homework_04

April 21, 2020

0.0.1 Exercise 1

Let θ_1 and θ_2 be real valued parameters of the model

$$y = \frac{\theta_1 x}{\theta_2 + x}.$$

a. Choose two suitable prior distributions for θ_1 and θ_2 and use HMC algorithm to find their posterior distributions, conditioning on the observations

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

[163]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.style as style
style.use('ggplot')
import seaborn as sns
import torch
import pyro
import pyro.distributions as dist
from pyro.infer.mcmc import MCMC, HMC, NUTS
import pyro.poutine as poutine
np.set_printoptions(precision=4, suppress=True)
```

[164]:
```
def model(x, d: torch.distributions.Distribution):

    pyro.clear_param_store()
    # Suitable prior distribution with mean 0 and variance 1
    theta1 = pyro.sample("theta1", d)
    theta2 = pyro.sample("theta2", d)

    # Calculated y variable
    yhat = (theta1*obs_x) / (theta2+obs_x)
    y = pyro.sample("y", dist.Normal(yhat,1))
    print("theta1: {}, theta2: {}\nx: {}\ny: {}".format(theta1,theta2,x,y))

    return y
```

```
def conditioned_model(y, x, d: torch.distributions.Distribution):

    pyro.clear_param_store()
    # Suitable prior distribution with mean 0 and variance 1
    theta1 = pyro.sample("theta1", d)
    theta2 = pyro.sample("theta2", d)


    # Calculated y variable
    yhat = (theta1*obs_x) / (theta2+obs_x)
    y = pyro.sample("y", dist.Normal(yhat,1), obs=y)
    return y

#model(obs_x, dist.Normal(0,1))
#conditioned_model(obs_y,obs_x,dist.Normal(0,1))
```

[165]:
```
# @param model conditioned pyro model
# @param obs_y observations of y value
# @param obs_x observation of x value
# @param dist prior dist of theta1 and theta2
# @param num_sample number of sample from stationary distribution
# @param warmup burn-in period, number of discarded samples before performing␣
↪the actual sampling
# @param chain number of independent MCMC runs
# @return Progress bars of chains, summary of mcmc, lineplots of chains, dist␣
↪plot of posterior
# P.S. This functions is written not to have code duplicates
# for trying different parameters for mcmc algortihm and prior dist, that is␣
↪why it is not that much generalize

def analyze_hmc(model,obs_y,obs_x, d:torch.distributions.Distribution,␣
↪num_sample, warmup, chain):

    '''This function runs HMC algorithm with given parameters
    and plot the chain results and posterior distribution'''

    hmc_kernel = HMC(model=model) # transition kernel
    mcmc = MCMC(hmc_kernel, num_samples=num_sample, warmup_steps=warmup,␣
↪num_chains=chain)

    # posteriors
    posterior = mcmc.run(y = obs_y,x = obs_x, d=d)

    # dictionary of sampled values
    print("Keys:",mcmc.get_samples().keys(), "\n")

    print("Summary of MCMC:\n")
```

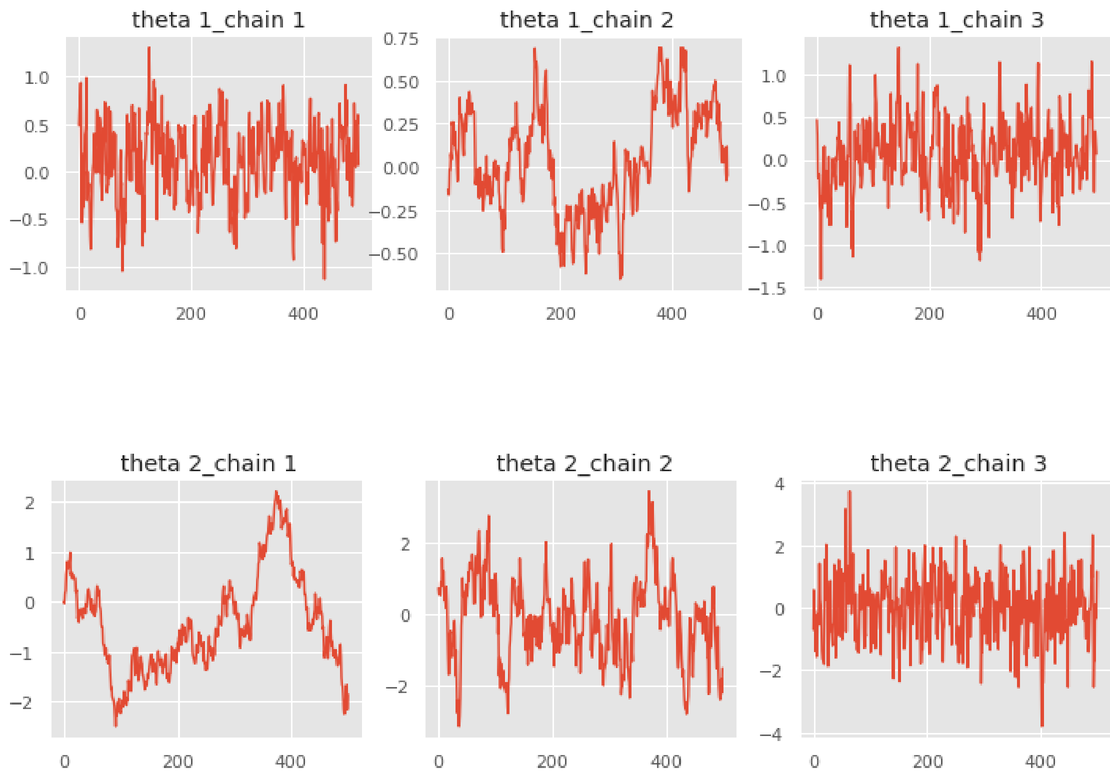
```
    mcmc.summary()

    mcmc_samples = mcmc.get_samples(group_by_chain=True)

    chains_1 = mcmc_samples["theta1"]
    chains_2 = mcmc_samples["theta2"]

    n_chains_1, n_samples_1 = chains_1.shape
    n_chains_2, n_samples_2 = chains_2.shape

    print("Shape of chain_1 {} and chain_2 {}\n".format(chains_1.shape,␣
↪chains_2.shape))

    fig, ax = plt.subplots(1, n_chains_1, figsize=(12,3))
    for i, chain in enumerate(chains_1):
        sns.lineplot(x=range(n_samples_1), y=chain, ax=ax[i])
        ax[i].set_title("theta 1_chain "+str(i+1))

    fig, ax = plt.subplots(1, n_chains_2, figsize=(12,3))
    for i, chain in enumerate(chains_2):
        sns.lineplot(x=range(n_samples_2), y=chain, ax=ax[i])
        ax[i].set_title(" theta 2_chain "+str(i+1))


    print("expected theta1 =", mcmc_samples['theta1'].mean().item())
    print("expected theta2 =", mcmc_samples['theta2'].mean().item())

    fig,a = plt.subplots(1,2, figsize=(14,4))

    sns.distplot(mcmc_samples['theta1'], ax = a[0])
    a[0].set_title("P(theta1 | x = obs_x, y = obs_y)")
    a[0].set_xlabel("Theta1")

    sns.distplot(mcmc_samples['theta2'], ax = a[1])
    a[1].set_title("P(theta2 | x = obs_x, y = obs_y)")
    a[1].set_xlabel("Theta2")
```

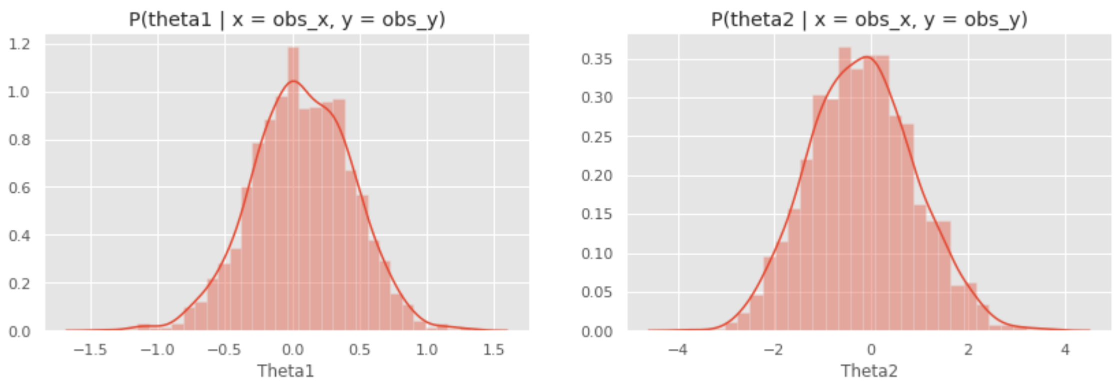
- Chosen prior distributions for θ_1 and θ_2 are standard normal distribution. Chosen parameters for HMC are `num_samples = 500`,`warmup_steps = 1000` and `num_chains = 3`.

[166]:
```
# observations
obs_y = torch.tensor([0.053,0.060,0.112,0.105,0.099,0.122])
obs_x = torch.tensor([(28,55,110,138,225,375)])
```

[167]:
```
analyze_hmc(conditioned_model,obs_y,obs_x,dist.
↪Normal(0,1),num_sample=500,warmup=1000,chain=3)
```

```
HBox(children=(FloatProgress(value=0.0, description='Warmup [1]', max=1500.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [2]', max=1500.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [3]', max=1500.0, style=ProgressSty



Keys: dict_keys(['theta1', 'theta2'])

Summary of MCMC:


                mean       std    median      5.0%     95.0%     n_eff     r_hat
    theta1      0.07      0.37      0.06     -0.50      0.69    104.04      1.01
    theta2     -0.19      1.08     -0.20     -1.95      1.58     55.27      1.06

Number of divergences: 0
Shape of chain_1 torch.Size([3, 500]) and chain_2 torch.Size([3, 500])

expected theta1 = 0.07132413983345032
expected theta2 = -0.19076305627822876
```





- There is no divergence in our hmc summary results but `n_eff` values are low and `r_hat` values are above 1 so we can say that some chains have not fully converged and above plots showed us that comment that is made before for `mcmc.summary` is true. Most of the chains are not consistent apart from theta1_chain 3. Also you can find the distributon of $P(\theta_1|x, y)$ and $P(\theta_2|x, y)$ as a result of MCMC algorithm

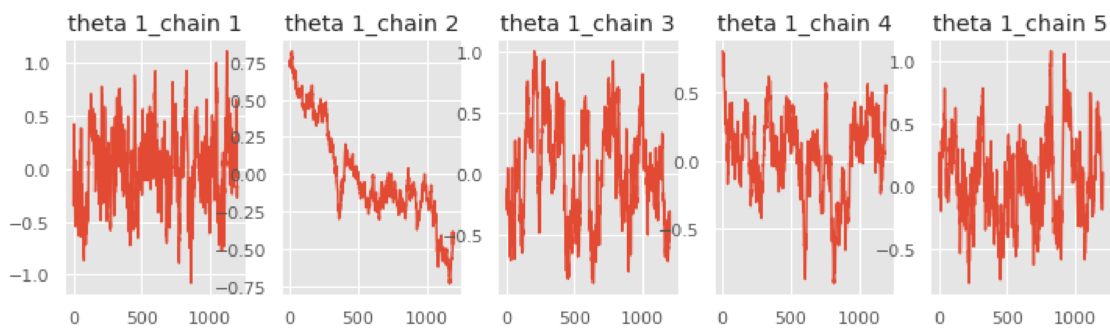
b. Discuss how different parameters for both priors and the HMC algorithm lead to different estimates.

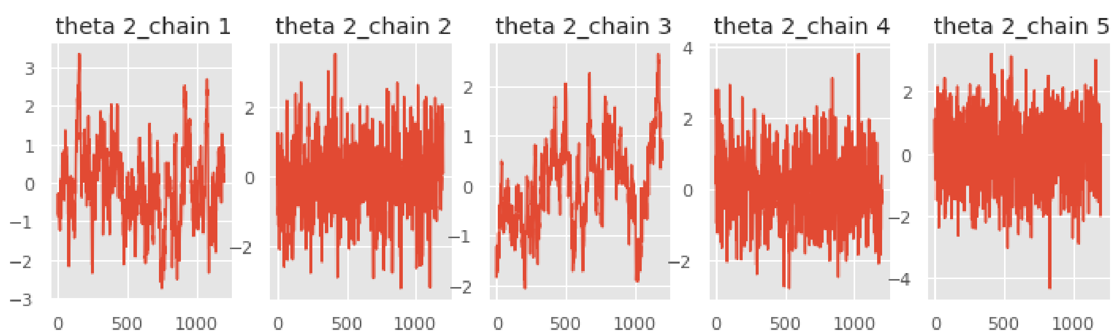
- Changing the parameters HMC algorithm may lead better estimates for posterior distribution. Especially increasing number of samples and number of chains may lead better estimations.
- Therefore parameters of mcmc algorithm are increased such as: `num_samples = 1200`,`warmup_steps = 400` and `num_chains = 5`.

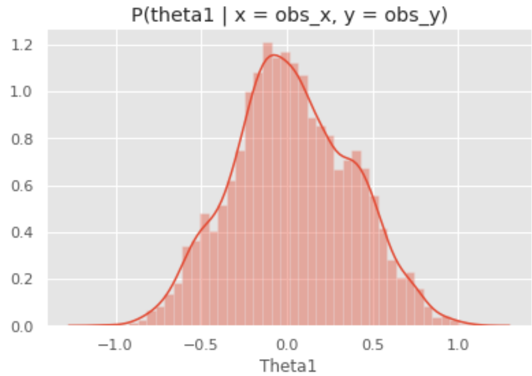
[168]:
```
analyze_hmc(conditioned_model,obs_y,obs_x,dist.Normal(0,1),1200, 400, 5)
```

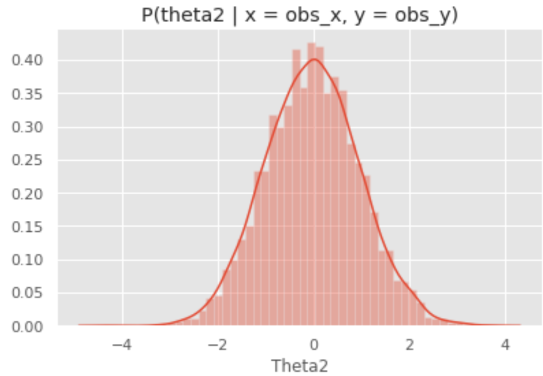
```
HBox(children=(FloatProgress(value=0.0, description='Warmup [1]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [2]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [3]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [4]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [5]', max=1600.0, style=ProgressSty
```

```
Keys: dict_keys(['theta1', 'theta2'])

Summary of MCMC:

                mean       std    median      5.0%     95.0%     n_eff     r_hat
    theta1      0.03      0.35      0.01     -0.58      0.57    106.30      1.06
    theta2     -0.02      0.98     -0.02     -1.61      1.61    398.26      1.01

Number of divergences: 0
Shape of chain_1 torch.Size([5, 1200]) and chain_2 torch.Size([5, 1200])

expected theta1 = 0.030372297391295433
expected theta2 = -0.019049106165766716
```









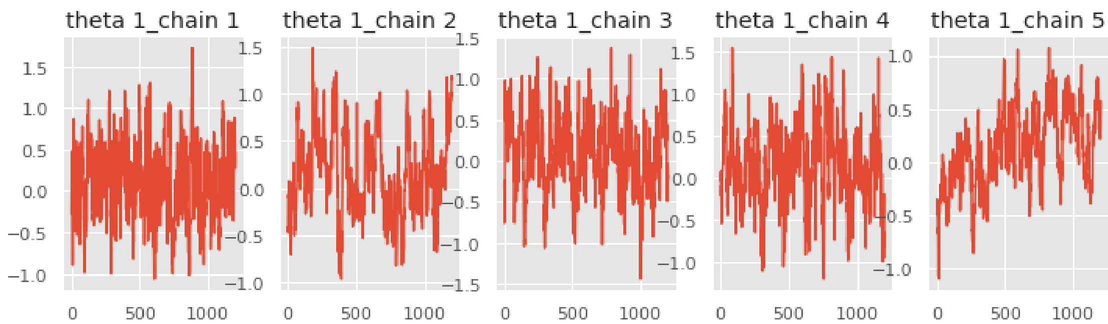
- For θ_1, we have high `n_eff` and close value of `r_hat` to 1. And most of the chains are consistent and converged just the second one seems diverged. So we can say that we have good results for θ_1. However for θ_2 even though we can say that result are quite better than previous ones, it is not as good as θ_1.
- Changing the parameters of prior distribution may lead better results. Therefore same analyze will be done for $\theta_k \sim \mathcal{N}(5, 10)$ where $k = 1, 2$

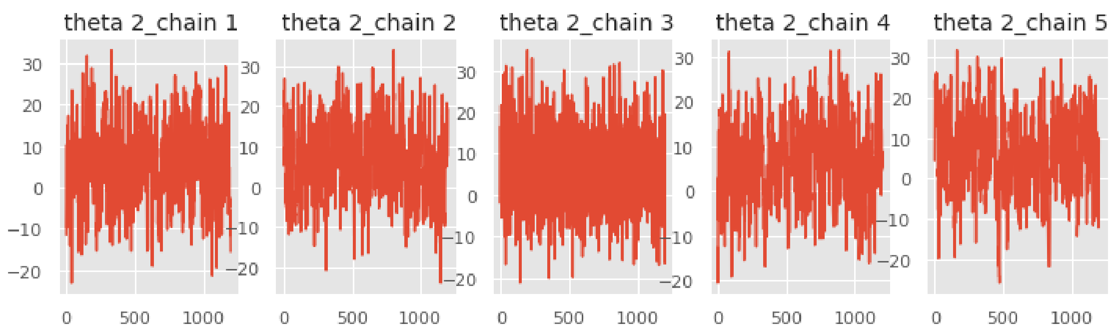
[169]:
```
analyze_hmc(conditioned_model,obs_y,obs_x,dist.Normal(5,10),1200, 400, 5)
```

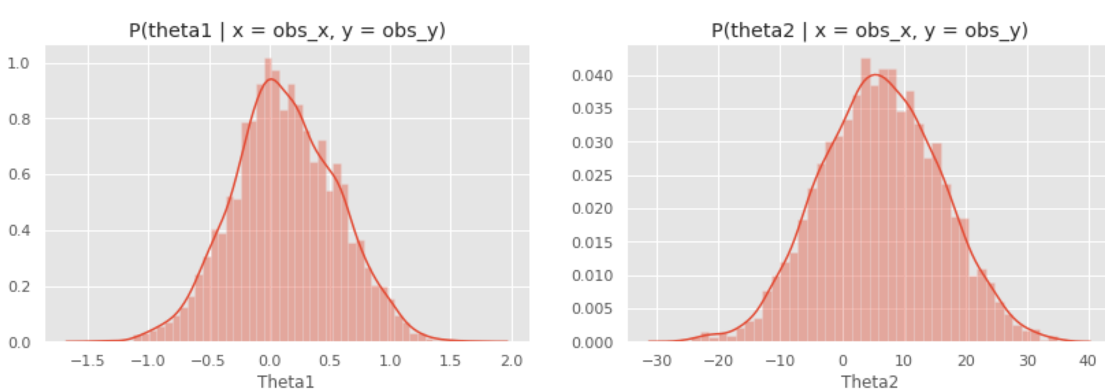
```
HBox(children=(FloatProgress(value=0.0, description='Warmup [1]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [2]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [3]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [4]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [5]', max=1600.0, style=ProgressSty
```

```
Keys: dict_keys(['theta1', 'theta2'])

Summary of MCMC:

                mean       std    median      5.0%     95.0%     n_eff     r_hat
    theta1      0.13      0.43      0.11     -0.56      0.85    329.08      1.02
```

```
    theta2      6.20      9.65      6.15    -10.74     21.00    1036.69      1.00

Number of divergences: 20
Shape of chain_1 torch.Size([5, 1200]) and chain_2 torch.Size([5, 1200])

expected theta1 = 0.12991181015968323
expected theta2 = 6.203857898712158
```







According to results (particularly checking `n_eff`, `r_hat` and plots), we have more consistent result for theta2 this time. But it is worth to point out that theta values have higher standard deviation this time.

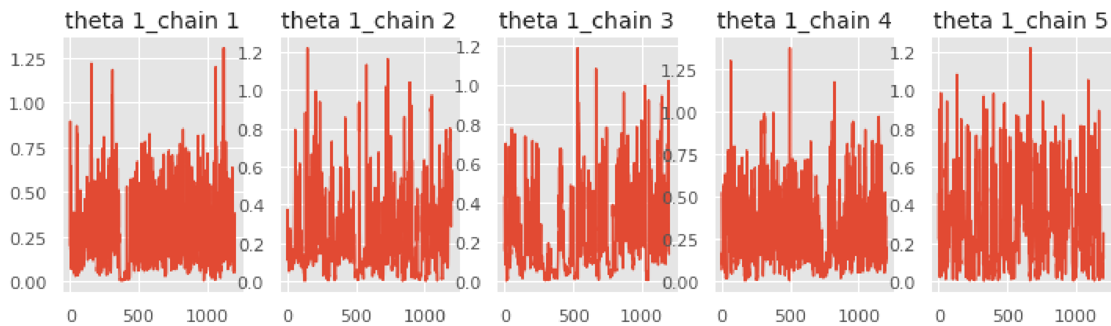
c. Plot the most reliable posterior distributions, according to convergence checks on the traces.

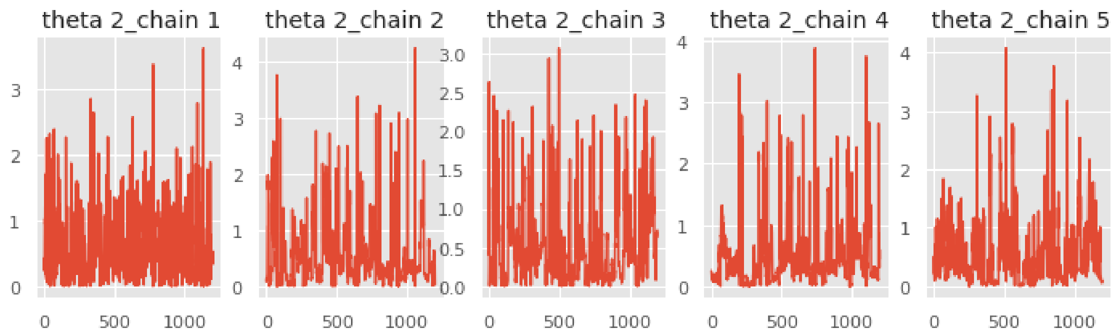
- After trying different prior distributions like Normal, Exponential, Half-Cauchy and Inverse-Gamma, Exponential distribution gave the best result according to `n_eff`, `r_hat` and plots. you can find the results below. Of course theta1 and theta2 may have different prior distribution but for simplicity I assume that they have same distribution with same parameters. But one can easily set a new parameter to assign them different distributions and parameters

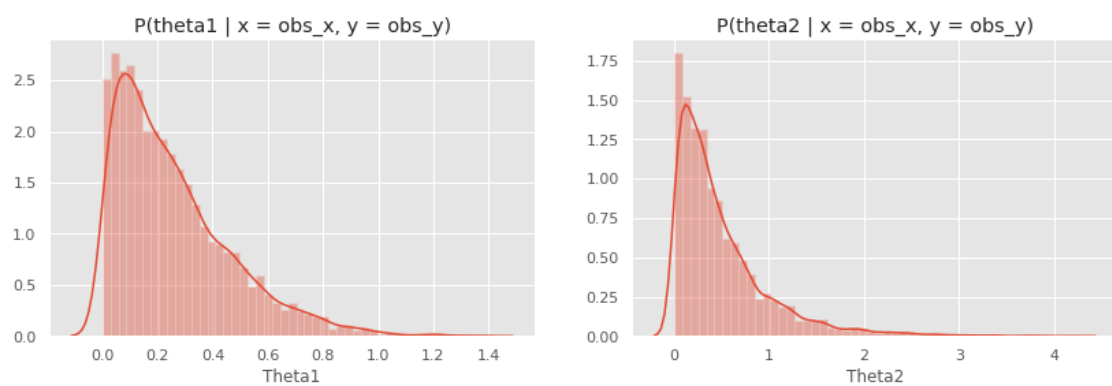
[171]:
```
analyze_hmc(conditioned_model,obs_y,obs_x,dist.Exponential(2),1200, 400, 5)
```

```
HBox(children=(FloatProgress(value=0.0, description='Warmup [1]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [2]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [3]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [4]', max=1600.0, style=ProgressSty

HBox(children=(FloatProgress(value=0.0, description='Warmup [5]', max=1600.0, style=ProgressSty
```

```
Keys: dict_keys(['theta1', 'theta2'])

Summary of MCMC:


                mean       std    median      5.0%     95.0%     n_eff     r_hat
    theta1      0.26      0.21      0.21      0.00      0.55    610.93      1.01
    theta2      0.49      0.50      0.34      0.00      1.12    730.47      1.01

Number of divergences: 0
Shape of chain_1 torch.Size([5, 1200]) and chain_2 torch.Size([5, 1200])

expected theta1 = 0.2577589452266693
expected theta2 = 0.4914016127586365
```







0.0.2 Exercise 2

A bivariate Gibbs sampler for a vector $x = (x_1, x_2)$ draws iteratively from the posterior conditional distributions in the following way: - choose a starting value $p(x_1|x_2^{(0)})$ - for each iteration i: - draw $x_2(i)$ from $p(x_2|x_1^{(i-1)})$ - draw $x_1(i)$ from $p(x_1|x_2^{(i)})$

```
[172]: def gibbs_sampler(iters, warmup ,mu1, mu2, rho):

    '''This is a bivarite implementation of gibbs sampling for normal␣
 ↪distribution'''

    # Starting points and ro
    rho = rho
    x0_1 = torch.tensor([0.])
    x0_2 = torch.tensor([0.])

    x1 = []
    x2 = []

    for i in pyro.plate("samples", iters):
        x_1 = pyro.sample("x_1", dist.Normal(mu1 + rho*(x0_2-mu2), (1-rho**2)))␣
 ↪   # P(A | B_{t-1})
        x_2 = pyro.sample("x_2", dist.Normal(mu2 + rho*(x_1-mu1), (1-rho**2))) ␣
 ↪  # P(B | A_{t})

        x1.append(x_1.item())
        x2.append(x_2.item())

        #update B_0 with B-t
        x0_2 = x_2

    x1 = torch.tensor(x1)
    x2 = torch.tensor(x2)

    return x1[warmup:], x2[warmup:]
```

a. Supposing that samples are drawn from a bivariate normal distribution

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right],$$

implement a Gibbs sampler for x which takes as inputs the number of iterations `iters` and the number of warmup draws `warmup`.

$$If: \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right],$$

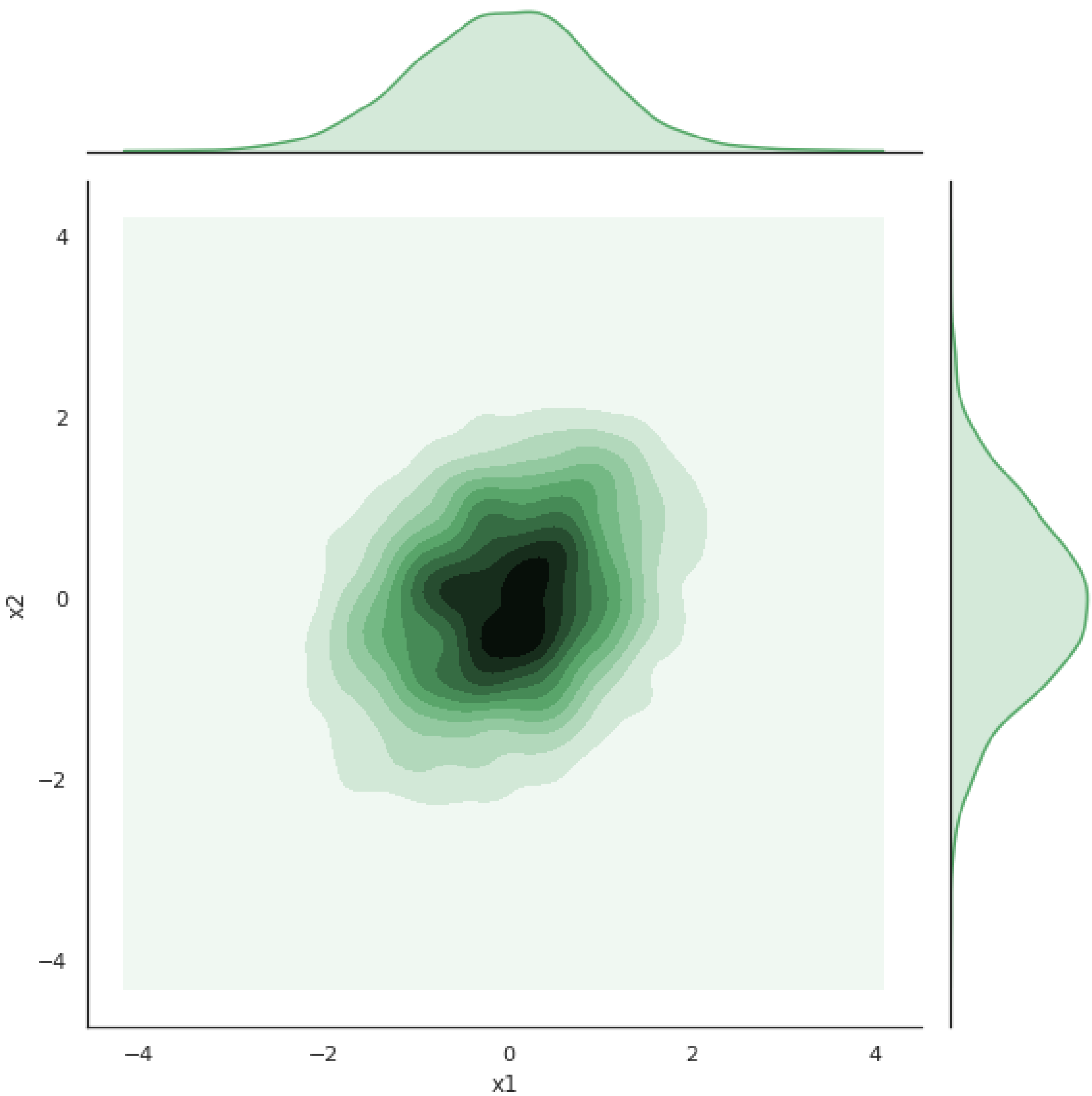
$$Then: x_1|x_2, y \sim \mathcal{N}(\mu_1 + \rho(x_2 - \mu_2), [1 - \rho^2]) x_2|x_1, y \sim \mathcal{N}(\mu_2 + \rho(x_1 - \mu_1), [1 - \rho^2])$$

referance_1 , referance_2

[173]:
```
x1,x2 = gibbs_sampler(3000,500,0,0,0.25)
torch.set_printoptions(precision=5)
print("Mean of x1 {0:.4f}, std of x1 {1:.4f}, mean of x2 {2:.4f}, std of x2 {3:.
 ↪4f}".format(x1.mean(),x1.std(),x2.mean(),x2.std()))
sns.set(style="white", color_codes=True)
x = sns.jointplot(x1,x2,kind="kde",color="g",height=8)
x.set_axis_labels('x1','x2')
```

```
Mean of x1 -0.0293, std of x1 0.9694, mean of x2 -0.0240, std of x2 0.9841
```

[173]: `<seaborn.axisgrid.JointGrid at 0x7fa814502d90>`



b. Use your implementation of Gibbs sampler to infer the parameters $\theta = (\theta_1, \theta_2)$ from **Exercise**

1.

By using the information from previous exercise:

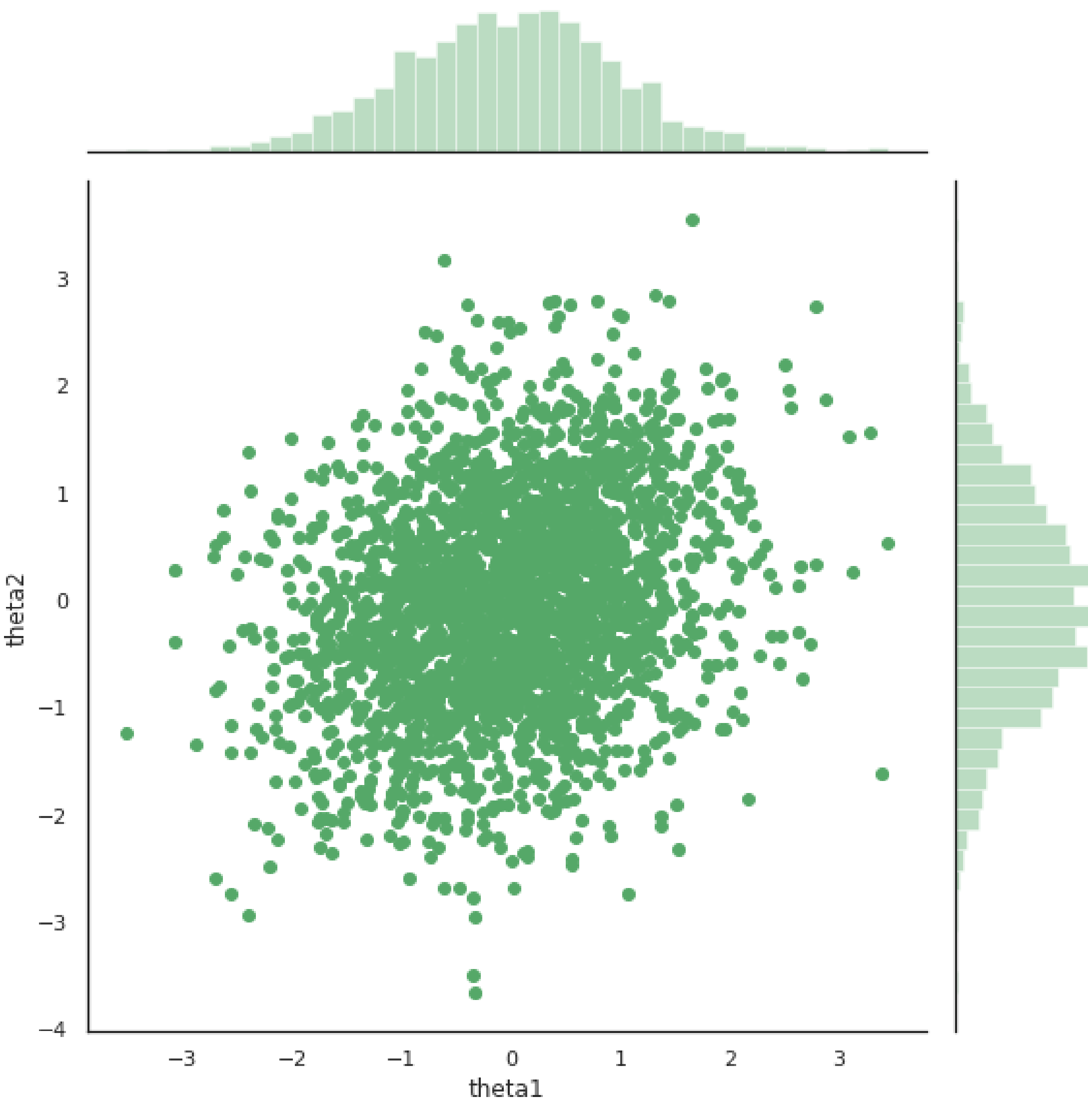
$$\begin{pmatrix}\theta_1\\ \theta_1\end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix}0.13\\ 6.20\end{pmatrix}, \begin{pmatrix}1 & 0.0336\\ 0.0336 & 1\end{pmatrix}\right],$$

P.S : I have used the best normal prior that I get from exercise 1 not the exponential one because of simplicity. 0.13 and 6.20 was found from `mcmc.summary` table and ρ was found by `np.corrcoef` by using all samples from chains.

[174]:
```
theta1, theta2 = gibbs_sampler(3000,500,0.13,6.20, 0.0336)



print("Mean of theta1 is {0:.4f} and std of theta1 is {1:.4f}".format(theta1.
 ↪mean(),theta1.std()))
print("Mean of theta2 is {0:.4f} and std of theta2 is {1:.4f}".format(theta2.
 ↪mean(),theta2.std()))

x = sns.jointplot(x1,x2,color="g",height=8)
x.set_axis_labels('theta1','theta2')
```

```
Mean of theta1 is 0.0958 and std of theta1 is 0.9821
Mean of theta2 is 6.2100 and std of theta2 is 0.9986
```

[174]: `<seaborn.axisgrid.JointGrid at 0x7fa814f3b290>`



theta1
theta2
−3
−2
−1
0
1
2
3
−4