Bayesian Data Analysis

Bayesian Data Analysis Techniques

$\neq$

Applying Bayes' Theorem

Probability

- Frequentist interpretation:

  Probability measures a proportion of outcomes

- Bayesian interpretation:

  Probability measures a degree of belief

Usefulness

When asking

Does my code have bugs?

What is more useful:

- Yes — Frequentist
- Yes (80%), No (20%) — Bayesian

Goals

1. Estimate parameter values

2. Predict data values

3. Model comparison

Bayes' Theorem

Posterior

Likelihood

Prior

$$P(A|B) = \frac{P(B|A)}{P(B)} P(A)$$

Evidence

Bayes' Theorem

$$P(A|B) = \frac{P(B|A)}{\int p(B|A)p(A)dA} P(A)$$

Analytical Solution

$$P(A|B) = \frac{P(B|A)}{\int p(B|A)p(A)dA} P(A)$$

$P(A)$ conjugate prior of $P(B|A)$

Applying To Models

$$p(\text{data values}|\text{model structure and parameters})$$

Applying To Models

$$p(\text{data values}|\text{model structure and parameters})$$

For instance: coin flip

p(coin=Heads | θ) = θ

p(coin=Tails | θ) = 1 - θ

Applying To Models

$$p(\text{data values}|\text{model structure and parameters})$$

$$\Longrightarrow$$

$$p(\text{model structure and parameters}|\text{data values})$$

Probability to evaluate model

Applying To Models

$$p(\text{data values}|\text{model structure and parameters})$$

$$\Longrightarrow$$

$$p(\text{model structure and parameters}|\text{data values})$$

For instance: coin flip

p(θ | coin=Heads)

The Scientific Method

- Most successful way till now.
  - Observe system with different inputs.
  - Guess rules.
  - Predict outputs.
  - **Check with real system!**
  - If prediction matches outputs, you **may** have the right rules.
  - If prediction doesn't match outputs, you are definitely wrong.

Updating

$$P(A|B) = \frac{P(B|A)}{P(B)} P(A)$$

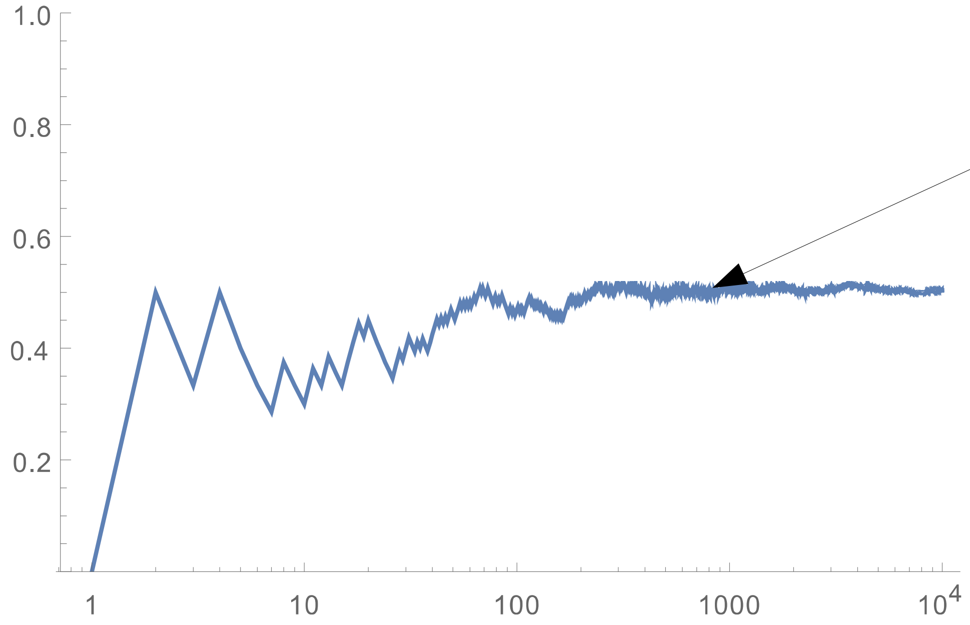
Updating

$$P(A|B',B) = \frac{P(B'|A)}{P(B')} \underbrace{\frac{P(B|A)}{P(B)} P(A)}_{\text{Old posterior}}$$

Most Probable Result

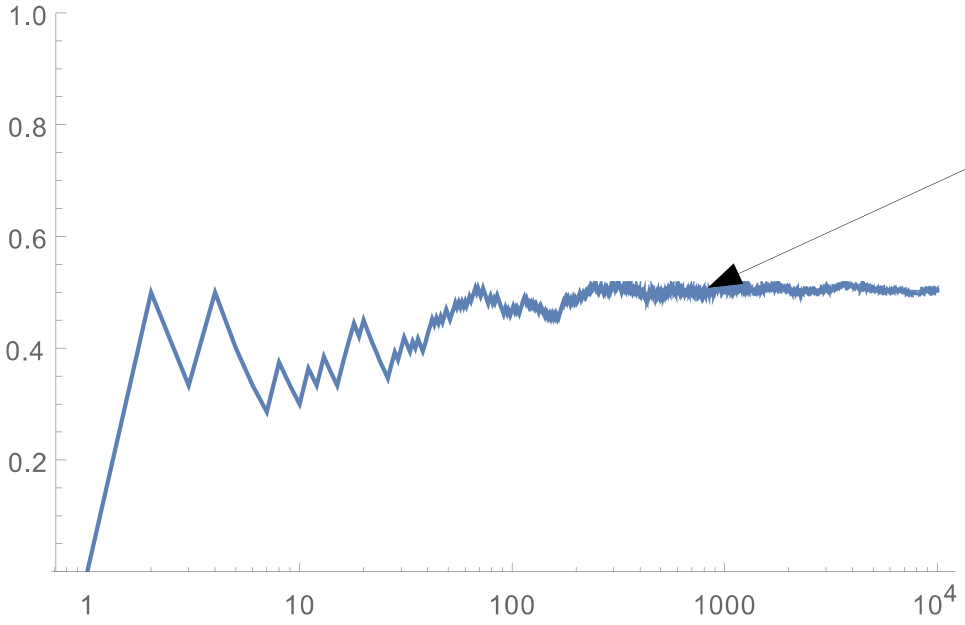
$$\hat{A} = \mathrm{argmax}_A \; p(A|B)$$

Coin Flips



Estimated number of "heads" over time

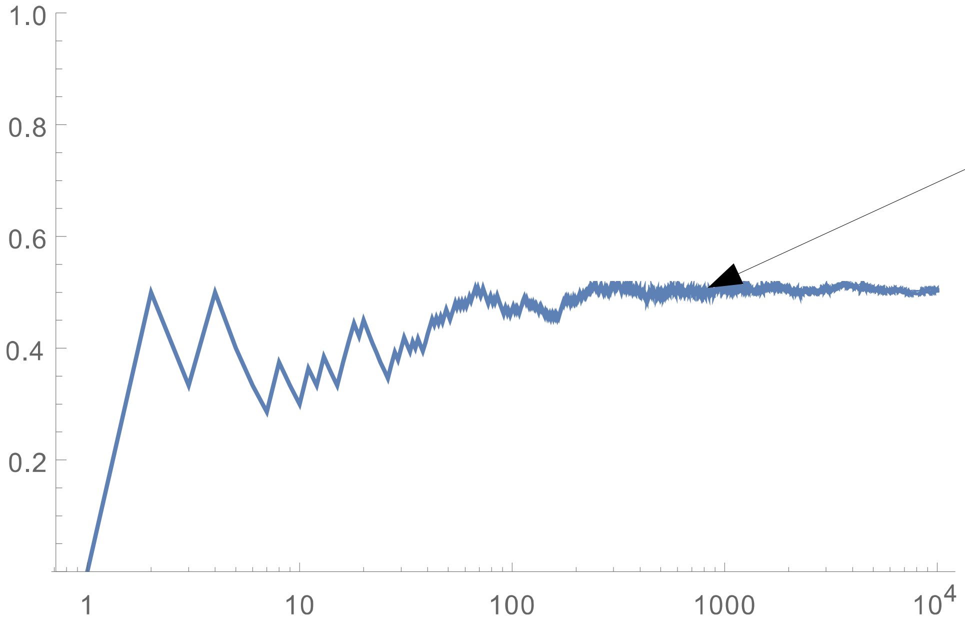
Coin Flips



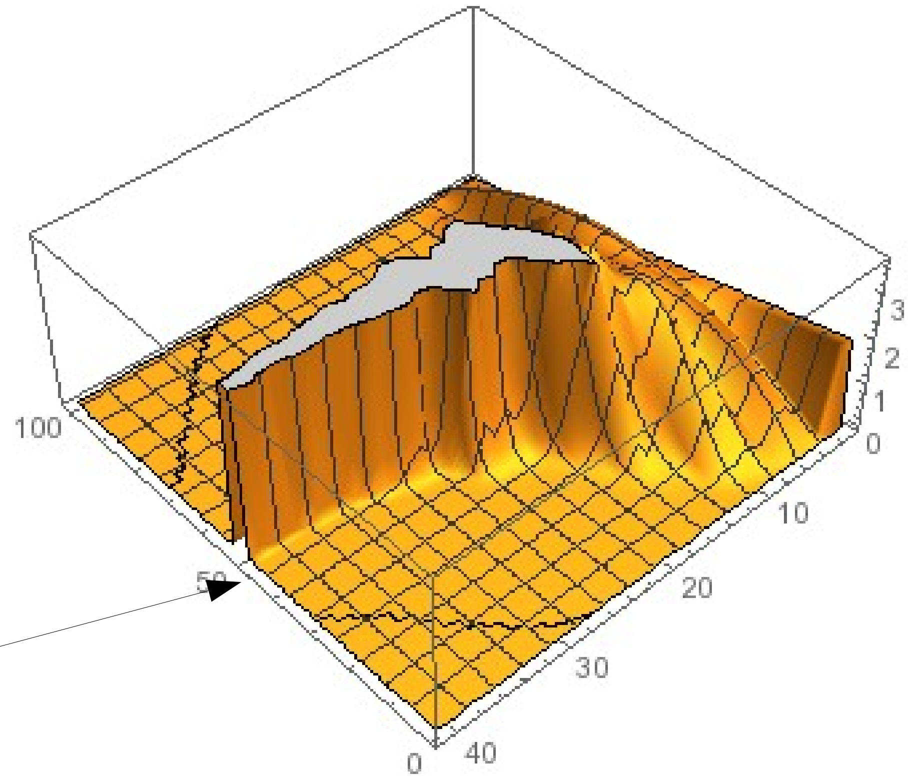
Estimated number of “heads” over time

Reliant on “Law of Large Number”

Coin Flips



Estimated number of “heads” over time



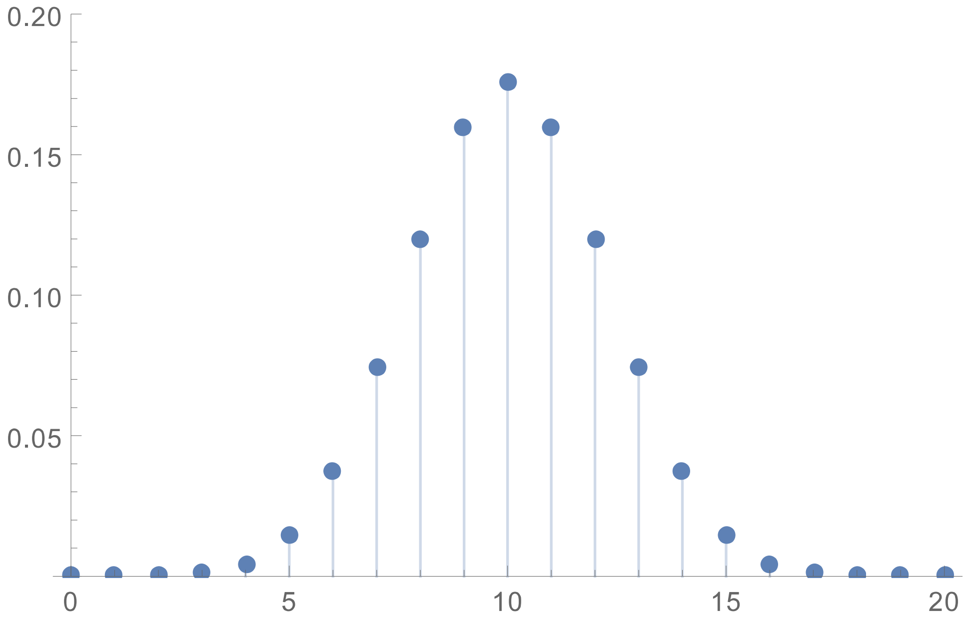
Bayes posterior probability over time

Example 1

- Write down information about gender of 20 random people
- Draw five of the notes, three indicate female
- How many females in original group?

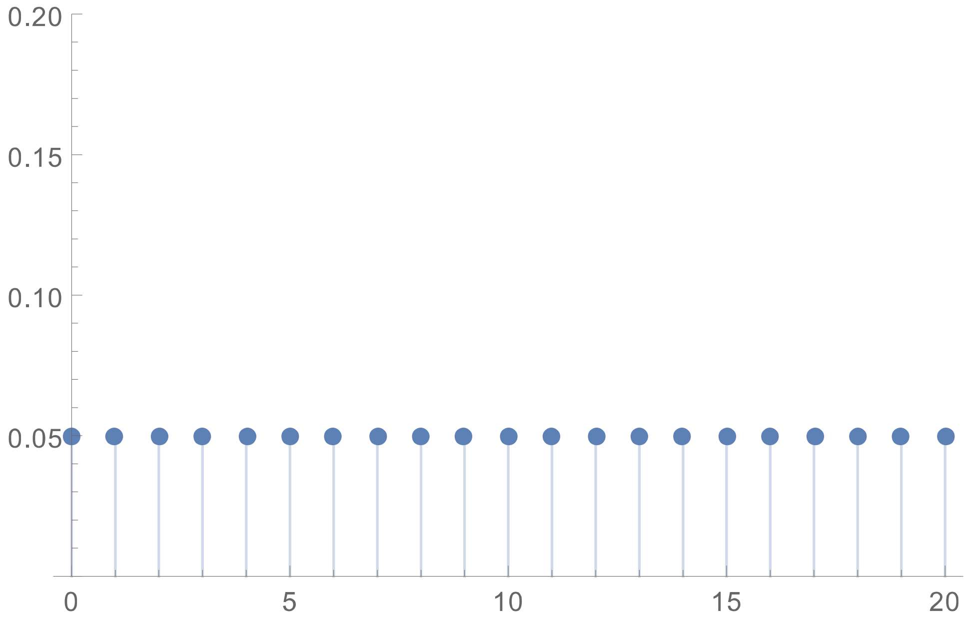
Example 1

- Without any knowledge: 50-50



Binomial Distribution B(20, 0.5)

- Maybe no knowledge



Uniform Distribution U(0, 20)

Prior

Example 1

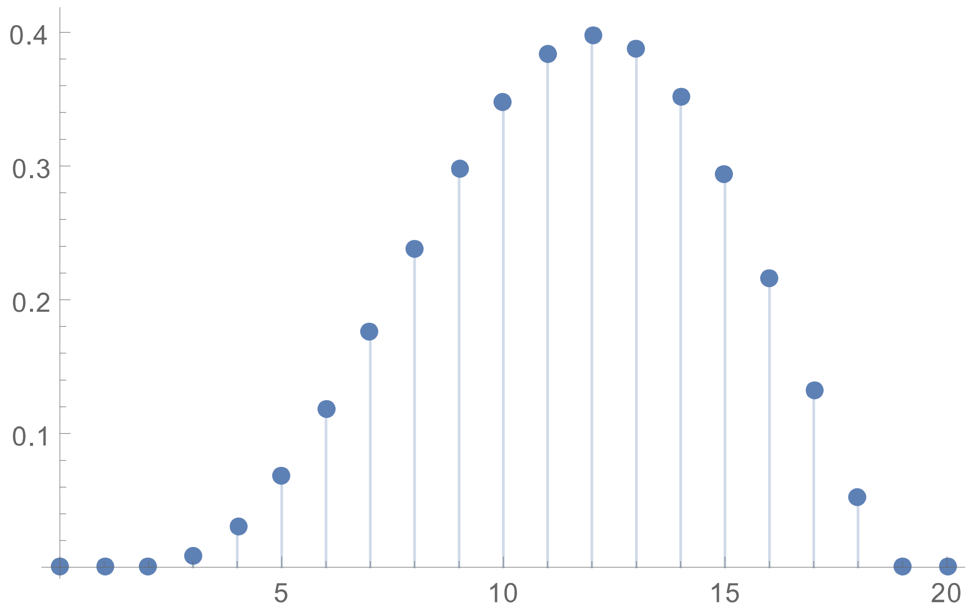
- Sampling: 5 out of 20 with positive result of 3
- Hypergeometric Distribution: H(N, K, n)
- Need Likelihood:

Likelihood of set of parameter values θ given outcome x is equal to probability of observed outcome.

$$\mathscr{L}(20,5,3|x) = \mathrm{PDF}(H(20,5,x), 3)$$

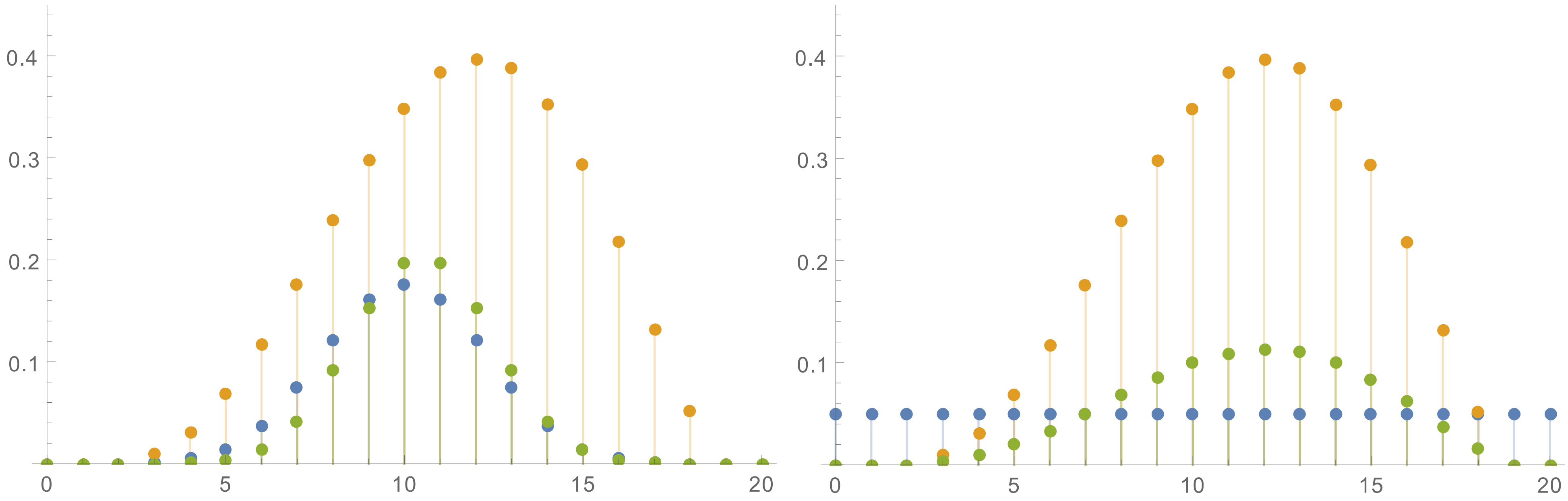
Example 1

- Likelihood



Example 1

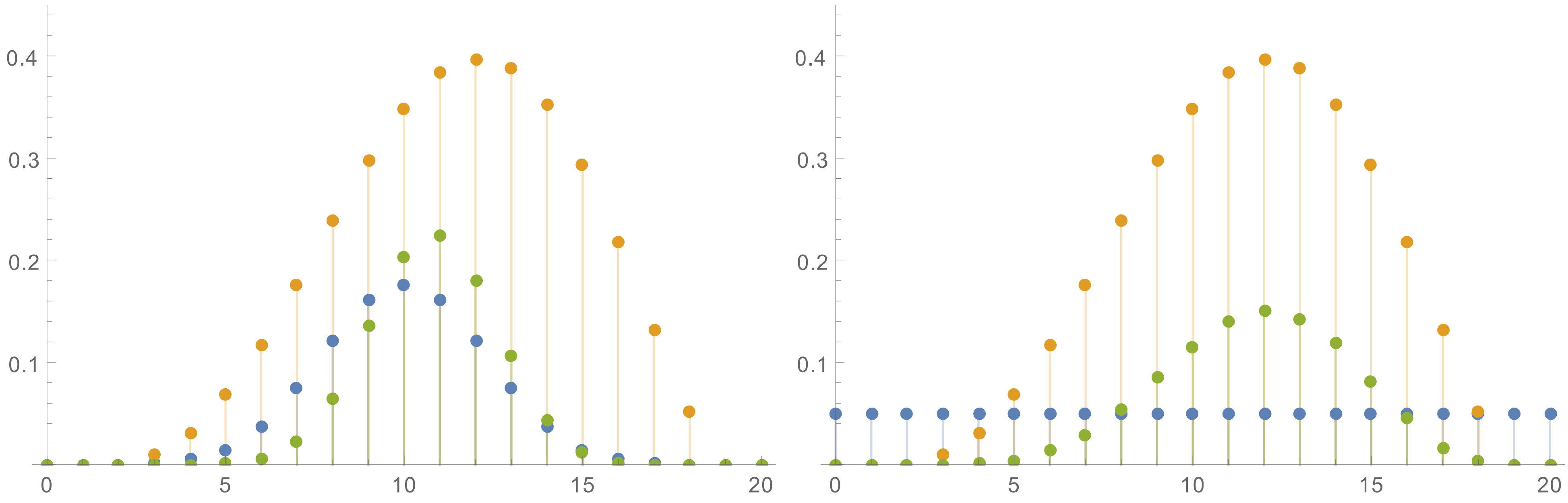
- Posterior: multiply prior with likelihood and normalize



- Original Prior
- Likelihood
- Posterior

Example 1

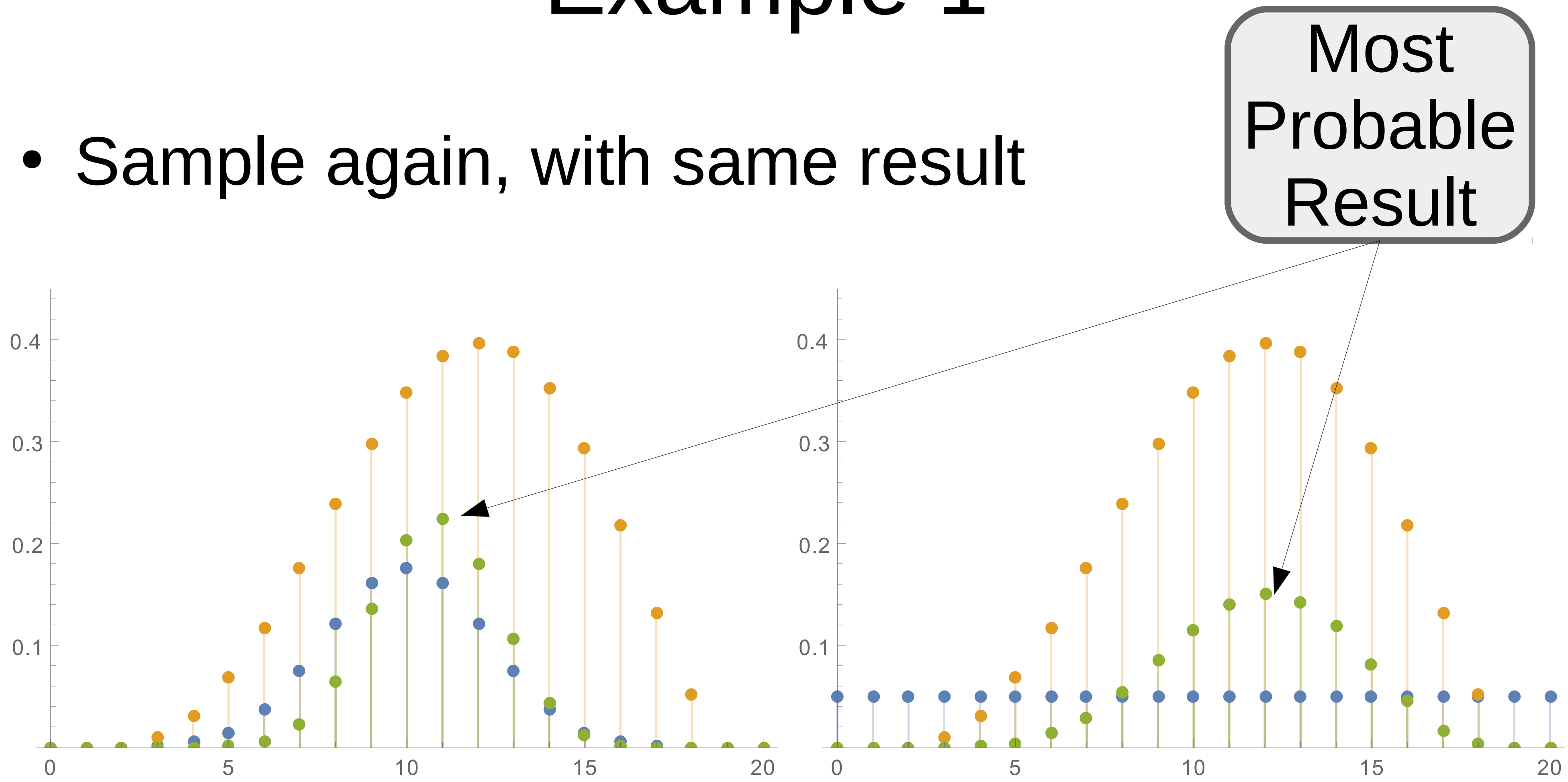
- Sample again, with same result



- Original Prior
- Likelihood
- Posterior

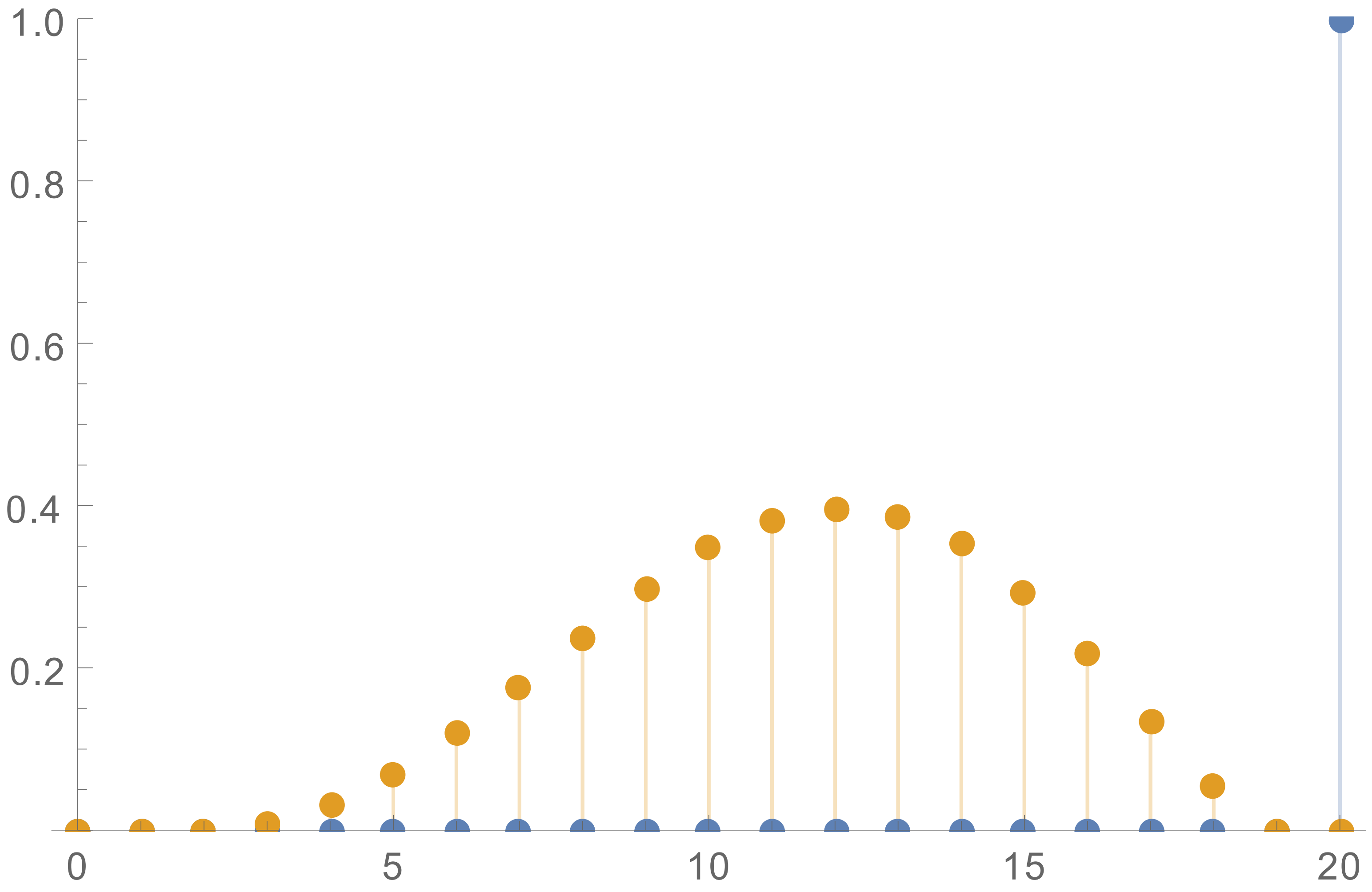
Example 1

- Sample again, with same result

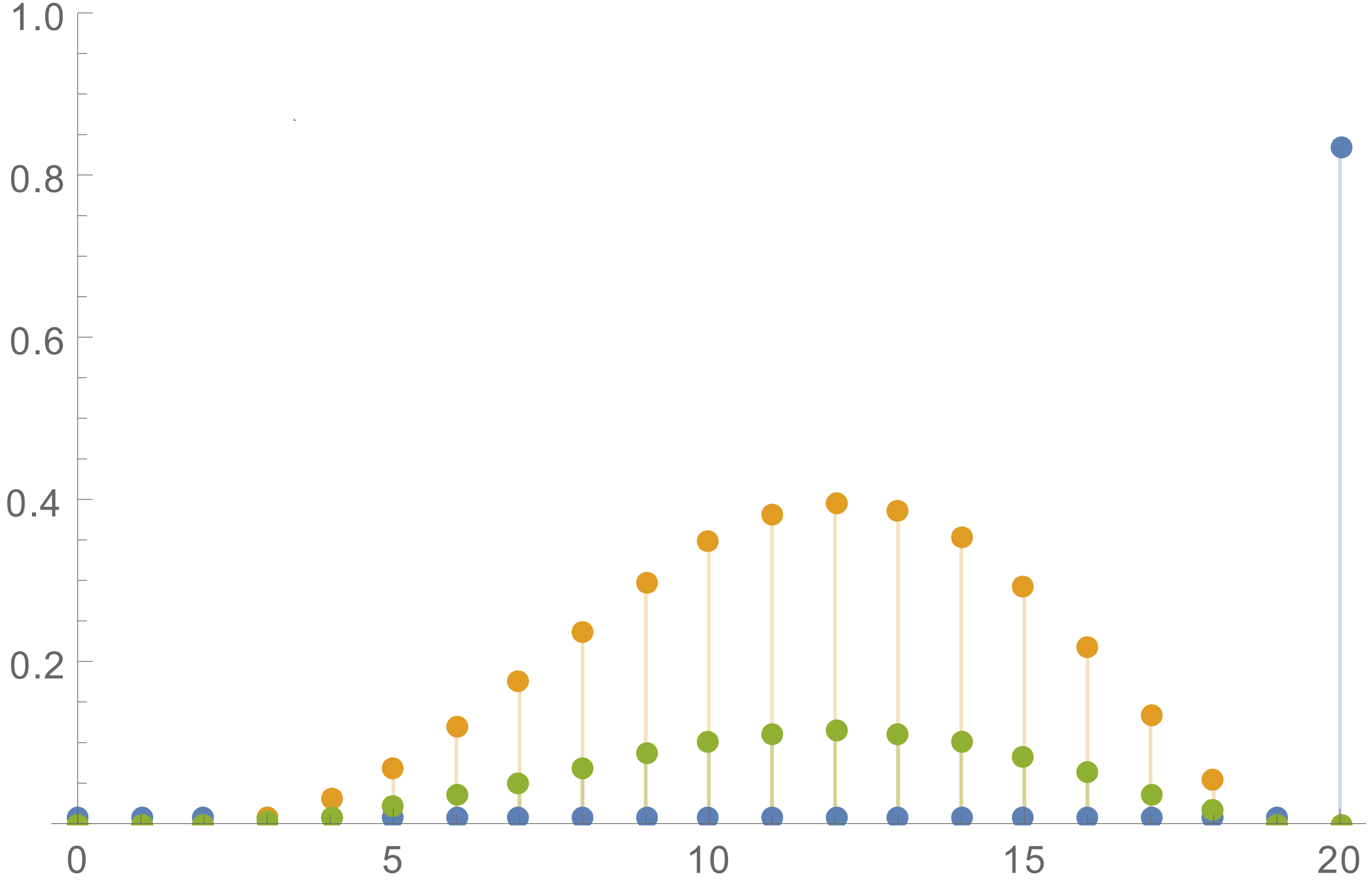


- Original Prior
- Likelihood
- Posterior

Do Not Be Too Sure

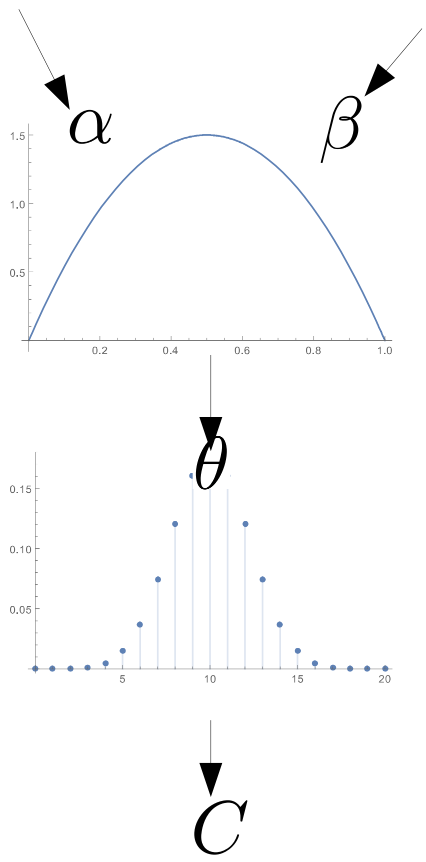


Use 0.01 or 0.99



Example 2: Coin Fairness

- Model for a coin:
  - Really Bernoulli with θ=0.5 ?
  - Have some throws
  - Update model
- Beta is prior for Binomial



Example 2: Coin Fairness

- Given sequence

  H, H, H, H, H, H, H, H, H, H, H, T, T, T

  what is the value of θ?

Example 2: Coin Fairness

- R code: use sampling program (JAGS)

```
modelString = "
model {
    # Likelihood:
    for ( i in 1:nFlips ) {
        y[i] ~ dbern( theta )
    }
    # Prior distribution:
    theta ~ dbeta( priorA , priorB )
}"
```

Example 2: Coin Fairness

- Initialization of parameters of JAGS program:

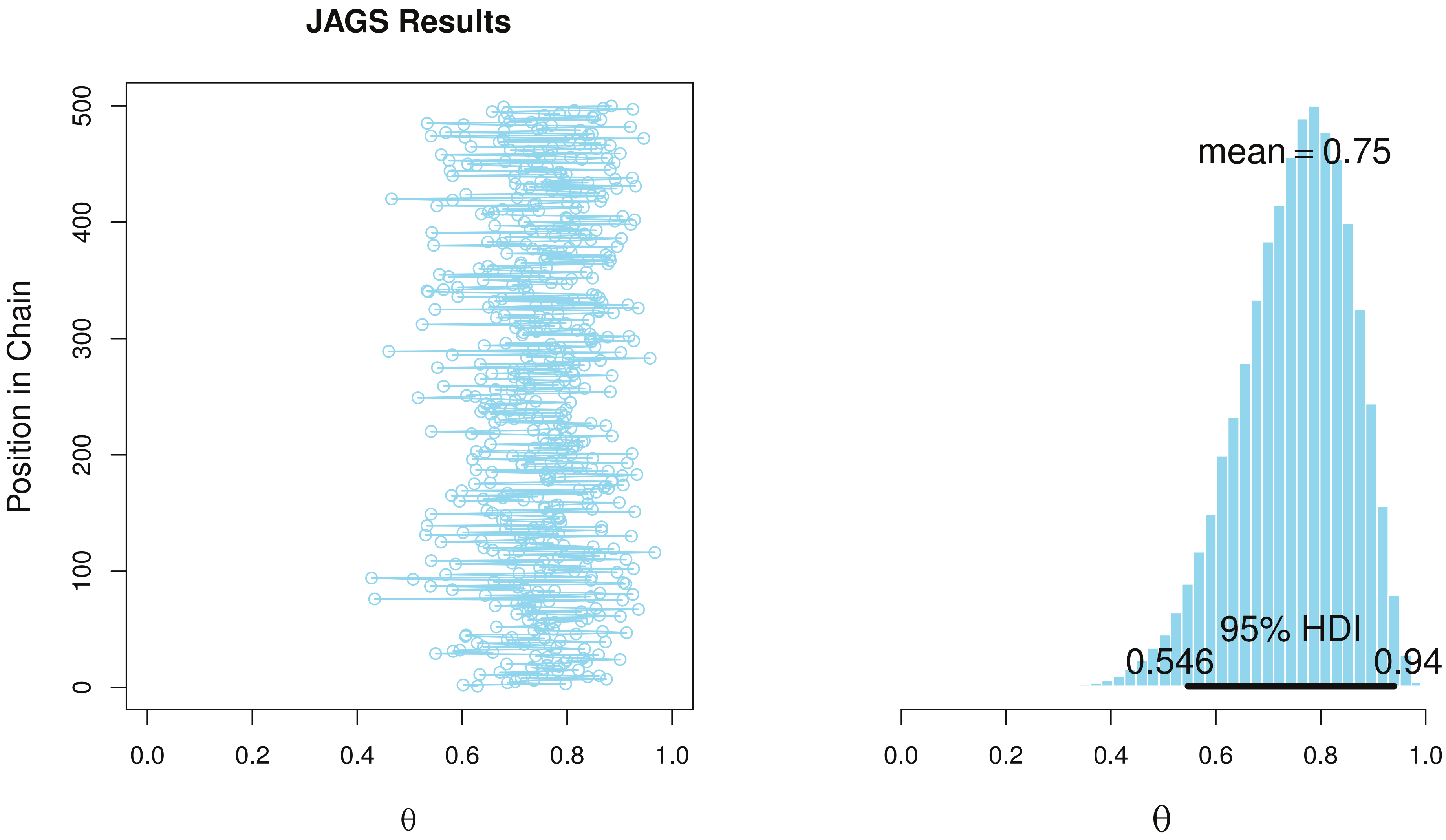
```
dataList = list(
    nFlips = 14 ,
    y = c( 1,1,1,1,1,1,1,1,1,1,1,0,0,0 ),
    priorA = 1,
    priorB = 1
)
```

Example 2: Coin Fairness

- Call JAGS:

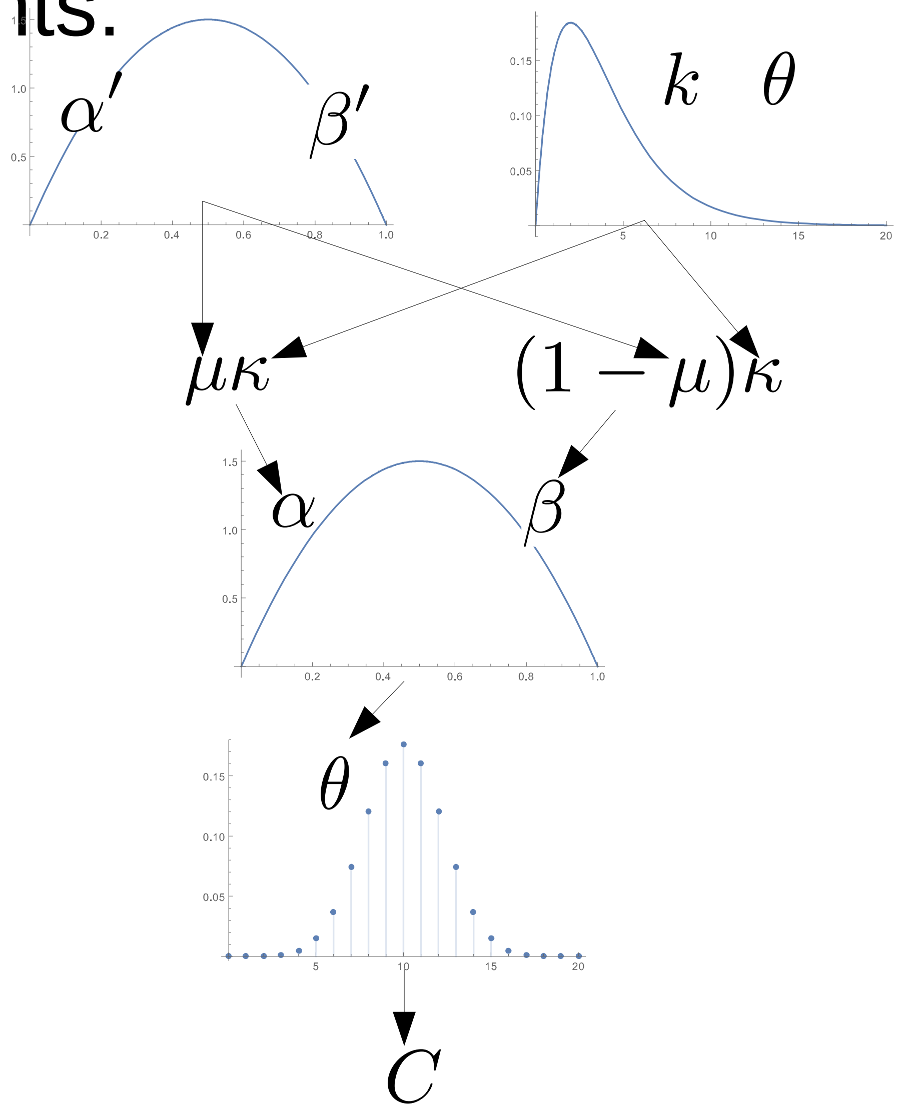
```
# Create, initialize, and adapt the model:
jagsModel = jags.model( "model.txt" , data=dataList ,
                        n.chains=nChains , n.adapt=adaptSteps )
# Burn-in:
cat( "Burning in the MCMC chain...\n" )
update( jagsModel , n.iter=burnInSteps )
# The saved MCMC chain:
cat( "Sampling final MCMC chain...\n" )
codaSamples = coda.samples(jagsModel , variable.names=parameters,
                           n.iter=nIter , thin=thinSteps )
```

Example 2: Coin Fairness



Even More Elaborate

- Model the mints:



Explicit Code Example

- Have data
- Have idea about model: a*x+b

C++ Code

```
typedef std::tuple<double,double,double> chainval_t;
{
  size_t iterations = 10000;
  chainval_t startvalue = std::make_tuple(0.1, 0.5, 0.3);
  auto chain = run_metropolis_MCMC(startvalue, iterations, x, y);

  size_t burnin = 5000;
  auto accum = std::accumulate(chain.begin() + burnin, chain.end(),
                               std::make_tuple(0.0, 0.0, 0.0),
                               [](const auto& l, const auto& r){
          return std::make_tuple(std::get<0>(l) + std::get<0>(r),
                                std::get<1>(l) + std::get<1>(r),
                                std::get<2>(l) + std::get<2>(r)); });
```

C++ Code

```
auto run_metropolis_MCMC(const chainval_t& startvalue, size_t iterations, const
std::vector<double>& x,
                         const std::vector<double>& y)
{
  std::vector<chainval_t> chain;
  chain.push_back(startvalue);

  __gnu_cxx::sfmt19937 rg((std::random_device())());
  std::uniform_real_distribution<double> udf;

  while (--iterations > 0) {
    auto proposal = proposalfunction(chain.back(), rg);

    auto probab = std::exp(logposterior(proposal, x, y) - logposterior(chain.back(), x, y));
    if (udf(rg) >= probab)
      proposal = chain.back();
    chain.push_back(proposal);
  }
```

C++ Code

```
double logposterior(const chainval_t& param,
                    const std::vector<double>& x,
                    const std::vector<double>& y)
{
  auto a = std::get<0>(param);
  auto b = std::get<1>(param);
  auto sd = std::get<2>(param);
  return loglikelihood(a, b, sd, x, y)
         + logprior(a, b, sd);
}
```

C++ Code

```
double loglikelihood(double a, double b, double sd,
                     const std::vector<double>& x, const std::vector<double>& y)
{
  std::vector<double> sl(x.size());
  std::transform(x.begin(), x.end(), y.begin(), sl.begin(),
                 [a,b,sd](auto x, auto y){return dnorm<double,true>(y, a*x+b, sd);});
  return std::accumulate(sl.begin(), sl.end(), 0.0);
}

double logprior(double a, double b, double sd)
{
  auto aprior = dunif<double,true>(a, 0.0, 10.0);
  auto bprior = dnorm<double,true>(b, 0.0, 5.0);
  auto sdprior = dunif<double,true>(sd, 0.0, 30.0);
  // addition because we have log probabilities
  return aprior + bprior + sdprior;
}
```

C++ Code

```
template<typename T, bool Log = false>
T dnorm(T x, T mu, T sd)
{
  constexpr T rt2pi = std::sqrt(2*__gnu_cxx::__math_constants<T>::__pi);
  auto a = x - mu;
  T res;
  if (Log)
    res = -a*a/(2*sd*sd)-std::log(sd)-std::log(rt2pi);
  else
    res = exp(-a*a/(2*sd*sd))/(sd * rt2pi);
  return res;
}

template<typename T, bool Log = false>
T dunif(T x, T mi, T ma)
{
  return (x < mi || x > ma
      ? (Log ? -std::numeric_limits<T>::infinity() : 0.0)
      : (Log ? std::log(1.0 / (ma - mi)) : 1.0 / (ma - mi)));
}
```

Questions?