PC6b - Optimisation

Question 1

Let $X_1, ..., X_n$ denote a random sample from the Poisson distribution with parameter θ, i.e. $X_i \sim \text{Poi}(\theta)$. We want to find the Maximum Likelihood Estimate (MLE) for this parameter. This is the value that maximizes the joint distribution function

$$\begin{aligned} f(x;\theta) &= \prod_{i=1}^{n} f(x_i;\theta) \\ &= \prod_{i=1}^{n} \frac{\theta^{x_i}}{x_i!} e^{-\theta} \\ &= e^{-n\theta} \frac{\theta^{\sum_{i=1}^{n} x_i}}{\prod_{i=1}^{n} x_i!}. \end{aligned}$$

The log-likelihood is defined as the log of $f(x;\theta)$ considered as function of the unknown parameter θ:

$$\begin{aligned} l(\theta) &= -n\theta + \sum_{i=1}^{n} x_i log(\theta) - log\left(\prod_{i=1}^{n} x_i!\right) \\ &\propto -n\theta + \sum_{i=1}^{n} x_i log(\theta), \end{aligned}$$

where $\propto$ is the 'proportional to' symbol. For maximizing with respect to the parameter θ, this last term is irrelevant. The derivative of the log-likelihood is given by:

$$\frac{\mathrm{d}l(\theta)}{\mathrm{d}\theta} = -n + \sum_{i=1}^{n} \frac{x_i}{\theta}.$$

This derivative is also called the score function.

Code the value of the log-likehood and the derivative in R (below you can find the headers of these functions). On Canvas, you can find a file `countdata.dat` that contains data. Determine the MLE value for θ.

```r
logLik<-function(theta,x) {
# CODE THE LOG-LIKELIHOOD HERE
}

gr_logLik<-function(theta,x) {
# CODE THE SCORE HERE
}

x <- scan("countdata.dat")
n <- length(x)

cat('Estimated parameters\n')
result1 <- optim(4,logLik,gr=NULL,x,method="Brent",
                 lower=0,upper=10,control=list(fnscale=-1))
print(result1$par)
result2 <- optim(4,logLik,x=x,gr_logLik, method="BFGS",
                 control=list(fnscale=-1,maxit=300,trace=TRUE,REPORT=5))
names(result2)  # all properties of the object result2
print(result2$par)
```

Question 2 [Exercise 4 of Section 12.8.4 of Jones et al.]

The Rosenbrock function is a commonly used test function, given by

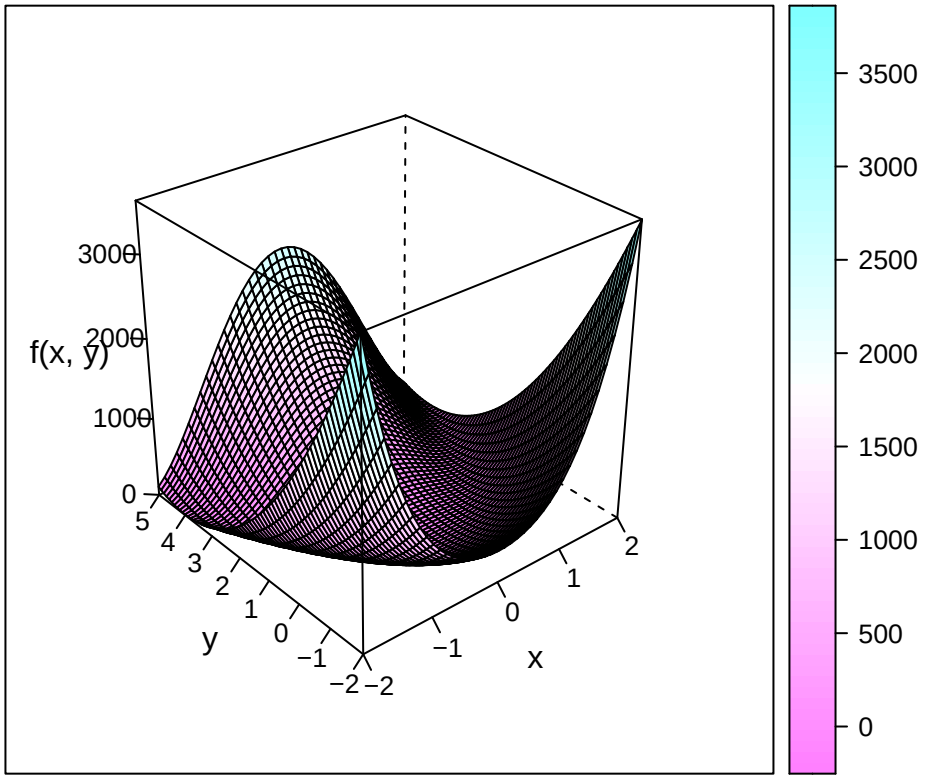
$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2.$$

Below, you can find a plot of the function in the region $[-2, 2] \times [-2, 5]$. It has a single global minimum at (1, 1).

```
# program spuRs/resources/scripts/Rosenbrock.r

Rosenbrock <- function(x) {
  g <- (1 - x[1])^2 + 100*(x[2] - x[1]^2)^2
  g1 <- -2*(1 - x[1]) - 400*(x[2] - x[1]^2)*x[1]
  g2 <- 200*(x[2] - x[1]^2)
  g11 <- 2 - 400*x[2] + 1200*x[1]^2
  g12 <- -400*x[1]
  g22 <- 200
  return(list(g, c(g1, g2), matrix(c(g11, g12, g12, g22), 2, 2)))
}

x <- seq(-2, 2, .1)
y <- seq(-2, 5, .1)
xyz <- data.frame(matrix(0, length(x)*length(y), 3))
names(xyz) <- c('x', 'y', 'z')
n <- 0
for (i in 1:length(x)) {
  for (j in 1:length(y)) {
    n <- n + 1
    xyz[n,] <- c(x[i], y[j], Rosenbrock(c(x[i], y[j]))[[1]])
  }
}
library(lattice)
print(wireframe(z ~ x*y, data = xyz, scales = list(arrows = FALSE),
  zlab = 'f(x, y)', drape = T))
```



Use the function `contour` to form a contour-plot of the Rosenbrock function over the region $[-2, 2] \times [-2, 5]$.

Next modify `ascent` so that it plots each step on the contour-plot, and use it to find the *minimum* of the Rosenbrock function (that is, the maximum of $-f$), starting at $(0, 3)$. Use a tolerance of 10^{-9} and increase the maximum number of iterations to at least $10,000$.

Now use `newton` to find the minimum (no need to use $-f$ in this case). Plot the steps made by the Newton method and compare them with those made by the steepest descent method.

P.S. You can find a basic R-script for this exercise on Canvas!

Question 3 [Exercise 4 of Section 12.8.6 of Jones et al.]

A simple way of using local search techniques to find a global maximum is to consider several different starting points, and hope that for one of them its local maximum is in fact the global maximum. If you have no idea where to start, then randomisation can be used to choose the starting point.

Consider the function

$$f(x,y) = -(x^2+y^2-2)(x^2+y^2-1)(x^2+y^2)(x^2+y^2+1)(x^2+y^{2+}) \\ \times (2-\sin(x^2-y^2)\cos(y-\exp(y)))\,.$$

It has several local maxima in the region $[-1.5, 1.5] \times [-1.5, 1.5]$. Using several randomly chosen starting points, use steepest ascent to find all of the local maxima of f, and thus the global maximum. You can use the command `runif(2, -1.5, 1.5)` to generate a random point (x, y) in the region $[-1.5, 1.5] \times [-1.5, 1.5]$.

Warning: if y is too large then $\exp(y)$ is `Inf` and $\cos(y - \exp(y))$ will return `NaN`. This can cause problems when trying to maximise f. A solution is to just ignore the term $\cos(y - \exp(y))$ when this happens. That is, code up the function f as follows:

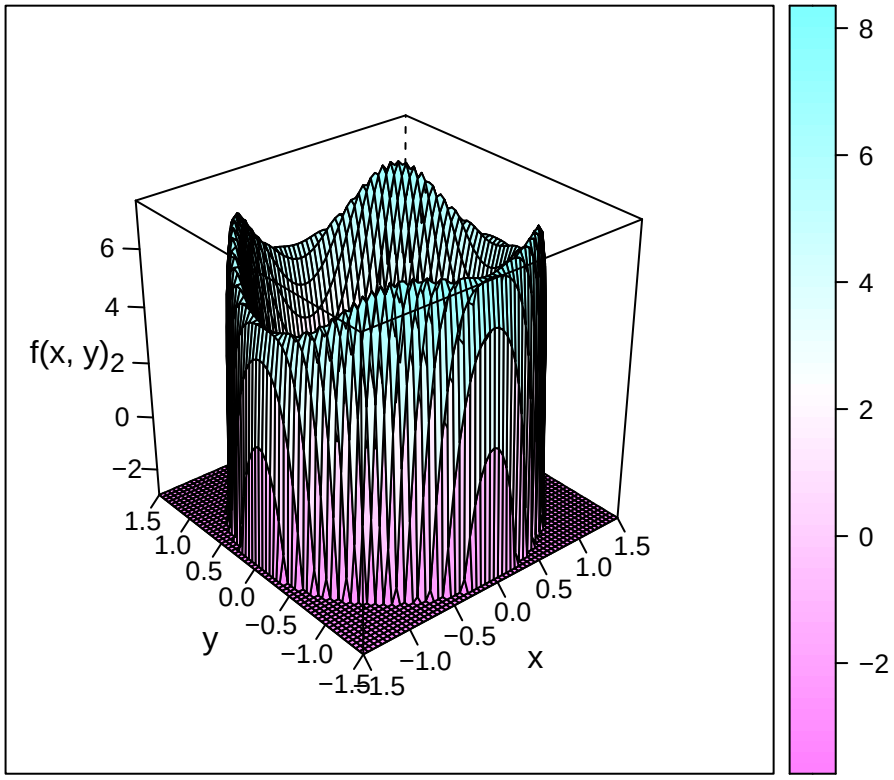
```
f <- function(x) {
  # x is a vector of length 2
  s <- sum(x^2)
  z <- -(s-2)*(s-1)*s*(s+1)*(s+2)*
    (2-sin(x[1]^2-x[2]^2)*cos(x[2]-exp(x[2])))
  if (is.nan(z)) {
    z <- -(s-2)*(s-1)*s*(s+1)*(s+2)*2
  }
  return(z)
}
```

∇f suffers analogous problems, which can be dealt with the same way.

A picture of f is given below. Note that f has been truncated below at -3.

```
x <- seq(-1.5, 1.5, .05)
y <- seq(-1.5, 1.5, .05)
xyz <- data.frame(matrix(0, length(x)*length(y), 3))
names(xyz) <- c('x', 'y', 'z')
n <- 0
for (i in 1:length(x)) {
  for (j in 1:length(y)) {
    n <- n + 1
    xyz[n,] <- c(x[i], y[j], max(-3,f(c(x[i], y[j]))))
  }
```

```
}
library(lattice)
print(wireframe(z ~ x*y, data = xyz, scales = list(arrows = FALSE),
                zlab = 'f(x, y)', drape = T))
```



P.S. Again, you can find a basic R-script for this exercise on Canvas!