Optimisation

Max Turgeon

STAT 3150–Statistical Computing

Lecture Objectives

- Give examples of optimisation problems in statistics.
- Apply the Newton-Raphson algorithm to compute Maximum Likelihood estimates.

Motivation

- **Optimisation** is a very old and broad field of applied mathematics.
    - It is older than calculus: see Heron's problem.
- It plays an important role in statistics, but also computational biology, economics, mathematical finance, etc.
- Our discussion will necessarily be brief, but I will try to give a sample of the types of statistical problems that can be solved using optimisation.
    - For more details, I recommend *Optimization* (2013) by Kenneth Lange.

Problem statement

- In very general terms, the basic optimisation problem is as follows.
- Let A be a set (e.g. a subset of $\mathbb{R}^n$, a set of integers, etc.), and let f be a real-valued function on A.
- We are looking for a value $x_0 \in A$ such that:
  - **Minimisation**: $f(x_0) \leq f(x)$ for all $x \in A$.
  - **Maximisation**: $f(x_0) \geq f(x)$ for all $x \in A$.
- The set A can be defined via a combination of equality and inequality constraints.
  - For a multinomial model, we want K probabilities p_k such that $0 \leq p_k \leq 1$ and $\sum_{k=1}^K p_k = 1$.

Lasso regression

- In linear regression, we find an estimate $\hat{\beta}$ by minimising least squares:

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \sum_{i=1}^{n} \left(Y_i - \mathbf{X}_i^T \beta \right)^2 .$$

- In lasso regression, we add a constraint on the L1 norm of β:

$$\hat{\beta}_{Lasso} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \sum_{i=1}^{n} \left(Y_i - \mathbf{X}_i^T \beta \right)^2 , \quad \sum_{j=1}^{p} |\beta_j| \leq \lambda.$$

- **Why?** By adding the L1 constraint, some of the components of the solution $\hat{\beta}_{Lasso}$ can be exactly zero, leading to *variable selection*.

Dimension reduction

- Let $\mathbf{Y}$ be a p-dimensional random vector.
- In dimension reduction, we are looking for a linear combination $w^T\mathbf{Y}$ that optimises a given criterion.
- For example:
    - In **Principal Component Analysis**, we are looking to maximise $\mathrm{Var}\left(w^T\mathbf{Y}\right)$.
    - In **Independent Component Analysis**, we are looking to minimise mutual information.

Mixture models

- Recall from earlier this semester *mixture models*:

$$F(x) = \sum_{k=1}^{K} \pi_k F_k(x),$$

  where F_k is a distribution function for all k and $\sum_{k=1}^{K} \pi_k = 1$.
- The likelihood can be difficult to maximise directly.
- Instead, we typically use the **Expectation-Maximization algorithm**.

Machine learning

- A common goal in machine learning is *prediction*: given a vector of features $\mathbf{X}$, find a function f such that

$$L(Y, f(\mathbf{X}))$$

  is minimised.
- Here $L(Y, \cdot)$ is a *loss function*. For example:
    - $L(Y, f(\mathbf{X})) = (Y - f(\mathbf{X}))^2$
    - $L(Y, f(\mathbf{X})) = |Y - f(\mathbf{X})|$

Newton-Raphson method i

- The Newton-Raphson method is the only optimisation technique we will discuss.
    - It is a very important technique and serves as motivation for more advanced methods.
- Suppose we have a twice differentiable function $f : A \to \mathbb{R}$, where $A \subseteq \mathbb{R}^n$. We want to find the maximum of f on A.
- We can approximate f using a Taylor expansion:

$$f(\mathbf{x} + \mathbf{h}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \frac{1}{2}\mathbf{h}^T D^2 f(\mathbf{x})\mathbf{h},$$

  where $\nabla f(\mathbf{x})$ is the vector of first-order derivatives and $D^2 f(\mathbf{x})$ is the matrix of second-order derivatives.

Newton-Raphson method ii

- They are called the **gradient** and the **Hessian**, respectively.
- In other words, we are approximating f in a neighbourhood of $\mathbf{x}$ using a quadratic function of $\mathbf{h}$.
- We know how to find the maximum of a quadratic function: take the derivative, set it equal to zero, and solve for $\mathbf{h}$.
    - **Why?** The derivative of a quadratic is a linear function. Easy to solve!
- The derivative of the Taylor approximation with respect to $\mathbf{h}$ is

$$\frac{d}{d\mathbf{h}}\left[f(\mathbf{x}) + \nabla f(\mathbf{x})^T\mathbf{h} + \frac{1}{2}\mathbf{h}^T D^2 f(\mathbf{x})\mathbf{h}\right] = \nabla f(\mathbf{x}) + D^2 f(\mathbf{x})\mathbf{h}.$$

Newton-Raphson method iii

- Setting the derivative equal to zero and solving for $\mathbf{h}$ gives

$$\mathbf{h} = -[D^2 f(\mathbf{x})]^{-1} \nabla f(\mathbf{x}).$$

- Therefore, the *approximate* maximum of our function f is a neighbourhood of $\mathbf{x}$ occurs at

$$\mathbf{x} + \mathbf{h} = \mathbf{x} - [D^2 f(\mathbf{x})]^{-1} \nabla f(\mathbf{x}).$$

- In one-dimension, this simplifies to

$$x + h = x - \frac{f'(x)}{f''(x)}.$$

Newton-Raphson method iv

Algorithm
Let $\mathbf{x}_0$ be an initial value. Then construct a sequence via

$$\mathbf{x}_{n+1} = \mathbf{x}_n - [D^2 f(\mathbf{x}_n)]^{-1} \nabla f(\mathbf{x}_n).$$

If the initial value $\mathbf{x}_0$ is "sufficiently close" to a maximum (or minimum) of f, then the sequence $\mathbf{x}_n$ will converge to that maximum (or minimum).

Example 1 i

- Consider the function $f(x) = -x^2 + \sin(x)$.
- The derivatives are

$$f'(x) = -2x + \cos(x), \quad f''(x) = -2 - \sin(x).$$

Example 1 ii

```r
# Create functions
gradient <- function(x) -2*x + cos(x)
hessian <- function(x) -2 - sin(x)
# Set-up parameters
x_current <- 1
x_next <- 0
tol <- 10^-10
iter <- 0
```

Example 1 iii

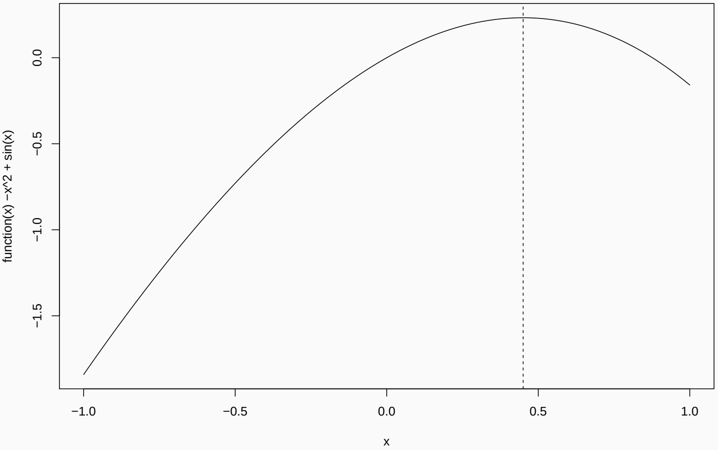
```r
while (abs(x_current - x_next) > tol &
       iter < 100) {
  iter <- iter + 1
  step <- gradient(x_current)/hessian(x_current)
  update <- x_current - step
  x_current <- x_next
  x_next <- update
}
x_next
```

```
## [1] 0.4501836
```

Example 1 iv

```r
plot(function(x) -x^2 + sin(x), from = -1, to = 1)
abline(v = x_next, lty = 2)
```

Example 1 v



When does it go wrong?

- Newton-Raphson usually goes wrong in one of two ways:
  - $\nabla f(\mathbf{x}_0) = 0$ and you're stuck at $\mathbf{x}_0$.
  - The sequence $\mathbf{x}_n$ does not converge.
- That's why we kept track of the number of iterations, so that we aren't stuck in an infinite loop.
- When you reach the maximum number of iterations, you can:
  - Try a different initial value.
  - Look at a graph and see if there is indeed a maximum/minimum.
  - Consider using a different approach (e.g. gradient descent).

Example (not convergent)

- Consider $f(x) = 0.25x^4 - x^2 + 2x$.
- The derivatives are

$$f'(x) = x^3 - 2x + 2, \quad f''(x) = 3x^2 - 2.$$

- If we start at $x_0 = 0$, then at the next iteration we get

$$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)} = 0 - \frac{2}{-2} = 1.$$

- Next we get

$$x_2 = x_1 - \frac{f'(x_1)}{f''(x_1)} = 1 - \frac{1}{1} = 0.$$

- Therefore, we are stuck in an infinite loop.
  - **Solution**: Use a different starting value.

Exercise

Find the minimum for the function

$$f(x) = 5x + 5x^2 + 5\log(1 + e^{-x}).$$

Solution i

We need to compute the first and second derivatives:

$$
\begin{aligned}
f'(x) &= 5 + 10x - \frac{5e^{-x}}{1+e^{-x}} = 5 + 10x - \frac{5}{1+e^{x}}, \\
f'' &= 10 + \frac{5e^{x}}{(1+e^{x})^2}.
\end{aligned}
$$

Solution ii

```r
gradient <- function(x) 5 + 10*x - 5/(1+exp(x))
hessian <- function(x) 10 + 5*exp(x)/(1+exp(x))^2

x_current <- 1
x_next <- 0
tol <- 10^-10
iter <- 0
```

Solution iii

```r
while (abs(x_current - x_next) > tol &
       iter < 100) {
  iter <- iter + 1
  step <- gradient(x_current)/hessian(x_current)
  update <- x_current - step
  x_current <- x_next
  x_next <- update
}
x_next
```

```
## [1] -0.2223235
```

- **Poisson regression** is a generalization of linear regression where the outcome variable follows a Poisson distribution.
- Let (Y_i, X_i), $i = 1, \ldots, n$, be pairs of outcome and covariate variables. We assume the following relationship:

$$Y_i \sim Pois(\mu(X_i)), \qquad \log(\mu(X_i)) = \beta_0 + \beta_1 X_i.$$

- We will use the Newton-Raphson method to find the Maximum Likelihood estimate $\hat{\theta} = (\hat{\beta}_0, \hat{\beta}_1)$.
- First, let's write down the likelihood:

$$L(\theta) = \prod_{i=1}^{n} \frac{\mu(X_i)^{Y_i} \exp\left(\mu(X_i)\right)}{Y_i!}.$$

Poisson regression ii

- Take the logarithm:

$$
\begin{aligned}
\ell(\theta) &= \sum_{i=1}^{n} Y_i \log(\mu(X_i)) - \mu(X_i) - \log(Y_i!) \\
&= \sum_{i=1}^{n} Y_i \left(\beta_0 + \beta_1 X_i\right) - \exp(\beta_0 + \beta_1 X_i) - \log(Y_i!) \\
&\propto \sum_{i=1}^{n} Y_i \left(\beta_0 + \beta_1 X_i\right) - \exp(\beta_0 + \beta_1 X_i).
\end{aligned}
$$

Poisson regression iii

- The first-order derivatives are

$$\frac{\partial \ell(\theta)}{\partial \beta_0} = \sum_{i=1}^{n} Y_i - \exp(\beta_0 + \beta_1 X_i),$$
$$\frac{\partial \ell(\theta)}{\partial \beta_1} = \sum_{i=1}^{n} Y_i X_i - X_i \exp(\beta_0 + \beta_1 X_i).$$

Poisson regression iv

- The second-order derivatives are

$$
\begin{aligned}
\frac{\partial^2 \ell(\theta)}{\partial \beta_0^2} &= -\sum_{i=1}^{n} \exp(\beta_0 + \beta_1 X_i), \\
\frac{\partial^2 \ell(\theta)}{\partial \beta_1^2} &= -\sum_{i=1}^{n} X_i^2 \exp(\beta_0 + \beta_1 X_i), \\
\frac{\partial^2 \ell(\theta)}{\partial \beta_0 \partial \beta_1} &= -\sum_{i=1}^{n} X_i \exp(\beta_0 + \beta_1 X_i).
\end{aligned}
$$

Poisson regression v

- We will fit a Poisson regression model to the “famous” horseshoe crab dataset.
  - The main covariate is the weight of female crabs.
  - The outcome is the number of satellite males for that female crab.
  - The dataset can be found in the `asbio` package.

Poisson regression vi

```r
library(asbio)
data(crabs)
y_vec <- crabs$satell
x_vec <- crabs$weight

gradient <- function(theta) {
  mu_x <- exp(theta[1] + theta[2]*x_vec)
  ell_b0 <- sum(y_vec - mu_x)
  ell_b1 <- sum(y_vec*x_vec - x_vec*mu_x)
  return(c(ell_b0, ell_b1))
}
```

Poisson regression vii

```
hessian <- function(theta) {
  mu_x <- exp(theta[1] + theta[2]*x_vec)
  ell_b0b0 <- -sum(mu_x)
  ell_b1b1 <- -sum(x_vec^2*mu_x)
  ell_b0b1 <- -sum(x_vec*mu_x)

  hess <- matrix(c(ell_b0b0, ell_b0b1,
                   ell_b0b1, ell_b1b1),
                 ncol = 2)
  return(hess)
}
```

Poisson regression viii

```r
# Set-up variables
x_current <- c(0, 1)
x_next <- c(0, 0)
tol <- 10^-10
iter <- 0
```

Poisson regression ix

```
while(sum((x_current - x_next)^2) > tol & iter < 100) {
  iter <- iter + 1
  step <- solve(hessian(x_current),
                gradient(x_current))
  update <- x_current - step
  x_current <- x_next
  x_next <- update
}
x_next
```

```
## [1] -0.4284089  0.5893057
```

Poisson regression x

- We can compare this to the built-in function `glm` in R:

```
fit <- glm(satell ~ weight, data = crabs,
           family = poisson)
coef(fit)
```

```
## (Intercept)      weight
##  -0.4284053   0.5893041
```

Poisson regression xi

```r
library(tidyverse)
log_lik <- function(row) {
    mu_vec <- exp(row$beta0 + row$beta1*x_vec)
    sum(dpois(y_vec, lambda = mu_vec, log = TRUE))
}

data_plot <- expand.grid(beta0 = seq(-2, 0, 0.1),
                         beta1 = seq(0, 1, 0.1)) %>%
  purrrlyr::by_row(log_lik, .collate = "cols",
                   .to = "loglik")
```

Poisson regression xii

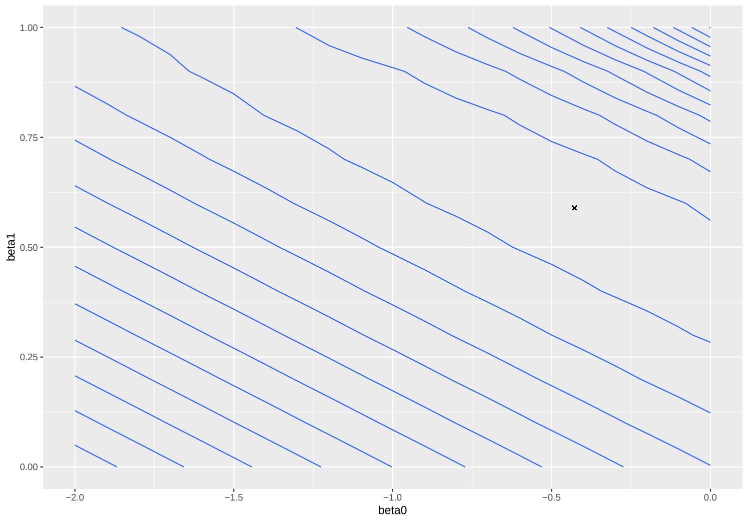
```r
head(data_plot)
```

```
## # A tibble: 6 x 3
##   beta0 beta1 loglik
##   <dbl> <dbl>  <dbl>
## 1  -2       0 -1563.
## 2  -1.9     0 -1515.
## 3  -1.8     0 -1468.
## 4  -1.7     0 -1420.
## 5  -1.6     0 -1373.
## 6  -1.5     0 -1326.
```

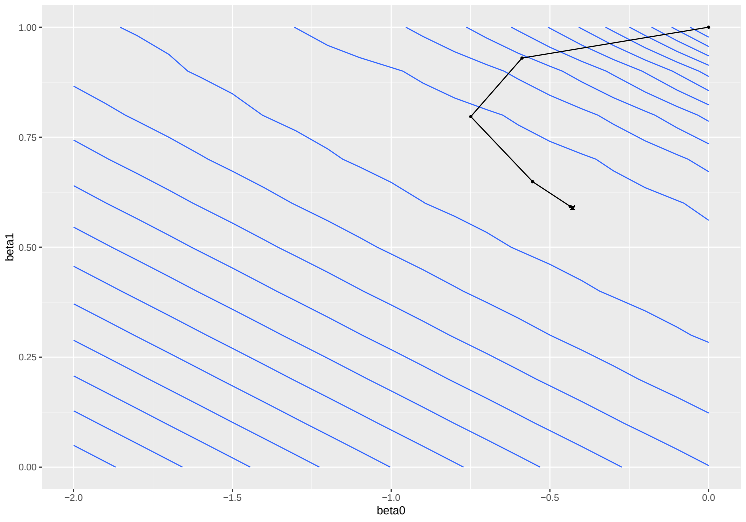
Poisson regression xiii

```r
data_plot %>%
  ggplot(aes(x = beta0, y = beta1,
             z = loglik)) +
  geom_contour() +
  geom_point(x = coef(fit)[1],
             y = coef(fit)[2],
             shape = 4)
```

Poisson regression xiv



Poisson regression xv



Some vocabulary

- In the context of Maximum Likelihood estimation, the gradient and Hessian are given different names.
- The gradient $\nabla \ell(\theta \mid \mathbf{x})$ is called the **score function**.
    - **Exercise**: The expected value of the score function is 0.
- The Hessian $D^2 \ell(\theta \mid \mathbf{x})$ is related to the Fisher information matrix $\mathcal{I}(\theta)$:

$$\mathcal{I}(\theta) = -E\left(D^2 \ell(\theta \mid \mathbf{x})\right).$$

Final remarks

- Newton-Raphson is an important optimisation method.
    - In particular, it is commonly used with *generalized linear models*.
- There exists optimisation methods that are **gradient-free**.
    - See Nelder-Mead and the function `optim`.
- Where to go from here:
    - Gradient descent
    - Regularized regression (e.g. lasso)
    - MM algorithms