# Solution 5: Bayesian theory

Set the seed for reproducibility

```
set.seed(1)
```

## Q.1: sequential updating

1. The posterior pdf or pmf $\pi(\theta|x_{1:n})$ of $\theta$ can be written as

$$\pi(\theta|x_{1:n}) \propto \rho(\theta) \prod_{i=1}^{n} \nu(x_i|\theta).$$

2. Replacing $n$ with $n+1$ above gives

$$\pi(\theta|x_{1:n+1}) \propto \rho(\theta) \prod_{i=1}^{n+1} \nu(x_i|\theta).$$

   On the other hand, using part 1 instead of $\rho$ as prior gives

$$\pi(\theta|x_{1:n+1}) \propto \left[\rho(\theta) \prod_{i=1}^{n} \nu(x_i|\theta)\right] \nu(x_{n+1}|\theta) = \rho(\theta) \prod_{i=1}^{n+1} \nu(x_i|\theta),$$

   so both approaches are equivalent. This is because if two probability density function are equal up to proportionality constant—i.e., $p_1 = cp_2$—then they are equal. Indeed,

$$\forall \theta \in \Theta: \; p_1(\theta) = cp_2(\theta) \implies \int_\Theta p_1(u)\mathrm{d}u = c \int_\Theta p_2(u)\mathrm{d}u \implies c = 1,$$

   because densities integrate to 1. It follows that $p_1 = p_2$.

## Q.2: Bayesian inference in the limit of increasing data

1. Setting the seed and loading the code provided

```r
set.seed(3)
posterior_distribution = function(rho, n_successes, n_observations) {
  K = length(rho) - 1
  gamma = rho * dbinom(n_successes, n_observations, (0:K)/K)
  normalizing_constant = sum(gamma)
  gamma/normalizing_constant
}
```

2. Function to compute the posterior mean

```r
posterior_mean = function(posterior_mass) {
  K = length(posterior_mass) - 1
  sum(posterior_mass*(0:K)/K)
}
```

3. Write a function to simulate the experiment

```r
simulate_posterior_mean_error = function(rho_true, rho_prior, n_observations){
  K = length(rho_true) - 1
  p_true = sample((0:K)/K, size = 1, prob = rho_true)
  simulated_heads = rbinom(1, n_observations, p_true)
  simulated_posterior = posterior_distribution(rho_prior, simulated_heads, n_observations)
  simulated_post_mean = posterior_mean(simulated_posterior)
  abs(simulated_post_mean-p_true)
}
```

4. Run the simulations

```r
K = 20
n_simulations = 1000
experiment_results = expand.grid(
  n_observations = 2^(0:6),
  replication = 1:n_simulations
  )
rho_true = rho_prior = 1:(K+1)
experiment_results$errors = sapply(
  experiment_results$n_observations, function(n_observations){
    simulate_posterior_mean_error(rho_true, rho_prior, n_observations)
  })
print(head(experiment_results))
```

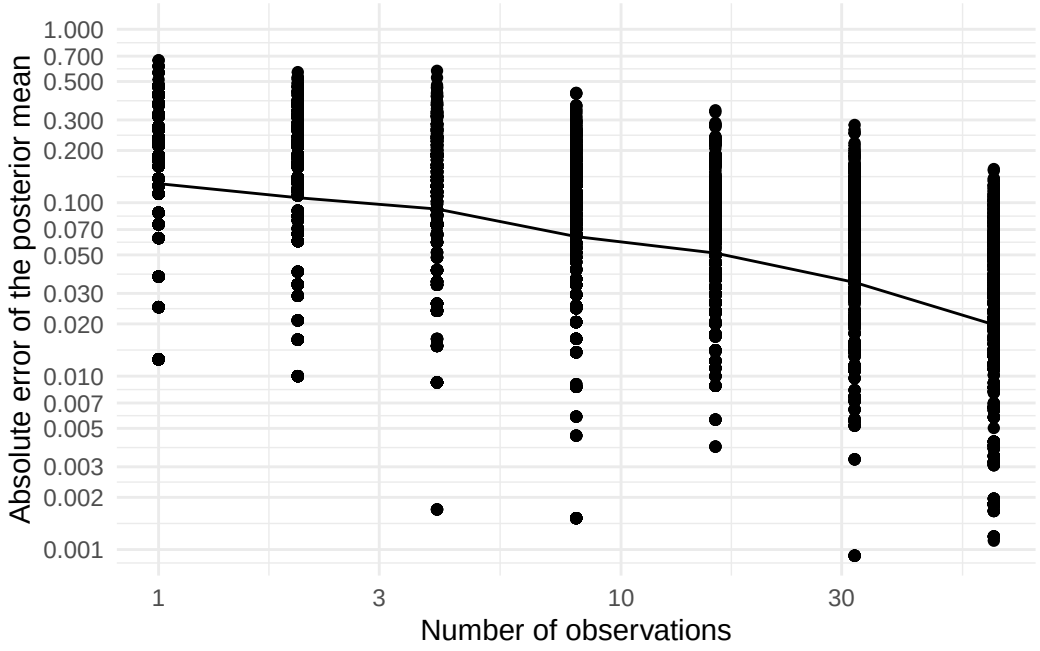
```
  n_observations replication     errors
1              1           1 0.18750000
2              2           1 0.01622951
3              4           1 0.08507303
4              8           1 0.02049715
5             16           1 0.01996920
6             32           1 0.01492196
```

```
print(tail(experiment_results))
```

```
     n_observations replication     errors
6995              2        1000 0.02095238
6996              4        1000 0.05919677
6997              8        1000 0.09100884
6998             16        1000 0.19000886
6999             32        1000 0.08627001
7000             64        1000 0.06184628
```

5. Plot the results

```
library(ggplot2)

ggplot(experiment_results, aes(x=n_observations, y=errors+1e-9)) + # avoid log(0)
  stat_summary(fun = mean, geom="line") + # Line averages over 1000 replicates
  scale_x_log10() +  # Show result in log-log scale
  scale_y_log10(n.breaks=16) +
  coord_cartesian(ylim = c(1e-3, 1)) +
  theme_minimal() +
  geom_point() +
  labs(x = "Number of observations",
       y = "Absolute error of the posterior mean")
```




6. Reading the values off the plot, we see that the slope is

```
(log10(0.025)-log10(0.05))/(log10(64)-log10(16))
```

```
[1] -0.5
```

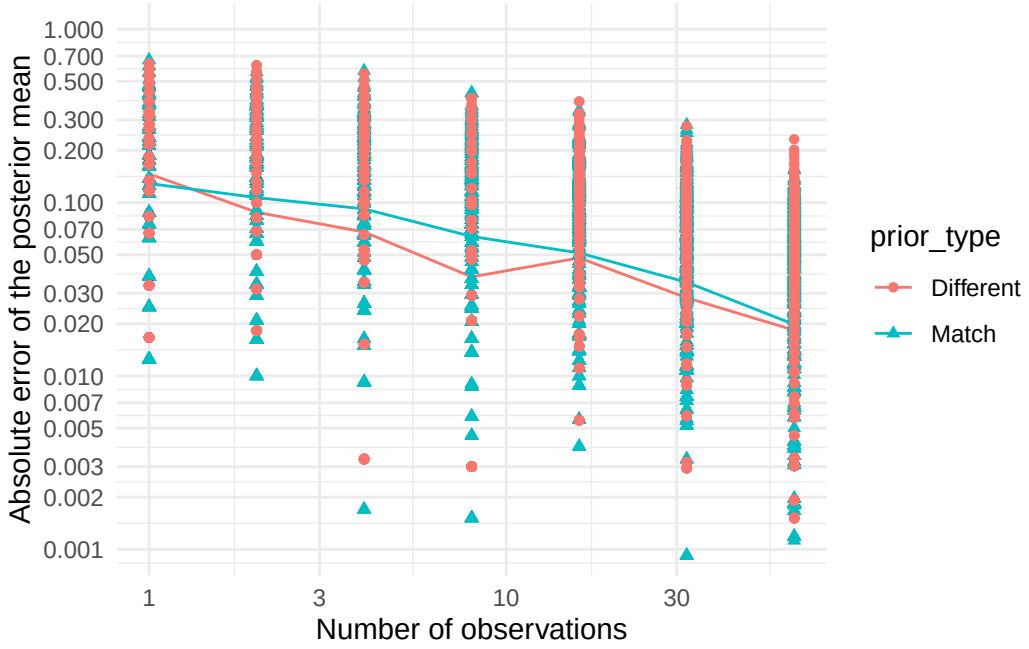
This means that the error decreases as $O(n^{-1/2})$.

7. Re-run simulations changing the prior, and merge the dataframes as directed

```
rho_prior = rep(1, K+1)
new_results = experiment_results
experiment_results$prior_type = "Match"
new_results$prior_type = "Different"
new_results$errors = sapply(
  new_results$n_observations, function(n_observations){
    simulate_posterior_mean_error(rho_true, rho_prior, n_observations)
  })
all_results = rbind(experiment_results, new_results)
```

Show the updated plot

```
ggplot(all_results, aes(x=n_observations, y=errors+1e-9, # avoid log(0)
                        color=prior_type, shape=prior_type)) +
  stat_summary(fun = mean, geom="line") + # Line averages over 1000 replicates
  scale_x_log10() +  # Show result in log-log scale
  scale_y_log10(n.breaks=16) +
  coord_cartesian(ylim = c(1e-3, 1)) +
  theme_minimal() +
  geom_point() +
  labs(x = "Number of observations",
       y = "Absolute error of the posterior mean")
```




We can see that having the wrong prior results in an initial discrepancy in the mean absolute error when the number of observations is low. But the effect quickly disappears as the number of observations increases, so the overall slope is unchanged. Somewhat surprisingly, with the incorrect prior our mean deviation is smaller than with the correct prior in this case (see comment below!!). The key takeaway, however, is that both model specifications result in a $O(1/\sqrt{n})$ decay in the error.

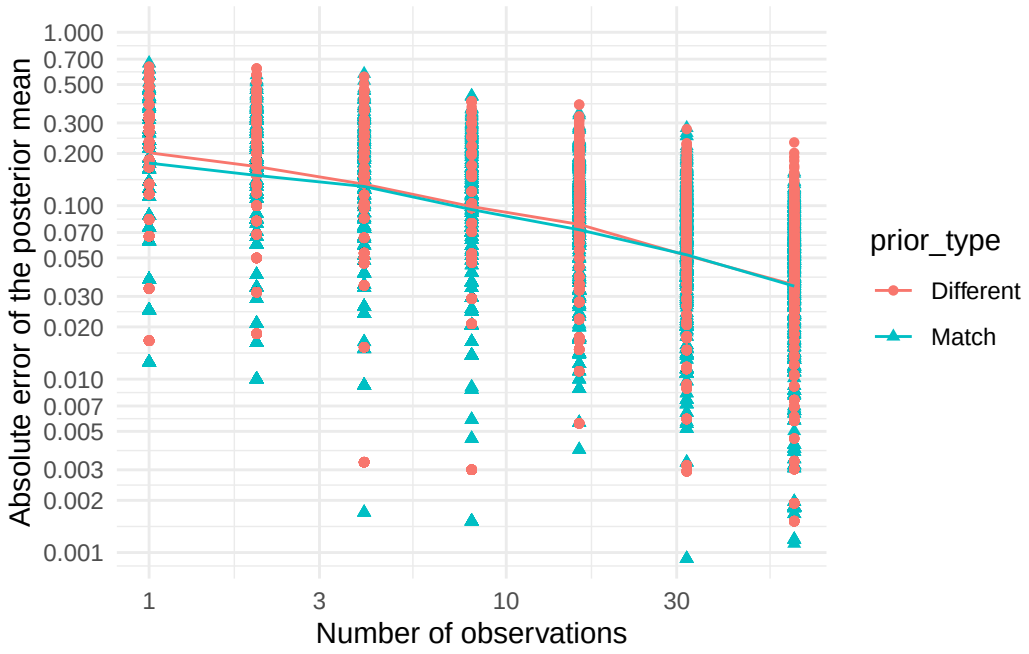
As it turns out, the `ggplot` R package has a bug. It seems to plot the mean trendlines incorrectly (possibly due to taking logs and averaging in the incorrect order). Here we manually compute the trendlines and find that the `match` prior setting performs better (as is expected).

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
# aggregate data by n_observations and prior_type
summary_results <- all_results %>%
  group_by(n_observations, prior_type) %>%
  summarise(avg_error = mean(errors), .groups = "drop")
```

```r
ggplot() +
  geom_point(data = all_results, aes(x=n_observations, y=errors+1e-9, # avoid log(0)
                        color=prior_type, shape=prior_type)) +
  geom_line(data = summary_results, aes(x = n_observations, y = avg_error, color = prior_type
  scale_x_log10() +  # Show result in log-log scale
  scale_y_log10(n.breaks=16) +
  coord_cartesian(ylim = c(1e-3, 1)) +
  theme_minimal() +
  geom_point() +
  labs(x = "Number of observations",
       y = "Absolute error of the posterior mean")
```




(The bug in the ggplot package seems to arise from accidentially interchanging expected values and taking logarithms. Note that by Jensen's inequality, we have

$$\mathbb{E}[\log(X)] \leq \log(\mathbb{E}[X]), \tag{1}$$

and so the two quantities are generally not the same.)