`optim()` in R – Exam Reference Sheet

Contents

1 Core Usage **1**
- 1.1 Function signature . . . 1
- 1.2 Returned object . . . 2

2 Objective function pattern **2**

3 Optimisation methods in `optim()` **3**
- 3.1 `"Nelder-Mead"` (default) . . . 3
- 3.2 `"BFGS"` . . . 3
- 3.3 `"CG"` (Conjugate Gradient) . . . 3
- 3.4 `"L-BFGS-B"` . . . 3
- 3.5 `"SANN"` (Simulated Annealing) . . . 4

4 Useful `control` options **4**

5 Debugging `optim()` **4**
- 5.1 Checklist . . . 4
- 5.2 Typical error messages and fixes . . . 5

6 Worked examples **5**
- 6.1 Example 1: Linear regression via BFGS . . . 5
- 6.2 Example 2: Same model, with constraint $b \in [0, 2]$ (L-BFGS-B) . . . 6
- 6.3 Example 3: Discrete optimisation with SANN (sketch) . . . 6

7 Slide example: "Why does it not work?" **7**

8 Rapid troubleshooting checklist **8**

1 Core Usage

1.1 Function signature

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

Key points:

- `par`: numeric vector of initial parameter values.
- `fn`: **objective function to minimise**. *First argument must be the parameter vector.* Must return a **single numeric value**.
- `gr`: gradient of `fn` (same first argument). Optional. If omitted, finite differences are used for gradient-based methods.

- `...`: extra arguments passed to `fn` and `gr` (typically data).
- `method`: optimisation algorithm (see Section 3).
- `lower`, `upper`: bounds (vectors same length as `par`) for `"L-BFGS-B"` and `"Brent"`.
- `control`: list of tuning options (Section 4).

1.2 Returned object

```
opt <- optim(...)
str(opt)
```

Important components:

- `opt$par`: parameter estimate at the found minimum.
- `opt$value`: objective value `fn(opt$par)`.
- `opt$counts`: number of function / gradient evaluations.
- `opt$convergence`: convergence code:
  - `0`: successful convergence.
  - `1`: iteration limit reached.
  - `10`: degenerate simplex (Nelder–Mead).
  - `51, 52`: L-BFGS-B warning / error.
- `opt$message`: text message (especially for L-BFGS-B).

Always check `opt$convergence` and `opt$message`.

2 Objective function pattern

General pattern for regression-style problems:

```
crit <- function(par, x, y) {
  # unpack parameters
  a <- par[1]
  b <- par[2]

  # model prediction
  yhat <- a + b * x

  # objective: sum of squared residuals
  residuals <- y - yhat
  sum(residuals^2)
}
```

Rules:

1. First argument = parameter vector (`par` / `theta`).
2. Return a *single* numeric value (scalar).
3. For maximisation (e.g. log-likelihood), minimise the negative:

```
crit <- function(par, data) {
  - logLik(par, data)  # minus sign!
}
```

or use `control = list(fnscale = -1)`.

3 Optimisation methods in `optim()`

3.1 `"Nelder-Mead"` (default)

- Derivative-free simplex method.
- Uses only function values, robust to noise / non-smoothness.
- Slower and less reliable in high dimensions, can stagnate (code 10).
- Good first try if gradients are hard to compute.

3.2 `"BFGS"`

- Quasi-Newton method; builds Hessian approximation.
- Needs gradient (supplied or finite-difference).
- Typically fast and accurate for smooth, unconstrained problems.
- No bounds; use when parameters are effectively unbounded.

Example pattern:

```
optim(par = c(a = 0, b = 0),
      fn  = crit,
      gr  = grad,  # optional
      x   = x, y = y,
      method = "BFGS")
```

3.3 `"CG"` (Conjugate Gradient)

- Uses gradient only, no Hessian stored (low memory).
- Generally slower / more fragile than BFGS, but lighter.
- Unconstrained only. Same calling pattern as BFGS.

3.4 `"L-BFGS-B"`

- Limited-memory BFGS with box constraints.
- Handles bounds: `lower`, `upper` (vectors, length = `length(par)`).
- Initial `par` must satisfy lower $\leq$ par $\leq$ upper.
- Requires *finite* `fn` values inside box; `NA` / `Inf` cause errors.
- Good for parameters like variances, probabilities, rates, etc. with natural bounds.

Example:

```
optim(par   = c(a = 0,  b = 1),
      fn    = crit,
      gr    = grad,
      x = x, y = y,
      method = "L-BFGS-B",
      lower  = c(-Inf, 0),   # a unbounded, b >= 0
      upper  = c( Inf,  2))  # b <= 2
```

3.5 "SANN" (Simulated Annealing)

- Stochastic global search (simulated annealing).
- Uses only function values.
- No convergence test; run for fixed number of evaluations.
- Very slow; mainly for rough, multimodal surfaces.
- Often used to get near good region, then refine with BFGS.

Skeleton:

```
optim(par = par0,
      fn  = crit,
      method = "SANN",
      control = list(maxit = 10000, temp = 10, trace = 1))
```

4 Useful control options

- maxit: maximum iterations (BFGS/CG/L-BFGS-B) or function evaluations (SANN).
- trace: if > 0, prints progress info (debugging).
- fnscale:
  - default $= 1$; if set to -1, minimising fn is equivalent to maximising it.
- parscale: scaling factors for parameters; helpful if parameters are on very different scales.
- REPORT: how often to print progress when trace > 0.

Example:

```
optim(par = par0, fn = crit, x = x, y = y,
      method = "BFGS",
      control = list(maxit = 1000, trace = 1))
```

5 Debugging optim()

5.1 Checklist

1. **Objective signature:** first argument is parameter vector; returns scalar.
2. **Test fn:** run fn(par0, ...) manually. It must be numeric and finite.
3. **Bounds:** when using L-BFGS-B, check

   ```
   all(par0 >= lower & par0 <= upper)
   ```

4. **Gradient:** if supplying gr, confirm dimensions and correctness. If in doubt, remove gr and let optim() finite-difference it.
5. **Inspect output:** always read opt$convergence and opt$message.

5.2 Typical error messages and fixes

`"L-BFGS-B needs finite values of 'fn'"`

- Your function returned `NA`, `NaN` or `Inf`.
- Check:

```
fn(par0, ...)  # is it finite?
```

- Either:
  - Restrict bounds slightly away from problematic value, or
  - Modify `fn` to return a large finite penalty instead of `Inf`.

Convergence code `1`: iteration limit reached

- Try increasing `maxit` or improving starting values.
- Consider a different method (e.g. BFGS instead of Nelder–Mead).

Convergence code `10`: degenerate simplex (Nelder–Mead)

- Simplex collapsed; algorithm stagnated.
- Try different initial `par`, rescaling parameters, or a gradient-based method.

Weird parameter at extreme value

- Often indicates wrong objective function (e.g. summing residuals instead of squared residuals).
- Verify that `fn` truly matches the quantity you intend to minimise.

6 Worked examples

6.1 Example 1: Linear regression via BFGS

Goal: estimate intercept a and slope b in

$$y_i = a + bx_i + \varepsilon_i$$

by minimising sum of squared residuals.

```
set.seed(1)
x <- 1:20
y <- 1.5 + 2.5 * x + rnorm(length(x), sd = 5)

# objective: sum of squared errors
crit <- function(par, x, y) {
  a <- par[1]; b <- par[2]
  yhat <- a + b * x
  resid <- y - yhat
  sum(resid^2)
}

# analytic gradient (optional but good practice)
grad <- function(par, x, y) {
  a <- par[1]; b <- par[2]
  yhat <- a + b * x
```

```
  resid <- y - yhat
  d_da <- -2 * sum(resid)       # derivative wrt a
  d_db <- -2 * sum(resid * x)   # derivative wrt b
  c(d_da, d_db)
}

opt <- optim(par = c(a = 0, b = 0),
             fn  = crit,
             gr  = grad,
             x = x, y = y,
             method = "BFGS")

opt$par       # estimated (a, b)
opt$value     # SSE at optimum
opt$convergence
```

Key points:

- Objective returns scalar SSE.
- Gradient matches the objective.
- **No bounds**, so we use BFGS.

6.2 Example 2: Same model, with constraint $b \in [0, 2]$ (L-BFGS-B)

```
optB <- optim(par = c(a = 0, b = 0),
              fn  = crit,
              gr  = grad,
              x = x, y = y,
              method = "L-BFGS-B",
              lower = c(-Inf, 0),
              upper = c( Inf, 2))

optB$par       # (a, b) with b constrained to [0, 2]
optB$convergence
```

If the unconstrained optimum has $b > 2$, the constrained optimum will sit at $b = 2$. Check whether any component of `optB$par` equals its bound to see if a constraint is active.

6.3 Example 3: Discrete optimisation with SANN (sketch)

When the surface is rough / multimodal:

```
crit <- function(par, data) {
  # e.g. negative log-likelihood with many local optima
}

optS <- optim(par = par0,
              fn  = crit,
              data = data,
              method = "SANN",
              control = list(maxit = 20000, temp = 10, trace = 1))

# Optionally refine with BFGS
optRef <- optim(par = optS$par,
                fn  = crit,
                data = data,
                method = "BFGS")
```

7 Slide example: "Why does it not work?"

From the lecture slide:

```
par(mfrow=c(1,2), font.lab=2, font.axis=2)
model <- function(x, theta){
  return(theta * x)
}

crit <- function(theta, x, y){
  # must have theta as 1st parameter and return a single value...
  return( sum( y - model(x, theta) ) )
}

thbar <- 1.8
x <- rep(c(1,3,7,8), len = 100)
y <- model(x, thbar) + rnorm(x)

plot(x, y, pch=20, cex=1.5, main="optim example 1: data")
points(x, model(x, thbar), col="green", pch=20, cex=2)

optim.out <- optim(par=c(1), fn=crit, x=x, y=y,
                   method="L-BFGS-B", lower=c(0.01), upper=c(3.05))
```

Issue 1: wrong objective (sum of residuals)

The criterion is

$$\text{crit}(\theta) = \sum_i (y_i - \theta x_i).$$

This is *not* a sensible loss:

- Residuals can cancel (positive vs. negative), so the sum can be small even if individual errors are large.
- For these simulated data, the sum is approximately linear in θ, so the minimum lies at one of the *bounds* (0.01 or 3.05), not near the true value 1.8.
- Consequently, `optim()` drives θ to a boundary instead of a meaningful estimate.

Fix

Use **sum of squared residuals**:

```
crit <- function(theta, x, y) {
  resid <- y - model(x, theta)
  sum(resid^2)
}
```

Now the function has a unique minimum near the true $\theta = 1.8$, and methods like `"BFGS"` or `"L-BFGS-B"` will find a sensible solution.

Other potential pitfalls in this pattern

- If `crit()` returned a vector, or if the first argument wasn't `theta`, `optim()` would either error or behave unexpectedly.
- If L-BFGS-B is used and the objective becomes `Inf` at a boundary, the call could fail with "L-BFGS-B needs finite values of 'fn"'.

8 Rapid troubleshooting checklist

When `optim()` "does not work", check in this order:

1. **Signature**: Is parameter vector first argument in `fn` and `gr`? Does `fn()` return a scalar?
2. **Finite values**: Is `fn(par0, ...)` finite? (Especially for L-BFGS-B.)
3. **Right objective**: Are you minimising the correct quantity (e.g. SSE, negative log-likelihood)?
4. **Bounds**: For L-BFGS-B, are `par0` inside bounds, and is the function well-defined throughout the box?
5. **Gradient**: If supplying `gr`, is it correct? If unsure, temporarily drop `gr`.
6. **Method**: Try a different method (e.g. BFGS instead of Nelder–Mead, or vice versa).
7. **Control**: Increase `maxit`, turn on `trace`, and inspect `opt$message`.