5 Statistical Inference

Statistical inference is the process of using data analysis to infer properties of an underlying probability distribution.

- We saw that statistical distributions can be adjusted to data using parameters.
- How can we find the best value for a parameter given data and
- how certain can we be about our estimate?

> **i** Motivating a example
>
> - Imagine a study area where we placed 5 camera traps to observe wildlife for two weeks (= 14 days). For each day we check if there is at least one picture of roe deer.
> - We found for camera i k_i days with at least one picture (with $i = 1 \dots 5$). Our data set is as follows: $k_1 = 1, k_2 = 10, k_3 = 2, k_4 = 5, k_5 = 0$.
> - What would be a suitable model for this situation?

5.1 Maximum Likelihood Estimation (MLE)

Likelihood How probable is it to observe the data we observed, given a set of parameter(s). This translates to $L(\theta) = f(x; \theta)$, where $f(\cdot)$ is an appropriate probability distribution, $\mathbf{x}$ the data and the parameter(s) of interest.

We usually have more than one observation. We assume that observations are independent and identically distributed (iid). Thus the likelihood is defined as

$$L(\theta) = \prod_{i=1}^{n} f(x_i; \theta).$$

The maximum is equivalent (and often computationally beneficial) to

$$\log(L(\theta)) = \sum_{i=1}^{n} \log(f(x_i; \theta))$$

$\log(L(\theta|x))$ is often referred to as the log likelihood.

Our Example continued

If we decide to use the binomial model we use the the follwing PMF

$$P(k; p, N) = \binom{N}{k} p^k (1-p)^{N-k}$$

where k are the number of days where we observed a deer, p the **unknown** probability of observing a deer and N the number of trials (here $N = 14$).
Since we have 5 camera traps, we want to calculate the likelihood of the data given a proposed value for p.

$$L(p) = \prod_{i=1}^{5} \binom{N}{k_i} p^{k_i} (1-p)^{N-k_i}$$

Its often better to work with the log-likelihood

$$\begin{aligned} l(p) &= \log\left(\prod_{i=1}^{5} \binom{N}{k_i} p^{k_i} (1-p)^{N-k_i}\right) \\ &= \sum_{i=1}^{5} \log\left(\binom{N}{k_i} p^{k_i} (1-p)^{N-k_i}\right) \\ &= \sum_{i=1}^{5} \left(\log\binom{N}{k_i} + k_i \log(p) + (N-k_i)\log(1-p)\right) \end{aligned}$$

To find the optimal estimate for θ (here p), we have several options:

1. Naivly try different values.
2. Analytically find the the maximum of the log-likelihood.
3. Solve the log-likelihood numerically.

Exercise 1: MLE

Use the data from the example from before and try to find an estimate for p by just trying different values.

1. What is a suitable range of values for p?
2. Try 100 values within this range, which one would you pick?

5.2 Confidence intervals

Confidence interval A $(1-\alpha)100\%$ confidence intervals, covers in $(1-\alpha)100\%$ cases the true but unknown parameter.

$$P\left(-z_{(1-\frac{\alpha}{2})} \leq \frac{\bar{x}-\mu}{\sigma/\sqrt{n}} \leq z_{(1-\frac{\alpha}{2})}\right) = 1-\alpha$$

where $z_{(1-\frac{\alpha}{2})}$ is the $1-\frac{\alpha}{2}$-quantile of a standard normal.

Solving for μ, we get:

$$\left[\bar{x} - z_{(1-\frac{\alpha}{2})}\sigma_{\bar{x}}; \bar{x} + z_{(1-\frac{\alpha}{2})}\sigma_{\bar{x}}\right]$$

In most situations, we do not know $\sigma_{\bar{x}}$, but have to estimate it.

We can solve this by using a t-distribution with $n-1$ degrees of freedom instead.

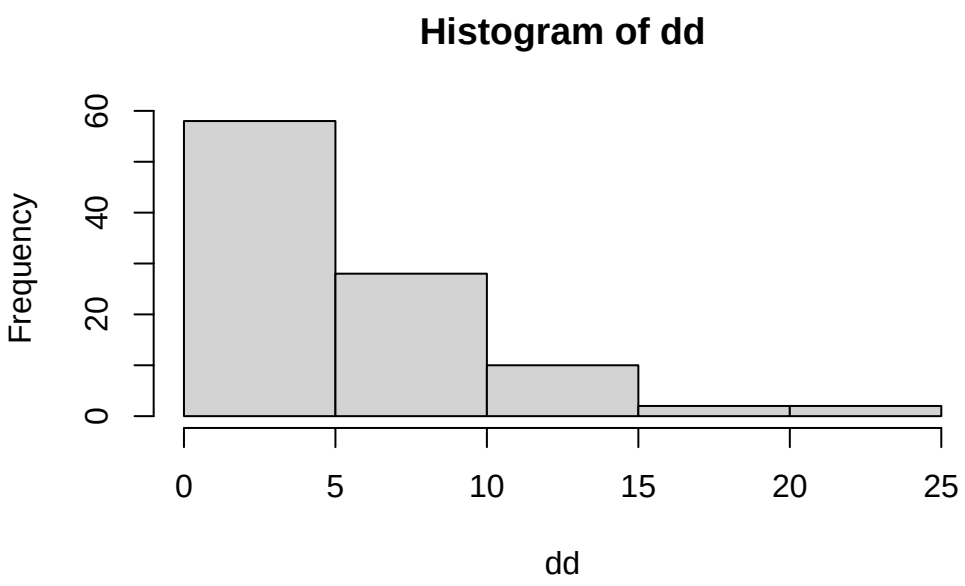
$$\left[\bar{x} - t_{(1-\frac{\alpha}{2};n-1)}SE_{\bar{x}}; \bar{x} + t_{(1-\frac{\alpha}{2};n-1)}SE_{\bar{x}}\right]$$

Standard error The standard error of a statistic is the standard deviation of its sampling distribution.

Sampling distribution The sampling distribution of a statistic is the distribution of that statistic, considered as a random variable, when derived of a random sample of size n.

> **i** Example: Sampling distribution
>
> We tracked a red fox for 100 days. The disribution of daily movement distances (in km) is exponential (we will just simulate the data here):

```r
set.seed(1) # set seed for reproducibility
dd <- rexp(100, rate = 0.2) # daily distances
hist(dd)
```



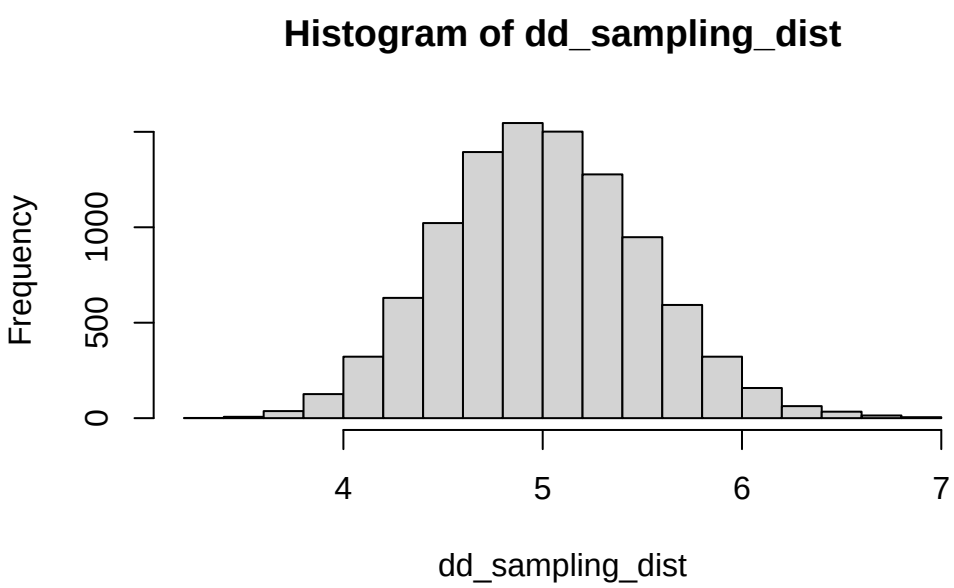
We can now calculate the mean distance moved:

```
mean(dd)
```

```
[1] 5.153382
```

Now lets calculate a sampling distribution for the mean. We can do this, because we have simulated the data. The idea is that we sample many times one fox, calculate the mean and then look at the distribution of the means. This is the sampling distribution.

```
dd_sampling_dist <- replicate(1e4, mean(rexp(100, rate = 0.2)))
hist(dd_sampling_dist)
```



This is almost a normal distribution and it converges to a normal distribution if the number of replicats increases. We can now calculate the standard deviation from the sampling distribution.

```
sd(dd_sampling_dist)
```

```
[1] 0.5017013
```

And compare the result to the SE we would obtain using the formula $SE = \frac{sd}{\sqrt{n}}$.

```
sd(dd) / sqrt(length(dd))
```

```
[1] 0.4682411
```

5.3 Bayesian Inference

- We always used a **Frequentist** approach to statistics so fare.
- **Bayesian** statistic is an alternative (and often more intuitive) way to look at parameter estimation.

Figure 5.1: Torok 2017

A good blog post about the conceptual differences: https://agostontorok.github.io/2017/03/26/bayes_vs_freque

5.3.1 The frequentist view

- We find parameter values that maximize the likelihood of the data and we assume a *true* but unknown parameter value.
- Uncertainty is captured using standard errors and confidence intervals.

5.3.2 The Bayesian view

- In Bayesian statistic we try estimate the full posterior distribution (i.e., the distribution of the parameter given the data).
- We are trying to find the posterior distribution ($p(\theta|data) = \frac{p(Y|\theta)p(\theta)}{p(Y)}$), with
  - the likelihood of the data ($p(Y|\theta)$), linking the data to the parameters.
  - the prior distribution of the parameter ($p(\theta)$), indicating our uncertainty of the parameter, before we see any data.
  - the marginal likelihood ($p(Y)$), which serves a normalizing constant.
- Uncertainty is captured in the posterior and so-called credibility intervals.
- Analytical solutions are almost never possible for the posterior.
- We need to simulate from the posterior mostly using Markov-Chains Monte Carlo (MCMC) methods.
- These methods are often slower than frequentist approaches and require a lot more computing power.
- Several programming languages/packages exist to fit Bayesian models (e.g., BUGS, JAGS, Nimble, Stan).
- The `brms` R-package builds on Stan and provides an easy to use interface.

5.4 MCMC - Metropolis Algorithm

- The idea is to sample from the posterior, even we do not know the functional form by using ($f(\theta|data) \propto f(\theta)L(\theta|data)$).
- This is done by proposing a new location from a suitable proposal distribution (i.e., a distribution that matches the parameters range).
- Decide whether or not go there or not with an acceptance probability α.

$$\alpha = \min\left(1, \frac{f(\theta')L(\theta'|data)}{f(\theta)L(\theta|data)}\right).$$

5.4.1 Implementing MCMC

We will have a look at a beta binomial MCMC alogrithm, were we estimate the π of a binomial distribution and use a beta distribution as a prior.

The first function `one_iter()` calls one iteration of the MCMC algorithm. The arguemnts are:

- `a` and `b` these are the parameters of the prior (a beta distribution)
- `current` the current position
- `y` are the observed data points
- `K` gives the number of trials of a binomial distribution.

```
one_iter <- function(a, b, current, y, K) {
  # a and b are parameters of the prior distribution
  # we use a beta prior here

  # Step 1: propose the next value
  proposal <- runif(1, 0, 1)

  # Step 2:
  prop_plaus <- dbeta(proposal, 2, 3) *
    prod(dbinom(y, K, prob = proposal)) # The likelihood

  prop_q <- dbeta(proposal, a, b)

  current_plaus <- dbeta(current, 2, 3) *
    prod(dbinom(y, K, prob = current)) # The likelihood
  current_q <- dbeta(current, a, b)

  alpha <- min(1, prop_plaus / current_plaus * current_q / prop_q)
```

```
  next_stop <- sample(c(proposal, current), size = 1, prob = c(alpha, 1 - alpha))
  data.frame(proposal, alpha, next_stop)
}
```

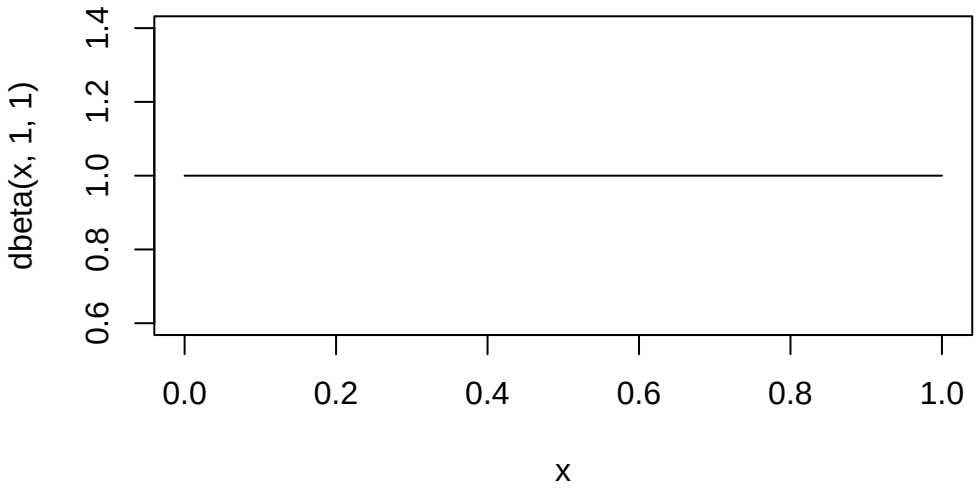
The main function is the `betabin()` function, where the function `one_iter()` is called `N` times.

```
# This is the mai
betabin <- function(N, a, b, y, K) {
  current <- 0.5
  pi <- rep(0, N)
  for (i in 1:N) {
    sim <- one_iter(a, b, current, y = y, K = K)
    pi[i] <- sim$next_stop
    current <- sim$next_stop
  }
  r <- data.frame(iter = 1:N, pi)
}
```

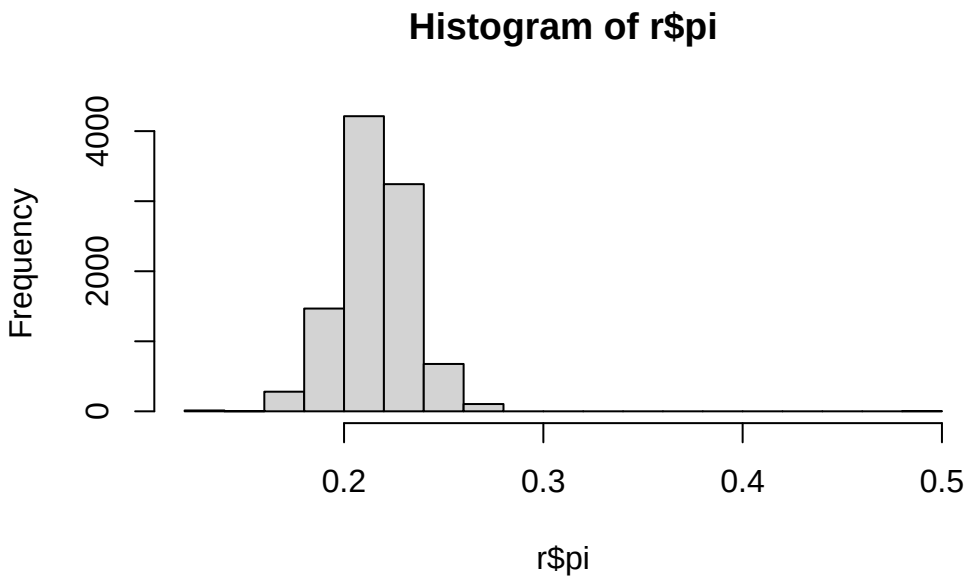
Using the function:

```
set.seed(123)
# observed data
y <- rbinom(50, 10, 0.2)

# flat prior
curve(dbeta(x, 1, 1), from = 0, to = 1)
```



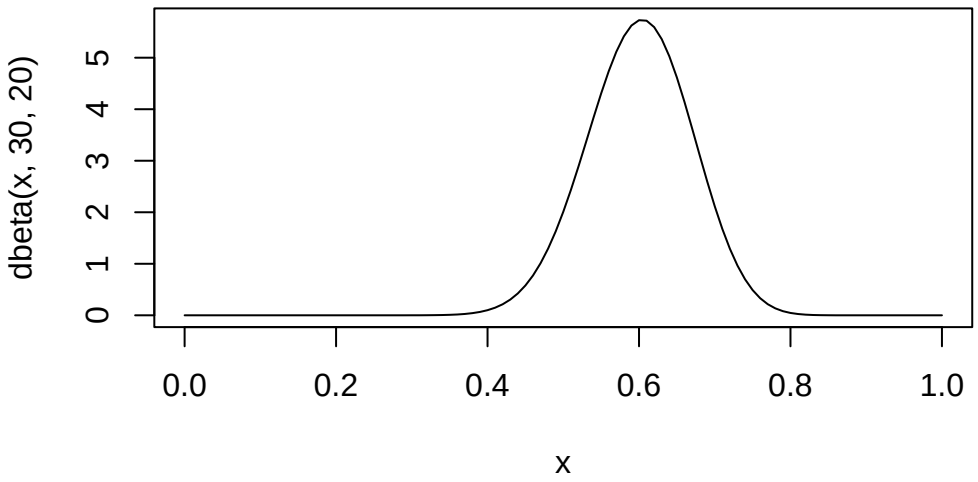
```
r <- betabin(10000, 1, 1, y, 10)
hist(r$pi)
```



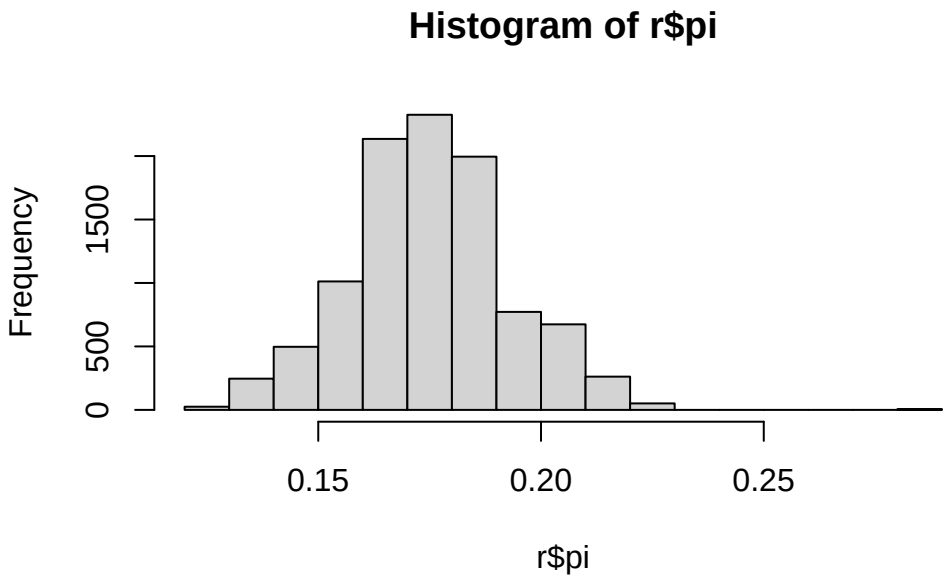

```
mean(r$pi)
```

```
[1] 0.2158838
```

```
# informative prior
curve(dbeta(x, 30, 20), from = 0, to = 1)
```



```
r <- betabin(10000, 30, 20, y, 10)
hist(r$pi)
```




```
mean(r$pi)
```

```
[1] 0.1751488
```

```
# MLE
sum(y)/(length(y) * 10)
```

```
[1] 0.214
```

Exercise 1: MLE

Use the data from the example from before and try to find an estimate for p by using the `betabin()` function. Try increasing the number of iterations an the information content of the prior. What is your estiamte p?

5.5 Stan (`rstan`)

We can use well established MCMC samplers, we will use Stan here.

```
library(rstan)
```

```
Loading required package: StanHeaders
```

```
rstan version 2.32.6 (Stan version 2.32.2)
```

```
For execution on a local, multicore CPU with excess RAM we recommend calling
options(mc.cores = parallel::detectCores()).
To avoid recompilation of unchanged Stan programs, we recommend calling
rstan_options(auto_write = TRUE)
For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
change `threads_per_chain` option:
rstan_options(threads_per_chain = 1)
```

```
Attaching package: 'rstan'
```

```
The following object is masked from 'package:tidyr':

    extract
```

First we need to define the Stan model:

```
scode <- "
  data {
    int<lower=1> N;
    int K;
    int y[N];
  }
  parameters {
    real pi;
  }
  model {
    pi ~ beta(30, 20);
    target += binomial_lpmf(y | K, pi);
  }
"
```

We can then fit model to data:

```
fit1 <- stan(
  model_code = scode,  # Stan program
  data = list(y = y, K = 10, N = length(y)),    # named list of data
  chains = 4,             # number of Markov chains
  warmup = 1000,          # number of warmup iterations per chain
  iter = 2000,            # total number of iterations per chain
  cores = 1,              # number of cores (could use one per chain)
  refresh = 1             # no progress shown
)

fit1
```