1 Optimization using the `optim` function

Consider a function $f(\mathbf{x})$ of a vector $\mathbf{x}$. Optimization problems are concerned with the task of finding $\mathbf{x}^\star$ such that $f(\mathbf{x}^\star)$ is a local maximum (or minimum). In the case of maximization,

$$\mathbf{x}^\star = \text{argmax } f(\mathbf{x})$$

and in the case of minimization,

$$\mathbf{x}^\star = \text{argmin } f(\mathbf{x})$$

Most statistical estimation problems are optimization problems. For example, if f is the likelihood function and $\mathbf{x}$ is a vector of parameter values, then $\mathbf{x}^\star$ is the **maximum likelihood estimator (MLE)**, which has many nice theoretical properties.

When f is the posterior distribution function, then $\mathbf{x}^\star$ is a popular bayes estimator. Other well known estimators, such as the *least squares estimator* in linear regression are optimums of particular objective functions.

We will focus on using the built-in R function `optim` to solve **minimization** problems, so **if you want to maximize you must supply the function multiplied by -1**. The default method for `optim` is a derivative-free optimization routine called the Nelder-Mead simplex algorithm. The basic syntax is

```
optim(init, f)
```

where `init` is a vector of initial values you must specify and `f` is the objective function. There are many optional argument– see the help file details
If you have also calculated the derivative and stored it in a function `df`, then the syntax is

```
optim(init, f, df, method="CG")
```

There are many choices for `method`, but `CG` is probably the best. In many cases the derivative calculation itself is difficult, so the default choice will be preferred (and will be used on the homework).

With some functions, particularly functions with many minimums, the initial values have a great impact on the converged point.

1.2 One dimensional examples

Example 1: Suppose $f(x) = e^{-(x-2)^2}$. The derivative of this function is

$$f'(x) = -2(x-2)f(x)$$

Since $f(x)$ is strictly positive, $f'(x) = 0$ is clearly satisfied at , so $\mathbf{x}^\star = 2$, which is a maximum. Now we solve this using R:

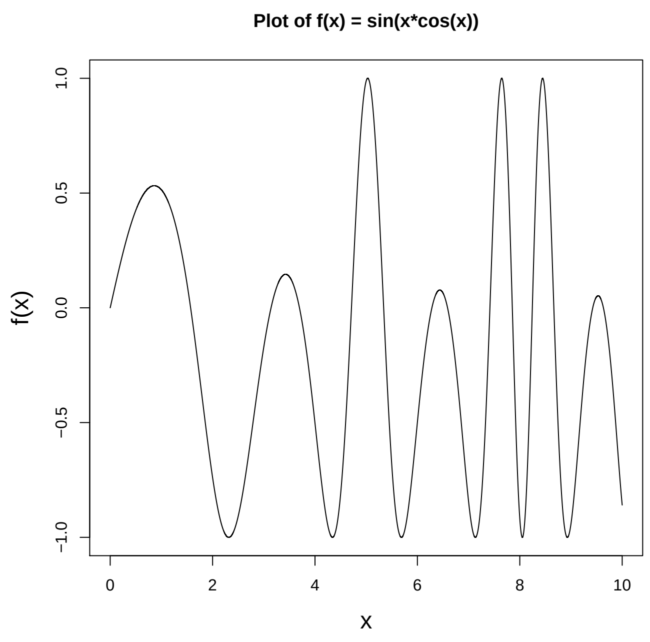

Figure 1: $\sin(x\cos(x))$ for $x \in (0, 10)$

```
# we supply negative f, since we want to maximize.
f <- function(x) -exp(-( (x-2)^2 ))

######### without derivative
# I am using 1 at the initial value
# $par extracts only the argmax and nothing else
optim(1, f)$par

######### with derivative
df <- function(x) -2*(x-2)*f(x)
optim(1, f, df, method="CG")$par
```

Notice the derivative free method appears more sensitive to the starting value and gives a warning message. But, the converged point looks about right when the starting value is reasonable.

Example 2: Suppose $f(x) = \sin(x\cos(x))$. This function has many local optimums. Let's see how `optim` finds minimums of this function which appear to occur around 2.1, 4.1, 5.8, and several others.

```
f <- function(x) sin(x*cos(x))
optim(2, f)$par
optim(4, f)$par
optim(6, f)$par
optim(8, f)$par
```

It appears that the algorithm finds the nearest minimum quite accurately.

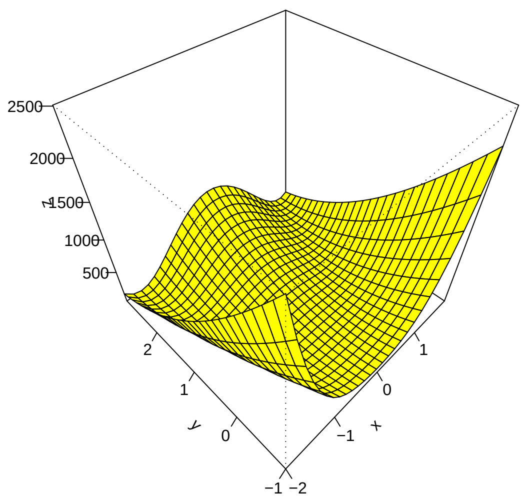

Figure 2: Rosenbrock function for $x \in (-2, 2)$ and $y \in (-1, 3)$.

1.3 Two dimensional examples

Example 3: Let $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$, which is called the Rosenbrock function. Let's plot the function.

```
f <- function(x1,y1) (1-x1)^2 + 100*(y1 - x1^2)^2
x <- seq(-2,2,by=.15)
y <- seq(-1,3,by=.15)
z <- outer(x,y,f)
persp(x,y,z,phi=45,theta=-45,col="yellow",shade=.00000001,ticktype="detailed")
```

This function is strictly positive, but is 0 when $y = x^2$, and $x = 1$, so $(1, 1)$ is a minimum. Let's see if `optim` can figure this out. When using `optim` for multidimensional optimization, the input in your function definition must be a single vector.

```
f <- function(x) (1-x[1])^2 + 100*(x[2]-x[1]^2)^2

# starting values must be a vector now
optim( c(0,0), f )$par
[1] 0.9999564 0.9999085
```

Example 4: Let $f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$, which is called Himmelblau's function. The function is plotted from belowin figure 3 by:

```
f <- function(x1,y1) (x1^2 + y1 - 11)^2 + (x1 + y1^2 - 7)^2
x <- seq(-4.5,4.5,by=.2)
y <- seq(-4.5,4.5,by=.2)
z <- outer(x,y,f)
persp(x,y,z,phi=-45,theta=45,col="yellow",shade=.65 ,ticktype="detailed")
```

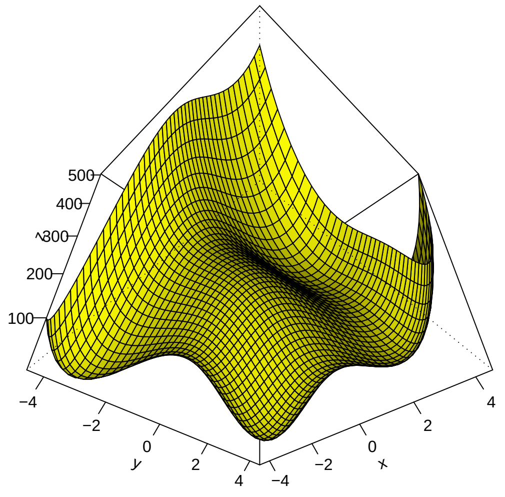

Figure 3: Himmelblau function for $x \in (-4, 4)$ and $y \in (-4, 4)$.

There appear to be four "bumps" that look like minimums in the realm of (-4,-4), (2,-2), (2,2) and (-4,4). Again this function is strictly positive so the function is minimized when $x^2 + y - 11 = 0$ and $x + y^2 - 7 = 0$.

```
f <- function(x) (x[1]^2 + x[2] - 11)^2 + (x[1] + x[2]^2 - 7)^2
optim(c(-4,-4), f)$par
[1] -3.779347 -3.283172

optim(c(2,-2), f)$par
[1]  3.584370 -1.848105

optim(c(2,2), f)$par
[1] 3.000014 2.000032

optim(c(-4,4),f)$par
[1] -2.805129  3.131435
```

which are indeed the true minimums. This can be checked by seeing that these inputs correspond to function values that are about 0.

2 Using `optim` to fit a probit regression model

Suppose we observe a binary variable $y_i \in \{0,1\}$ and d covariates $x_{1,i}, ..., x_{d,i}$ on n units. We assume that

$$y_i \sim \text{Bernoulli}(p_i)$$

where

$$p_i = \Phi(\beta_0 + \beta_1 x_{1,i} + ... + \beta_d x_{d,i}) \tag{1}$$

where Φ is the standard normal CDF. This is called the *probit* regression model and is considered an alternative to logistic regression. Notice this is basically the same as logistic regression except the *link function* is $\Phi^{-1}(p)$ instead of $log(p/(1-p))$. For notational simplicity, define

$$\mathbf{X}_i = (1, x_{1,i}, ..., x_{d,i})$$

that is, the row vector of covariates for subject i. Also define

$$\beta = (\beta_0, \beta_1, ..., \beta_d)'$$

the column vector of regression coefficients. Then (1) can be re-written as

$$p_i = \Phi(\mathbf{X}_i \beta) \tag{2}$$

Instead of doing maximum likelihood estimation, we will place a multivariate normal prior on β. That is $\beta \sim N(0, cI_{d+1})$, where I_{d+1} is the $d+1$ dimensional identity matrix, and $c = 10$. **This means that we are assuming that the β's are each independent N(0,10) random variables**.

The goal of this exercise is to write a program that determines estimated coefficients, $\hat{\beta}$ for this model by maximizing the posterior density function. This estimator is called the *posterior mode* or the *maximum aposteriori* (MAP) estimator. Once we finish writing this program we will fit the probit regression model to estimate the outcome of the 1988 presidential based on the ctools dataset.

Step 1: Determine the log-likelihood for β

We know that $y_i \sim \text{Bernoulli}(p_i)$. This means

$$p(y_i) = p_i^{y_i}(1-p_i)^{1-y_i}$$

i.e. $p(1) = p_i^1 + (1-p_i)^0 = p_i$, and similarly $p(0) = 1 - p_i$. Since we know that p_i has the structure defined by (2), this becomes

$$p(y_i|\beta) = \Phi(\mathbf{X}_i\beta)^{y_i}(1-\Phi(\mathbf{X}_i\beta))^{1-y_i}$$

which is the likelihood for subject i. So, **the log-likelihood for subject i is**

$$\begin{aligned}\ell_i(\beta) &= \log(p(y_i|\beta)) \\ &= y_i \log\left(\Phi(\mathbf{X}_i\beta)\right) + (1-y_i)\log\left(1-\Phi(\mathbf{X}_i\beta)\right) \\ &= \log\left(1-\Phi(\mathbf{X}_i\beta)\right) + y_i \log\left(\frac{\Phi(\mathbf{X}_i\beta)}{1-\Phi(\mathbf{X}_i\beta)}\right)\end{aligned}$$

By independence, the log-likelihood for the entire sample is

$$\begin{aligned}L(\beta) &= \sum_{i=1}^{n} \ell_i(\beta) \\ &= \sum_{i=1}^{n} \log\left(1-\Phi(\mathbf{X}_i\beta)\right) + y_i \log\left(\frac{\Phi(\mathbf{X}_i\beta)}{1-\Phi(\mathbf{X}_i\beta)}\right)\end{aligned}$$

2 Determine the log-posterior

Remember the posterior density is $p(\beta|y)$, and that

$$p(\beta|y) \propto \underbrace{\exp(L(\beta))}_{\text{likelihood}} \cdot \underbrace{p(\beta)}_{\text{prior}}$$

therefore the log-posterior is

$$\log(p(\beta|y)) = \text{const} + L(\beta) + \log(p(\beta))$$

We want to optimize $\log(p(\beta|y))$. The constant term does not effect the location of the maximum, so we need only maximize $L(\beta) + \log(p(\beta))$. Recall that $\beta_0, ..., \beta_d$ are assumed iid $N(0, 10)$, so

$$\begin{aligned}p(\beta_j) &= \frac{1}{\sqrt{20\pi}}\exp\left(-\beta_j^2/20\right) \\ &\propto \exp\left(-\beta_j^2/20\right)\end{aligned}$$

for each $j = 0, ..., d$. Therefore

$$\log(p(\beta_j)) = \text{const} - \beta^2/20$$

Again, dropping the constant, and using the fact that the β's are independent,

$$\log(p(\beta)) = \sum_{j=0}^{d} -\beta^2/20 = -\frac{1}{20}\sum_{j=0}^{d}\beta_j^2$$

Finally the log-posterior is

$$\begin{aligned}\log(p(\beta|y)) &= L(\beta) + \log(p(\beta)) \\ &= \sum_{i=1}^{n} \log\left(1 - \Phi(\mathbf{X}_i\beta)\right) + y_i \log\left(\frac{\Phi(\mathbf{X}_i\beta)}{1 - \Phi(\mathbf{X}_i\beta)}\right) - \frac{1}{20}\sum_{j=0}^{d}\beta_j^2\end{aligned}$$

Step 3: Write a generic function that minimizes the negative log posterior

The following R code does this.

```
# Y is the binary response data.
# X is the covariate data, each row is the response data
# for a single subject. If an intercept is desired, there
# should be a column of 1's in X
# V is the prior variance (10 by default)
# when V = Inf, this is maximum likelihood estimation.
posterior.mode <- function(Y, X,V=10)
{

   # sample size
   n <- length(Y)

   # number of betas
   d <- ncol(X)

   # the log posterior as a function of the vector beta for subject i data
   log.like_one <- function(i, beta)
   {

       # p_{i}, given X_{i} and beta
       Phi.Xb <- pnorm( sum(X[i,] * beta) )

       return( log(1 - Phi.Xb) + Y[i]*log( Phi.Xb/(1 - Phi.Xb) )  )

    }

    # log likelihood of the entire sample
    loglike <- function(beta)
    {
       L <- 0
       for(ii in 1:n) L <- L + log.like_one(ii,beta)
       return(L)
    }

    # *negative* log posterior of the entire sample
    log.posterior <- function(beta) -loglike(beta) + (1/(2*V))*sum(beta^2)

    # return the beta which optimizes the log posterior.
    # initial values are arbitrarily chosen to be all 0's.
    return( optim( rep(0,d), log.posterior)$par )

}
```

We will generate some fake data such that

$$P(Y_i = 1) = \Phi\left(.43 - 1.7x_{i,1} + 2.8x_{i,2}\right)$$

to demonstrate that this works.

```
# The covariates are standard normal
X <- cbind( rep(1, 20), rnorm(20), rnorm(20) )
Y <- rep(0,20)
Beta <- c(.43, -1.7, 2.8)
for(i in 1:20)
{

  # p_i
  pp <- pnorm( sum(Beta * X[i,]) )
  if( runif(1) < pp ) Y[i] <- 1

}

# fit the model with prior variance equal to 10
posterior.mode(Y,X)
[1]  0.8812282 -2.0398078  2.4052765

# very small prior variance
posterior.mode(Y,X,.1)
[1]  0.07609242 -0.47507906  0.44481367

# maximum likelihood estimate
posterior.mode(Y,X,Inf)
[1]  0.8766873 -2.0678152  2.5047624
```

We can see that when the prior variance is smaller, the coefficients are shrunken more toward 0. When the prior variance is very large, we get the smallest amount of shrinkage.

Step 4: 1988 elections data analysis

We have data from 8 surveys where the responses are '1' if the person voted for George Bush and '0' otherwise. We have demographic predictors: age, education, female, black. Each of these predictors are **categorical** with 4, 4, 2, and 2 levels, respectively, so we effectively have 10 predictors (viewing the indicator of each level as a distinct predictor).

For each of the 8 polls, we will use our function to estimate β, and therefore the predicted probabilities for each "class" of individuals– a class being defined as a particular configuration of the predictor variables. Call this quantity p_c:

$$p_c = \Phi(\mathbf{X}_c \hat{\beta})$$

Since not all classes are equally prevalent in the population, we obtain N_c, the number of people in each class from the 1988 census data, and calculate

$$\hat{p}_{\text{bush}} = \frac{\sum_{c=1}^{K} p_c N_c}{\sum_{c=1}^{K} N_c}$$

which a weighted average and is a reasonable estimate of the proportion of the population that actually voted for George Bush.

First we read in the data:

```
library('foreign')
A <- read.dta('polls.dta')
D <- read.dta('census88.dta')
```

Since we will need it for each poll, we might as well get N_c for each class first. There are $4 \cdot 4 \cdot 2 \cdot 2 = 64$ different classes, comprised of all possible combinations of $\{1,2,3,4\}_{age} \times \{1,2,3,4\}_{edu} \times \{0,1\}_{sex} \times \{0,1\}_{race}$.

```
# storage for the class labels and total numbers
C <- matrix(0, 64, 2)
k <- 1
for(age in 1:4)
{

   for(edu in 1:4)
   {

      for(female in 0:1)
      {

         for(black in 0:1)
         {
            # which rows of D correspond to this 'class'
            w <- which( (D$edu==edu) & (D$age==age)
            & (D$female==female) & (D$black==black) )

            # the 'label' for this class
            class <- 1000*age + 100*edu + 10*female + black

            C[k,] <- c(class, sum(D$N[w]))
            k <- k + 1

         }

      }

   }

}
```

Now for each poll we estimate β, and then calculate $\hat{p}_{\text{bush}}$. There are 12 predictors, that are the indicators of a particular level of the demographic variables. For example, $x_{i,1}$ is the indicator that subject i is in education group 1. There is no intercept, because it would not be identified.

```
# the labels for the 8 polls
surveys <- unique(A$survey)

# loop through the surveys
for(jj in 2:8)
{

   poll <- surveys[jj]

   # select out only the data corresponding to the jj'th poll
   data <- A[which(A$survey==poll),]

   # Each covariate is the indicator of a particular level of
   # the predictors
   X <- matrix(0, nrow(data), 12)
   X[,1] <- (data$age==1); X[,2] <- (data$age==2);
   X[,3] <- (data$age==3); X[,4] <- (data$age==4);
   X[,5] <- (data$ed==1); X[,6] <- (data$ed==2);
   X[,7] <- (data$ed==3); X[,8] <- (data$ed==4);
   X[,9] <- (data$female==0); X[,10] <- (data$female==1);
   X[,11] <- (data$black==0); X[,12] <- (data$black==1);

   # response values
   Y <- data$bush
```

```
# take out NAs
w <- which(is.na(Y)==1)
Y <- Y[-w]
X <- X[-w,]

# Parameter estimates
B <- posterior.mode(Y,X)

# storage for each class's predicted probability
p <- rep(0, 64)

for(j in 1:64)
{

   # this class label
   class <- C[j,1]

   ### determine the demographic variables from the class
   # the first digit in class is age
   age <- floor(class/1000)

   # the second digit in class is ed
   ed <- floor( (class%%1000)/100 )

   # the third digit in class is sex
   female <- floor( (class - 1000*age - 100*ed)/10 )

   # the fourth digit in class
   black <- (class - 1000*age - 100*ed - 10*female)

   # x holds the predictor values for this class
   x <- rep(0, 12)
   x[age] <- 1
   x[ed+ 4] <- 1
   if( female == 1 ) x[10] <- 1 else x[9] <- 1
   if( black == 1 ) x[12] <- 1 else x[11] <- 1

   # predicted probability
   p[j] <- pnorm( sum(x*B) )

}

# the final estimate for this poll
P <- sum( C[,2]*p )/sum(C[,2])

# print the results to the screen
print( sprintf("Poll %s estimates %.5f will vote for George Bush", poll, P) )
```

```
}
```

```
[1] "Poll 9152 estimates 0.53621 will vote for George Bush"
[1] "Poll 9153 estimates 0.54021 will vote for George Bush"
[1] "Poll 9154 estimates 0.52811 will vote for George Bush"
[1] "Poll 9155 estimates 0.53882 will vote for George Bush"
[1] "Poll 9156a estimates 0.55756 will vote for George Bush"
[1] "Poll 9156b estimates 0.57444 will vote for George Bush"
[1] "Poll 9157 estimates 0.55449 will vote for George Bush"
[1] "Poll 9158 estimates 0.54445 will vote for George Bush"
```

The final results of the 1988 election had George Bush with 53.4% of the vote and Michael Dukakis with 45.6%, with the remaining 1% apparently voting for third party candidates. Most of the polls seem to all slightly overestimate the proportion who would vote for Bush.