ST227: Applications of R in Life Insurance

Optimisation in MLE & Applications in Markovian Models

Dr. Viet Dang

27/02/2021

- Often in mathematical sciences, one is concerned with the following question: given an input domain $\mathcal{O}$ and a criterion function $f : \mathcal{O} \to \mathbb{R}$,
  - Is there a minimum point $x \in \mathcal{O}$?
  - Is this explicitly solvable? If it's not, can we efficiently find it numerically?
  - The nature of optimal point: min? max? inflection? local? global?
- If $\mathcal{O}$ is a subset of $\mathbb{R}^n$, then this procedure often involves:
  - Solving for the candidate solution, which satisfies:

$$\partial_i f = 0, i = 1, \ldots, n.$$

  - Verify the nature of this point by consider the positive or negative-definiteness of the Hessian matrix:

$$D^2 f \triangleq \begin{pmatrix} \partial^2_{11} f & \partial^2_{12} f & \ldots & \partial^2_{1n} f \\ \partial^2_{21} f & \partial^2_{22} f & \ldots & \partial^2_{2n} f \\ \vdots & \vdots & \ddots & \vdots \\ \partial^2_{n1} f & \partial^2_{n2} f & \ldots & \partial^2_{nn} f \end{pmatrix}$$

- Let us consider a toy example where $f : x \to x^2$. Then:

$$f'(x) = 2x, \quad x = 0$$

$$f''(x) = 2, \quad f''(x)|_{x=0} = 2 > 0.$$

The candidate optimiser is $x = 0$. The second derivative test implies that it is a local minimum.

- We **cannot** conclude that it is a global minimum without further considerations.
- If the second derivative test is 0, we cannot make a conclusion.
- Example: $g(x) = x^4$, $h(x) = -x^4$ and $k(x) = x^3$ all have $x = 0$ as the unique candidate - yet the nature of this point is different.

- In practice - it is extremely difficult to solve $f' = 0$ analytically.
- Example: $\mathcal{O} = (0, \infty)$ and $f : x \to \frac{1}{2}x^2 \ln(x) - \frac{1}{4}x^2 - 2x$. It has a (exercise!) unique global minimiser. The first order condition is:

$$f'(x) = x \ln(x) - 2 = 0.$$

  Can you solve it "by hand"?
- Potential numerical approaches:
  - Interval Bisection
  - Newton-Raphson algorithm
- You can write your own algorithm, or ...

Optimisation

- `optim` is a general-purpose optimisation routine in R.
- General syntax: `optim(par,fn)`, where:
  - `par`: initial values of parameters;
  - `fn`: a function to be minimised.

- Additional fine tuning is available. We will introduce them as we need them. Type `?optim` for its documentation. Some additional arguments of interest are:
  - `method`: Method of optimisation. Default value is the "Nelder-Mead" method.
  - `lower` and `upper`: bounds for the candidate solution.
- Available methods inlucde: "Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", and "Brent". Their algorithms are beyond the scope of ST227.

Optimisation | Example: 1-dimensional Optimisation

- Consider the toy example: $f : x \to x^2$. Use 1 as the initial value.

```
sqr <- function(x){x^2}
optim(par=1,sqr)
```

- The output is a list. We are interested in the following items: `$par = -8.881784e-16` is the minimiser, and `$value = 7.888609e-31` is the the function value at this minimiser.
- Be careful of the warning, though: "Warning in optim(par = 1, sqr): one-dimensional optimization by Nelder-Mead is unreliable: use"Brent" or optimize() directly"
- The default method is designed for multi-dimensional optimisation. Let us use the Brent method.
- The Brent method requires `lower` and `upper` arguments.

```
optim(par=1,sqr,method="Brent",lower=-10,upper=10)
```

- What if a function does not have a maximum/minimum?

```
id <- function(x){x}
optim(par=0,id)
optim(par=0,id,method="Brent",lower=-10,upper=10)
```


Does this result make sense?

- Use numeric optimisation with reservation.
- Exercise: Use `optim` with `method = "Brent"` to minimise on $(0, \infty)$ the function:

$$f : x \to \frac{1}{2}x^2 \ln(x) - \frac{1}{4}x^2 - 2x$$

- Consider the function: $f : (x, y) \to x^2 + y^2$, which has a global minimum at the origin.
- The underlying syntax of `optim` requires f to have a **single vectorial argument**, instead of two real-valued arguments. That is, instead of:


```
f <- function(x,y){
  x^2 + y^2
}
```


we will need:

```
f <- function(x){
  x[1]^2 + x[2]^2
}
```


- For a multi-dimensional optimisation problem, we can use the default Nelder-Mead method:

```
optim(par = c(1,1), f)
```

General Warnings

- How does one choose initial values for an optimisation algorithm? There are no definitive answers. Some approaches are:
  - Numerical Experimentation,
  - Qualitative insight of the problem, e.g. constrained optimisation,
  - Inspiration from similar, but simpler problems,
  - Guesses.

- Minimisation algorithms actually search for a turning point.
  - No guarantee on the uniqueness/nature of the turning point.
  - Might get trapped in local optima.
- Definitely NOT a replacement for mathematical insights.
- Widespread use in Statistics: Maximum Likelihood Estimation, Least Square Estimation (e.g. Linear Regression and GLM), Stochastic Gradient Descent (Deep Learning, Neural Network).

Maximum-Likelihood Estimation

- Suppose that we observe a sample $(X_1, ..., X_n)$, which depend on a common parameter $\theta \in \mathbb{R}^d$, where $X_i \sim f_i(x, \theta)$.
- Questions: 1. How to estimate θ based on $(X_1, ..., X_n)$? and 2. How sure are we of this estimate?
- There are various approaches:
  - Method of Moments;
  - Least Square Estimation;
  - Maxmium Likelihood Estimation
- The likelihood $L(\theta; X)$ and log-likelihood $l(\theta; X)$ are defined as follows:

$$L(\theta; X) = \prod_{i=1}^{n} f_i(x_i, \theta) \tag{1a}$$

$$l(\theta; X) = \sum_{i=1}^{n} \log\Big(f_i(x_i, \theta)\Big) \tag{1b}$$

- The maximum likelihood estimator is defined as the solution of:

$$\max_{\theta} l(\theta; X),$$

  which is an optimisation problem.
- For most distributions seen in ST102, this problem is explicitly solvable. In general, though, we will have to solve it numerically.

- A random variable X follows a $\mathcal{N}(\mu, \sigma^2)$ if it has density:

$$f(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in \mathbb{R}.$$

- Given an i.i.d. normal sample $X = (X_1, \ldots, X_n)$, let us estimate μ and σ^2 using numerical MLE method.
- Suppose $n = 100$. We will simulated some dummy date for this task:

```
X <- rnorm(n=100,mean=1.5,sd=5)
```

- Maximising log-likelihood means minimising its negative. That is, we are minimising the function:

$$\sum_{i=1}^{n} -\log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}\right)$$

- There are at least three ways I can think of to define this sum, with varying level of difficulty:
  - 1: using for loop and elementary calculations only.
  - 2: utilising R's vectorisation.
  - 3: utilising R's built-in statistical functions.

- The first method is a *low-level method*. If you have studied a "traditional" programming course before, this is probably how you define a sum:

```
nLL <- function(param){
  mu <- param[1]
  sigma2 <- param[2]
  out <- 0
  for(i in 1:length(X)){
    out <- out - log((2*pi*sigma2)^{-1/2}*exp(-(X[i]-mu)^2/(2*sigma2)))
  }
  return(out)
}
```

- This displays all the fine details involved.
- Very reminiscent of low-level languages e.g. C.
- This is **not** recommended. Too much bookkeeping obfuscates the essential ideas.

Maximum Likelihood Estimation | Example: Normal Distribution

- The second method employs R's vectorised calculations. Those of you with some Matlab experience will be familiar with this.


```
nLL <- function(param){
  mu <- param[1]
  sigma2 <- param[2]
  - sum(
    log((2*pi*sigma2)^{-1/2}*exp(-(X-mu)^2/(2*sigma2)))
  )
}
```


- Why does this work? Due to vectorisation, the code `X-mu` actually returns:

$$(X_1 - \mu, X_2 - \mu, \ldots., X_n - \mu).$$

- The code `(X-mu)^2` returns: $\left((X_1 - \mu)^2, ..(X_n - \mu)^2\right)$.
- Tracing along all the calculations, we see that the code `log((2*pi*sigma2)^{-1/2}*exp(-(X-mu)^2/(2*sigma2)))` returns the vector $(\log f(X_1; \mu, \sigma^2), .., \log f(X_n; \mu, \sigma^2))$. Summing this gives you the log-likelihood function.

Maximum Likelihood Estimation | Example: Normal Distribution

- The third method is faster still:

```
nLL <- function(params){
  mu <- params[1]
  sigma2 <- params[2]
  - sum(
    dnorm(X,mean=mu,sd=sqrt(sigma2),log=TRUE)
  )
}
```

- `dnorm` stands **d**istribution function of the **norm**al distribution (in this case, the value of the density).
- The optional argument `log=TRUE` returns the log of the density - which is convenient for log-likelihood calculations.
- This makes your codes succinct and human-readable and idiomatic to the R language.

- The **negative** log-likelihood has been defined. Let us optimise it.

```
optim(par = c(1,1),nLL)
```

- Above, I just guessed the initial parameter values.
- Difficult optimisation problems tend to be numerically unstable. That is, if initial parameters are not chosen carefully:
  - the algorithm might not converge at all, or
  - might be very slow even when it converges.
- We want to choose a "good" initial value. One option is to use the Method of Moment Estimator.


```
optim(par = c(mean(X),var(X)),nLL)
```

- The previous example seems trivial:
  - The explicit MLE estimator is known.
  - MLE and Method of Moments estimator coincide.
- Let's consider a different example. Suppose that X follows a Gamma distribution with parameters (α, β), i.e.

$$f_X(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}, \quad x \geq 0.$$

- We create some dummy data for this task:

```
Y <- rgamma(100,shape=10, rate=10)
```


- Exercise: Derive the method of moment estimators for α and β. Use them as initial values, retro-fit an MLE estimator on the data Y.

Maximum Likelihood Estimation | Example: Weibull Distribution (Not Examinable)

- Say that a variable X follows a $\mathrm{Weibull}(\lambda, k)$ distribution if it has density:

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- We observe a sample $(X_1, \ldots X_n)$ which are i.i.d. $\mathrm{Weibull}(\lambda, k)$. Let us estimate λ and k.
- Suppose $n = 100$. Let's simulate some dummy data for this task


```
dummydata <- rweibull(100,shape=10,scale=20)
```

- Maximising log-likelihood means minimising its negative. The negative log-likelihood can be defined as:

```
negLL <- function(param){
  shape <- params[1]; scale <- prams[2]
  -sum(
   log(scale/shape*(dummyData/shape)^{scale-1}*exp(-(dummyData/shape)^scale))
  )
}
```


- R has a shorthand for this. Use the dweibull function to calculate the **d**istribution function of the **weibull** distribution


```
negLL <- function(params){
  -sum(dweibull(dummydata,shape=params[1],scale=params[2],log=TRUE))
}
```

Maximum Likelihood Estimator | Example: Weibull Distribution (Not Examinable) (3)

- Let's optimise this:

```
optim(par =c(10,10),fn = negLL)
```


- The output is $\lambda = 10.25368$ and $k = 19.99069$. This is pretty close to the original.
- How did we choose the initial values?
  - From other estimation methods e.g. Method of Moments.
  - Taking "shots in the dark", i.e. guessing.