MTH3045: Statistical Computing

Dr. Ben Youngman
b.youngman@exeter.ac.uk
Laver 817; ext. 2314

24/3/2025

Week 11 lecture 1

Root finding

- Consider some function $f(x)$ for $x \in \mathbb{R}$ and wanting to find the value of x, $\tilde{x}$ say, such that $f(\tilde{x}) = 0$
- Sometimes we can analytically find $\tilde{x}$, but sometimes not
- We'll just consider the latter case where we'll need to find $\tilde{x}$ numerically, such as through some iterative process
- We won't go into the details of root-finding algorithms; instead we'll just look at `R`'s function `uniroot()`
- This is `R`'s go-to function for root finding
- This chapter will just demonstrate its use by example

Example: Root-finding in R I

- Use `uniroot()` in `R` to find the root of

$$f(x) = (x+3)(x-1)^2$$

  i.e. to find $\tilde{x}$, where $f(\tilde{x}) = 0$, given that $\tilde{x} \in [-4, 4/3]$

Example: Root-finding in R II

- We'll start by writing a function to evaluate $f()$, which we'll call `f`

```
f <- function(x) (x + 3) * (x - 1)^2
```

- Then we'll call `uniroot()`

```
uniroot(f, c(-4, 4.3))
```

```
## $root
## [1] -2.999997
##
## $f.root
## [1] 4.501378e-05
##
## $iter
## [1] 7
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 6.103516e-05
```

Example: Root-finding in R III

- We see that its output includes various details
- Most important are
  - `root`, its estimate of $\tilde{x}$, which is $\tilde{x} \simeq -2.9999972$
  - `f.root`, the value of $f()$ at the root, i.e. $f(\tilde{x})$, which is $f(\tilde{x}) \simeq 4.5013782 \times 10^{-5}$
- We note that $f(\tilde{x})$ is sufficiently close to zero that we should be confident that we've reached a root

Example: Root-finding in R IV

- *Remark*: We can ask `uniroot()` to extend the search range for the root through its argument `extendInt`
- Options are `'no'`, `'yes'`, `'downX'` and `'upX'`, which correspond to not extending the range (the default) or allowing it to be extended to allow upward and downward crossings, just downward or just upward, respectively
- (If we want to extend the search interval for the root, `extendInt = 'yes'` is usually the best option. Otherwise, we need to think about how $f()$ behaves at the roots, i.e. whether it's increasing or decreasing. See the help file for `uniroot()` for more details.)
- If we return to the above example and consider the search range $[-2, -1]$ instead, then by issuing

```
uniroot(f, c(-2, -1), extendInt = 'yes')$root
```

```
## [1] -2.999991
```

we do still find the root, even though it's outside of our specified range.

Challenges I

- Go to Challenges I of the week 11 lecture 1 challenges at
  https://byoungman.github.io/MTH3045/challenges

One-dimensional optimisation in R

- We'll only briefly look at how we can perform one-dimensional optimisation in R, which is through its `optimize()` function
- As described by its help file, `optimize()` uses '*a combination of golden section search and successive parabolic interpolation, and was designed for use with continuous functions*'
- We can instead use `optimize()` by calling `optim(..., method = 'Brent')`
  - the two are equivalent
  - the only reason I can see for using `optim(..., method = 'Brent')` over `optimize()` is that `optim()` is R's preferred numerical optimisation function, and hence users my benefit from familiarity with its output, as opposed to that of `optimize()`
- By default `optim()` uses the Nelder-Mead polytope method, which we'll cover in Section 5.4.7, which doesn't usually work well in one dimension

Example: Numerical maximum likelihood estimation I

- Consider a sample of data $y_1, \ldots, y_n$ as independent realisations from the $\text{Exp}(\lambda)$ distribution with pdf

$$f(y \mid \lambda) = \lambda \exp(-\lambda y) \qquad \text{for } y > 0$$

  where $\lambda > 0$ is an unknown parameter that we want to estimate
- Its mle is $1/\bar{y}$, where $\bar{y} = n^{-1} \sum_{i=1}^{n} y_i$
- Confirm this numerically in `R` using `optimize()` by assuming that the sample of data

$$0.4, 0.5, 0.8, 1.8, 2.1, 3.7, 8.2, 10.6, 11.6, 12.8$$

  are independent $\text{Exp}(\lambda)$ realisations

Example: Numerical maximum likelihood estimation II

- By default `optimize()` will find the minimum, so we want to write a function that will evaluate the exponential distribution's log-likelihood

$$\log f(\mathbf{y} \mid \lambda) = n \log \lambda - \lambda \sum_{i=1}^{n} y_i$$

  and then negate it
- We'll call this `negloglik(lambda, y)`

```
negloglik <- function(lambda, y) {
  # Function to evaluate Exp(lambda) neg. log likelihood
  # lambda is a scalar
  # y can be scalar or vector
  # returns a scalar
  -n * log(lambda) + lambda * sum(y)
}
```

Example: Numerical maximum likelihood estimation III

- We then pass this on to `optimize()` with our sample of data, which we'll call `y`

```
y <- c(0.4, 0.5, 0.8, 1.8, 2.1, 3.7, 8.2, 10.6, 11.6, 12.8)
optimize(negloglik, lower = .1, upper = 10, y = y)
```

```
## $minimum
## [1] 0.1904839
##
## $objective
## [1] 26.58228
```

- We see that `R`'s numerical maximum likelihood estimate of λ is 0.1904839, and the true value is $1/5.25 \simeq 0.1904762$
  - so the two agree to five decimal places
- *Remark 1*: We can ask `optimize()` to be more precise through its `tol` argument, which has default `tol = .Machine$double.eps^0.25`
  - smaller values of `tol` will give more accurate numerical estimates
- *Remark 2*: Calling `optimise()` is equivalent to calling `optimize()`, for those that don't like American spellings of English words.

Challenges I

- Go to Challenges I of the week 11 lecture 2 challenges at
  https://byoungman.github.io/MTH3045/challenges

Bibliographic notes

- By far the best resource for reach up on numerical optimisation is Nocedal and Wright (2006)
    - Chapter 3 covers Newton's method and line search
    - Chapter 6 covers quasi-Newton methods
    - Chapter 8 covers derivative-free optimisation, including the Nelder-Method in Section 9.5
- Optimisation is also covered in Monahan (2011, chap. 8) and in Wood (2015, sec. 5.1)
- Simulated annealing is covered in Press et al. (2007, sec. 10.12)
- Root-finding is covered in Monahan (2011, sec. 8.3) and Press et al. (2007, chap. 9)

Exam tips

1. If you're unsure whether your derivative function is correct, why not use finite-differencing to check it? Having generic finite-differencing functions to hand, such as `fd()` in the lecture notes, could be very helpful. Finite-differencing can also be used to approximate Hessian matrices.
2. Balance correcting code with moving on to the next question. If your code has a small error that's causing it to not run properly, then you may still pick up marks for a partially correct answer.

References

Monahan, John F. 2011. *Numerical Methods of Statistics*. 2nd ed. Cambridge University Press. https://doi.org/10.1017/CBO9780511977176.

Nocedal, J., and S. Wright. 2006. *Numerical Optimization*. 2nd ed. Springer Series in Operations Research and Financial Engineering. Springer New York. https://books.google.co.uk/books?id=VbHYoSyelFcC.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. 2007. *Numerical Recipes: The Art of Scientific Computing*. 3rd ed. Cambridge University Press. https://books.google.co.uk/books?id=1aAOdzK3FegC.

Wood, Simon N. 2015. *Core Statistics*. Institute of Mathematical Statistics Textbooks. Cambridge University Press. https://doi.org/10.1017/CBO9781107741973.