COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

GR5206: lecture 9

Computational Statistics
And Introduction to Data Science

Thibault Vatter

Department of Statistics, Columbia University

11/08/2019

Today

- Distributions as models
- Method of Moments
- Maximum Likelihood Estimation
- Optimization

Outline

Outline

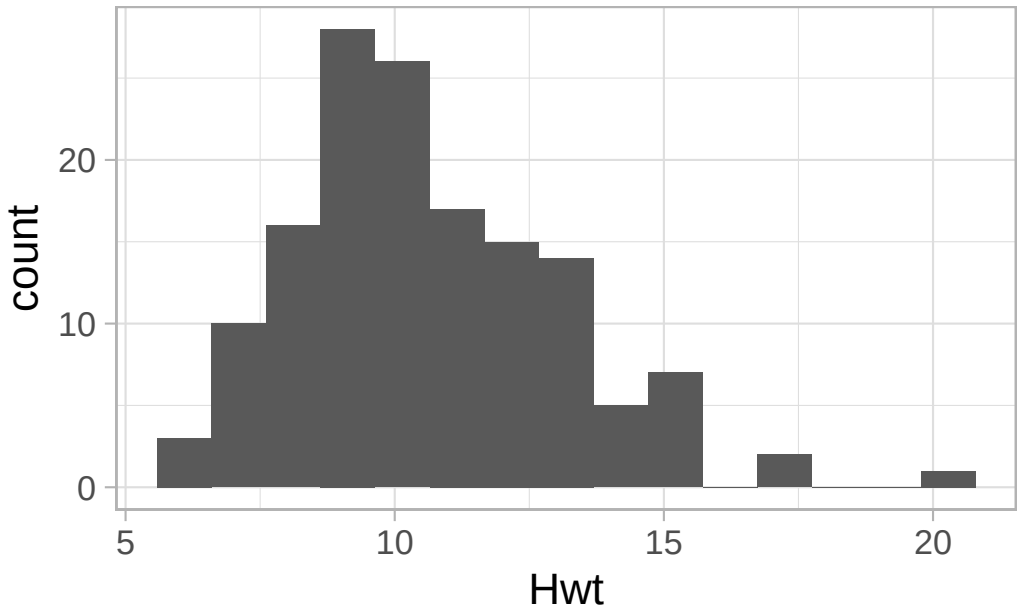
The cats dataset

- Samples of heart and body weights of male/female cats.
- All the cats are adults and over 2 kg in body weight.

```
library(MASS)
(cats <- as_tibble(cats))
#> # A tibble: 144 x 3
#>    Sex     Bwt   Hwt
#>    <fct> <dbl> <dbl>
#>  1 F       2     7
#>  2 F       2     7.4
#>  3 F       2     9.5
#>  4 F       2.1   7.2
#>  5 F       2.1   7.3
#>  6 F       2.1   7.6
#>  7 F       2.1   8.1
#>  8 F       2.1   8.2
#>  9 F       2.1   8.3
#> 10 F       2.1   8.5
#> # ... with 134 more rows
```

The distribution of the data


```
cats %>%
  ggplot(aes(x = Hwt)) +
  geom_histogram(bins = 15)
```




```
hwt <- cats %>%
  pull(Hwt)
summary(hwt)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>    6.30    8.95   10.10   10.63   12.12   20.50
```

Quantiles

- Cut points that divide the range of a probability distribution:
  - Into continuous intervals.
  - With equal probabilities.
- Sample quantiles are defined as weighted averages of consecutive order statistics.

$$Q(p) = (1 - \gamma)X_{(j)} + \gamma X_{(j+1)},$$

If $j/n \leq p < (j + 1)/n$, where:

- $X_{(j)}$ is the jth order statistic,
- $j = \lfloor np \rfloor$, $\gamma \in [0, 1]$.

```
quantile(hwt)
#>    0%   25%   50%   75%  100%
#>  6.30  8.95 10.10 12.12 20.50
```

Empirical distribution

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Let
  - X_i, $i = 1, \ldots, n$ be *iid* random variables with distribution F.
- Then:
  - The empirical distribution is

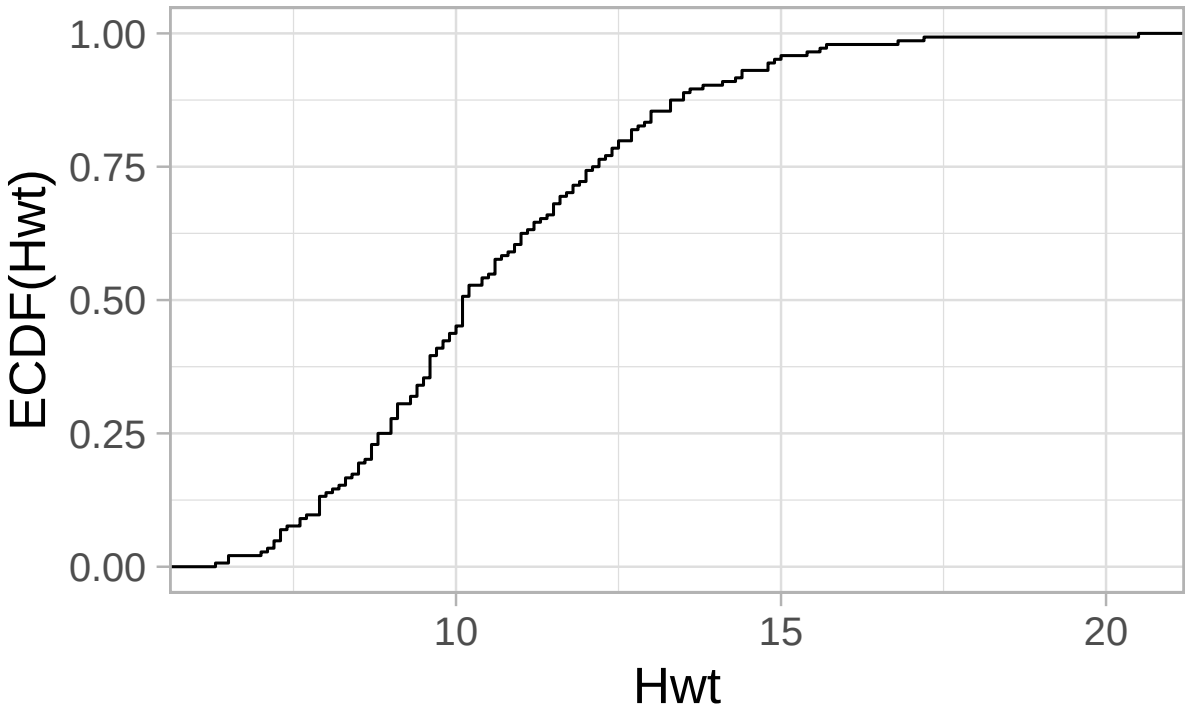
$$\widehat{F}_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(X_i \leq x),$$

  - Glivenko-Cantelli theorem: $\sup_{x \in \mathbb{R}} \left| \widehat{F}_n(x) - F(x) \right| \overset{a.s.}{\to} 0$.
  - No assumptions about the distribution beyond the observations.
- In R:
  - `ecdf()` for Emprical Cumulative Distribution Function.
  - `quantile()` and `ecdf()` are (almost) inverses to each other.

```
(hwt_ecdf <- ecdf(hwt))
#> Empirical CDF
#> Call: ecdf(hwt)
#>  x[1:73] =  6,  6,  7,  ..., 2e+01, 2e+01
hwt_ecdf(quantile(hwt))
#> [1] 0.00694 0.25000 0.50694 0.75000 1.00000
```

Empirical distribution cont'd


```
cats %>%
  ggplot(aes(x = Hwt)) +
  stat_ecdf() +
  ylab("ECDF(Hwt)")
```



Kernel density estimation

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Let
  - X_i, $i = 1, \ldots, n$ be *iid* random variables with density f.
  - K be a symmetric probability density function on $\mathbb{R}$.
- Then, for $b \searrow 0$,

$$\mathrm{E}\left[\frac{1}{b}K\left(\frac{X_i - x}{b}\right)\right] \to f(x), \quad \text{for all } x \in \mathbb{R}.$$

- Kernel density estimator (Rosenblatt, 1956; Parzen, 1962):

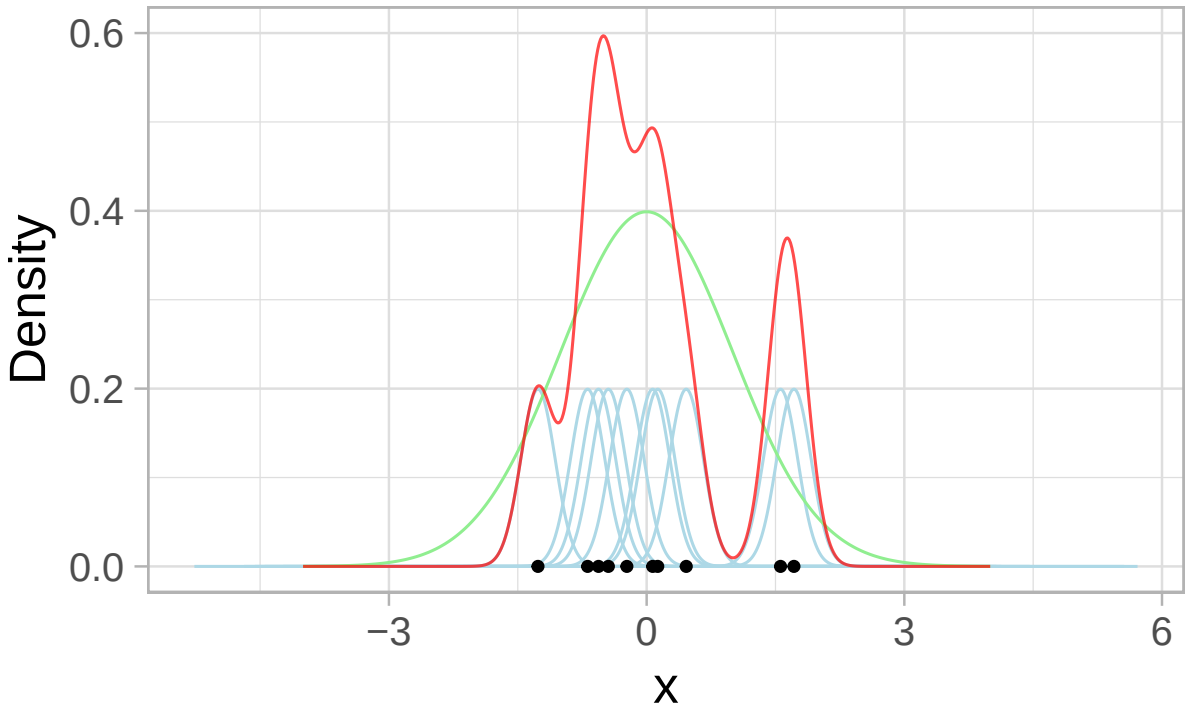
$$\widehat{f}(x) = \frac{1}{nb}\sum_{i=1}^{n} K\left(\frac{X_i - x}{b}\right) \xrightarrow{p} f(x), \quad \text{for all } x \in \mathbb{R}.$$

- K is called the *kernel*, b is called the *bandwidth*.

Kernel density estimation cont'd

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

$$f = \mathcal{N}(0,1), \quad n = 10, \quad b = 0.2$$

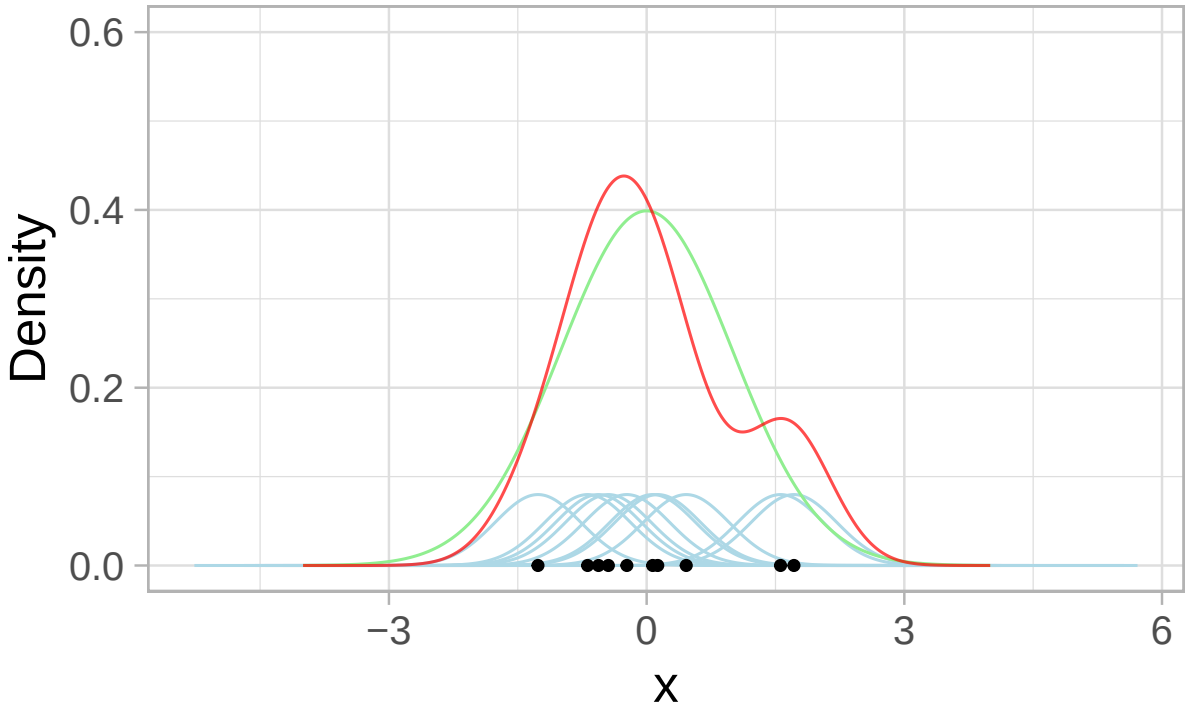


- Sample in black
- $\frac{1}{nb} K\left(\frac{X_i - x}{b}\right)$ in blue
- True distribution in green
- Kernel density in red

Kernel density estimation cont'd

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

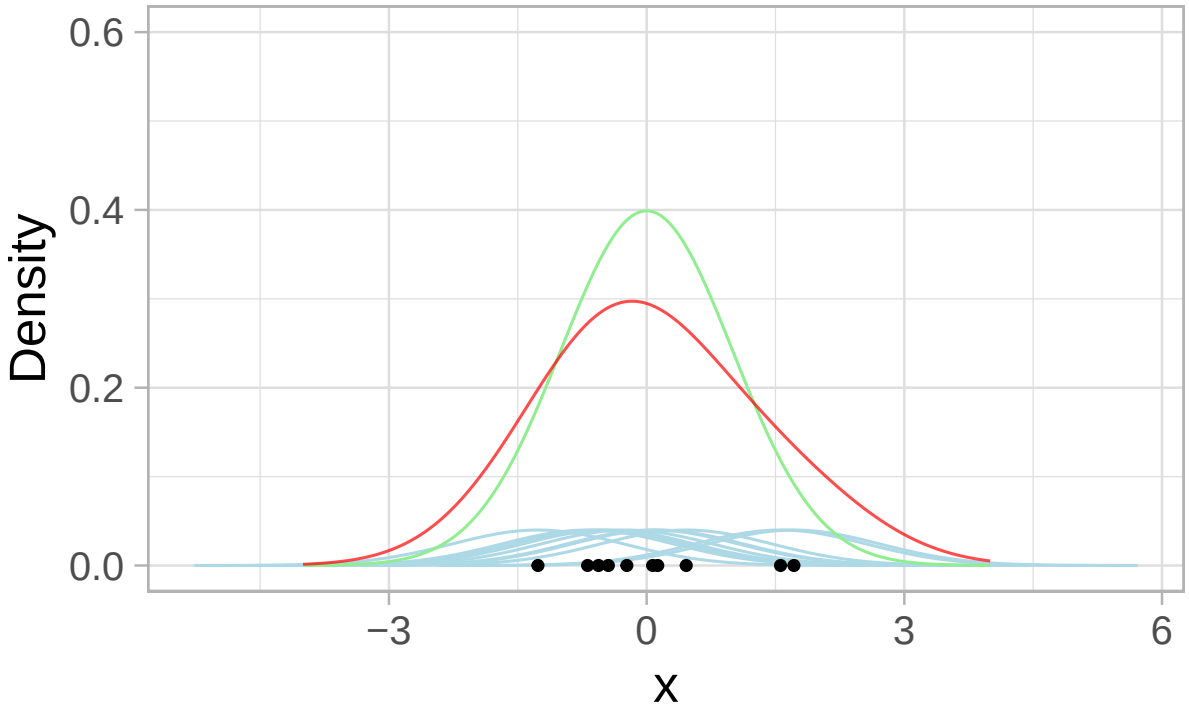
$$f = \mathcal{N}(0,1), \quad n = 10, \quad b = 0.5$$



- Sample in black
- $\frac{1}{nb} K\left(\frac{X_i - x}{b}\right)$ in blue
- True distribution in green
- Kernel density in red

Kernel density estimation cont'd

$$f = \mathcal{N}(0,1), \quad n = 10, \quad b = 1$$



- Sample in black
- $\frac{1}{nb} K\left(\frac{X_i - x}{b}\right)$ in blue
- True distribution in green
- Kernel density in red

Kernel density estimation cont'd

- How to select the bandwidth?
- Idea: Minimize the mean integrated squared error

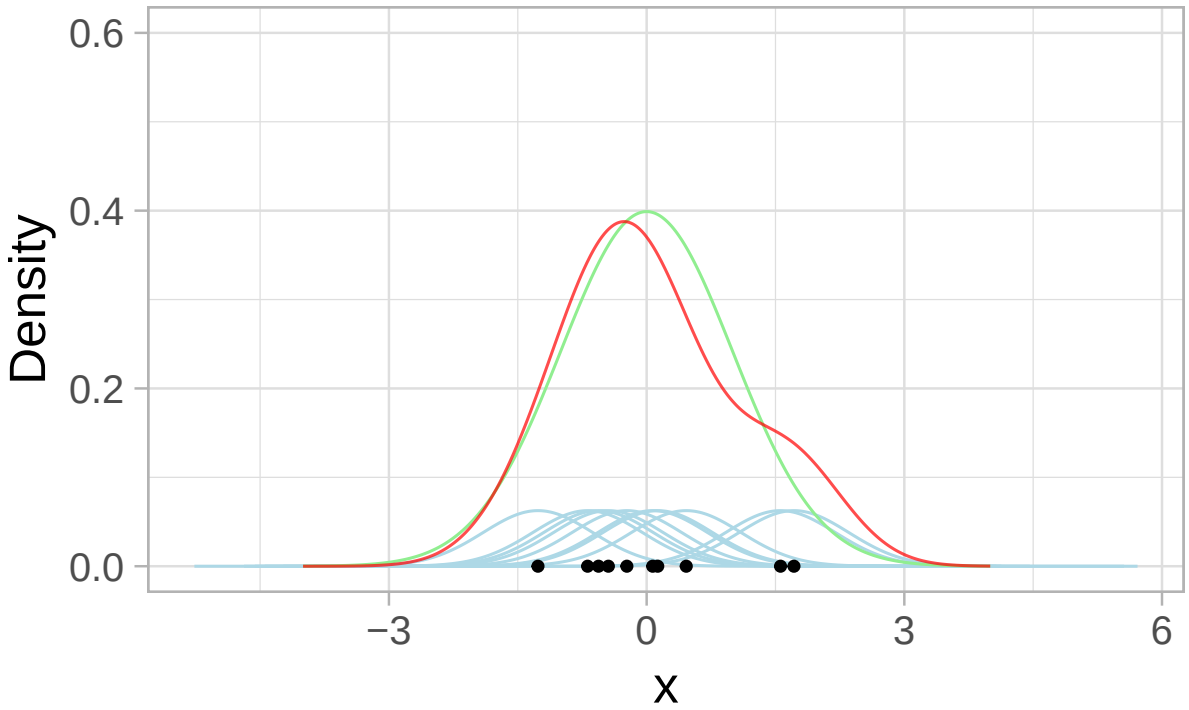
$$\text{MISE} = \int \left[\widehat{f}(x) - f(x)\right]^2 dx$$

  - Rule of thumb: $b = 1.06\sigma n^{-1/5} \Rightarrow$ asymptotically optimal for Gaussian f and K.
  - There are more sophisticated techniques for non-Gaussian data.

Kernel density estimation cont'd

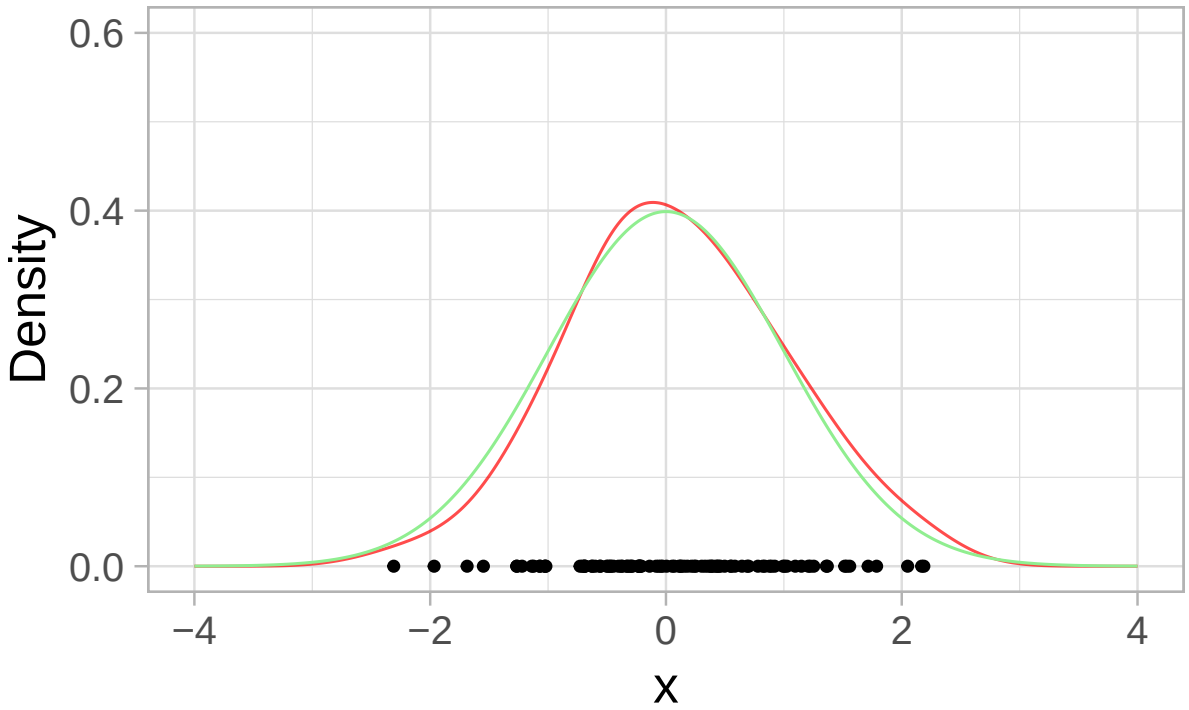
COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

$$f = \mathcal{N}(0,1), \quad n = 10, \quad b = 1.06\sigma n^{-1/5}$$



Kernel density estimation cont'd

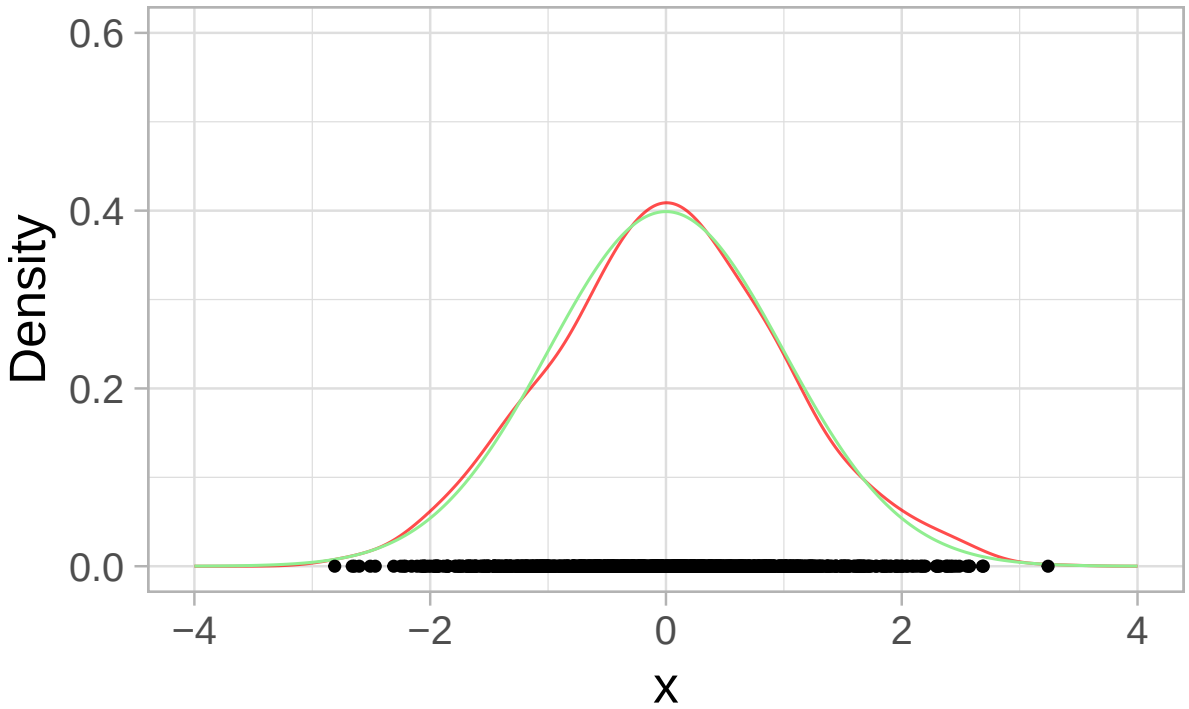
$$f = \mathcal{N}(0,1), \quad n = 100, \quad b = 1.06\sigma n^{-1/5}$$



Kernel density estimation cont'd

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

$$f = \mathcal{N}(0,1), \quad n = 1000, \quad b = 1.06\sigma n^{-1/5}$$



Kernel density estimation in R

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

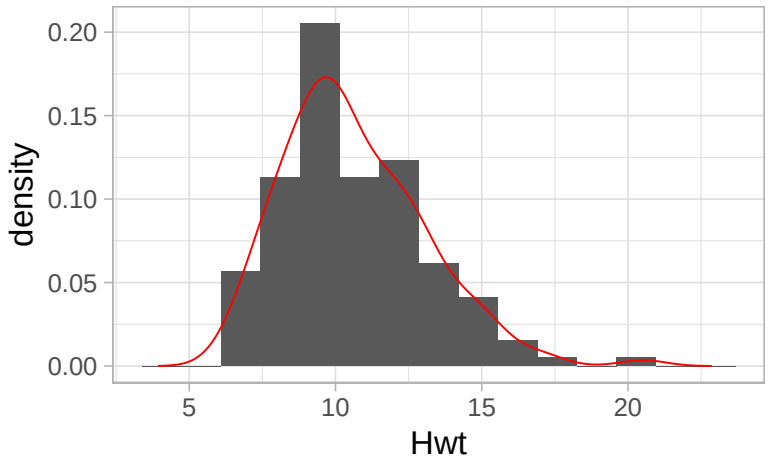
```
(hwt_density <- density(hwt))
#>
#> Call:
#>  density.default(x = hwt)
#>
#> Data: hwt (144 obs.);    Bandwidth 'bw' = 0.7892
#>
#>        x                y
#>  Min.   : 3.93   Min.   :0.0000
#>  1st Qu.: 8.67   1st Qu.:0.0027
#>  Median :13.40   Median :0.0253
#>  Mean   :13.40   Mean   :0.0528
#>  3rd Qu.:18.13   3rd Qu.:0.1020
#>  Max.   :22.87   Max.   :0.1730
hwt_density <- hwt_density %>% broom::tidy()
hwt_density %>% print(n = 3)
#> # A tibble: 512 x 2
#>       x         y
#>   <dbl>     <dbl>
#> 1  3.93 0.0000814
#> 2  3.97 0.0000946
#> 3  4.01 0.000110
#> # ... with 509 more rows
```

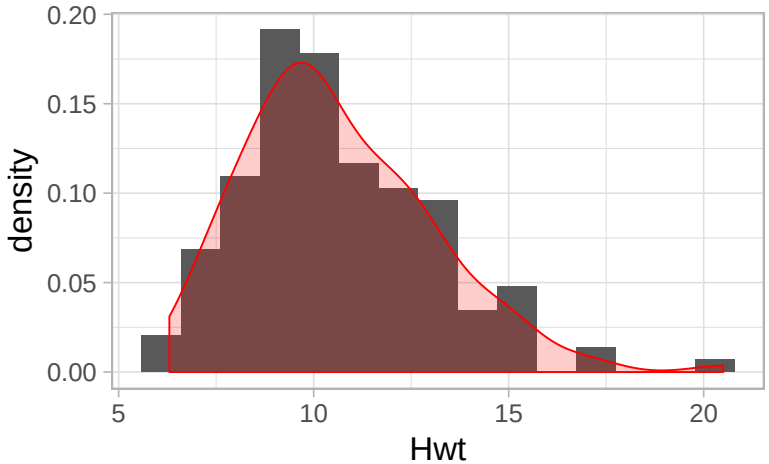
Kernel density estimation in R cont'd

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
cats %>%
  ggplot(aes(x = Hwt)) +
  geom_histogram(aes(y = ..density..), bins = 15) +
  geom_line(aes(x, y), data = hwt_density, color = "red")

cats %>%
  ggplot(aes(x = Hwt)) +
  geom_histogram(aes(y = ..density..), bins = 15) +
  geom_density(color = "red", alpha = .2, fill = "red")
```





Elements of a statistical model

- Let the observed outcome of an experiment be
  - a sample $X_1, \ldots, X_n$ of n i.i.d. random variables,
  - with $X_i \in E$ for some measurable space E (e.g. $E \subset \mathbb{R}^d$),
  - and denote by F their common distribution (i.e., $X_i \sim F$).
- A *statistical model* associated to that experiment is a pair

$$(E, (F_\theta)_{\theta \in \Theta}),$$

where

- E is the **sample space**,
- $(F_\theta)_{\theta \in \Theta}$ is a family of probability measures on E.
- Θ is the **parameter set**.

Goal

Learn about $\theta \in \Theta$ given the data $X_1, \ldots, X_n$.

Identification

Definition

The parameter θ is called identified if and only if the map $\theta \in \Theta \mapsto F_\theta$ is one-to-one, i.e.

$$\theta \neq \theta' \Rightarrow F_\theta \neq F_{\theta'}$$

- Examples
  - If $X_1, \ldots, X_n \overset{iid}{\sim} Ber(p)$, the parameter p is identified.
  - If $X_1, \ldots, X_n \overset{iid}{\sim} N(\mu, \sigma^2)$ with unknown $\mu \in \mathbb{R}$ and $\sigma^2 > 0$, the parameters μ and σ^2 are identified.
  - If $X_1, \ldots, X_n \overset{iid}{\sim} N(\mu, \sigma^2)$ but we only observe $Y_i = \mathbb{1}_{X_i \geq 0}$, then μ and σ^2) are not identified. Note that in this case $\theta = \mu/\sigma$ is identified.

Parametric vs nonparametric

- **Parametric model:**
  - Θ is a a Euclidean space.
  - E.g. $\Theta \subseteq R^p$ for some $p \geq 1$.
- **Nonparametric model:**
  - Θ is not Euclidean.
  - E.g., the kernel densities from before
- **Semiarametric model:**
  - Θ is a product of a Euclidean and a non-Eucidean space.
  - **We will focus on parametric models.**

Why do we care about distributions?

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- The data itself is too much information and *overly detailed*:
  - Don't need to keep around every single data point.
  - The exact same data would never repeat itself if we re-sampled.
- **Goal**: store the information that *summarizes* what will *generalize* to other situations.
- How? Use a model and only keep the parameters!
  - E.g. (μ, σ^2) for the Gaussian distribution.
  - *Fitting* a model to data means finding the parameters such that the model best "matches" the data.
    - Match moments (mean, variances, etc.).
    - Match other summary statistics.
    - Maximize the likelihood.

The Problem of Point Estimation

- Recall our setup:
  - A collection of r.v.'s $X_1, \ldots, X_n$.
  - $X_i \sim F$ with some distribution F.
  - A **parametric model** F_θ with $\Theta \subseteq \mathbb{R}^p$ for some $p \geq 1$.
- Assume that the model is **well specified:**
  - $F = F_{\theta_0}$ for some $\theta_0 \in \Theta$.
  - θ_0 is called the true parameter.

The Problem of Point Estimation

1. Assume F_θ known up to the parameter θ which is unknown.
2. Estimate the value of θ that generated the sample.

Key notions

Definition

A **statistic** is any quantity $T_n = T_n(X_1, \ldots, X_n)$ that can be calculated from the data $X_1, \ldots, X_n$.

- Ex: sample mean, sample variance, mininum, p-value, etc.

Definition

An **estimator** of θ_0 is a statistic whose "purpose" is to estimate θ_0.

- Note that:
  - An estimator can't be a function of θ_0.
  - An estimator $\hat{\theta}_n$ of θ_0 is **unbiased** if $\mathbb{E}[\hat{\theta}_n] = \theta_0$.
  - An estimator $\hat{\theta}_n$ is (weakly) **consistent** for the parameter θ_0 if

$$\hat{\theta}_n \xrightarrow[n \to \infty]{P} \theta_0$$

Outline

Method of moments

- Let $X_1, \ldots, X_n$ be i.i.d. with $X_i \sim F_{\theta_0}$.
  - Population moments:
$$m_k(\theta_0) = \mathbb{E}[X_1^k], \qquad 1 \leq k \leq d$$
  - Empirical moments:
$$\widehat{m}_k = \frac{1}{n}\sum_{i=1}^{n} X_i^k, \qquad 1 \leq k \leq d$$
- Assumme that $\psi(\theta) = (m_1(\theta), \ldots, m_d(\theta))$ is bijective, we have that $\theta = \psi^{-1}(m_1(\theta), \ldots, m_d(\theta))$.
- The **moment of methods estimator** of θ_0 is
$$\widehat{\theta}_n = \psi^{-1}(\widehat{m}_1, \ldots, \widehat{m}_d)$$
provided it exists.

Method of moments

1. Pick enough moments that they *identify* the parameters.
   - I.e., at least one moment per parameter.
   - Denote them by $m_k = E(X^k)$, for $k \in \{1, \dots, p\}$.
2. Write equations for the moments in terms of the parameters.
   - $m_k = \psi_k(\theta_1, \dots, \theta_K)$.
3. Solve the moment equations for the parameters.
   - $\theta_0 = \psi^{-1}(m_1, \dots, m_K)$.
4. Replace the population moments by their estimators.
   - $\widehat{\theta} = \psi^{-1}(\widehat{m}_1, \dots, \widehat{m}_K)$ where $\widehat{m}_k = \frac{1}{n}\sum_{i=1}^n X_i^k$.

- E.g., for the Gaussian:
  1. Need 2 moments to estimate μ and σ^2.
  2. $\mathbb{E}(X) = \mu$ and $\mathbb{E}(X^2) = \sigma^2 + \mu^2$.
  3. $\mu = \mathbb{E}(X)$ and $\sigma^2 = \mathbb{E}(X^2) - \mathbb{E}(X)^2$.
  4. $\widehat{\mu} = \frac{1}{n}\sum_{i=1}^n X_i$ and $\widehat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^n X_i^2 - \frac{1}{n}\sum_{i=1}^n X_i$.

Theory

- $\widehat{\theta}_n$ is consistent under mild assumptions.
- Let
  - $M(\theta_0) = (m_1(\theta_0), \ldots, m_p(\theta_0))$,
  - $\Sigma(\theta_0) = \text{cov}[(X_1, X_1^2, \ldots, X_1^d)]$.

Theorem

If ψ^{-1} is continuously differentiable at $M(\theta_0)$ then

$$\sqrt{n}(\widehat{\theta}_n - \theta_0) \xrightarrow[n\to\infty]{\mathcal{D}} \mathcal{N}(0, V(\theta_0)),$$

where $V(\theta_0) = [\nabla\psi^{-1}(M(\theta_0))]\Sigma(\theta_0)[\nabla\psi^{-1}(M(\theta_0))]^T$.

MOM for the gamma distribution

- Defined by the density function

$$f(x; a, s) = \frac{x^{a-1} e^{-x/s}}{s^a \Gamma(a)},$$

where $\Gamma(a) = \int_0^\infty u^{a-1} e^{-u} du$.

  - a is the **shape**, and s is the **scale**.
  - The expected value is as, and the variance as^2.

- MOM for the gamma distribution:

1. 2 moments are needed.
2. $m_1 = as$ and $m_2 = as^2 + (as)^2$.
3. $a = \frac{m_1^2}{m_2 - m_1^2}$ and $s = \frac{m_2 - m_1^2}{m_1}$.
4. $\widehat{a} = \frac{\widehat{m}_1^2}{\widehat{m}_2 - \widehat{m}_1^2}$ and $\widehat{s} = \frac{\widehat{m}_2 - \widehat{m}_1^2}{\widehat{m}_1}$.

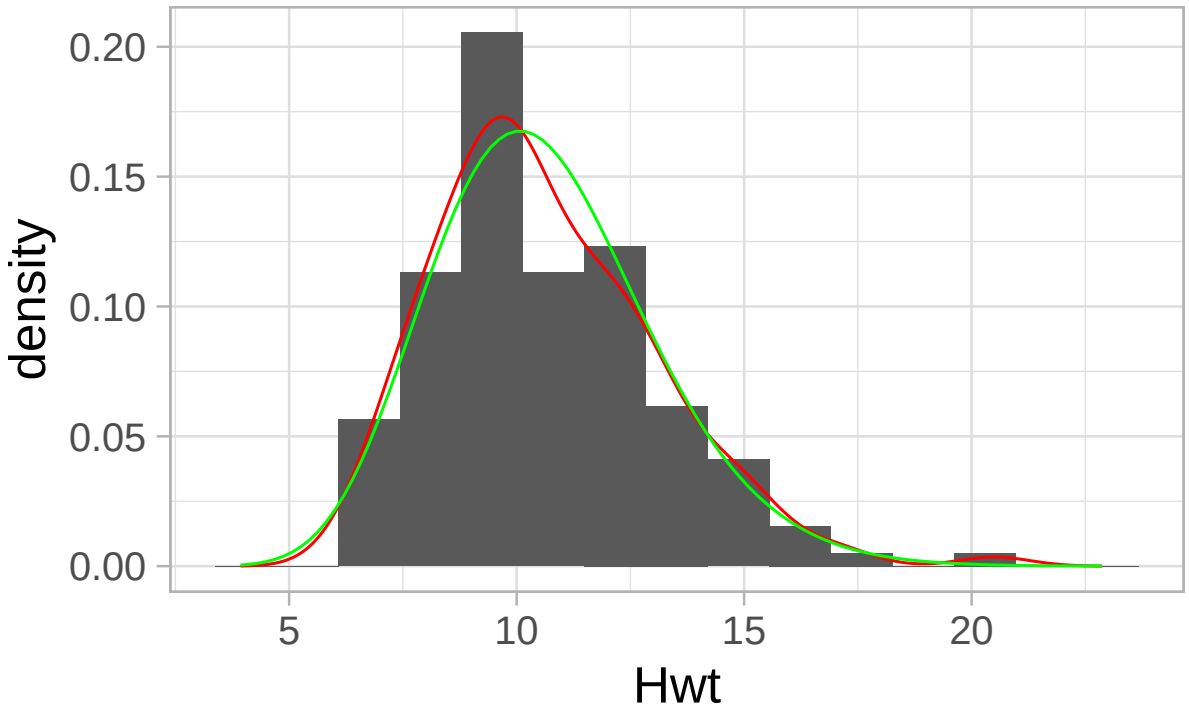
```
par_gamma <- function(x) {
  m <- c(mean(x), var(x))
  c(a = m[1]^2/m[2], s = m[2]/m[1])
}

(hwt_gamma <- par_gamma(hwt))
#>       a       s
#> 19.065   0.558
```

MOM for the gamma distribution

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
hwt_gamma_density <- partial(dgamma,
                             shape = hwt_gamma[1],
                             scale = hwt_gamma[2])
cats %>%
  ggplot(aes(x = Hwt)) +
  geom_histogram(aes(y = ..density..), bins = 15) +
  geom_line(aes(x, y), data = hwt_density, color = "red") +
  stat_function(fun = hwt_gamma_density, color = "green")
```



What if no closed form exist?

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Need numerical optimization to replace step 3!
  - Find $\left\{(\widehat{\theta}_1, \ldots, \widehat{\theta}_K) : \sum_{k=1}^{K} |\widehat{m}_k - \psi_k(\theta_1, \ldots, \theta_K)| = 0\right\}$.
  - Or equivalently $(\widehat{\theta}_1, \ldots, \widehat{\theta}_K) = \underset{(\theta_1, \ldots, \theta_K)}{\text{argmin}} \sum_{k=1}^{K} (\widehat{m}_k - \psi_k(\theta_1, \ldots, \theta_K))^2$.

```r
mom_gamma <- function(par) {
  moments <- c(par[1]*par[2],
               par[1]*par[2]^2)
  return(moments)
}
```

```r
obj_gamma <- function(par, x) {
  differences <- mom_gamma(par) -
    c(mean(x), var(x))
  return(sum(differences^2))
}
```

```r
par_gamma2 <- function(x) {
  optim(c(1, 1), obj_gamma, x = x)$par
}

par_gamma2(hwt)
#> [1] 19.064  0.558
hwt_gamma
#>      a      s
#> 19.065  0.558
```

Using function factories

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
obj_gamma_factory <- function(x) {
  moments <- c(mean(x), var(x))
  function(par) {
    differences <- mom_gamma(par) - moments
    return(sum(differences^2))
    }
}

par_gamma3 <- function(x) {
  obj_gamma_x <- obj_gamma_factory(x)
  optim(c(1, 1), obj_gamma_x)$par
}

par_gamma3(hwt)
#> [1] 19.064  0.558
microbenchmark::microbenchmark(par_gamma(hwt),
                               par_gamma2(hwt),
                               par_gamma3(hwt))
#> Unit: microseconds
#>              expr    min      lq   mean median   uq  max neval
#>   par_gamma(hwt)   11.6   12.8   51.6   13.8   16 3707   100
#>  par_gamma2(hwt) 1890.1 1963.8 2056.1 1991.6 2011 4810   100
#>  par_gamma3(hwt)  276.7  294.0  386.8  299.2  309 6065   100
```

More generally...

- Nothing special about moments. Could match other data summaries too.
  - E.g., the median, quantiles...
- Try to solve for parameters exactly by hand.
  - If you can't, set up a discrepancy function and minimize it.
- Make sure your summaries converge to the population values!
- How? Simulate then estimate and estimates should converge as the sample grows.

Task

- Simulate 100 random variables from a gamma distribution with shape parameter equal to 19 and scale parameter equal to 45.
- Run the `par_gamma` with these values as the input.
- Do the same thing but simulate $1,000$ random variables. Next, $10,000$ random variables.
- Does it seem like our estimates are converging to the truth?

A quick simulation study

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
set.seed(0)
tibble(n = 10^(2:6),
       x = map(n, function(n) rgamma(n, shape = 19, scale = 45)),
       par = map(x, ~ par_gamma(.) %>% enframe(name = "parameter"))) %>%
  unnest(par) %>%
  mutate(error = ifelse(parameter == "a",
                        abs(value - 19), abs(value - 45))) %>%
  dplyr::select(-x)
#> # A tibble: 10 x 4
#>          n parameter value    error
#>      <dbl> <chr>     <dbl>    <dbl>
#>  1     100 a          27.3  8.28
#>  2     100 s          31.5 13.5
#>  3    1000 a          17.4  1.55
#>  4    1000 s          48.5  3.50
#>  5   10000 a          19.0  0.00468
#>  6   10000 s          45.0  0.0303
#>  7  100000 a          19.0  0.0157
#>  8  100000 s          45.0  0.0435
#>  9 1000000 a          18.9  0.0661
#> 10 1000000 s          45.1  0.149
```

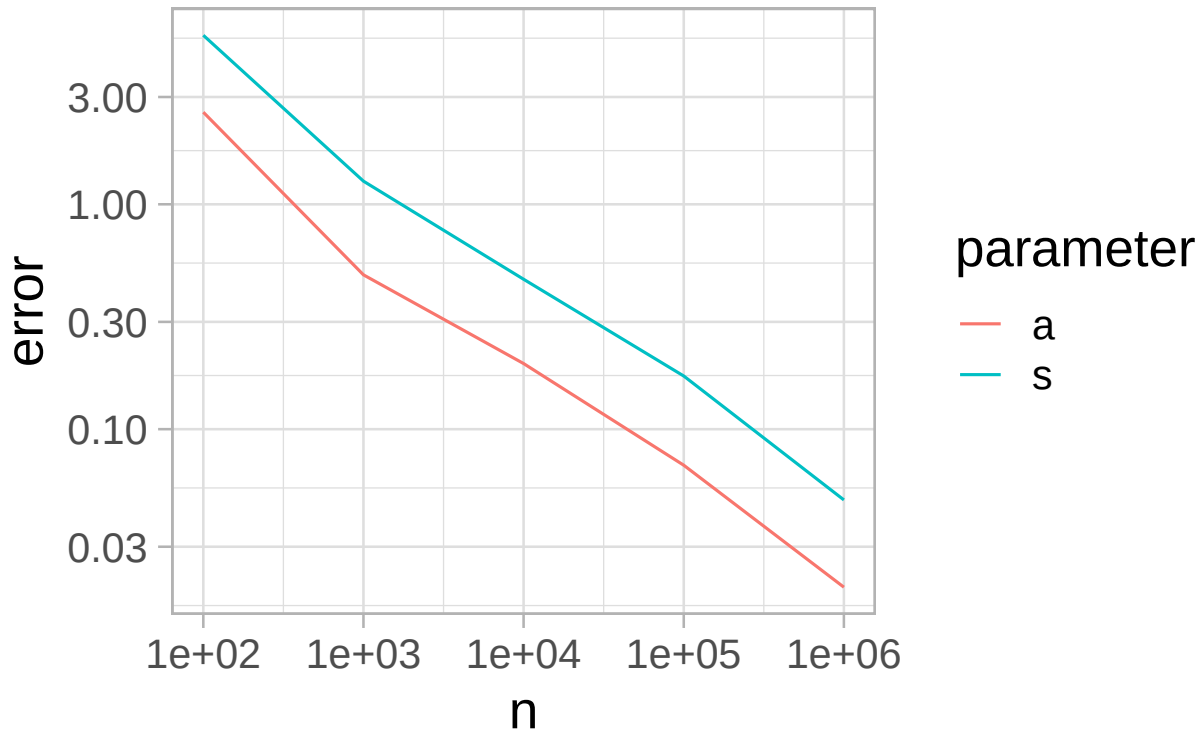
A quick simulation study cont'd

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
do_one <- function(n) {
  tibble(n = n,
         x = map(n, function(n) rgamma(n, shape = 19, scale = 45)),
         par = map(x, ~ par_gamma(.) %>% enframe(name = "parameter"))) %>%
  unnest(par) %>%
  mutate(error = ifelse(parameter == "a",
                        abs(value - 19), abs(value - 45))) %>%
  dplyr::select(-x)
}
gamma_sim <- tibble(replicate = seq(1, 20),
                    sim = map(replicate, ~ do_one(10^(2:6)))) %>%
  unnest(sim)
gamma_sim %>%
  print(n = 5)
#> # A tibble: 200 x 5
#>   replicate     n parameter value  error
#>       <int> <dbl> <chr>     <dbl>  <dbl>
#> 1         1   100 a          24.9  5.86
#> 2         1   100 s          34.4 10.6
#> 3         1  1000 a          18.4  0.552
#> 4         1  1000 s          46.6  1.64
#> 5         1 10000 a          19.2  0.175
#> # ... with 195 more rows
```

A quick simulation study cont'd


```
gamma_sim %>%
  group_by(n, parameter) %>%
  summarize(error = mean(error)) %>%
  ggplot(aes(n, error, color = parameter)) +
  geom_line() +
  scale_x_log10() + scale_y_log10()
```



Outline

The likelihood function

- A central theme in statistics, introduced by Ronald Fisher.
- Let $X_1, \ldots, X_n$ be an i.i.d. sample of rvs with density or frequency $f(x; \theta_0)$.

Definition

Likelihood function for $\Theta \subseteq \mathbb{R}^p$ is $L(\theta; X_1, \ldots, X_n) = \prod_{i=1}^n f(X_i; \theta)$.

- Examples:
  - If $X_1, \ldots, X_n \overset{iid}{\sim} Ber(p)$,

$$L(p; X_1, \ldots, X_n) = \prod_{i=1}^n f(X_i; p) = \prod_{i=1}^n p^{X_i}(1-p)^{1-X_i}.$$

  - If $X_1, \ldots, X_n$ are i.i.d. Exponential rvs with mean $1/\lambda$

$$L(\lambda; X_1, \ldots, X_n) = \prod_{i=1}^n f(X_i; \lambda) = \prod_{i=1}^n \lambda e^{-\lambda X_i} = \lambda^n e^{-\lambda \sum_{i=1}^n X_i}.$$

Maximum likelihood estimator

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

Definition

A **maximum likelihood estimator** of θ_0 is

$$\hat{\theta}_n^{MLE} = \underset{\theta \in \Theta}{\text{argmax}}\ L(\theta; x_1, \ldots, x_n) = \underset{\theta \in \Theta}{\text{argmax}} \log L(\theta; x_1, \ldots, x_n)$$

- Invariance property:
  - For any function $\tau(\theta)$, its MLE is $\tau(\hat{\theta}^{MLE})$.
- Fisher Information:

$$I(\theta) = \text{var}\Big[\frac{\partial}{\partial \theta} \log L(\theta; X_1)\Big] = -\mathbb{E}\Big[\frac{\partial^2}{\partial \theta \partial \theta^T} \log L(\theta; X_1)\Big]$$

- We need the following regularity conditions:
  - The model is identified
  - The support of $f(x;\theta)$ does not depend on θ.
  - θ_0 is not in the boundary of Θ
  - $I(\theta)$ is invertible in a neighborhood of θ_0
  - A few more technical conditions (see Theorem 5.4 in Knight (2000))

Theorem

Under the above conditions $\hat{\theta}_n^{MLE}$ satisfies

- $\hat{\theta}_n^{MLE} \xrightarrow[n\to\infty]{\mathcal{P}} \theta_0$
- $\sqrt{n}(\hat{\theta}_n^{MLE} - \theta_0) \xrightarrow[n\to\infty]{\mathcal{D}} \mathcal{N}(0, I(\theta_0)^{-1})$

In R


```
nll_gamma <- function(par, x) {
  return(sum(dgamma(x, shape = par[1], scale = par[2], log = TRUE)))
}

par_gamma4 <- function(x) {
  par0 <- par_gamma(x)
  optim(par0, obj_gamma, x = x)$par
}

hwt_gamma
#>      a      s
#> 19.065  0.558
par_gamma4(hwt)
#>      a      s
#> 19.065  0.558
```

Using function factories

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Recall that the density is $f(x; a, s) = \frac{x^{a-1} e^{-x/s}}{s^a \Gamma(a)}$.
- So we have
  $L(a, s) = (a-1) \sum_{i=1}^n \log X_i - n \log \Gamma(a) - na \log s - \sum_{i=1}^n X_i / s$.

```
nll_gamma_factory <- function(x) {
  n <- length(x)
  function(par) {
    a <- par[1]
    s <- par[2]
    return(-(a - 1) * sum(log(x)) + n * lgamma(a)
           + n * a * log(s) + sum(x) / s)
  }
}

par_gamma5 <- function(x) {
  par0 <- par_gamma(x)
  nll_gamma_x <- nll_gamma_factory(x)
  optim(par0, nll_gamma_x, lower = 0, method = "L-BFGS-B")$par
}

par_gamma5(hwt)
#>      a      s
#> 20.298  0.524
```

Can we do better?


```
nll_gamma_factory2 <- function(x) {
  n <- length(x)
  sx <- sum(x)
  slx <- sum(log(x))
  function(par) {
    a <- par[1]
    s <- par[2]
    return(-(a - 1) * slx + n * lgamma(a) + n * a * log(s) + sx / s)
  }
}

par_gamma6 <- function(x) {
  par0 <- par_gamma(x)
  nll_gamma_x <- nll_gamma_factory2(x)
  optim(par0, nll_gamma_x, lower = 0, method = "L-BFGS-B")$par
}

par_gamma6(hwt)
#>       a       s
#> 20.298   0.524
```

The benchmark

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

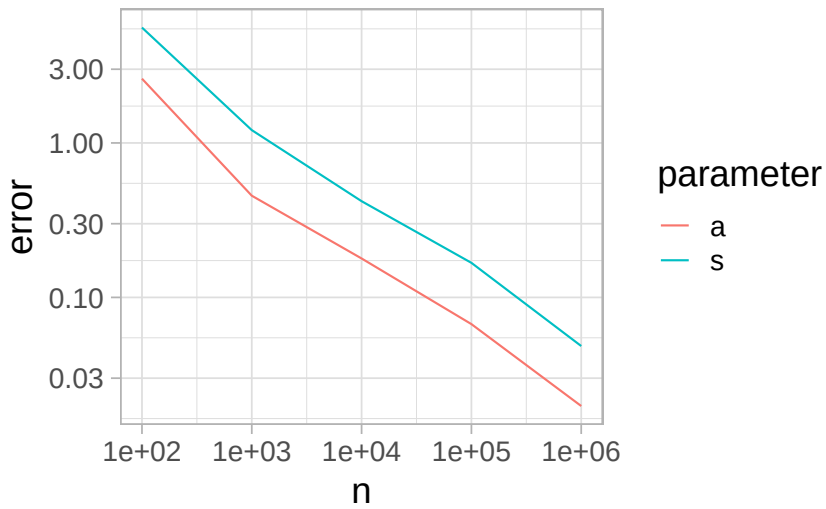
```
microbenchmark::microbenchmark(par_gamma(hwt),
                               par_gamma4(hwt),
                               par_gamma5(hwt),
                               par_gamma6(hwt))
#> Unit: microseconds
#>              expr    min   lq    mean median     uq      max neval
#>   par_gamma(hwt)   11.9   13   15.3   16.2   16.9    20.1   100
#>  par_gamma4(hwt) 1761.4 1844 2677.8 1889.9 1955.3 71973.5   100
#>  par_gamma5(hwt)  509.4  528  652.8  542.8  561.0 11089.0   100
#>  par_gamma6(hwt)  268.9  282  312.7  290.0  297.7  2446.2   100
```

A quick simulation study again

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
set.seed(0)
mydf <- tibble(n = 10^(2:6),
       x = map(n, function(n) rgamma(n, shape = 19, scale = 45)),
       par = map(x, ~ par_gamma6(.) %>% enframe(name = "parameter"))) %>%
  unnest(par) %>%
  mutate(error = ifelse(parameter == "a",
                        abs(value - 19), abs(value - 45))) %>%
  dplyr::select(-x)
```

```
mydf %>%
  print(n = 6)
#> # A tibble: 10 x 4
#>       n parameter value   error
#>   <dbl> <chr>     <dbl>   <dbl>
#> 1   100 a          26.7  7.73
#> 2   100 s          32.2 12.8
#> 3  1000 a          17.9  1.12
#> 4  1000 s          47.3  2.33
#> 5 10000 a          18.9  0.0679
#> 6 10000 s          45.2  0.181
#> # ... with 4 more rows
```



What about standard errors?

- Recall that $\sqrt{n}(\hat{\theta}_n - \theta) \xrightarrow{d} \mathcal{N}\left(0, I^{-1}(\theta)\right)$, where $I(\theta) = \text{cov}_\theta[\nabla \ell(X_i; \theta)] = -\mathbb{E}_\theta[\nabla^2 \ell(X_i; \theta)]$.

```
par_gamma7 <- function(x) {
  par0 <- par_gamma(x)
  nll_gamma_x <- nll_gamma_factory2(x)
  fit <- optim(par0, nll_gamma_x, hessian = TRUE,
               lower = 0, method = "L-BFGS-B")
  fisher_info <- solve(fit$hessian)
  se <- sqrt(diag(fisher_info))
  tibble(parameter = c("a", "s"), value = fit$par, se = se,
         lower = fit$par - 1.96 * se, upper = fit$par + 1.96 * se)
}

par_gamma7(hwt)
#> # A tibble: 2 x 5
#>   parameter  value     se  lower  upper
#>   <chr>      <dbl>  <dbl>  <dbl>  <dbl>
#> 1 a         20.3   2.37   15.6   24.9
#> 2 s          0.524 0.0620  0.402  0.645
```

Outline

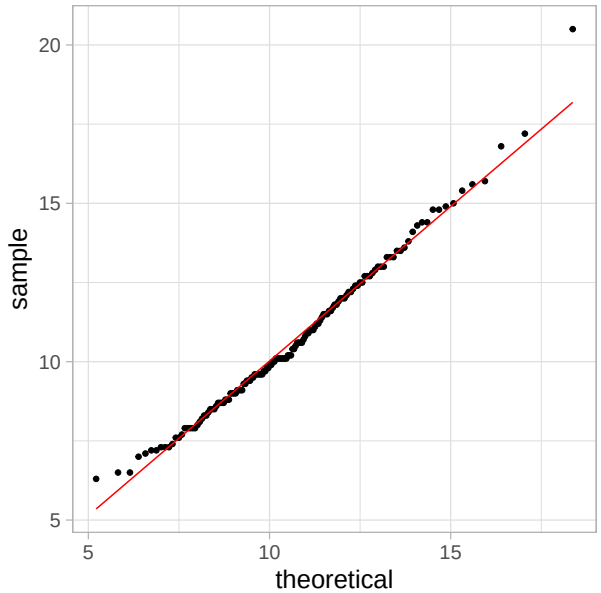
Goodness-of-fit

- Visual
  - QQ plots
- Hypothesis tests
  - KS test

Quantile-Quantile (Q-Q) plots

- Plots theoretical vs. actual quantiles.
- Ideally, a straight line when the distributions are the same.

```
dparams <- list(shape = hwt_gamma[1], scale = hwt_gamma[2])
ggplot(data = cats, aes(sample = Hwt)) +
  geom_qq(distribution = qgamma, dparams = dparams) +
  stat_qq_line(distribution = qgamma, dparams = dparams, color = "red")
```



- Could also plot quantiles of two samples against each other.
  - `geom_qq(aes(x = x, y = y))` gives a Q-Q plot of one vector against another.

Kolmogorov-Smirnoff test

- How much should the Q-Q plot wiggle around the diagonal?
- Answer a different question: define the biggest gap between theoretical and empirical CDF

$$D_{KS} = \sup_{x \in \mathbb{R}} \left| F(x) - \hat{F}(x) \right|$$

- D_{KS} always has the same distribution *if* the theoretical CDF is fixed and correct.

```
ks.test(hwt, pgamma, shape = hwt_gamma[1], scale = hwt_gamma[2])
#>
#>  One-sample Kolmogorov-Smirnov test
#>
#> data:  hwt
#> D = 0.07, p-value = 0.5
#> alternative hypothesis: two-sided
```

Kolmogorov-Smirnoff test cont'd

- Also works for comparing empirical CDF of two samples to see if they come from the same distribution.

```
ks.test(cats$Hwt[cats$Sex == "F"],
        cats$Hwt[cats$Sex == "M"])
#>
#>  Two-sample Kolmogorov-Smirnov test
#>
#> data:  cats$Hwt[cats$Sex == "F"] and cats$Hwt[cats$Sex == "M"]
#> D = 0.5, p-value = 4e-07
#> alternative hypothesis: two-sided
```

Outline

The setup

- Given an **objective function** $f : \mathcal{D} \mapsto R$, find

$$\theta^* = \arg\min_{\theta \in \mathcal{D}} f(\theta).$$

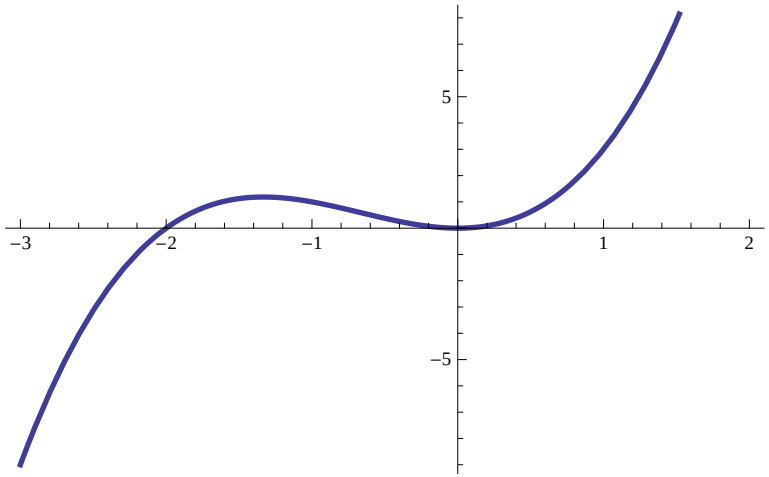
- Examples:
  - Minimize mean-squared error of regression surface (Gauss, c. 1800)
  - Maximize likelihood of distribution (Fisher, c. 1918)
- Basic facts:
  - Maximizing f is minimizing $-f$:

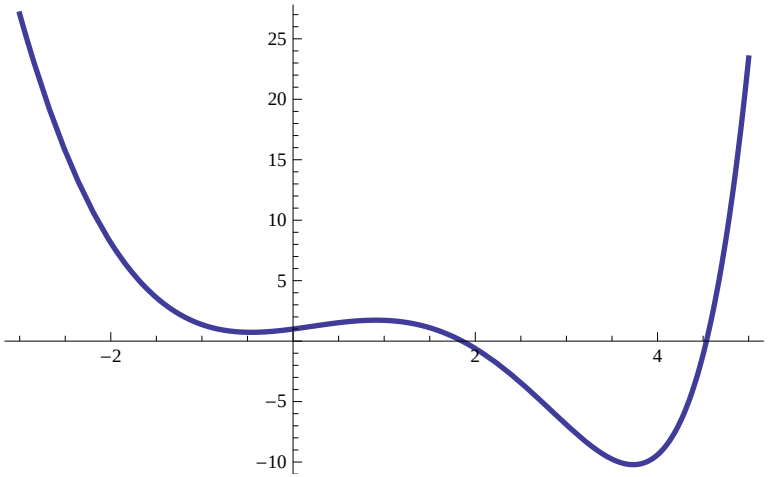
$$\arg\max_{\theta \in \mathcal{D}} f(\theta) = \arg\min_{\theta \in \mathcal{D}} -f(\theta).$$

  - If h is strictly increasing (e.g., log), then

$$\arg\min_{\theta \in \mathcal{D}} f(\theta) = \arg\min_{\theta \in \mathcal{D}} h(f(\theta)).$$

Types of minima





- A minimum of f at θ^* is called:
  - **Global** if f assumes no smaller value on its domain.
  - **Local** if there is some open neighborhood U of θ^* such that $f(\theta^*)$ is a global minimum of f restricted to U.
- Recall that one-dimensional x^* is a local minimum of $f : \mathbb{R} \to \mathbb{R}$ if
  - $f'(x^*) = 0$,
  - $f''(x^*) > 0$.
  - **Carries over to multiple dimensions!**

Remember multivariate calculus?

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Let $f : \mathbb{R}^d \to \mathbb{R}$, then x^* is a local minimum of f if
  - $\nabla f(\theta^*) = 0$,
  - $H_f(\theta^*) = \left(\frac{\partial f}{\partial \theta_i \partial \theta_j}(\theta^*)\right)_{i,j=1,\ldots,d}$ is positive definite.
- The **gradient** of f, denoted ∇f, is a vector of length n with each element equal to the partial derivative of f:

$$\nabla f(\theta) = \left(\frac{\partial f(\theta)}{\partial \theta_1}, \frac{\partial f(\theta)}{\partial \theta_2}, \ldots, \frac{\partial f(\theta)}{\partial \theta_d}\right).$$

- The **Hessian matrix** of f is a square matrix of the second order partial derivatives:

$$[H_f(\theta)]_{i,j} = \frac{\partial f(\theta)}{\partial \theta_i \partial \theta_j} \qquad \text{for } i, j = 1, 2, \ldots, d.$$

- The Hessian matrix is **positive definite** if $z^T H_f z > 0$ for all non-zero vectors $z \in \mathbb{R}^n$. (Positive semidefinite if $z^T H_f z \geq 0$.)

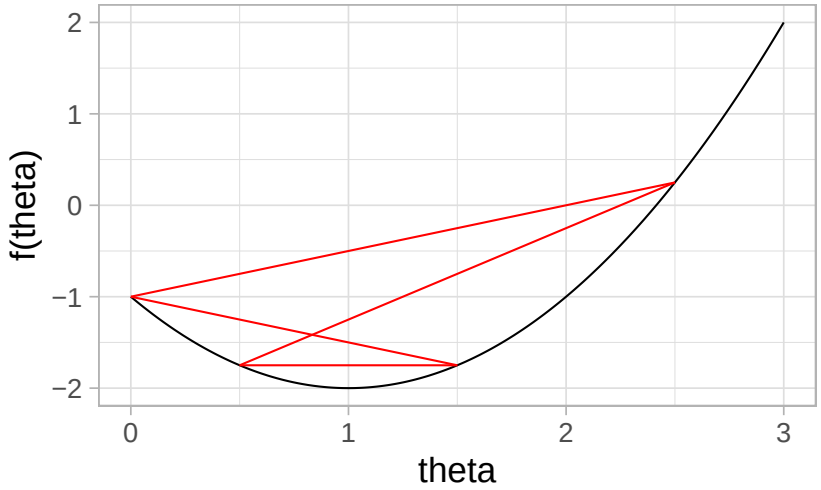
Numerical methods

- All numerical minimization methods perform roughly the same steps:
  - Start with some point θ_0.
  - Our goal is to find a sequence $\theta_0, \ldots, \theta_m$ such that $f(\theta_m)$ is a minimum.
  - At a given point θ_n, compute properties of f (such as $f'(\theta_n)$ and $f''(\theta_n)$).
  - Based on these values, choose the next point θ_{n+1}.
- The information $f'(\theta_n)$, $f''(\theta_n)$ etc is always *local at* θ_n, and we can only decide whether a point is a local minimum, not whether it is global.

Convex functions

Definition (Convex function)

A function f is convex if every line segment between function values lies above the graph of f .



- Analytic criterion:
  - A twice differentiable function is convex if $f''(\theta) \geq 0$ (or $H_f(\theta)$ positive semidefinite) for all θ.
- Implications for optimization
  - $f'(\theta) = 0$ is a sufficient criterion for a minimum.
  - Local minima are global.
  - If f is **strictly convex** ($f'' > 0$ or H_f positive definite), there is only one minimum (which is both gobal and local).

Gradients tell us how f changes

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Recall

$$f'(\theta_0) = \left.\frac{\partial f}{\partial \theta}\right|_{\theta=\theta_0} = \lim_{\theta \to \theta_0} \frac{f(\theta) - f(\theta_0)}{\theta - \theta_0}.$$

- Therefore,

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0) f'(\theta_0).$$

  - Locally, the function looks linear!
  - Minimizing a linear function = moving down the slope.
- Multivariate Version:

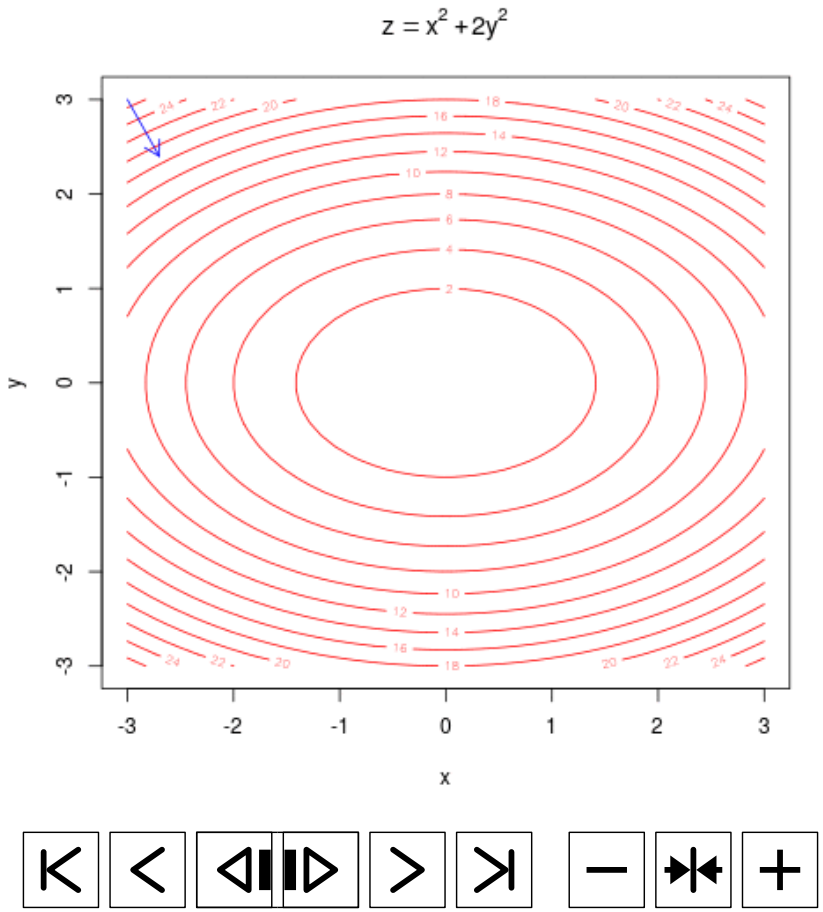
$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0) \cdot \nabla f(\theta_0).$$

  - $\nabla f(\theta_0)$ points in the direction of fastest *descent* at θ_0.

Gradient descent

- The algorithm
  - Choose the step size $\eta > 0$ and the gradient's threshold $\delta > 0$ (typically η and δ are small numbers).
  - Start with an initial guess for θ denoted by θ_0.
  - **for t=1,2,...**
    - Compute gradient $\nabla f(\theta_{t-1})$.
    - **Break** the algorithm if $\|\nabla f(\theta_{t-1})\| < \delta$.
    - **Otherwise** update θ by setting: $\theta_t \leftarrow \theta_{t-1} - \eta \nabla f(\theta_{t-1})$.
  - Return final θ as approximation of θ^*.

Gradient descent cont'd



Gradient descent cont'd

- Pros:
  - Moves in direction of greatest immediate improvement.
  - If η is small enough, gets to a local minimum eventually, and then stops.
- Con:
  - "Small enough" η could be *very* small.
  - Slowness or zig-zagging if components of ∇f are very different sizes.

A simple implementation

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
grad.descent <- function(f, x0, max.iter = 200, step.size = 0.05,
                         stopping.deriv = 0.01, ...) {
  n <- length(x0)
  xmat <- matrix(0, nrow = n, ncol = max.iter)
  xmat[, 1] <- x0
  for (k in 2:max.iter) {
    grad.cur <- numDeriv::grad(f, xmat[, k - 1], ...) # compute gradient
    if (all(abs(grad.cur) < stopping.deriv)) {  #should we stop?
      k <- k - 1
      break
    }
    # Move in the opposite direction of the grad
    xmat[, k] <- xmat[, k - 1] - step.size * grad.cur
  }
  xmat <- xmat[, 1:k] # Trim
  return(list(x = xmat[, k], xmat = xmat, k = k))
}
str(grad.descent(function(x) x^2, c(2, -1), step.size = 1/3))
#> List of 3
#>  $ x   : num [1:2] 0.00274 -0.00137
#>  $ xmat: num [1:2, 1:7] 2 -1 0.667 -0.333 0.222 ...
#>  $ k   : num 7
```

Linear regression

- A small simulation:

```
set.seed(0)
n <- 100
p <- 2
X <- matrix(rnorm(n*p), n, p)
beta <- c(1, 4)
Y <- X %*% beta + rnorm(n)
```

- Using `lm`:

```
coef(lm(Y ~ X + 0))
#>   X1   X2
#> 1.08 3.98
```

- Using `gradient.descent`:

```
mse <- function(beta) sum((Y - X %*% beta)^2)

str(grad.descent(mse, x0 = c(0,0), step.size = 0.05, max.iter = 200))
#> Error in grad.default(f, xmat[, k - 1], ...): function returns NA at
#> 1.587446109648 48e+1493.1045511286547 8e+149 distance from x.
```

Linear regression

- A small simulation:

```
set.seed(0)
n <- 100
p <- 2
X <- matrix(rnorm(n*p), n, p)
beta <- c(1, 4)
Y <- X %*% beta + rnorm(n)
```

- Using `lm`:

```
coef(lm(Y ~ X + 0))
#>    X1    X2
#> 1.08 3.98
```

- Using `gradient.descent` with appropriate step size:

```
str(grad.descent(mse, x0 = c(0,0), step.size = 1e-3, max.iter = 200))
#> List of 3
#>  $ x   : num [1:2] 1.08 3.98
#>  $ xmat: num [1:2, 1:61] 0 0 0.251 0.759 0.448 ...
#>  $ k   : num 61
```

How about quadratic approximations?

- Let's consider a quadratic approximation to f:

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0)f'(\theta_0) + \frac{1}{2}(x - \theta_0)^2 f''(\theta_0).$$

  - Assume θ_0 is a minimum, then $f'(\theta_0) = 0$ and the above simplifies to

$$f(\theta) \approx f(\theta_0) + \frac{1}{2}(\theta - \theta_0)^2 f''(\theta_0).$$

  - **Near a minimum, a smooth function looks like a parabola!**

- The same is true in the multivariate case:

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0)\nabla f(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H_f(\theta_0)(\theta - \theta_0)$$

  - For a minimizing value θ_0, $\nabla f(\theta_0) = 0$ and we have

$$f(\theta) \approx f(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H_f(\theta_0)(\theta - \theta_0).$$

Newton's method

- Let θ_0 be the current guess at a minimizing value.
- Find a quadratic approximation of f at θ_0:

$$f(\theta) \approx f(\theta_0) + (\theta - \theta_0)\nabla f(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H_f(\theta_0)(\theta - \theta_0),$$

which implies that

$$\nabla f(\theta) = \nabla f(\theta_0) + H_f(\theta_0)(\theta - \theta_0).$$

- Update by minimizing the approximation with $\nabla f(\theta) = 0$, which implies

$$\theta = \theta_0 - (H_f(\theta_0))^{-1}\nabla f(\theta_0).$$

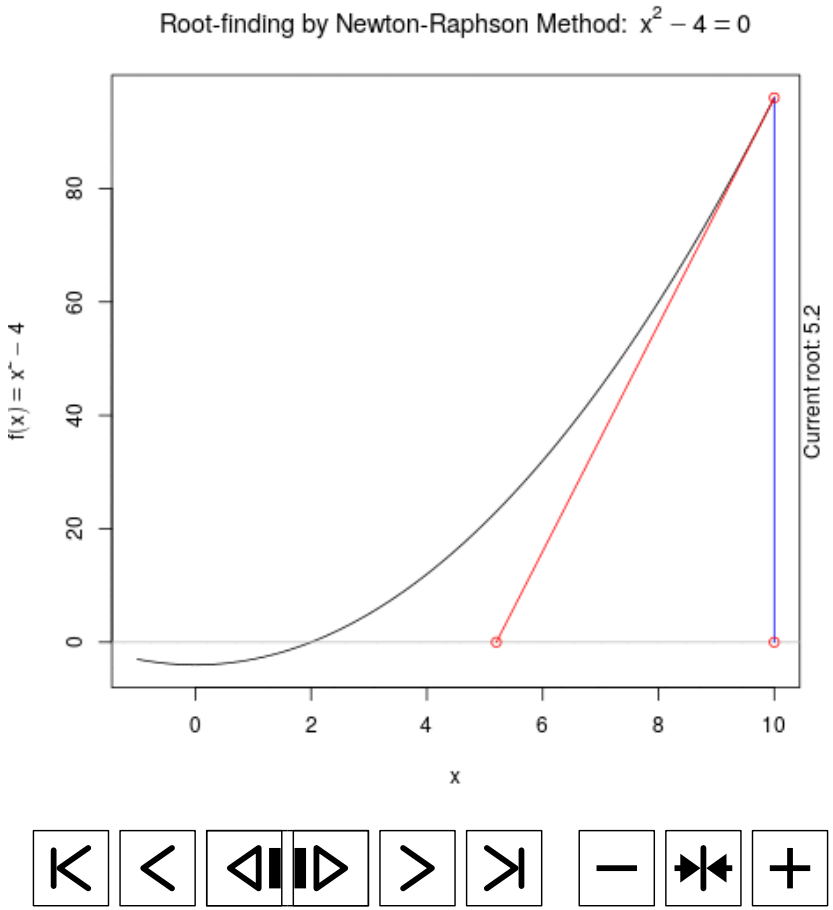
  - If f is exactly quadratic (and $H_f(\theta)^{-1}$ exists), this finds the minimizer exactly.
  - If f isn't quadratic, iterate until convergence.

Newton's method cont'd

- The algorithm
  - Choose the gradient's threshold $\delta > 0$ (typically δ is a small number).
  - Start with an initial guess for θ denoted by θ_0.
  - **for t=1,2,...**
    - Compute both the gradient $\nabla f(\theta)$ and the Hessian $H_f(\theta)$.
    - **Break** the algorithm if $\|\nabla f(\theta_{t-1})\| < \delta$. (Note: we don't need the Hessian for this step)
    - **Otherwise** update θ by setting: $\theta_t \leftarrow \theta_{t-1} - H_f(\theta)^{-1}\nabla f(\theta_{t-1})$.
  - Return final θ as approximation of θ^*.

Newton's method cont'd

- Assume $f(x) = x^3/3 - 4x$ so that $f'(x) = x^2 - 4$.



Newton's method cont'd

- Pros:
  - Step-size chosen adaptively through second derivatives, much harder to get zig-zagging, over-shooting, etc.
  - Always converges if $f''(x) > 0$ (or H_f positive definite).
  - Need $O(\epsilon^{-2})$ steps to get within ϵ of the optimum.
  - Typically many fewer steps than gradient descent.
- Cons:
  - Hopeless if H_f doesn't exist or isn't invertible.
  - Need to take $O(d^2)$ second derivatives plus d first derivatives.
  - Inverting H_f is $O(d^3)$ (i.e., problematic in high dimensions).
- Getting around the Hessian:
  - Use knowledge of the system to get approximations of the Hessian, instead of taking derivatives ("Fisher scoring").
  - Use only diagonal entries (d unmixed second derivatives).
  - Use $H_f(\theta_0)$ at initial guess and hope H_f doesn't change too much.
  - Recompute $H_f(\theta)$ only every k steps for $k > 1$.
  - Approximate updates to the Hessian at each step (BFGS).

Optimization in R: optim()

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- optim(par, fn, gr, method, control, hessian)
  - par: Inital parameter guess; mandatory.
  - fn: Function to be minimized; mandatory.
  - gr: Gradient function; only needed for some methods.
  - method: Defaults to a gradient-free method ('Nedler-Mead'), could be BFGS (Newton-ish).
  - control: Optional list of control settings.
    - E.g., max iterations, step size, tolerance for convergence, etc.
  - hessian: Should the final Hessian be returned? Default is FALSE.
- Returns the location ($par) and the value ($val) of the optimum, diagnostics, possibly the $hessian.

Linear regression again

- A small simulation:

```
set.seed(0)
n <- 2e2
p <- 2
X <- matrix(rnorm(n*p), n, p)
beta <- c(1, 4)
Y <- X %*% beta + rnorm(n)
```

- Using `lm`:

```
coef(lm(Y ~ X + 0))
#>   X1   X2
#> 1.15 4.03
```

- Using `optim`:

```
mse <- function(beta) sum((Y - X %*% beta)^2)
optim(c(0,0), mse)$par
#> [1] 1.15 4.03
```

Linear regression cont'd


```
optim(c(0,0), mse, hessian = TRUE)
#> $par
#> [1] 1.15 4.03
#>
#> $value
#> [1] 211
#>
#> $counts
#> function gradient
#>       65       NA
#>
#> $convergence
#> [1] 0
#>
#> $message
#> NULL
#>
#> $hessian
#>         [,1]   [,2]
#> [1,] 339.34   6.63
#> [2,]   6.63 397.15
```

Linear regression cont'd

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
library(microbenchmark)
grad_mse <- function(beta) -2 *  t(X) %*% (Y - X %*% beta)
tX <- t(X) # Pre-compute the transpose for speed
grad_mse2 <- function(beta) -2 * tX %*% (Y - X %*% beta)
microbenchmark(lm(Y ~ X + 0),
               optim(c(0,0), mse),
               optim(c(0,0), mse, method = "BFGS"),
               optim(c(0,0), mse, grad_mse, method = "BFGS"),
               optim(c(0,0), mse, grad_mse2, method = "BFGS"))
#> Unit: microseconds
#>                                                 expr min  lq mean median
#>                                         lm(Y ~ X + 0) 459 478  499    493
#>                                    optim(c(0, 0), mse) 199 207  219    211
#>               optim(c(0, 0), mse, method = "BFGS") 143 149  156    151
#>   optim(c(0, 0), mse, grad_mse, method = "BFGS") 124 130  165    135
#>  optim(c(0, 0), mse, grad_mse2, method = "BFGS") 103 110  135    114
#>    uq  max neval
#>   515  759   100
#>   225  550   100
#>   163  178   100
#>   142 2937   100
#>   119 2121   100
```

Optimization in R: `nls()`

- `optim()` vs `nls()`
  - `optim()` is a general-purpose optimizer.
    - So is `nls()` – try them both if one doesn't work!
  - `nls()` is for non-linear least squares.
    - The default optimization method for `nls()` is a version of Newton's method.
- `nls(formula, data, start, control, [lot of others...])`
  - `formula`: Mathematical expression with response variables, predictor variable(s), and unknown parameter(s).
  - `data`: Data frame with variable names matching `formula`.
  - `start`: Initial guess at parameters (optional).
  - `control`: Like with `optim()` (optional).
- Returns an `nls` object with fitted values, prediction methods, etc.

Summary

- Trade-offs: complexity of iteration vs. number of iterations vs. precision of approximation.
  - **Gradient descent**: less complex iterations, more guarantees, less adaptive.
  - **Newton's method**: more complex iterations, but few of them for good functions, more adaptive, less robust.
- Start with pre-built code like `optim()` and `nls()`, implement your own as needed.

Outline

Constrained optimization

- Given an **objective function** $f : \mathcal{D} \mapsto R$, find

$$\theta^* = \arg\min_{\theta \in \mathcal{G}} f(\theta),$$

  where $\mathcal{G} \subset \mathcal{D}$ is called the **feasible set**.
- The set $\mathcal{G}$ is often defined by equations, e.g.

$$\mathcal{G} = \{\theta : g(\theta) \geq 0\}.$$

  The equation g is called a **constraint**.
- So far:
  - If f is differentiable, use gradient descent.
  - If f is twice differentiable, use Newton's method.
- Constrained problems
  - Numerical minimizers (like those discussed) use the criterion $\nabla f(\theta) = 0$ for the minimum.
  - In a constrained problem, the minimum is **not** identified by this criterion.

Linear programming: an example

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

Policy	Urban	Suburban	Rural
Building roads	-2	5	3
Gun control	8	2	-5
Farm subsidies	0	0	10
Gasoline tax	10	0	2
Population	100,000	200,000	50,000

Table 1: votes obtained per dollar spent advertising in support of an issue

- **What is the minimum amount of money we can spend to guarantee majority in all demographics?**
- Assume the following:
  - linearity of the objective function and constraints,
  - divisibility of the variables.

Linear programming: an example

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

Policy	Urban	Suburban	Rural
Building roads	-2	5	3
Gun control	8	2	-5
Farm subsidies	0	0	10
Gasoline tax	10	0	2
Population	100,000	200,000	50,000

Table 2: votes obtained per dollar spent advertising in support of an issue

- The objective: $f(\theta) = \theta_{\text{roads}} + \theta_{\text{guns}} + \theta_{\text{farms}} + \theta_{\text{gas}}$.
- The feasible set:

$$\mathcal{G} = \left\{ \theta : \begin{array}{r} -2\theta_{\text{roads}} + 8\theta_{\text{guns}} + 10\theta_{\text{gas}} \geq 100000 \\ 5\theta_{\text{roads}} + 2\theta_{\text{guns}} \geq 200000 \\ 3\theta_{\text{roads}} - 5\theta_{\text{guns}} + 10\theta_{\text{farms}} + 2\theta_{\text{gas}} \geq 50000 \end{array} \right\}$$

Linear programming: general form

$$
\begin{aligned}
\underset{x}{\text{Opt}} \quad & c^\top x \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \ i \in I \subseteq \{1, \ldots, m\} \\
& \sum_{j=1}^{n} a_{kj} x_j \geq b_k, \ k \in K \subseteq \{1, \ldots, m\} \\
& \sum_{j=1}^{n} a_{rj} x_j = b_r, \ r \in R \subseteq \{1, \ldots, m\} \\
& l_j \leq x_j \leq u_j
\end{aligned}
$$

Opt is maximize or minimize, I, K, R are disjoints and $I \cup K \cup R = \{1, \ldots, m\}$, $l_j = -\infty$ and $u_j = \infty$ are possible.

Linear programming in R: lp()

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
library(lpSolve)
const.mat <- matrix(c(-2, 8, 0, 10,
                      5, -5, 0, 0,
                      3, -5, 10, 2),
                      ncol = 4, byrow = TRUE)
solution <- lp(direction = "min",
   objective.in = rep(1, 4),
   const.mat = const.mat,
   const.dir = rep(">=", 3),
   const.rhs = c(100000, 200000, 50000))
solution$objective
#> [1] 1 1 1 1
solution$solution
#> [1] 40000     0     0 18000
const.mat %*% solution$solution
#>        [,1]
#> [1,] 100000
#> [2,] 200000
#> [3,] 156000
```

Quadratic programming: an example

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Let the vector of expected values and variance-covariance matrix for the returns on 3 assets be

$$\mu = \begin{pmatrix} 9 \\ 3 \\ 10 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} 356.25808 & 12.31581 & 261.88302 \\ 12.31581 & 27.24840 & 18.50515 \\ 261.88302 & 18.50515 & 535.45960 \end{pmatrix}$$

- **Problem**: find the portfolio weights that minimize the portfolio variance under the constraints that
  - the sum of the weights equal to 1,
  - the portfolio's expected return equals to 5.2%,
  - each asset weight greater than 0,
  - each asset weight smaller than 0.5.

Quadratic programming: an example

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- **Problem**: find the portfolio weights that minimize the portfolio variance under the constraints that
  - the sum of the weights equal to 1,
  - the portfolio's expected return equals to 5.2%,
  - each asset weight greater than 0,
  - each asset weight smaller than 0.5.
- The objective: $f(\theta) = \theta^\top \Sigma \theta$.
- The feasible set:

$$\mathcal{G} = \left\{ \theta : \begin{array}{rl} \theta_1 + \theta_2 + \theta_3 = 1 & \\ \theta^\top \mu \geq 5.2 & \\ \theta_j \geq 0 & j \in \{1,2,3\} \\ \theta_j \leq 0.5 & j \in \{1,2,3\} \end{array} \right\}$$

Quadratic programming: general form

$$
\begin{aligned}
\underset{x}{\text{Opt}} \quad & -d^\top x + \frac{1}{2} x^\top D x) \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \ i \in I \subseteq \{1, \ldots, m\} \\
& \sum_{j=1}^{n} a_{kj} x_j \geq b_k, \ k \in K \subseteq \{1, \ldots, m\} \\
& \sum_{j=1}^{n} a_{rj} x_j = b_r, \ r \in R \subseteq \{1, \ldots, m\} \\
& l_j \leq x_j \leq u_j
\end{aligned}
$$

Opt is maximize or minimize, I, K, R are disjoints and $I \cup K \cup R = \{1, \ldots, m\}$, $l_j = -\infty$ and $u_j = \infty$ are possible.

Quadprog in R: solve.QP()

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
library(quadprog)
Sigma <- matrix(c(356.25808, 12.31581, 261.88302,
                  12.31581, 27.24840, 18.50515,
                  261.88302, 18.50515, 535.45960),
                nrow = 3, ncol = 3)
mu <- matrix(c(9, 3, 10), nrow = 3, ncol = 1)

A.Equality <- matrix(c(1, 1, 1), ncol = 1)
Amat <- cbind(A.Equality, mu, diag(3), -diag(3))
bvec <- c(1, 5.2, rep(0, 3), rep(-0.5, 3))
qp <- solve.QP(Sigma, rep(0, 3), Amat, bvec, meq = 1)
qp$solution
#> [1] 0.38 0.50 0.12
t(qp$solution) %*% mu
#>      [,1]
#> [1,] 6.12
t(qp$solution) %*% Sigma %*% qp$solution
#>      [,1]
#> [1,] 96.8
```

Linearly constrained optimization

$$
\begin{aligned}
\underset{x}{\text{Opt}} \quad & f(x) \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \ i \in I \subseteq \{1, \ldots, m\} \\
& \sum_{j=1}^{n} a_{kj} x_j \geq b_k, \ k \in K \subseteq \{1, \ldots, m\} \\
& \sum_{j=1}^{n} a_{rj} x_j = b_r, \ r \in R \subseteq \{1, \ldots, m\} \\
& l_j \leq x_j \leq u_j
\end{aligned}
$$

Opt is maximize or minimize, I, K, R are disjoints and $I \cup K \cup R = \{1, \ldots, m\}$, $l_j = -\infty$ and $u_j = \infty$ are possible.

Barrier Methods

- A.K.A. “interior point”, “central path”, etc.
- Having constraints switch on and off abruptly is annoying, especially with gradient methods.
- Fix $\mu > 0$ and try minimizing

$$f(\theta) - \mu \log (d - h(\theta))$$

- It “pushes away" from the barrier – more and more weakly as $\mu \to 0$
- Algorithm
  - Choose initial guess of θ in the feasible set and initial μ.
  - While ((not too tired) and (making adequate progress))
    - Minimize $f(\theta) - \mu \log (d - h(\theta))$,
    - Reduce μ.
  - Return final θ as approximation of θ^*.

In R: constrOptim()

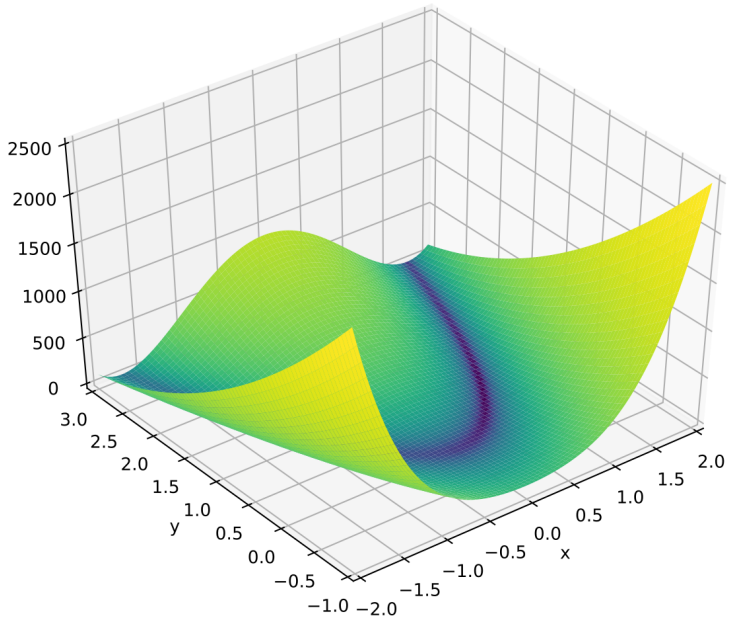
- Rosenbrock banana:

```
frb <- function(x) {
    x1 <- x[1]
    x2 <- x[2]
    100 * (x2 - x1 * x1)^2 +
      (1 - x1)^2
}
```

- Gradient of `frb()`:

```
grrb <- function(x) {
    x1 <- x[1]
    x2 <- x[2]
    c(-400 * x1 * (x2 - x1 * x1) -
        2 * (1 - x1),
       200 *      (x2 - x1 * x1))
}
```

- From wikipedia:



In R: constrOptim()

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

- Rosenbrock banana:

```
frb <- function(x) {
    x1 <- x[1]
    x2 <- x[2]
    100 * (x2 - x1 * x1)^2 +
      (1 - x1)^2
}
```

- Gradient of frb():

```
grrb <- function(x) {
    x1 <- x[1]
    x2 <- x[2]
    c(-400 * x1 * (x2 - x1 * x1) -
        2 * (1 - x1),
       200 *      (x2 - x1 * x1))
}
```

```
optim(c(-1.2, 1), frb, grrb)$par
#> [1] 1 1
# Box-constraint, optimum on the boundary
constrOptim(c(-1.2, 0.9), frb, grrb,
            ui = rbind(c(-1, 0), c(0, -1)), ci = c(-1, -1))$par
#> [1] 1 1
# x <= 0.9,  y - x > 0.1
constrOptim(c(.5, 0), frb, grrb,
            ui = rbind(c(-1, 0), c(1, -1)), ci = c(-0.9, 0.1))$par
#> [1] 0.889 0.789
```

CRAN Task View on optimization

- Check the CRAN Task View on optimization!
  - Optimization infrastructure packages.
  - General purpose solvers.
  - Linear programming.
  - Quadratic optimization.
  - Least-squares problems.
  - Convex solvers.
  - Combinatorial optimization.
  - Global and stochastic optimization.
  - Interfaces to open-source as well as commercial optimizers.