MCMC

June 6, 2023

1 STAT 207: Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) methods have a wide range of applications:

- Bayesian Inference: complex posterior distributions.
- Finance: option pricing, portfolio optimization, and risk assessment.
- Machine learning and AI: best moves.
- Physics and Chemistry: molecular dynamics simulations, protein folding, and quantum Monte Carlo methods.
- Genetics and Genomics, Social Sciences, and Many more.

Two major approches:

- Hastings-Metropolis Algorithm
- Gibbs Sampling

1.1 Hastings-Metropolis Algorithm

Construct a Markov chain with a prescribed equilibrium distribution π on a given state space.

- Proposal stage: From stage i to stage j according to a probability density: $q_{ij} = q(j|i)$.
- Acceptance stage: $U \sim Unif[0,1]$ compared to

$$a_{ij} = \min\left\{\frac{\pi_j q_{ji}}{\pi_i q_{ij}}, 1\right\}.$$

Remarks:

- If q is symmetric with $q_{ji} = q_{ij}$, then the acceptance probability reduces to

$$a_{ij} = \min\left\{\frac{\pi_j}{\pi_i}, 1\right\}.$$

- Check the detailed balance condition. Assume $\pi_i > 0$ for all i, wlog,

$$0 < \frac{\pi_j q_{ji}}{\pi_i q_{ij}} \leq 1$$

for some $i \neq j$. Then

$$\begin{aligned}\pi_i q_{ij} a_{ij} =& \pi_i q_{ij} \frac{\pi_j q_{ji}}{\pi_i q_{ij}} \\ =& \pi_j q_{ji} \\ =& \pi_j q_{ji} a_{ji}.\end{aligned}$$

- Irreducibility holds provided that $\pi_i > 0$ for all i and the proposal matrix$Q = (q_{ij})$ is irreducible.
- Aperiodicity: the acceptance-rejection step allows the chain to remain in place (Problem 4).

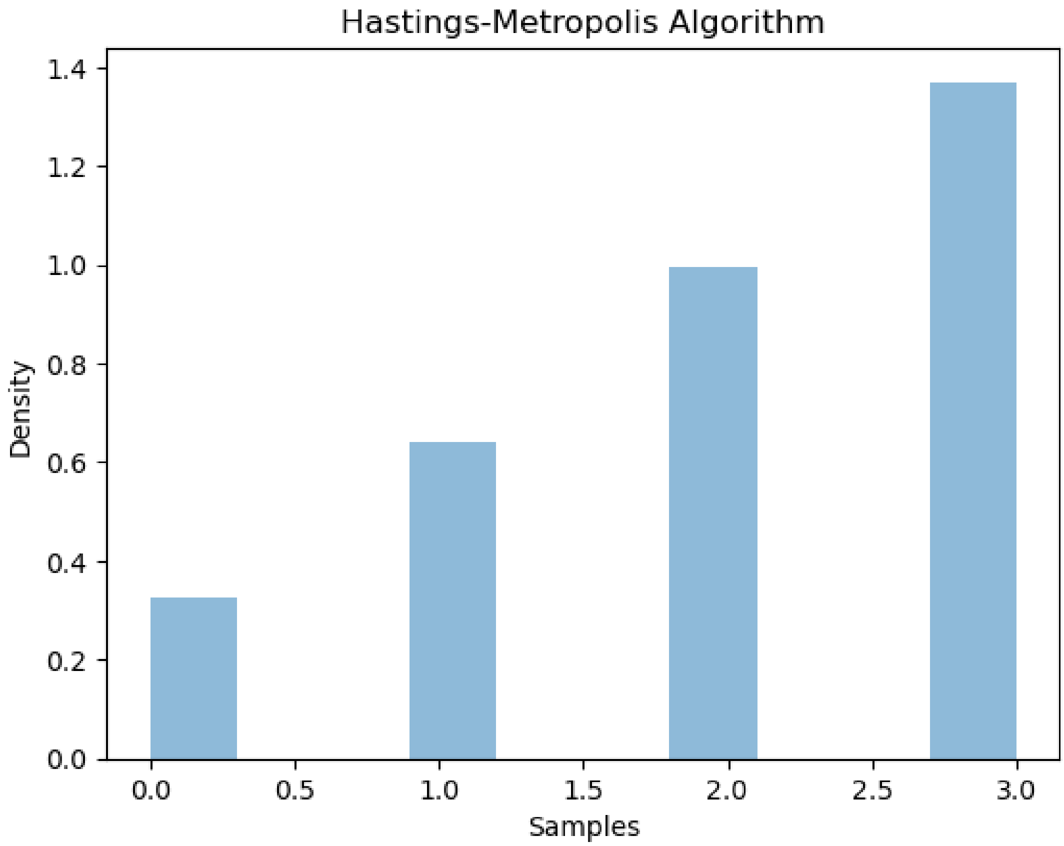
[1]:
```
import numpy as np

def target_distribution(x):
    # Define the target distribution probabilities
    probabilities = [0.1, 0.2, 0.3, 0.4]  # Example probabilities for discrete␣
↪distribution
    return probabilities[x]

def proposal_distribution(x):
    # Define the proposal distribution probabilities
    probabilities = np.full((4, 4), 0.25)  # Example probabilities for discrete␣
↪distribution
    return probabilities[x,]

def hastings_metropolis(target_dist, proposal_dist, num_samples):
    # Initialize the current sample
    current_sample = 0
    samples = []

    for _ in range(num_samples):
        # Generate a proposal sample from the proposal distribution
        proposal_sample = np.random.choice([0,1,2,3],␣
↪p=proposal_dist(current_sample))

        # Calculate the acceptance ratio
        acceptance_ratio = min(1, target_dist(proposal_sample) /␣
↪target_dist(current_sample))

        # Accept or reject the proposal sample based on the acceptance ratio
        if np.random.uniform(0, 1) < acceptance_ratio:
            current_sample = proposal_sample

        # Save the current sample
        samples.append(current_sample)

    return samples
```

```
# Set the parameters
num_samples = 10000

# Run the algorithm
samples = hastings_metropolis(target_distribution, proposal_distribution,␣
 ↪num_samples)
```

[2]:
```
unique, counts = np.unique(samples, return_counts=True)
print(np.asarray((unique, counts)).T)
```

```
[[   0  980]
 [   1 1924]
 [   2 2988]
 [   3 4108]]
```

[3]:
```
# Plot the samples
import matplotlib.pyplot as plt
plt.hist(samples,  density=True, alpha=0.5)
plt.xlabel('Samples')
plt.ylabel('Density')
plt.title('Hastings-Metropolis Algorithm')
plt.show()
```



1.2 Gibbs Sampling

- The Gibbs sampler is a special case of the Hastings-Metropolis algorithmfor Cartesian product state spaces.
- Our random variable of interest has d components, $x = (x_1, x_2, ..., x_d)$,
- Suppose we can simulate the distribution of each component conditional on the others, i.e., $\pi(x_k|x_1, ..., x_{k-1}, x_{k+1}, ..., x_d)$ for all k.
- We want to sample from the joint distribution, $\pi(x)$.
- Gibbs sampling constructs a Markov Chain, $x^{(1)} \to x^{(2)} \to ...$, with step $x^{(j)} \to x^{(j+1)}$ given by:

  Simulate

  - $x_1^{(j+1)}$ from $\pi(x_1^{(j+1)}|x_2^{(j)}, ..., x_d^{(j)})$
  - $x_2^{(j+1)}$ from $\pi(x_2^{(j+1)}|x_1^{(j+1)}, x_3^{(j)}, ..., x_d^{(j)})$
  - $x_3^{(j+1)}$ from $\pi(x_3^{(j+1)}|x_1^{(j+1)}, x_2^{(j+1)}, x_4^{(j)}, ..., x_d^{(j)})$
  - ...
- By iteratively updating each component, the Gibbs sampler generates a sequence of samples

that approximates the joint distribution of interest.

- To see this, for a component x_c transit from i to j, the transition probability is

$$q_{ij} = \frac{1}{d}\frac{\pi_j}{\sum_{j\in\{k:k_{-c}=i_{-c}\}}\pi_k},$$

which satisfies $\pi_i q_{ij} = \pi_j q_{ji}$.

- Denote $P^{(c)}$ the transition matrix for changing component c while leaving other components unaltered.
  - Random sampling: $R = \frac{1}{d}\sum_c P^{(c)}$;
  - Sequencial sampling: $S = \prod_c P^{(c)}$.
  - $\pi R = \pi$ and $\pi S = \pi$.
  - R satisfies detailed balance while S ordinarily does not (Problem 6).

1.2.1 Conjugate distributions

A likelihood $p(x|\theta)$ and a prior density $p(\theta)$ are said to be conjugate provided the posterior density $p(\theta|x)$ has the same functional form as the prior density.

TABLE 26.1. Conjugate Pairs

Likelihood	Density	Prior	Density
Binomial	$\binom{n}{x}p^x(1-p)^{n-x}$	Beta	$\frac{1}{B(\alpha,\beta)}p^{\alpha-1}(1-p)^{\beta-1}$
Poisson	$\frac{\lambda^x}{x!}e^{-\lambda}$	Gamma	$\frac{\beta^\alpha\lambda^{\alpha-1}}{\Gamma(\alpha)}e^{-\beta\lambda}$
Geometric	$(1-p)^x p$	Beta	$\frac{1}{B(\alpha,\beta)}p^{\alpha-1}(1-p)^{\beta-1}$
Multinomial	$\binom{n}{x_1\dots x_k}\prod_{i=1}^k p_i^{x_i}$	Dirichlet	$\frac{\Gamma\left(\sum_{i=1}^k\alpha_i\right)}{\prod_{i=1}^k\Gamma(\alpha_i)}\prod_{i=1}^k p_i^{\alpha_i-1}$
Normal	$\sqrt{\frac{\tau}{2\pi}}e^{-\tau(x-\mu)^2/2}$	Normal	$\sqrt{\frac{\omega}{2\pi}}e^{-\omega(\mu-\theta)^2/2}$
Normal	$\sqrt{\frac{\tau}{2\pi}}e^{-\tau(x-\mu)^2/2}$	Gamma	$\frac{\beta^\alpha\tau^{\alpha-1}}{\Gamma(\alpha)}e^{-\beta\tau}$
Exponential	$\lambda e^{-\lambda x}$	Gamma	$\frac{\beta^\alpha\lambda^{\alpha-1}}{\Gamma(\alpha)}e^{-\beta\lambda}$

1.2.2 Example: Ising Model

- Consider m elementary particles equally spaced around the boundary of the unit circle.
- Each particle c can be in one of two magnetic states—spin up with $i_c = 1$ or spin down with $i_c = -1$.
- The Gibbs distribution

$$\pi_i = \exp(\beta\sum_c i_c i_{c+1})$$

takes into account nearest-neighbor interactions in the sense that states like $(1, 1, 1, ..., 1, 1, 1)$ are favored and states like $(1, -1, 1, ..., 1, -1, 1)$ are less likely for $\beta > 0$.

- If we elect to resample component c, then the choices $j_c = -i_c$ and $j_c = i_c$ are made with respective probabilities

$$\frac{e^{\beta(-i_{c-1}i_c - i_c i_{c+1})}}{e^{\beta(i_{c-1}i_c + i_c i_{c+1})} + e^{\beta(-i_{c-1}i_c - i_c i_{c+1})}} = \frac{1}{e^{2\beta(i_{c-1}i_c + i_c i_{c+1})} + 1}$$

$$\frac{e^{\beta(i_{c-1}i_c + i_c i_{c+1})}}{e^{\beta(i_{c-1}i_c + i_c i_{c+1})} + e^{\beta(-i_{c-1}i_c - i_c i_{c+1})}} = \frac{1}{1 + e^{-2\beta(i_{c-1}i_c + i_c i_{c+1})}}$$

1.2.3 Example: Capture-Recapture Estimation

- To estimate the number of fish f in a lake.
- Fish t times, mark each fish caught. Data (c_i, r_i).
- $u_i = \sum_{j=1}^{r}(c_i - r_i)$ unique fish.
- Independent binomial sampling with success probability p_i for trial i,

$$\begin{aligned}
&\prod_{i=1}^{t} \binom{u_{i-1}}{r_i} p_i^{r_i}(1-p_i)^{u_{i-1}-r_i} \binom{f-u_{i-1}}{c_i - r_i} p_i^{c_i - r_i}(1-p_i)^{f-u_{i-1}-c_i+r_i} \\
=&\binom{u_{i-1}}{r_i}\binom{f-u_{i-1}}{c_i - r_i} p_i^{c_i}(1-p_i)^{f-c_i} \\
=&\frac{f!}{(f-u_t)!}\prod_{i=1}^{t}\binom{u_{i-1}}{r_i} p_i^{c_i}(1-p_i)^{f-c_i}
\end{aligned}$$

  where $u_0 = r_1 = 0$.

- Poisson prior for f and independent beta priors on p_i's. The joint density is

$$\begin{aligned}
&\frac{f!}{(f-u_t)!}\prod_{i=1}^{t}\binom{u_{i-1}}{r_i} p_i^{c_i}\left(1-p_i\right)^{f-c_i} \frac{\lambda^f e^{-\lambda}}{f! B(\alpha,\beta)^t}\prod_{i=1}^{t} p_i^{\alpha-1}\left(1-p_i\right)^{\beta-1} \\
=&\frac{\lambda^f e^{-\lambda}}{(f-u_t)! B(\alpha,\beta)^t}\prod_{i=1}^{t}\binom{u_{i-1}}{r_i} p_i^{\alpha+c_i-1}\left(1-p_i\right)^{\beta+f-c_i-1}.
\end{aligned}$$

- The Gibbs update for p_i is $Beta(\alpha + c_i, \beta + f - c_i)$.
- The Gibbs update for f is that $f - u_t$ following Poisson with mean $\lambda \prod_{i=1}^{t}(1-p_i)$.

1.3 Example: Slice Sampling

- X with density $f(x)$
- Gibbs sampling on the pair (X, Y) with auxiliary Y.
- Sample uniformly from the region under the graph of $f(x)$.
- Given X, Y is uniform from $[0, f(X)]$; given $Y = y$, X is uniform from $\{x : f(x) \geq y\}$.

$$\Pr(X \in A) = \int_A \frac{1}{f(x)} \int_0^{f(x)} dy f(x) dx = \int_A \int_0^{f(x)} dy dx.$$

- For example, $f(x) = e^{-x^2/2}$,

- When $f(x)$ is concave or log-concave, a top set is convex and therefore connected. Otherwise, it can be disconnected.

1.4 Example: Independence Sampler

- If $q_{ij} = q_j$, then candidate points are drawn independently of the current point.
- Want q_i close to π_i, introducing the importance ratios $w_i = \pi_i/q_i$ and the acceptance probability:
$$a_{ij} = \min\left\{\frac{w_j}{w_i}, 1\right\}.$$

1.5 Example: Random Walk

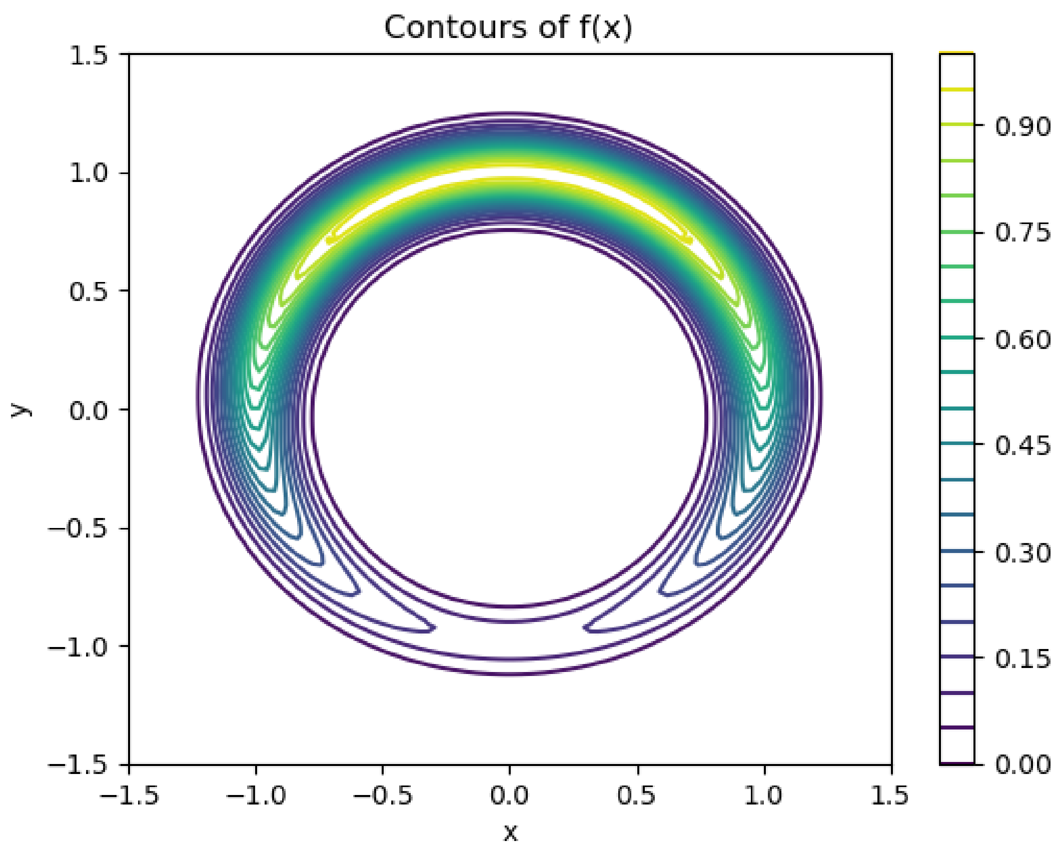
- If $q_{ij} = q_{j-i}$ for some density q_k, and $q_k = q_{-k}$, then the acceptance probability is
$$a_{ij} = \min\left\{\frac{\pi_j}{\pi_i}, 1\right\}.$$

- Consider the density on R^2:
$$f(x) = e^{-\frac{(\|x\|_2-1)^2}{2\sigma^2}} e^{-\frac{(x_2-1)^2}{2\delta^2}} = f_1(x)f_2(x)$$

- The proposed symmetric density:
$$g(y) = \frac{1}{2\pi\gamma^2} e^{-\frac{\|y\|_2^2}{2\gamma^2}}$$

[4]:
```
import numpy as np
import matplotlib.pyplot as plt

def f(x, sigma, delta):
    norm_term = np.exp(-((np.linalg.norm(x, axis=0)-1)**2)/(2*sigma**2))
    exp_term = np.exp(-((x[1]-1)**2)/(2*delta**2))
    return norm_term * exp_term

# Define the range of x and y values
x = np.linspace(-1.5, 1.5, 100)
y = np.linspace(-1.5, 1.5, 100)

# Create a grid of (x, y) points
X, Y = np.meshgrid(x, y)

# Evaluate the function at each point of the grid
```

```
Z = f(np.array([X, Y]), sigma=0.1, delta=1)



# Plot the contours
plt.contour(X, Y, Z, levels=20)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Contours of f(x)')
plt.colorbar()
plt.show()
```



[5]:
```
def g(y, gamma):
    norm_term = np.exp(-(np.linalg.norm(y)**2)/(2*gamma**2))
    return norm_term / (2*np.pi*gamma**2)

def hastings_metropolis_sampling(target_density, proposed_density,
 ↪initial_sample, num_samples, gamma):
    samples = [initial_sample]
    accepted_samples = 0
```

```
    for _ in range(num_samples):
        current_sample = samples[-1]
        proposed_sample = np.random.normal(current_sample, gamma)

        acceptance_prob = min(1, (target_density(proposed_sample) *␣
 ↪proposed_density(current_sample, gamma)) /
                                        (target_density(current_sample) *␣
 ↪proposed_density(proposed_sample, gamma)))

        u = np.random.uniform()
        if u < acceptance_prob:
            samples.append(proposed_sample)
            accepted_samples += 1
        else:
            samples.append(current_sample)

    acceptance_rate = accepted_samples / num_samples
    return np.array(samples), acceptance_rate

# Define the target density function f(x)
sigma = 0.1
delta = 1
target_density = lambda x: f(x, sigma, delta)
```

[6]:
```
# Define the proposed density function g(y)
gamma_values = np.logspace(np.log10(0.01), np.log10(8), num=10)

print(gamma_values)
proposed_density = lambda y, gamma: g(y, gamma)

# Set the number of samples to generate
num_samples = 1000
```

```
[0.01       0.02101675 0.04417038 0.09283178 0.19510222 0.41004146
 0.86177388 1.81116859 3.80648772 8.        ]
```

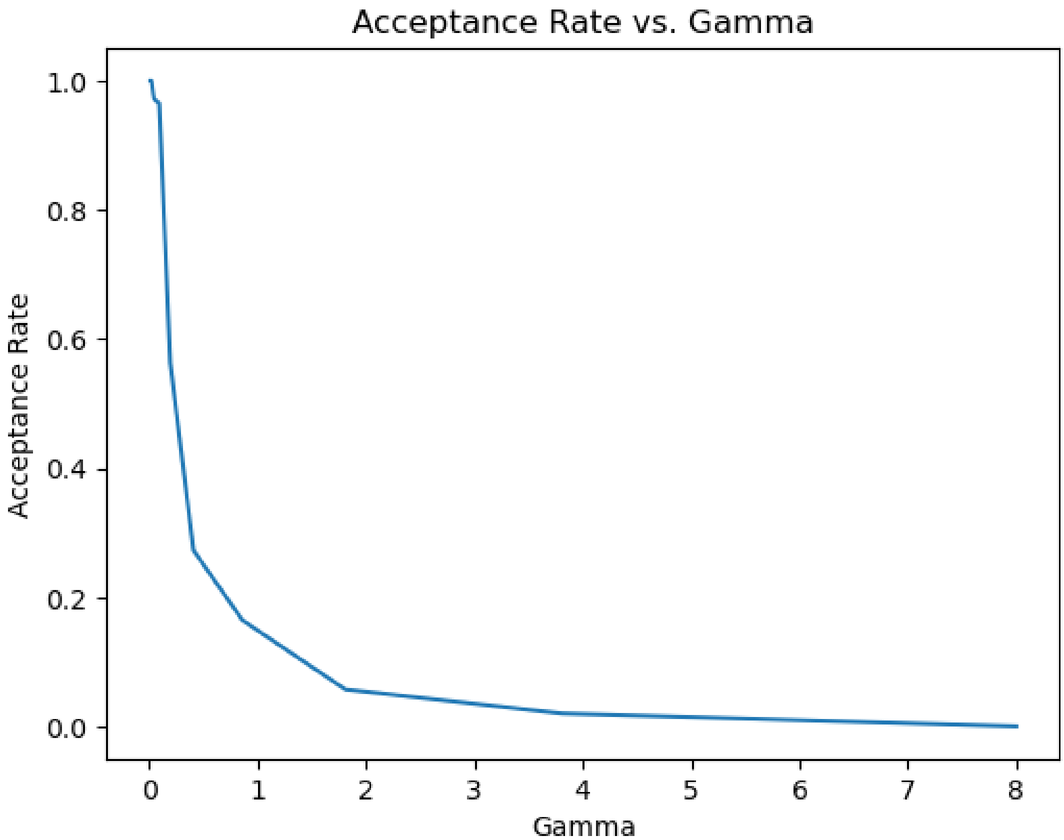
[7]:
```
# Set the initial sample
initial_sample = np.array([1,1])  # Adjust as needed

# Generate samples using Hastings-Metropolis sampling for different gamma values
samples = []
acceptance_rates = []
for gamma in gamma_values:
    result = hastings_metropolis_sampling(target_density, proposed_density,␣
 ↪initial_sample, num_samples, gamma)
    samples.append(result[0])
```

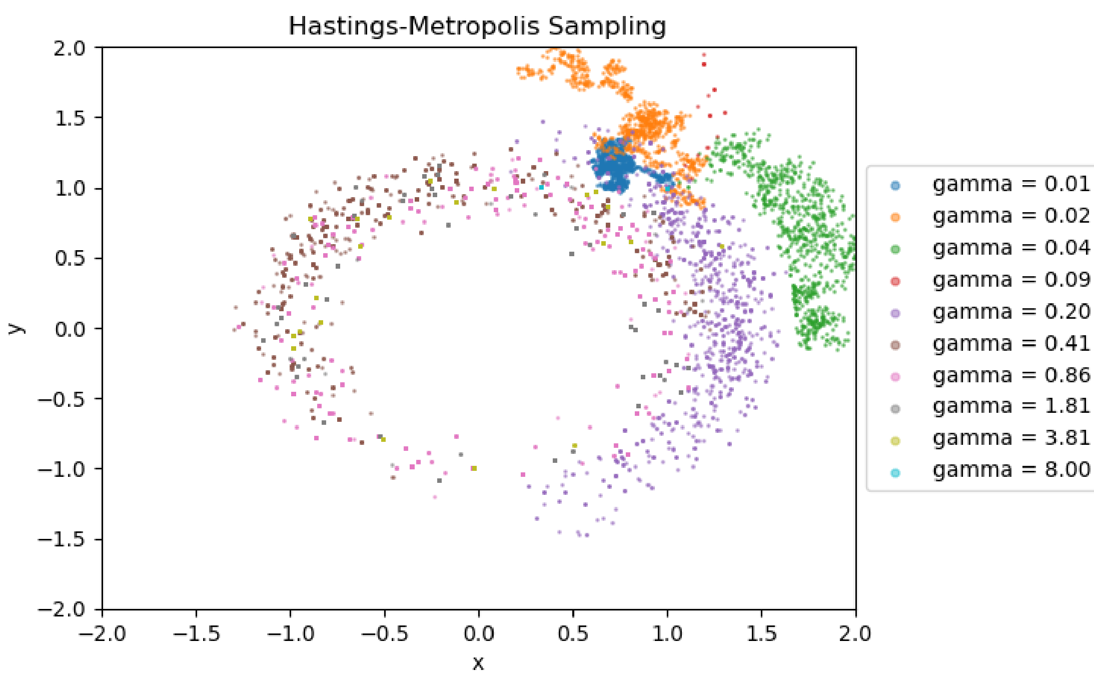
```
    acceptance_rates.append(result[1])
```

```
/var/folders/22/nmmy2bvn43dby4g5jhlr3jzw0000gn/T/ipykernel_10619/123529884.py:13
: RuntimeWarning: invalid value encountered in double_scalars
  acceptance_prob = min(1, (target_density(proposed_sample) *
proposed_density(current_sample, gamma)) /
/var/folders/22/nmmy2bvn43dby4g5jhlr3jzw0000gn/T/ipykernel_10619/123529884.py:13
: RuntimeWarning: divide by zero encountered in double_scalars
  acceptance_prob = min(1, (target_density(proposed_sample) *
proposed_density(current_sample, gamma)) /
```

[8]:

```
# Plotting the acceptance rates
plt.plot(gamma_values, acceptance_rates)
plt.xlabel('Gamma')
plt.ylabel('Acceptance Rate')
plt.title('Acceptance Rate vs. Gamma')
plt.show()

# Plotting the samples
for i, gamma in enumerate(gamma_values):
    plt.scatter(samples[i][:, 0], samples[i][:, 1], s=1, alpha=0.5,␣
 ↪label=f'gamma = {gamma:.2f}')
plt.xlim(-2, 2)
plt.ylim(-2, 2)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Hastings-Metropolis Sampling')
plt.legend(markerscale=3.5, loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```

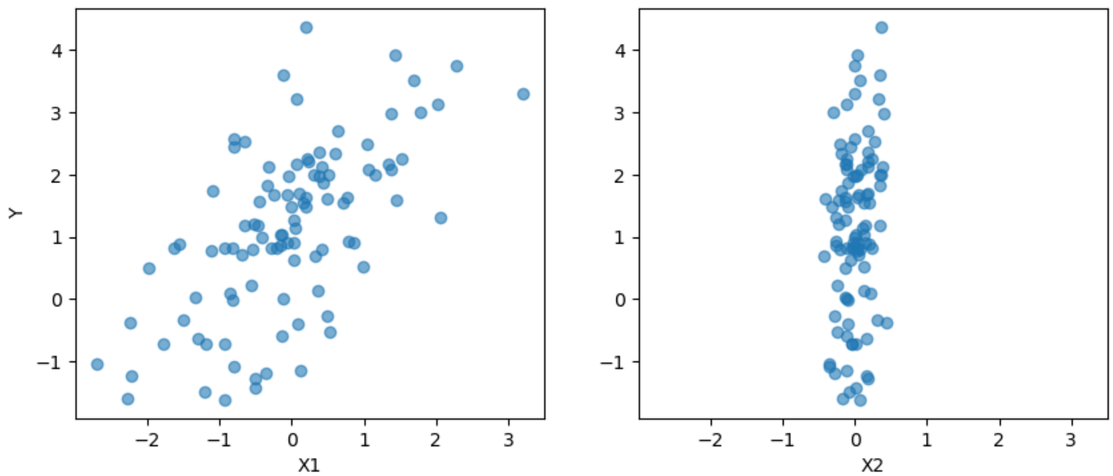

Acceptance Rate vs. Gamma
Acceptance Rate
Gamma
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8


Hastings-Metropolis Sampling
y
x
gamma = 0.01
gamma = 0.02
gamma = 0.04
gamma = 0.09
gamma = 0.20
gamma = 0.41
gamma = 0.86
gamma = 1.81
gamma = 3.81
gamma = 8.00

[9]:
```
## https://www.pymc.io/projects/docs/en/stable/learn/core_notebooks/
↪pymc_overview.html#pymc-overview

import numpy as np
import pymc as pm

print(f"Running on PyMC v{pm.__version__}")
```

Running on PyMC v5.4.1

[10]:
```
import arviz as az
import pandas as pd
import matplotlib.pyplot as plt

RANDOM_SEED = 8927
rng = np.random.default_rng(RANDOM_SEED)
# True parameter values
alpha, sigma = 1, 1
beta = [1, 2.5]

# Size of dataset
size = 100

# Predictor variable
X1 = np.random.randn(size)
X2 = np.random.randn(size) * 0.2

# Simulate outcome variable
Y = alpha + beta[0] * X1 + beta[1] * X2 + rng.normal(size=size) * sigma

fig, axes = plt.subplots(1, 2, sharex=True, figsize=(10, 4))
axes[0].scatter(X1, Y, alpha=0.6)
axes[1].scatter(X2, Y, alpha=0.6)
axes[0].set_ylabel("Y")
axes[0].set_xlabel("X1")
axes[1].set_xlabel("X2");
```



[11]:
```
basic_model = pm.Model()

with basic_model:
    # Priors for unknown model parameters
    alpha = pm.Normal("alpha", mu=0, sigma=10)
    beta = pm.Normal("beta", mu=0, sigma=10, shape=2)
    sigma = pm.HalfNormal("sigma", sigma=1)

    # Expected value of outcome
    mu = alpha + beta[0] * X1 + beta[1] * X2

    # Likelihood (sampling distribution) of observations
    Y_obs = pm.Normal("Y_obs", mu=mu, sigma=sigma, observed=Y)
```

[12]:
```
with basic_model:
    # draw 1000 posterior samples
    idata = pm.sample(1000)
```

```
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alpha, beta, sigma]
```

```
<IPython.core.display.HTML object>
```

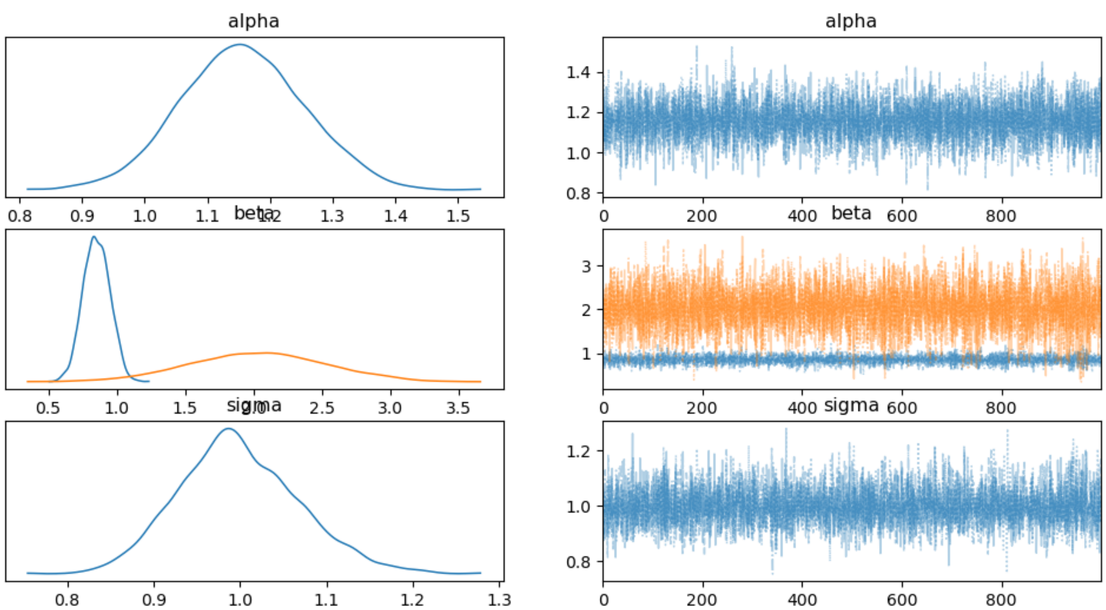
```
<IPython.core.display.HTML object>
```

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws
total) took 1 seconds.
```

[13]:
```
idata.posterior["alpha"].sel(draw=slice(0, 4))
```

[13]:
```
<xarray.DataArray 'alpha' (chain: 4, draw: 5)>
array([[1.09940941, 1.31344334, 1.20623435, 1.08871712, 1.0559109 ],
       [1.23086922, 1.36363135, 1.37651772, 0.99086544, 1.27511779],
       [1.36839289, 1.28650864, 1.08145372, 1.11220016, 1.04912181],
       [1.02794591, 1.26310488, 1.05876462, 1.05876462, 1.15713395]])
Coordinates:
  * chain    (chain) int64 0 1 2 3
  * draw     (draw) int64 0 1 2 3 4
```

[14]:
```python
az.plot_trace(idata, combined=True);
```



[15]:
```python
az.summary(idata, round_to=2)
```

[15]:
```
         mean    sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  ess_tail  \
alpha    1.16  0.10    0.98     1.34       0.00      0.0   6626.42   3198.90
beta[0]  0.85  0.10    0.68     1.04       0.00      0.0   5746.07   3456.17
beta[1]  2.03  0.50    1.09     2.97       0.01      0.0   5960.64   3143.41
sigma    1.00  0.07    0.87     1.13       0.00      0.0   6039.31   3261.14

         r_hat
alpha      1.0
beta[0]    1.0
beta[1]    1.0
sigma      1.0
```

[ ]: