Stat 243
Problem set 8

A. Strand, ID: 540441, GitHub: AndreasStrand

December 7, 2015

Problem 1

(a) I would do the standard linear regression method of the comparison using the `lm()` function in R, and the corresponding prediction interval can be found using `predict.lm()`. Denote the new regression variation NR and the standard regression by SR. The method of comparison I suggest is

1. Create a $s \times n \times m$ matrix with s sets of n observations on m covariates including the response. Assign each set to one out of a few types of underlying multivariate distributions $\mathbf{t}$ of m dimensions, such that there is about the same number of sets representing each type.
2. For each set choose the m parameters $\boldsymbol{\beta}$ and the entries of the covariance matrix Σ_s in a range of real values. For Σ_s we draw a upper triangular matrix and multiply it by its transpose. This will ensure Σ_s positive definite.
3. Draw n responses for each set s from the corresponding distribution $f(\mathbf{t}_s, \boldsymbol{\beta}_s, \Sigma_s)$.
4. Now, we will create outliers. For each outcome, with a probability 0.01, replace the current value. The replacement value is the current value multiplied by $\mathcal{U}(2,4) \cdot \sigma(\boldsymbol{\beta}_s)$, where $\sigma(\boldsymbol{\beta}_s)$ is the standard deviation of $\boldsymbol{\beta}_s$.
5. Initialize e_{NR} and e_{SR} to zero. These are the number of new observations that appears outside of the prediction interval of NR and SR respectively.
6. Create a loop across the s sets of data. Regress using NR and SR and create the corresponding prediction intervals π_{NR} and π_{SR}.
7. Within the loop draw k new observations from $f(\mathbf{t}_s, \boldsymbol{\beta}_s, \Sigma_s)$. Add the number of new obs. that appear outside π_{NR} to e_{NR}. Add the number of new obs. that appear outside π_{SR} to e_{SR}.
8. We now have the overall proportions $e_{SR}/(k \cdot s)$ and $e_{NR}/(k \cdot s)$ of all new observations that missed the prediction intervals. If the methods and construction of intervals are good, these proportions should be similar to the level of the prediction intervals. In order to compare NR and SR simply look at these proportions.

note: I would change the method of creating the prediction intervals depending on the new regression strategy. We may create a general method, but it could be that specific methods are better. Say NR identifies outliers and ignore them. Then we could create a non-parametric $n - 1/n + 1$ level prediction interval $\pi_{NR}(n - 1/n + 1) = [\min(\mathbf{Y}_s), \max(\mathbf{Y}_s)]$ for NR, where $\mathbf{Y}_s$ denote the outcomes of the set s. The corresponding interval for SR would be $\pi_{SR}(n-1/n+1) =$ `predict.lm(..., level=n-1/n+1)`. Through this approach we obtain comparable e_{NR} and e_{SR} in a simple fashion. It do however require that we balance the parameters of the simulation well. For a large n, the prediction intervals will be tight and we can risk having very few new observations appearing within.

(b) For the implemented example I chose $s = 1000$, $n = 500$ and $m = 100$. Half the sets were drawn from the multivariate normal distribution, while the other half were drawn from the multivariate Cauchy distribution. The simulation data is created in the following code.

```
library(MASS)
library(stats)
library(mvtnorm)

# The parameters are named as in the answer to part 1a
# note: data[,,1] corresponds to outcomes

# Chosen values
  s = 1000
  n = 500
  m = 100
  data = array(0,dim=c(s,n,m))
  t = c(rep("normal",s/2),rep("cauchy",s/2))
  range = c(-100, 100)

# Lopp over all sets
  for(S in 1:s){
    # Creating distribution parameters randomly
    beta = runif(m,range[1],range[2])
    sigma = apply(matrix(1:m), 1, function(x){
      c(runif(x,range[1],range[2]), rep(0,m-x))})
    sigma = sigma*t(sigma)

    # Random draws from distribution
    if (t[S]=="normal"){
      data[S,,] = mvrnorm(n, beta, sigma)
    }else{ # cauchy
      data[S,,] = rmvt(n, sigma, delta=beta, df=1)
    }

    # Creating outliers
    outliers = sample(c(0,1), 100, replace=T, prob=c(.99,.01))
    data[S,outliers,1] = runif(sum(outliers),2,4)*sd(beta)
  }
  #data # The generated dataset
```

Problem 2

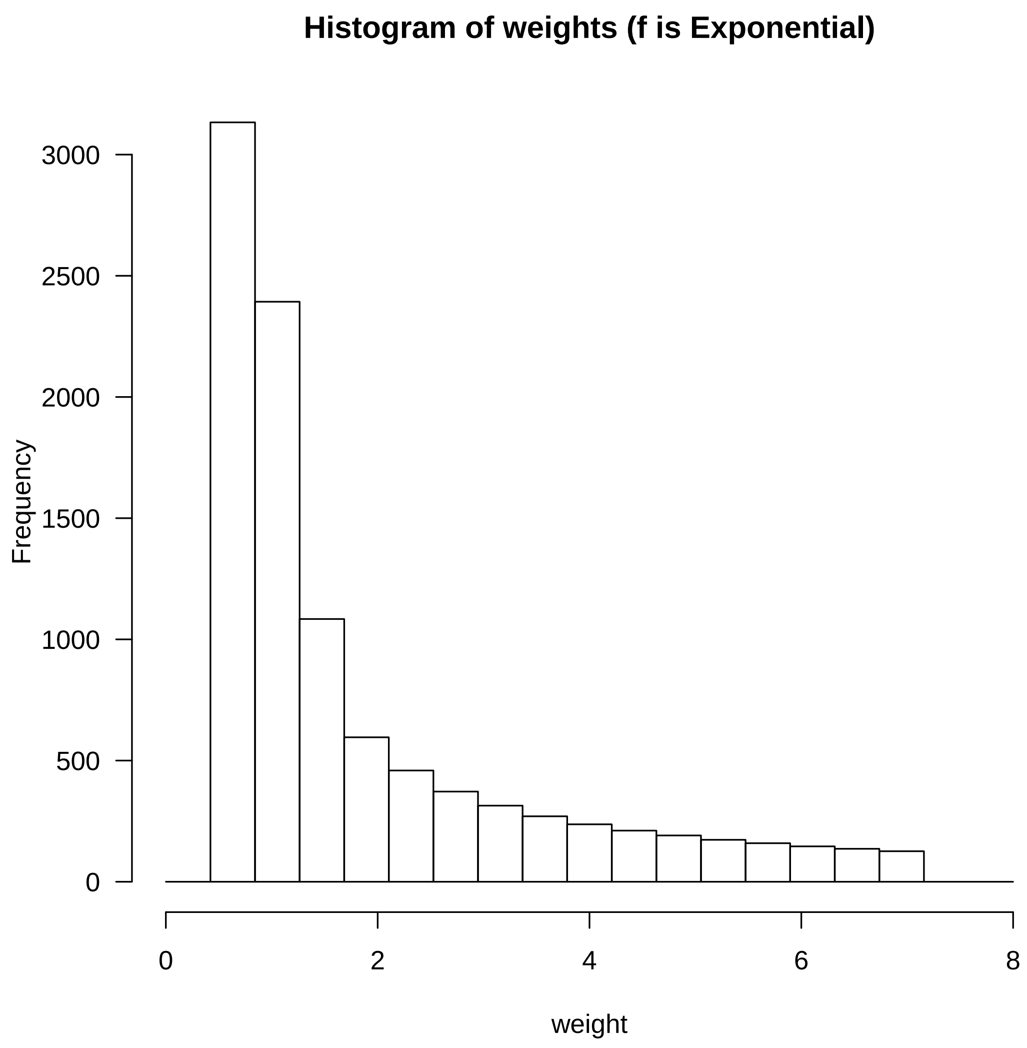
(a) The Pareto distribution is a subexponential heavy-tailed distribution, meaning that it decays slower than the exponential distribution. The Pareto cdf has a denominator with a polynomial in x, while the exponential pdf has a denominator with an exponent in x. The fact that an exponent increases faster than a polynomial justifies the statement that the Pareto distribution has the fattest tail.

Table 1: Summary of the 1st and 2nd raw moment of X.

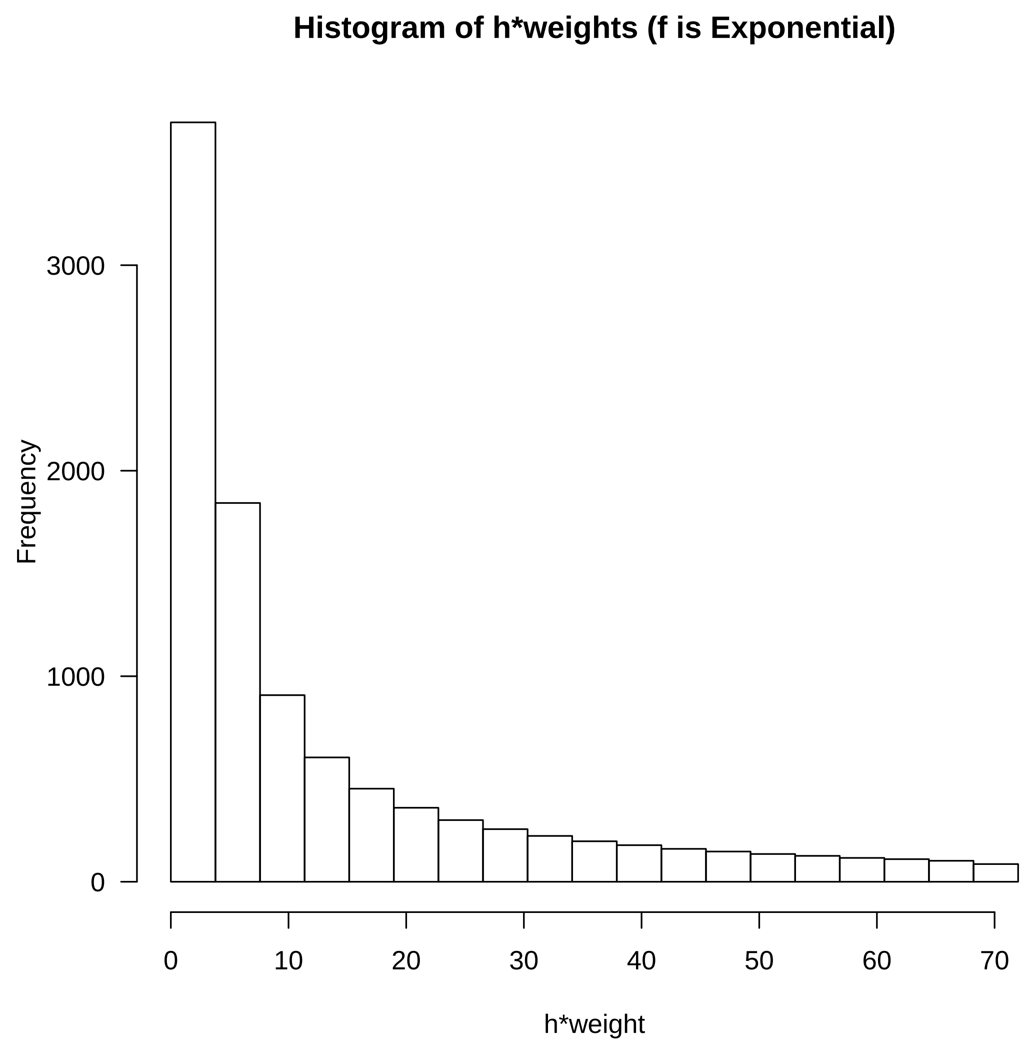
	EX	$E(X^2)$
(b) sample pdf is Pareto	14.2	5.7e+35
(c) sample pdf is Exponetial	4.2	0.7

(b) I looked at the interval $x \in [2, 10]$. The first moment and second moment for X is shown included in table 1. Histograms of $h(x)f(x)/g(x)$ and $f(x)/g(x)$ are shown in figure 1. From figure 1b we can see that there are a large variance in the dataset $h(x_i)f(x_i)/g(x_i)$. This corresponds with the huge value of the 2nd raw moment of X from table 1. The code is included below.

```r
## Parameter specicfication
  lLim = 10
  rLim = 2
  m = 10000
  h1 = seq(rLim,lLim, l=m)
  h2 = h1^2

## Functions specified by the problem
  # Pareto cdf
  dpar = function(x){
    # parameters: alfa=2, beta=3
    return(24*x^(-4))
  }

  # Exponential cdf
  dexp = function(x){
    # parameter=1, right shift of 2
    return(exp(2-x))
  }

  # Weights
  weights = function(h, f = "exp"){
    if (f == "exp"){
      return(dpar(h)/dexp(h))
    }else{
      return(dexp(h)/dpar(h))
    }

  }

  # Expectation
  expect = function(h, f = "exp"){
    return(sum(h*weights(h,f))/(length(h)-1))
  }

## Computations part a
  # Expectation estimates
  aEX = expect(h1)
  aEX2 = expect(h2)

  # Histograms
  hist(weights(h1), breaks=seq(0,ceiling(max(weights(h1))),l=20),
       main = "Histogram of weights (f is Exponential)", las=1, xlab="weight")
```

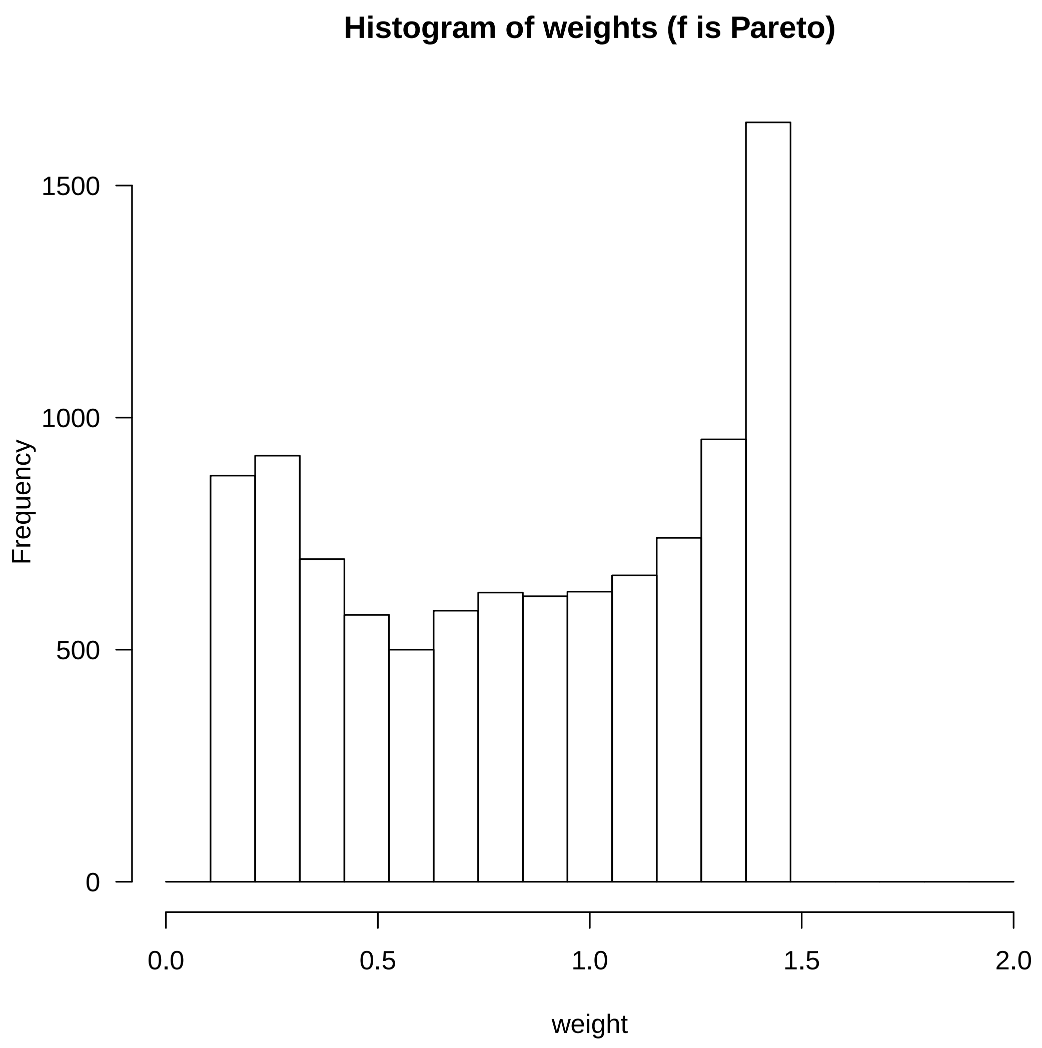


```
hist(h1*weights(h1), breaks=seq(0,ceiling(max(h1*weights(h1))),l=20),
     main = "Histogram of h*weights (f is Exponential)", las=1, xlab="h*weight")
```

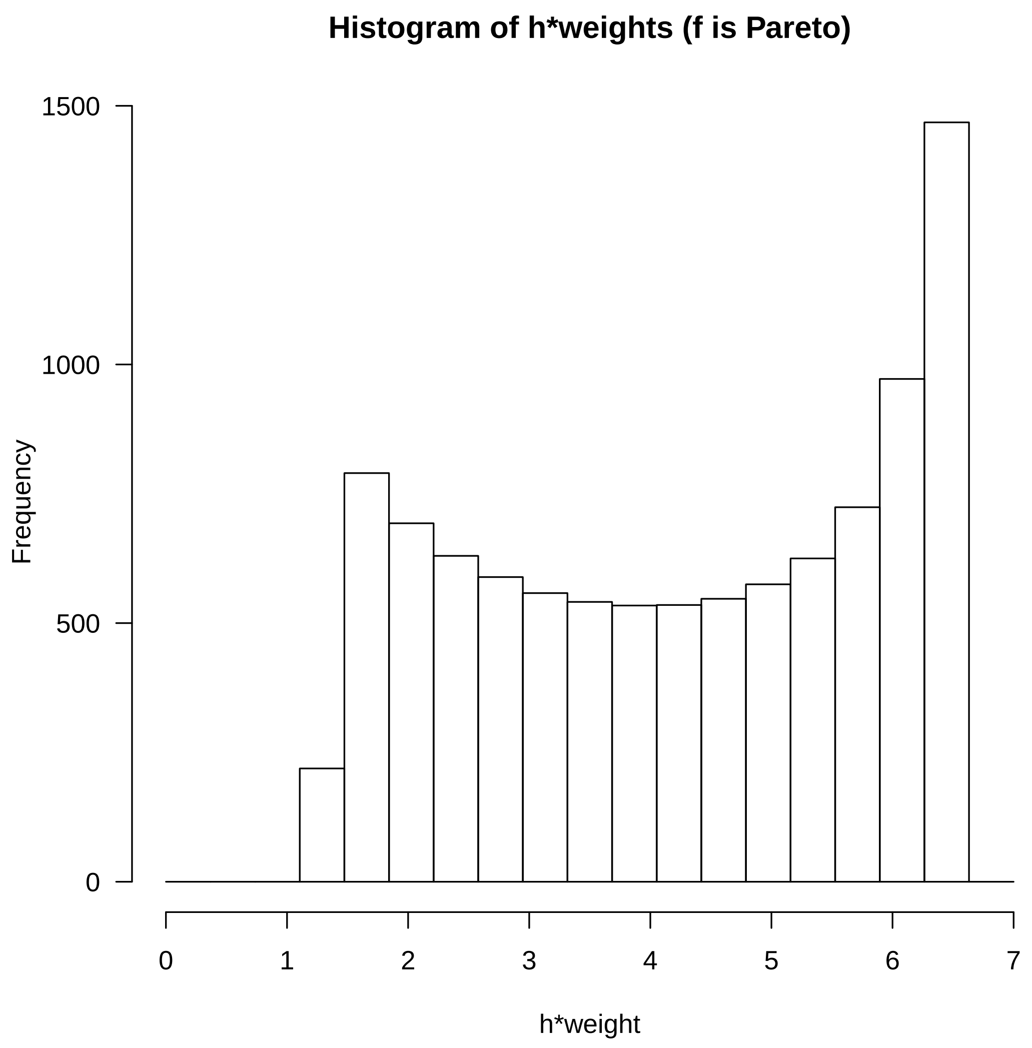


(c) The results are summarized in table 1 and figure 1. From figure 1d we can see that the variance of $h(x_i)f(x_i)/g(x_i)$ is much smaller than for part (b). The raw moments of X is displayed in table 1. The variance of X did not blow up when f is the Pareto and g the exponential distribution. The code is below.

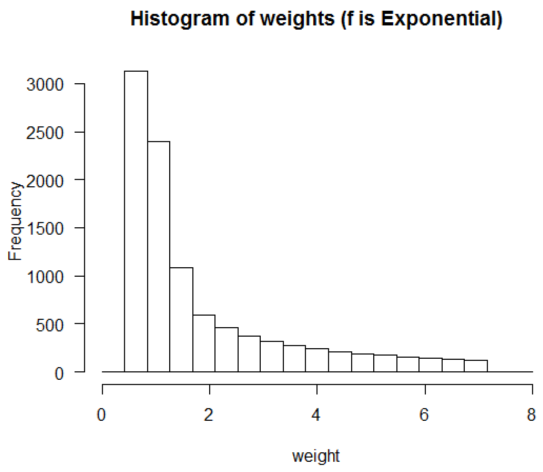
```
## Computations part b
  # Expectation estimates
  bEX = expect(h1, f= "par")
  bEX2 = expect(h2, f= "par")

  # Histograms
  hist(weights(h1, f="par"),breaks=seq(0,ceiling(max(weights(h1, f="par"))),l=20),
       main = "Histogram of weights (f is Pareto)", las=1, xlab="weight")
```



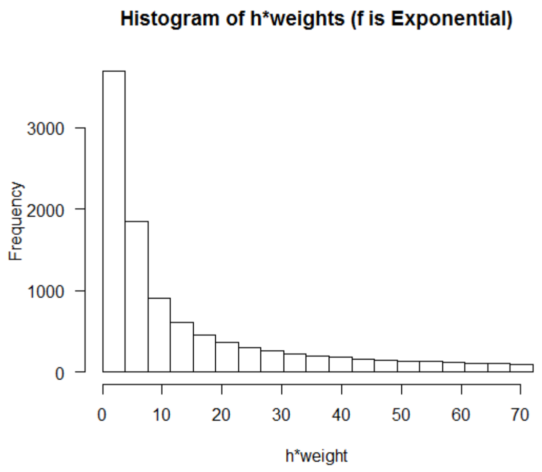
```r
hist(h1*weights(h1, f="par"),breaks=seq(0,ceiling(max(h1*weights(h1, f="par"))),
     l=20),main = "Histogram of h*weights (f is Pareto)",las=1,xlab="h*weight")
```



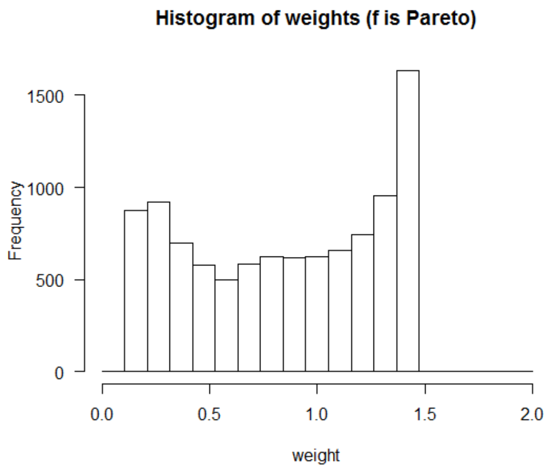
Histogram of h*weights (f is Pareto)
Frequency
0
500
1000
1500
0
1
2
3
4
5
6
7
h*weight



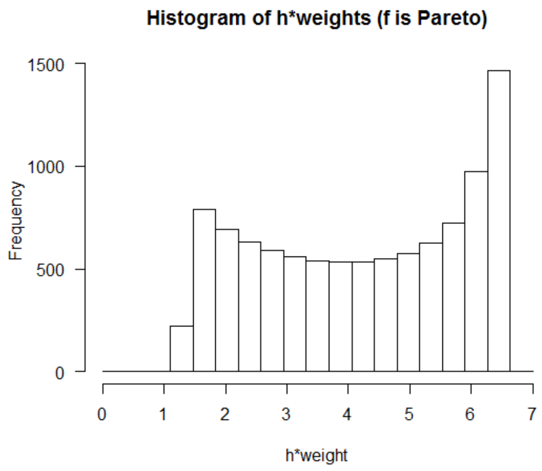
(a) A view on the size of the weights in the IS. f if exp, g is pareto and $h(x)$ is equally spaced on $[2, 10]$.



(b) As on the left, but counting h·weights.



(c) A view on the size of the weights in the IS. f if pareto, g is exp. and $h(x)$ is equally spaced on $[2, 10]$.



(d) As on the left, but counting h·weights.

Figure 1: Histograms summarizing the quantities $h(x)f(x)/g(x)$ and $f(x)/g(x)$ of IS, using a pareto distribution and an exponential distribution.

Problem 3

(a) The densities of the latent variable z_i given the observed values y_i can be written

$$f(z_i|y_i = 1) = \phi(z_i; X_i^T\beta, 1) \cdot \mathbf{1}(z_i > 0),$$
$$f(z_i|y_i = 0) = \phi(z_i; X_i^T\beta, 1) \cdot \mathbf{1}(z_i < 0).$$

The corresponding conditional expected values and variances can, according to the suggested Wikipedia page on truncated normal distributions, be written as

$$\mathbb{E}[z_i|y_i=1] = X_i^T\beta + \frac{\phi(z_i;X_i^T\beta)}{\Phi(z_i;X_i^T\beta)}\,,$$
$$\mathbb{V}[z_i|y_i=1] = 1 + X_i^T\beta\cdot\frac{\phi(X_i^T\beta)}{\Phi(X_i^T\beta)} - \left(\frac{\phi(X_i^T\beta)}{\Phi(X_i^T\beta)}\right)^2,$$
$$\mathbb{E}[z_i|y_i=0] = X_i^T\beta - \frac{\phi(z_i;X_i^T\beta)}{\Phi(z_i;X_i^T\beta)}\,,$$
$$\mathbb{V}[z_i|y_i=1] = 1 + X_i^T\beta\cdot\frac{\phi(X_i^T\beta)}{\Phi(-X_i^T\beta)} - \left(\frac{\phi(X_i^T\beta)}{\Phi(-X_i^T\beta)}\right)^2.$$

The EM algorithm is involves first expressing the likelihood of the observed outcomes for the given design matrix. Second, we take the expected value of the logarithm of the likelihood. Finally, we find coefficients maximizing this expectation. For our case that is

$$\begin{aligned}
\mathcal{L}(\beta|y) &\propto \prod_{i=1}^n \exp(-\tfrac{1}{2}(z_i - X_i^T\beta)^2)\\
\log\mathcal{L}(\beta|y) &\propto -\frac{1}{2}\sum_{i=1}^n (z_i - X_i^T\beta)^2\\
Q(\beta|\beta^t) &= \mathbb{E}\left[-\frac{1}{2}\sum_{i=1}^n (z_i - X_i^T\beta)^2\right]\\
&= -\frac{1}{2}\sum_{i=1}^n \left(\mathbb{E}[{z_i}^2|\beta=\beta^t] - 2X_i^T\beta\mathbb{E}[z_i|\beta=\beta^t] + (X_i^T\beta^t)^2\right)\\
&= -\frac{1}{2}\sum_{i=1}^n \left(\mathbb{V}[z_i|\beta=\beta^t] + \mathbb{E}[z_i|\beta=\beta^t]^2 - 2X_i^T\beta\mathbb{E}[z_i|\beta=\beta^t] + (X_i^T\beta^t)^2\right)\\
&= -\frac{1}{2}\sum_{i=1}^n \left(\mathbb{V}[z_i|\beta=\beta^t] + (\mathbb{E}[z_i|\beta=\beta^t] + X_i^T\beta^t)^2\right).
\end{aligned}$$

This is *E-step* calculation. Writing $\mathbb{E}[z_i|\beta=\beta^t] = \tilde{y_i}$ we can see that the maximization step is just a regression and we obtain the coefficients

$$\begin{aligned}
\beta_{1,t+1} &= \frac{\sum_{i=1}^n (x_{1i} - \bar{x_1})(\tilde{y_i} - \bar{\tilde{y}})}{\sum_{i=1}^n (x_{1i} - \bar{x_1})^2},\\
\beta_{2,t+1} &= \frac{\sum_{i=1}^n (x_{2i} - \bar{x_2})(\tilde{y_i} - \bar{\tilde{y}})}{\sum_{i=1}^n (x_{2i} - \bar{x_2})^2},\\
\beta_{3,t+1} &= \frac{\sum_{i=1}^n (x_{3i} - \bar{x_3})(\tilde{y_i} - \bar{\tilde{y}})}{\sum_{i=1}^n (x_{3i} - \bar{x_3})^2},\\
\beta_{3,t+1} &= \bar{\tilde{y}} - \beta_{1,t+1}\bar{x_1} - \beta_{2,t+1}\bar{x_2} - \beta_{3,t+1}\bar{x_3}.
\end{aligned}$$

(b) I calculated reasonable starting values using the `initial()` function below. The resulting values are $\beta = (0.6, 0.2, 0, 0)$.

```
design = function(n,p,lower,upper){
  X = matrix(rep(1,n*p), ncol=p)
  for(col in 2:p){
    X[,col]= runif(n,lower,upper)
  }
  return(X)
}
```

```r
test = function(){
  set.seed(200)
  beta0 = 0;beta1 = 0;beta2 = 0;beta3 = 0
  beta0t = seq(0,1,by=0.1)
  beta1t = beta0t
  n = 100
  best = 300
  y = rep(0,n)
  X = design(n, 4, -5, 5)
  for (i in 1:length(beta0t)){
    for (j in 1:length(beta1t)){
      beta = c(beta0t[i],beta1t[j], beta2, beta3)
      y = rbinom(n, 1, prob=pnorm(X%*%beta))
      data = data.frame(y,X)
      reg = summary(lm(y~X2+X3+X4,data))$coeff
      if(abs((reg[2,1]/reg[2,2])-2)< best){
        best = abs((reg[2,1]/reg[2,2])-2)
        beta0 = beta0t[i]
        beta1 = beta1t[j]
      }
    }
  }
  return(c(beta0,beta1,beta2,beta3))
}
test() # Proposed starting coefficients

## [1] 1.0 0.1 0.0 0.0
```

(c) The criteria for when to stop the optimization is that the sum of the relative change in the current coefficients is smaller than the square root of the machine epsilon, which for my R version is approximately $2.22 \cdot 10^{-16}$. A way of implementing the EM algorithm in R is shown in the following code. A summary of the method with iterations and the coefficients are shown in table 2.

Table 2: Comparison of the EM algorithm and the method of directly maximizing the log-likelihood.

	Iterations	β_0	β_1	β_3	β_4
EM	66	1.2470	0.4356	-0.1201	-0.0614
`optim()`	31	1.2470	0.4356	-0.1201	-0.0614

```r
# Function doing EM
em = function(X,y,iCoeff,tol=sqrt(.Machine$double.eps), maxI = 2e3){
  set.seed(300)
  change=Inf
  beta = iCoeff
  i = 1
  while(change>tol & i<maxI){
    i=i+1
    prev = beta
    mu = X%*%beta
    z = ifelse(y==1, mu+dnorm(mu,0,1)/pnorm(mu,0,1),mu-dnorm(mu,0,1)/pnorm(-mu,0,1))
    beta=unname(lm(z~X1+X2+X3,data.frame(z,X[,2:4]))$coeff)
    change=sum(abs(prev-beta))/sum(abs(prev))
  }
  out = c(i,beta); names(out)=c("Iterations", "beta0", "beta1", "beta2", "beta3")
  return(out)
}

# Test of the em algorithm
n = 100
p = 4
X = cbind(1, matrix(rep(runif(n*(p-1),-5,5)),ncol=p-1))
y = rbinom(n, size = 1, prob = pnorm(X))
em(X,y,initial())

## Error in em(X, y, initial()):  could not find function "initial"
```

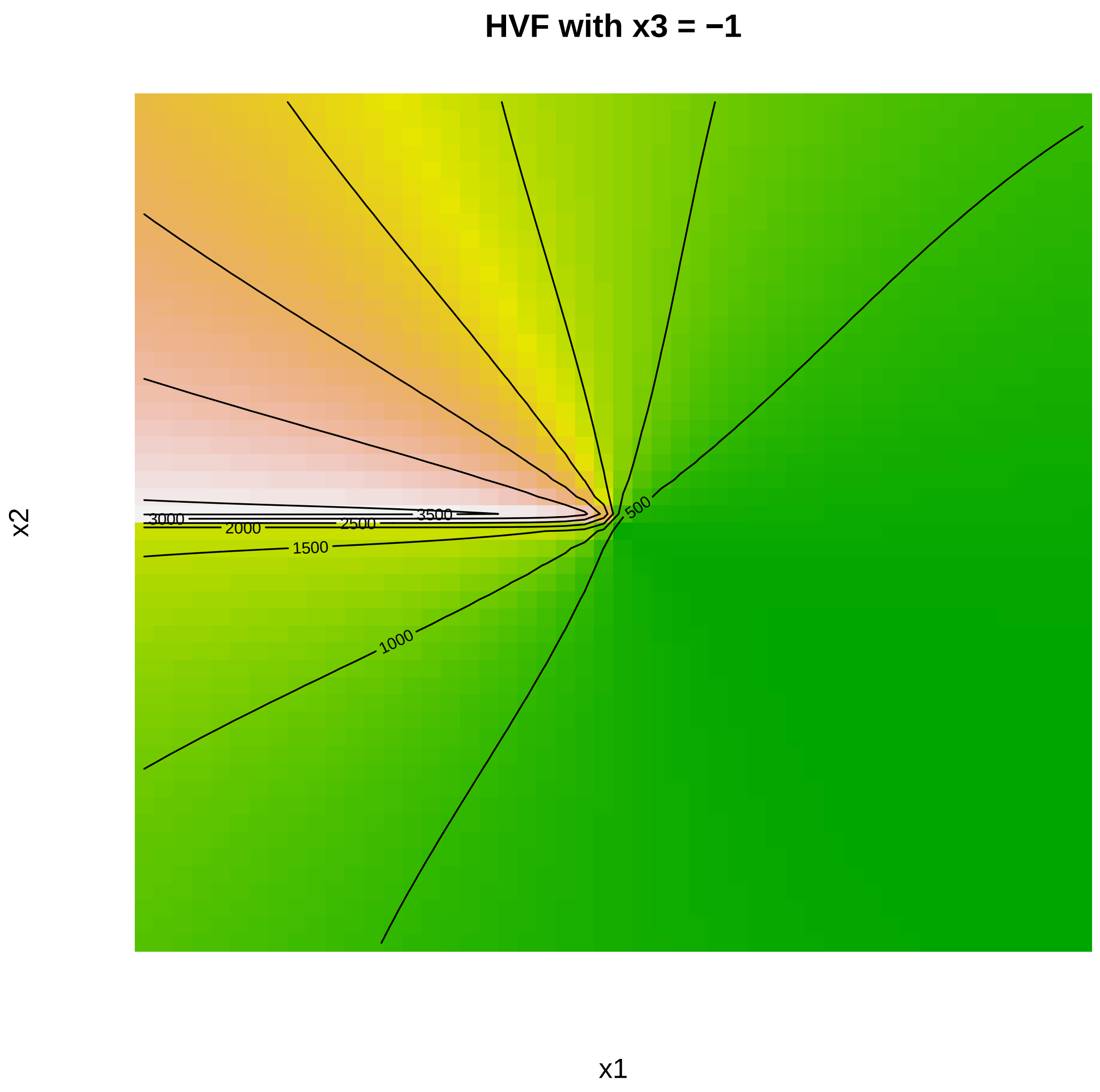
(d) A method of directly maximizing the log-likely directly is shown in the code below. The summary of the algorithm is shown in table 2. This algorithm used about half the iterations of the EM algorithm. The coefficients from the two methods are exactly the same on four decimal places.

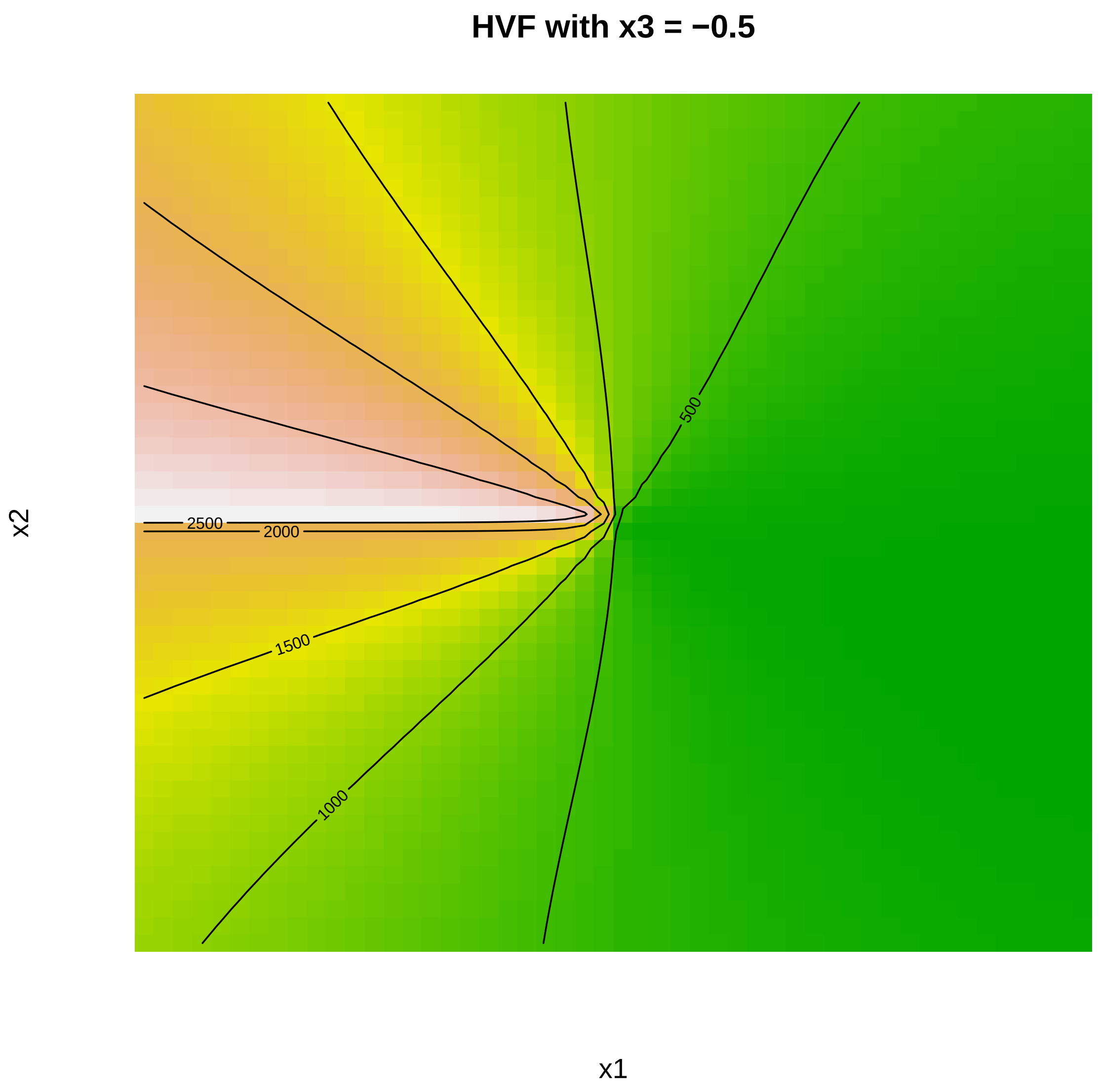
```
## Direct maximization of log-likelyhood
dirLikelihood =function(par, X, y) {
  npdf = pnorm(X%*%par, 0, 1, lower.tail = T, log.p = F)
  return(sum(y*log(npdf)+(1-y)*log(1-npdf)))
}

first = as.double(lm(y~X2+X3+X4,data.frame(y,X))$coeff)
out = optim(first, fn=dirLikelihood, gr=NULL, y=y, X=X, method="BFGS",
            control=list(trace=T,maxit=2e3,fnscale=-1), hessian=T)

## initial  value 51.049722
## final  value 50.691967
## converged

out$par     # estimates

## [1]  0.828003740  0.008302039  0.053736606 -0.018770855

out$counts  # iterations

## function gradient
##       40        7

sqrt((-1)*diag(solve(out$hessian))) # se of estimates

## [1] 0.14558892 0.04985648 0.04846936 0.05135393
```

Problem 4

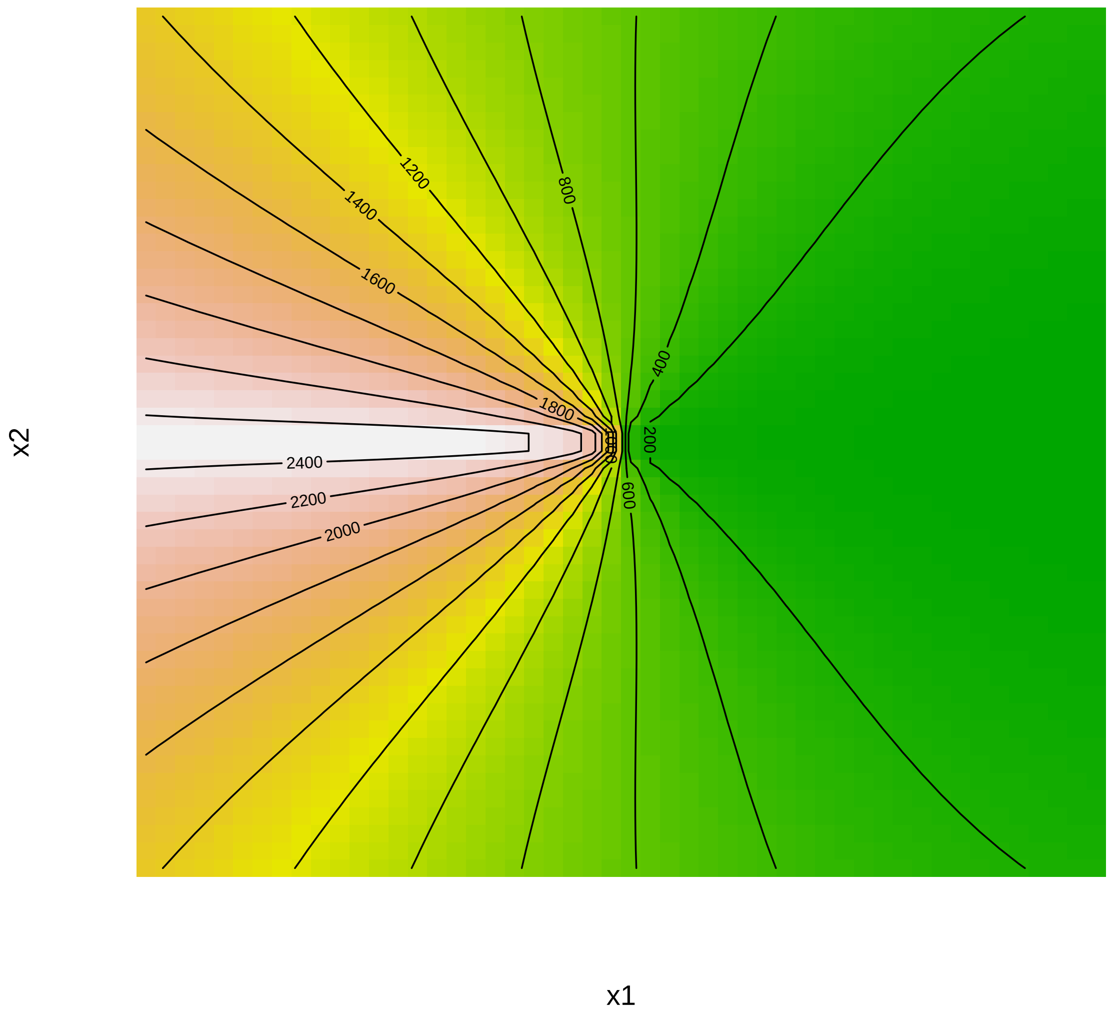
I implemented the helical valley function as a function `hvf()` in R. I plotted slices for the third coordinate equal to the five values $(-1, -0.5, 0, 0.5, 1)$. I also did a check for whether extrema calculated by `optim()` and `nlm()` changed by starting points. The returned values changed by the starting points. Code is included below.

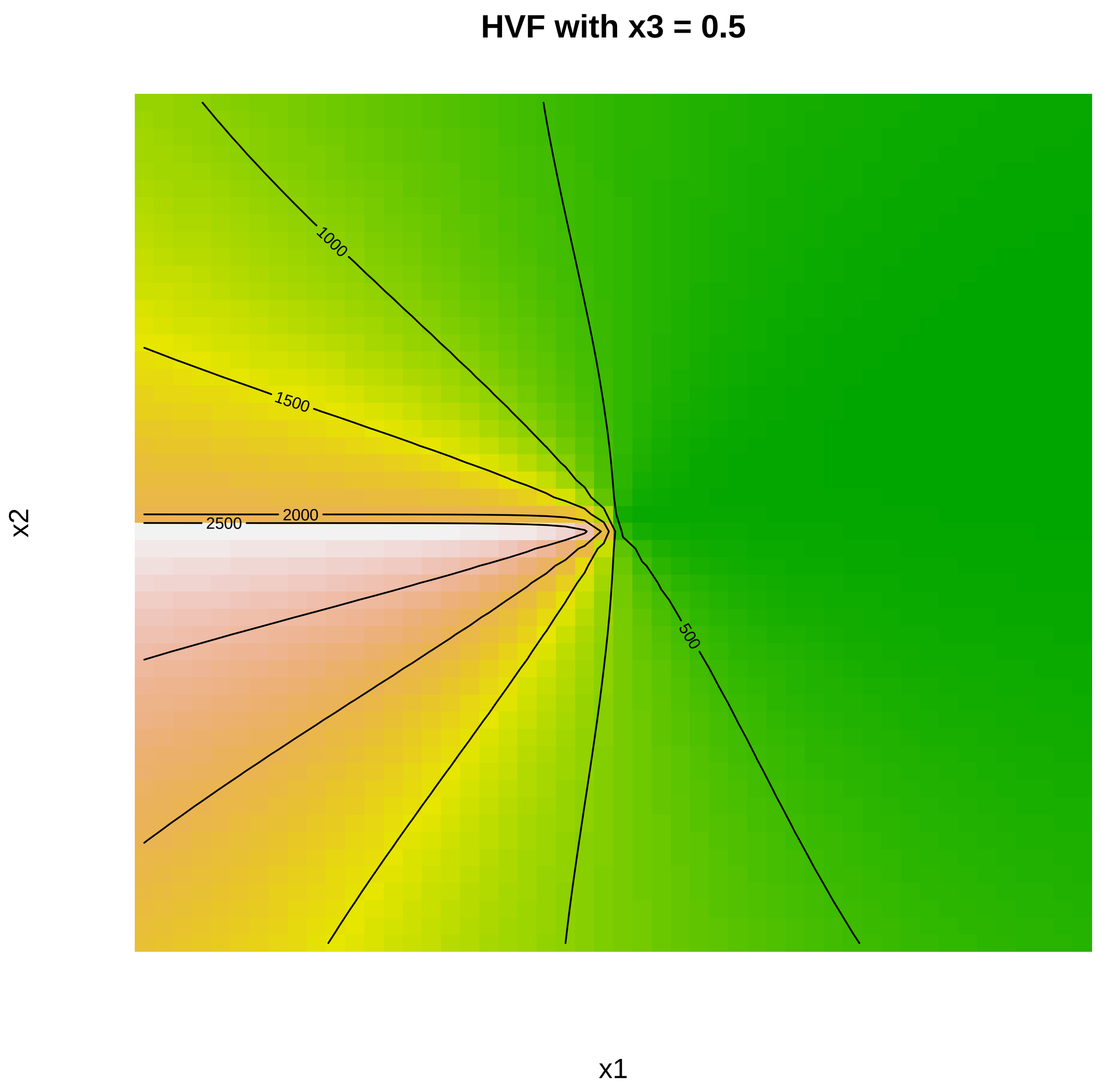
```
library(fields)
## Helical valley function
hvf <- function(x) {
  a <- 10*(x[3] - (5/pi)*atan2(x[2],x[1]))
  b <- 10*(sqrt(x[1]^2+x[2]^2)-1)
  return(a^2+b^2+x[3]^2)
}

## Surface plots
testx3 = seq(-1,1,0.5)
for(i in testx3){
  x1 <- seq(-1, 1, length.out=50)
  x2 <- seq(-1, 1, length.out=50)
  z <- apply(as.matrix(expand.grid(x1, x2)), 1, function(x) hvf(c(x, i)))
  image(x1, x2, matrix(z, 50, 50), col = terrain.colors(100),
        axes = FALSE, main=paste("HVF with x3 =",i))
  contour(x1, x2, matrix(z, 50, 50),add = TRUE)
}
```



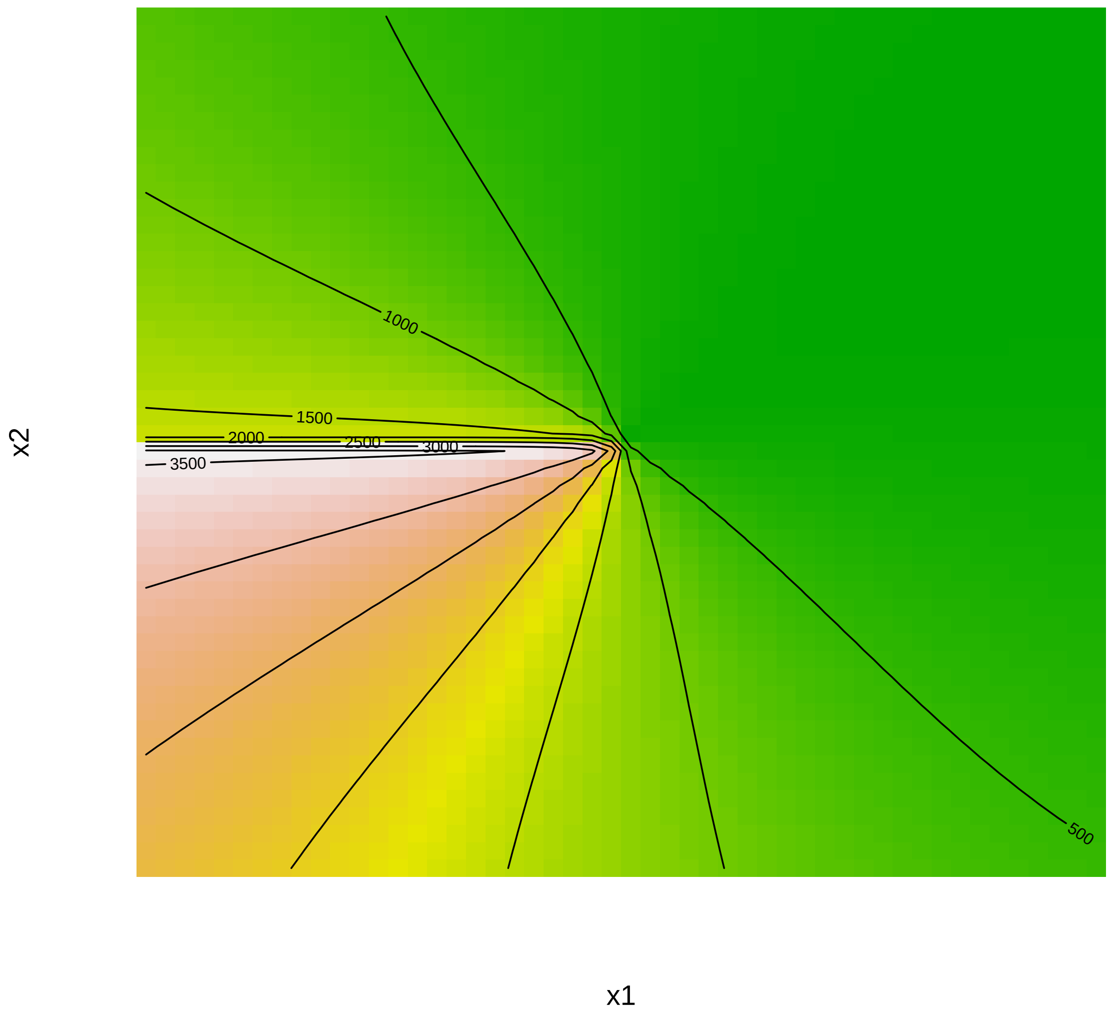

HVF with x3 = −0.5
x2
x1
2500
2000
1500
1000
500

HVF with x3 = 0




HVF with x3 = 0.5
1000
1500
2000
2500
500
x2
x1

HVF with x3 = 1



```
## Otimized values by starting point
optim(c(-1,1,2), hvf)$value

## [1] 4.062675e-06

optim(c(1,2,3), hvf)$value

## [1] 1.863176e-06

nlm(hvf, c(-1,1,2))$minimum

## [1] 2.446594e-18

nlm(hvf, c(1,2,3))$minimum

## [1] 1.701348e-08
```