Statistical Computing with R

Lecture 10: numeric optimization; maximum likelihood; confidence intervals; hypothesis testing

Mirko Signorelli
: mirkosignorelli.github.io
: statcompr [at] gmail.com

Mathematical Institute
Leiden University

Master in Statistics and Data Science (2023-2024)

Universiteit Leiden

Announcements

- A minor change:
  1. we noticed that the questions and solutions to exercise 2 of coding session 7 were somewhat unclear
  2. we uploaded an updated version of this exercise (both questions, and solution) to Brightspace:

Lecture 7

- Lecture: slides
- Coding session exercises and solutions, with 14/11/2023 update to exercise 2 (files with old version of exercise 2: exercises and solutions)

- Assignment 3 will be published after lecture 11 (tentative: Nov. 24)

Recap

Lecture 9:

- general programming tips
- tracking computing time
- making your code faster
- probability calculus in `R`
- discrete distributions
- continuous distributions

Today:

- numeric optimization
- maximum likelihood
- confidence intervals
- hypothesis testing

Warning: important PREREQUISITES

- "Statistical Computing" wouldn't exist without Statistics, and Statistics wouldn't exist without Mathematics and Probability Theory
- Last week we started using concepts from probability calculus
- This and next week's lectures and coding sessions ⚠ require you to be familiar ⚠ with:
  1. basics of probability theory, maximum likelihood estimation, confidence intervals and hypothesis testing → Probability and Statistics course
  2. derivatives and optimization → Mathematics for Statisticians course

Optimization problems

Numeric optimization

Maximum likelihood estimation

Confidence intervals

Hypothesis testing

Optimization

- Many problems in statistics involve the optimization of an objective function
- Optimization = finding the maxima / minima of a function
  - Simpler optimization problems can often be solved analytically
  - More complex problems require numeric optimization

⚠ If a problem can be solved analytically, it is better to solve it both analytically and numerically:

- if the two solutions agree: you're (probably?) done ✓
- if they disagree: one of the two must be wrong! $\Rightarrow$ double-check both until you figure out where the problem is ⚠

Example

- We want to find the maximum and minimum of

$$f(x) = x\log(x),\ x > 0$$

- We can solve this problem analytically by studying the sign of $f'(x)$:
  - $f'(x) > 0$: the function is increasing
  - $f'(x) < 0$: the function is decreasing
  - $f'(x) = 0$: you have reached a (local) optimum! ☺☺☺

Example (cont'd)

$$f(x) = x\log(x),\ x > 0$$

- $f'(x)$ can be computed by applying the product rule:

$$f'(x) = 1\log(x) + x\frac{1}{x} = \log(x) + 1$$

- $f'(x) = \log(x) + 1 > 0 \iff \log(x) > -1 \iff x > \frac{1}{e}$
- $f'(x) < 0 \iff 0 < x < \frac{1}{e}$
- $f'(x) = 0 \iff x = \frac{1}{e}$

Conclusions:

- $x = \frac{1}{e}$ is a global minimum for $f(x)$
- no global maximum: $\lim_{x\to\infty} f(x) = \infty$

Optimization problems

Numeric optimization

Maximum likelihood estimation

Confidence intervals

Hypothesis testing

Numeric optimization

- When dealing with optimization problems, we often need to implement computational ("numeric") approaches to either:
  1. double-check the solution of a problem that we were able to solve analytically
  2. find an (approximate) solution of a problem that we cannot solve analytically

R functions for numeric optimization

R has several built-in functions for optimization. A few examples:

- `optimize( )`: unidimensional optimization
- `optim( )`: multidimensional optimization
- `constrOptim( )`: constrained optimization (with linear inequality constraints)
- `nlm( )`: multidimensional optimization
- `mle( )`: just a wrapper of `optim( )`

We will focus on `optimize( )` and `optim( )`!

optimize( )

- Syntax of `optimize( )`:

```
optimize(f, interval, maximum = FALSE, ...)
```

- ⚠ by default, `optimize( )` minimizes `f` ⚠
- Set `maximum = TRUE` if you are after a maximum!

Example (cont'd)

$$f(x) = x\log(x),\ x > 0$$

```
f = \(x) x * log(x)
optimize(f, interval = c(0.001, 10))
```

```
## $minimum
## [1] 0.3678774
##
## $objective
## [1] -0.3678794
```

- We already solved this analytically ⇒ let's double-check the result!

```
# analytical solution:
exp(-1) # x (minimum)
```

```
## [1] 0.3678794
```

```
f(exp(-1)) # f(x)
```

```
## [1] -0.3678794
```

Example (cont'd)

- What happens if we look for the maximum?

```r
optimize(f, interval = c(0.001, 10), maximum = TRUE)
```

```
## $maximum
## [1] 9.999944
##
## $objective
## [1] 23.02567
```

```r
optimize(f, interval = c(0.001, 20), maximum = TRUE)
```

```
## $maximum
## [1] 19.99993
##
## $objective
## [1] 59.91435
```

```r
optimize(f, interval = c(0.001, 30), maximum = TRUE)
```

```
## $maximum
## [1] 29.99993
##
## $objective
## [1] 102.0356
```

optim( )

- Syntax of `optim( )` is more complex:

```
optim(par, fn, gr = NULL, ...,
      method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B",
                 "SANN", "Brent"),
      lower = -Inf, upper = Inf,
      control = list(), hessian = FALSE)
```

- `par` = vector of starting values, `fn` = function to minimize
- ⚠ `optim( )` always performs minimizations ⚠ ⇒ supply `fn = -f` if you are after a maximum!
- Multiple algorithms available. Most commonly used `methods`: `Nelder-Mead` (default; slower, "derivative-free", very reliable), `BFGS` (faster quasi-Newton algorithm, derivative-based, more likely to fail than N-M), `L-BFGS-B` (constrained optimization)
- See `?optim` for more details
- Examples in a few slides ☺

Optimization problems

Numeric optimization

Maximum likelihood estimation

Confidence intervals

Hypothesis testing

Likelihood

- Likelihood = joint density (pdf) of an observed sample $(x_1, ..., x_n)$

$$L(\theta) = f(x_1, ..., x_n; \theta),$$

where $\theta \in \mathbb{R}^p$ denotes the parameter vector in the pdf f

- Log-likelihood: $\ell(\theta) = \log[L(\theta)]$
- If $X_1, ..., X_n$ are i.i.d. $\sim f(x_i; \theta)$, then

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

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

Maximum likelihood

- The maximum likelihood estimator $\hat{\theta}$ is the value that maximizes the (log-)likelihood:

$$\hat{\theta} = \text{argmax}_{\theta \in \Theta} \, \ell(\theta)$$

- ⚠ Use of $\ell(\theta)$ generally preferable (a.k.a. easier) than of $L(\theta)$ ⚠
- Maximum likelihood is an optimization problem $\Rightarrow \hat{\theta}$ can be found by studying the sign of the (partial) derivative(s) of $\ell(\theta)$ wrt θ

Example: likelihood for $X \sim N(\mu, \sigma^2)$

- Let's consider $X_1, ..., X_n$ i.i.d. $X \sim N(\mu, \sigma^2)$:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \; x \in (-\infty, \infty)$$

- After a few intermediate steps, we get:

$$\ell(\mu, \sigma) = -n\log(\sigma) - n\log(\sqrt{2\pi}) - \frac{1}{2}\sum_i \left(\frac{x_i - \mu}{\sigma}\right)^2$$

$$= -n\log(\sigma) - n\log(\sqrt{2\pi}) - \frac{1}{2}\sum_i \frac{x_i^2 + \mu^2 - 2\mu x_i}{\sigma^2}$$

Example: likelihood for $X \sim N(\mu, \sigma^2)$ (cont'd)

- Here we have two variables, so we need to compute both $\frac{\partial \ell}{\partial \mu}$ and $\frac{\partial \ell}{\partial \sigma}$:

$$\ell(\mu, \sigma) = -n \log(\sigma) - n \log(\sqrt{2\pi}) - \frac{1}{2} \sum_i \frac{x_i^2 + \mu^2 - 2\mu x_i}{\sigma^2}$$

$$\frac{\partial \ell}{\partial \mu} = -\frac{1}{2} \sum_i \frac{2\mu - 2x_i}{\sigma^2}$$

$$\frac{\partial \ell}{\partial \mu} > 0 \iff \sum_i (\mu - x_i) < 0 \iff \mu < \frac{1}{n} \sum_i x_i$$

$$\Rightarrow \hat{\mu} = \frac{1}{n} \sum_i x_i$$

Example: likelihood for $X \sim N(\mu, \sigma^2)$ (cont'd)

$$\ell(\mu, \sigma) = -n\log(\sigma) - n\log(\sqrt{2\pi}) - \frac{1}{2}\sigma^{-2}\sum_i (x_i - \mu)^2$$

$$\frac{\partial \ell}{\partial \sigma} = -\frac{n}{\sigma} + \frac{\sum_i (x_i - \mu)^2}{\sigma^3} > 0 \iff \sigma^{-2}\sum_i (x_i - \mu)^2 > n$$

$$\iff \sigma^2 < \frac{\sum_i (x_i - \mu)^2}{n}$$

$$\Rightarrow \hat{\sigma}^2 = \frac{\sum_i (x_i - \hat{\mu})^2}{n}$$

- Note that to be able to compute $\hat{\sigma}^2$ we need to replace the unknown μ with its "plug-in estimator" $\hat{\mu} = \bar{x}$ (that we obtained in the previous slide)

Analytical vs numerical optimization

- The analytical solution yields a "general" formula to compute the MLE for any given $(x_1, ..., x_n)$

$$\hat{\mu} = \frac{1}{n} \sum_i x_i$$

$$\hat{\sigma}^2 = \frac{\sum_i (x_i - \hat{\mu})^2}{n}$$

- The numerical solution is less general and less insightful: it only yields a numeric value for each specific $(x_1, ..., x_n)$ = no general recipe on what to do for any $(x_1, ..., x_n)$!
- Take-home message: numeric optimization does not (and should not) replace math knowledge!

Example: likelihood for $X \sim N(\mu, \sigma^2)$ (cont'd)

- Set random seed for reproducibility:

```
set.seed(19931101)
```

- Generate random numbers from $X \sim N(\mu, \sigma^2)$:

```
n = 20; mu = 3.1; sigma = 1.3
x = rnorm(n, mu, sigma)
```

- Implement function that evaluates $-\log L(\theta)$, where $\theta = (\mu, \sigma)$:

```
neg.logl = function(theta) {
  if (length(theta) != 2) stop('theta should be a
                                2-dimensional vector')
  mu = theta[1]
  sigma = theta[2]
  if (sigma < 0) stop('theta[2] (sigma) should be non-negative')
  -sum(dnorm(x, mu, sigma, log = T))
}
```

How to solve this problem numerically?

- Is the optimization problem constrained, or unconstrained?
- If the problem is unconstrained, you can directly use `optim` with the Nelder-Mead or BFGS-B algorithms as `method`
- If, instead, the problem is constrained, then you should either
  1. use `optim` with the L-BFGS-B algorithm as `method`, and speficy proper constraints using `lower` and `upper`
  2. reparametrize the original (constrained) problem into an unconstrained optimization problem, and then use the Nelder-Mead or BFGS-B algorithms as `method` within `optim`

Solution using Nelder-Mead (default)

- By default, `optim( )` uses the Nelder-Mead algorithm, which is designed to solve unconstrained optimization problems:

```
optim(c(0, 1), neg.logl)
```

- Although in this case you still obtain the correct solution, here using Nelder-Mead straight away is technically WRONG. Why?
- $\sigma \geq 0 \Rightarrow$ ⚠ this optimization problem is constrained! ⚠

Constrained problems: the L-BFGS-B algorithm

- L-BFGS-B designed for constrained optimization. Here's how to use it:

```
mle1 = optim(c(7, 1), neg.logl, method = 'L-BFGS-B',
      lower = c(-Inf, 0), upper = c(Inf, Inf))
```

```
mle1
```

```
## $par
## [1] 3.316312 1.353255
##
## $value
## [1] 34.42902
##
## $counts
## function gradient
##       17       17
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Double-check

- Numeric solution:

```
mle1$par
```

```
## [1] 3.316312 1.353255
```

- Analytical solution:

```
mean(x)
```

```
## [1] 3.316315
```

```
sqrt(var(x)*(n-1)/n)
```

```
## [1] 1.353255
```

- Approximate (numeric) solution very close to exact (analytical) solution ✓

Turning a constrained into an unconstrained problem

- L-BFGS-B is usually "less robust" than Nelder-Mead, and it can sometimes fail to converge
- An alternative, often more reliable, way to solve the problem is to reparametrize your objective function so as to make the problem unconstrained (see next slide), and then use Nelder-Mead

Reparametrization + Nelder Mead

- Nelder-Mead (and BFGS) are designed to solve unconstrained optimization problems
- Here we have: $\mu \in \mathbb{R}$, $\sigma > 0 \Rightarrow$ a constrained problem
- We can make the problem unconstrained by setting

$$\tau = \log(\sigma) \in \mathbb{R}, \quad \texttt{i.e.,} \quad \sigma = \exp(\tau) > 0$$

```
reparametrized.nlogl = function(theta) {
  if (length(theta) != 2) stop('theta should be a
                                2-dimensional vector')
  mu = theta[1]
  sigma = exp(theta[2])
  -sum(dnorm(x, mu, sigma, log = T))
}
```

- This often solves problems encountered byL-BFGS-B with constrained optimization ☺

Reparametrization + Nelder Mead (cont'd)

- Numeric solution:

```
mle2 = optim(c(0, 0), reparametrized.nlogl)
```

- ⚠ the MLE from `optim( )` is now on the unconstrained (μ, τ) scale ⚠

```
mle2$par
```

```
## [1] 3.3166003 0.3025853
```

- Don't forget to bring it back to the original (μ, σ) scale:

```
c(mle2$par[1], exp(mle2$par[2]))
```

```
## [1] 3.316600 1.353353
```

```
# exact values:
c(mean(x), sqrt(var(x)*(n-1)/n))
```

```
## [1] 3.316315 1.353255
```

Concluding remarks

- Numeric optimization allows us to find an approximate solution for a given sample $(x_1, ..., x_n)$
- However, it does not help us in finding:
  - the exact solution for the given sample
  - a general solution (i.e., a closed-form expression) that applies for any sample
- Analytical solution more difficult and not always feasible, but:
  - it's more general
  - it yields an exact solution
  - it allows us to understand how the MLE is obtained, and to study its properties

⚠ Numeric optimization useful when we cannot work out the math, but it does not replace math knowledge! ⚠

Optimization problems

Numeric optimization

Maximum likelihood estimation

Confidence intervals

Hypothesis testing

Confidence intervals

- Let $X_1, ..., X_n$ i.i.d. $\sim f(x, \theta)$, with $\theta \in \mathbb{R}$ an unknown parameter
- A confidence interval $(a(X), b(X))$ for θ with $1 - \alpha$ confidence level is an interval that satisfies

$$P(a(X) < \theta < b(X)) = 1 - \alpha$$

- How can we compute a confidence interval for θ in `R`?

Confidence intervals in R

- R offers several built-in functions that compute confidence intervals for the most common problems
- Often, these functions also compute the corresponding hypothesis test
- Most common CIs:

Function	Confidence interval for...
`binom.test(x, n, ...)`	probability of success of a Bernoulli trial
`t.test(x, ...)`	mean of the normal distribution
`t.test(x, y, ...)`	mean difference for two normal distributions

- Confidence level: default is usually `conf.level = 0.95`, but it can be changed!

Example

```r
set.seed(10) # important: set random seed for reproducibility!
n = 50
x1 = sample(x = c(0, 1), size = n, replace = T, prob = c(0.3, 0.7))
n.successes = sum(x1 == 1)
n.successes
```

```
## [1] 38
```

```r
# how to compute the confidence interval:
ci1 = binom.test(x = c(n.successes, n-n.successes),
                 conf.level = 0.95)
ls(ci1)
```

```
## [1] "alternative" "conf.int"    "data.name"
## [4] "estimate"    "method"      "null.value"
## [7] "p.value"     "parameter"   "statistic"
```

```r
# confidence interval is stored in object$conf.int
ci1$conf.int
```

```
## [1] 0.6183093 0.8693901
## attr(,"conf.level")
## [1] 0.95
```

Example (cont'd)

- Note how the output of `binom.test( )` contains much more information than just the CI:

```r
binom.test(x = c(n.successes, n-n.successes), conf.level = 0.95)
```

```
##
##  Exact binomial test
##
## data:  c(n.successes, n - n.successes)
## number of successes = 38, number of trials = 50,
## p-value = 0.0003059
## alternative hypothesis: true probability of success is not equal to
## 95 percent confidence interval:
##  0.6183093 0.8693901
## sample estimates:
## probability of success
##                   0.76
```

Example (cont'd)

As expected, the width of the interval increases with higher confidence levels:

```
x = c(n.successes, n-n.successes)
binom.test(x, conf.level = 0.90)$conf.int
```

```
## [1] 0.6403443 0.8552818
## attr(,"conf.level")
## [1] 0.9
```

```
binom.test(x, conf.level = 0.95)$conf.int
```

```
## [1] 0.6183093 0.8693901
## attr(,"conf.level")
## [1] 0.95
```

```
binom.test(x, conf.level = 0.99)$conf.int
```

```
## [1] 0.5745078 0.8944499
## attr(,"conf.level")
## [1] 0.99
```

Optimization problems

Numeric optimization

Maximum likelihood estimation

Confidence intervals

Hypothesis testing

Hypothesis testing

- Let $\theta \in \Theta$ and suppose we want to test

$$H_0 : \theta = \theta_0 \text{ vs } H_1 : \theta \neq \theta_0$$

- We use a test statistic $T = g(X)$ to determine whether H_0 can be rejected or not based on the observed sample $x = (x_1, ..., x_n)$
- We prespecify the type I error probability / significance level, i.e. the probability to reject H_0 when H_0 is true:
  $\alpha = \pi(\theta_0) = P(\text{reject } H_0 | H_0)$
- We compute the p-value p and draw the following conclusions:
  1. $p < \alpha \Rightarrow$ we reject H_0
  2. If $p \geq \alpha \Rightarrow$ not enough evidence to reject H_0

Hypothesis testing in R

- R offers several built-in functions that compute hypothesis tests
- Often, these functions also return a CI
- Some common simple tests:

Function	Test for...
`binom.test`	probability of success of a Bernoulli trial
`t.test(x, ...)`	mean of the normal distribution
`t.test(x, y, ...)`	mean difference for two normal distributions

- No need to specify `conf.level`, simply retrieve the p-value and compare it to your α!

Example

Let $X \sim N(\mu_X, \sigma_X)$ and $Y \sim N(\mu_Y, \sigma_Y)$. We want to test

$$H_0 : \mu_X = \mu_Y \text{ vs } H_1 : \mu_X \neq \mu_Y$$

```
set.seed(10) # et random seed for reproducibility
x = rnorm(100, mean = 3.3, sd = 1.5)
y = rnorm(100, mean = 3.8, sd = 1.2)
# how to compute a T test for the mean difference:
test = t.test(x = x, y = y, alternative = 'two.sided',
              var.equal = FALSE)
# p-value is stored in object$p.value
test$p.value
```

```
## [1] 0.001458384
```

- If we fix $\alpha = 0.05$, $p = 0.0014 < \alpha \Rightarrow$ we can reject $H_0 : \mu_X = \mu_Y$

Useful readings

- Numeric optimization: Sections 14.1-14.2 of Rizzo (2019)
- Nelder-Mead algorithm: Section 8.3 of Braun and Murdoch (2021)
- Maximum likelihood: Sections 13.4 and 14.3-14.4 of Rizzo (2019)