MTH3045: Statistical Computing

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

26/3/2024

Week 11 lecture 1

Extra challenge

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

Nelder-Mead polytope method I

- So far we have considered derivative-based optimisation algorithms
- When we cannot analytically calculate derivatives, we can use finite-difference approximations
- However, sometimes we may want to find the minimum point on a surface for a surface that is not particularly smooth
- Then derivative information may not be helpful
- Instead, we might want an algorithm that explores a surfaces differently
- The Nelder-Mead polytope algorithm is one such approach (Nelder and Mead (1965))
- In fact, it is `R`'s default if we use `optim()`, i.e. if we don't supply `method = '...'`

Nelder-Mead polytope method II

- For the Nelder-Mead algorithm, consider $\boldsymbol{\theta} \in \mathbb{R}^p$. The algorithm starts with $p+1$ test points, $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{p+1}$, which we call *vertices*, and then proceeds as follows. . .

1. Order $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{p+1}$ so that
$$f(\boldsymbol{\theta}_1) \leq f(\boldsymbol{\theta}_2) \leq \ldots \leq f(\boldsymbol{\theta}_{p+1})$$
and check whether the termination criteria have been met (which are given later). If not, proceed to Step 2.
2. Calculate the centroid, $\boldsymbol{\theta}_o$, of $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_p$, i.e. omitting $\boldsymbol{\theta}_{p+1}$, because $\boldsymbol{\theta}_{p+1}$ is the worst vertex.
3. Reflection. Compute the reflected point $\boldsymbol{\theta}_r = \boldsymbol{\theta}_o + \alpha(\boldsymbol{\theta}_o - \boldsymbol{\theta}_{p+1})$. If $f(\boldsymbol{\theta}_1) \leq f(\boldsymbol{\theta}_r) < f(\boldsymbol{\theta}_p)$, replace $\boldsymbol{\theta}_{p+1}$ with $\boldsymbol{\theta}_r$ and return to Step 1. Otherwise, proceed to Step 4.

Nelder-Mead polytope method III

4. Expansion. If $f(\boldsymbol{\theta}_r) < f(\boldsymbol{\theta}_1)$, i.e. is the best point so far, compute the expanded point $\boldsymbol{\theta}_e = \boldsymbol{\theta}_o + \gamma(\boldsymbol{\theta}_r - \boldsymbol{\theta}_o)$ for $\gamma > 1$. If $f(\boldsymbol{\theta}_e) < f(\boldsymbol{\theta}_r)$, replace $\boldsymbol{\theta}_{p+1}$ with $\boldsymbol{\theta}_e$ and return to Step 1. Otherwise, replace $\boldsymbol{\theta}_{p+1}$ with $\boldsymbol{\theta}_r$ and return to Step 1.
5. Contraction. Now $f(\boldsymbol{\theta}_r) \geq f(\boldsymbol{\theta}_p)$. Compute the contracted point $\boldsymbol{\theta}_c = \boldsymbol{\theta}_o + \rho(\boldsymbol{\theta}_{p+1} - \boldsymbol{\theta}_o)$ for $0 < \rho \leq 0.5$. If $f(\boldsymbol{\theta}_c) < f(\boldsymbol{\theta}_{p+1})$ then replace $\boldsymbol{\theta}_{p+1}$ with $\boldsymbol{\theta}_c$ and return to Step 1. Otherwise proceed to Step 6.
6. Shrink. For $i = 2, \ldots, p+1$ set $\boldsymbol{\theta}_i = \boldsymbol{\theta}_1 + \sigma(\boldsymbol{\theta}_i - \boldsymbol{\theta}_1)$ and return to Step 1.

Nelder-Mead polytope method IV

- Often the values $\alpha = 1$, $\gamma = 2$, $\rho = 0.5$ and $\sigma = 0.5$ are used
- In Step 1, termination is defined in terms of tolerances
- The main criterion is for $f(\boldsymbol{\theta}_{p+1}) - f(\boldsymbol{\theta}_1)$ to be sufficiently small, so that $f(\boldsymbol{\theta}_i)$ for $i = 1, \ldots, p+1$ are close together for all θ_i
  - hence we are hoping that *all* the $\boldsymbol{\theta}_i$ values are in the region of the true minimum
- We won't code the Nelder-Mead algorithm in `R`; instead we'll just use `optim()` and look what it does, by requesting a trace with `control = list(trace = TRUE)`

Example: Weibull maximum likelihood: Nelder-Mead I

- Use the Nelder-Mead method and `R`'s `optim()` function to find the maximum likelihood estimates of the Weibull distribution for the data of Example 5.5
- We've got everything we need for this example, i.e. the data, which we stored earlier as `y0`, and a function to evaluate the Weibull distribution's log-likelihood, `weib_d0()`
- So we just pass these to `optim()`, ensuring that `mult = -1`, so that we find the negative log-likelihood's minimum

Example: Weibull maximum likelihood: Nelder-Mead II

```r
fit_nelder <- optim(c(1.6, .6), weib_d0, y = y0, mult = -1,
                    control = list(trace = TRUE))
```

```
##   Nelder-Mead direct search function minimizer
## function value for initial parameters = 55.509247
##   Scaled convergence tolerance is 8.27152e-07
## Stepsize computed as 0.159049
## BUILD              3 60.369866 55.293827
## LO-REDUCTION       5 55.509247 55.045750
## REFLECTION         7 55.293827 55.038084
## HI-REDUCTION       9 55.045750 54.993703
## REFLECTION        11 55.038084 54.962562
## HI-REDUCTION      13 54.993703 54.961598
## ** quite a few lines suppressed **
## HI-REDUCTION      41 54.953162 54.953159
## HI-REDUCTION      43 54.953160 54.953159
## Exiting from Nelder Mead minimizer
##     45 function evaluations used
```

- The above output is telling us what `optim()` is doing as it's going along
- Specifically, `LO-REDUCTION` corresponds to a contraction, `HI-REDUCTION` to an expansion and `REFLECTION` to a reflection
- On this occasion, there was no need to shrink the simplex, which is identified as `SHRINK`

Example: Weibull maximum likelihood: Nelder-Mead III

```
fit_nelder
```

```
## $par
## [1] 1.8900970 0.5374706
##
## $value
## [1] 54.95316
##
## $counts
## function gradient
##       43       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

- What `optim()` returns is essentially the same as we saw before for the BFGS method, i.e. roughly the same parameter estimates and function minimum, except that now there are no gradient evaluations

Example: Weibull maximum likelihood: Nelder-Mead IV

- *Remark*: The Nelder-Mead method is usually great if you're quickly looking to find a function's minimum without its gradient
  - provided it's a relatively low-dimensional function, and the function is fairly quick to evaluate
  - this is probably why it's `R`'s default
- It can be very slow for high-dimensional optimisation problems, and is also typically less reliable than gradient-based methods, except for strangely-behaved surfaces

Challenges I

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

Week 11 lecture 2

Global optimisation

- So far we have considered optimisation algorithms that usually home in on local minima, given starting points
- For multimodal surfaces, this can be undesirable
- Here we consider approaches to **global optimisation**, which are designed to find the *overall* minimum

Stochastic optimisation

- The optimisation algorithms that have been introduced so far have been deterministic
    - given a set of starting values, they will always return the same final value
- Stochastic optimisation algorithms go from one point to another probabilistically, and so will go through different sets of parameters
- We should expect them to converge to the same final value, though

Simulated annealing I

- Simulated annealing gains its name from the physical annealing process
    - a heat treatment that alters the physical and sometimes chemical properties of a material to increase its ductility and reduce its hardness, making it more workable
    - you can of course read more about it on Wikipedia[1]
- Consider a current parameter value $\boldsymbol{\theta}$ and some function we seek to minimise $f()$
- For example, $f()$ might be a negated log-likelihood
- The key to simulated annealing is that a *random* point, $\boldsymbol{\theta}^*$, is proposed
    - $\boldsymbol{\theta}^*$ is drawn from some proposal density $q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta})$, which depends on the current value $\boldsymbol{\theta}$
- The proposal density $q()$ is chosen to be symmetric, but otherwise its choice is arbitrary

[1] https://en.wikipedia.org/wiki/Annealing_(materials_science)

Simulated annealing II

- The main aspect of simulated annealing is to work with the function

$$\pi_T(\boldsymbol{\theta}) = \exp\{-f(\boldsymbol{\theta})/T\}$$

  for some *temperature* T
- We note that as $T \searrow 0$, $\pi_T(\boldsymbol{\theta}) \to \exp\{-f(\boldsymbol{\theta})\}$
- Put algorithmically, simulated annealing works as follows

1. Propose $\boldsymbol{\theta}^*$ from $q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta})$.
2. Generate $U \sim \text{Uniform}[0,1]$.
3. Calculate

$$\begin{aligned}\alpha(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}) &= \min\left\{\frac{\exp\left[-f(\boldsymbol{\theta}^*)/T\right]}{\exp\left[-f(\boldsymbol{\theta})/T\right]}, 1\right\} \\ &= \min\left(\exp\left\{-\left[f(\boldsymbol{\theta}^*) - f(\boldsymbol{\theta})\right]/T\right\}, 1\right).\end{aligned}$$

4. Accept $\boldsymbol{\theta}^*$ if $\alpha(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}) > U$; otherwise keep $\boldsymbol{\theta}$.
5. Decrease T.

Simulated annealing III

- It is worth noting that steps 1 to 4 implement a special case of the Metropolis-Hastings algorithm for symmetric $q()$
- This algorithm is a heavily used in statistics, especially Bayesian statistics, to sample from posterior densities that do not have or have unwieldy closed forms, typically as part of Markov chain Monte Carlo (MCMC) sampling.
- *Remark*: `R`'s default is to use a Gaussian distribution for $q()$ and the temperature at iteration i, T_i, is chosen according to $T_i = T_1 / \log\{t_{\text{max}} \lfloor (i-1)/t_{\text{max}} \rfloor + \exp(1)\}$ with $T_1 = t_{\text{max}} = 10$ the default values

Example: Weibull maximum likelihood: Simulated annealing I

- Write a function to update the simulated annealing temperature according to R's rule and another function to generate Gaussian proposals with standard deviation 0.1
- Then use simulated annealing to repeat Example 5.5 with $N = 1000$ iterations and plot λ_i and k_i at each iteration using initial temperatures of $T_1 = 10$, 1 and 0.1
- The following function, `update_T()`, updates the temperature according to R's rule

```r
update_T <- function(i, t0 = 10, t1 = 10) {
  # Function to update simulated annealing temperature
  # i is an integer giving the current iteration
  # t0 is a scalar giving the initial temperature
  # t1 is a integer giving how many iterations of each temperature to use
  # returns a scalar
  t0 / log(((i - 1) %/% t1) * t1 + exp(1))
}
```

Example: Weibull maximum likelihood: Simulated annealing II

- Then the following function, `q_fn()`, generates Gaussian proposals with standard deviation 0.1

```r
q_fn <- function(x) {
  # Function to generate Gaussian proposals with standard deviation 0.1
  # x is the Gaussian mean as either a scalar or vector
  # returns a scalar or vector, as x
  rnorm(length(x), x, .1)
}
```

- The following function can perform simulated annealing. You won't be asked to write such a function for MTH3045, but it's illuminating to see how such a function can be written.

```r
sa <- function(p0, f, N, q, T1, ...) {
# Function to perform simulated annealing
# p0 p-vector of initial parameters
# f() function to be minimised
# N number of iterations
# q proposal function
# T1 initial temperature
# ... arguments to pass to f()
# returns p x N matrix of parameter estimates at each iteration
# commands suppressed
```

Example: Weibull maximum likelihood: Simulated annealing III

```r
sa <- function(p0, f, N, q, T1, ...) {
# comments suppressed
out <- matrix(0, N, length(p0)) # matrix to store estimates at each iteration
out[1, ] <- p0 # fill first row with initial parameter estimates
for (i in 2:N) { # N iterations
  T <- update_T(i, T1) # update temperature
  U <- runif(1) # generate U
  out[i, ] <- out[i - 1,] # carry over last parameter estimate, by default
  proposal <- q(out[i - 1,]) # generate proposal
  if (min(proposal) >= 0) { # ensure proposal valid
    f0 <- f(out[i - 1, ], ...) # evaluate f for current theta
    f1 <- f(proposal, ...) # evaluate f for proposed theta
    alpha <- min(exp(- (f1 - f0) / T), 1) # calculate M-H ratio
    if (alpha >= U) # accept if ratio sufficiently high
      out[i, ] <- proposal # swap last with proposal
  }
}
out # return all parameter estimates
}
```

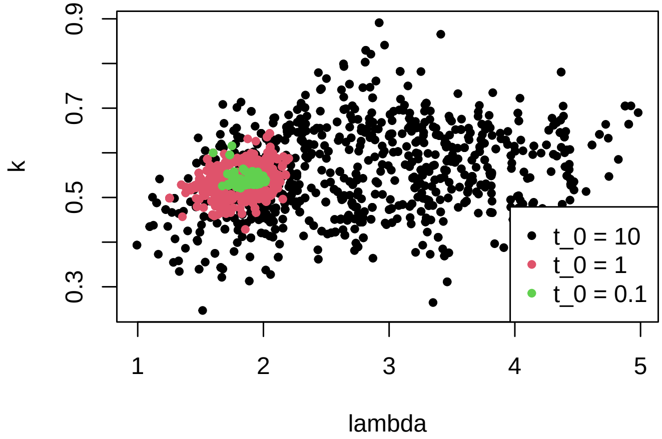
Example: Weibull maximum likelihood: Simulated annealing IV

- Then we'll specify out initial temperatures as `T_vals`, and loop over these with `sa()`...

```r
# values to use for initial temperature
T_vals <- c(10, 1, .1)
# loop over values, and plot
for (j in 1:length(T_vals)) {
  T1 <- T_vals[j]
  sa_result <- sa(c(1.6, .6), weib_d0, 1e3, q_fn, T1, y = y0, mult = -1)
  if (j == 1) {
    plot(sa_result, col = j, pch = 20, xlab = 'lambda', ylab = 'k')
  } else {
    points(sa_result, col = j, pch = 20)
  }
}
legend('bottomright', pch = 20, col = 1:length(T_vals),
       legend = paste("t_0 =", T_vals), bg = 'white')
```

Example: Weibull maximum likelihood: Simulated annealing V

- ... and then plot the resulting parameter estimates for each temperature and each iteration



- We see that lower initial temperatures bring smaller clouds of parameter estimates.

Simulated annealing in `R` I

- Simulated annealing is built in to `R`'s `optim()` function and requires `method = 'SANN'`
- **Example**: Repeat Example 5.10 using `R`'s `optim()` function to perform simulated annealing
- Report the best value of the objective function every 100 iterations

Simulated annealing in R II

- We'll use the following call

```r
optim(c(1.6, .6), weib_d0, y = y0, mult = -1, method = 'SANN',
      control = list(trace = 1, REPORT = 10, maxit = 1e3))
```

```
## sann objective function values
## initial       value 55.677933
## iter      100 value 54.978464
## iter      200 value 54.955248
## iter      300 value 54.955248
## iter      400 value 54.955248
## iter      500 value 54.955248
## iter      600 value 54.955248
## iter      700 value 54.955248
## iter      800 value 54.955248
## iter      900 value 54.955248
## iter      999 value 54.955248
## final         value 54.955248
## sann stopped after 999 iterations
```

```
## $par
## [1] 1.8967831 0.5422952
##
## $value
## [1] 54.95525
##
## $counts
```

Simulated annealing in R III

```r
optim(c(1.6, .6), weib_d0, y = y0, mult = -1, method = 'SANN',
      control = list(maxit = 1e3))[1]
```

```
## $par
## [1] 1.8628985 0.5384591
```

- The parameter estimates are some way off those from earlier
- With more iterations, simulated annealing would gradually get closer to the true minimum
- Note that the `control$REPORT` argument specifies the frequency in terms of how often the temperature changes; so R reports the status of the optimiser each (`control$tmax * control$REPORT`)th iteration, noting that `control$tmax` defaults to 10.
- *Remark*: It's sometimes a good tactic to use simulated annealing to get close to the minimum, and then to employ one of the previously discussed deterministic optimisation methods to get a more accurate estimate
- This is especially useful if we're unsure whether we're starting off with sensible initial parameter estimates

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. Remember your differentiation rules. If you're asked to write a function to evaluate derivatives, don't lose marks because you've forgotten a differentiation rule.
2. Following on from above, 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.
3. 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.

Nelder, J. A., and R. Mead. 1965. "A Simplex Method for Function Minimization." *The Computer Journal* 7 (4): 308–13. https://doi.org/10.1093/comjnl/7.4.308.

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.