MTH3045: Statistical Computing

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

17/3/2025

Week 10 lecture 1

Example: Weibull maximum likelihood: Newton's method I

- The Weibull distribution is sometimes used to model wind speeds
- For a wind speed y its pdf is given by

$$f(y \mid \lambda, k) = \frac{k}{\lambda}\left(\frac{y}{\lambda}\right)^{k-1} e^{-(y/\lambda)^k} \quad \text{for } y > 0$$

  and where $\lambda, k > 0$ are its parameters
- (Note that this is the scale parameterisation of the Weibull distribution)
- For observed wind speeds $y_1, \ldots, y_n$ its corresponding log-likelihood is therefore

$$\log f(\mathbf{y} \mid \lambda, k) = n \log k - nk \log \lambda + (k-1)\sum_{i=1}^{n} \log y_i - \sum_{i=1}^{n}\left(\frac{y_i}{\lambda}\right)^k$$

Example: Weibull maximum likelihood: Newton's method II

- To implement Newton's method, we need to find the first and second derivatives of $\log f(\mathbf{y} \mid \lambda, k)$ w.r.t. λ and k
- The first derivatives are

$$\begin{pmatrix} \dfrac{\partial \log f(\mathbf{y} \mid \lambda, k)}{\partial \lambda} \\ \dfrac{\partial \log f(\mathbf{y} \mid \lambda, k)}{\partial k} \end{pmatrix} = \begin{pmatrix} \dfrac{k}{\lambda}\left(\sum_{i=1}^{n}\left(\dfrac{y_i}{\lambda}\right)^k - n\right) \\ \dfrac{n}{k} - n\log\lambda + \sum_{i=1}^{n}\log y_i - \sum_{i=1}^{n}\left[\left(\dfrac{y_i}{\lambda}\right)^k \log\left(\dfrac{y_i}{\lambda}\right)\right] \end{pmatrix}$$

Example: Weibull maximum likelihood: Newton's method III

- ... and the second derivatives are stored in the matrix

$$\begin{pmatrix} \dfrac{\partial^2 \log f(\mathbf{y} \mid \lambda, k)}{\partial \lambda^2} & \dfrac{\partial^2 \log f(\mathbf{y} \mid \lambda, k)}{\partial \lambda \partial k} \\ \dfrac{\partial^2 \log f(\mathbf{y} \mid \lambda, k)}{\partial k \partial \lambda} & \dfrac{\partial^2 \log f(\mathbf{y} \mid \lambda, k)}{\partial k^2} \end{pmatrix}$$

where

$$\begin{aligned} \frac{\partial^2 \log f(\mathbf{y} \mid \lambda, k)}{\partial \lambda^2} &= \frac{k}{\lambda^2} \left(n - (1+k) \sum_{i=1}^{n} \left(\frac{y_i}{\lambda} \right)^k \right) \\ \frac{\partial^2 \log f(\mathbf{y} \mid \lambda, k)}{\partial \lambda \partial k} &= \frac{\partial^2 \log f(\mathbf{y} \mid \lambda, k)}{\partial k \partial \lambda} \\ &= \frac{1}{\lambda} \left(\sum_{i=1}^{n} \left(\frac{y_i}{\lambda} \right)^k - n + k \sum_{i=1}^{n} \left[\left(\frac{y_i}{\lambda} \right)^k \log \left(\frac{y_i}{\lambda} \right) \right] \right) \\ \frac{\partial^2 \log f(\mathbf{y} \mid \lambda, k)}{\partial k^2} &= -\frac{n}{k^2} - \sum_{i=1}^{n} \left(\frac{y_i}{\lambda} \right)^k \left[\log \left(\frac{y_i}{\lambda} \right) \right]^2 \end{aligned}$$

Example: Weibull maximum likelihood: Newton's method IV

- Consider the following wind speed measurements (in m/s) for the month of March

```r
y0 <- c(3.52, 1.95, 0.62, 0.02, 5.13, 0.02, 0.01, 0.34, 0.43, 15.5,
        4.99, 6.01, 0.28, 1.83, 0.14, 0.97, 0.22, 0.02, 1.87, 0.13, 0.01,
        4.81, 0.37, 8.61, 3.48, 1.81, 37.21, 1.85, 0.04, 2.32, 1.06)
```

- Use five iterations of Newton's method to estimate $\hat{\lambda}$ and $\hat{k}$, assuming the above wind speeds are independent from one day to the next and follow a Weibull distribution

Weibull first derivative

```r
weib_d1 <- function(pars, y, mult = 1) {
  # Function to evaluate first derivative of Weibull log-likelihood
  # pars is a vector
  # y can be scalar or vector
  # mult is a scalar defaulting to 1; so -1 returns neg. gradient
  # returns a vector
  n <- length(y)
  z1 <- y / pars[1]
  z2 <- z1^pars[2]
  out <- numeric(2)
  out[1] <- (sum(z2) - n) * pars[2] / pars[1] # derivative w.r.t. lambda
  out[2] <- n * (1 / pars[2] - log(pars[1])) +
    sum(log(y)) - sum(z2 * log(z1)) # w.r.t k
  mult * out
}
```

Weibull second derivative

```r
weib_d2 <- function(pars, y, mult = 1) {
  # Function to evaluate second derivative of Weibull log-likelihood
  # pars is a vector
  # y can be scalar or vector
  # mult is a scalar defaulting to 1; so -1 returns neg. Hessian
  # returns a matrix
  n <- length(y)
  z1 <- y / pars[1]
  z2 <- z1^pars[2]
  z3 <- sum(z2)
  z4 <- log(z1)
  out <- matrix(0, 2, 2)
  out[1, 1] <- (pars[2] / pars[1]^2) * (n - (1 + pars[2]) * z3) # w.r.t. (lambd
  out[1, 2] <- out[2, 1] <- (1 / pars[1]) * ((z3 - n) +
    pars[2] * sum(z2 * z4)) # w.r.t. (lambda, k)
  out[2, 2] <- -n/pars[2]^2 - sum(z2 * z4^2) # w.r.t. k^2
  mult * out
}
```

Newton's method in R I

- We've just put together some simple code that implemented Newton's method
- There are various ways of performing variants of Newton's method in R, but not Newton's method itself
- So here we'll look at `nlminb()`, which is described as 'Unconstrained and box-constrained optimization using PORT routines'
- We can use our functions `weib_d1` and `weib_d2` from earlier for the first and second derivatives of the negative log-likelihood w.r.t. λ and k

Newton's method in R II

- We now just need a function to evaluate the negative log-likelihood itself
- We'll call this `weib_d0`
- Note, though, that it's important to ensure that invalid parameters, i.e. $\lambda \leq 0$ and/or $k \leq 0$, are avoided
- Below we achieve this by setting the log-likelihood to be extremely low (-10^8) for such parts of parameter space

```r
weib_d0 <- function(pars, y, mult = 1) {
  # Function to evaluate Weibull log-likelihood
  # pars is a vector
  # y can be scalar or vector
  # mult is a scalar defaulting to 1; so -1 returns neg. log likelihood
  # returns a scalar
  n <- length(y)
  if (min(pars) <= 0) {
    out <- -1e8
  } else {
    out <- n * (log(pars[2]) - pars[2] * log(pars[1])) +
    (pars[2] - 1) * sum(log(y)) - sum((y / pars[1])^pars[2])
  }
  mult * out
}
```

Newton's method in R II

```r
nlminb(c(1.6, .6), weib_d0, weib_d1, weib_d2, y = y0, mult = -1)
```

```
## $par
## [1] 1.8900689 0.5375279
##
## $objective
## [1] 54.95316
##
## $convergence
## [1] 0
##
## $iterations
## [1] 5
##
## $evaluations
## function gradient
##        6        6
##
## $message
## [1] "relative convergence (4)"
```

Newton's method in R III

- We see that nlminb's output is a list comprising…
    - par, the parameter estimates
    - objective, the final value of the negative log-likelihood
    - convergence, whether the algorithm has converged (where 0 indicates successful convergence)
    - iterations, the number iterations before convergence was achieved
    - evaluations, how many times the function and gradient were evaluated
    - message provides further details on the type of convergence achieved

Challenges I

- Go to Challenges I of the week 10 lecture 1 challenges at https://byoungman.github.io/MTH3045/challenges

Quasi-Newton methods I

- Between Newton's method and steepest descent lie **quasi-Newton** methods
- These essentially employ Newton's method, but with some approximation to the Hessian matrix
- Instead of the Newton's method search direction

$$\mathbf{p}_i = -\left[\nabla^2 f(\boldsymbol{\theta}_i)\right]^{-1} \nabla f(\boldsymbol{\theta}_i)$$

  consider the search direction

$$\tilde{\mathbf{p}}_i = -\mathbf{H}_i^{-1} \nabla f(\boldsymbol{\theta}_i)$$

  where $\mathbf{H}_i$ is an approximation to the Hessian matrix $\nabla^2 f(\boldsymbol{\theta}_i)$ at the ith iteration

Quasi-Newton methods II

- We might, for example, want to avoid explicitly calculating $\nabla^2 f(\boldsymbol{\theta}_i)$ because it's a matrix that's sufficiently more difficult to calculate than the gradient (e.g. mathematically, or just in terms of time)
    - so that using an approximation to the Hessian matrix (provided it is an adequate approximation) gives a more efficient approach to optimisation than using the Hessian matrix itself
- We should typically expect quasi-Newton methods to converge slower than Newton's method, but provided that convergence isn't too much slower or less reliable, then we may prefer this over analytically forming the Hessian matrix

BFGS (Broyden–Fletcher–Goldfarb–Shanno) I

- In MTH3045 we shall consider the so-called **BFGS** (shorthand for Broyden–Fletcher–Goldfarb–Shanno) quasi-Newton algorithm
- Put simply, at iteration i, the BFGS algorithm assumes that

$$\nabla^2 f(\boldsymbol{\theta}_i) \simeq \mathbf{H}_i = \mathbf{H}_{i-1} + \frac{\mathbf{y}_i \mathbf{y}_i^{\mathrm{T}}}{\mathbf{y}_i^{\mathrm{T}} \mathbf{s}_i} - \frac{(\mathbf{H}_{i-1})^{-1} \mathbf{s}_i \mathbf{s}_i^{\mathrm{T}} (\mathbf{H}_{i-1})^{-T}}{\mathbf{s}_i^{\mathrm{T}} (\mathbf{H}_{i-1})^{-1} \mathbf{s}_i},$$

  where $\mathbf{s}_i = \boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}$ and $\mathbf{y}_i = \nabla f(\boldsymbol{\theta}_i) - \nabla f(\boldsymbol{\theta}_{i-1})$
- Hence the BFGS algorithm uses differences in the gradients of successive iterations to approximate the Hessian matrix
- We now note that we use $\mathbf{H}_i$ in $\mathbf{p}_i = -[\mathbf{H}_i]^{-1} \nabla f(\boldsymbol{\theta}_i)$
- We can avoid solving this system of linear equations by instead directly obtaining $[\mathbf{H}_i]^{-1}$ through

$$[\mathbf{H}_i]^{-1} = \left(\mathbf{I}_p - \frac{\mathbf{s}_i \mathbf{y}_i^{\mathsf{T}}}{\mathbf{y}_i^{\mathsf{T}} \mathbf{s}_i} \right) [\mathbf{H}_{i-1}]^{-1} \left(\mathbf{I}_p - \frac{\mathbf{y}_i \mathbf{s}_i^{\mathsf{T}}}{\mathbf{s}_i^{\mathsf{T}} \mathbf{y}_i} \right) + \frac{\mathbf{s}_i \mathbf{s}_i^{\mathsf{T}}}{\mathbf{y}_i^{\mathsf{T}} \mathbf{y}_i}$$

BFGS (Broyden–Fletcher–Goldfarb–Shanno) II

- The following R function updates $[\mathbf{H}_{i-1}]^{-1}$ to $[\mathbf{H}_i]^{-1}$ given $\boldsymbol{\theta}_i$, $\boldsymbol{\theta}_{i-1}$, $\nabla f(\boldsymbol{\theta}_{i-1})$ and $\nabla f(\boldsymbol{\theta}_i)$, which are the arguments x0, x1, g0 and g1, respectively

```r
iH1 <- function(x0, x1, g0, g1, iH0) {
  # Function to update Hessian matrix
  # x0 and x1 are p-vectors of second to last and last estimates, respectively
  # g0 and g1 are p-vectors of second to last and last gradients, respectively
  # iH0 is previous estimate of p x p Hessian matrix
  # returns a p x p matrix
  s0 <- x1 - x0
  y0 <- g1 - g0
  denom <- sum(y0 * s0)
  I <- diag(rep(1, 2))
  pre <- I - tcrossprod(s0, y0) / denom
  post <- I - tcrossprod(y0, s0) / denom
  last <- tcrossprod(s0) / denom
  pre %*% iH0 %*% post + last
}
```

Example: Weibull maximum likelihood: BFGS I

- Repeat Example 5.5 using the BFGS method
- Comment on how it compares to Newton's method

Example: Weibull maximum likelihood: BFGS II

- The following code implements five iterations of the BFGS method

```
## This build of rgl does not include OpenGL functions.  Use
##  rglwidget() to display results, e.g. via options(rgl.printRglwidget = TRUE)
```

```r
iterations <- 5
xx <- matrix(0, iterations + 1, 2)
dimnames(xx) <- list(paste('iter', 0:iterations), c('lambda', 'k'))
xx[1, ] <- c(1.6, .6)
g <- iH <- list()
for (i in 2:(iterations + 1)) {
  g[[i]] <- weib_d1(xx[i - 1, ], y0, mult = -1)
  if (sqrt(sum(g[[i]]^2)) < 1e-6)
    break
  if (i == 2) {
    iH[[i]] <- diag(1, 2)
  } else {
    iH[[i]] <- iH1(xx[i - 2, ], xx[i - 1, ], g[[i - 1]], g[[i]], iH[[i - 1]])
  }
  search_dir <- -(iH[[i]] %*% g[[i]])
 alpha <- line_search(xx[i - 1, ], search_dir, weib_d0, y = y0, mult = -1)
 xx[i, ] <- xx[i - 1,] + alpha * search_dir
}
```

Example: Weibull maximum likelihood: BFGS III

- Our estimates at each iteration are

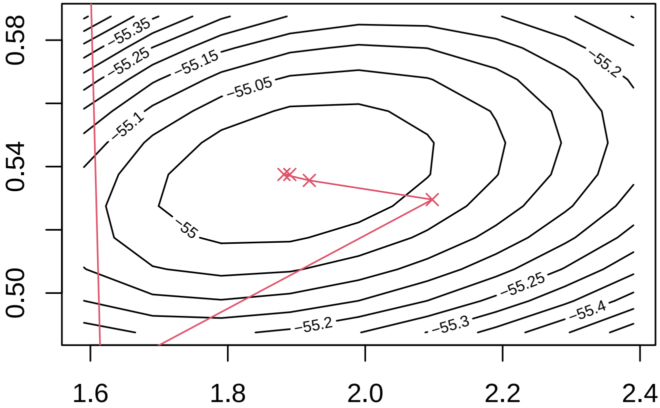
  ```
  xx
  ```

  ```
  ##          lambda         k
  ## iter 0 1.600000 0.6000000
  ## iter 1 1.615241 0.4736904
  ## iter 2 2.097571 0.5295654
  ## iter 3 1.918661 0.5356343
  ## iter 4 1.881726 0.5375145
  ## iter 5 1.890167 0.5374986
  ```

  and we see that we need two more iterations than Newton's method to reach convergence to three decimal places

Example: Weibull maximum likelihood: BFGS IV

- Finally, we'll plot the course of the iterations



and we can see the slightly less direct route we've taken

Quasi-Newton methods in R I

- There are various options for performing quasi-Newton methods in R
- For these, we just need to supply the function to be minimised and its gradient
- The first option is to use `nlminb()` again
    - if we don't supply a function to evaluate the Hessian, then `nlminb()` uses a quasi-Newton approach
- The alternative, and possibly preferred option, is to use `optim()` with option `method = 'BFGS'`
- We'll repeat Example 5.5 using BFGS instead

Quasi-Newton methods in R II

- To use `nlminb()` we can use the following.

```
nlminb(c(1.6, .6), weib_d0, weib_d1, y = y0, mult = -1)
```

```
## $par
## [1] 1.8900689 0.5375279
##
## $objective
## [1] 54.95316
##
## $convergence
## [1] 0
##
## $iterations
## [1] 7
##
## $evaluations
## function gradient
##        9        8
##
## $message
## [1] "relative convergence (4)"
```

Quasi-Newton methods in `R` III

- To use `optim()` we can use the following.

```r
optim(c(1.6, .6), weib_d0, weib_d1, y = y0, mult = -1, method = 'BFGS')
```

```
## $par
## [1] 1.8900632 0.5375283
##
## $value
## [1] 54.95316
##
## $counts
## function gradient
##       14        6
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Quasi-Newton methods in R IV

- We note nlminb() and optim() have essentially given the same value of par, i.e. for the $\hat{\lambda}$ and $\hat{k}$, which is reassuring
- Note that nlminb() has used fewer function evaluations than optim()
    - we won't go into the details of the cause of this, but it is worth noting that the functions use different stopping criteria, and slightly different variants of the BFGS algorithm
- Note also that nlminb() has used three more function evaluations with the BFGS method than with Newton's method
    - this is is typically the case, and reflects the improved convergence achieved by using the actual Hessian matrix with Newton's method, as opposed to the approximation that's used with the BFGS approach

Challenges I

- Go to Challenges I of the week 10 lecture 3 challenges at https://byoungman.github.io/MTH3045/challenges

Week 10 lecture 2

Gradient descent

- If we consider small $\boldsymbol{\Delta}$ in (5.1) then we get the first order approximation

$$f(\boldsymbol{\theta} + \boldsymbol{\Delta}) \simeq f(\boldsymbol{\theta}) + \left[\nabla f(\boldsymbol{\theta})\right]^{\mathsf{T}} \boldsymbol{\Delta},$$

  which is appropriate for small $\boldsymbol{\Delta}$
- The concept behind gradient descent is simple: we want to minimise $\left[\nabla f(\boldsymbol{\theta})\right]^{\mathsf{T}} \boldsymbol{\Delta}$, which requires that we follow the direction of $-\nabla f(\boldsymbol{\theta})$
- To allow for different magnitudes of gradient, we will choose

$$\boldsymbol{\Delta} = -\frac{\nabla f(\boldsymbol{\theta})}{||\nabla f(\boldsymbol{\theta})||}$$

- Now that we know the direction in which we want to head, we need to know how far in that direction we should go. For this we'll consider some $\alpha > 0$, so that

$$\begin{aligned} f(\boldsymbol{\theta} + \boldsymbol{\Delta}) &\simeq f(\boldsymbol{\theta}) - \alpha \frac{\left[\nabla f(\boldsymbol{\theta})\right]^{\mathsf{T}} \left[\nabla f(\boldsymbol{\theta})\right]}{||\nabla f(\boldsymbol{\theta})||}, \\ &= f(\boldsymbol{\theta}) - \alpha ||\nabla f(\boldsymbol{\theta})||, \end{aligned}$$

  which means that $\boldsymbol{\Delta} = -\nabla f(\boldsymbol{\theta}) / ||\nabla f(\boldsymbol{\theta})||$ brings a decrease in $f(\boldsymbol{\theta} + \boldsymbol{\Delta})$ that's proportional to $||\nabla f(\boldsymbol{\theta})||$ for $\alpha > 0$, and is the fastest possible rate at which $f(\boldsymbol{\theta} + \boldsymbol{\Delta})$ can decrease

Example: Weibull maximum likelihood: gradient descent I

- Repeat Example 5.5 using gradient descent with $\alpha = 0.5$ and $\alpha = 0.1$, using 30 iterations for each
- Comment on how these compare to each other, and to Newton's method

```
alpha_seq <- c(.5, .1)
iterations <- 30
for (j in 1:length(alpha_seq)) {
  xx2 <- matrix(0, iterations + 1, 2)
  dimnames(xx2) <- list(paste('iter', 0:iterations), c('lambda', 'k'))
  xx2[1, ] <- lk0
  for (i in 2:(iterations + 1)) {
    gi <- weib_d1(xx2[i - 1, ], y0, mult = -1)
    gi <- gi / sqrt(crossprod(gi)[1, 1])
    xx2[i, ] <- xx2[i - 1,] - alpha_seq[j] * gi
  }
}
```

Example: Weibull maximum likelihood: gradient descent II

- Here's a plot of the iterations

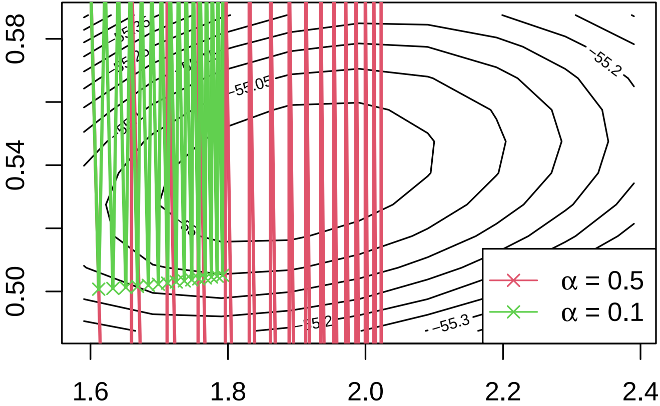


Figure 1: Iterations of the gradient descent algorithm with $\alpha = 0.5$ and $\alpha = 0.1$.

Example: Weibull maximum likelihood: gradient descent III

- In the above example we see that convergence towards $\hat{\lambda}$ and $\hat{k}$ is slow and has not been achieved after 30 iterations of $\alpha = 0.5$ and $\alpha = 0.1$, whereas Newton's method had essentially converged after four or five iterations
- Worse still, if we allowed more iterations, we'd see that both eventually converge, but away from $\hat{\lambda}$ and $\hat{k}$, as opposed to converging

Line search I

- Above we see that, for fixed α, gradient descent has diverged, i.e. not homed in on the minimum of $f()$
- This often happens with gradient descent
- A solution, which also applies to Newton's method, is to use a *line search*
- Consider Newton's method and a search direction of $\mathbf{p}_i = -\left[\nabla^2 f(\boldsymbol{\theta}_i)\right]^{-1} \nabla f(\boldsymbol{\theta}_i)$
- We want $f(\boldsymbol{\theta}_i + \mathbf{p}_i) < f(\boldsymbol{\theta}_i)$ in order for $\boldsymbol{\theta}_i + \mathbf{p}_i$ to be an improvement on $\boldsymbol{\theta}_i$
- If we employ a line search, we instead consider $\boldsymbol{\theta}_i + \alpha\mathbf{p}_i$ for some $\alpha > 0$ and ideally want α to minimise $f(\boldsymbol{\theta}_i + \alpha\mathbf{p}_i)$

Line search II

- In practice line search can be done informally, through the following process

1. Choose an initial value for α, α_0, say, and set $j = 0$
2. Evaluate $f(\boldsymbol{\theta}_i + \alpha_j \mathbf{p}_i)$
3. Set $j = j + 1$
4. Set $\alpha_j = \rho\alpha_{j-1}$, for $0 < \rho < 1$
5. Evaluate $f(\boldsymbol{\theta}_i + \alpha_j \mathbf{p}_i)$
6. If $f(\boldsymbol{\theta}_i + \alpha_j \mathbf{p}_i) < f(\boldsymbol{\theta}_i + \alpha_{j-1} \mathbf{p}_i)$, repeat steps 3 to 6 until $f(\boldsymbol{\theta}_i + \alpha_j \mathbf{p}_i) \geq f(\boldsymbol{\theta}_i + \alpha_{j-1} \mathbf{p}_i)$
7. Choose $\alpha = \alpha_{j-1}$

Line search III

- We can implement this in R

```
line_search <- function(theta, p, f, alpha0 = 1, rho = .5, ...) {
  best <- f(theta, ...)
  cond <- TRUE
  while (cond & alpha0 > .Machine$double.eps) {
    prop <- f(theta + alpha0 * p, ...)
    cond <- prop >= best
    if (!cond)
      best <- prop
    alpha0 <- alpha0 * rho
  }
  alpha <- alpha0 / rho
  alpha
}
```

- *Remark*: Notice the use of the `...` argument here, which passes any extra arguments given to `line_search()` on to `f()`, and hence avoids the need to include `f()`'s arguments in `line_search()`
- This is useful because it makes `line_search()` applicable to any `f()`

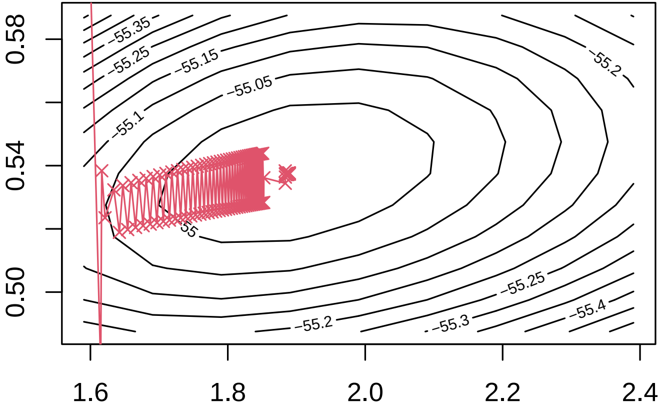
Example: Weibull maximum likelihood: gradient descent with line search I

- Repeat Example 5.5 using gradient descent but with line search and 200 iterations.

```
iterations <- 200
xx2 <- matrix(0, iterations + 1, 2)
dimnames(xx2) = list(paste('iter', 0:iterations), c('lambda', 'k'))
xx2[1, ] <- c(1.6, .6)
for (i in 2:(iterations + 1)) {
  gi <- weib_d1(xx2[i - 1, ], y0, mult = -1)
  gi <- gi / sqrt(crossprod(gi)[1, 1])
  alpha_i <- line_search(xx2[i - 1, ], -gi, weib_d0, y = y0, mult = -1)
  xx2[i, ] <- xx2[i - 1,] - alpha_i * gi
}
```

Example: Weibull maximum likelihood: gradient descent with line search I

- We see that line search does at least bring us convergence of the parameter estimates, but that it's also very slow



Line search: Wolfe conditions

- *Remark*: So far we've adopted an informal approach to line search
- A more formal approach is to choose α so that it satisfies the **Wolfe conditions**
- A step length α_k is said to satisfy the Wolfe conditions, restricted to the direction $\mathbf{p}_i$, if the following two inequalities hold:

$$\textbf{i)} \quad f(\boldsymbol{\theta}_i + \alpha_i \mathbf{p}_i) \leq f(\boldsymbol{\theta}_i) + c_1 \alpha_i \mathbf{p}_i^{\mathrm{T}} \nabla f(\boldsymbol{\theta}_i),$$

$$\textbf{ii)} \quad -\mathbf{p}_i^{\mathrm{T}} \nabla f(\boldsymbol{\theta}_i + \alpha_i \mathbf{p}_i) \leq -c_2 \mathbf{p}_i^{\mathrm{T}} \nabla f(\boldsymbol{\theta}_i),$$

  with $0 < c_1 < c_2 < 1$. c_1 is usually chosen to be quite small while c_2 is much larger
- Nocedal and Wright (2006, sec. 6.1) give example values of $c_1 = 10^{-4}$ and $c_2 = 0.9$ for Newton or quasi-Newton methods

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.