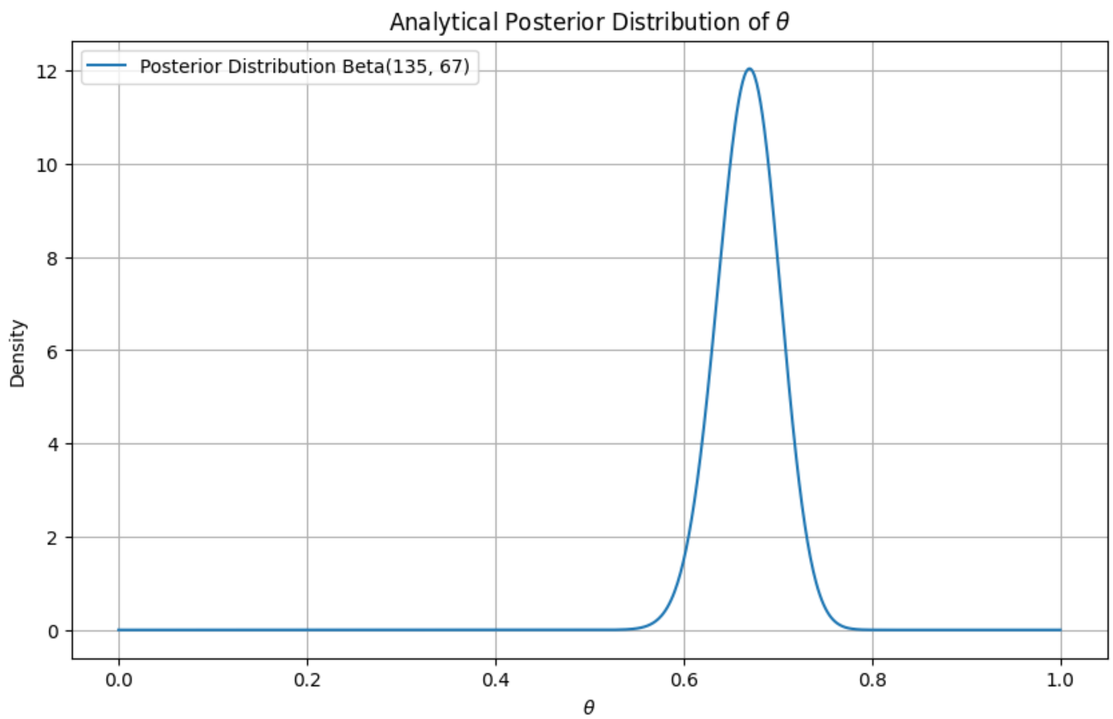
untitled7

June 21, 2024

[23]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import beta,binom,uniform,norm
```

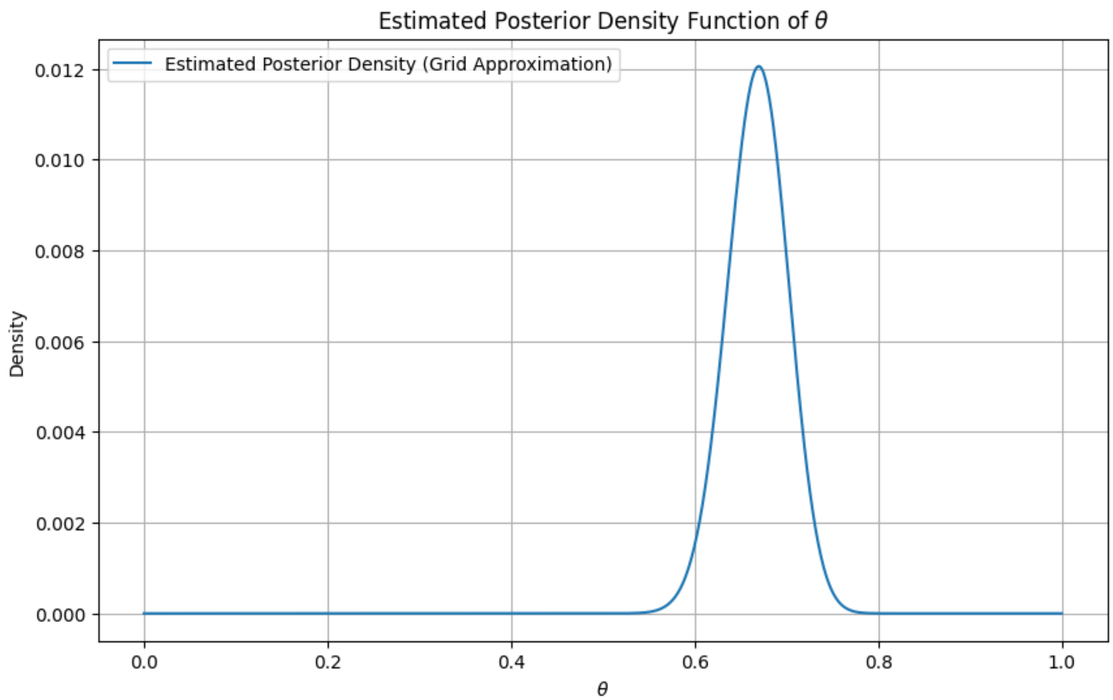
[24]:
```
theta = np.linspace(0, 1, 1000)
analytical_posterior_pdf = beta.pdf(theta, a=135, b=67)
```

[25]:
```
# Plot the posterior distribution
plt.figure(figsize=(10, 6))
plt.plot(theta, analytical_posterior_pdf, label='Posterior Distribution␣
 ↪Beta(135, 67)')
plt.title('Analytical Posterior Distribution of $\\theta$')
plt.xlabel('$\\theta$')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()
```



[26]:
```
data = [10, 15, 15, 14, 14, 14, 13, 11, 12, 16]
n = 20 # no of trials
theta_grid = np.linspace(0, 1, 1000)
prior = np.ones_like(theta_grid) # for beta(1,1) means all priors just 1
likelihood = np.ones_like(theta_grid)
for y in data:
    likelihood *= binom.pmf(y, n, theta_grid)

unnormalized_posterior = prior * likelihood
# Normalize the posterior
posterior = unnormalized_posterior / np.sum(unnormalized_posterior)
```

[27]:
```
# Plot the estimated posterior density function
plt.figure(figsize=(10, 6))
plt.plot(theta_grid, posterior, label='Estimated Posterior Density (Grid␣
 ↪Approximation)')
plt.title('Estimated Posterior Density Function of $\\theta$')
plt.xlabel('$\\theta$')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()
```



[28]:
```
# Q1-3
num_samples = 100000
theta_samples = beta.rvs(1, 1, size=num_samples)
likelihoods = np.ones(num_samples)
for y in data:
    likelihoods *= binom.pmf(y, n, theta_samples)

marginal_likelihood = np.mean(likelihoods)
print(f'Marginal Likelihood = {marginal_likelihood}')
```

```
Marginal Likelihood = 1.4046500608815386e-10
```

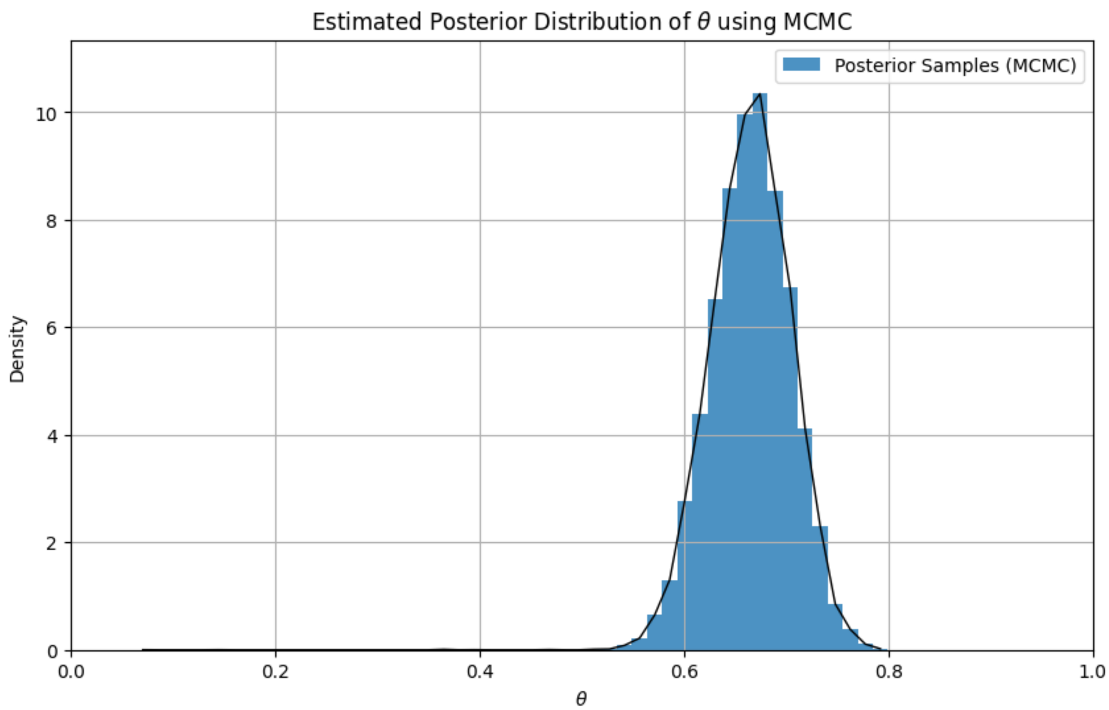
[29]:
```
# Q1-4
proposal_samples = uniform.rvs(0, 1, size=num_samples)
likelihoods = np.ones(num_samples)
for y in data:
    likelihoods *= binom.pmf(y, n, proposal_samples)

priors = beta.pdf(proposal_samples, 1, 1)
proposal_density = uniform.pdf(proposal_samples, 0, 1)

weights = likelihoods * priors / proposal_density
# Normalize the weights
weights /= np.sum(weights)
```

```
# dataframe
df = pd.DataFrame({'theta': proposal_samples, 'weights': weights})
# Sample N/4 samples from the initial samples based on their weights
posterior_samples = df.sample(n=num_samples//4, weights='weights',␣
↪replace=True)['theta']
# Display the posterior samples
posterior_samples.head()
```

[29]:
```
87194    0.672540
79435    0.669235
67134    0.622364
26978    0.577717
79802    0.671584
Name: theta, dtype: float64
```

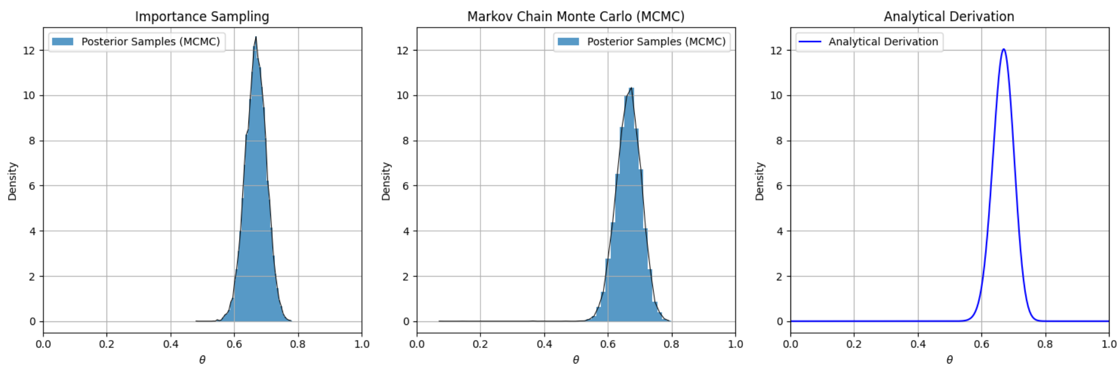
[30]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom, beta, norm

# Number of samples for MCMC
num_samples = 30000

# Data and prior parameters
data = [10, 15, 20, 25, 30]  # Example data, replace with actual data
n = 30  # Number of trials in binomial distribution

# Initialize the theta chain
theta_chain = np.empty(num_samples)
theta_chain[0] = np.random.beta(1, 1)
step_size = 0.08

# MCMC Sampling
for i in range(1, num_samples):
    # Propose a new theta
    proposed_theta = np.random.normal(theta_chain[i - 1], step_size)

    if 0 < proposed_theta < 1:
        # Calculate posterior for the proposed theta
        posterior_proposed = (np.prod([binom.pmf(data_point, n, proposed_theta)␣
↪for data_point in data]) *
                              beta.pdf(proposed_theta, 1, 1))
        # Calculate posterior for the current theta
        posterior_current = (np.prod([binom.pmf(data_point, n, theta_chain[i -␣
↪1]) for data_point in data]) *
                             beta.pdf(theta_chain[i - 1], 1, 1))
        # Hastings ratio
        hastings_ratio = (posterior_proposed *
```

```
                          norm.pdf(theta_chain[i - 1], proposed_theta,␣
↪step_size)) / (
                          posterior_current * norm.pdf(proposed_theta,␣
↪theta_chain[i - 1], step_size))

        # Acceptance probability
        acceptance_prob = min(1, hastings_ratio)

        # Accept or reject the proposed theta
        if np.random.uniform(0, 1) < acceptance_prob:
            theta_chain[i] = proposed_theta
        else:
            theta_chain[i] = theta_chain[i - 1]
    else:
        theta_chain[i] = theta_chain[i - 1]

# Plot the estimated posterior distribution
plt.figure(figsize=(10, 6))
counts, bins, _ = plt.hist(theta_chain, bins=50, density=True, alpha=0.8,␣
↪label='Posterior Samples (MCMC)')
# Calculate bin centers
bin_centers = (bins[:-1] + bins[1:]) / 2
# Plot a line connecting bin centers
plt.plot(bin_centers, counts, linestyle='-', color='black', linewidth=1)
plt.xlim(0, 1)
plt.ylim(0, np.max(counts) + 1)
plt.title('Estimated Posterior Distribution of $\\theta$ using MCMC')
plt.xlabel('$\\theta$')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()
```



[31]:
```
plt.figure(figsize=(15, 5))

# Subplot 1: Importance Sampling
plt.subplot(1, 3, 1)
counts, bins, _ = plt.hist(posterior_samples, bins=50, density=True,␣
↪label='Posterior Samples (MCMC)', alpha=0.75)
# Calculate bin centers
bin_centers = (bins[:-1] + bins[1:]) / 2
# Plot a line connecting bin centers
plt.plot(bin_centers, counts, linestyle='-', color='black', linewidth=0.75)
plt.title('Importance Sampling')
plt.xlabel('$\\theta$')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.xlim(0, 1)
plt.ylim(-0.5, 13)


# Subplot 2: MCMC
plt.subplot(1, 3, 2)
counts, bins, _ = plt.hist(theta_chain, bins=50, density=True, label='Posterior␣
↪Samples (MCMC)', alpha=0.75)
# Calculate bin centers
```

```
bin_centers = (bins[:-1] + bins[1:]) / 2
# Plot a line connecting bin centers
plt.plot(bin_centers, counts, linestyle='-', color='black', linewidth=0.75)
plt.title('Markov Chain Monte Carlo (MCMC)')
plt.xlabel('$\\theta$')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.xlim(0, 1)
plt.ylim(-0.5, 13)

# Subplot 3: Analytical Derivation
plt.subplot(1, 3, 3)
theta__ = np.linspace(0, 1, 1000)
plt.plot(theta__, analytical_posterior_pdf, label='Analytical Derivation',␣
  ↪color='blue')
plt.title('Analytical Derivation')
plt.xlabel('$\\theta$')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.ylim(-0.5, 13)
plt.xlim(0, 1)

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



[31]: