MTH3045: Statistical Computing

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

12/3/2024

Week 9 lecture 1

Last lecture's challenge

- Recall Challenges II of the week 8 lecture 3 challenges at https://byoungman.github.io/MTH3045/challenges

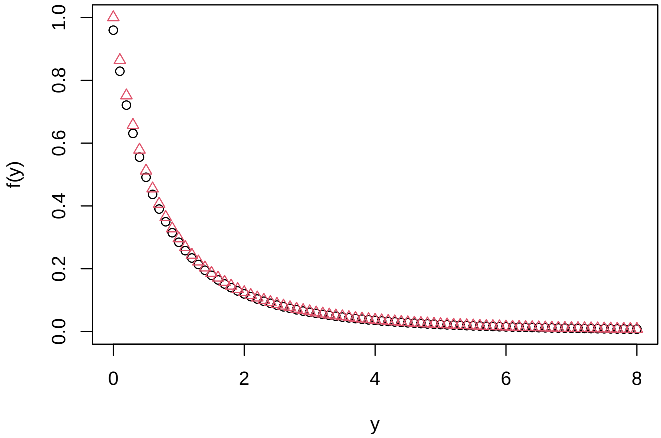
Solution

- We first notice that, given Equation (4.4), we take $f()$ to be $-\log f(y, \lambda)$ and x to be λ, as that's what we're integrating out.
- Then we get $f(y, \lambda) = f(y \mid \lambda) f(\lambda) = \lambda e^{-\lambda y} \times 4\lambda e^{-2\lambda} = 4\lambda^2 e^{-\lambda(2+y)}$
- So $-\log f(y, \lambda) = -2\log\lambda + \lambda(2+y) - \log 4$
- Then $-\dfrac{\partial \log f(y, \lambda)}{\partial \lambda} = -\dfrac{2}{\lambda} + (2+y)$, which is minimised at $\hat{\lambda} = 2/(2+y)$
- Also $-\dfrac{\partial^2 \log f(y, \lambda)}{\partial \lambda^2} = 2/\lambda^2$ and $2/\hat{\lambda}^2 = 2/(4/(2+y)^2) = (2+y)^2/2$
- So, if $g''(\lambda)$ denotes $-\partial^2 \log f(y, \lambda)/\partial\lambda^2$, then

$$
\begin{aligned}
I_n &= f(y, \tilde{\lambda}) \sqrt{\frac{2\pi}{g''(\tilde{\lambda})}} \\
&= 4\hat{\lambda}^2 e^{-\hat{\lambda}(2+y)} \sqrt{2\pi \times 2(2+y)^{-2}} = \frac{32\sqrt{\pi}}{e^2(2+y)^3}
\end{aligned}
$$

Solution continued

```
par(mar = c(4, 4, 1, 1))
y <- seq(0, 8, by = .1)
plot(y, 32 * sqrt(pi) * exp(-2) / (2 + y)^3,
     ylim = 0:1, xlab = 'y', ylab = 'f(y)')
points(y, 8 / (y + 2)^3, pch = 2, col = 2)
```



Week 9 lecture 1

Monte Carlo integration I

- So far we have considered *deterministic* approaches to numerical integration; these will always give the same answer
- It is worth, however, considering a stochastic approach to integration known as *Monte Carlo* integration
- Now consider $\mathbf{X} = (X_1, \ldots, X_d)$ on some finite region $\mathcal{X}$ with volume $V(\mathcal{X})$, where, additionally $\mathbf{X}$ is uniformly distributed on $\mathcal{X}$
- Then

$$I_{\mathcal{X}} = \int_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) \mathrm{d}\mathbf{x} = \mathrm{E}[f(\mathbf{X})] V(\mathcal{X}) \tag{4.5}$$

- The standard Monte Carlo estimate for $I_{\mathcal{X}}$ is given by

$$\hat{I}_{\mathcal{X}} \simeq \frac{V(\mathcal{X})}{N} \sum_{i=1}^{N} f(\tilde{\mathbf{x}}_i)$$

where $\tilde{\mathbf{x}}_i$, for $i = 1, \ldots, N$, are drawn *uniformly* from $\mathcal{X}$

Monte Carlo integration II

- We can immediately see that Monte Carlo integration is a powerful tool as, provided we can uniformly generate points on $\mathcal{X}$, $\tilde{\mathbf{x}}$, and evaluate $f(\tilde{\mathbf{x}})$, then we can approximate $I_{\mathcal{X}}$
- A natural next question is the accuracy of this approximation and, as $\hat{I}_{\mathcal{X}}$ is unbiased, its variance

$$\mathrm{Var}(\hat{I}_{\mathcal{X}}) = \frac{V(\mathcal{X})^2}{N}\mathrm{Var}[f(\mathbf{X})]$$

  is key
- As $\mathrm{Var}(\hat{I}_{\mathcal{X}})$ decreases linearly with $1/N$, convergence is rather slow
- The following example confirms this

Example: One-dimensional Monte Carlo integration I

- Recall the integral $\int_0^1 \exp(x)\mathrm{d}x = \exp(1) - 1 \simeq 1.7182818$ from Example 4.5

- Use R and Monte Carlo integration to approximate the integral with $N = 100$, 1000 and 10000, using the mean of $m = 100$ replicates for each value of N as your approximation

- Then compare the relative absolute error for each value of N

- We'll start with a function, `mc(f, N)`, which we'll use to approximate the integral for given N

```
mc <- function(f, N, a = 0, b = 1, ...) {
  x <- runif(N, a, b)
  V <- b - a
  V * mean(f(x, ...))
}
```

and will quickly test for $N = 100$

```
mc(function(x) exp(x), 100)
```

```
## [1] 1.832858
```

Example: One-dimensional Monte Carlo integration I

- Note that as the vector `x` contains random variables, you may get a slightly different result; hence the replicates

- Now we'll perform $m = 100$ replicates for each Monte Carlo sample size N

```r
estimates <- numeric(3)
N_vals <- 10^c(2:4)
m <- 100
for (i in 1:length(N_vals))
  estimates[i] <- mean(replicate(m, mc(function(x) exp(x), N_vals[i])))
estimates
```

```
## [1] 1.715251 1.719208 1.718545
```

- Finally, we'll compare these to the true integral, `true` below, by calculating the relative absolute error, `rel_err`

```r
true <- exp(1) - 1
rel_err <- abs(true - estimates) / true
rel_err
```

```
## [1] 0.0017637093 0.0005390542 0.0001529152
```

- We see that the higher values of N give more accurate approximations, once we allow for variability in Monte Carlo samples

Challenges 1

- Go to Challenges I of the week 9 lecture 1 challenges at https://byoungman.github.io/MTH3045/challenges

Example: Multi-dimensional Monte Carlo integration I

- Recall Example 4.12, i.e.

$$I = \int_0^1 \dots \int_0^1 \exp(\sum_{i=1}^{5} x_i) \mathrm{d}x_1 \dots \mathrm{d}x_5$$

- Use Monte Carlo integration with Monte Carlo samples of size $N = 10, 100, 10^3, 10^4$ and 10^5 to estimate I and estimate the relative absolute error for each N

Example: Multi-dimensional Monte Carlo integration II

- The following code approximates the integral for the different values of N

```
d <- 5
true <- (exp(1) - 1)^d
NN <- 10^c(1:5)
f <- numeric(length(NN))
for (i in 1:length(NN))
  f[i] <- mean(exp(rowSums(matrix(runif(d * NN[i]), NN[i]))))
cbind(true = true, monte_carlo = f, rel.err = abs(f - true) / true)
```

```
##          true monte_carlo     rel.err
## [1,] 14.97863    17.62581 0.176730779
## [2,] 14.97863    13.99063 0.065960649
## [3,] 14.97863    14.91521 0.004233631
## [4,] 14.97863    14.92255 0.003743597
## [5,] 14.97863    15.00143 0.001522209
```

- We see that the relative absolute error decreases as N increases, even with just one replicate

Challenges II

- Go to Challenges II of the week 9 lecture 1 challenges at https://byoungman.github.io/MTH3045/challenges

Week 9 lecture 2

Multi-dimensional Monte Carlo integration over a pdf

- The above integral, $I_{\mathcal{X}} = \int_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})\mathrm{d}\mathbf{x}$ can be seen as a special case of

$$I_{g(\mathcal{X})} = \int_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})g(\mathbf{x})\mathrm{d}\mathbf{x},$$

  for a pdf $g(\mathbf{x})$
- This can be estimated by Monte Carlo as

$$\hat{I}_{g(\mathcal{X})} \simeq \frac{1}{N}\sum_{i=1}^{N} f(\tilde{\mathbf{x}}_i) \tag{4.6}$$

  where $\tilde{\mathbf{x}}_i$ are draws from the pdf $g(\mathbf{x})$

Example: Poisson marginal approximation using Monte Carlo integration I

- Recall Example 4.8, i.e. $Y \mid \lambda \sim \text{Poisson}(\lambda)$, where $\lambda \sim \text{N}(\mu, \sigma^2)$ with $\mu = 10$ and $\sigma = 3$.
- Use Monte Carlo integration with $N = 1000$ to approximate $f(y)$
- We need to draw a sample of size $N = 1000$ from $f(\lambda)$, which corresponds to $g(\mathbf{x})$ in Equation (4.6)
- The following code does this, simply by calling `rnorm()`
- We'll call the sample `lambda`

```
N <- 1e3
lambda <- rnorm(N, 10, 3)
```

- By definition, $\lambda > 0$, however...

```
range(lambda)
```

```
## [1] -0.8751556 19.3958499
```

shows that we have some $\lambda < 0$, which we'll just set to zero

```
lambda_pos <- pmax(0, lambda)
```

Example: Poisson marginal approximation using Monte Carlo integration II

- We can estimate $f(y)$ as

$$\hat{f}(y) = \frac{1}{N}\sum_{i=1}^{N} f(y \mid \tilde{\lambda}_i)$$

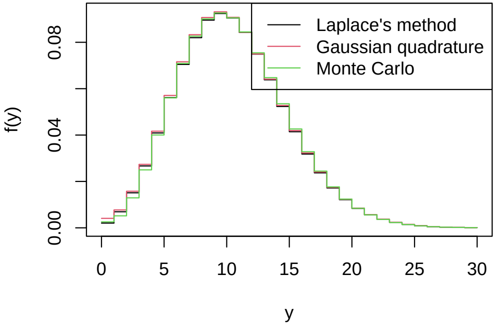
where $\tilde{\lambda}_i$ for $i = 1, \ldots, N$ are draws from the $N(10, 3^2)$ pdf, which we'll store as `fhat3`

```r
fhat3 <- sapply(y_vals, function(z) mean(dpois(z, lambda_pos)))
```

Example: Poisson marginal approximation using Monte Carlo integration III

- Finally, we'll plot the resulting estimate alongside those of Gaussian quadrature and Laplace's method

```
matplot(y_vals, cbind(fhat, fhat2, fhat3),
        lty = 1, type = 'l', xlab = 'y', ylab = 'f(y)')
legend('topright', c("Laplace's method", "Gaussian quadrature", "Monte Carlo"))
```



Example: Poisson marginal approximation using Monte Carlo integration III

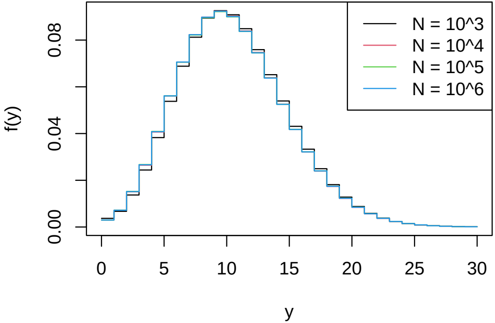
- *Remark*: The larger the Monte Carlo sample size, the more accurate the approximation
- The following repeats Example 4.8 for $N = 10^4$, 10^5 and 10^6, comparing the results against $N = 10^3$

```r
NN <- 10^c(3:6)
fhat4 <- sapply(NN, function(N) {
  lambda <- rnorm(N, 10, 3)
  sapply(y_vals, function(z) mean(dpois(z, pmax(0, lambda))))
}
)
```

Example: Poisson marginal approximation using Monte Carlo integration IV

- Although we don't have the true $f(y)$ against which to compare the Monte Carlo estimates, we are seeing in the Figure below approximations that increase with accuracy as N increases

```
matplot(y_vals, fhat4, lty = 1, type = 'l', xlab = 'y', ylab = 'f(y)')
```



Bibliographic notes

- For details on numerical differentiation see Wood (2015, sec. 5.5.2) or Nocedal and Wright (2006, chap. 8)
- For details on quadrature see Monahan (2011, chap. 10) or Press et al. (2007, chap. 4)
- For details on Laplace's method see Davison (2003, sec. 11.3.1), Wood (2015, sec. 5.3.1) or Monahan (2011, sec. 12.6)
- For details on Monte Carlo integration see Monahan (2011, chap. 12) or Davison (2003, sec. 3.3)

Optimisation

- When fitting a statistical model we often use maximum likelihood
- For this we have some likelihood function $f(\mathbf{y} \mid \boldsymbol{\theta})$, data $\mathbf{y} = (y_1, \ldots, y_n)$ and unknown but *fixed* parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$
- We are then required to find

$$\hat{\boldsymbol{\theta}} = \arg \max_{\boldsymbol{\theta}} f(\mathbf{y} \mid \boldsymbol{\theta})$$

- Sometimes, it will be possible to find $\hat{\boldsymbol{\theta}}$ analytically, but sometimes not
- The normal linear model is one example of the former, whereas the gamma distribution is an example of the latter

Example: Maximum likelihood estimation with the gamma distribution I

- Consider an independent sample of data $\mathbf{y} = (y_1, \ldots, y_n)$, and suppose that these are modelling as Gamma(α, β) realisations, i.e. with pdf

$$f(y \mid \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} y^{\alpha-1} e^{-\beta y} \quad \text{for } y > 0$$

  where $\alpha, \beta > 0$ and $\Gamma()$ is the gamma function
- The log-likelihood is then

$$\log f(\mathbf{y} \mid \alpha, \beta) = n\alpha \log \beta - n \log \Gamma(\alpha) + (\alpha - 1) \sum_{i=1}^{n} \log y_i - \beta \sum_{i=1}^{n} y_i$$

Example: Maximum likelihood estimation with the gamma distribution II

- We can write its derivatives w.r.t. (α, β) in the vector

$$\begin{pmatrix} \dfrac{\partial \log f(\mathbf{y} \mid \alpha, \beta)}{\partial \alpha} \\ \dfrac{\partial \log f(\mathbf{y} \mid \alpha, \beta)}{\partial \beta} \end{pmatrix} = \begin{pmatrix} n \log \beta - n \dfrac{\partial \log \Gamma(\alpha)}{\partial \alpha} + \sum_{i=1}^{n} \log y_i \\ n\alpha/\beta - \sum_{i=1}^{n} y_i \end{pmatrix}$$

- Unfortunately, we cannot analytically find both the maximum likelihood estimates, $(\hat{\alpha}, \hat{\beta})$
    - i.e. we cannot find α and β simultaneously that satisfy that $\partial \log f(\mathbf{y} \mid \alpha, \beta)/\partial\alpha$ and $\partial \log f(\mathbf{y} \mid \alpha, \beta)/\partial\beta$ are both zero
- Fortunately, we can still find $(\hat{\alpha}, \hat{\beta})$, but just not analytically

Numerical maximum likelihood estimation

- When we cannot find $\hat{\boldsymbol{\theta}}$ analytically, our next option is to find it numerically
    - that is, to adopt some kind of iterative process that we expect will ultimately result in the $\hat{\boldsymbol{\theta}}$ that we want, as opposed to an estimate that we don't want
- Although in maximum likelihood estimation interest lies in finding $\boldsymbol{\theta}$ that *maximises* $f(\mathbf{y} \mid \boldsymbol{\theta})$, it is much more common in mathematics to want to *minimise* a function
- Fortunately, maximising $f(\mathbf{y} \mid \boldsymbol{\theta})$ is equivalent to minimising $-f(\mathbf{y} \mid \boldsymbol{\theta})$, i.e. $f(\mathbf{y} \mid \boldsymbol{\theta})$ negated
- Therefore,

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \{-f(\mathbf{y} \mid \boldsymbol{\theta})\}$$

- So that we can better follow the literature on numerical optimisation, we'll just consider finding minima

Root finding

- We'll start this chapter with a brief aside on root finding, because our main concern will be finding values that maximise (or minimise) functions
- 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.

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)

```r
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 9 lecture 2 challenges at https://byoungman.github.io/MTH3045/challenges

References

Week 9 lecture 3

Newton's method in one-dimension I

- Recall Theorem 4.1
- The second-order approximation, if we swap from x to θ,

$$f(\theta) \simeq f(\theta_0) + (\theta - \theta_0)f'(\theta_0) + \frac{1}{2}(\theta - \theta_0)^2 f''(\theta_0)$$

  can be re-written for small δ as

$$f(\theta + \delta) \simeq f(\theta) + \delta f'(\theta) + \frac{1}{2}\delta^2 f''(\theta)$$

  if we consider values near θ for some twice-differentiable function $f()$

- If we're trying to find $\theta^* = \theta + \delta$ that minimises $f(\theta + \delta)$ iteratively, then we want θ^* to be an improvement on θ, i.e. $f(\theta + \delta) < f(\theta)$
- The best value of $\theta + \delta$ therefore minimises $f(\theta) + \delta f'(\theta) + \frac{1}{2}\delta^2 f''(\theta)$, and if we differentiate this w.r.t. δ we get

$$f'(\theta) = -\delta f''(\theta)$$

  so that

$$\delta = -\frac{f'(\theta)}{f''(\theta)}$$

Newton's method in one-dimension II

- The above result

  $$\delta = -\frac{f'(\theta)}{f''(\theta)}$$

  is the basis for **Newton's method** whereby, if we assume we have a value of θ at iteration i, then we update this at the next iteration so that

  $$\theta_{i+1} = \theta_i - \frac{f'(\theta_i)}{f''(\theta_i)}$$

- *Remark*: For the one dimensional case of Newton's method, we will refer to $p_i = -f'(\theta_i)/f''(\theta_i)$ as the **Newton step**

Example: Poisson maximum likelihood I

- Let Y_i, $i = 1, \ldots, n$, denote the numbers of cars passing the front of the Laver building between 9 and 10am on weekdays during term time
- Assume that these are independent from one day to the next, and that $Y_i \sim \text{Poisson}(\mu)$, for some unknown μ
- The likelihood for a sample of data is given by

  $$f(\mathbf{y} \mid \mu) = \prod_{i=1}^{n} \frac{\mu^{y_i} \mathrm{e}^{-\mu}}{y_i!}$$

  and for which we can write the log-likelihood as

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

- Of course, we can solve this analytically, which gives a maximum likelihood estimate of $\hat{\mu} = \bar{y} = \sum_{i=1}^{n} y_i / n$

Example: Poisson maximum likelihood II

- Given the counts below

$$20, 21, 23, 25, 26, 26, 30, 37, 38, 41,$$

  we find that $\hat{\mu} = 28.7$

- Use five iterations of Newton's method to find $\hat{\mu}$ iteratively, starting with $\mu_0 = 28$, and comment on how many iterations are required to find $\hat{\mu}$ to two decimal places

- We'll start by reading in the data.

```r
y <- c(20, 21, 23, 25, 26, 26, 30, 37, 38, 41)
```

- Then we'll find the first and second derivatives of $-\log f(\mathbf{y} \mid \mu)$ w.r.t. μ, which are $n - (\sum_{i=1}^{n} y_i)/\mu$ and $(\sum_{i=1}^{n} y_i)/\mu^2$, respectively

Example: Poisson maximum likelihood III

- Next we'll write functions in R, which we'll call d1 and d2, to evaluate these analytical derivatives

```r
d1 <- function(mu, y) {
  # Function to evaluate first derivative w.r.t. mu of
  # Poisson(mu) neg. log likelihood
  # mu is a scalar
  # y can be scalar or vector
  # returns a scalar
  length(y) - sum(y) / mu
}

d2 <- function(mu, y) {
  # Function to evaluate second derivative w.r.t. mu of
  # Poisson(mu) neg. log likelihood
  # mu is a scalar
  # y can be scalar or vector
  # returns a scalar
  sum(y) / mu^2
}
```

Example: Poisson maximum likelihood IV

- Then we'll iterate estimates of μ using Newton's method

```
mu_i <- numeric(6)
mu_i[1] <- 28
for (i in 1:(length(mu_i) - 1))
  mu_i[i + 1] <- mu_i[i] - d1(mu_i[i], y) / d2(mu_i[i], y)
mu_i
```

```
## [1] 28.00000 28.68293 28.69999 28.70000 28.70000 28.70000
```

- Finally, we'll compare our estimate from Newton's method with the true maximum likelihood estimate

```
(mu_its <- min(which(round(abs(mu_i - mu_hat), 2) == 0)) - 1)
```

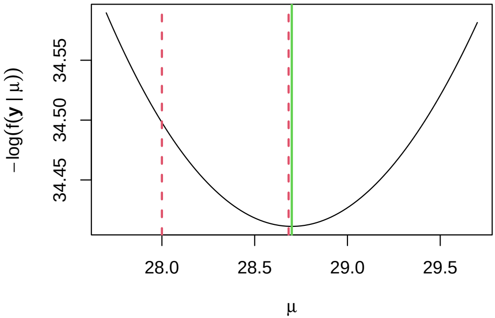
```
## [1] 2
```

and find that after 2 iterations our iterative estimate matches the true value of $\hat{\mu}$ to three decimal places

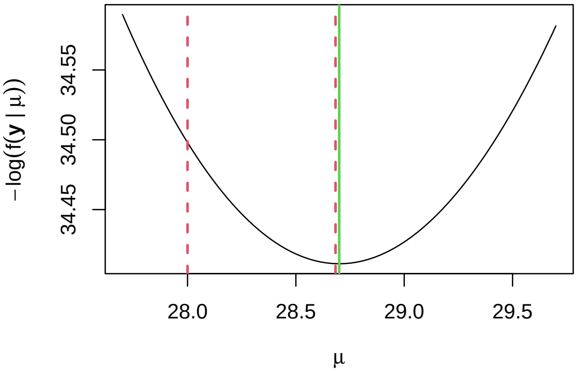
Example: Poisson maximum likelihood V

- Let's quickly look at how our iterations have gone
- We'll superimpose them on top of a plot of the negative log-likelihood as horizontal, dotted red lines, with the true value of $\hat{\mu}$ shown in green.

```
mu_seq <- seq(mu_hat - 1, mu_hat + 1, l = 1e2)
f_seq <- sapply(mu_seq, function(z) -sum(log(dpois(y, z))))
plot(mu_seq, f_seq, type = "l",
     xlab = expression(mu), ylab = expression(-log(f(bold(y) * " | " *mu))))
abline(v = mu_hat, col = 3, lwd = 2)
abline(v = mu_i[seq_len(mu_its)], col = 2, lty = 2, lwd = 2)
```



Example: Poisson maximum likelihood VI



- We see that Newton's method has quickly homed in on the true value of $\hat{\mu}$
- Although this example is simple, and is one in which Newton's method will typically perform well, Newton's method is incredibly powerful, and will often perform well in a wide range of scenarios

Challenge I

- Go to Challenges I of the week 9 lecture 3 challenges at https://byoungman.github.io/MTH3045/challenges

Week 10 lecture 1

Newton's multi-dimensional method

Taylor's theorem (multivariate) I

- The above extension to Taylor's theorem in the univariate case applies to the multivariate case, after swapping $\mathbf{x}$ for $\boldsymbol{\theta}$, so that

$$f(\boldsymbol{\theta}) \simeq f(\boldsymbol{\theta}_0) + \left[\nabla f(\boldsymbol{\theta}_0)\right]^{\mathsf{T}} (\boldsymbol{\theta} - \boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^{\mathsf{T}} \left[\nabla^2 f(\boldsymbol{\theta}_0)\right] (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

  can be re-written as

$$f(\boldsymbol{\theta} + \boldsymbol{\Delta}) \simeq f(\boldsymbol{\theta}) + \left[\nabla f(\boldsymbol{\theta})\right]^{\mathsf{T}} \boldsymbol{\Delta} + \frac{1}{2}\boldsymbol{\Delta}^{\mathsf{T}} \left[\nabla^2 f(\boldsymbol{\theta})\right] \boldsymbol{\Delta} \qquad (5.1)$$

  As similar argument to that above, i.e. finding $\boldsymbol{\theta} + \boldsymbol{\Delta}$ that minimises equation (5.1), gives

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

  for which we require that $\nabla^2 f(\boldsymbol{\theta}_i)$ is positive semi-definite in order to ensure that $f(\boldsymbol{\theta}_{i+1}) \leq f(\boldsymbol{\theta}_i)$

Taylor's theorem (multivariate) II

- *Remark 1*: For the multi-dimensional case of Newton's method, we will refer to $\mathbf{p}_i = -\left[\nabla^2 f(\boldsymbol{\theta}_i)\right]^{-1} \nabla f(\boldsymbol{\theta}_i)$ as the **Newton step**
- *Remark 2*: Sometimes it may turn out that $\nabla^2 f(\boldsymbol{\theta}_i)$ is not positive semi-definite
- Fortunately, this does not prohibit use of Newton's method because we can *perturb* $\nabla^2 f(\boldsymbol{\theta}_i)$ so that it is positive semi-definite, which will then guarantee that $f(\boldsymbol{\theta}_{i+1}) \leq f(\boldsymbol{\theta}_i)$
- There are various options for perturbation, but a common choice is to use $\nabla^2 f(\boldsymbol{\theta}_i) + \gamma \mathbf{I}_p$, where $\mathbf{I}_p$ is the $p \times p$ identity matrix, and we choose γ very small, and sequentially increase its value until $\nabla^2 f(\boldsymbol{\theta}_i) + \gamma \mathbf{I}_p$ is positive semi-definite
- For example, we might proceed through $\gamma = 10^{-12}, 10^{-10}, \ldots$

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

```
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

Example: Weibull maximum likelihood: Newton's method V

- We'll start by plotting the log-likelihood surface
- We wouldn't normally do this, but it can be useful to quickly judge whether the log-likelihood surface is well-behaved, such as being unimodal and approximately quadratic about its maximum

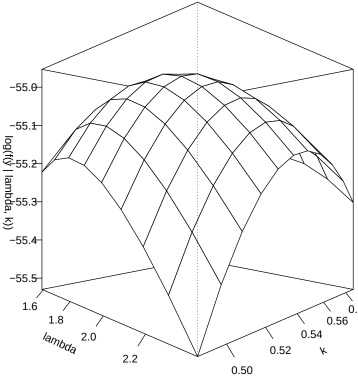


Figure 1: Log-likelihood surface of Weibull distribution model for wind speed data with different λ and k values.

Example: Weibull maximum likelihood: Newton's method VI

- Then we'll write functions `weib_d1` and `weib_d2` to evaluate the first and second derivatives of the Weibull distribution's log-likelihood w.r.t. λ and k
- We'll create these as functions of...
  - `pars`, the vector of parameters
  - `y` the vector of data
  - `mult`, a multiplier of the final log-likelihood, which defaults to `1`
- Introducing `mult` makes it much simpler when we later need the negative log-likelihood, as we don't have to write separate functions

Example: Weibull maximum likelihood: Newton's method VII

- Often when calculating derivatives w.r.t. multiple parameters, we find that calculations are repeated
- It is worth avoiding repetition, and instead storing the results of any calculations that are used multiple times as objects. We'll do this in the next few lines of code, storing the re-used objects as z1, z2, etc.

```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
}
```

Example: Weibull maximum likelihood: Newton's method VIII

```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
}
```

Example: Weibull maximum likelihood: Newton's method IX

- Next we'll perform five iterations of Newton's method, although we see that reasonable convergence is achieved after two or three iterations

```
iterations <- 5
xx <- matrix(0, iterations + 1, 2)
dimnames(xx) <- list(paste('iter', 0:iterations), c('lambda', 'k'))
lk0 <- c(1.6, .6)
xx[1, ] <- lk0
for (i in 2:(iterations + 1)) {
  gi <- weib_d1(xx[i - 1, ], y0)
  Hi <- weib_d2(xx[i - 1, ], y0)
  xx[i, ] <- xx[i - 1,] - solve(Hi, gi)
}
xx
```

```
##          lambda         k
## iter 0 1.600000 0.6000000
## iter 1 1.712945 0.5328618
## iter 2 1.866832 0.5375491
## iter 3 1.889573 0.5375304
## iter 4 1.890069 0.5375279
## iter 5 1.890069 0.5375279
```

Example: Weibull maximum likelihood: Newton's method X

- Finally we can plot the course of the iterations, and see that Newton's method quickly homes in on the log-likelihood surface's maximum

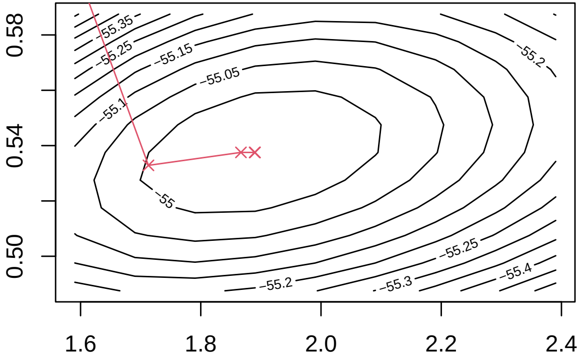


Figure 2: Five iterations of Newton's method to find Weibull maximum likelihood estimates.

References

Davison, A. C. 2003. *Statistical Models*. Cambridge University Press. https://encore.exeter.ac.uk/iii/encore/record/C__Rb3441825__SStatistical%20Models__Orightresult__U__X7?lang=eng&suite=cobalt.

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.