CGS698C, Assignment 3

Jiyanshu Dhaka 220481

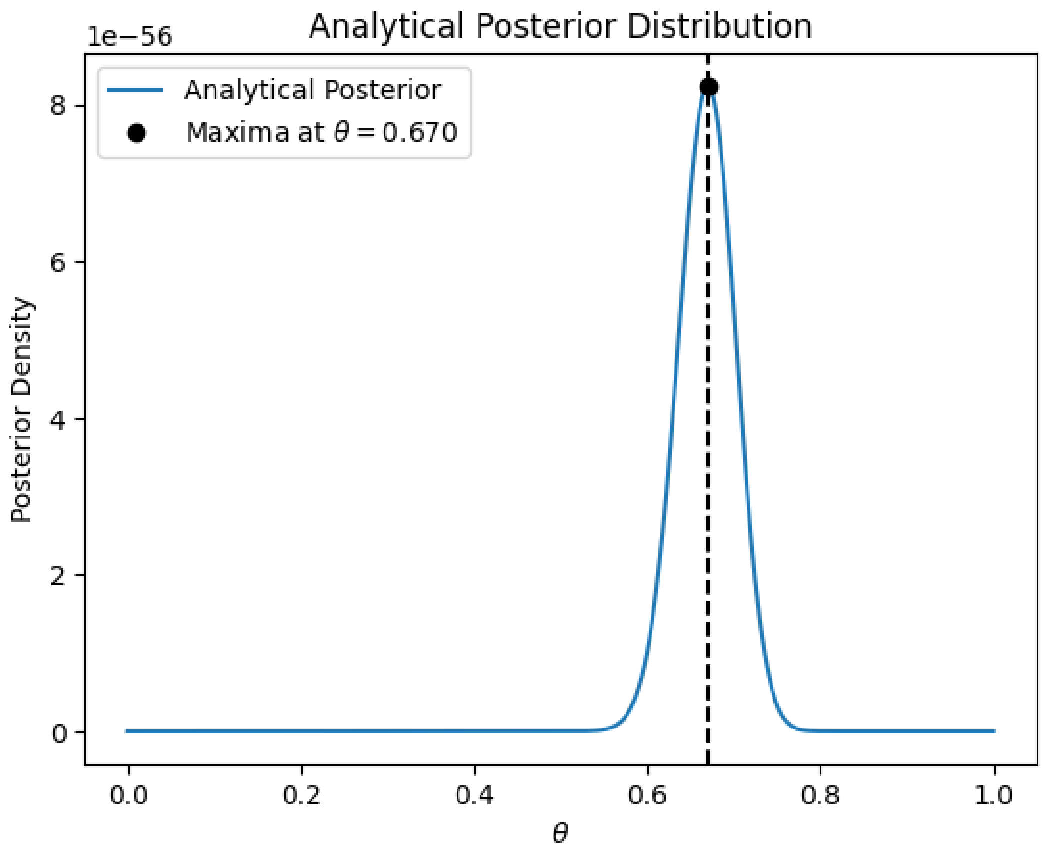
Part 1: Estimating the Posterior Distribution Using Different Computational Methods

[107]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.special import factorial
from scipy.stats import norm
import seaborn as sns

exp = [10, 15, 15, 14, 14, 14, 13, 11, 12, 16]
s = 10
N = 20

def pe(theta, alpha, beta):
    return (theta ** (alpha - 1)) * ((1 - theta) ** (beta - 1))

def le(theta, N, e):
    return ((theta ** e) * ((1 - theta) ** (N - e)))*(factorial(N)/
 ↪(factorial(e)*factorial(N - e)))
```

1.1 Graph Analytical Posterior

[108]:
```
theta_s = np.linspace(0,1, 1000)
y_pa = [pe(theta, 135, 67) for theta in theta_s]
max_i = np.argmax(y_pa)
theta_max = theta_s[max_i]
pdf_max = y_pa[max_i]
plt.plot(theta_s, y_pa, label="Analytical Posterior")
plt.plot(theta_max, pdf_max, 'ok', label=f'Maxima at $\\theta={theta_max:.3f}$')
plt.axvline(x=theta_max, color='k', linestyle='--')
plt.xlabel("$\\theta$")
```

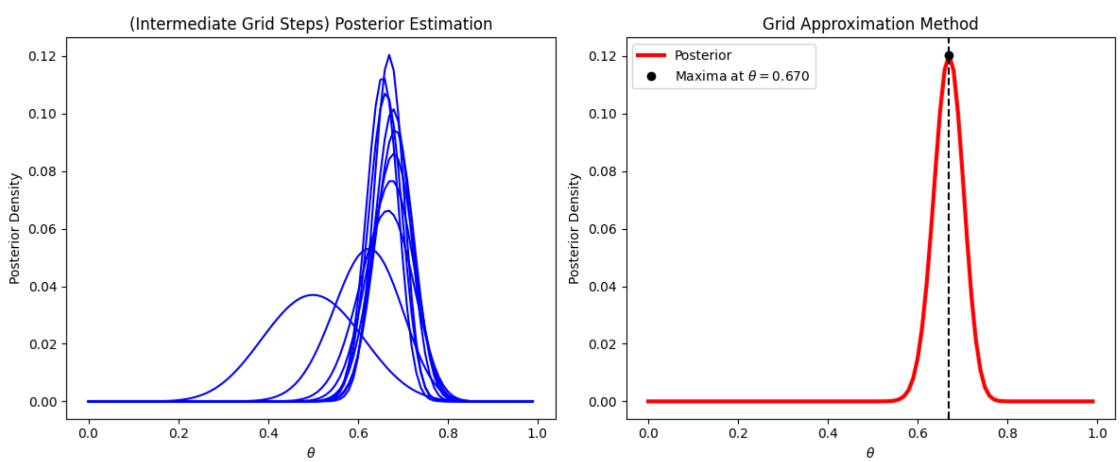
```python
plt.ylabel("Posterior Density")
plt.title("Analytical Posterior Distribution")
plt.legend()
plt.show()
```



1.2 Estimate Posterior with Grid

[109]:
```python
lb = 0.
ub = 1.
samples_ = 100
grid_s = [lb + ((ub - lb)*i)/samples_ for i in range(samples_)]
y_l = np.ones(samples_)
fig, ax = plt.subplots(1, 2, figsize=(12, 5))
for e in exp:
    for i in range(samples_):
        y_l[i] *= le(grid_s[i], N, e)
    y_post_i = y_l/(np.sum(y_l))
    ax[0].plot(grid_s, y_post_i, color = 'b', label = None)
ax[0].set_xlabel("$\\theta$")
```

```
ax[0].set_ylabel("Posterior Density")
ax[0].set_title("(Intermediate Grid Steps) Posterior Estimation")

Approx_ML = np.sum(y_l*1)
y_post_grid = y_l/Approx_ML
max_i = np.argmax(y_post_grid)
theta_max = grid_s[max_i]
pdf_max = y_post_grid[max_i]
ax[1].plot(grid_s, y_post_grid, color = 'r', linestyle = 'solid', linewidth =␣
↪3, label="Posterior")
ax[1].plot(theta_max, pdf_max, 'ok', label=f'Maxima at $\\theta={theta_max:.
↪3f}$')
ax[1].axvline(x=theta_max, color='k', linestyle='--')
ax[1].set_xlabel("$\\theta$")
ax[1].set_ylabel("Posterior Density")
ax[1].set_title("Grid Approximation Method")
ax[1].legend()
plt.tight_layout()
plt.show()
```



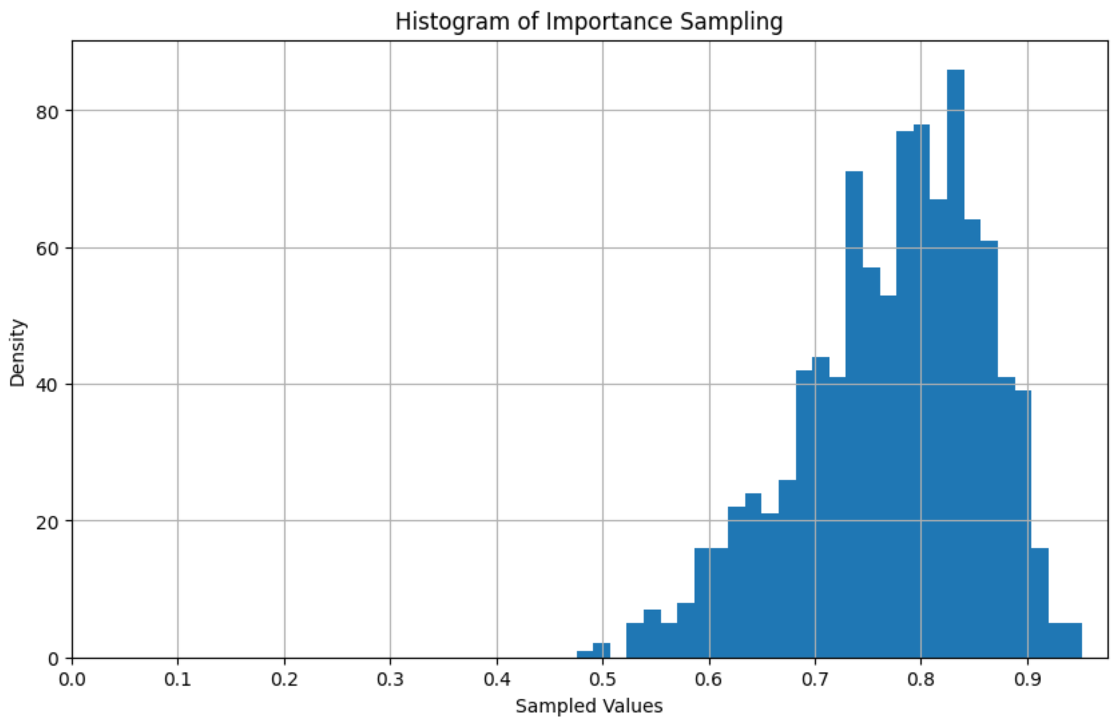
1.3 Estimate Marginal Likelihood

[110]:
```
theta_s = np.random.beta(1, 1, 100000)
y_l = np.ones(100000)
for e in exp:
    for i in range(100000):
        y_l[i] *= le(theta_s[i], N, e)
ML = np.mean(y_l)
print("Average Likelihood = ", ML)
```

```
Average Likelihood =  1.423430051730923e-10
```

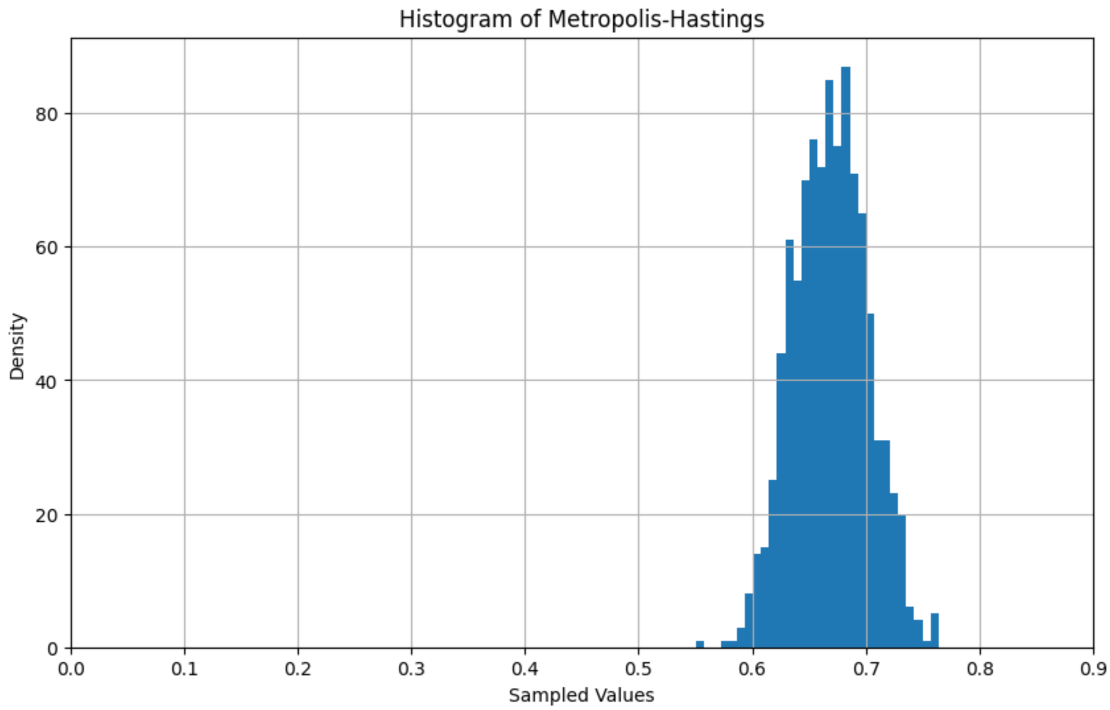
1.4 Importance Sampling

[111]:
```
n = 4000
theta_s= abs(np.random.normal(0.,1., n))
DF = pd.DataFrame(columns = ['theta', 'weight'], index= range(n))
for e in exp:
    for i in range(n):
        DF.loc[i, 'theta'] = theta_s[i]
        DF.loc[i, 'weight'] = (le(theta_s[i], N, e)*(theta_s[i] <= 1.))/norm.
 ↪pdf(theta_s[i], 0., 1.)
y_posterior_imp = np.random.choice(DF['theta'],size = n//4, p = DF['weight']/np.
 ↪sum(DF['weight']))
plt.figure(figsize=(10, 6))
plt.hist(y_posterior_imp, bins=30, label = 'density')
plt.title("Histogram of Importance Sampling")
plt.xlabel("Sampled Values")
plt.ylabel("Density")
plt.grid(True)
plt.xticks(np.arange(0., 1., 0.1))
plt.show()
```



1.5 Markov Chain Monte Carlo

[112]:
```
num_samples = 1000
theta_s = np.zeros(num_samples)
theta_s[0] = np.random.beta(1, 1)
prev_post = 0.
for e in exp:
    prev_post = prev_post + np.log(le(theta_s[0], N, e))
i = 1
step = 0.04
reject = 0
while i < num_samples:
    proposed_theta = np.random.normal(theta_s[i-1], step)
    if(0. <= proposed_theta <= 1.):
        proposed_posterior = 0.
        for e in exp:
            proposed_posterior = proposed_posterior + np.log(le(proposed_theta, N , e))
        h_r = proposed_posterior - prev_post
        hastings_ratio = np.exp(h_r)
        p_str = min(1, hastings_ratio)
        if(np.random.uniform(0, 1, 1) < p_str):
            theta_s[i] = proposed_theta
            prev_post = proposed_posterior
            i += 1
        else:
            reject += 1
    else:
        reject += 1
plt.figure(figsize=(10, 6))
plt.hist(theta_s, bins=30, label = 'density')
plt.title("Histogram of Metropolis-Hastings")
plt.xlabel("Sampled Values")
plt.ylabel("Density")
plt.grid(True)
plt.xticks(np.arange(0., 1., 0.1))
plt.show()
print("Rejections rate= ", (reject/num_samples)*100)
y_posterior_mc = theta_s
```

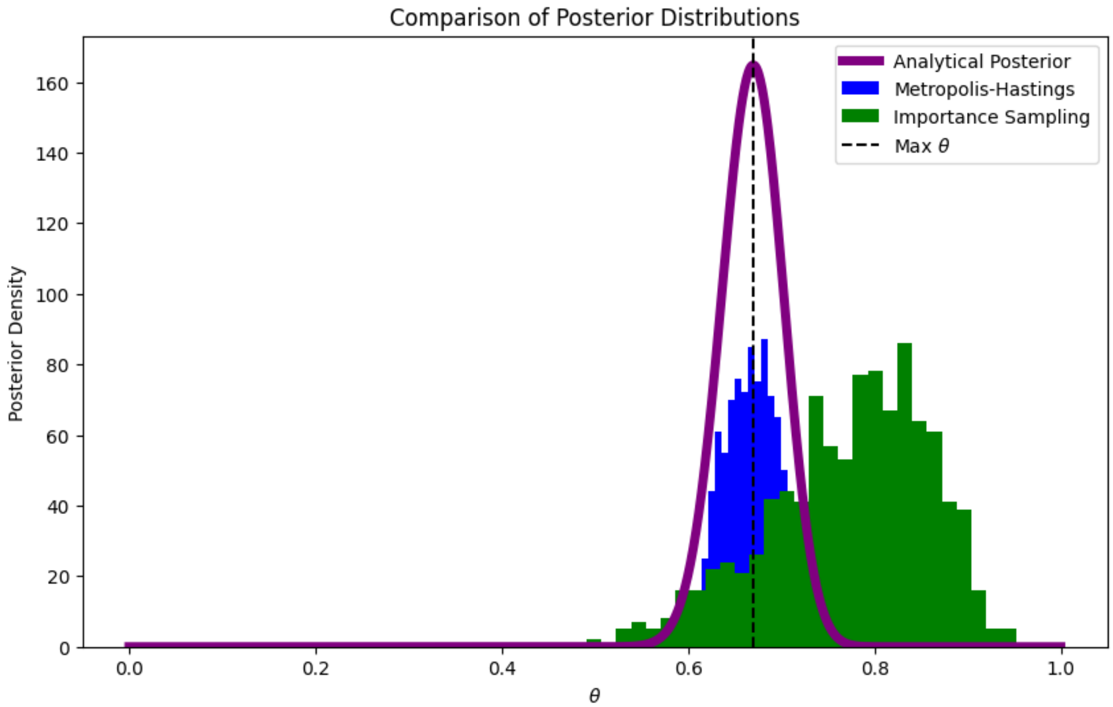



```
Rejections rate=  49.0
```

1.6 Comparison of Posterior Distributions

[113]:
```python
theta_s = np.linspace(0, 1, 1000)
y_pa_scaled = [pe(theta, 135, 67)*(2*(1e+57)) for theta in theta_s]
plt.figure(figsize=(10, 6))
plt.plot(theta_s, y_pa_scaled, label="Analytical Posterior", linewidth=5,␣
 ↪color='purple')
plt.hist(y_posterior_mc, color='blue', bins=30, label="Metropolis-Hastings")
plt.hist(y_posterior_imp, color='green', bins=30, label="Importance Sampling")
plt.axvline(x=theta_max, color='k', linestyle='--', label="Max $\\theta$")
plt.xlabel("$\\theta$")
plt.ylabel("Posterior Density")
plt.title("Comparison of Posterior Distributions")
plt.legend()
plt.show()
print("Here, I have scaled Analytical Posterior for better analysis of the␣
 ↪graphs.")
print("From the graph:")
print("(1). Metropolis-Hastings Method fits better to the Analytical Method.")
print("(2). Maxima for Metropolis-Hastings Method and Analytical Method are␣
 ↪nearly the same.")
```

```
print("(3). Importance Sampling fits less to both of the other methods.")
```



Here, I have scaled Analytical Posterior for better analysis of the graphs.
From the graph:
(1). Metropolis-Hastings Method fits better to the Analytical Method.
(2). Maxima for Metropolis-Hastings Method and Analytical Method are nearly the same.
(3). Importance Sampling fits less to both of the other methods.

Part 2: Writing your own sampler for Bayesian inference

2.5.1 Markov Chain Monte Carlo

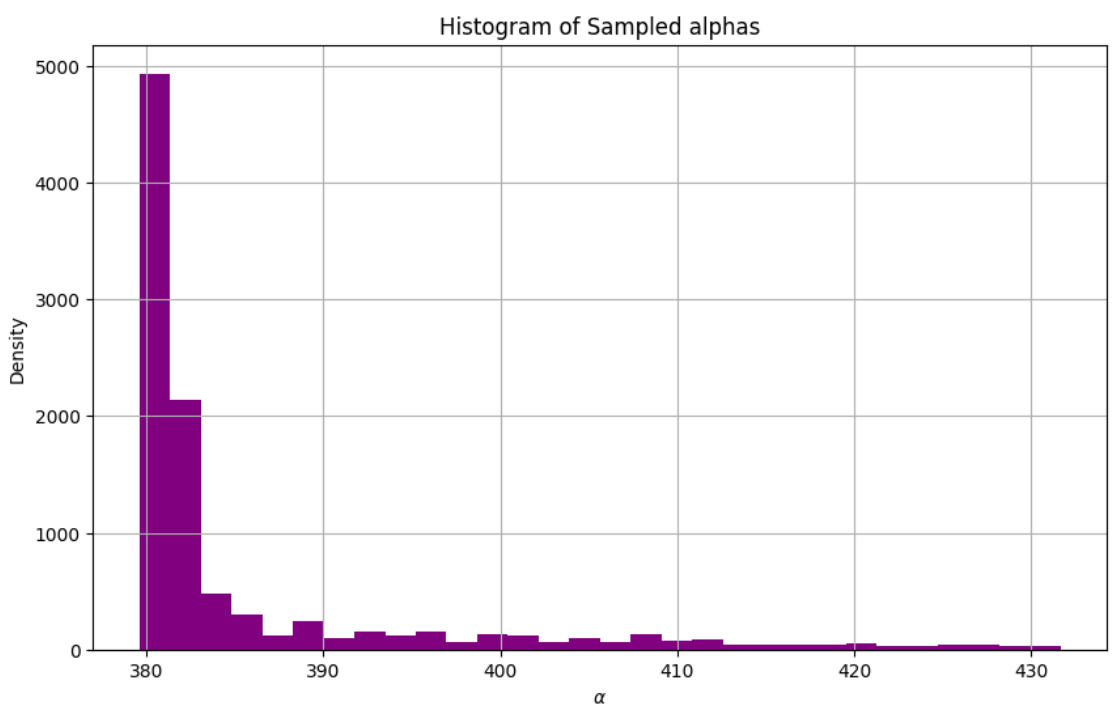
[114]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import norm

url = "https://raw.githubusercontent.com/yadavhimanshu059/CGS698C/main/notes/
↪Data/word-recognition-times.csv"
data = pd.read_csv(url)

n = 4000
```

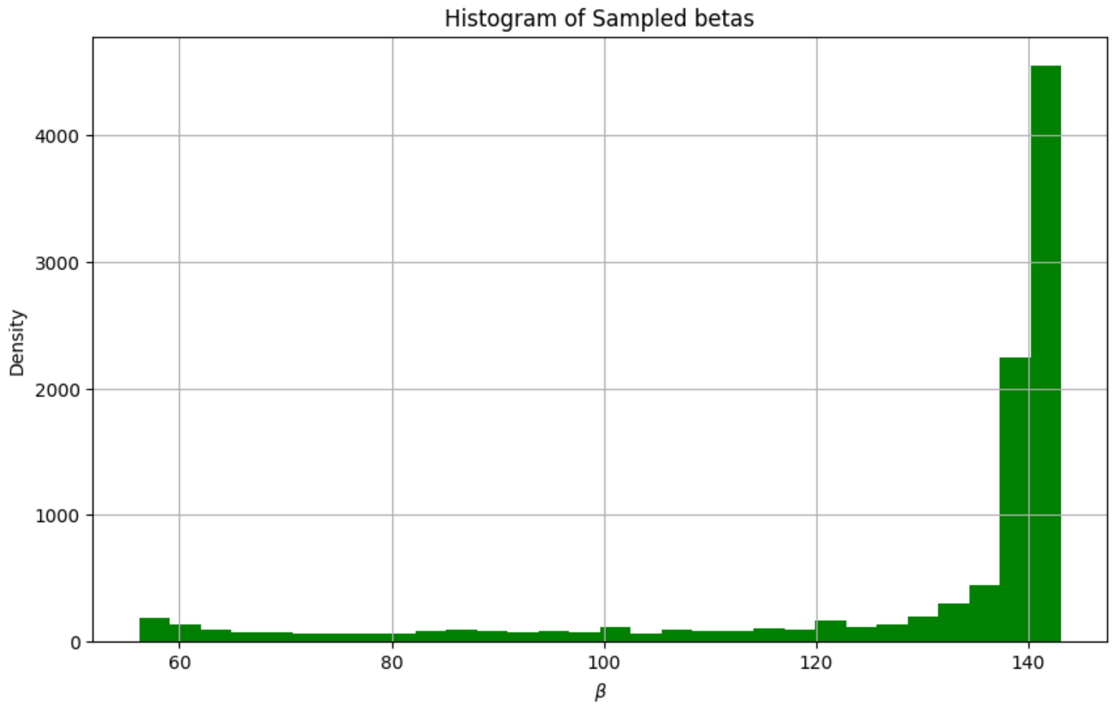
```python
s = 30.

def posterior(data, a, b):
    t = (data['type'] == 'non-word').astype(float)
    m = a + t*b
    p = norm.logpdf(data['RT'], m, s) + norm.logpdf(a, 400, 50) + (t)*abs(norm.
    ↪logpdf(b, 0, 50))
    p = np.sum(p)
    return p



i = 1
step = 0.08
reject = 0
num_samples = 10000
a_samples = np.zeros(num_samples)
b_samples = np.zeros(num_samples)
a_samples[0] = np.random.normal(400, 50)
b_samples[0] = abs(np.random.normal(0, 50))
prev_post = posterior(data, a_samples[0], b_samples[0])

while i < num_samples:
    proposed_a = np.random.normal(a_samples[i-1], step)
    proposed_b = abs(np.random.normal(b_samples[i-1], step))
    post_new = posterior(data, proposed_a, proposed_b)
    h_r = post_new - prev_post
    hastings_ratio = np.exp(h_r)
    p_str = min(1, hastings_ratio)

    if(np.random.uniform(0, 1) < p_str):
        a_samples[i] = proposed_a
        b_samples[i] = proposed_b
        prev_post = post_new
        i += 1
    else:
        reject += 1

print(f"Rejections rate= {(reject/n)*100}")
```

```
Rejections rate= 44.574999999999996
```

2.5.2 Histogram Plotting

[115]:
```
plt.figure(figsize=(10, 6))
plt.hist(a_samples, bins=30, color='purple', label='alpha')
plt.title("Histogram of Sampled alphas")
plt.xlabel("$\\alpha$")
plt.ylabel("Density")
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))
plt.hist(b_samples, bins=30, color='green', label='beta')
plt.title("Histogram of Sampled betas")
plt.xlabel("$\\beta$")
plt.ylabel("Density")
plt.grid(True)
plt.show()
```





Credible Intervals

[116]:
```
burn = 1000
a_samples = a_samples[burn:]
b_samples = b_samples[burn:]
a_cred_interval = np.percentile(a_samples, [2.5, 97.5])
b_cred_interval = np.percentile(b_samples, [2.5, 97.5])

print(f"95% credible interval for alpha: {a_cred_interval}")
print(f"95% credible interval for beta: {b_cred_interval}")
```

```
95% credible interval for alpha: [380.20700143 397.71151624]
95% credible interval for beta: [ 98.9429619  142.26249731]
```

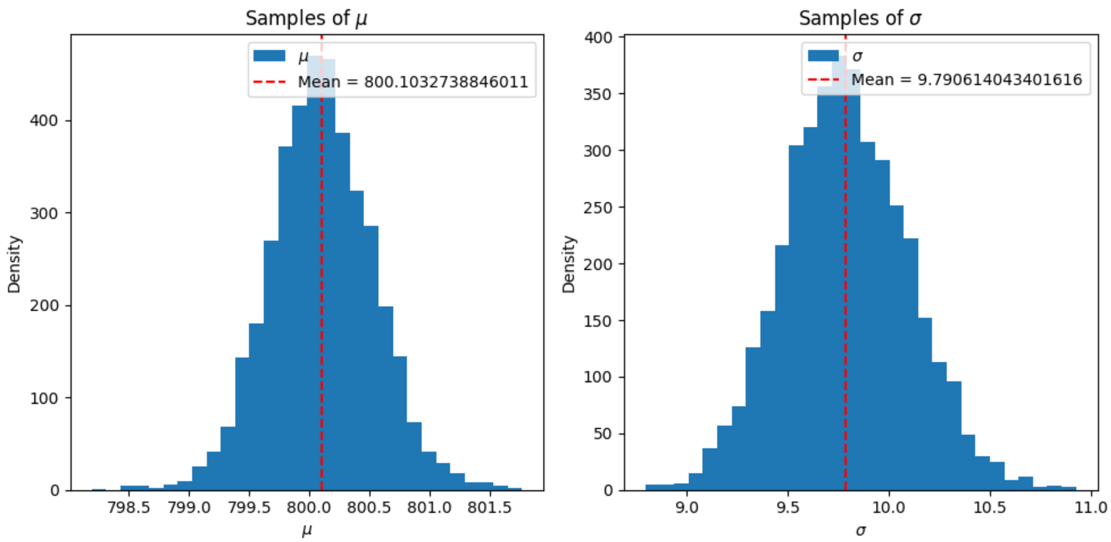
Part 3: Hamiltonian Monte Carlo sampler

3.1 HMC sampler

[117]:
```
mu_true = 800
sigma_true = 100
data = np.random.normal(mu_true, np.sqrt(sigma_true), 500)
```

```python
def grad(mu, sigma, y, n, m, s, a, b):
    grad_mu = (((n*mu) - np.sum(y))/(sigma**2)) + ((mu - m)/(s**2))
    grad_sigma = (n/sigma) - (np.sum((y - mu)**2)/(sigma**3)) + ((sigma - a)/(b**2))
    return np.array([grad_mu, grad_sigma])

def V(mu, sigma, y, n, m, s, a, b):
    nlpd = -(np.sum(norm.logpdf(y, mu, sigma)) + norm.logpdf(mu, m, s) + norm.logpdf(sigma, a, b))
    return nlpd

def HMC(data, n, m, s, a, b, step, L, initial_q, nsamp, nburn):
    mu_samples = np.empty(nsamp)
    sigma_samples = np.empty(nsamp)
    reject = 0
    mu_samples[0] = initial_q[0]
    sigma_samples[0] = initial_q[1]
    i = 0
    while i < nsamp - 1:
        q = np.array([mu_samples[i], sigma_samples[i]])
        p = np.random.normal(0, 1, size=len(q))
        current_q = q.copy()
        current_p = p.copy()
        current_V = V(current_q[0], current_q[1], data, n, m, s, a, b)
        current_T = np.sum(current_p**2)/2
        for l in range(L):
            p -= ((step/2)*grad(q[0], q[1], data, n, m, s, a, b))
            q += (step*p)
            p -= ((step/2)*grad(q[0], q[1], data, n, m, s, a, b))
        proposed_q = q
        proposed_p = p
        proposed_V = V(proposed_q[0], proposed_q[1], data, n, m, s, a, b)
        proposed_T = np.sum(proposed_p**2)/2
        delta_energy = (current_V + current_T) - (proposed_V + proposed_T)
        if(delta_energy < 0.):
            mu_samples[i+1] = proposed_q[0]
            sigma_samples[i+1] = proposed_q[1]
            i += 1
        else:
            if(delta_energy > 100):
                accept_prob = 1
            else:
                accept_prob = min(1, np.exp(delta_energy))
            if(accept_prob > np.random.uniform(0,1)):
                mu_samples[i+1] = proposed_q[0]
                sigma_samples[i+1] = proposed_q[1]
                i += 1
```

```
            else:
                reject += 1
    posteriors = pd.DataFrame({'mu_samples': mu_samples[nburn:], 'sigma_samples': sigma_samples[nburn:]})
    posteriors['sample_id'] = np.arange(1, len(posteriors) + 1)
    return posteriors

n = len(data)
mean_prior = 1000
std_prior = 100
a_prior = 10
b_prior = 2
step_size = 0.02
leapfrog_steps = 12
initial_values = [1000, 11]
burn_samples = 2000

posteriors = HMC(data, n, mean_prior, std_prior, a_prior, b_prior, step_size, leapfrog_steps, initial_values, 6000, burn_samples)

mean_mu_posterior = np.mean(posteriors['mu_samples'])
mean_sigma_posterior = np.mean(posteriors['sigma_samples'])

fig, axes = plt.subplots(1, 2, figsize=(10, 5))
axes[0].hist(posteriors['mu_samples'], bins=30, label='$\\mu$')
axes[0].axvline(x=mean_mu_posterior, color='r', linestyle='--', label=f'Mean = {mean_mu_posterior}')
axes[0].set_xlabel('$\\mu$')
axes[0].set_ylabel('Density')
axes[0].set_title('Samples of $\\mu$')
axes[0].legend()

axes[1].hist(posteriors['sigma_samples'], bins=30, label='$\\sigma$')
axes[1].axvline(x=mean_sigma_posterior, color='r', linestyle='--', label=f'Mean = {mean_sigma_posterior}')
axes[1].set_xlabel('$\\sigma$')
axes[1].set_ylabel('Density')
axes[1].set_title('Samples of $\\sigma$')
axes[1].legend()

plt.tight_layout()
plt.show()
print("\nConclusions for Exercise 3.1:")
print("The posterior distributions for both μ and  have converged to certain values.")
print("The mean values of the posteriors provide estimates for the parameters μ and .")
```



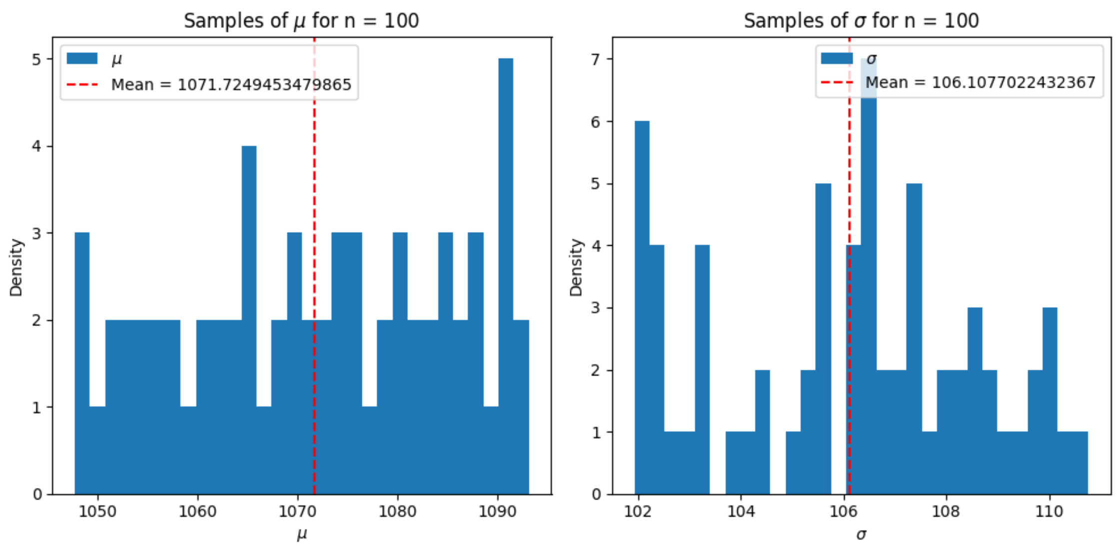

```
Conclusions for Exercise 3.1:
The posterior distributions for both μ and   have converged to certain values.
The mean values of the posteriors provide estimates for the parameters μ and  .
```

3.2 posterior sensitivity

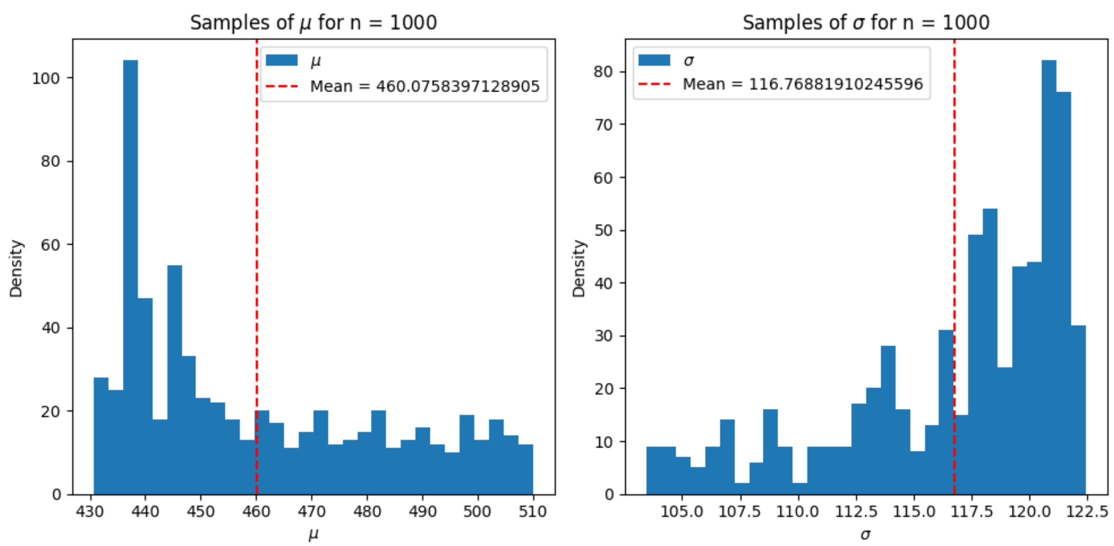
[130]:
```python
n_ = [100, 1000, 6000]
for n in n_:
    posteriors_ = HMC(data, n, mean_prior, std_prior, a_prior, b_prior,␣
 ↪step_size, leapfrog_steps, initial_values, n, n // 3)
    print(f"Posterior Distribution for n = {n}\n")

    mean_mu = np.mean(posteriors_['mu_samples'])
    mean_sigma = np.mean(posteriors_['sigma_samples'])

    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    axes[0].hist(posteriors_['mu_samples'], bins=30, label='$\\mu$')
    axes[0].axvline(x=mean_mu, color='r', linestyle='--', label=f'Mean =␣
 ↪{mean_mu}')
    axes[0].set_xlabel('$\\mu$')
    axes[0].set_ylabel('Density')
    axes[0].set_title(f'Samples of $\\mu$ for n = {n}')
    axes[0].legend()

    axes[1].hist(posteriors_['sigma_samples'], bins=30, label='$\\sigma$')
    axes[1].axvline(x=mean_sigma, color='r', linestyle='--', label=f'Mean =␣
 ↪{mean_sigma}')
```

```
    axes[1].set_xlabel('$\\sigma$')
    axes[1].set_ylabel('Density')
    axes[1].set_title(f'Samples of $\\sigma$ for n = {n}')
    axes[1].legend()

    plt.tight_layout()
    plt.show()
```

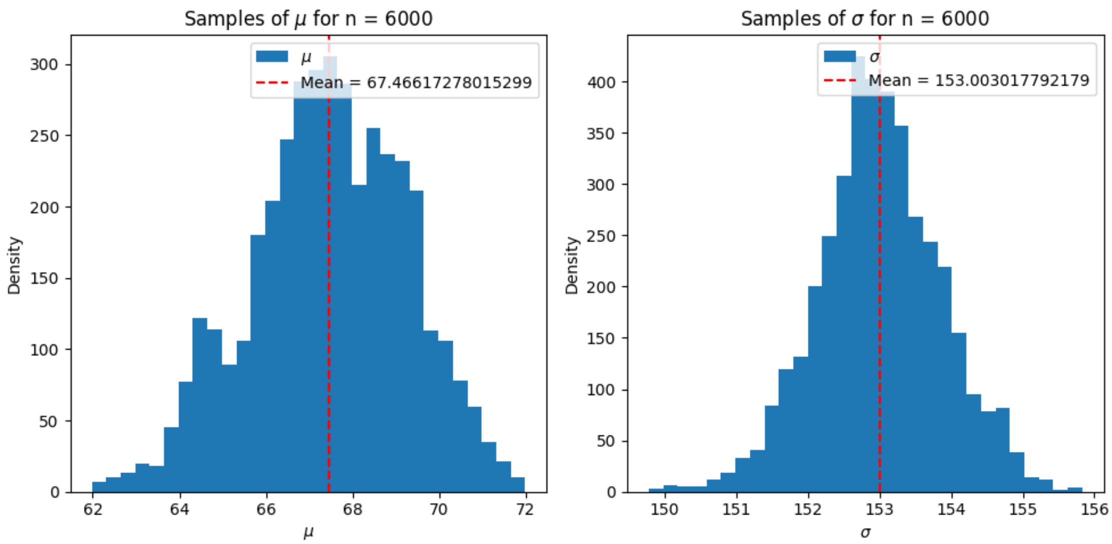
```
Posterior Distribution for n = 100
```




```
Posterior Distribution for n = 1000
```



Posterior Distribution for n = 6000



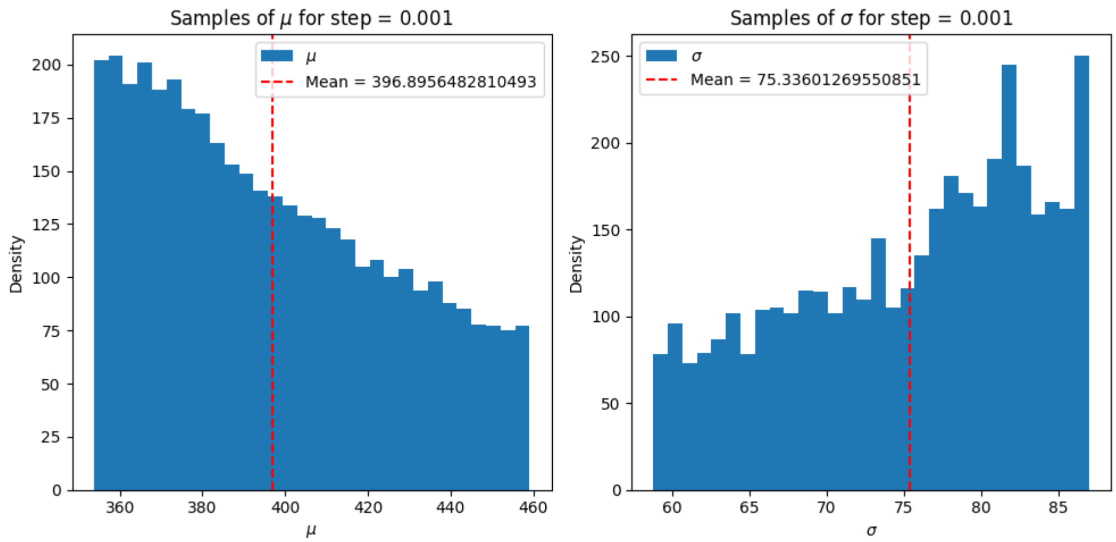
[131]:
```
print("\nHere are the findings from the graphs:")
    print("(1). For Large nsamp values (6000), the posterior is more precise␣
↪and less steep.")
    print("(2). For nsamp values around the same order (1000, 6000), the mean␣
↪value remains close.")
    print("(3). For smaller nsamp values (100), the values are more random due␣
↪to fewer samples after burn-in. Drawing conclusions on mean or sd values is␣
↪unreliable with very small samples.")
    print("Conclusion: As the sample size increases, the posterior␣
↪distributions become more stable and the estimates for µ and   tend to␣
↪converge.")
```

```
Here are the findings from the graphs:
(1). For Large nsamp values (6000), the posterior is more precise and less
steep.
(2). For nsamp values around the same order (1000, 6000), the mean value remains
close.
(3). For smaller nsamp values (100), the values are more random due to fewer
samples after burn-in. Drawing conclusions on mean or sd values is unreliable
with very small samples.
Conclusion: As the sample size increases, the posterior distributions become
more stable and the estimates for µ and   tend to converge.
```

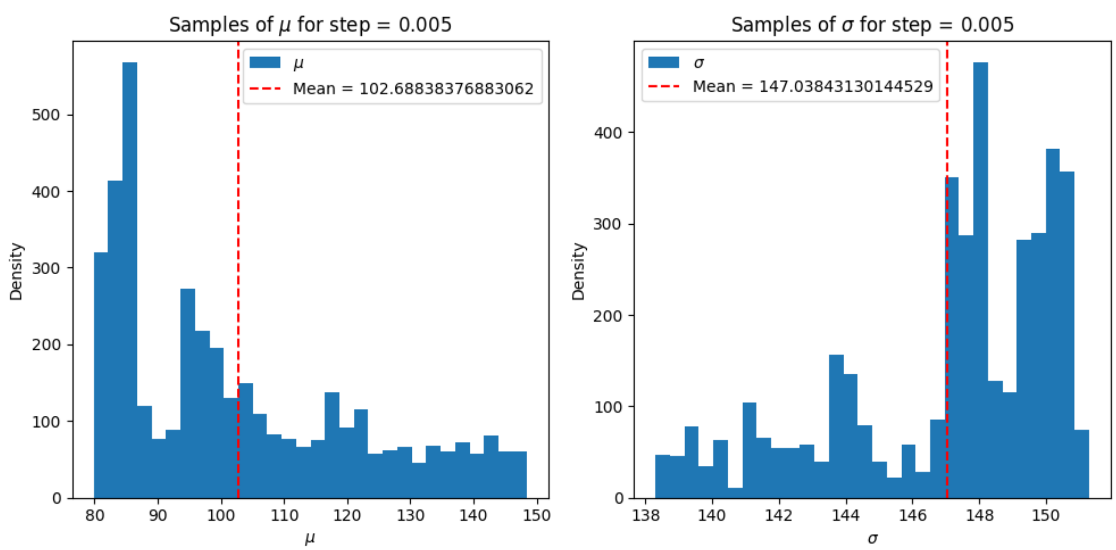
3.3 posteriors change with change in step-size parameter

[121]:
```
step_ = [0.001, 0.005, 0.02]
for step in step_:
    posteriors_ = HMC(data, n, mean_prior, std_prior, a_prior, b_prior, step,␣
 ↪leapfrog_steps, initial_values, 6000, burn_samples)
    print(f"Posterior Distribution for step = {step}\n")

    mean_mu = np.mean(posteriors_['mu_samples'])
    mean_sigma = np.mean(posteriors_['sigma_samples'])

    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    axes[0].hist(posteriors_['mu_samples'], bins=30, label='$\\mu$')
    axes[0].axvline(x=mean_mu, color='r', linestyle='--', label=f'Mean =␣
 ↪{mean_mu}')
    axes[0].set_xlabel('$\\mu$')
    axes[0].set_ylabel('Density')
    axes[0].set_title(f'Samples of $\\mu$ for step = {step}')
    axes[0].legend()

    axes[1].hist(posteriors_['sigma_samples'], bins=30, label='$\\sigma$')
    axes[1].axvline(x=mean_sigma, color='r', linestyle='--', label=f'Mean =␣
 ↪{mean_sigma}')
    axes[1].set_xlabel('$\\sigma$')
    axes[1].set_ylabel('Density')
    axes[1].set_title(f'Samples of $\\sigma$ for step = {step}')
    axes[1].legend()

    plt.tight_layout()
    plt.show()
```

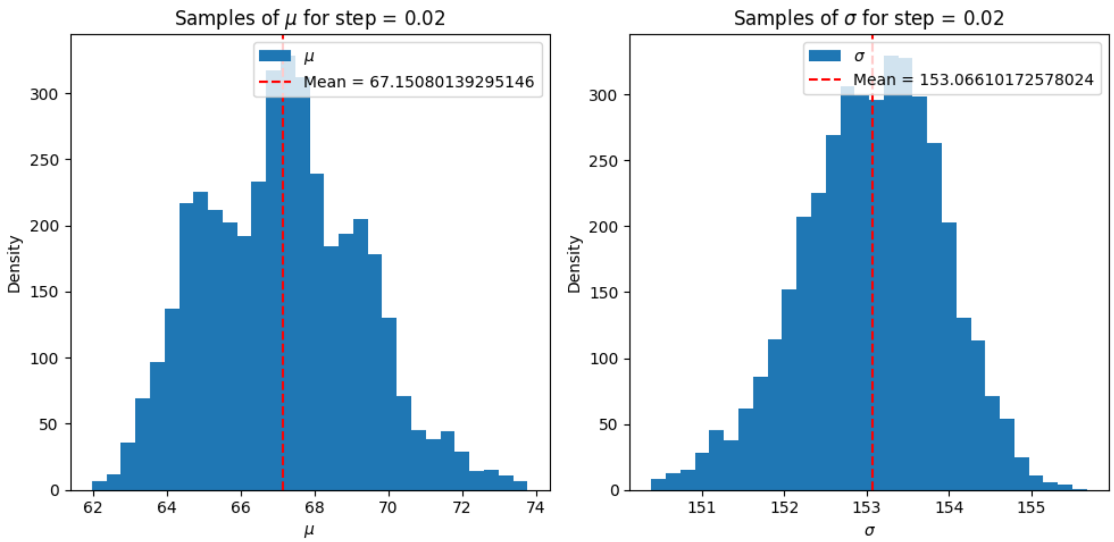
```
Posterior Distribution for step = 0.001
```



Posterior Distribution for step = 0.005



Posterior Distribution for step = 0.02



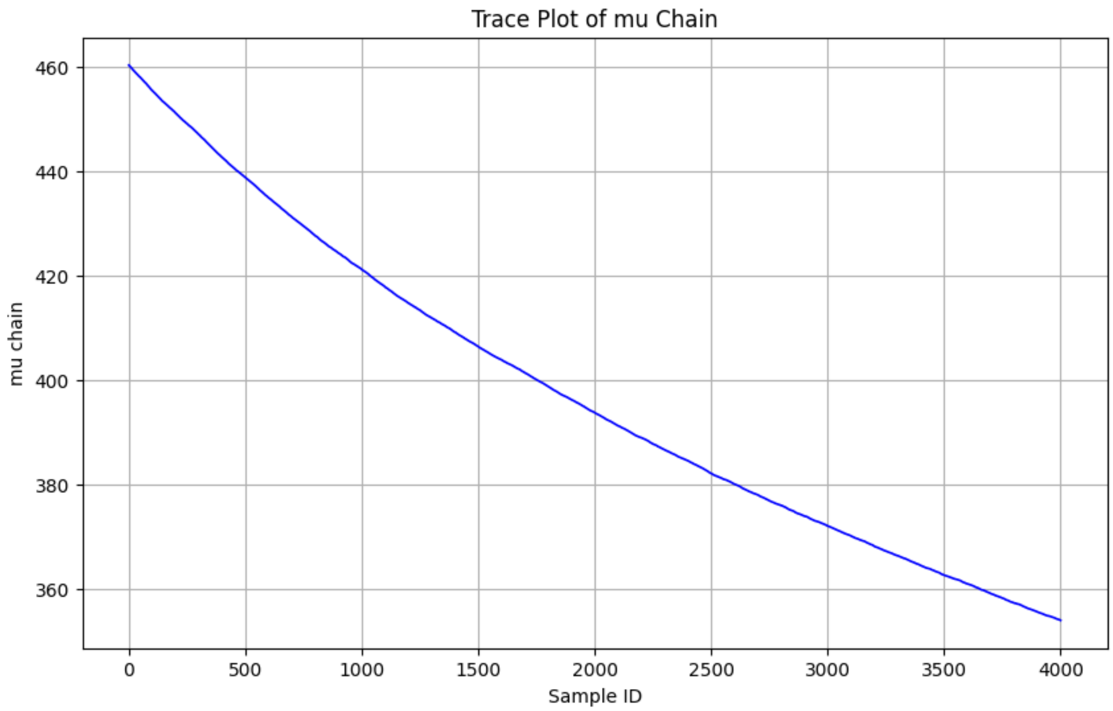
[125]:
```
print("\nBy comparing step sizes:")
print("(1). A very small step-size leads to sampling similar to smaller nsamp.")
print("(2). A significantly large step-size (0.2) results in sampling similar␣
 ↪to larger nsamp.")
print("(3). Increasing step-size shifts the mean of the distribution left for␣
 ↪mu_samples, but not as much for sigma_samples.")
print("(4). For step sizes 0.001 and 0.005, samples follow a linear path, while␣
 ↪for 0.2, samples traverse a wider range. This is evident in the 'Trace Plot'.
 ↪")
print("Conclusion: Optimal step sizes balance exploration and exploitation of␣
 ↪the parameter space in HMC sampling.")
```

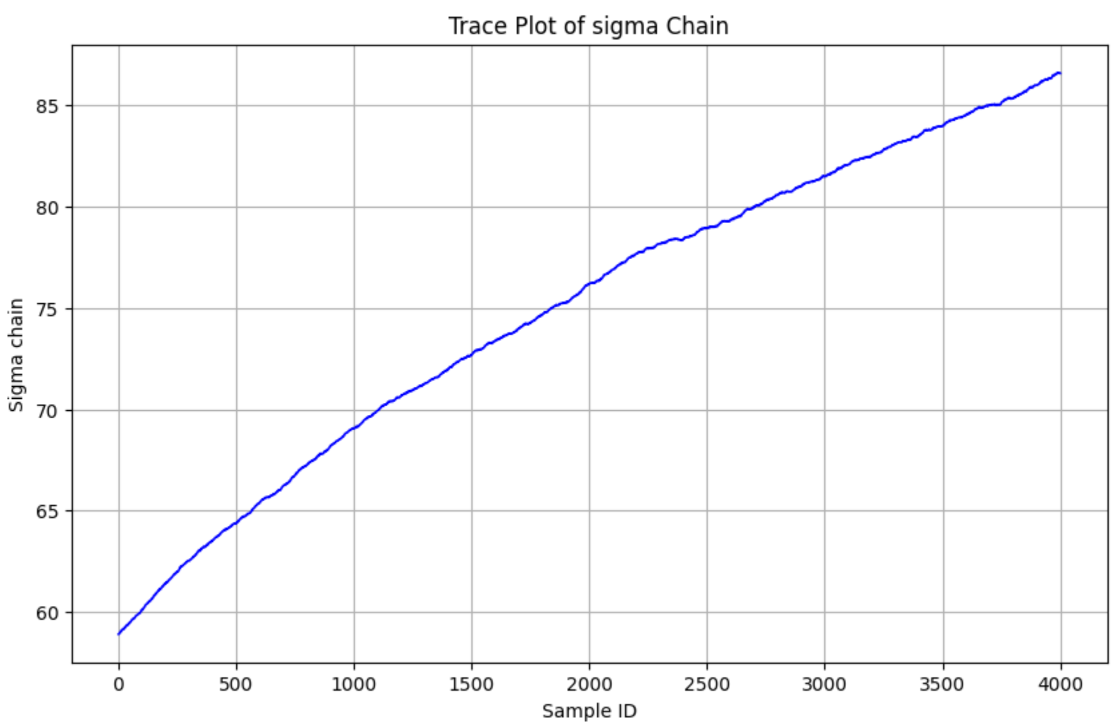
```
By comparing step sizes:
(1). A very small step-size leads to sampling similar to smaller nsamp.
(2). A significantly large step-size (0.2) results in sampling similar to larger
nsamp.
(3). Increasing step-size shifts the mean of the distribution left for
mu_samples, but not as much for sigma_samples.
(4). For step sizes 0.001 and 0.005, samples follow a linear path, while for
0.2, samples traverse a wider range. This is evident in the 'Trace Plot'.
Conclusion: Optimal step sizes balance exploration and exploitation of the
parameter space in HMC sampling.
```

3.4 Visually inspect the mu and sigma chains

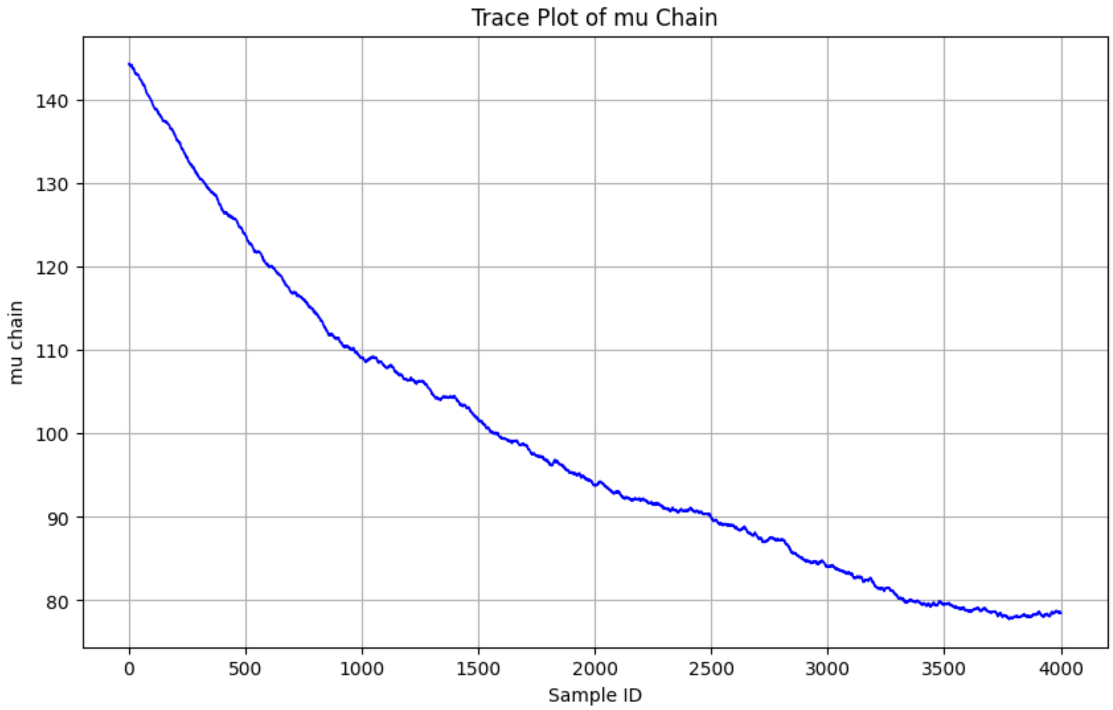
[126]:
```
import seaborn as sns

step_sizes = [0.001, 0.005, 0.02]
for step in step_sizes:
    posteriors_ = HMC(data, n, mean_prior, std_prior, a_prior, b_prior, step,␣
 ↪leapfrog_steps, initial_values, 6000, burn_samples)
    print(f"Posterior Distribution for step = {step}\n")

    plt.figure(figsize=(10, 6))
    sns.lineplot(data=posteriors_, x='sample_id', y='mu_samples', color='blue',␣
 ↪linewidth=1.2)
    plt.xlabel("Sample ID")
    plt.ylabel("mu chain")
    plt.title("Trace Plot of mu Chain")
    plt.grid(True)
    plt.show()

    plt.figure(figsize=(10, 6))
    sns.lineplot(data=posteriors_, x='sample_id', y='sigma_samples',␣
 ↪color='blue', linewidth=1.2)
    plt.xlabel("Sample ID")
    plt.ylabel("Sigma chain")
    plt.title("Trace Plot of sigma Chain")
    plt.grid(True)
    plt.show()
```

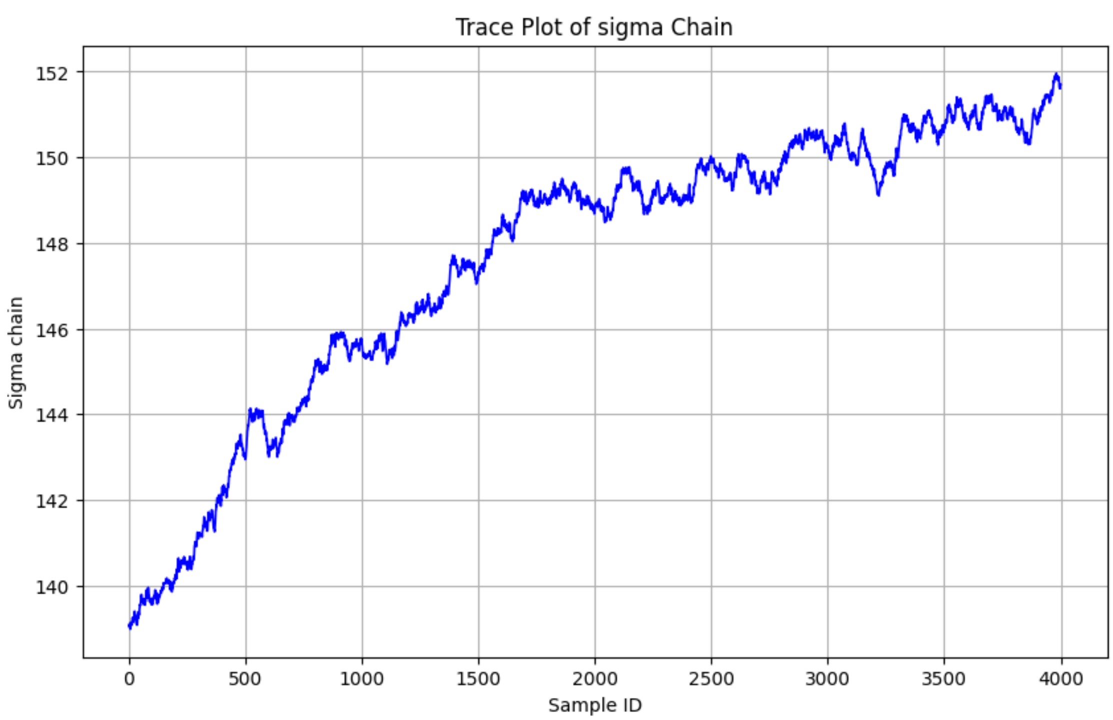
```
Posterior Distribution for step = 0.001
```



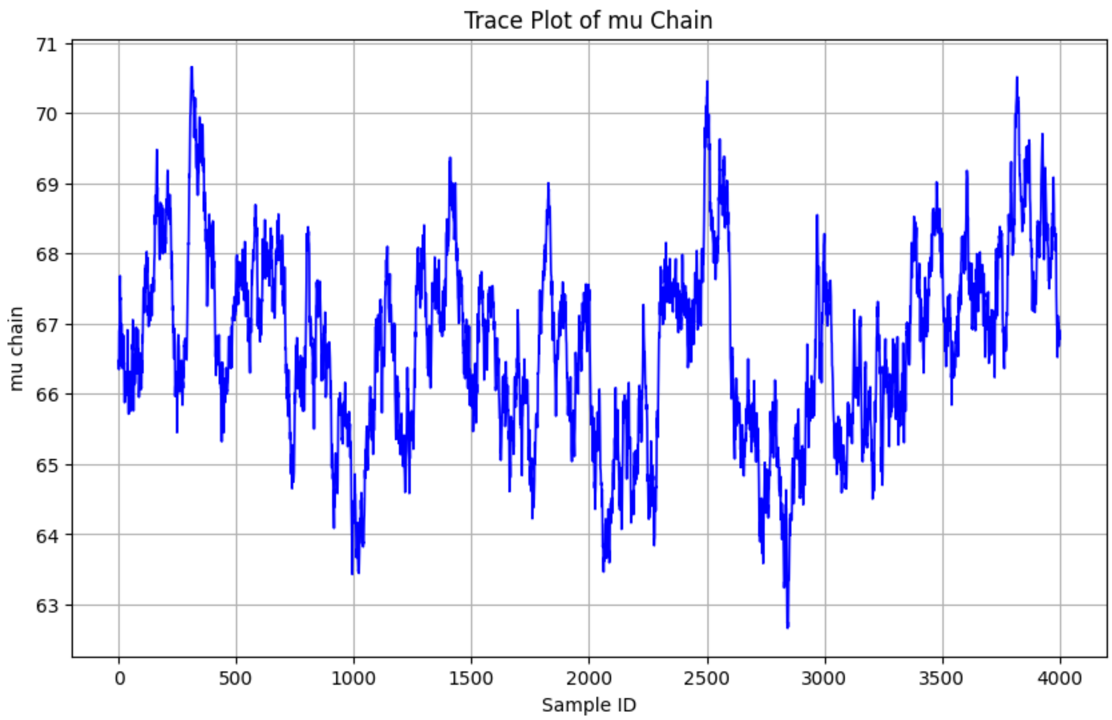


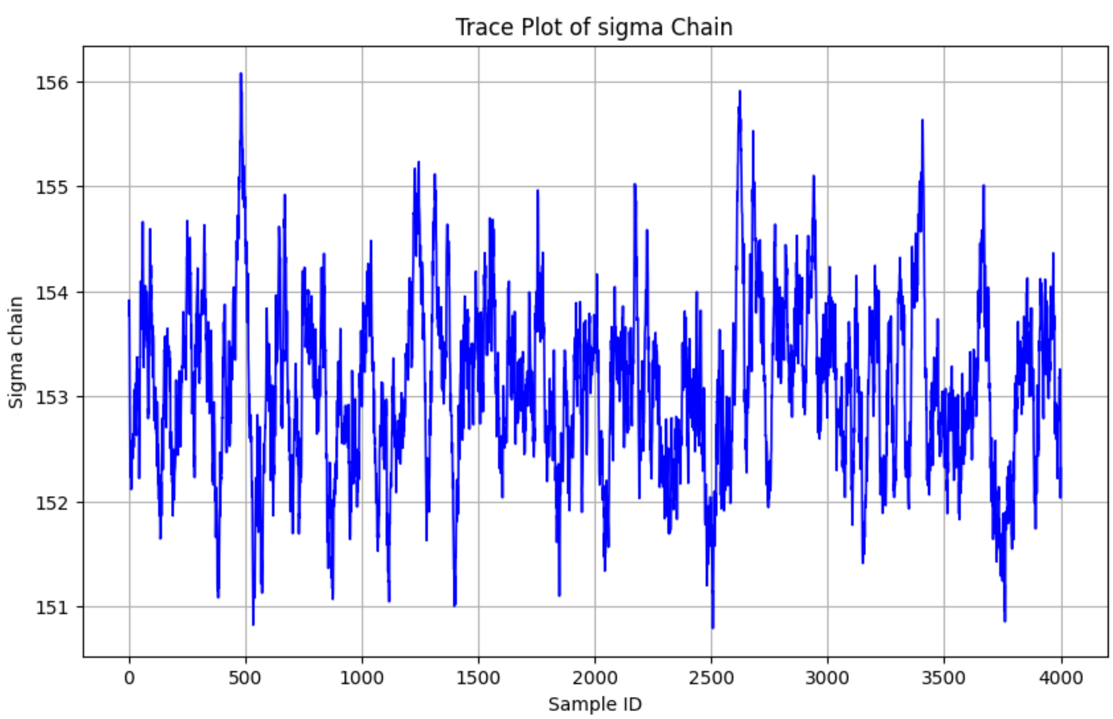
`Posterior Distribution for step = 0.005`


Trace Plot of mu Chain
mu chain
Sample ID
140
130
120
110
100
90
80
0
500
1000
1500
2000
2500
3000
3500
4000


Trace Plot of sigma Chain
Sigma chain
Sample ID
152
150
148
146
144
142
140
0
500
1000
1500
2000
2500
3000
3500
4000

```
Posterior Distribution for step = 0.02
```





[127]:
```python
print("- Increasing step_size leads to less steep mu_samples and a leftward␣
↪shift in their mean. But for sigma_samples, the mean increases from 0.001 to␣
↪0.005.")
print("Posterior Distribution for step = 0.001")
print("Posterior Distribution for step = 0.005")
print("Posterior Distribution for step = 0.02")
```

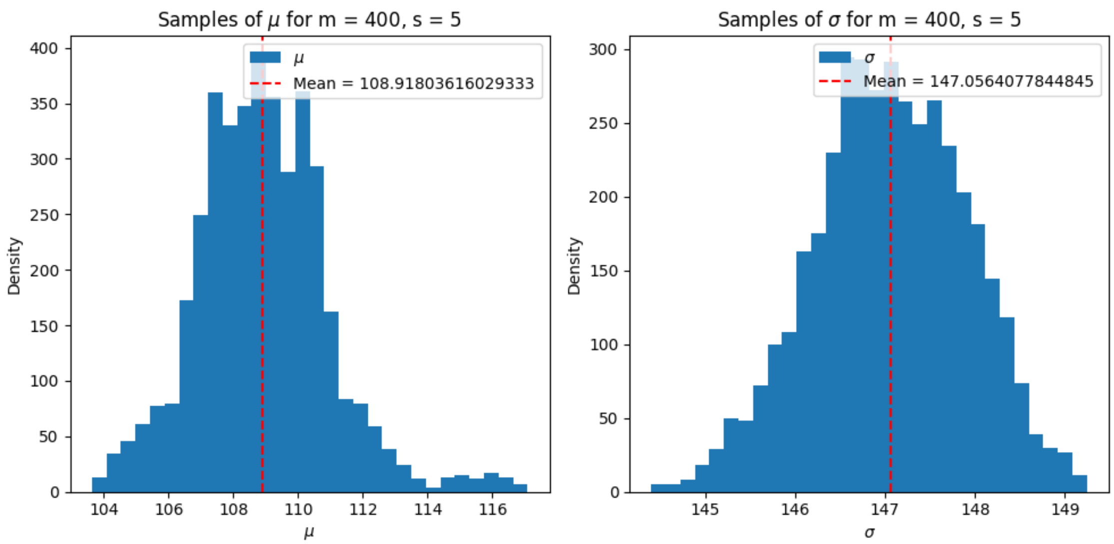
```
- Increasing step_size leads to less steep mu_samples and a leftward shift in
their mean. But for sigma_samples, the mean increases from 0.001 to 0.005.
Posterior Distribution for step = 0.001
Posterior Distribution for step = 0.005
Posterior Distribution for step = 0.02
```

3.5 prior sensitivity for the μ .Comparison of the posterior distribution of μ

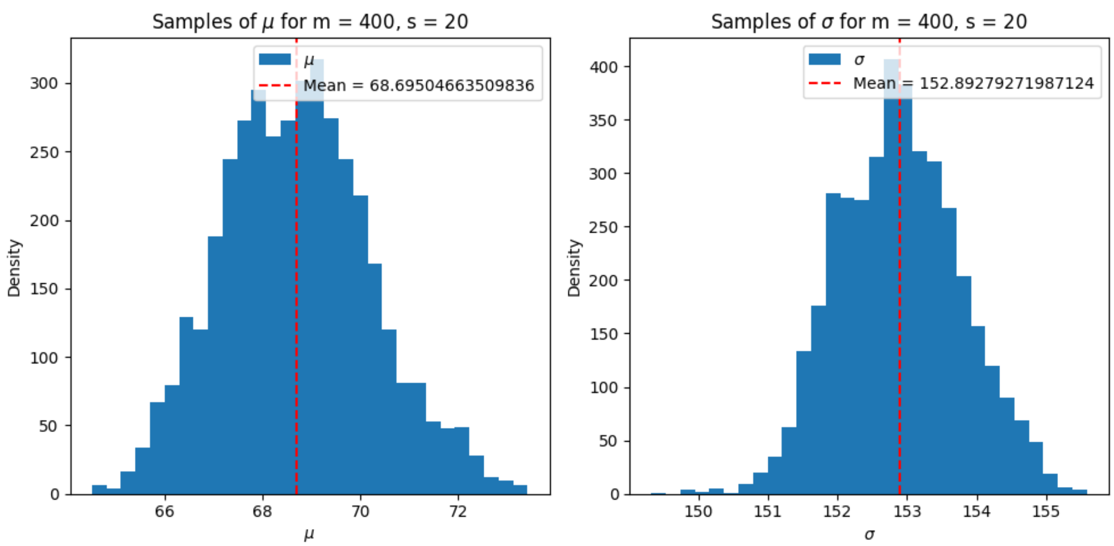
[128]:
```python
mu_prior_vals = [400, 400, 1000, 1000, 1000]
sigma_prior_vals = [5, 20, 5, 20, 100]

for m_val, s_val in zip(mu_prior_vals, sigma_prior_vals):
    y_posterior = HMC(data, n, m_val, s_val, a_prior, b_prior, step_size,␣
↪leapfrog_steps, initial_values, 6000, burn_samples)
    print(f"Posterior Distribution for m = {m_val}, s = {s_val}\n")

    mean_mu = np.mean(y_posterior['mu_samples'])
    mean_sigma = np.mean(y_posterior['sigma_samples'])

    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    axes[0].hist(y_posterior['mu_samples'], bins=30, label='$\\mu$')
    axes[0].axvline(x=mean_mu, color='r', linestyle='--', label=f'Mean =␣
↪{mean_mu}')
    axes[0].set_xlabel('$\\mu$')
    axes[0].set_ylabel('Density')
    axes[0].set_title(f'Samples of $\\mu$ for m = {m_val}, s = {s_val}')
    axes[0].legend()

    axes[1].hist(y_posterior['sigma_samples'], bins=30, label='$\\sigma$')
    axes[1].axvline(x=mean_sigma, color='r', linestyle='--', label=f'Mean =␣
↪{mean_sigma}')
    axes[1].set_xlabel('$\\sigma$')
    axes[1].set_ylabel('Density')
    axes[1].set_title(f'Samples of $\\sigma$ for m = {m_val}, s = {s_val}')
    axes[1].legend()
```

```
plt.tight_layout()
plt.show()
```

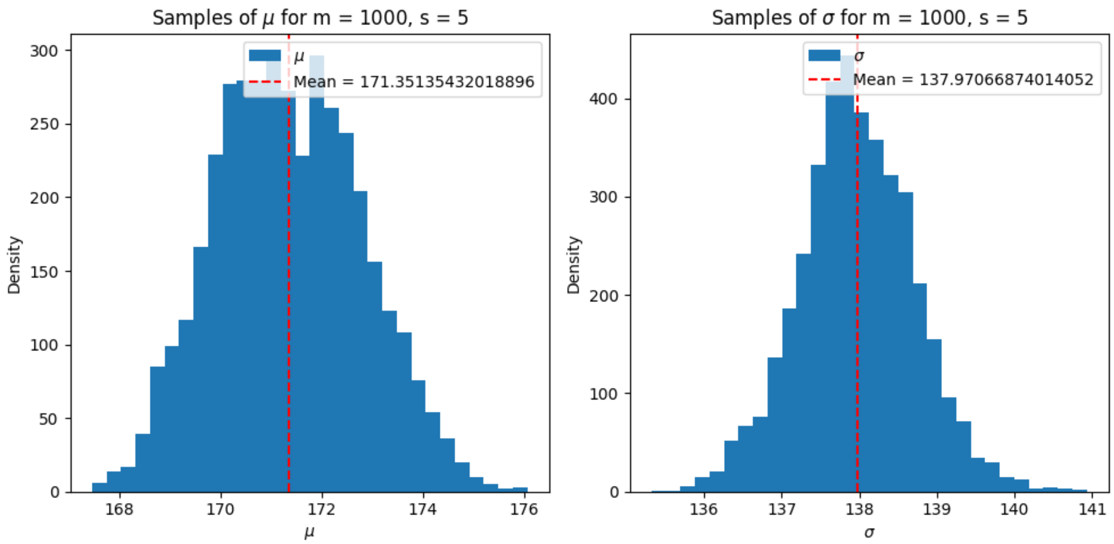
```
Posterior Distribution for m = 400, s = 5
```



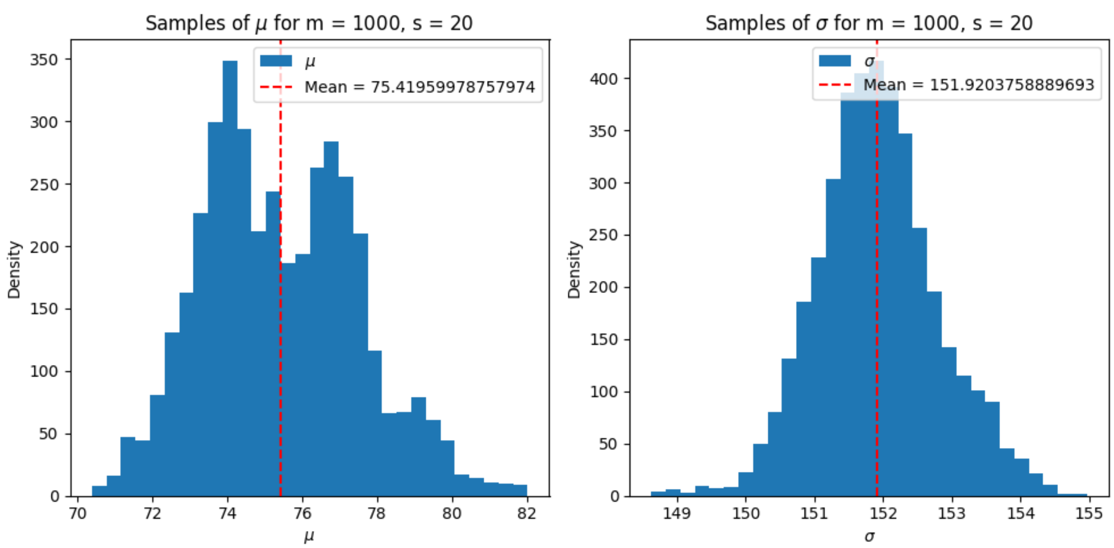

```
Posterior Distribution for m = 400, s = 20
```



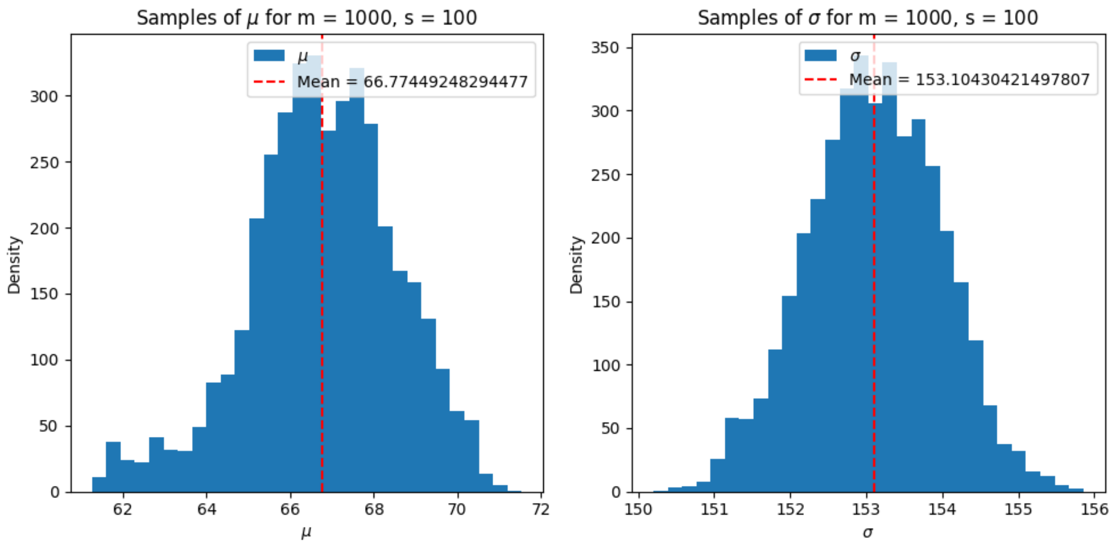

```
Posterior Distribution for m = 1000, s = 5
```



Posterior Distribution for m = 1000, s = 20



Posterior Distribution for m = 1000, s = 100



[129]:
```
print("\nObservations:")
print("(1). For small m values, the mean of the posterior diverges slightly␣
↪from the actual mean due to prior bias, although the difference is minor.")
print("(2). For small s values, there is less spread, resulting in sharp peaks.
↪")
```

```
Observations:
(1). For small m values, the mean of the posterior diverges slightly from the
actual mean due to prior bias, although the difference is minor.
(2). For small s values, there is less spread, resulting in sharp peaks.
```

[129]:
```
```