Statistical Computing 2

숙제 7

2019년 가을학기

응용통계학과 석사과정 최석준

1. Using Hamiltonian Monte Carlo method, generate samples from

$$\frac{1}{3}N(\mathbf{0}, I) + \frac{1}{3}N\left(\begin{bmatrix}-8\\-6\end{bmatrix}, \begin{bmatrix}1 & 0.9\\0.9 & 1\end{bmatrix}\right) + \frac{1}{3}N\left(\begin{bmatrix}8\\6\end{bmatrix}, \begin{bmatrix}1 & -0.9\\-0.9 & 1\end{bmatrix}\right)$$

먼저 Hamiltonian Monte Carlo (이하 HMC)의 로직만을 구현한다.

[Python]

```
#python 3 file created by Choi, Seokjun

#using Hamiltonian monte carlo,
#get standard normal samples.

#더 생각해볼것/할일:
# 1. log(0)에 가까운경우
# underflow시 chain생성 끊지말고 그냥 그 proposal reject한담에 또 시도하도록? <<완료(일단이렇게구현해둠)
# (<흠... 하지만 이거 conv.prop을 흔들수도 있을것같음)
# 2. method 정리 (2d전용은 아니지만 core에두기엔 애매한것들 어디에 둘까 생각해볼것
# #(중간class를 만들면 클래스 계층이 쓸데없이 너무 복잡해지는것이아닌지))

from math import log, exp, pi
from random import uniform, normalvariate, seed
from statistics import mean
import time
import multiprocessing as mp
import os

import matplotlib.pyplot as plt



class MC_Hamiltonian:
    def __init__(self, log_target_pdf, log_target_grad, leapfrog_step_num, step_size, initial):
        self.dim = len(initial)
        self.log_target_pdf = log_target_pdf
        self.log_target_grad = log_target_grad
        self.leapfrog_step_num = leapfrog_step_num #L
        self.step_size = step_size #epsilon
        self.MC_sample   = [tuple(initial)]
        self.MC_momentum = [tuple([normalvariate(0, 1) for _ in range(self.dim)])]
        # self.MC_momentum = [(0,0)]
        self.momentum_stdNormPrior_cov = 1 #음 lecture note상 임의의 matrix M인데 당장은 이것 1로 fix함. inverse 구현이까다로움
        self.momentum_stdNormPrior_cov_inv = 1

        self.HMCiter = 0
        self.HMCaccept = 0
        self.HMC_underflowexception_count = 0
```

```
        self.pid = None


    def leap_frog_step(self, start_sample_point, start_momentum):
        log_target_gradient = self.log_target_grad(start_sample_point)
        momentum = [start_momentum[i] + 0.5 * self.step_size * log_target_gradient[i] for i in range(self.dim)]
        sample_point = [start_sample_point[i] + self.step_size * self.momentum_stdNormPrior_cov_inv * momentum[i] for i in range(self.dim)]
        for _ in range(1, self.leapfrog_step_num):
            log_target_gradient = self.log_target_grad(start_sample_point)
            momentum = [momentum[i] + self.step_size * log_target_gradient[i] for i in range(self.dim)]
            sample_point = [sample_point[i] + self.step_size * self.momentum_stdNormPrior_cov_inv * momentum[i] for i in range(self.dim)]
            #cov_inv 제대로 구현시 여기는 matrix 연산임
        momentum = [start_momentum[i] + 0.5 * self.step_size * log_target_gradient[i] for i in range(self.dim)]
        return (sample_point, momentum)

    def log_normal_pdf(self, x_vec):
        #not lognormal pdf. but log(std.normal.pdf)
        #need only kernel part (not constant)
        log_kernel = -0.5 * sum([x**2 for x in x_vec])
        return log_kernel

    def log_r_calculator(self, last_sample_point, last_momentum,
                                proposed_sample_point, proposed_momentum):
        log_r = self.log_target_pdf(proposed_sample_point) + self.log_normal_pdf(proposed_momentum) ₩
            - self.log_target_pdf(last_sample_point) - self.log_normal_pdf(last_momentum)
        return log_r

    def MH_rejection_step(self, last_sample_point, last_momentum,
                                proposed_sample_point, proposed_momentum):
        unif_sample = uniform(0,1)

        try:
            log_HMC_ratio = self.log_r_calculator(last_sample_point, last_momentum, proposed_sample_point, proposed_momentum)
        except ValueError: #underflow error catch (문제 있을 수 있음. 나중에 feedback 받을 것)
            #나중에라도 이 코드 쓸거면 윗줄에서 error를 e로 받을것 (except ValueError as e)
            # print("pid: ", self.pid, ": early ended at "+ str(self.HMCiter) +"-th iteration.")
            # print("exception catched!: ", e)
            # print("replcate previous point and restart at there.")

            self.HMC_underflowexception_count += 1
            return False

        if log(unif_sample) < log_HMC_ratio:
            return True
        else:
            return False

    def sampler(self):
        last_sample_point = self.MC_sample[-1]
        last_momentum = self.MC_momentum[-1]
        new_momentum = tuple([normalvariate(0, 1) for _ in range(self.dim)])
        proposal_sample_point, proposal_momentum = self.leap_frog_step(last_sample_point, new_momentum)
        accept_bool = self.MH_rejection_step(last_sample_point, last_momentum, proposal_sample_point, proposal_momentum)
        self.HMCiter += 1
        if accept_bool:
```

```
                self.MC_sample.append(tuple(proposal_sample_point))
                self.MC_momentum.append(tuple(proposal_momentum))
                self.HMCaccept += 1
            else :
                self.MC_sample.append(tuple(last_sample_point))
                self.MC_momentum.append(tuple(last_momentum))

    def generate_samples(self, num_samples, verbose=True):
        start_time = time.time()
        for i in range(1, num_samples):
            self.sampler()
            if i%100000 == 0 and verbose and self.pid is not None:
                print("pid:",self.pid," iteration", i, "/", num_samples)
            elif i%100000 == 0 and verbose and self.pid is None:
                print("iteration", i, "/", num_samples)
        elap_time = time.time()-start_time
        if self.pid is not None and verbose:
            print("pid:",self.pid, "iteration", num_samples, "/", num_samples, " done! (elapsed time for execution: ",
                        elap_time//60,"min ", elap_time%60,"sec)")
            print("pid:",self.pid, "exception catched: ", self.HMC_underflowexception_count)
        elif self.pid is None and verbose:
            print("iteration", num_samples, "/", num_samples, " done! (elapsed time for execution: ",
                        elap_time//60,"min ", elap_time%60,"sec)")
            print("exception catched: ", self.HMC_underflowexception_count)

    def burnin(self, num_burn_in):
        self.MC_sample = self.MC_sample[num_burn_in-1:]
```

다음으로는 chain 진단용 2-dimension case 전용 그림그리는 함수들과 acf 관련 함수를 구현한다.

```
[Python]
class MC_Hamiltonian_withUtil_2dim(MC_Hamiltonian):
    def __init__(self, log_target_pdf, log_target_grad, leapfrog_step_num, step_size, initial):
        super().__init__(log_target_pdf, log_target_grad, leapfrog_step_num, step_size, initial)

    def get_specific_dim_samples(self, dim_idx):
        #dim_idx 0부터 시작함
        if dim_idx >= self.dim:
            raise ValueError("dimension index should be lower than number of dimension. note that index starts at 0")
        return [smpl[dim_idx] for smpl in self.MC_sample]

    def show_scatterplot(self, show=True):
        x_vec = self.get_specific_dim_samples(0)
        y_vec = self.get_specific_dim_samples(1)
        plt.plot(x_vec, y_vec, '-o', marker=".")
        if show:
            plt.show()

    def show_momentum(self, show=True):
        norm_momentum_vec = [(momentum[0]**2+momentum[1]**2)**0.5 for momentum in self.MC_momentum]
        plt.plot(range(len(norm_momentum_vec)),norm_momentum_vec)
        if show:
            plt.show()

    def get_accept_rate(self):
        try:
```

```
            acc_rate = self.HMCaccept / self.HMCiter
        except ZeroDivisionError:
            acc_rate = 0
        return acc_rate

    def get_autocorr(self, dim_idx, maxLag):
        y = self.get_specific_dim_samples(dim_idx)
        acf = []
        y_mean = mean(y)
        y = [elem - y_mean   for elem in y]
        n_var = sum([elem**2 for elem in y])
        for k in range(maxLag+1):
            N = len(y)-k
            n_cov_term = 0
            for i in range(N):
                n_cov_term += y[i]*y[i+k]
            try:
                k_lag_acf = n_cov_term / n_var
            except ZeroDivisionError:
                # raise ZeroDivisionError("n_var is too small (underflow raised.)") #전파 필요시
                k_lag_acf = 1
            acf.append(k_lag_acf)
        return acf

    def show_acf(self, dim_idx, maxLag, show=True):
        grid = [i for i in range(maxLag+1)]
        acf = self.get_autocorr(dim_idx, maxLag)
        plt.ylim([-1,1])
        plt.bar(grid, acf, width=0.3)
        plt.axhline(0, color="black", linewidth=0.8)
        if show:
            plt.show()
```

그리고, 이제 우리 문제의 setting 을 위한 요소를 구현한다.

```
[Python]
#our case
# (1/3)*N((0,0), I) + (1/3)*N((-8,-6), [1, 0.9; 0.9, 1]) + (1/3)*N((8,6), [1, -0.9; -0.9, 1]))
def bivariate_normal_pdf(x_2d, corr, mu_x1, mu_x2, sigmasq_1=1, sigmasq_2=1):
    x1, x2 = x_2d
    ker = (x1-mu_x1)**2/sigmasq_1 + (x2-mu_x2)**2/sigmasq_2 - 2*corr*(x1-mu_x1)*(x2-mu_x2)/(sigmasq_1*sigmasq_2)**0.5
    ker *= (-0.5)/(1-corr**2)
    const = 2*pi*(sigmasq_1*sigmasq_2*(1-corr**2))**0.5
    return exp(ker)/const

def mixture_log_pdf(x_2d):
    try:
        logpdf_val = log(
            bivariate_normal_pdf(x_2d,0,0,0)/3
            + bivariate_normal_pdf(x_2d, 0.9, -8, -6)/3
            + bivariate_normal_pdf(x_2d, -0.9, 8, 6)/3)
    except ValueError:
        err_pdfval = (bivariate_normal_pdf(x_2d,0,0,0)/3
            + bivariate_normal_pdf(x_2d, 0.9, -8, -6)/3
            + bivariate_normal_pdf(x_2d, -0.9, 8, 6)/3)
        err_str = "Underflow: at "+ str(x_2d) + ", pdf value: " + str(err_pdfval) + " is too close to 0."
```

```
            raise ValueError(err_str)
        return logpdf_val

def mixture_log_grad(x_2d):
        x,y = x_2d
        const = (bivariate_normal_pdf(x_2d, 0, 0, 0)/3
            + bivariate_normal_pdf(x_2d, 0.9, -8, -6)/3
            + bivariate_normal_pdf(x_2d, -0.9, 8, 6)/3)
        # const = 1
        ker1 = exp(-(x**2+y**2)/2) / (6*pi)
        ker2 = exp(-(x**2 + y**2 - 1.8*x*y)/(2*(1-0.9**2))) / (6*pi*(1-0.9**2)**0.5)
        ker3 = exp(-(x**2 + y**2 + 1.8*x*y)/(2*(1-0.9**2))) / (6*pi*(1-0.9**2)**0.5)
        diff_by_x = ker1*(-x) + ker2 * (-(x - 0.9*y)/(1 - 0.9**2)) + ker3 * (-(x + 0.9*y)/(1 - 0.9**2))
        diff_by_y = ker1*(-y) + ker2 * (-(y - 0.9*x)/(1 - 0.9**2)) + ker3 * (-(y + 0.9*x)/(1 - 0.9**2))
        diff_by_x /= const
        diff_by_y /= const
        return (diff_by_x, diff_by_y)

def setting_contourplot(start=-10, end=10):
        grid = [x/10 + start for x in range(10*(end-start))]
        mixture_val = [[mixture_log_pdf([x,y]) for x in grid] for y in grid]
        plt.contour(grid, grid, mixture_val, levels=20)
```

HMC module 을 main 으로써 실행 시, 각 cpu core 가 동시에 8 개 chain 을 만들도록 병렬 처리할 것이다. 다음은 각 core 가 할 작업을 지시하기 위한 함수이다.

```
[Python]
#for multiprocessing
def multiproc_1unit_do(result_queue, leapfrog_step_num, step_size, initial, each_num_iter):
        func_pid = os.getpid()
        print("pid: ", func_pid, "start!")
        HMCchain = MC_Hamiltonian_withUtil_2dim(mixture_log_pdf, mixture_log_grad, leapfrog_step_num, step_size, initial)
        HMCchain.pid = func_pid

        HMCchain.generate_samples(each_num_iter)

        result_queue.put(HMCchain)
        print("pid: ", func_pid, " end!")
```

마지막으로, 프로그램의 main 부분이다. 크게 3 부분으로 나뉘어 있다. 첫 부분은 각 cpu core 에 넘길 HMC setting 묶음을 정의한다. 두번째 부분은 multiprocessing 구현이다. 마지막 부분은 동시에 생성된 8 개의 chain 을 가지고 한꺼번에 그림을 그려 출력한다.

```
[Python]
if __name__ == "__main__":
        seed(2019-311-252)

        core_num = 8 #띄울 process 수
        mode = 1 #돌릴 chain setting mode 선택
        #mode setting
        #0. prof. Jin's setting (on c code)
        #1. changing epsilon
        #2. changing leapfrog step L
        #3. set initial point randomly around 0
        #4. set initial point randomly around -8,-6
```

```
#5. set initial point randomly around 8,6

if mode==0:
    #Prof.Jin's c code setting
    #구현이 완전히 같질 않아서(gradient 크기 등) 같은 상황으로 돌지는 않는 것 같음
    each_num_iter = 800000
    each_leapfrog_step_num = [50 for _ in range(8)]
    each_step_size = [0.3 for _ in range(8)]
    each_initial = [(normalvariate(0,10), normalvariate(0,10)) for _ in range(8)] #이건 임의로 주었음

if mode==1:
    each_num_iter = 200000
    each_leapfrog_step_num = [20 for _ in range(8)]
    each_step_size = [0.001, 0.005, 0.01, 0.02, 0.05, 0.1, 0.15, 0.2]
    each_initial = [(normalvariate(0,5), normalvariate(0,5)) for _ in range(8)]

if mode==2:
    each_num_iter = 200000
    each_leapfrog_step_num = [1,2,4,6,8,12,16,20]
    each_step_size = [0.3 for _ in range(8)]
    each_initial = [(normalvariate(0,10), normalvariate(0,10)) for _ in range(8)]

if mode==3:
    each_num_iter = 200000
    each_leapfrog_step_num = [25 for _ in range(8)]
    each_step_size = [0.02 for _ in range(8)]
    each_initial = [(normalvariate(0,1), normalvariate(0,1)) for _ in range(8)]

if mode==4:
    each_num_iter = 200000
    each_leapfrog_step_num = [25 for _ in range(8)]
    each_step_size = [0.05 for _ in range(8)]
    each_initial = [(normalvariate(-8,1), normalvariate(-6,1)) for _ in range(8)]

if mode==5:
    each_num_iter = 200000
    each_leapfrog_step_num = [50 for _ in range(8)]
    each_step_size = [0.05 for _ in range(8)]
    each_initial = [(normalvariate(8,1), normalvariate(6,1)) for _ in range(8)]



#generate HMC chains using parallel multiprocessing
print("start.mp")
proc_vec = []
proc_queue = mp.Queue()

for i in range(core_num):
    unit_proc = mp.Process(target = multiproc_1unit_do,
        args=(proc_queue, each_leapfrog_step_num[i], each_step_size[i], each_initial[i], each_num_iter))
    proc_vec.append(unit_proc)

for unit_proc in proc_vec:
    unit_proc.start()
```

```
    mp_result_vec = []
    for _ in range(core_num):
        each_result = proc_queue.get()
        # print("mp_result_vec_object:", each_result)
        mp_result_vec.append(each_result)

    for unit_proc in proc_vec:
        unit_proc.join()
    print("exit.mp")



    #check traceplot
    grid_column= 8
    grid_row = 4
    plt.figure(figsize=(5*grid_column, 3*grid_row))
    for i, chain in enumerate(mp_result_vec):
        #plot 1
        plt.subplot(grid_row, grid_column, 4*i+1)
        plt.subplots_adjust(hspace=0.6)
        setting_contourplot(-12,12)
        chain.show_scatterplot(show=False)
        title_str = "step size:" + str(round(chain.step_size,4)) ₩
            + "₩nleapfrog iter num:" + str(chain.leapfrog_step_num) ₩
            + "₩ninitial: " + str(round(chain.MC_sample[0][0],4)) + ", " + str(round(chain.MC_sample[0][1],4))
        plt.title(title_str)

        #plot 2
        plt.subplot(grid_row, grid_column, 4*i+2)
        title_str = "total iter num:" + str(chain.HMCiter) ₩
            + "₩nacceptance rate:" + str(round(chain.get_accept_rate(),5))
        plt.title(title_str)
        chain.show_momentum(show=False)

        #plot 3
        plt.subplot(grid_row, grid_column, 4*i+3)
        chain.show_acf(0,10,show=False)
        title_str = "underflow catched:" + str(chain.HMC_underflowexception_count)
        plt.title(title_str)

        #plot 4
        plt.subplot(grid_row, grid_column, 4*i+4)
        chain.show_acf(1,10,show=False)
    plt.show()
```

아래는 코드를 돌린 결과이다.

그래프 출력은 각 체인마다 4 개씩이고, 차례대로 다음과 같다.

1. trace plot on 2D contour plot of target distribution
2. momentum (over each HMC sample index)
3. autocorrelation (x 축 방향)
4. autocorrelation (Y 축 방향)

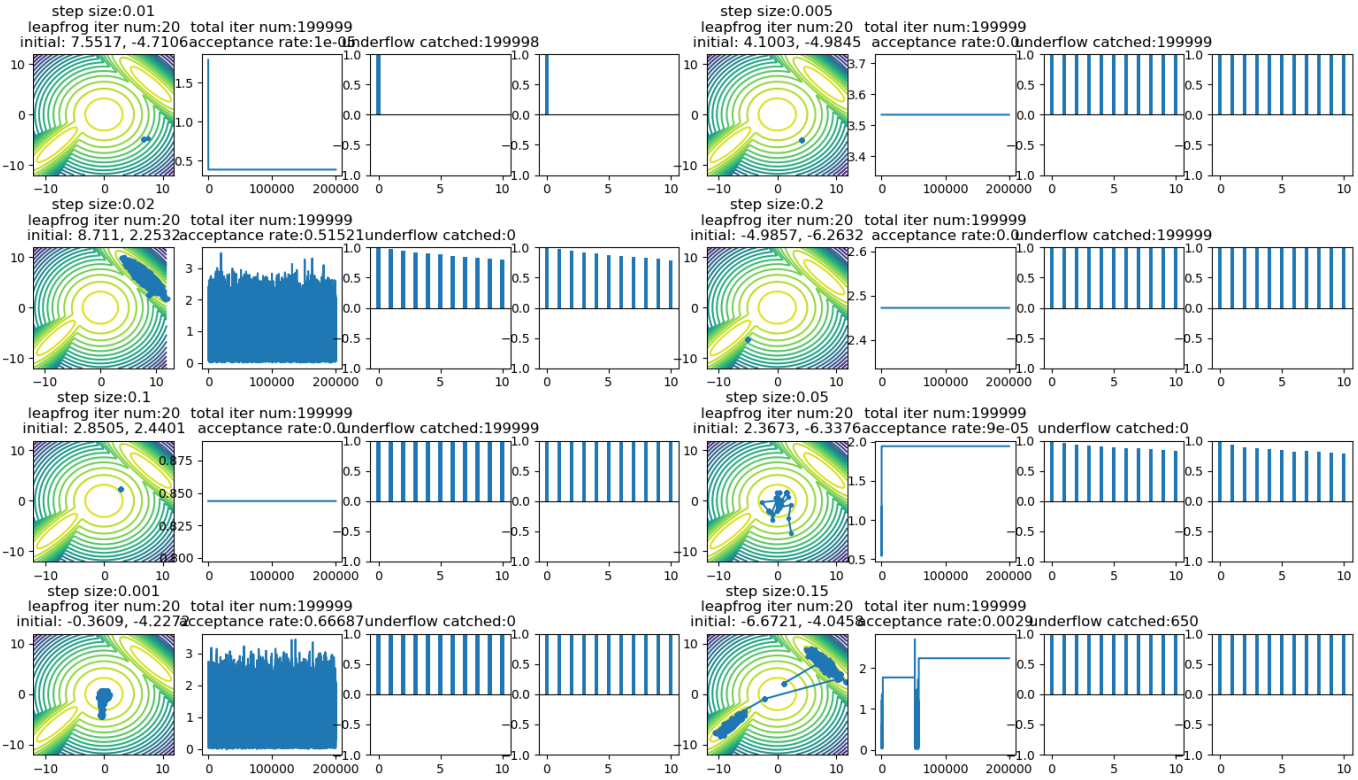
그리고 각 chain 정보를 4 개의 그래프 위에 출력하였다. 각각을 설명하면, step size 는 lecture note 의 epsilon 에 해당하고, leapfrog iter num 은 lecture note 의 L 에 해당한다. initial point 는 말그대로 초기값이고, 모든 mode 에서

random 으로 뽑히도록 되어 있다. iter num 은 chain 을 만들기 위해 돈 반복수로, mode 1~5 의 setting 하에서는 199999 이다. 모든 mode 에서 체인을 20 만개 길이로 만들었는데 시작점이 처음에 들어가서 포함되기 때문이다. 다음으로 accept-reject step 에서의 acceptance rate, 그리고 마지막으로 underflow catch 이다. 이는 Hamiltonian dynamics 가 준 proposal 이 target pdf 가 underflow 가 나서 log target pdf 값이 -inf 로 튀어버리는 영역에 있을 경우 이를 예외처리하고 MC step 에서 해당 sample 을 reject 한다음 chain 을 기존 점에서 재시작한 횟수이다. (위의 HMC 구현에서, mixture_log_pdf 함수의 예외 전파와 MC_Hamiltonian.MH_rejection_step 에서의 예외처리 부분이 이를 구현한 부분이다.) **(질문: 이러면 이론적으로 문제가 있나요? Underflow 나면 해당 예외 잡고 해당 chain 을 끊고 거기까지를 살펴볼까 아니면 지금 코드처럼 이어서 계속 돌릴까 했는데... 일단은 구현은 (실상 low-pdf value 구간에서는 reject 확률이 거의 1 이므로) reject 하고 이어서 다시 proposal 하도록 해놓았습니다.)**

그 외에, 콘솔 출력도 있다. 진행상황/os 상 process id/chain 생성 후 그에 걸린 시간을 출력한다. 출력 예시 및 생성시 걸리는 시간을 보고서에 첨부하기 위해, 가장 오래 걸리는 mode 5 출력만 보고서에 옮긴다.(맨 마지막에 있다.) 나머지 mode 는 L 이 작아 mode 5 보다 훨씬 빨리 끝난다.

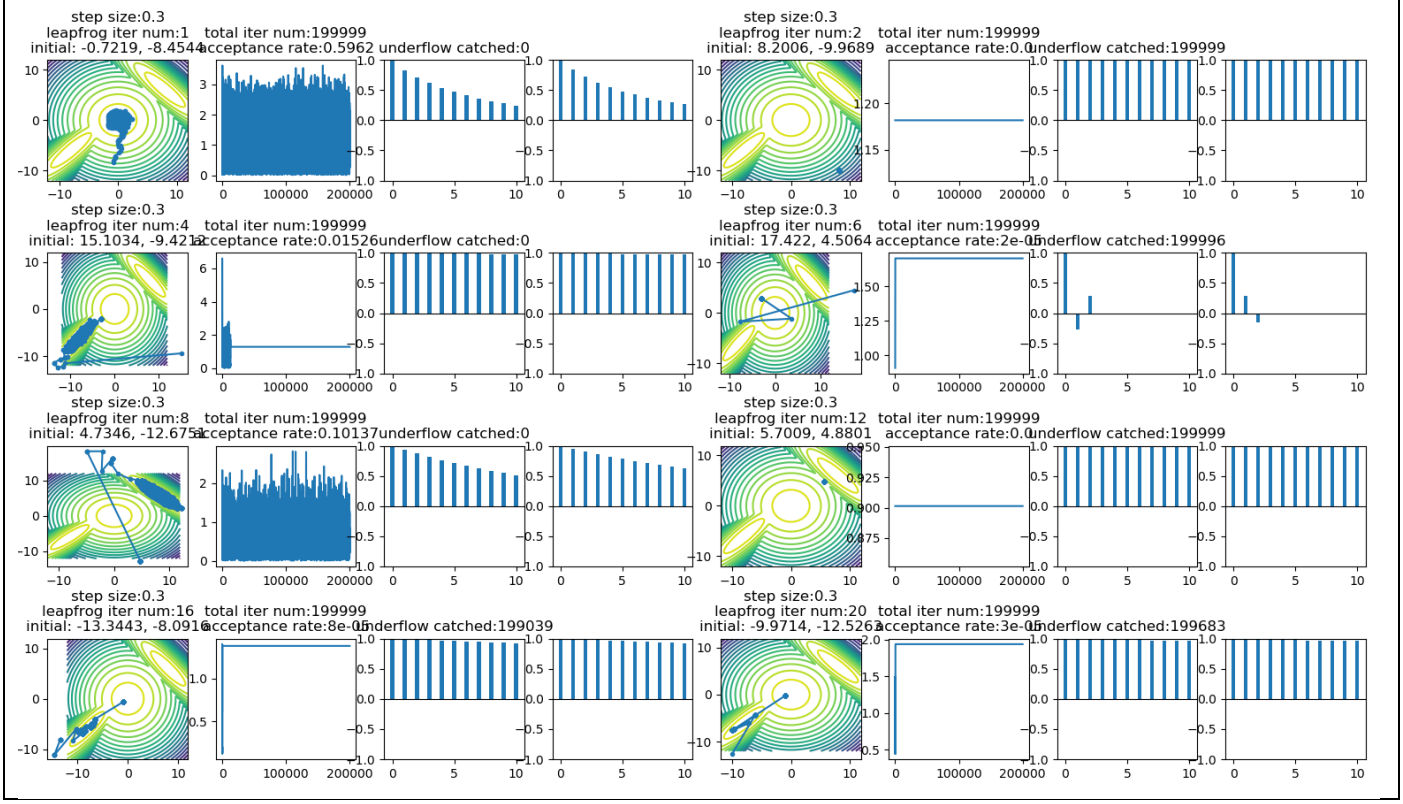
아래에 mode 1~ mode 5 의 출력을 첨부한다. (각 mode 에 대한 정보는 위 main 부분 코드를 참고하라.) 참고로 chain 의 수렴 모습을 눈으로 보기 위해 일부로 burn in 은 하지 않았다. (함수는 만들어 두었다.)

[mode1]



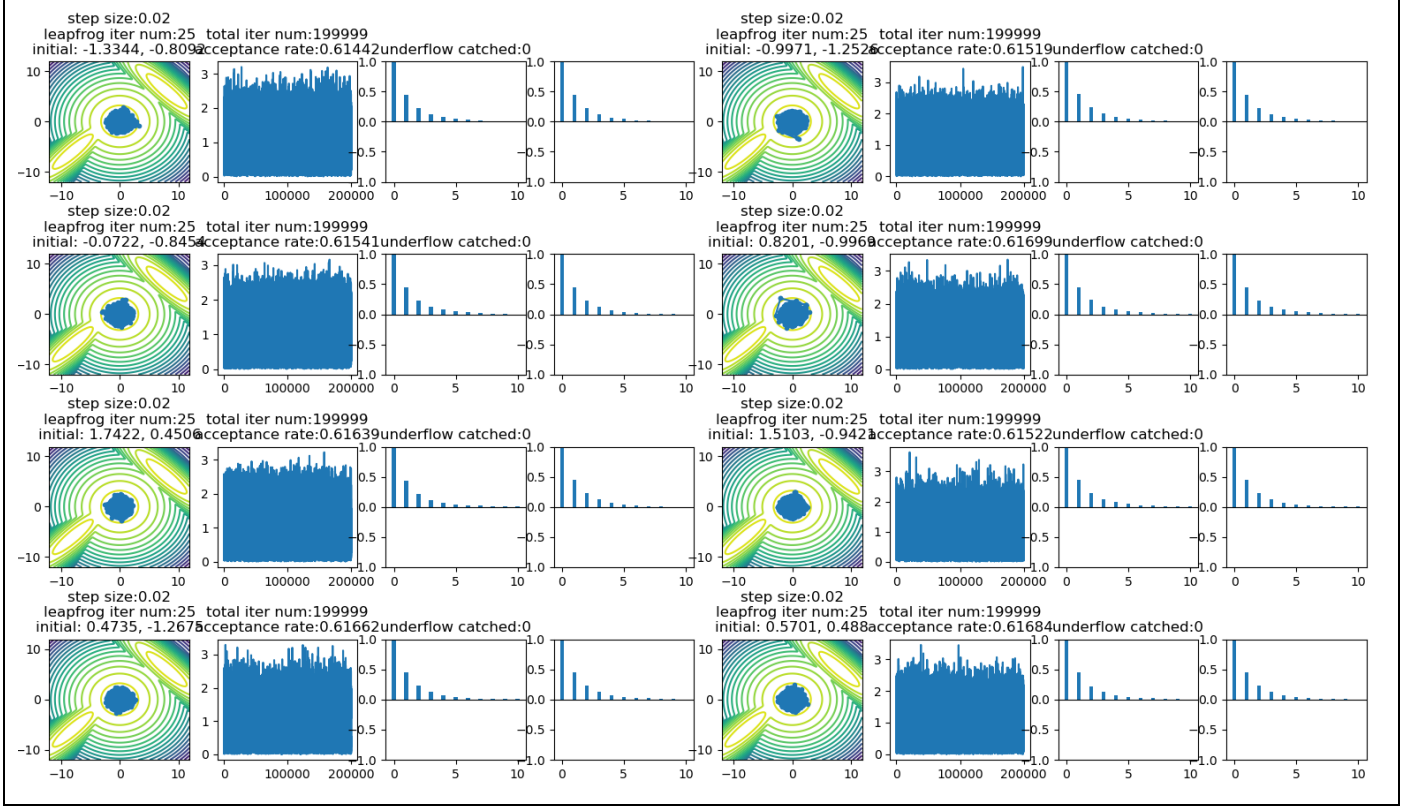
위는 epsilon 을 변화시키며 만든 8 개의 chain 이다. 슬프게도 8 개 체인이 모두 실패작인데, 특정 epsilon 에서는 Hamiltonian dynamic 을 따라 만들어진 proposal 이 너무 확률이 낮은 부분으로 나가 죄다 기각되거나(momentum 이 수평인 부분이 기각된 것이다.), 심지어 underflow 가 나는 영역으로 튕겨 나가서 해당 sample 이 기각되고 chain 이 재시작되었다(그래프 위에 그려진 underflow catched 카운트가 이러한 재시작 횟수를 나타낸다). 아예 못 움직인 chain 도 좀 보인다. 좀 이상한 점은, 직관적으로는 epsilon 값이 커질수록 proposal 이 튕겨 나가는 문제가 많이 생기게 되는 경향이 있을 듯한데, 8 개 chain 의 결과를 보면 동작이 이상해지는 경우에 epsilon 값이 별 규칙이 없어 보인다는 것이다.

[mode2]



위는 L 을 변화시키며 만든 chain 이다. L=1 짜리는 acf 도 양호해보이고 0,0 주변으로 잘 갔지만, 거기서 나오질 못하고 빙빙 돌고 있다. (또한 L=1 이면... HMC 의 장점을 못 살리는 것이 아닐까 싶다.) 그 외 나머지는 모두 문제가 있다. epsilon 이 고정되어 있을 때 L 이 어느 수준 이상으로 커지면 proposal 이 너무 튕겨나가는 경향을 보인다는 직관은 있으나 L=2 일때는 또 알 수 없는 문제로 한 스텝도 움직이지 못했다.

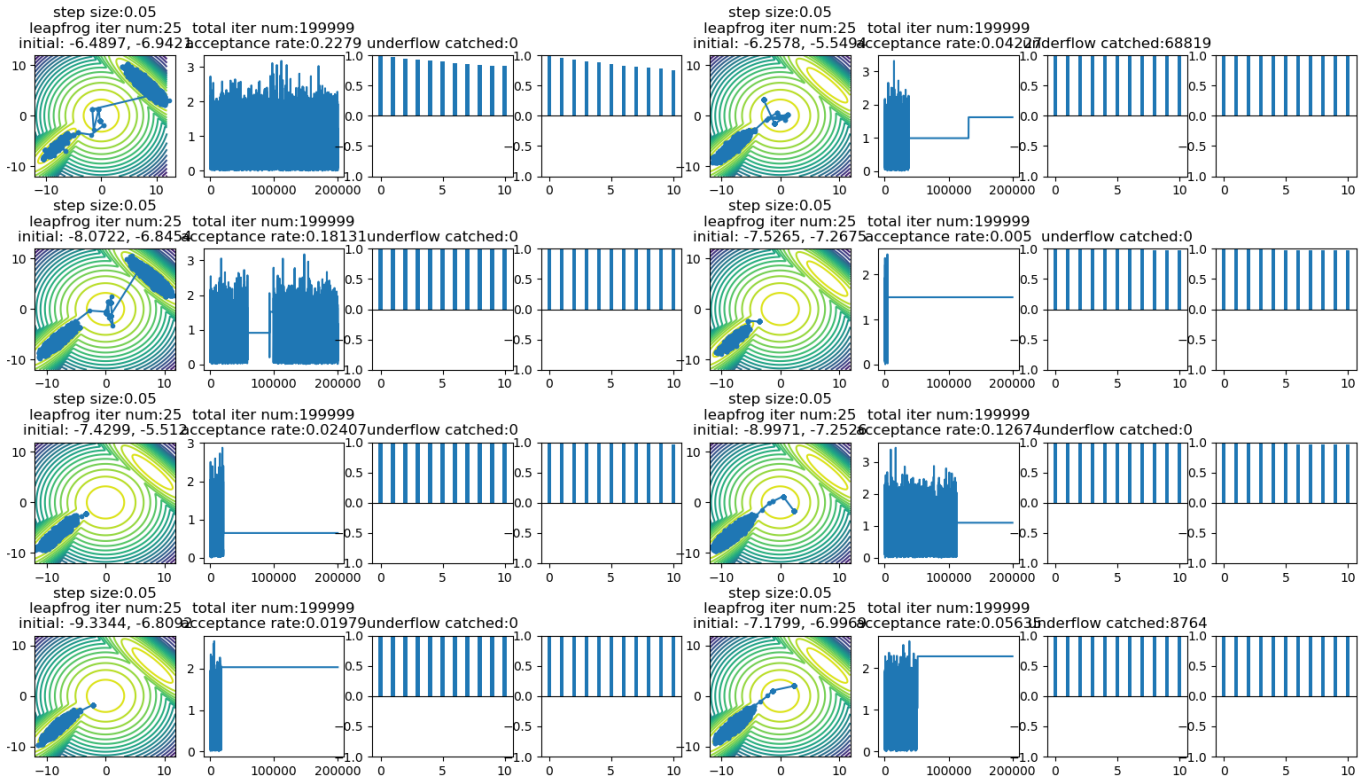
[mode3]



위는 0,0 주변에서 initial point 를 random 으로 뽑고 시작한 8 개의 chain 이다. 전혀 그 다른 mode 로 못 나가고 있다. 그 외 다른 모든 요소는 만족스럽지만 target distribution 전체에서 적절히 sample 을 뽑았다고는 말 할 수 없다. 보고서에 첨부는 안 하지만, 이를 나가게 해 보려고 L, epsilon 을 늘려 시도해봤는데 쉽지 않았다...

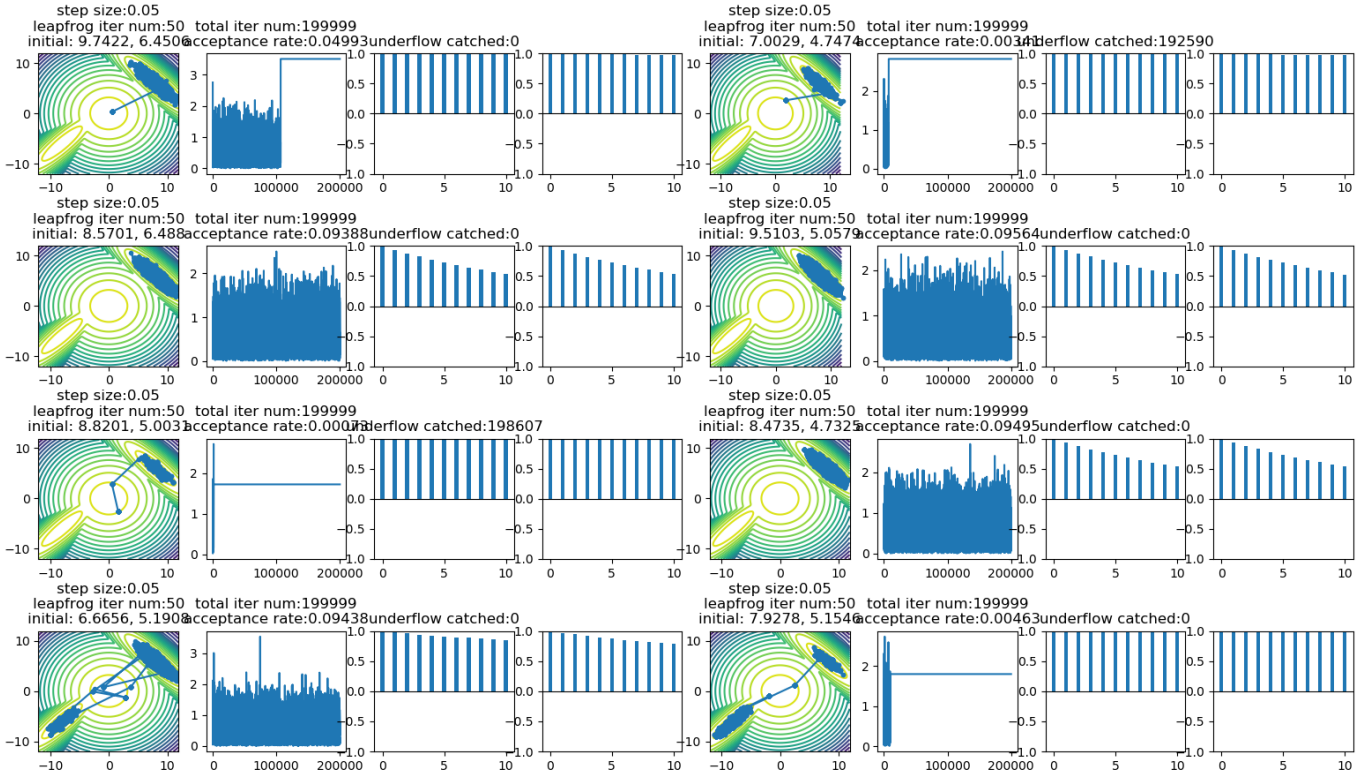
이 결과에서 긍정적인 부분을 굳이 생각해보면, unimodal target 일때는 HMC 가 나름 잘 돌 것이라 생각해볼 수 있다.

[mode4]



위는 -8,-6 주변에서 랜덤으로 뽑은 initial point 에서 시작한 8 개의 chain 이다. 운 좋게 2 경우는 반대쪽까지 옮겨갔는데, acf 를 보면 이것도 문제가 없진 않은 것을 알 수 있다. 나머지는 가운데 봉우리에서 뭔가의 이유로 계속 reject 되고 있다. (왜 가운데를 계속 돌지 않는지 잘 모르겠다. 한 iteration 마다 momentum 이 완전히 새로 뽑히는데... 이렇게 계속 잘못 뽑힐 수가 있나 싶다.)

[mode5]



이번엔 8,6 주변에서 initial point 를 랜덤으로 뽑고 시작한다. 참고로 이 경우가 다섯 경우 중 L 이 가장 커서 생성에 가장 오래 걸리는 mode 이다. 20 만개 sample 을 생성하고, 각 iteration 당 L=50 으로 50 번 뛰기 때문이다. 실행시간을 보기 위해 콘솔 출력을 이번만 옮기겠다. 참고로 콘솔 출력에서 pid 는 os 가 정하므로 돌릴 때마다 달라지고, 8 개 chain 은 동시에 만들어지므로 마지막 완료된 chain 의 실행시간이 총 실행시간이다.

[Console]

```
$ C:/Users/Rjun/AppData/Local/Programs/Python/Python37/python.exe c:/gitProject/statComputing2/HW7/HW7_Hamiltonian_MC.py
start.mp
pid:   14304 start!
pid:   10844 start!
pid:   10000 start!
pid:   9568 start!
pid:   10496 start!
pid:   2716 start!
pid:   12088 start!
pid:   13560 start!
pid: 10496   iteration 100000 / 200000
pid: 14304   iteration 100000 / 200000
pid: 10000   iteration 100000 / 200000
pid: 13560   iteration 100000 / 200000
pid: 9568   iteration 100000 / 200000
pid: 12088   iteration 100000 / 200000
pid: 2716   iteration 100000 / 200000
pid: 10844   iteration 100000 / 200000
pid: 14304 iteration 200000 / 200000   done! (elapsed time for execution:   2.0 min   17.15667200088501 sec)
pid: 14304 exception catched:   0
pid:   14304   end!
pid: 10496 iteration 200000 / 200000   done! (elapsed time for execution:   2.0 min   17.574554443359375 sec)
pid: 10496 exception catched:   192590
pid:   10496   end!
pid: 10000 iteration 200000 / 200000   done! (elapsed time for execution:   2.0 min   19.51354193687439 sec)
```

```
pid: 10000 exception catched:   0
pid:   10000   end!
pid: 13560 iteration 200000 / 200000   done! (elapsed time for execution:   2.0 min   19.750908136367798 sec)
pid: 13560 exception catched:   0
pid:   13560   end!
pid: 2716 iteration 200000 / 200000   done! (elapsed time for execution:   2.0 min   21.18670964241028 sec)
pid: 2716 exception catched:   198607
pid:   2716   end!
pid: 9568 iteration 200000 / 200000   done! (elapsed time for execution:   2.0 min   21.500813961029053 sec)
pid: 9568 exception catched:   0
pid:   9568   end!
pid: 12088 iteration 200000 / 200000   done! (elapsed time for execution:   2.0 min   22.0835018157959 sec)
pid: 12088 exception catched:   0
pid:   12088   end!
pid: 10844 iteration 200000 / 200000   done! (elapsed time for execution:   2.0 min   24.384503602981567 sec)
pid: 10844 exception catched:   0
pid:   10844   end!
exit.mp
```

2 분 24.4 초만에 8 개 chain 생성이 모두 완료되었다.

그래프 출력을 보면, 두 케이스는 acf 가 완만하게 줄지만 시작점이 있는 봉우리에서 탈출을 못 했고, 다른 세 경우는 탈출했으나 가운데에 도달한 후 알 수 없는 이유로 계속 underflow 가 나는 영역을 propose 하여 더 이상 돌질 못하고 있으며, 나머지 두 경우는 반대쪽에 도달했으나 (왜 가운데를 안 돌까?) acf 는 좋지 않다. 어쨌든 제각각 문제가 있음을 알 수 있다.

결론: Multimodal 일 때 HMC 는 좀... 문제가 있음을 확인할 수 있다. (음... 코드가 문제일수도 있다...)

시도한 40 개 chain 이 다 나름의 문제를 가지고 있었다. 멀쩡하게 도는 chain 을 setting 을 하기가 쉽지가 않다.