
        _, samples = HMC_sampling(init, eps, L, mean, sig)
        x, y = [s[0] for s in samples],[s[1] for s in samples]

```

```

In [ ]: mu_x = np.mean(x)
        mu_y = np.mean(y)
        var_x = np.var(x)
        var_y = np.var(y)
        print(mu_x, mu_y, var_x, var_y)

```

```

In [ ]: def energy_target(q, data):
        y = np.copy(data)
        ene = (q[0]**2)/2+(q[1]**2)/2+sum((y-q[0]-q[1]**2)**2)/4
        return ene
    def proposal():
        y = np.random.normal(0, eps)
        return y
    def enegy_prop(prop, eps):
        p = np.copy(prop)
        inv = np.array([[1/eps**2,0],[0,1/eps**2]])
        return 0.5 * np.dot(np.dot(p.T, inv), p)
    def MH_step(x):
        prop = [proposal() for xt in x]
        alpha = np.exp(energy_target(x, data)-energy_target(prop, data)+enegy_prop(prop, eps)-enegy_prop(x, eps))
        # print(alpha)
        # print(energy_target(x, inv),energy_target(prop, inv),enegy_prop(prop, eps),enegy_prop(x, eps))
        if alpha >= 1:
            new = prop
        else:
            u = np.random.uniform(0,1)
            if u <= alpha:
                new = prop
            else:
                new = x
        return new
    def MH_sampling(init, num_burn=int(1e3), num_sample=int(1e4)):
        x = init
        burns = [init]
        for tb in range(num_burn):
            new = MH_step(x)
            burns.append(new)
            x = new
        samples = []
        for ts in range(num_sample):
            new = MH_step(x)
            samples.append(new)
            x = new
        return burns, samples

```

```

In [ ]: K = int(25)
        eps = 0.05
        init = [0, 0]
        bs, ss = MH_sampling(init, num_burn=int(1e3*K), num_sample=int(1e4*K))
        burns = []
        samples = []
        for i in range(int(1e3)):
            burns.append(bs[int(i*K)])
        for i in range(int(1e4)):
            samples.append(ss[int(i*K)])
        xb, yb = [c[0] for c in burns], [c[1] for c in burns]
        xs, ys = [c[0] for c in samples], [c[1] for c in samples]

```

```

In [ ]: plt.title('Burn-in path of Metropolis-Hasting for banana distribution')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.xlim(-0.1,0.5)
        plt.ylim(-0.2,0.2)
        plt.contour(x1, x2, np.array(e).reshape(1000,1000), [80, 100], colors='orange', linewidths=1)
        plt.scatter(xb, yb, s=7)
        plt.plot(xb, yb)
        plt.savefig("3-1-1.png")

```

```

In [ ]: plt.title('Sampling values of Metropolis-Hasting for banana distribution')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.xlim(-0.1,0.5)
        plt.ylim(-0.2,0.2)
        plt.contour(x1, x2, np.array(e).reshape(1000,1000), [80, 100], colors='orange', linewidths=1)
        plt.scatter(xs, ys, s=1)
        plt.savefig("3-1-2.png")

```

```
In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[: (k+1)])
    # mu_y = np.mean(y[: (k+1)])
    # var_x = np.var(x[: (k+1)])
    # var_y = np.var(y[: (k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)
```

### Metropolis-Hasting with random walk

```
In [ ]: def MH_step_rw(x):
    prop = [xt+proposal() for xt in x]
    alpha = np.exp(energy_target(x, data)-energy_target(prop, data))
    # print(alpha)
    if alpha >= 1:
        new = prop
    else:
        u = np.random.uniform(0,1)
        if u <= alpha:
            new = prop
        else:
            new = x
    return new
def MH_sampling_rw(init, num_burn=int(1e3), num_sample=int(1e4)):
    x = init
    burns = []
    for tb in range(num_burn):
        new = MH_step_rw(x)
        burns.append(new)
        x = new
    samples = []
    for ts in range(num_sample):
        new = MH_step_rw(x)
        samples.append(new)
        x = new
    return burns, samples
```

```
In [ ]: K = int(25)
eps = eps
init = [0, 0]
bs, ss = MH_sampling_rw(init, num_burn=int(1e3*K), num_sample=int(1e4*K))
burns = []
samples = []
for i in range(int(1e3)):
    burns.append(bs[int(i*K)])
for i in range(int(1e4)):
    samples.append(ss[int(i*K)])
xb, yb = [c[0] for c in burns], [c[1] for c in burns]
xs, ys = [c[0] for c in samples], [c[1] for c in samples]
```

```
In [ ]: plt.title('Burn-in path of random walk Metropolis-Hasting for banana distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-2.5, 2.5)
plt.xlim(-3, 3)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200, 225], colors='orange', 1
inewidths=1)
plt.scatter(xb, yb, s=7)
plt.plot(xb, yb)
plt.savefig("3-2-1.png")
```

```
In [ ]: plt.title('Sampling values of random walk Metropolis-Hasting for banana distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-2.5, 2.5)
plt.xlim(-3, 3)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200], colors='orange', linewidths=1)
plt.scatter(xs, ys, s=1)
plt.savefig("3-2-2.png")
```

```
In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[: (k+1)])
    # mu_y = np.mean(y[: (k+1)])
    # var_x = np.var(x[: (k+1)])
    # var_y = np.var(y[: (k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)
```

### LMC for banana distribution

```
In [ ]: init = [0, 0]
mean = [0, 0]
sig = np.array([[1,0],[0,1]])
eps = 0.05
L = 1
burns, samples = HMC_sampling(init, eps, L, mean, sig, num_burn=int(1e3), num_sample=int(1e4))
xb, yb = [c[0] for c in burns], [c[1] for c in burns]
xs, ys = [c[0] for c in samples], [c[1] for c in samples]
```

```
In [ ]: plt.title('Burn-in path of LMC (K=1) for banana distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-2.5, 2.5)
plt.xlim(-3, 3)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200, 225], colors='orange', 1
inewidths=1)
plt.scatter(xb, yb, s=7)
plt.plot(xb, yb)
plt.savefig("3-1-1.png")
```

```
In [ ]: plt.title('Sampling values of LMC (K=1) for banana distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-2.5, 2.5)
plt.xlim(-3, 3)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200], colors='orange', linewidths=1)
plt.scatter(xs, ys, s=1)
plt.savefig("3-1-2.png")
```

```
In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[(k+1)])
    # mu_y = np.mean(y[(k+1)])
    # var_x = np.var(x[(k+1)])
    # var_y = np.var(y[(k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)
```

## K=25

```
In [ ]: init = [0, 0]
mean = [0, 0]
sig = np.array([[1,0],[0,1]])
eps = 0.05
L = 1
K = int(25)
bs, ss = HMC_sampling(init, eps, L, mean, sig, num_burn=int(1e3*K), num_sample=int(1e4*K))
burns = []
samples = []
for i in range(int(1e3)):
    burns.append(bs[int(i*K)])
for i in range(int(1e4)):
    samples.append(ss[int(i*K)])
xb, yb = [c[0] for c in burns], [c[1] for c in burns]
xs, ys = [c[0] for c in samples], [c[1] for c in samples]
```

```
In [ ]: plt.title('Burn-in path of LMC (K=25) for banana distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-2.5, 2.5)
plt.xlim(-3, 3)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200, 225], colors='orange', linewidths=1)
plt.scatter(xb, yb, s=7)
plt.plot(xb, yb)
plt.savefig("3-3-1.png")
```

```
In [ ]: plt.title('Sampling values of LMC (K=25) for banana distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-2.5, 2.5)
plt.xlim(-3, 3)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200], colors='orange', linewidths=1)
plt.scatter(xs, ys, s=1)
plt.savefig("3-3-2.png")
```

```
In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[: (k+1)])
    # mu_y = np.mean(y[: (k+1)])
    # var_x = np.var(x[: (k+1)])
    # var_y = np.var(y[: (k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)
```

### HMC L=5

```
In [ ]: init = [0, 0]
mean = [0, 0]
eps = 0.05
sig = np.array([[1,0],[0,1]])
L = 5
burns, samples = HMC_sampling(init, eps, L, mean, sig)
xb, yb = [c[0] for c in burns], [c[1] for c in burns]
xs, ys = [c[0] for c in samples], [c[1] for c in samples]
```

```
In [ ]: plt.title('Burn-in path of HMC (L=5) for banana distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-2.5, 2.5)
plt.xlim(-3, 3)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200, 225], colors='orange', 1
inewidths=1)
plt.scatter(xb, yb, s=7)
plt.plot(xb, yb)
plt.savefig("3-4-1.png")
```

```
In [ ]: plt.title('Sampling values of HMC (L=5) for banana distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-2.5, 2.5)
plt.xlim(-3, 3)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200], colors='orange', linewidths=1)
plt.scatter(xs, ys, s=1)
plt.savefig("3-4-2.png")
```

```
In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[: (k+1)])
    # mu_y = np.mean(y[: (k+1)])
    # var_x = np.var(x[: (k+1)])
    # var_y = np.var(y[: (k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)
```



## HMC L=25

```
In [ ]: init = [0, 0]
mean = [0, 0]
eps = 0.05
sig = np.array([[1,0],[0,1]])
L = 25
burns, samples = HMC_sampling(init, eps, L, mean, sig)
xb, yb = [c[0] for c in burns], [c[1] for c in burns]
xs, ys = [c[0] for c in samples], [c[1] for c in samples]
```

```
In [ ]: plt.title('Burn-in path of HMC (L=25) for banana distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-2.5, 2.5)
plt.xlim(-3, 3)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200, 225], colors='orange', 1
inewidths=1)
plt.scatter(xb, yb, s=7)
plt.plot(xb, yb)
plt.savefig("3-5-1.png")
```

```
In [ ]: plt.title('Sampling values of HMC (L=25) for banana distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-2.5, 2.5)
plt.xlim(-3, 3)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200], colors='orange', linewidths=1)
plt.scatter(xs, ys, s=1)
plt.savefig("3-5-2.png")
```

```
In [ ]: K = int(1e4)
s_x = 0
s_y = 0
for k in range(K-1):
    c = (K-(k+1))
    rho_x = 0
    rho_y = 0
    # mu_x = np.mean(x[(k+1)])
    # mu_y = np.mean(y[(k+1)])
    # var_x = np.var(x[(k+1)])
    # var_y = np.var(y[(k+1)])
    for j in range(k+1, K):
        rho_x += (xs[j]-mu_x)*(xs[j-k]-mu_x)
        rho_y += (ys[j]-mu_y)*(ys[j-k]-mu_y)
    s_x += (1-(k+1)/K)*rho_x/c/var_x
    s_y += (1-(k+1)/K)*rho_y/c/var_y
ess_x = K/(1+2*s_x)
ess_y = K/(1+2*s_y)
print(ess_x, ess_y)
```

## Acceptance and rejection path

```

In [ ]: def HMC_step(p, q, eps, L):
    # proposal
    p_half = p - 0.5*eps*deriv_energy(q, data)
    q1 = np.copy(q)
    path = [q]
    for l in range(L-1):
        #print(type(q1[0]),type(eps*np.dot(sig, p_half)))
        q1 += eps*np.dot(sig, p_half)
        path.append(np.copy(q1))
        p_half -= eps*deriv_energy(q1, data)
    qL = q1 + eps*np.dot(sig, p_half)
    path.append(np.copy(qL))
    pL = p_half - 0.5*eps*deriv_energy(qL, data)
    ene = energy_target(q, data)-energy_target(qL, data) \
        +energy_prop(p)-energy_prop(pL)
    alpha = np.exp(ene)
    # print(alpha)
    if alpha >= 1:
        mask = 1
        new = qL
    else:
        u = np.random.uniform(0,1)
        if u <= alpha:
            new = qL
            mask = 1
        else:
            new = q
            mask = 0
    return new, path, mask
def HMC_sampling(init, eps, L, mean, sig, num_burn=int(1e3), num_sample=int(1e4)):
    q = np.array(init, dtype=float)
    # sig_inv = np.linalg.inv(sig)
    burns = [init]
    paths = []
    masks = []
    for tb in range(num_burn):
        p = momentum(mean, sig)
        new, path, mask = HMC_step(p, q, eps, L)
        burns.append(new)
        paths.append(path)
        masks.append(mask)
        q = new
    samples = []
    for ts in range(num_sample):
        p = momentum(mean, sig)
        new, path, _ = HMC_step(p, q, eps, L)
        samples.append(new)
        q = new
    return burns, samples, paths, masks

```

```

In [ ]: init = [0, 0]
mean = [0, 0]
eps = 0.05
sig = np.array([[1,0],[0,1]])
L = 5
burns, samples, paths, masks = HMC_sampling(init, eps, L, mean, sig, num_burn=int(1e3), num_sample=int(1))

```

```

In [ ]: for i in range(999, -1, -1):
    if masks[i] == 0:
        rej = paths[i]
        break
    else:
        acc = paths[i]

```

```
In [ ]: plt.title('Rejected and Accepted paths for HMC L=5')
plt.xlim(-0.5,2)
plt.ylim(-0.5,0.5)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200], colors='orange', linewidths=1)
plt.scatter([r[0] for r in rej], [r[1] for r in rej], s=7, label='rejected')
plt.plot([r[0] for r in rej], [r[1] for r in rej])
plt.scatter([r[0] for r in acc], [r[1] for r in acc], s=7, label='accepted')
plt.plot([r[0] for r in acc], [r[1] for r in acc])
plt.legend()
plt.savefig("3-4-0.png")
```

```
In [ ]: init = [0, 0]
mean = [0, 0]
eps = 0.05
sig = np.array([[1,0],[0,1]])
L = 25
burns, samples, paths, masks = HMC_sampling(init, eps, L, mean, sig, num_burn=int(1e3), num_sample=int(1))
```

```
In [ ]: for i in range(999, -1, -1):
    if masks[i] == 0:
        rej = paths[i]
        break
    else:
        acc = paths[i]
```

```
In [ ]: print(rej[0], acc[0])
```

```
In [ ]: plt.title('Rejected and Accepted paths for HMC L=25')
plt.xlim(-1.5,0.5)
plt.ylim(-2.5,0)
plt.contour(x1, x2, np.array(e).reshape(1000,1000), [100, 125, 150, 175, 200], colors='orange', linewidths=1)
plt.scatter([r[0] for r in rej], [r[1] for r in rej], s=7, label='rejected')
plt.plot([r[0] for r in rej], [r[1] for r in rej])
plt.scatter([r[0] for r in acc], [r[1] for r in acc], s=7, label='accepted')
plt.plot([r[0] for r in acc], [r[1] for r in acc])
plt.legend()
plt.savefig("3-5-0.png")
```