homework 4 solutions

Exercise 1 first solution

Ex 1 point a

Let $y(x) = \frac{\theta_1 x}{\theta_2 + x}$ and $\sigma = 1$, then:

$$y|\theta_1, \theta_2, x \sim \mathcal{N}(y(x), 1).$$

So, the prior distribution $\pi(\theta_1, \theta_2|x)$ that, given the observation x, the random variables θ_1 and θ_2 are indipendent is:

$$\pi(\theta_1, \theta_2|x) = \pi(\theta_1|x)\pi(\theta_2|x).$$

From the Bayes Theorem we obtain:

$$p(\theta_1, \theta_2|y, x) = \frac{p(y|\theta_1, \theta_2)\pi(\theta_1|x)\pi(\theta_2|x)}{p(y, x)}.$$

We will see three different distributions for the priors of θ_1 and θ_2.

```python
def model(x1, distr : pyro.distributions):
    theta_1 = pyro.sample('theta_1', distr)
    theta_2 = pyro.sample('theta_2', distr)
    y1 =  x1*theta_1/(theta_2+x1)
    y = pyro.sample( 'y', dist.Normal( y1, 1 ) )
    return y

y = torch.tensor([0.053, 0.060, 0.112, 0.105, 0.099, 0.122])
x = torch.tensor([28, 55, 110, 138, 225, 375])

conditioned_model = pyro.condition(model, data={'y': y})
```

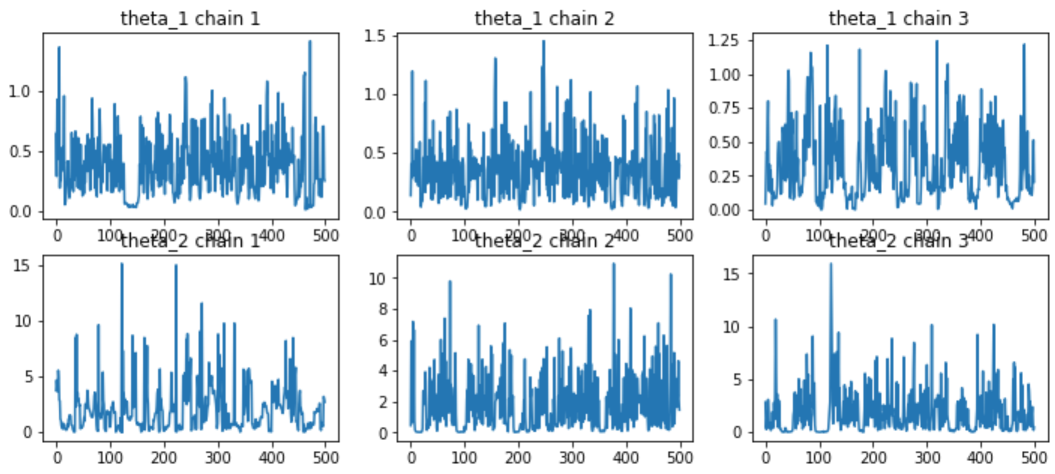
```python
# useful functions
def customed_mcmc(hmc_kernel, samples, steps, chains, data, dist : torch.distributions.Dist
ribution):
    mcmc = MCMC(hmc_kernel, num_samples = samples, warmup_steps = steps, num_chains = chain
s)
    mcmc.run(x1 = data, distr = dist)
    return mcmc

def chains(mcmc):
    mcmc_samples = mcmc.get_samples(group_by_chain = True)
    n_chains = mcmc.num_chains
    n_samples = mcmc.num_samples
    print("chains = ",n_chains, "samples = ", n_samples)

    fig, ax = plt.subplots(len(mcmc_samples), n_chains, figsize = (12,5))
    for i, key in enumerate(mcmc_samples.keys()):
        for j, chain in enumerate(mcmc_samples[key]):
            sns.lineplot(x = range(n_samples), y = chain, ax = ax[i][j])
            ax[i][j].set_title(key + " chain " + str(j + 1))

def posterior(mcmc):
    mcmc_samples = mcmc.get_samples(group_by_chain = True)

    for key in mcmc_samples.keys():
        print("expected", key, "=", mcmc_samples[key].mean().item())

    fig, ax = plt.subplots(1, 2, figsize = (12,5))
    for i, key in enumerate(mcmc_samples.keys()):
        sns.distplot(mcmc_samples[key], ax = ax[i])
        ax[i].set_title("P(" + key + " | y=obs)")
        ax[i].set_xlabel(xlabel = key)
```

Ex 1 points b+c

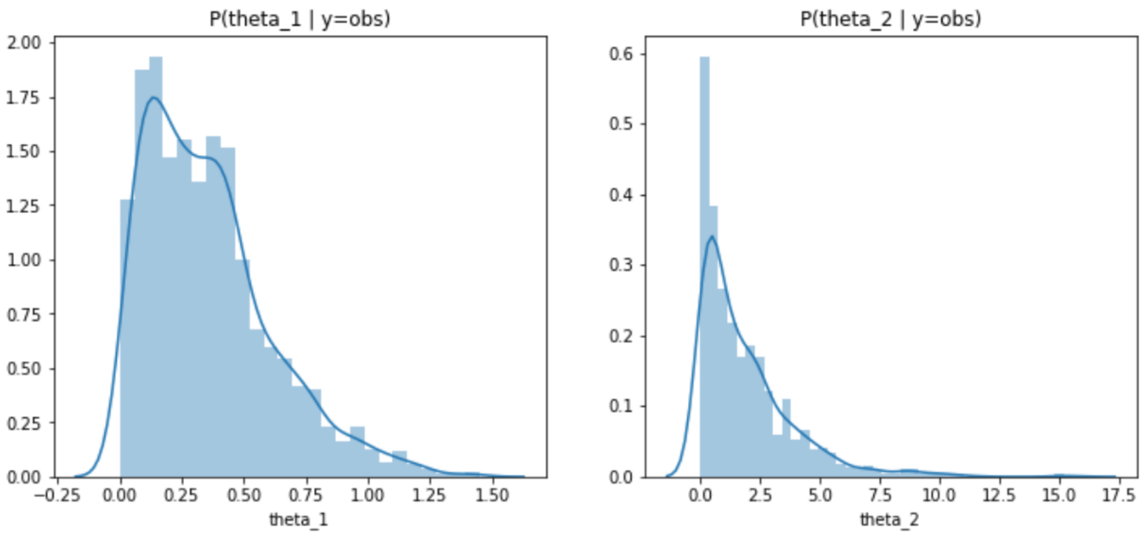
Exponential

First of all, we compute an Exponential distributions with parameter $\lambda = 0.5$.

```
                mean       std    median      5.0%     95.0%     n_eff     r_hat
   theta_1      0.36      0.26      0.31      0.00      0.71    380.38      1.00
   theta_2      1.89      2.01      1.27      0.00      4.46    495.40      1.00

Number of divergences: 0
```



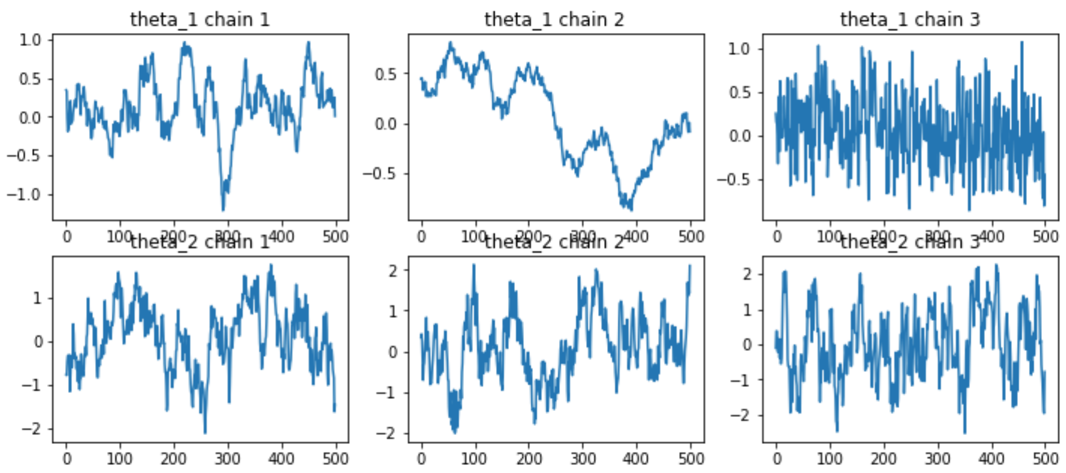
```
expected theta_1 = 0.35643160343170166
expected theta_2 = 1.8913582563400269
```



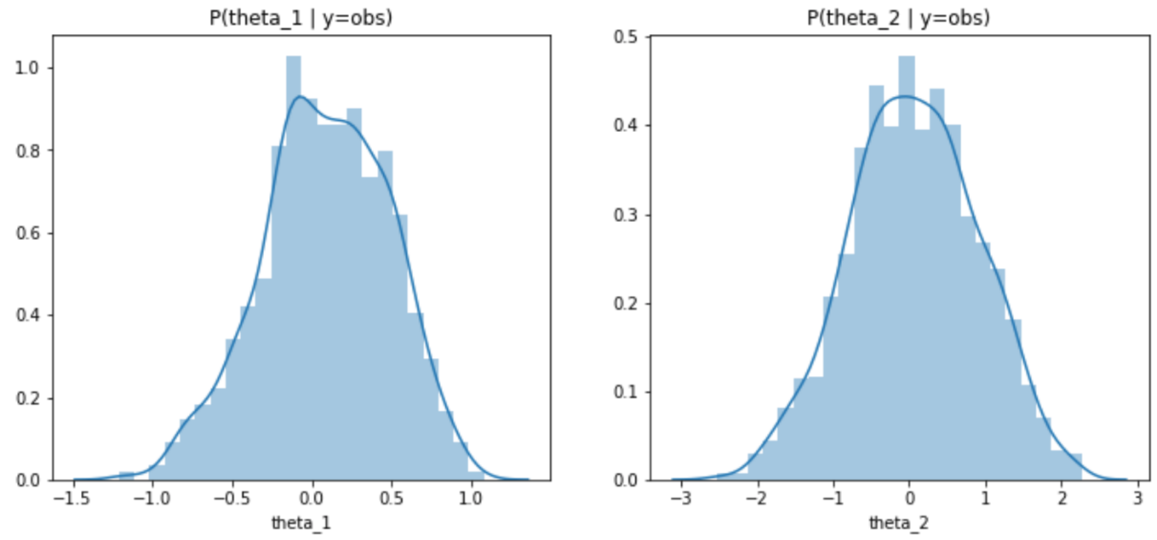
Normal

Then we have studied a Normal distribution, with $\mu = 0$ and $\sigma^2 = 1$.

```
                mean       std    median      5.0%     95.0%     n_eff     r_hat
   theta_1      0.08      0.40      0.09     -0.57      0.72     24.69      1.24
   theta_2      0.07      0.85      0.05     -1.19      1.58     95.04      1.03

Number of divergences: 0
```

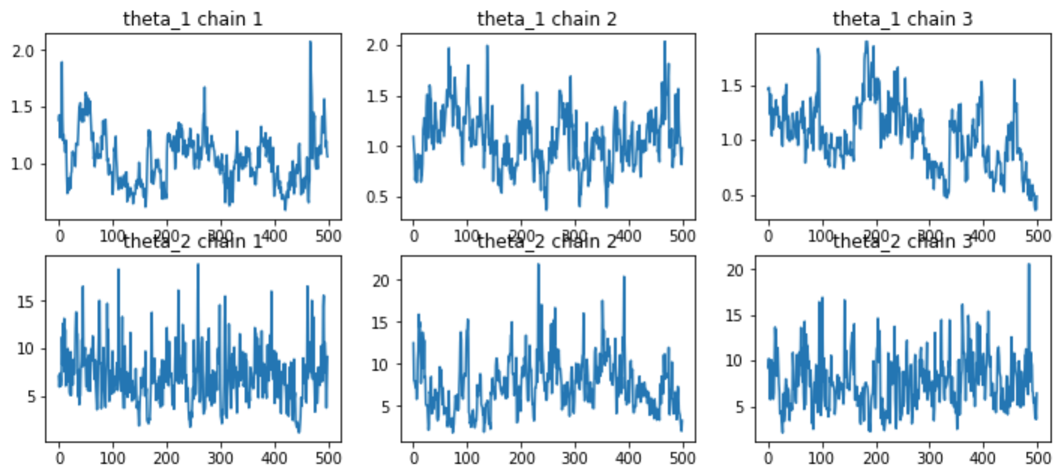


```
expected theta_1 = 0.08452938497066498
expected theta_2 = 0.06543359905481339
```

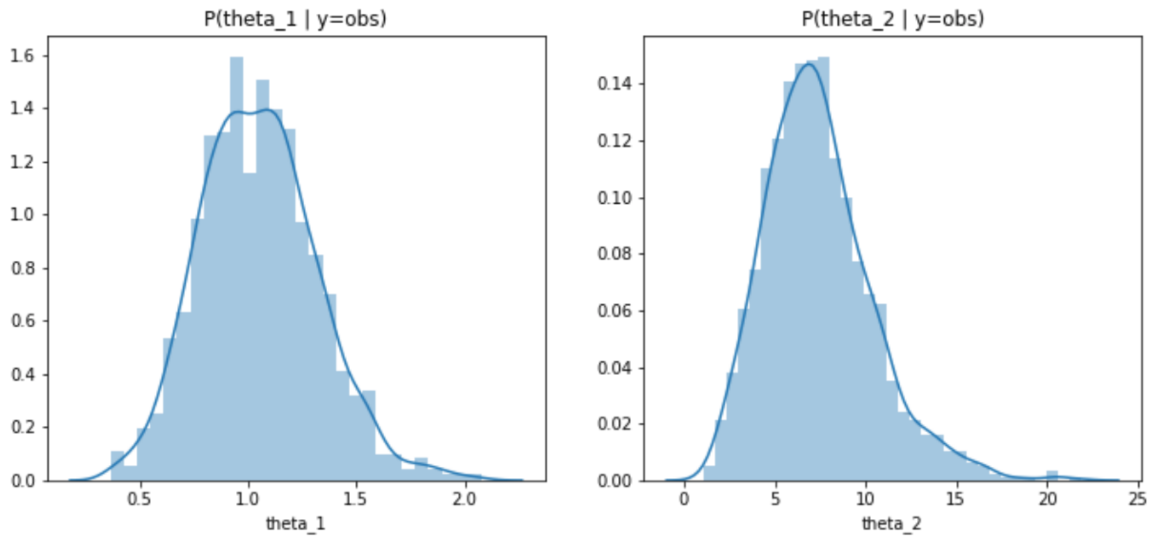


Gamma
At last, we have tried a Gamma distribution with parameters $(\alpha, \beta) = (5, 1)$.

```
                mean       std    median      5.0%     95.0%     n_eff     r_hat
     theta_1    1.05      0.27      1.04      0.60      1.46     59.80      1.06
     theta_2    7.40      2.96      7.12      2.41     11.54    220.52      1.01

Number of divergences: 0
```



```
expected theta_1 = 1.0495789051055908
expected theta_2 = 7.40355920791626
```



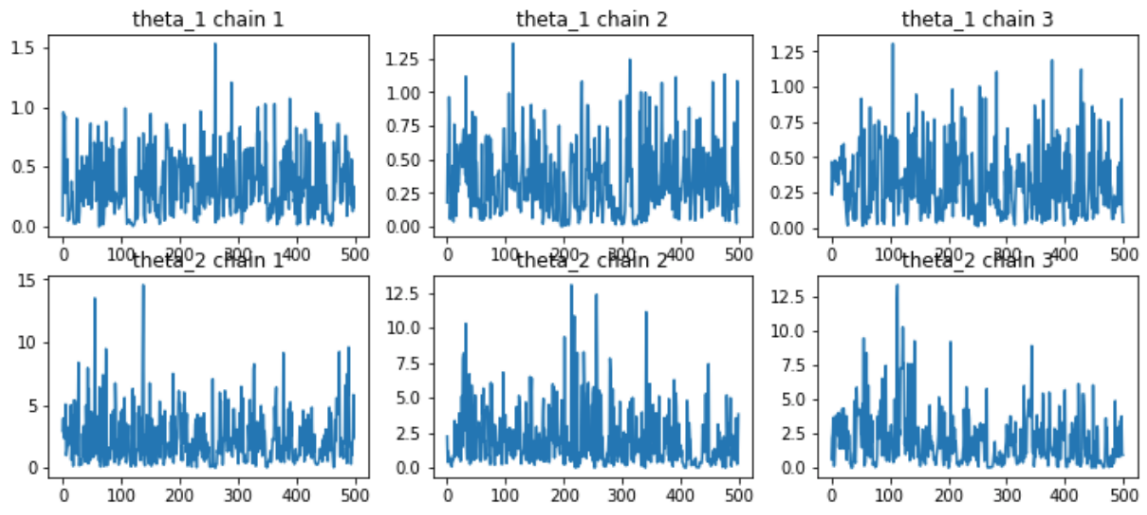
Conclusion

We can observe that for Exponential distribution the n_{eff} is really higher and $\hat{R}$ is closer to 1, with respect to other tests. We can conclude that Exponential distribution with parameters $\lambda = 0.5$ is more adeguate in order to evaluate the proposal distributions of θ_1 and θ_2, than other distributions.

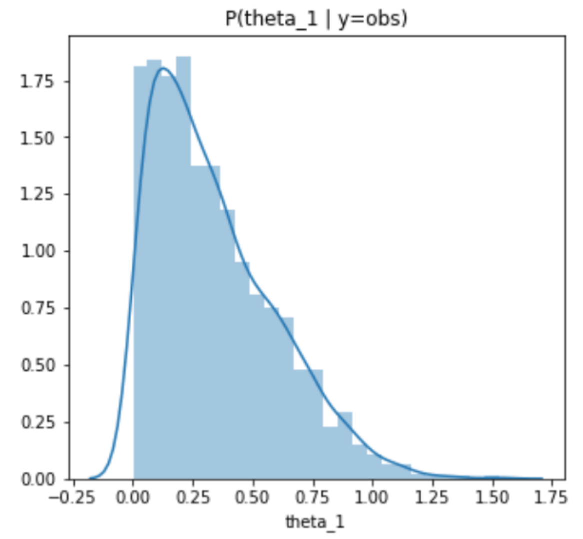
Exercise 1 with NUTS

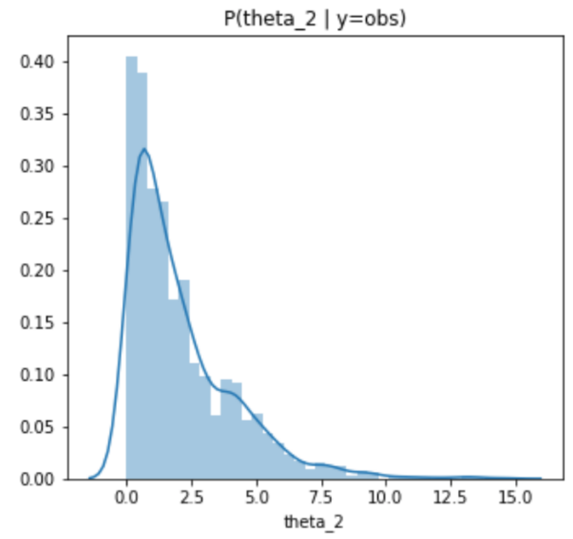
```
nuts_exp = customed_mcmc( NUTS(model=conditioned_model), 500, 1000, 3, x, dist.Exponential
(0.5) )
chains(nuts_exp)
nuts_exp.summary()
```

```
                mean       std    median      5.0%     95.0%     n_eff     r_hat
   theta_1      0.34      0.26      0.29      0.00      0.71    726.61      1.00
   theta_2      2.12      2.06      1.46      0.00      4.95    504.16      1.01

Number of divergences: 0
```



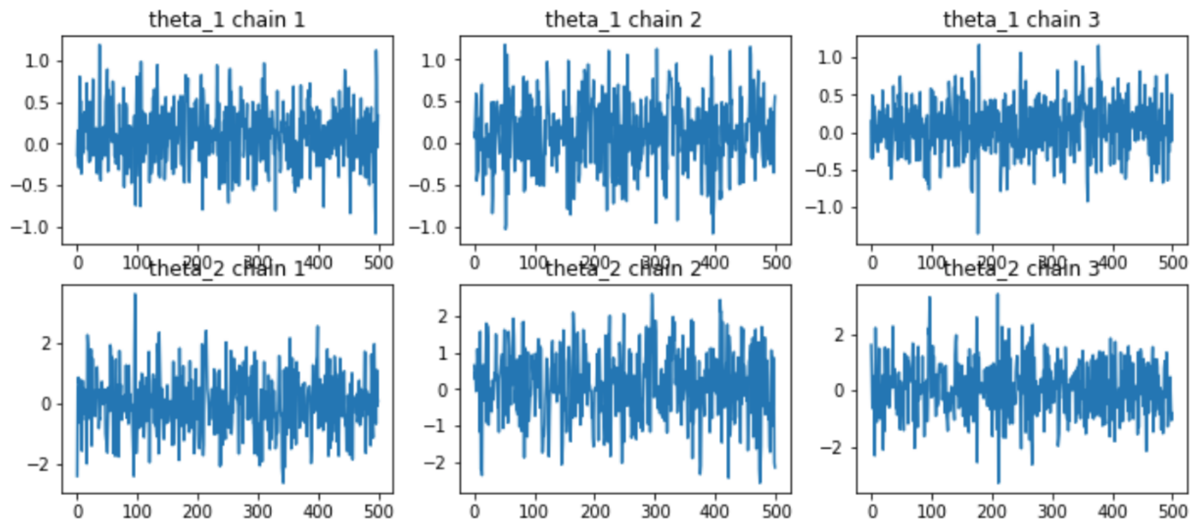
```
expected theta_1 = 0.34177327156066895
expected theta_2 = 2.1221184730529785
```



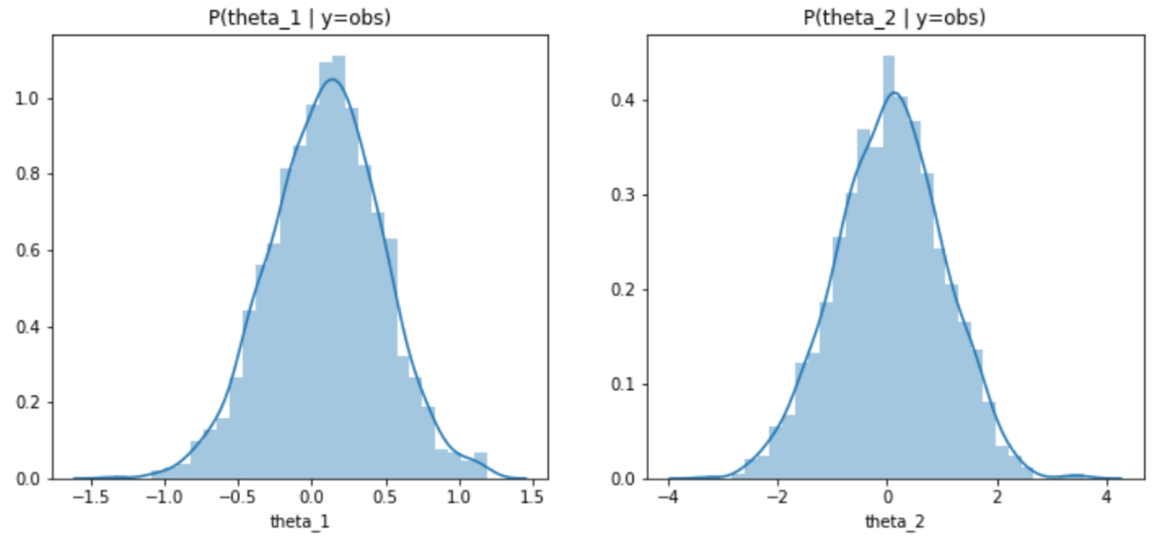


```
nuts_norm = customed_mcmc( NUTS(model=conditioned_model), 500, 1000, 3, x, dist.Normal(0,1)
)
chains(nuts_norm)
nuts_norm.summary()
```

```
chains =  3 samples =  500

                mean       std    median      5.0%     95.0%     n_eff     r_hat
    theta_1     0.09      0.38      0.11     -0.53      0.70   1337.59      1.00
    theta_2     0.05      0.98      0.07     -1.48      1.74   1383.78      1.00

Number of divergences: 0
```

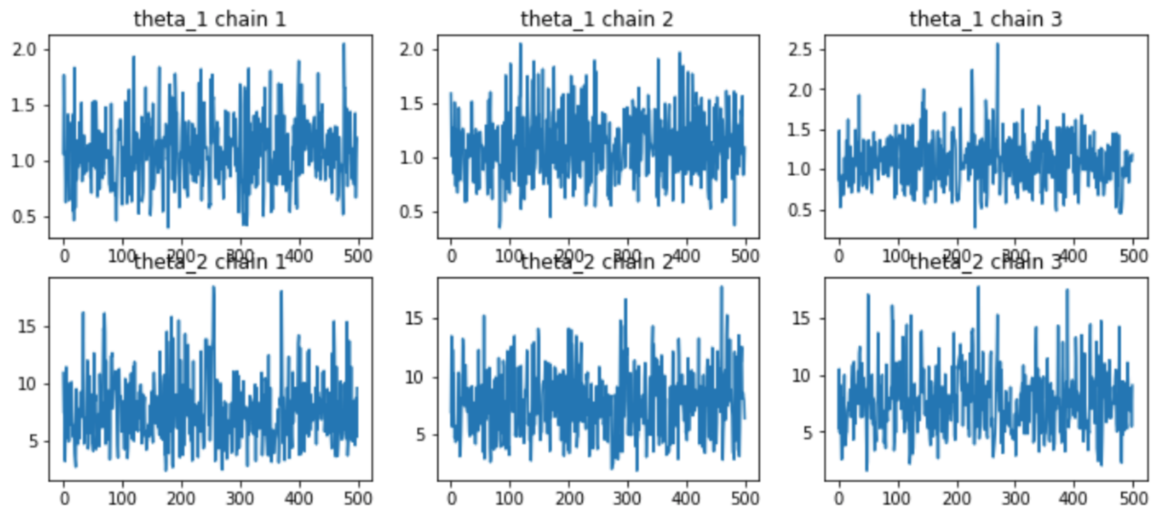


```
expected theta_1 = 0.09414684027433395
expected theta_2 = 0.053859956562519073
```

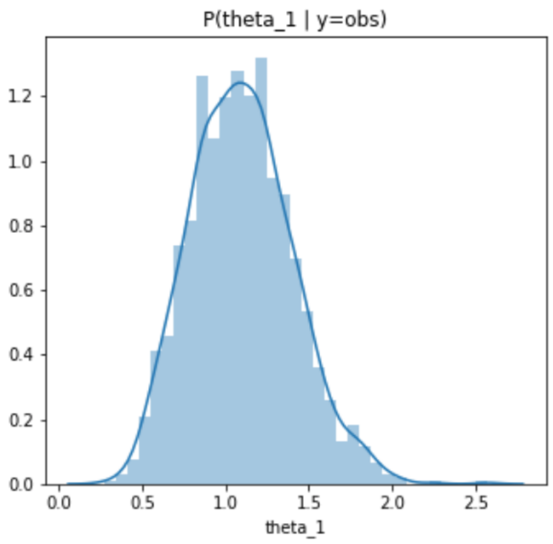


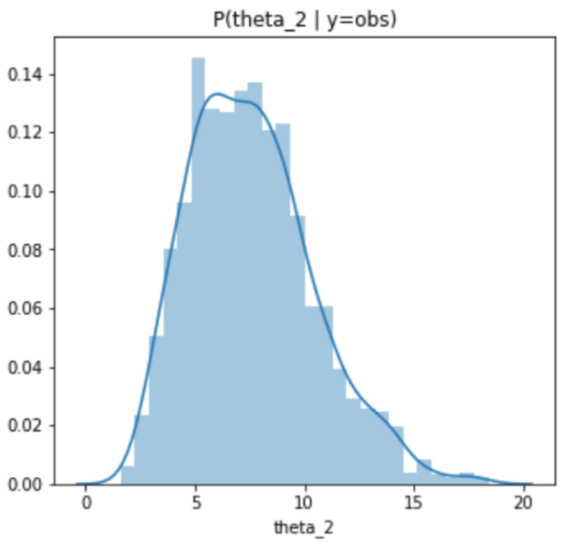
```
nuts_gamma = customed_mcmc( NUTS(model=conditioned_model), 500, 1000, 3, x, dist.Gamma(7,1)
)
chains(nuts_gamma)
nuts_gamma.summary()
```

```
chains =  3 samples =  500

                mean       std    median      5.0%     95.0%     n_eff     r_hat
   theta_1      1.10      0.31      1.08      0.59      1.57   1197.98      1.00
   theta_2      7.58      2.86      7.32      2.86     11.88    993.77      1.00

Number of divergences: 0
```



```
expected theta_1 = 1.0975621938705444
expected theta_2 = 7.578394412994385
```





Conclusion

NUTS provides better results for all three tested distribution (we kept the same parameters): n_{eff} is significativily higher and $\hat{R}$ is 1 for all the results. Despite to the previous conclusion, we can observe that proposal distributions of θ_1 and θ_2 are better estimated by Normal distribution with $\mu = 0$ and $\sigma^2 = 1$.

Exercise 1 second solution

Ex 1 point a

In this exercise, starting from the model:

$$y \sim \mathcal{N}(\hat{y}, 1),\ \hat{y} = \frac{\theta_1}{\theta_2 + x}$$

and from the priors on θ_1 and θ_2, the posterior distributions of θ_1 and θ_2 can be computed, conditioning on the observations:

$$x = (28, 55, 110, 138, 225, 375),\ \ y = (0.053, 0.060, 0.112, 0.105, 0.099, 0.122)$$

This is done through Hamiltonian Monte Carlo, sampling from 3 independent chains and taking `num_samples=3000` samples, after `warmup_steps=1000` discarded samples.

```
x_obs=torch.tensor([28,55,110,138,225,375])
y_obs=torch.tensor([0.053,0.060,0.112,0.105,0.099,0.122])

def conditioned_model(t1,t2,obs,x=x_obs):
    theta1=pyro.sample("theta1",t1)
    theta2=pyro.sample("theta2",t2)
    yhat = (theta1*x)/(theta2+x)
    with pyro.plate(len(obs)):
        y = pyro.sample("y", dist.Normal(yhat,1), obs=obs)
    return y
hmc_kernel = HMC(model=conditioned_model)
def hmc_plot(mcmc):
    mcmc.summary()
    mcmc_samples = mcmc.get_samples()
    sns.distplot(mcmc_samples['theta1'])
    plt.title("P( theta1 | y=y_obs, x=x_obs )")
    plt.xlabel("theta1")
    plt.show()
    sns.distplot(mcmc_samples['theta2'])
    plt.title("P( theta2 | y=y_obs, x=x_obs )")
    plt.xlabel("theta2")
    plt.show()
```

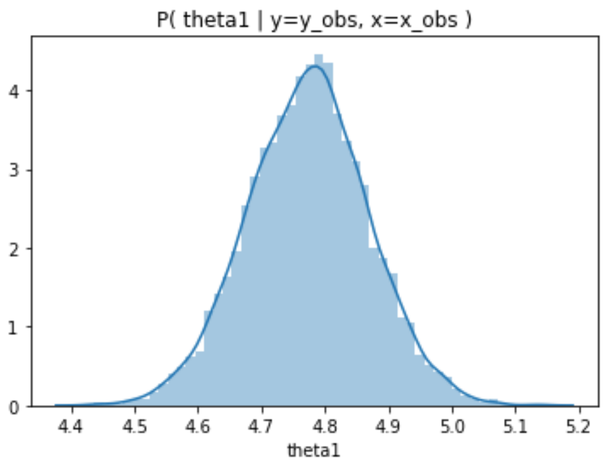
Ex 1 points b+c

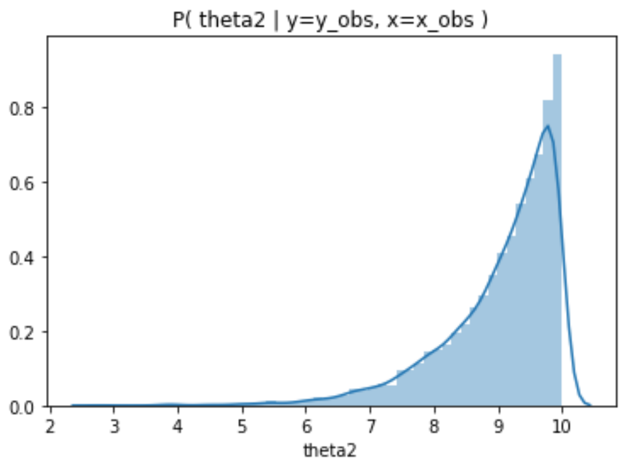
The first and most obvious choice that influences the HMC procedure is the one on the prior distributions: for example, choosing for θ_1 a prior normal distribution with a very low variance and for θ_2 an uninformative prior distribution (for example a uniform over a very broad interval), a significant modification of θ_2 can be seen. In fact, its distribution tends to accumulate nearby one border of the considered interval, meaning that the initial choice of the prior did not suit data particularly well.
On the other hand, choosing normal distributions with reasonable means and variances results in posteriors which are more similar to priors, being still normally distributed, but shifted and rescaled in terms of means and variances.
The main algorithmic parameters that influence the estimates obtained with HMC are the number of samples to draw and the number of warmup steps. Both these values should be large enough to ensure good estimates: in particular, a value for `warmup_steps` lower than 100 leads to a $\hat{R}$ statistic higher than 1.3, which is symptom of a very bad convergence in the MCMC process (reference), while a low number of samples `num_samples` may result in very distorted posterior distributions.
The chosen parameters of HMC (`num_samples=3000` and `warmup_steps=1000`) ensure a good convergence as shown by the $\hat{R}$ value which is exactly 1 for both θ_1 and θ_2.

```
def cov(xsamples,ysamples,xmean,ymean):
    cov=0
    for i in range(len(xsamples)):
        cov+=(xsamples[i]-xmean)*(ysamples[i]-ymean)
    return cov/len(xsamples)
print("First run with low-variance normal and uniform priors")
mcmc = MCMC(hmc_kernel, num_samples=3000, warmup_steps=1000, num_chains=3)
posterior_uninformativeprior = mcmc.run(t1=dist.Normal(5,0.1),t2=dist.Uniform(-10,10),obs=y
_obs)
hmc_plot(mcmc)

print("Second run with normal priors")
mcmc = MCMC(hmc_kernel, num_samples=3000, warmup_steps=1000, num_chains=3)
posterior_gaussian = mcmc.run(t1=dist.Normal(10,1),t2=dist.Normal(4,5),obs=y_obs)
hmc_plot(mcmc)
```

```
First run with low-variance normal and uniform priors
```

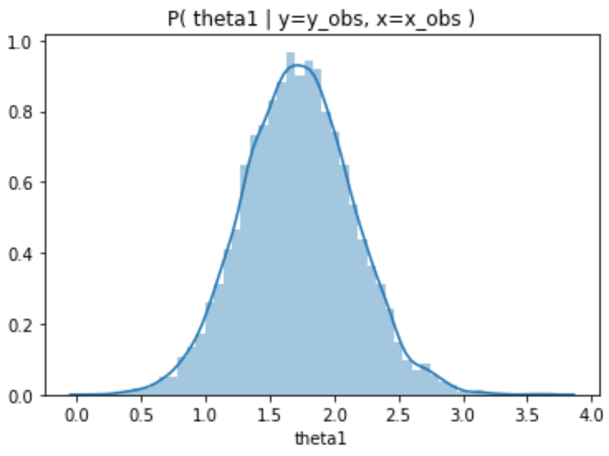
```
              mean       std    median      5.0%     95.0%     n_eff     r_hat
theta1        4.77      0.10      4.77      4.62      4.93   1115.91      1.00
theta2        9.05      0.89      9.31      7.83     10.00    932.10      1.00

Number of divergences: 0
```

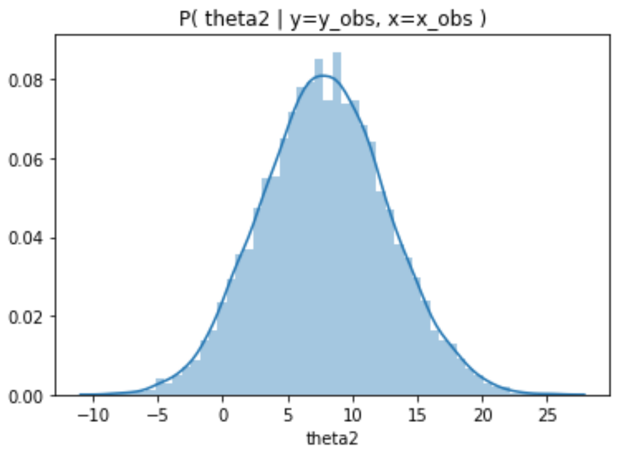




Second run with normal priors

```
              mean       std    median      5.0%     95.0%     n_eff     r_hat
theta1        1.72      0.43      1.72      1.02      2.41   1687.12      1.00
theta2        7.88      4.87      7.85      0.01     15.97   1732.40      1.00

Number of divergences: 0
```





Exercise 2

Ex 2 point a

Using the properties of the multivariate normal distribution

$$x_1|x_2,\ x \sim N(\rho x_2,\ 1-\rho^2) \sim \rho x_2 + \sqrt{1-\rho^2}\,N(0,\ 1)$$

and

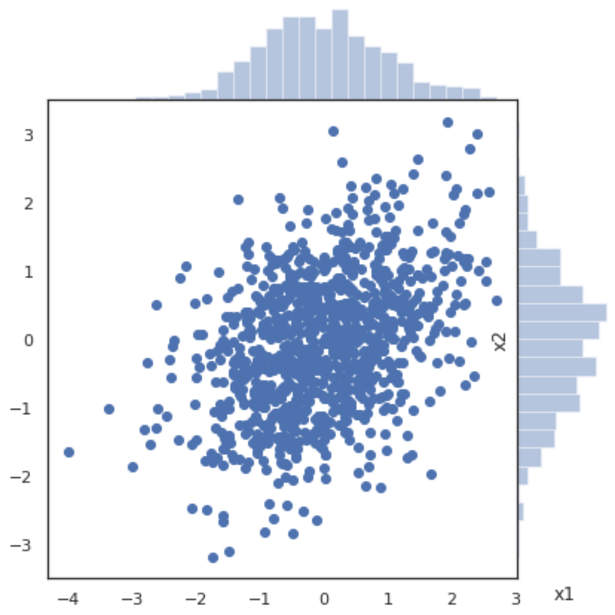
$$x_2|x_1,\ x \sim N(\rho x_1,\ 1-\rho^2) \sim \rho x_1 + \sqrt{1-\rho^2}\,N(0,\ 1)$$

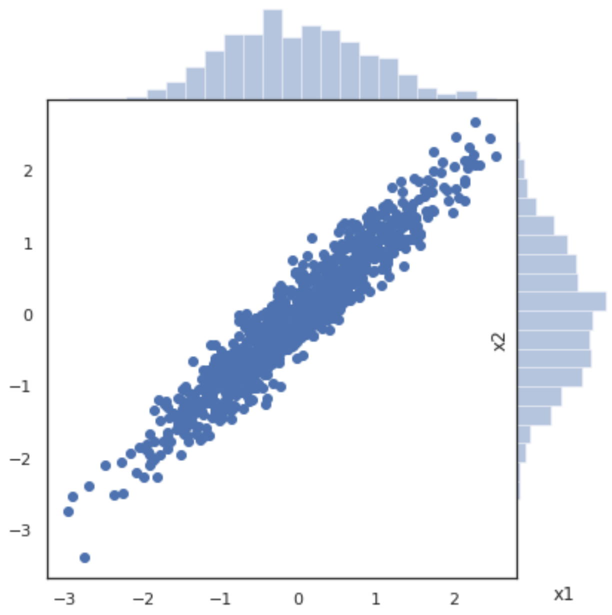
Then, given ρ we can implement our `gibbs` function.

```python
def gibbs(rho, iters, warmup):
    x1 = torch.zeros(warmup + iters, 1)
    x2 = torch.zeros(warmup + iters, 1)

    x2[0] = pyro.sample('x2', dist.Normal(0,1))
    x1[0] = pyro.sample('x1', dist.Normal(rho*x2[0].item(), np.sqrt(1-rho**2)))

    for i in range(1,warmup+iters-1):
        # draw x2 given x1
        x2[i] = pyro.sample('x2', dist.Normal(rho*x1[i-1].item(), np.sqrt(1-rho**2)))
        # draw x1 given x2
        x1[i] = pyro.sample('x1', dist.Normal(rho*x2[i].item(), np.sqrt(1-rho**2)))

    x1 = x1[warmup:iters]
    x2 = x2[warmup:iters]

    return x1, x2
```

```python
# Gibbs sampling with rho = 0.4
plt.figure(figsize=(12,5))

x1, x2 = gibbs(0.4, 1500, 500)

sns.set(style="white")
sns.jointplot(x1, x2, space=0, color="b");

plt.xlabel("x1", fontsize=12)
plt.ylabel("x2", fontsize=12)
plt.show()
```

```python
# Gibbs sampling with rho = 0.96
plt.figure(figsize=(12,5))

x1, x2 = gibbs(0.96, 1500, 500)

sns.set(style="white")
sns.jointplot(x1, x2, space=0, color="b");

plt.xlabel("x1", fontsize=12)
plt.ylabel("x2", fontsize=12)
plt.show()
```





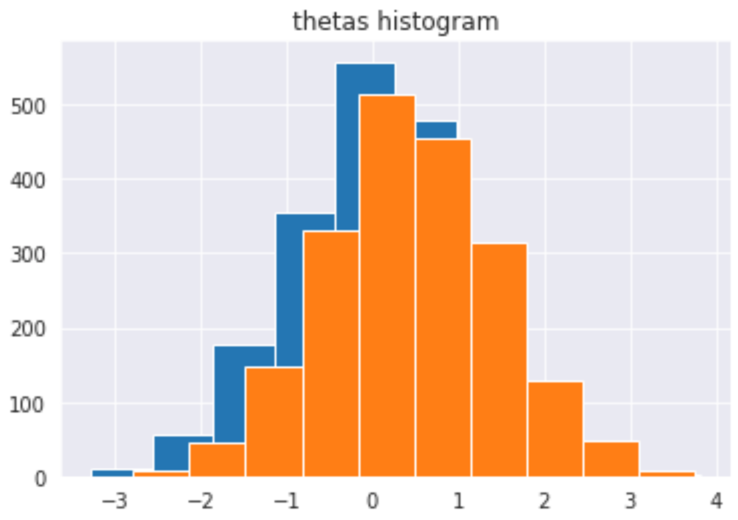
Ex 2 point b

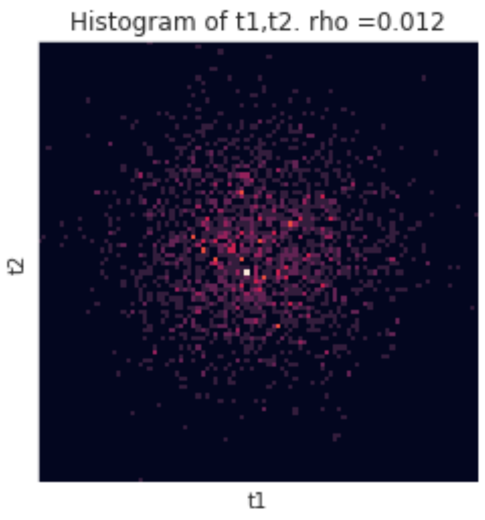
```
def new_mu(idx, mu, rho, x_other):
    return mu[idx] + rho * (x_other - mu[(idx + 1) % 2])

def gibbs_sampler(guess: float, iters: int, warmup: int, rho: float, mu = [0,0]):

    x = [0.,guess]
                                 #p(x1|x2)
    x[0] = pyro.sample('x10',dist.Normal(new_mu(0,mu,rho,x[1]), np.sqrt(1-rho**2)))

    for i in range(warmup):
        x[1] = pyro.sample('x2',dist.Normal(new_mu(1,mu,rho,x[0]), np.sqrt(1-rho**2)))
        x[0] = pyro.sample('x10',dist.Normal(new_mu(0,mu,rho,x[1]), np.sqrt(1-rho**2)))

    x2_list=[]
    x1_list=[]
    for i in range(iters):
        x[1] = pyro.sample('x2',dist.Normal(new_mu(1,mu,rho,x[0]), np.sqrt(1-rho**2)))
        x[0] = pyro.sample('x10',dist.Normal(new_mu(0,mu,rho,x[1]), np.sqrt(1-rho**2)))
        x2_list.append(x[1])
        x1_list.append(x[0])
    return (x1_list, x2_list)
```

```
t1_sample = mcmc.get_samples()['theta1']
t2_sample = mcmc.get_samples()['theta2']

mu = [float(t1_sample.mean()), float(t2_sample.mean())]
print("Mean vector: " + str(mu))

t1_sample = np.array(t1_sample)
t2_sample = np.array(t2_sample)

sigma = np.corrcoef(np.array([t1_sample,t2_sample]))
print("Covariance matrix flattened:")
print(sigma.flatten())

estimated_rho = sigma[0,1]
```

```
Mean vector: [0.0651819184422493, 0.49141475558280945]
Covariance matrix flattened:
[1.         0.01223405 0.01223405 1.        ]
```

```
t1_g, t2_g = gibbs_sampler(0.1, 2000, 1000, estimated_rho, mu)
plt.hist(t1_g)
plt.title("thetas histogram")
plt.hist(t2_g)
plt.show()

heatmap, xedges, yedges = np.histogram2d(t1_g, t2_g, bins=100)
plt.clf()
plt.title('Histogram of t1,t2. rho =' + str(round(estimated_rho,3)))
plt.ylabel('t2')
plt.xlabel('t1')
plt.xticks(())
plt.yticks(())
plt.imshow(heatmap)
plt.show()
```


thetas histogram
500
400
300
200
100
0
−3
−2
−1
0
1
2
3
4


Histogram of t1,t2. rho =0.012
t2
t1