Numerical Optimisation

Emmanuel S. Tsyawo

10/16/2021

Contents

Several estimators in Econometrics, e.g., Ordinary Least Squares, Maximum Likelihood, and GMM can be cast as optimisers of given criteria. These estimators may not always have closed-form expressions and do usually require numerical optimisation techniques. This explains why some emphasis ought to be put on numerical optimisation.

Luckily, numerical algorithms for optimisation are already built into R. What matters is to write a correct function for the criterion to be optimised.

One-dimensional Optimisation

The base R function for finding minima (the default) or maxima of univariate functions is `optimize()` or `optimise()`.

Let us quickly look up the documentation.

```
?optimize
```

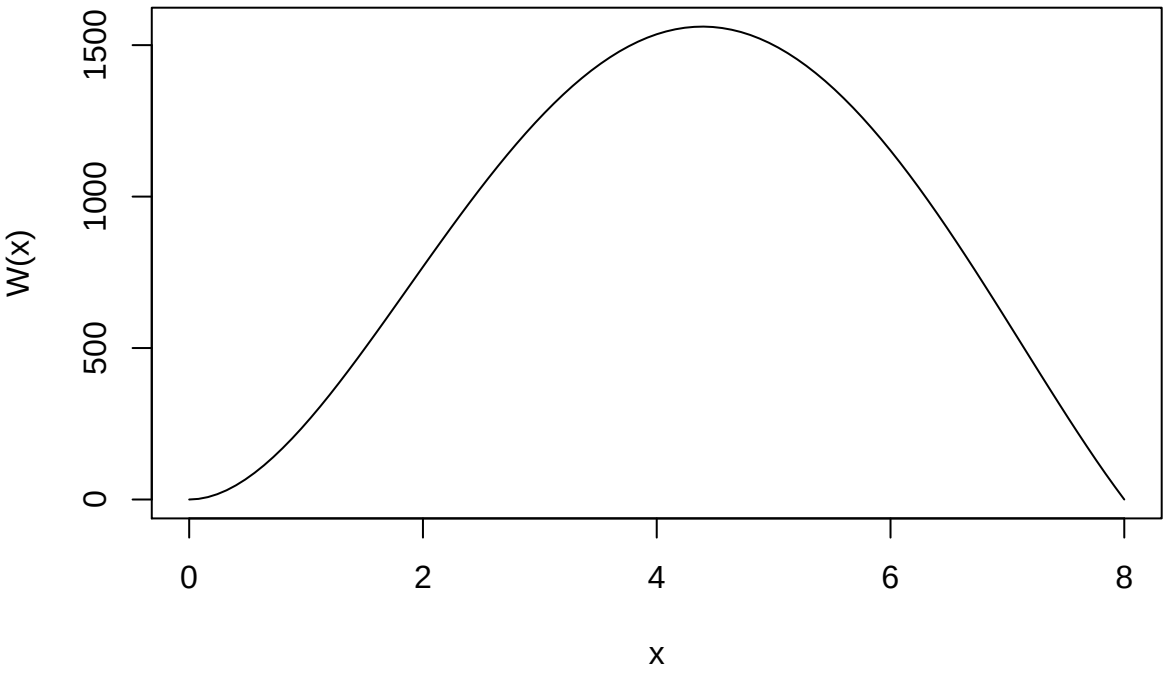
Example 1 Consider the following anonymous function $x^2(20-2x)(16-2x)$ which we wish to maximise.

```
optimize(function(x) x^2*(20-2*x)*(16-2*x), c(0,8), maximum=T)
```

```
## $maximum
## [1] 4.391515
##
## $objective
## [1] 1561.202
```

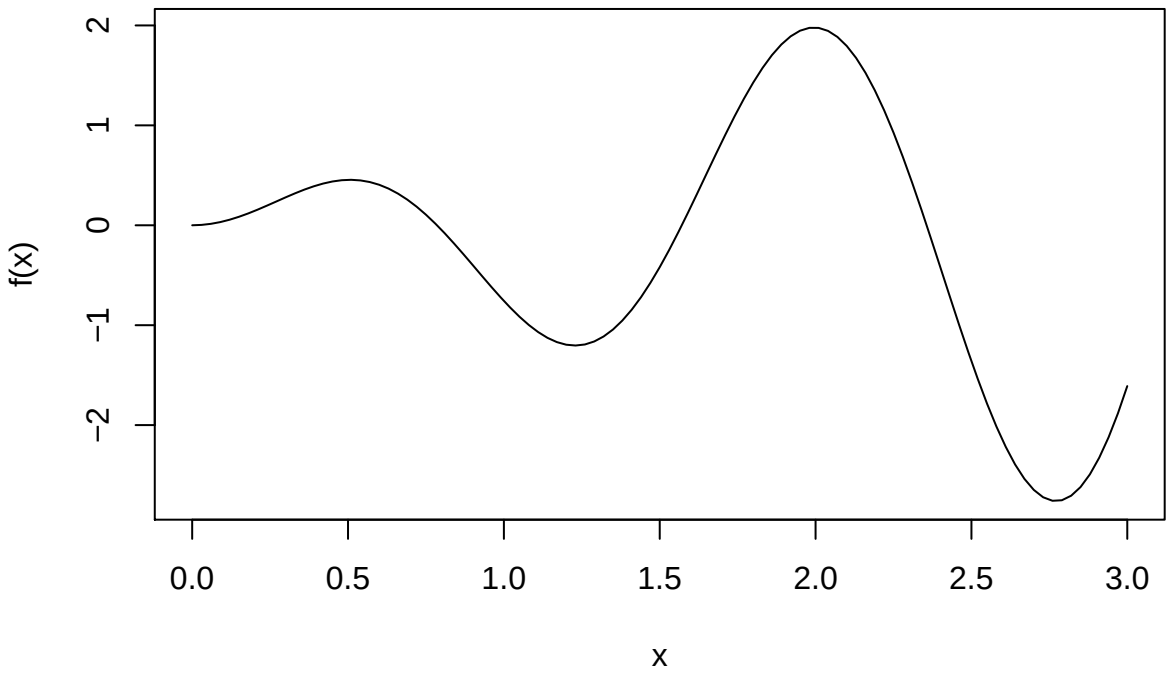
Can you plot the curve?

```
W = function(x) x^2*(20-2*x)*(16-2*x)
curve(W,0,8)
```



Example 2 Consider next the use of `optimize()` to minimise the function $f(x) = x\sin(4x)$ over the interval $[0, 3]$. Let us begin by coding and plotting the function.

```r
f = function(x) x*sin(4*x)
curve(f,0,3)
```



Applying optimize() in the simplest way yields

```r
optimize(f,c(0,3))
```

```
## $minimum
## [1] 1.228297
##
## $objective
## [1] -1.203617
```

Because we have a plot of the function, we can see that we must exclude the local minimum from the lower and upper endpoints of the search interval.

```r
(op<-optimize(f,c(1.5,3)))
```

```
## $minimum
## [1] 2.771403
##
## $objective
## [1] -2.760177
```

To find the global maximum we enter

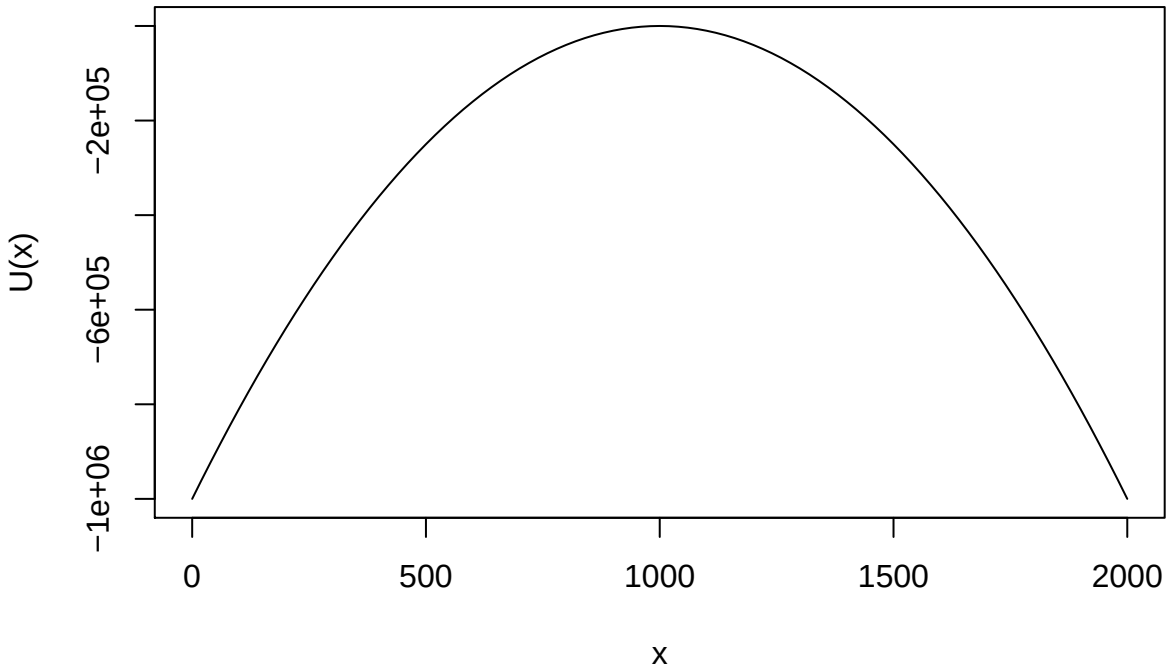
```r
optimize(f,c(1,3),maximum=TRUE)
```

```
## $maximum
## [1] 1.994684
##
## $objective
## [1] 1.979182
```

Example 3 Suppose a utility function in wealth $u(w) = -(w - 1000)^2$. Plot a curve of the utility function over the interval $[0, 2000]$, numerically compute the level of wealth that maximises utility, and give the

maximum utility attainable.

```
# Code the utility function
U = function(w) -((w-1000)^2)
# Plot the utility function
curve(U,0,2000)
```




```
# What is the level of wealth that maximises utility?
optimize(U,c(0,2000),maximum = T)
```

```
## $maximum
## [1] 1000
##
## $objective
## [1] 0
```

Multi-dimensional optimisation with optim()

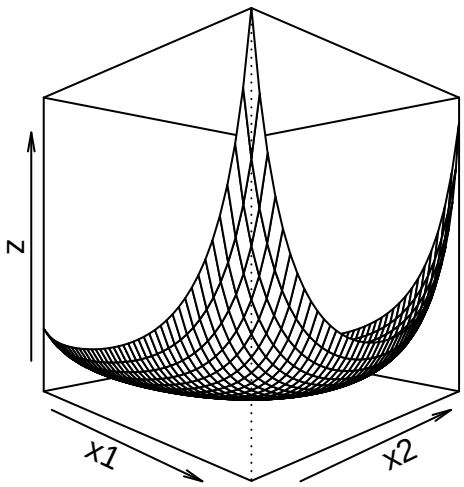
Optimisation in more than one dimension is harder to visualise and to compute. In R, the built-in function for mimisation is `optim()`. Let us look up its documentation.

```
?optim
```

The surface defined by a two-dimensional function may be visualised by the `persp()` function in R.

Example 1 Consider the bivariate function $f(x_1, x_2) = \frac{1}{x_1} + \frac{1}{x_2} + \frac{1-x_2}{x_2(1-x_1)} + \frac{1}{(1-x_1)(1-x_2)}$.

```
x1 = x2 = seq(.1,.9,.02) #create a grid of values over which to evaluate the function.
z = outer(x1,x2,FUN=function(x1,x2) 1/x1 + 1/x2 +
            (1-x2)/(x2*(1-x1)) + 1/((1-x1)*(1-x2)))
persp(x1,x2,z,theta=45,phi=0)
```



Code the function for minimisation,

```
fn = function(x) {
  x1 = x[1]
  x2 = x[2]
  return(1/x1 + 1/x2 + (1-x2)/(x2*(1-x1)) +
           1/((1-x1)*(1-x2)))
}
```

To minimize f with respect to x_1 and x_2, we run

```
(opt.f=optim(c(.5,.5),fn,hessian = T))
```

```
## $par
## [1] 0.3636913 0.5612666
##
## $value
## [1] 9.341785
##
## $counts
## function gradient
##       55       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##           [,1]      [,2]
## [1,] 65.338567  4.990949
## [2,]  4.990949 66.308139
```

It is good practice to examine the Hessian matrix at the minimiser to be sure a minimum is attained.

```
eigen(opt.f$hessian)$values
```

```
## [1] 70.83779 60.80891
```

Both eigen-values are positive; the Hessian is thus positive definite.

Example 2 Let us maximise our likelihood function of the normal linear regression model from Session 1.1.

Let us load the data set. Ensure the working directory is set to the location of the data set.

```
dat<- read.csv("dat.csv",header = T,sep = " ") # load data set
```

The log-likelihood is repeated here for convenience.

```
llike_lnorm<- function(pars,Y,X){ #pars is the vector [beta,sigma]
  N = length(Y) #obtain number of observations
  X = as.matrix(cbind(1,X)) # include 1's for the intercept term
  k=ncol(X) # number of parameters to estimate
  np = length(pars) #obtain number of parameters (including sigma)
  if(np!=(k+1)){stop("Not enough parameters.")} #ensure the number of parameters is correct
  beta = matrix(pars[-np],ncol = 1) #obtain column vector of slope parameters beta
  sig2 = pars[np] #sigma^2
  U<- Y - X%*%beta #compute U
  ll = -(N/2)*log(2*pi*sig2) -sum(U^2)/(2*sig2)#obtain log joint likelihood
  return(ll)
}
```

Let us maximise the log-likelihood function.

```
optp<-optim(par=c(rep(0,4),2),fn=llike_lnorm,Y=dat$nonwife,
              X=dat[c("age","education","experience")],control=list(fnscale=-1),
              hessian = T)
optp
```

```
## $par
## [1] -32.2393118   0.4169752   3.1426379  -0.3975256  96.1863300
##
## $value
## [1] -2913.92
##
## $counts
## function gradient
##      502       NA
##
## $convergence
## [1] 1
##
## $message
## NULL
##
## $hessian
##               [,1]          [,2]          [,3]          [,4]        [,5]
## [1,]   -7.82855534 -3.330099e+02   -96.1883046 -8.322388e+01 -0.01985416
## [2,] -333.00989855 -1.467501e+04 -4074.3419553 -3.710267e+03 -0.35191113
## [3,]  -96.18830461 -4.074342e+03 -1222.5021993 -1.032091e+03  0.37129905
## [4,]  -83.22388442 -3.710267e+03 -1032.0905266 -1.393784e+03  0.04800404
## [5,]   -0.01985416 -3.519111e-01     0.3712991  4.800404e-02 -0.06798712
```

Because the problem is a maximisation problem, we use `control=list(fnscale=-1)` option or we return a negative function value and use minimisation.

Compare the MLE to the built-in OLS function `lm()`.

```r
reg<- lm(nonwife~age + education + experience,data=dat)
summary(reg)
```

```
##
## Call:
## lm(formula = nonwife ~ age + education + experience, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -25.506  -6.623  -1.827   3.801  66.976
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6.92553    3.22125  -2.150   0.0319 *
## age          0.26147    0.05235   4.994 7.36e-07 ***
## education    1.61326    0.17508   9.214  < 2e-16 ***
## experience  -0.36588    0.05211  -7.021 4.94e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.8 on 749 degrees of freedom
## Multiple R-squared:  0.1421, Adjusted R-squared:  0.1387
## F-statistic: 41.35 on 3 and 749 DF,  p-value: < 2.2e-16
```

```r
(sig2.OLS=mean(reg$residuals^2)) #estimate of sigma^2
```

```
## [1] 115.9796
```

The results are not close. What is wrong? Let us vary the starting values to see.

```r
start.v=c(-5,0.2,1.6,-0.3,110)
optp1<-optim(par=start.v,fn=llike_lnorm,Y=dat$nonwife,
             X=dat[c("age","education","experience")],control=list(fnscale=-1),
             hessian = T)
optp1
```

```
## $par
## [1]  -6.9239491   0.2615488   1.6127236  -0.3658985 115.9764913
##
## $value
## [1] -2858.121
##
## $counts
## function gradient
##      204       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,] -6.492695e+00 -2.761853e+02 -7.977479e+01 -6.902261e+01 -9.464429e-05
## [2,] -2.761853e+02 -1.217087e+04 -3.379099e+03 -3.077149e+03 -3.680157e-03
```

```
## [3,] -7.977479e+01 -3.379099e+03 -1.013895e+03 -8.559752e+02 -1.330307e-03
## [4,] -6.902261e+01 -3.077149e+03 -8.559752e+02 -1.155950e+03 -1.014882e-03
## [5,] -9.464429e-05 -3.680157e-03 -1.330307e-03 -1.014882e-03 -2.799277e-02
```

Compare OLS and ML parameter estimates

```
res=rbind(c(reg$coefficients,sig2.OLS),optp1$par)
row.names(res)=c("OLS","MLE"); colnames(res)[5]="Sigma2"
res
```

```
##     (Intercept)       age education experience   Sigma2
## OLS   -6.925533 0.2614656  1.613260 -0.3658783 115.9796
## MLE   -6.923949 0.2615488  1.612724 -0.3658985 115.9765
```

Compare OLS and ML parameter standard errors

```
stes=sqrt(rbind(diag(vcov(reg)),diag(-solve(optp1$hessian))[-5]))
row.names(stes)=c("OLS","MLE")
stes
```

```
##     (Intercept)        age education experience
## OLS    3.221248 0.05235361 0.1750833 0.05211057
## MLE    3.212641 0.05221370 0.1746154 0.05197129
```

Voilà! Both results are similar now. Lesson: we need to vary our starting values and choose the estimates that yield the smallest (largest) function value under minimsiation (maximisation).

Example 3 Estimating parameters of a CES Production function $Q = d(aK^r + (1-a)L^r)^{1/r}$ where $[a, d, r]$ is a vector of parameters. A first step is to derive an econometric structural model from the CES economic model by assuming a form of disturbance, e.g., in measurement of labour or capital. We assume multiplicative disturbances $\dot{Q} = d\exp(U)(aK^r + (1-a)L^r)^{1/r}$ where U is a zero-mean random variable.

Let us simulate data following the structural model. First, simulate some data on labour L and capital K.

```
n=1000
set.seed(14)
L = rpois(n=n,lambda=8) #quantity of labour sampled from the Poisson distribution
K = rbeta(n,2,4)*15 #quantity of capital sampled from the beta distribution
summary(L)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   6.000   8.000   7.898  10.000  18.000
```

```
summary(K)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1231  2.8660  4.5488  4.9402  6.7672 14.1202
```

Now, set parameters to be recovered in estimation $a = 0.4$, $d = 3$, and $ r=2 $.

```
a = 0.4; r = 2; d = 3; set.seed(0)
Q = d*exp(runif(n,-1,1))*(a*K^r + (1-a)*L^r)^(1/r) # generate output
summary(Q)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.197  12.284  20.256  25.324  34.387  98.465
```

Choice of estimator: non-linear least squares. Can you think of other appropriate estimators? Dividing

through by L and taking the log gives the following expression.

$$(\log(Q) - \log(L)) = \log(d) + \frac{1}{r}\log(1 + a((K/L)^r - 1)) + U/L \tag{1}$$

$$= \alpha + \frac{1}{r}\log(1 + a((K/L)^r - 1)) + \varepsilon \tag{2}$$

Let us code the criterion. Recall the criterion for non-linear least squares is $SS(\theta) = \sum_{i=1}^{n} \varepsilon_i(\theta)^2$.

```
fnCES = function(pars,Q,L,K){
  alf = pars[1]; r = pars[2]; a = pars[3] #assign parameter values
  eps = log(Q)-log(L)-alf - (1/r)*log(1 + a*((K/L)^r-1))
  sum(eps^2)
}
```

It is good advice to test the function before plugging it into the minimiser `optim()`.

```
fnCES(pars = c(2,2,.2),Q=Q,L=L,K=K) #test function
```

```
## [1] 1252.589
```

Minimise the sum of squares:

```
(ans = optim(par = c(1,2,.5),fn=fnCES,Q=Q,L=L,K=K,hessian = TRUE))
```

```
## $par
## [1] 1.121365 2.153030 0.487127
##
## $value
## [1] 331.6696
##
## $counts
## function gradient
##      142       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##            [,1]       [,2]      [,3]
## [1,] 2000.00000  99.090347 -705.3727
## [2,]   99.09035   9.078341  -58.2914
## [3,] -705.37267 -58.291400  662.0800
```

```
# check values
d.est=exp(ans$par[1]); r.est=ans$par[2]; a.est = ans$par[3]
res.Q=rbind(c(d.est,r.est,a.est),c(d,r,a)) #estimates on first row, true values on the second
colnames(res.Q)=c("d","r","a"); rownames(res.Q)=c("NLS","True")
res.Q
```

```
##             d       r        a
## NLS  3.069041 2.15303 0.487127
## True 3.000000 2.00000 0.400000
```

```r
eigen(ans$hessian)$values #examine the eigen-values of the Hessian
```

```
## [1] 2308.837793  359.481696    2.838846
```

Constrained Optimisation

Sometimes, optimisation problems do have (natural) bounds on the parameter support. E.g., regressions with restricted parameter spaces, constrained utility optimisation, etc.

R has two packages, `alabama` and `Rsolnp` that implement the augmented Lagrange multiplier method for general nonlinear optimisation with both equality and inequality constraints.

Example 1 Consider this example with the ALABAMA package:

$$\min_x \sin(x_1 x_2 + x_3) \text{ subject to} \tag{3}$$

$$-x_1 x_2^2 + x_1^2 x_3^2 - 5 = 0 \tag{4}$$

$$x_1 - x_2 \geq 0 \tag{5}$$

$$x_2 - x_3 \geq 0 \tag{6}$$

```r
f = function(x) sin(x[1]*x[2]+x[3]) # objective function

heq = function(x) -x[1]*x[2]^2 + x[1]^2*x[3]^2 -5 # equality constraint

hin = function(x) { # inequality constraint
  h = rep(NA,2)
  h[1] = x[1]-x[2] # set as non-negativity constraints
  h[2] = x[2] -x[3]
  h
}

p0 = c(3,2,1) # starting values for constrained optimisation
require(alabama) # Also loads numDeriv package
```

```
## Loading required package: alabama
```

```
## Loading required package: numDeriv
```

```r
(ans = constrOptim.nl(par=p0, fn = f, heq=heq, hin = hin,control.outer = list(trace=FALSE)))
```

```
## $par
## [1] 2.862271 1.268103 1.082649
##
## $value
## [1] -1
##
## $counts
## function gradient
##      232       54
##
## $convergence
## [1] 0
##
## $message
## NULL
##
```

```
## $hessian
##           [,1]       [,2]      [,3]
## [1,]  32537.99  -46292.35  113134.3
## [2,] -46292.35   65882.78 -160973.8
## [3,] 113134.26 -160973.82  393377.6
##
## $outer.iterations
## [1] 9
##
## $lambda
## [1] 0.0004030553
##
## $sigma
## [1] 1250
##
## $barrier.value
## [1] 0.01348658
##
## $K
## [1] 5.33497e-08
```

A more robust option from the `alabama` package is the `auglag()` function. It allows starting values that violate the inequality constraints.

```
p0 = c(1,2,3) # starting values for constrained optimisation
(ans = constrOptim.nl(par=p0, fn = f, heq=heq, hin = hin))
```

Now try:

```
(ans = auglag(par=p0, fn = f, heq=heq, hin = hin,control.outer = list(trace=FALSE)))
```

```
## $par
## [1] 4.2581563 0.9431374 0.6961698
##
## $value
## [1] -1
##
## $counts
## function gradient
##      240       75
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $outer.iterations
## [1] 5
##
## $lambda
## [1]  0.000000e+00  0.000000e+00 -8.853461e-06
##
## $sigma
## [1] 1250
##
```

```
## $gradient
## [1] -0.0003776645  0.0006828772 -0.0020176071
##
## $ineq
## [1] 3.3150188 0.2469677
##
## $equal
## [1] -6.649378e-08
##
## $kkt1
## [1] TRUE
##
## $kkt2
## [1] FALSE
##
## $hessian
##           [,1]       [,2]      [,3]
## [1,]  13106.42  -32505.63  102182.8
## [2,] -32506.86   80664.76 -253478.2
## [3,] 102188.65 -253482.99  796743.3
```

Example 2 Consider the following constrained utility maximisation problem

$$u(c,m) = (c^r + (1.2m)^r)^{1/r} \text{subject to } c \geq 20,\ m \geq 20,\ \text{and } c+m \leq 100 \tag{7}$$

where $r = -4$.

```
uF<- function(a) - (a[1]^(-4) + (1.2*a[2])^(-4))^(-0.25) # return negative fn value for minimisation
p0=c(20,20) # starting values
uF(p0) # test function value at starting values
```

```
## [1] -18.12589
```

```
hin<- function(a){ #code inequality constraints and non-negativity conditions
  h = rep(NA,3)
  h[1]=a[1]-20
  h[2]=a[2]-20
  h[3]=100-a[1] - a[2]
  h
}
(opF<- auglag(par = p0,fn=uF,hin = hin,control.outer = list(trace=FALSE)))
```

```
## $par
## [1] 53.63969 46.36031
##
## $value
## [1] -45.90508
##
## $counts
## function gradient
##      151       32
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
##
## $outer.iterations
## [1] 4
##
## $lambda
## [1] 0.000000 0.000000 0.458622
##
## $sigma
## [1] 10000
##
## $gradient
## [1] -1.442850e-05  1.297714e-05
##
## $ineq
## [1]  3.363969e+01  2.636031e+01 -4.259395e-08
##
## $equal
## [1] NA
##
## $kkt1
## [1] TRUE
##
## $kkt2
## [1] TRUE
##
## $hessian
##          [,1]     [,2]
## [1,] 5415.980 5481.251
## [2,] 5415.938 5481.300
```

```
opF$par
```

```
## [1] 53.63969 46.36031
```