CE410/510: Modern Statistics for Engineers, Homework 3

Completed by Brandon Auyeung for Dr. Peyman Abbaszadeh

Due Date: 25th of February, 2026

Problem 3

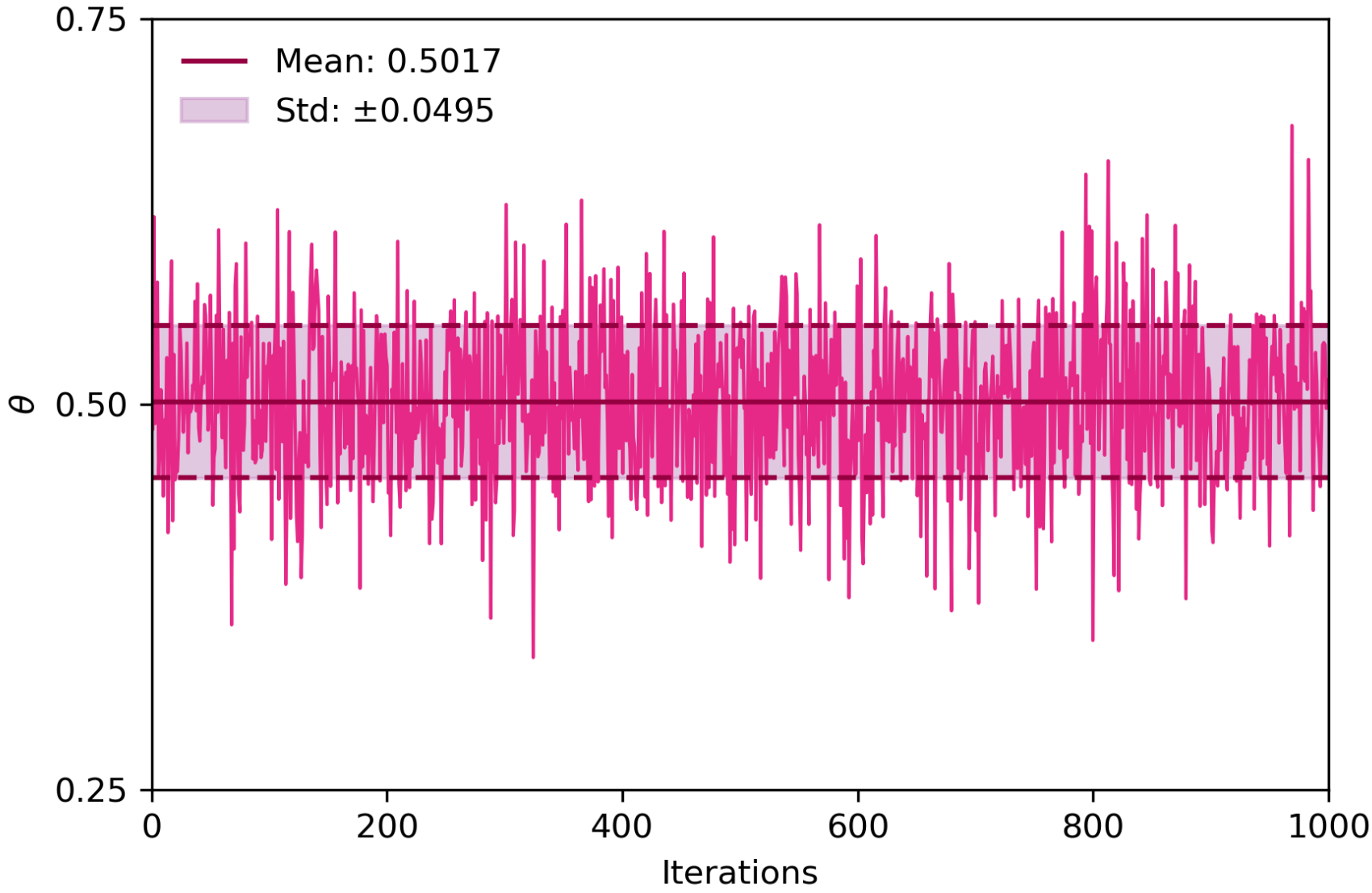
Coin toss problem: Drive the posterior distribution of parameter θ, which is the probability that a coin toss results in heads. Suppose that the prior distribution is an uninformative (or flat) Beta distribution with parameters of $a, b = 1, 1$. We use Binomial likelihood function to quantify the data from our experiment which resulted in 4 heads out of 10 tosses.

Part 3a

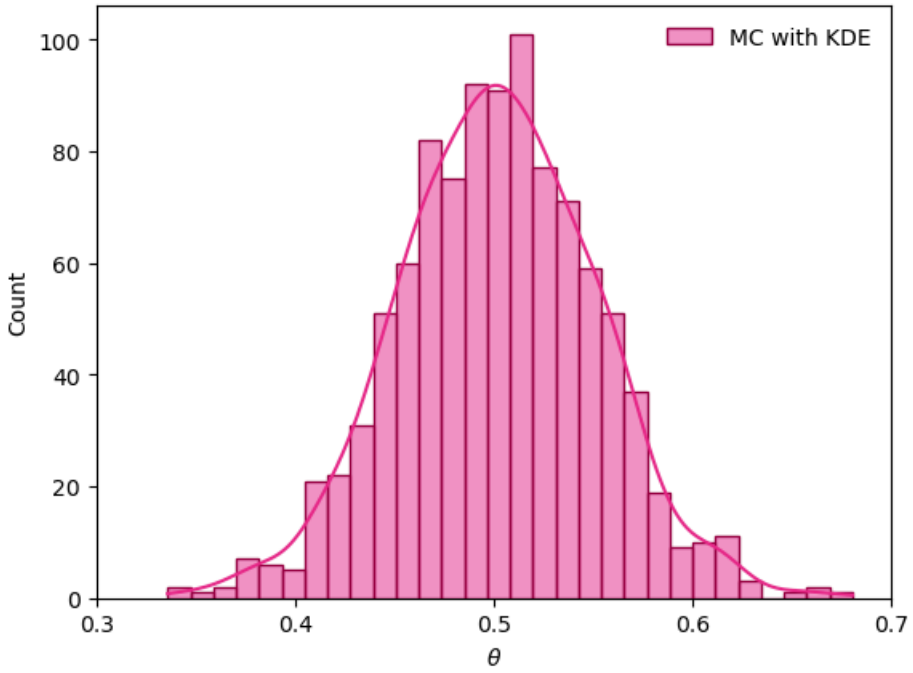
Use Monte Carlo (MC) approach. Visualize the MC evolution and the density plot for θ based on 1000 iterations. Use np.random.seed(5) and the proposal distribution that follows a Gaussian distribution with a mean and standard deviation of 0.5 and 0.05, respectively. Please discuss what the results would be if we increased the iterations to 10000. (2 points)

In [ ]:
```
x = np.linspace(0, 1000, 1000) # define how many iterations will be run
theta_mc = np.zeros(1000) # define empty theta array for monte carlo simulation

np.random.seed(5) # set random seed

for i in range(1000): # for each iteration...
    theta_mc[i] = stats.norm.rvs(loc = 0.5, scale = 0.05, size = 1)[0]
    # draw a sample from a normal distribution with mean 0.5 and standard deviation 0.05

fig, ax = subplots(figsize=(6, 4), dpi=300, tight_layout=True)

ax.plot(x, theta_mc, color = '#e7298a', linewidth = 1) # plot theta values over iterations

# visualize mean and standard deviation of theta samples
ax.hlines(np.mean(theta_mc), xmin = 0, xmax = 1000, color = '#980043', label = f'Mean: {np.mean(theta_mc):.4f} ')
ax.hlines(np.mean(theta_mc) + np.std(theta_mc), xmin = 0, xmax = 1000, linestyles= "--",color = '#980043')
ax.hlines(np.mean(theta_mc) - np.std(theta_mc), xmin = 0, xmax = 1000, linestyles= "--",color = '#980043')
ax.fill_between(x, np.mean(theta_mc) - np.std(theta_mc), np.mean(theta_mc) + np.std(theta_mc), color = '#c994c7', alpha = 0.5, label = f'

# format plot
ax.set_xlabel('Iterations')
ax.set_xlim(0, 1000)
ax.set_ylabel('$\\theta$')
ax.set_ylim(0.25, 0.75)
ax.set_yticks([0.25, 0.5, 0.75])[0]
plt.legend(frameon = False, loc = 'best')
```

Out[ ]: <matplotlib.legend.Legend at 0x7fe181480ad0>



In [ ]:
```
# create and format histogram with kernel density estimate
sns.histplot(theta_mc, kde=True, bins=30, color = '#e7298a', edgecolor = '#980043', label = 'MC with KDE')
plt.xlabel('$\\theta$')
plt.ylabel('Count')
plt.xticks([0.3, 0.4, 0.5, 0.6, 0.7])
plt.legend(frameon = False, loc = 'best')
plt.show()
```



Discussion for Part 3a

As the number of iterations becomes large (i.e. as $N \to \infty$), the resultant posterior distribution approaches the proposal distribution. No evolution can occur since all values draw directly from the proposal distribution. Therefore, if $N = 10000$, the shape of the KDE would approach the proposal distribution's shape.

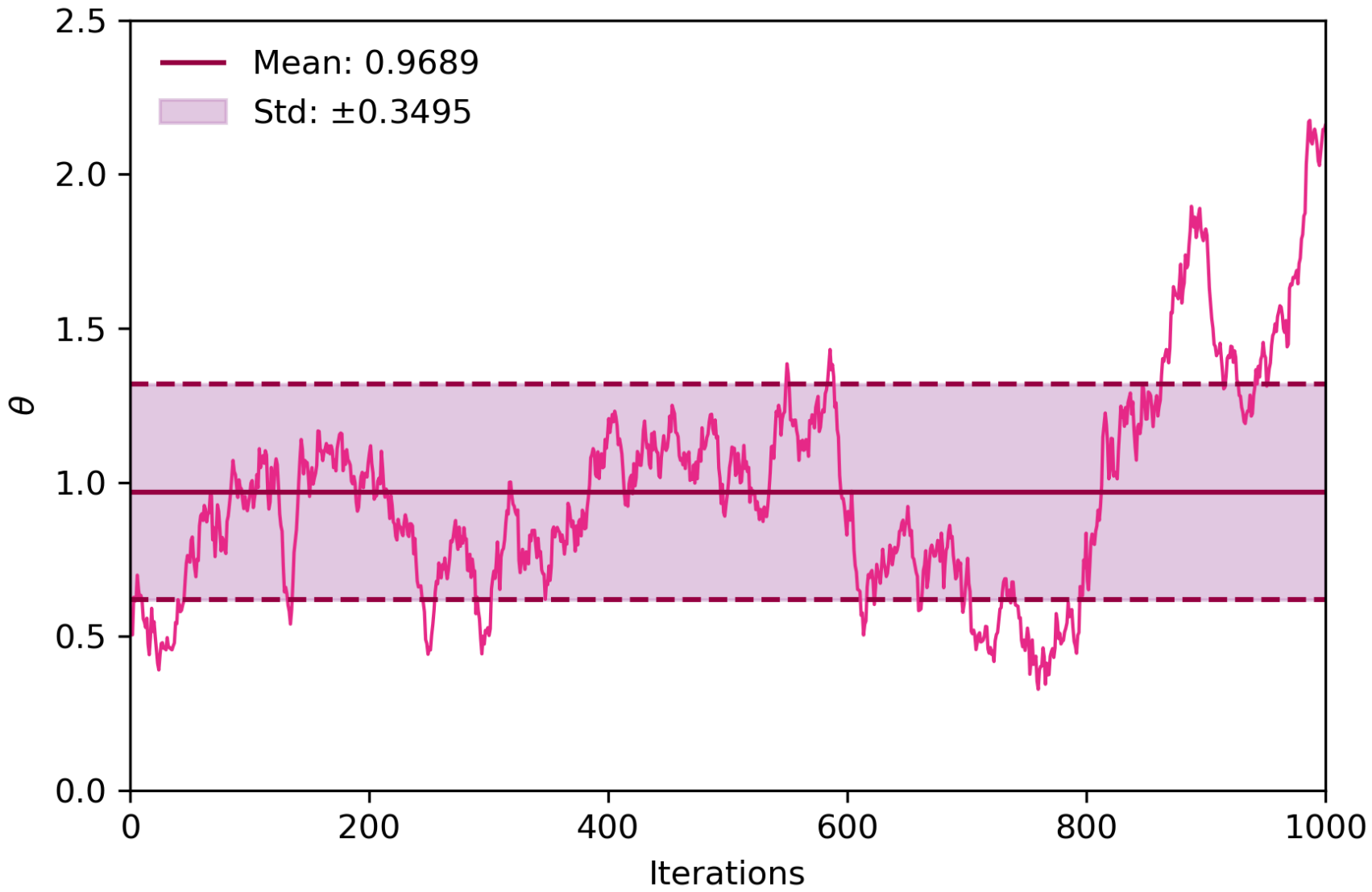
Part 3b

Use Markov Chain Monte Carlo (MCMC) approach. Visualize the MCMC evolution and the density plot for θ based on 1000 iterations. Use np.random.seed(5) and the proposal distribution that follows a Gaussian distribution with a mean and standard deviation of 0.5 and 0.05, respectively. Please discuss what the results would be if we increased the iterations to 10000. (2 points)

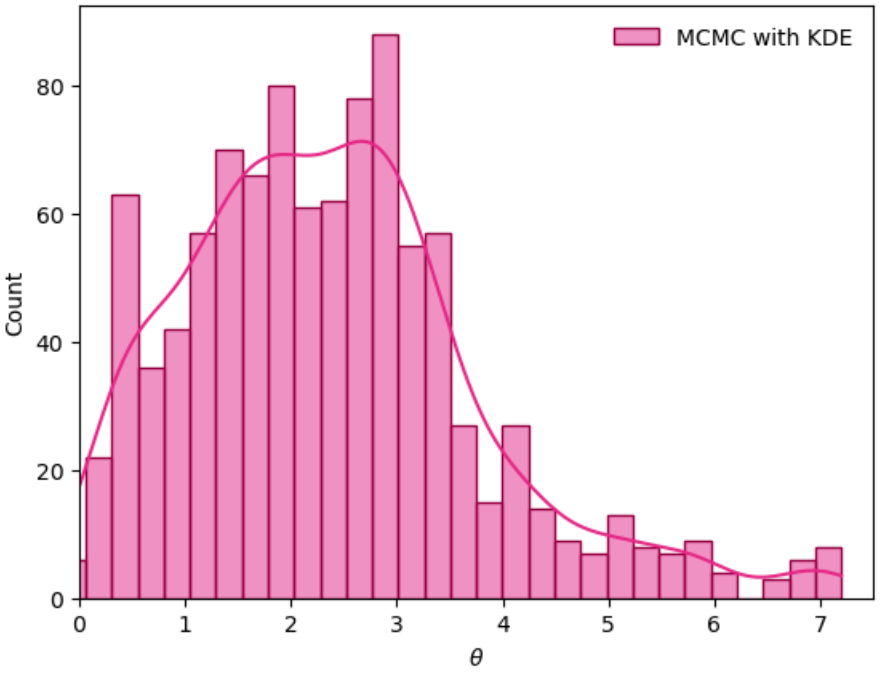
In [81]:
```
theta_mcmc = np.zeros(1000) # define empty theta array for mcmc simulation
np.random.seed(5) # set random seed

for i in range(1000): # for each iteration...
    if i == 0:
        theta_mcmc[i] = 0.5 # set theta to 0.5 for iteration 0
    else:
        theta_mcmc[i] = stats.norm.rvs(loc = theta_mcmc[i-1], scale = 0.05, size = 1)[0]
        # for all other iterations, draw a sample from a normal distribution with mean theta from the previous iteration and standard dev

fig, ax = subplots(figsize=(6,4), dpi=300, tight_layout=True)
```

```
ax.plot(x, theta_mcmc, color = '#e7298a', linewidth = 1) # plot theta values over iterations

# visualize mean and standard deviation of theta samples
ax.hlines(np.mean(theta_mcmc), xmin = 0, xmax = 1000, color = '#980043', label = f'Mean: {np.mean(theta_mcmc):.4f} ')
ax.hlines(np.mean(theta_mcmc) + np.std(theta_mcmc), xmin = 0, xmax = 1000, linestyles= "--",color = '#980043')
ax.hlines(np.mean(theta_mcmc) - np.std(theta_mcmc), xmin = 0, xmax = 1000, linestyles= "--",color = '#980043')
ax.fill_between(x, np.mean(theta_mcmc) - np.std(theta_mcmc), np.mean(theta_mcmc) + np.std(theta_mcmc), color = '#c994c7', alpha = 0.5, la

# format plot
ax.set_xlabel('Iterations')
ax.set_xlim(0, 1000)
ax.set_ylabel('$\\theta$')
ax.set_ylim(0, 2.5)
plt.legend(frameon = False, loc = 'best')
```

Out[81]: <matplotlib.legend.Legend at 0x7fe1814d5160>



In [ ]:
```
# create and format histogram with kernel density estimate
sns.histplot(theta_mcmc, kde=True, bins=30, color = '#e7298a', edgecolor = '#980043', label = 'MCMC with KDE')
plt.xlabel('$\\theta$')
plt.ylabel('Count')
plt.xlim(0, 7.5)
plt.legend(frameon = False, loc = 'best')
plt.show()
```



Discussion for Part 3b

As the number of iterations becomes large (i.e. as $N \to \infty$), the resultant posterior distribution is allowed to evolve without bound. Therefore, if $N = 10000$, the shape of the KDE may spiral into a non-sensical result.

Part 3c

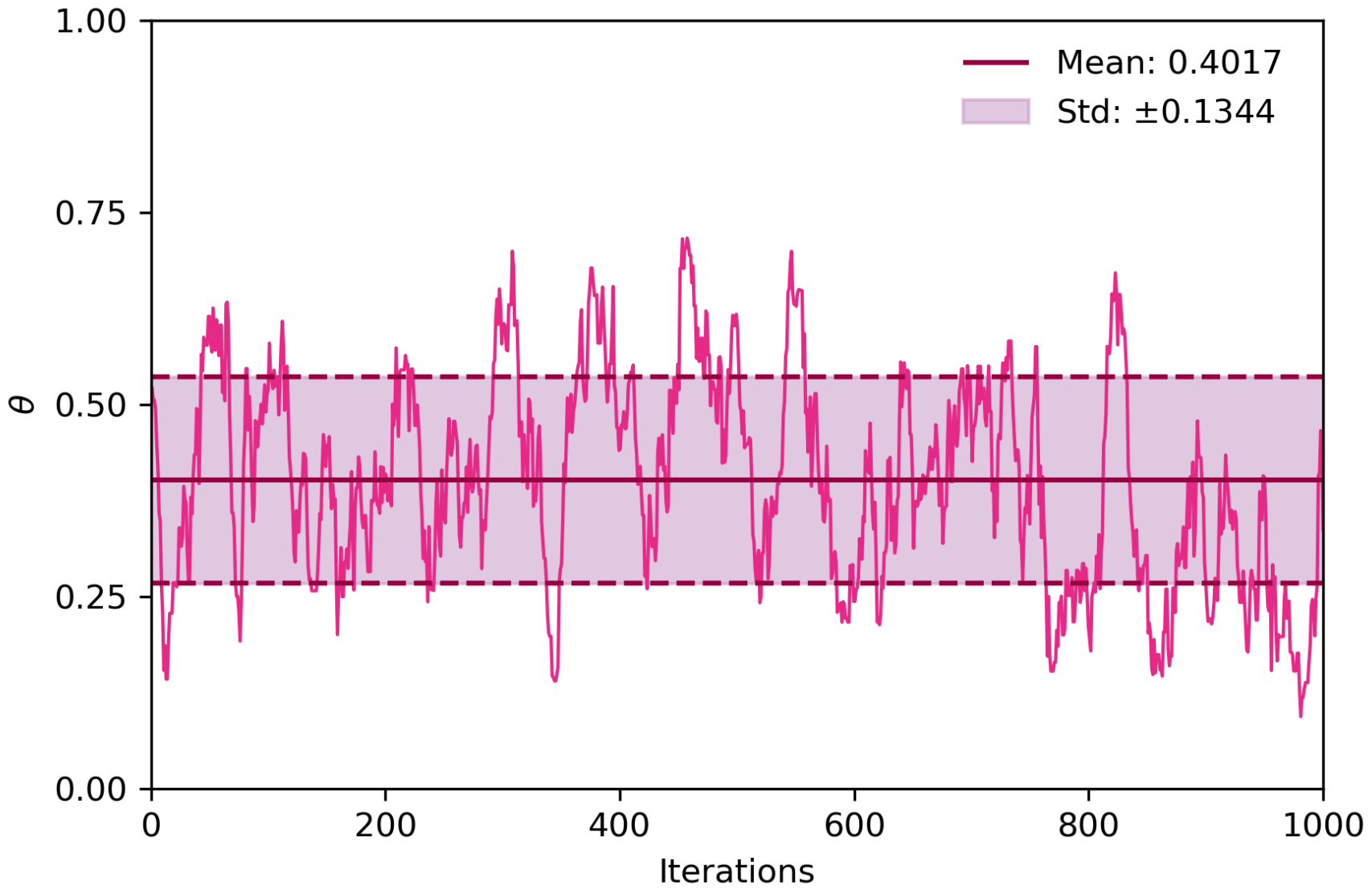
Use Markov Chain Monte Carlo (MCMC) with Metropolis-Hasting (MH) approach. Visualize the MCMC evolution and the density plot for θ based on 1000 iterations. Use np.random.seed(5) and the proposal distribution that follows a Gaussian distribution with a mean and standard deviation of 0.5 and 0.05, respectively. Please discuss what the results would be if we increased the iterations to 10000 with a burn-in period of 200. (6 points)

i) Compare the density plot of θ provided by MCMC-MH with the theoretical posterior distribution, and discuss the results.

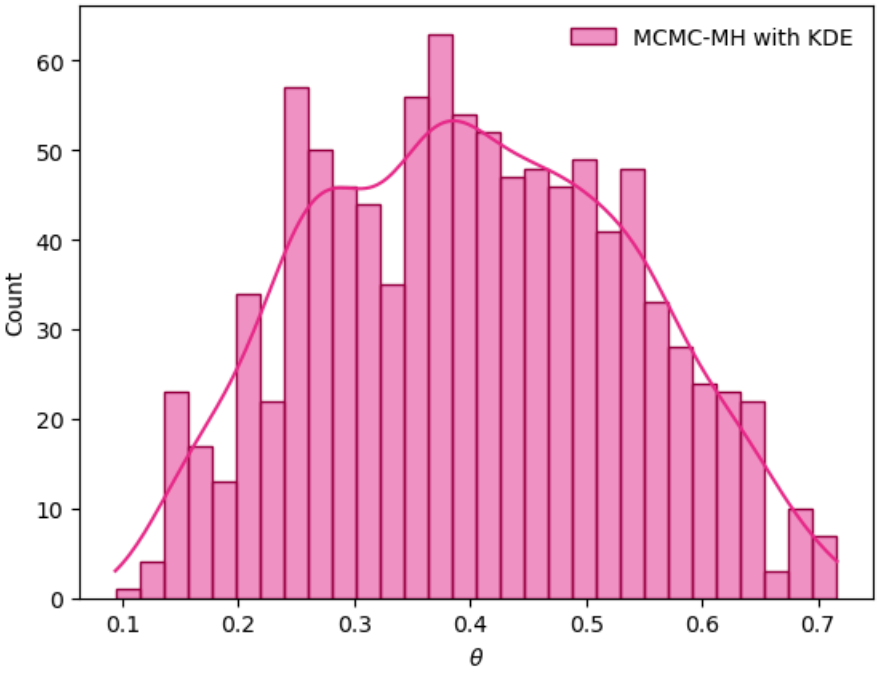
In [82]:
```
theta_vec = np.zeros(1000) # define empty theta array for mcmc-mh simulation
np.random.seed(5) # set random seed

for i in range(1000): # for each iteration...
    if i == 0:
        theta_vec[i] = 0.5 # set theta to 0.5 for iteration 0
    else:
        n, k = 10, 4 # 10 tosses, pick 4 heads
        theta_vec[i] = stats.norm.rvs(loc = theta_vec[i-1], scale = 0.05, size = 1)[0]
```

```
        # for all other iterations, draw a sample from a normal distribution with mean theta from the previous iteration and standard dev
        r_MH = stats.beta.pdf(theta_vec[i], k+1, n - k + 1)/stats.beta.pdf(theta_vec[i-1], k+1, n-k+1)
        #Posterior(New)/Posterior(Old) computes the metropolis-hastings ratio
        acceptance_alpha = np.min([1, r_MH])
        # acceptance probability is the minimum of 1 and the MH ratio
        u_MH = np.random.uniform(0, 1) # draw a random number between 0 and 1
        if u_MH < acceptance_alpha:
            # print('Accept')
            theta_vec[i] = theta_vec[i] # accept the new mean
        else:
            # print('Reject')
            theta_vec[i] = theta_vec[i-1] # reject the new mean and use the old mean

fig, ax = subplots(figsize=(6,4), dpi=300, tight_layout=True)

ax.plot(x, theta_vec, color = '#e7298a', linewidth = 1) # plot theta values over iterations

# visualize mean and standard deviation of theta samples
ax.hlines(np.mean(theta_vec), xmin = 0, xmax = 1000, color = '#980043', label = f'Mean: {np.mean(theta_vec):.4f} ')
ax.hlines(np.mean(theta_vec) + np.std(theta_vec), xmin = 0, xmax = 1000, linestyles= "--",color = '#980043')
ax.hlines(np.mean(theta_vec) - np.std(theta_vec), xmin = 0, xmax = 1000, linestyles= "--",color = '#980043')
ax.fill_between(x, np.mean(theta_vec) - np.std(theta_vec), np.mean(theta_vec) + np.std(theta_vec), color = '#c994c7', alpha = 0.5, label

# format plot
ax.set_xlabel('Iterations')
ax.set_xlim(0, 1000)
ax.set_ylabel('$\\theta$')
ax.set_ylim(0, 1)
ax.set_yticks([0, 0.25, 0.5, 0.75, 1])[0]
plt.legend(frameon = False, loc = 'best')
```

Out[82]: <matplotlib.legend.Legend at 0x7fe181dbb0e0>



In [ ]:
```
# create and format histogram with kernel density estimate
sns.histplot(theta_vec, kde=True, bins=30, color = '#e7298a', edgecolor = '#980043', label = 'MCMC-MH with KDE')
plt.xlabel('$\\theta$')
plt.ylabel('Count')
plt.legend(frameon = False, loc = 'best')
plt.show()
```



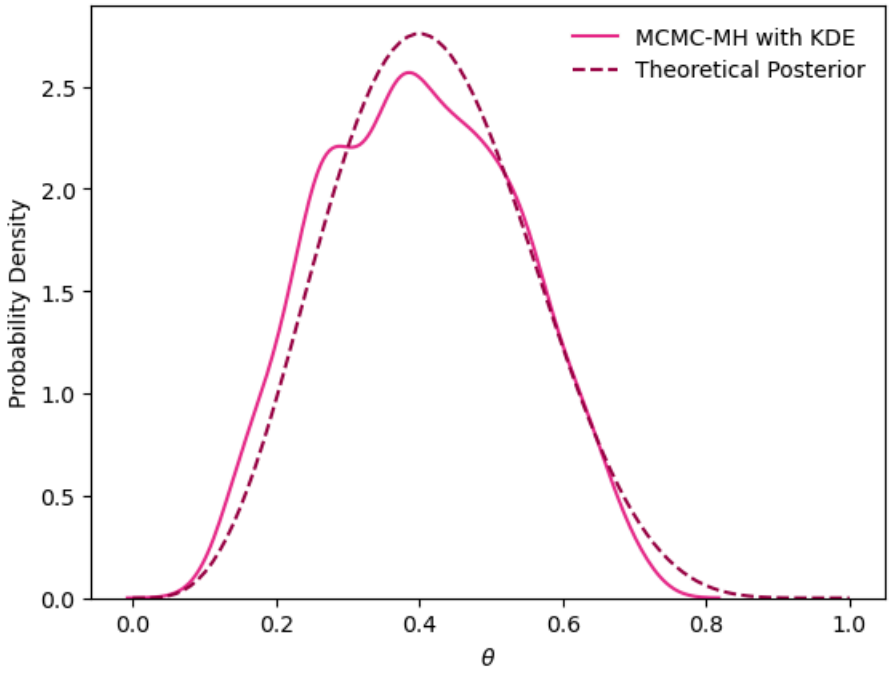
This coin flip problem uses a flat prior of $\mathcal{P}rior \sim \beta(a = 1, b = 1)$ and a binomial likleyhood of $\mathcal{L} \sim \text{Binom}(\theta, n = 10, k = 4)$. This selection of prior is a *conjugate prior* which allows the posterior to follow the same distribution as the prior with updated parameters.

The theoretical posterior may be computed as follows.

$$\mathcal{P}ost = \beta(\theta | a_{post} = n + a_{prior}, b_{post} = n - k + b_{prior})$$

In [ ]:
```
# plot kernel density estimate for MCMC-MH by itself
sns.kdeplot(theta_vec, color = '#e7298a', label = 'MCMC-MH with KDE')

# compute theoretical posterior
post_theo = stats.beta.pdf(np.linspace(0, 1, 100), 4+1, 10 - 4 + 1)

# plot theoretical posterior and format
plt.plot(np.linspace(0, 1, 100), post_theo, color = '#980043', linestyle = '--', label = 'Theoretical Posterior')
plt.legend(frameon = False, loc = 'best')
plt.xlabel('$\\theta$')
plt.ylabel('Probability Density')
```

Out[ ]:
```
Text(0, 0.5, 'Probability Density')
```



Discussion for Part 3c

As the number of iterations becomes large (i.e. as $N \to \infty$), the resultant posterior distribution is allowed to evolve in a more resonable fashion. The r_{MH} (i.e. Metropolis Hastings Ratio) prevents the issue of non-sensical evolution without preventing evolution altogether. Therefore, if $N = 10000$, the shape of the KDE is able to reflect an accuratly updated posterior distrubiton that has combined prior knowledge with experimental results (i.e. likleyhood values).

As can be seen in the above figure, the KDE density plot is able to resemble the theoretical posterior well. With more experiments, the posterior is eventually likely able to converge to the theoretical posterior.