Optimization problems constrained by parameter sums

Gabor Grothendieck, GKX Group,
John C. Nash, Telfer School of Management, University of Ottawa, and
Ravi Varadhan, Johns Hopkins University Medical School

November 2016

Abstract

This article presents a discussion of optimization problems where the objective function $f(\mathbf{x})$ has parameters that are constrained by some scaling, so that $q(\mathbf{x}) = constant$, where this function $q()$ involves a sum of the parameters, their squares, or similar simple function.

1 Background

We consider problems where we want to minimize or maximize a function subject to a constraint that the sum of some function of the parameters, e.g., their sum of squares, must equal some constant. We refer to these problems as **sumscale** optimization problems. We have observed questions about problems like this on the R-help mailing list:

```
Jul 19, 2012 at 10:24 AM, Linh Tran <Tranlm@berkeley.edu> wrote:
> Hi fellow R users,
>
> I am desperately hoping there is an easy way to do this in R.
>
> Say I have three functions:
>
> f(x) = x^2
> f(y) = 2y^2
> f(z) = 3z^2
```

```
>
> constrained such that x+y+z=c (let c=1 for simplicity).
>
> I want to find the values of x,y,z that will minimize
f(x) + f(y) + f(z).
```

If the parameters x, y and z are non-negative, this problem can actually be solved as a Quadratic Program. We revisit this problem at the end of this article.

Other examples of this type of objective function are:

- The maximum volume of a regular polyhedron where the sum of the lengths of the sides is fixed.
- The minimum negative log likelihood for a multinomial model.
- The Rayleigh Quotient for the maximal or minimal eigensolutions of a matrix, where the eigenvectors should be normalized so the square norm of the vector is 1.

For the moment, let us consider a basic example, which is

Problem A : Minimize $(-\prod \mathbf{x})$ subject to $\sum \mathbf{x} = 1$ It is **assumed** for multinomial problems that the x elements are positive.

This is a very simplified version of the multinomial maximum likelihood problem.

Because these problems all have an objective that is dependent on a scaled set of parameters where the scale is defined by a sum, sum of squares, or similar sum of the parameters, we will refer to them as **sumscale** optimization problems. The condition that the parameters must be positive is often implicit. In practice it can be important to have it imposed explicitly if it is part of the actual problem.

2 Using general optimization with sumscale problems

Let us use the basic example above to consider how we might formulate Problem A to try to find a computational solution with R .

2.1 A direct approach

One possibility is to select one of the parameters and solve for it in terms of the others. Let this be the last parameter x_n, so that the set of parameters to be optimized is $\mathbf{y} = (x_1, x_1, ..., x_{n-1})$ where n is the original size of our problem. We now have the unconstrained problem

$$minimize(-(\prod \mathbf{y}) * (1 - \sum y))$$

This is easily coded and tried. We will use a very simple start, namely, the sequence $1, 2, ..., (n-1)$ scaled by $1/n^2$. We will also specify that the gradient is to be computed by a central approximation (Nash, 2013). At this point we are not requiring positive parameters, but our methods do return solutions with all positive parameters when started as in the example directly below.

```
cat("try loading optimrx\n")

## try loading optimrx

require(optimrx, quietly=TRUE)
pr <- function(y) {
- prod(y)*(1-sum(y))
}
cat("test the simple product for n=5\n")

## test the simple product for n=5

meth <- c("Nelder-Mead", "BFGS")
n<-5
m<-n-1
  st<-1:m/(m*m)
   ans<-opm(st, pr, gr="grcentral", control=list(trace=0))
   ao<-summary(ans,order=value)
print(ao)

##                    p1        p2        p3        p4         value fevals
## Nelder-Mead 0.1999973 0.1999963 0.2000065 0.2000013 -0.0003200000    201
## BFGS        0.2000550 0.1999015 0.2000793 0.1999630 -0.0003199999     20
##             gevals convergence kkt1 kkt2 xtime
## Nelder-Mead     NA           0 TRUE TRUE 0.004
## BFGS            17           0 TRUE TRUE 0.004

par <- as.double(ao[1,1:m])
par <- c(par, 1-sum(par))
par <- par/sum(par)
cat("Best parameters:")

## Best parameters:

print(par)

## [1] 0.1999973 0.1999963 0.2000065 0.2000013 0.1999986
```

While these codes work fine for small n, it is fairly easy to see that there will be computational difficulties as the size of the problem increases. Since the sum of the parameters is constrained to be equal to 1, the parameters are of the order of $1/n$, and the function therefore of the order of $1/(n^n)$, which underflows around $n = 144$ in R .

We do need to think about the positivity of the parameters. Let us start with a different set of values, some of which are negative. If pairs are negative, the product is still positive.

```
stm <- st*((-1)^(1:4))
print(stm)

## [1] -0.0625  0.1250 -0.1875  0.2500

ansm<-opm(stm, pr, gr="grcentral", control=list(trace=0))
aom <- summary(ansm, order=value)
print(aom)

##                        p1           p2            p3           p4
## BFGS        -2.954862e+57 2.003069e+57 -2.955447e+57 1.953545e+57
## Nelder-Mead -5.411717e+47 3.177738e+47 -2.292733e+46 2.395192e+47
##                     value fevals gevals convergence kkt1  kkt2 xtime
## BFGS        -6.676311e+286    300     30           0 TRUE FALSE 0.004
## Nelder-Mead -6.427455e+234   1001     NA           1 TRUE FALSE 0.004
```

Clearly this is not what we intended.

2.2 A log-likelihood approach

Traditionally, statisticians solve maximum likelihood problems by **minimizing** the negative log-likelihood. That is, the objective function is formed as (-1) times the logarithm of the likelihood. Using this idea, we convert our product to a sum. Choosing the first parameter to be the one determined by the summation constraint, we can write the function and gradient quite easily. Programs that try to find the minimum may try sets of parameters where some are zero or negative, so that logarithms of non-positive numbers are attempted, so we have put some safeguards in the function `nll` below. At this point we have assumed the gradient calculation is only attempted if the function can be computed satisfactorily, so we have not put similar safeguards in the gradient. Note that we work with $n-1$ parameters in the optimization, and expand to the full set for reporting.

```
nll <- function(y) { # large result if near zero arguments to log()
  if ((any(y <= 10*.Machine$double.xmin)) || (sum(y)>1-.Machine$double.eps))
         .Machine$double.xmax
```

```
  else   - sum(log(y)) - log(1-sum(y))
}
nll.g <- function(y) { - 1/y + 1/(1-sum(y))} # so far not safeguarded
```

We can easily try several optimization methods using the `opm()` function of `optimrx` package. Here are the calls, which overall did not perform as well as we would like. Note that we do not ask for `method="ALL"`. For one thing, this can take a lot of computing effort. Also, we found that some of the methods, in particular those using Powell's quadratic approximation methods, seem to get "stuck". The reasons for this have not been sufficiently understood to report at this time.

```
require(optimrx, quietly=TRUE)
n<-5
## mset<-c("L-BFGS-B", "BFGS", "CG", "spg", "ucminf", "nlm", "nlminb", "Rvmmin", "Rcgmin")
mset<-c("L-BFGS-B", "spg", "nlm", "nlminb", "Rvmmin", "Rcgmin")
a5<-opm(2:n/n^2, nll, gr="grfwd", method=mset, control=list(dowarn=FALSE))
a5g<-opm(2:n/n^2, nll, nll.g, method=mset, control=list(dowarn=FALSE))
a5gb<-opm(2:n/n^2, nll, nll.g, lower=0, upper=1, method=mset, control=list(dowarn=FALSE))
#- a5x <- opm(2:n/n^2, nll, nll.g, method="ALL", control=list(dowarn=FALSE))
summary(a5,order=value)

##                 p1        p2        p3        p4         value fevals
## Rcgmin   0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     51
## Rvmmin   0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     46
## spg      0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     17
## nlm      0.2000005 0.1999995 0.2000000 0.2000000  8.047190e+00     NA
## nlminb   0.2000004 0.1999990 0.1999989 0.1999992  8.047190e+00     23
## L-BFGS-B        NA        NA        NA        NA 8.988466e+307     NA
##          gevals convergence kkt1 kkt2 xtime
## Rcgmin       18           0 TRUE TRUE 0.004
## Rvmmin       13           0 TRUE TRUE 0.004
## spg          13           0 TRUE TRUE 0.044
## nlm          11           0 TRUE TRUE 0.000
## nlminb       12           0 TRUE TRUE 0.000
## L-BFGS-B     NA        9999   NA   NA 0.000

summary(a5g,order=value)

##                 p1        p2        p3        p4         value fevals
## Rvmmin   0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     43
## spg      0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     17
## Rcgmin   0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     28
## nlm      0.2000006 0.1999995 0.2000000 0.2000000  8.047190e+00     NA
## nlminb   0.2000004 0.1999990 0.1999989 0.1999992  8.047190e+00     23
## L-BFGS-B        NA        NA        NA        NA 8.988466e+307     NA
##          gevals convergence kkt1 kkt2 xtime
## Rvmmin       12           0 TRUE TRUE 0.000
## spg          13           0 TRUE TRUE 0.044
## Rcgmin       12           0 TRUE TRUE 0.004
## nlm          11           0 TRUE TRUE 0.000
## nlminb       12           0 TRUE TRUE 0.000
## L-BFGS-B     NA        9999   NA   NA 0.000
```

```
summary(a5gb,order=value)

##                 p1       p2        p3        p4         value fevals
## Rvmmin   0.2000000 0.200000 0.2000000 0.2000000  8.047190e+00     38
## Rcgmin   0.2000000 0.200000 0.2000000 0.2000000  8.047190e+00     18
## spg      0.2000000 0.200000 0.2000000 0.2000000  8.047190e+00     18
## nlminb   0.2000004 0.199999 0.1999989 0.1999992  8.047190e+00     23
## L-BFGS-B        NA       NA        NA        NA 8.988466e+307     NA
## nlm             NA       NA        NA        NA 8.988466e+307     NA
##          gevals convergence kkt1 kkt2 xtime
## Rvmmin       14           0 TRUE TRUE 0.004
## Rcgmin       10           0 TRUE TRUE 0.000
## spg          13           0 TRUE TRUE 0.044
## nlminb       12           0 TRUE TRUE 0.000
## L-BFGS-B     NA        9999   NA   NA 0.000
## nlm          NA        9999   NA   NA 0.000

#- summary(a5x,order=value)
```

Most, but not all, of the methods find the solution for the $n = 5$ case. The exception (L-BFGS-B) is due to the optimization method trying to compute the gradient where sum(x) is greater than 1. We have not tried to determine the source of this particular issue. However, it is almost certainly a consequence of too large a step. This method uses a quite sophisticated line search and its ability to use quite large search steps often results in very good performance. Here, however, the particular form of $log(1 - sum(x))$ is undefined once the argument of the logarithm is negative. Indeed, this is the basis of logarithmic barrier functions for constraints. There is a similar issue if any of the $n - 1$ parameters approach or pass zero. Negative parameter values are inadmissible in this formulation.

Numerical gradient approximations can similarly fail, particularly as step sizes are often of the order of 1E-7 in size. There is generally no special check within numerical gradient routines to apply bounds. Note also that a lower bound of 0 on parameters is not adequate, since $log(0)$ is undefined. Choosing a bound large enough to avoid the logarithm of a zero or negative argument while still being small enough to allow for parameter optimization is non-trivial.

2.3 Projected search directions

Objective functions defined by $(-1) * \prod \mathbf{x}$ or $(-1) * \sum log(\mathbf{x})$ will change with the scale of the parameters. Moreover, the constraint $\sum \mathbf{x} = 1$ effectively imposes the scaling $\mathbf{x_{scaled}} = \mathbf{x} / \sum \mathbf{x}$. The optimizer `spg` from package `BB` allows us to project our search direction to satisfy constraints. Thus, we could use the following approach.

```
require(BB, quietly=TRUE)
nllrv <- function(x) {- sum(log(x))}
nllrv.g <- function(x) {- 1/x }
proj <- function(x) {x/sum(x)}
n <- 5
tspg<-system.time(aspg <- spg(par=(1:n)/n^2, fn=nllrv, gr=nllrv.g, project=proj))[[3]]

## iter:  0  f-value:  11.30689  pgrad:  0.3607565

tspgn<-system.time(aspgn <- spg(par=(1:n)/n^2, fn=nllrv, gr=NULL, project=proj))[[3]]

## iter:  0  f-value:  11.30689  pgrad:  0.1333334

cat("Times: with gradient =",tspg,"   using numerical approx.=", tspgn,"\n")

## Times: with gradient = 0.045    using numerical approx.= 0.041

cat("F_optimal: with gradient=",aspg$value,"  num. approx.=",aspgn$value,"\n")

## F_optimal: with gradient= 8.04719   num. approx.= 8.04719

pbest<-rep(1/n, n)
cat("fbest = ",nllrv(pbest),"  when all parameters = ", pbest[1],"\n")

## fbest =  8.04719   when all parameters =  0.2

cat("deviations:  with gradient=",max(abs(aspg$par-pbest)),"   num. approx.=",max(abs(aspgn$par-pbest)),"\n")

## deviations:  with gradient= 3.81244e-06    num. approx.= 6.367897e-08
```

Here the projection `proj` is the key to success of method `spg`. The near-equality of timings with and without analytic gradient is because the approximation attempt only uses one iteration and two function evaluations to finish. In fact, the solution with approximate gradient is actually better, and this seems to carry over to cases with more parameters, e.g., 100 of them.

```
n<-100
tspgh<-system.time(aspgh <- spg(par=(1:n)/n^2, fn=nllrv, gr=nllrv.g, project=proj))[[3]]

## iter:  0  f-value:  557.2947  pgrad:  0.1925703

tspgnh<-system.time(aspgnh <- spg(par=(1:n)/n^2, fn=nllrv, gr=NULL, project=proj))[[3]]

## iter:  0  f-value:  557.2947  pgrad:  0.00980205

cat("Times: with gradient =",tspgh,"   using numerical approx.=", tspgnh,"\n")

## Times: with gradient = 0.054    using numerical approx.= 0.044
```

```
cat("F_optimal: with gradient=",aspgh$value,"  num. approx.=",aspgnh$value,"\n")

## F_optimal: with gradient= 460.517   num. approx.= 460.517

pbesth<-rep(1/n, n)
cat("fbest = ",nllrv(pbesth),"  when all parameters = ", pbesth[1],"\n")

## fbest =  460.517   when all parameters =  0.01

cat("deviations:  with gradient=",max(abs(aspgh$par-pbesth)),"   num. approx.=",max(abs(aspgnh$par-pbesth)),"\n")

## deviations:  with gradient= 5.055715e-07    num. approx.= 7.006446e-08
```

Larger n values eventually give difficulties as non-positive parameters are produced at intermediate stages of the optimization.

Minimization methods other than `spg` do not have the flexibility to impose the projection directly. We would need to carefully build the projection into the function(s) and/or the method codes. This was done by Geradin (1971) for the Rayleigh quotient problem, but requires a number of changes to the program code.

2.4 *log*() transformation of parameters

When problems give difficulties, it is common to re-formulate them by transformations of the function or the parameters. A common method to ensure parameters are positive is to use a log transform. In the present case, optimizing over parameters that are the logarithms of the parameters above ensures we have positive arguments to most of the elements of the negative log likelihood. Here is the code. Note that the parameters used in optimization are "lx" and not x.

```
enll <- function(lx) {
    x<-exp(lx)
    fval<-  - sum( log( x/sum(x) ) )
}
enll.g <- function(lx){
    x<-exp(lx)
    g<-length(x)/sum(x) - 1/x
    gval<-g*exp(lx)
}
```

But where is our constraint that the sum of parameters must be 1? Here we have noted that we could define the objective function only to within the scaling $\mathbf{x}/\sum(\mathbf{x})$. There is a minor nuisance, in that we need to re-scale our parameters after solution to have them in a standard form. This is most noticeable if one uses `optimrx` function `opm()` and displays the results of `method = mset`, a collection of six gradient-based minimizers. In the following, we extract the best solution for the 5-parameter problem.

```
require(optimrx, quietly=TRUE) # just to be sure
st<-1:5/10 # 5 parameters, crude scaling to start
st<-log(st)
n <- 5
mset<-c("L-BFGS-B", "spg", "nlm", "nlminb", "Rvmmin", "Rcgmin")
a5x<-opm(st, enll, enll.g, method=mset, control=list(trace=0))
a5xbyvalue<-summary(a5x, order=value)
print(a5xbyvalue)[(n+1):(n+7)]

##                 p1        p2        p3        p4        p5   value fevals
## spg      -1.345087 -1.345087 -1.345087 -1.345087 -1.345087 8.04719      6
## Rvmmin   -1.345087 -1.345087 -1.345087 -1.345087 -1.345087 8.04719     26
## Rcgmin   -1.345087 -1.345087 -1.345087 -1.345087 -1.345087 8.04719     11
## nlminb   -1.345087 -1.345087 -1.345087 -1.345087 -1.345087 8.04719      8
## nlm      -1.345084 -1.345085 -1.345086 -1.345088 -1.345091 8.04719     NA
## L-BFGS-B -1.345080 -1.345087 -1.345085 -1.345088 -1.345093 8.04719      7
##          gevals convergence kkt1  kkt2 xtime
## spg           5           0 TRUE FALSE 0.044
## Rvmmin        8           0 TRUE FALSE 0.000
## Rcgmin        8           0 TRUE FALSE 0.000
## nlminb        8           0 TRUE FALSE 0.000
## nlm           6           0 TRUE FALSE 0.000
## L-BFGS-B      7           0 TRUE FALSE 0.000
##            value fevals gevals convergence kkt1  kkt2 xtime
## spg      8.04719      6      5           0 TRUE FALSE 0.044
## Rvmmin   8.04719     26      8           0 TRUE FALSE 0.000
## Rcgmin   8.04719     11      8           0 TRUE FALSE 0.000
## nlminb   8.04719      8      8           0 TRUE FALSE 0.000
## nlm      8.04719     NA      6           0 TRUE FALSE 0.000
## L-BFGS-B 8.04719      7      7           0 TRUE FALSE 0.000

xnor<-exp(a5xbyvalue[1, 1:5]) # get the 5 parameters of "best" solution, exponentiate
xnor<-xnor/sum(xnor)
cat("best normalized parameters:")

## best normalized parameters:

print(xnor)

##      p1  p2  p3  p4  p5
## spg 0.2 0.2 0.2 0.2 0.2
```

While there are reasons to think that the indeterminacy might upset the optimization codes, in practice, the objective and gradient above are generally well-behaved, though they did reveal that tests of the size of the gradient used, in particular, to decide to terminate iterations in `Rcgmin` were too hasty in stopping progress for problems with larger numbers of parameters. A user-specified tolerance is now allowed; for example control=list(tol=1e-32), the rather extreme setting we have used below.

Let us try a larger problem in 100 parameters. We emply the conjugate gradient algorithm in package Rcgmin.

```
## Initial function value = 460.5587
## Time =  0.001   fval= 460.517
## Average parameter is  0.01  with max deviation 6.290762e-08
```

We have a solution. However, a worrying aspect of this solution is that the objective function at the start and end differ by a tiny amount.

2.5 A transformation inspired by the n-sphere

A slightly different transformation or projection is inspired by spherical coordinates. See https://en.wikipedia.org/wiki/N-sphere.

The idea here is to transform a set of n parameters by specifying $n-1$ values and letting a special projection transform this set of $n-1$ numbers into a set of n parameters that always sum to 1.

The first such transformation uses the trigonometric identity that $sin^2(theta) + cos^2(theta) = 1$.

This identity is extended to n dimensions. We can do this via the projection

```
proj1 <- function(theta) {
   s2 <- sin(theta)^2
    cumprod(c(1, s2)) * c(1-s2, 1)
}
```

You can easily verify that this produces a set of parameters that sum to 1 by setting $theta = (a, b)$ for a problem in 3 parameters. However, we do not need to use the $sin()$ function, as any transformation onto the unit line segment [0,1] will work. proj2 below works fine, and can be verified for 3 parameters as with proj1, though the parameters are different. (Caution!)

We solve problems in 5 and 100 parameters using the spg() function from package BB, and by not specifying a gradient function use an internal approximation.

```
## ?? need to explain this better
proj2 <- function(theta) {
    theta2 <- theta^2
    s2 <- theta2 / (1 + theta2)
    cumprod(c(1, s2)) * c(1-s2, 1)
}
obj <- function(theta) { - sum(log(proj2(theta))) }
n <- 5
ans <- spg(seq(n-1), obj)
```

```
## iter:  0  f-value:  11.15175  pgrad:  3
## iter:  10  f-value:  8.78015  pgrad:  0.5806909
## iter:  20  f-value:  8.04719  pgrad:  3.925749e-06

proj2(ans$par) # The parameters

## [1] 0.2000000 0.2000007 0.2000002 0.1999996 0.1999995
```

```
n<-100
ans100 <- spg(seq(n-1), obj, control=list(trace=FALSE), quiet=TRUE)
proj2(ans100$par)[1:5] # Display only 1st 5 parameters

## [1] 0.009999999 0.010000001 0.010000000 0.010000000 0.009999999
```

In the above, we note that the transformation is embedded into the objective function, so we could run any of the optimizers in `optimrx` as follows. This can take some time, and the derivative-free methods do an awful lot of work with this formulation, though they do seem to get the best solution. We have omitted the results for these, as they make the rendering of this document unacceptably slow with `knitr`. Moreover, `Rcgmin` and `Rvmmin` are not recommended when an analytic gradient is not provided. Here we have specified that a simple forward difference approximation to the gradient should be used.

```
sans<- opm(seq(n-1), obj, gr="grfwd", method=mset, control=list(dowarn=FALSE))
## summary(allans, order = "list(round(value, 3), fevals)", par.select = FALSE)
summary(sans, order = value, par.select = FALSE)

##              value fevals gevals convergence  kkt1  kkt2 xtime
## Rvmmin   460.5170    398    272           2  TRUE  TRUE 0.664
## spg      460.5170    230    212           0  TRUE  TRUE 0.440
## Rcgmin   460.5170   1612   1023           0  TRUE  TRUE 1.868
## nlm      460.5170     NA    203           0  TRUE  TRUE 0.640
## L-BFGS-B 460.5170    280    280           0  TRUE  TRUE 0.492
## nlminb   482.2372    189    151           1 FALSE FALSE 0.332
```

2.6 Fixing parameters

Some function minimizers can specify that some parameters are fixed. `Rvmmin` and `Rcgmin` are two such methods. Let us work with the `enll` objective function that works with the logarithms of the parameters. We need to rescale the parameters after solution and recompute the objective if we need it.

```
n<-5
mmth <- c("Rvmmin", "Rcgmin")
```

```
strt <- (1:n)/n
lo <- c(rep(-100, (n-1)),strt[n])
up <- c(rep(100, (n-1)),strt[n])
amsk1 <- opm(strt, enll, enll.g, lower=lo, upper=up, method=mmth)

## Warning in bmchk(par, lower = lower, upper = upper):  Masks (fixed parameters) set
by bmchk due to tight bounds.  CAUTION!!
## Warning in bmchk(par, lower = lower, upper = upper):  Masks (fixed parameters) set
by bmchk due to tight bounds.  CAUTION!!
## Warning in bmchk(par, lower = lower, upper = upper):  Masks (fixed parameters) set
by bmchk due to tight bounds.  CAUTION!!

print(amsk1)

##               p1        p2        p3        p4 p5   value fevals gevals
## Rvmmin 1.0000000 1.0000000 1.0000000 1.0000000  1 8.04719     32     10
## Rcgmin 0.9999998 0.9999998 0.9999998 0.9999997  1 8.04719     29     13
##        convergence kkt1  kkt2 xtime
## Rvmmin           0 TRUE FALSE 0.000
## Rcgmin           0 TRUE FALSE 0.004

amsk1 <- summary(amsk1, order=value)
parmsk <- amsk1[1, 1:n]
parmsk <- parmsk/sum(parmsk)
print(parmsk)

##         p1  p2  p3  p4  p5
## Rvmmin 0.2 0.2 0.2 0.2 0.2
```

This also works well for $n = 100$ with both methods that allow fixed parameters.

Note that for the product problem, no parameter can be zero or the product is zero. In the case of the Rayleigh Quotient below, we may have to be concerned that the parameter we have chosen to fix may have zero as its optimal value, in which case the approach of fixing one parameter is not useful unless we are prepared to indulge some trial and error.

2.7 Use the gradient equations

Another approach is to "solve" the gradient equations as a nonlinear equations problem. We could do this with a sum of squares minimizer, though the **nls** function in R is specifically NOT useful as it cannot deal with small or zero residuals. However, **nlfb** from package **nlmrt** is capable of dealing with such problems. Unfortunately, it will be slow as it has to generate the Jacobian by numerical approximation unless we can provide a function to prepare the Jacobian analytically. Moreover, the determination of the Jacobian is still subject to the unfortunate scaling issues we have been confronting throughout this article. A better approach would likely be via package **nleqslv**, which is constructed to attempt solutions of nonlinear equations problems.

```
## Problem of order 5 using nll.g
## nleqslv: rescaled parameter mean= 0.2  with SD= 0.09352086
## Problem of order 5 using enll.g
## nleqslv: rescaled parameter mean= 0.2  with SD= 9.573944e-10
## nlmrt: rescaled parameter mean= 0.2  with SD= 1.962616e-17
##
## Now 100 parameters using enll.g
## nleqslv: rescaled parameter mean= 0.01  with SD= 0.00574485
## nlmrt: rescaled parameter mean= 0.01  with SD= 0.005999938
```

The results with the larger problem are not acceptable.

3 The Rayleigh Quotient

This is another typical sumscale problem. The maximal and minimal eigensolutions of a symmetric matrix A are extrema of the Rayleigh Quotient

$R(x) = (x'Ax)/(x'x)$

We can also deal with generalized eigenproblems of the form

$Ax = eBx$

where B is symmetric and positive definite by using the Rayleigh Quotient

$R_g(x) = (x'Ax)/(x'Bx)$

Once again, the objective is scaled by the parameters, this time by their sum of squares. Alternatively, we may think of requiring the **normalized** eigensolution, which is given as

$x_{normalized} = x/sqrt(x'x)$

We will first try the projected gradient method `spg` from `BB`. Below is the code, where our test uses a matrix called the Moler matrix (Nash, 1979, Appendix 1). We caution that there are faster ways to compute this matrix in `R` (Nash, 2012), where different approaches to speed up `R` computations are discussed. Here we are concerned with getting the solutions correctly rather than the speed of so doing. Note that to get the solution with the most-positive eigenvalue, we minimize the Rayleigh quotient of the matrix multiplied by -1. This is solution `tmax`.

```
## Test spg on Moler matrix of order= 5
## minimal eigensolution: Value= 0.008646272 in time  0.041
## [1] 0.86189793 0.43282060 0.22109168 0.12038989 0.08014382
## Eigenvalue solution test: 9.95032e-07    Normtest: 0
## [1] "OK"
## maximal eigensolution: Value= -7.4875 in time  0.038
```

```
## [1] -0.25565401  0.04658842  0.34034057  0.57207137  0.69955213
## Eigenvalue solution test: 2.882608e-06    Normtest: 0
## [1] "OK"
## Test spg on Moler matrix of order= 100
## minimal eigensolution: Value= 9.6134e-08 in time  3.132
##   [1]  8.660199e-01  4.330273e-01  2.165021e-01  1.082429e-01  5.413318e-02
##   [6]  2.707444e-02  1.351570e-02  6.765715e-03  3.397268e-03  1.676827e-03
##  [11]  8.529906e-04  4.286361e-04  1.874283e-04  1.419646e-04  1.543479e-05
##  [16]  5.551854e-05 -2.948453e-06  1.187565e-05  2.541557e-06  5.982898e-07
##  [21]  6.157890e-07  1.435287e-06  2.010798e-06 -1.858601e-06 -9.563769e-06
##  [26]  2.296545e-05 -2.413628e-05  6.941454e-06  1.756824e-05 -3.022414e-05
##  [31]  2.168006e-05 -6.356585e-06 -5.216143e-06  1.525886e-05 -2.837107e-05
##  [36]  3.163633e-05 -1.295166e-05 -1.509423e-05  2.113325e-05 -4.667952e-06
##  [41] -1.073051e-05  1.472164e-05 -1.740072e-05  1.851975e-05 -1.226904e-05
##  [46] -8.233470e-07  7.453441e-06  1.517933e-06 -2.236582e-05  3.467387e-05
##  [51] -2.314354e-05 -2.837550e-06  1.240029e-05  2.042405e-06 -1.586118e-05
##  [56]  1.261238e-05 -1.108035e-05  2.494323e-05 -3.368587e-05  1.395187e-05
##  [61]  1.725784e-05 -3.034134e-05  2.366341e-05 -9.802624e-06 -7.208625e-06
##  [66]  1.959506e-05 -2.139587e-05  2.109486e-05 -2.506720e-05  3.032335e-05
##  [71] -3.744645e-05  4.624850e-05 -4.885395e-05  4.365542e-05 -3.302046e-05
##  [76]  1.482298e-05  5.412972e-06 -9.574262e-06  1.756240e-06 -3.610572e-06
##  [81]  1.163764e-05 -3.404909e-06 -1.463169e-05  2.683693e-05 -3.231970e-05
##  [86]  2.923217e-05 -1.458679e-05  8.775778e-06 -2.258634e-05  4.090556e-05
##  [91] -4.723804e-05  3.996571e-05 -2.098811e-05  8.017338e-06 -1.124266e-05
##  [96]  1.293982e-05  1.528742e-06 -8.489635e-06 -1.335459e-06  4.599613e-06
## Eigenvalue solution test: 0.0001728295    Normtest: 4.440892e-16
## [1] "OK"
## maximal eigensolution: Value= -3934.277 in time  0.097
##   [1] -2.271738e-03  5.278741e-07  2.272797e-03  4.544487e-03  6.815021e-03
##   [6]  9.083821e-03  1.135031e-02  1.361392e-02  1.587406e-02  1.813016e-02
##  [11]  2.038166e-02  2.262797e-02  2.486853e-02  2.710276e-02  2.933010e-02
##  [16]  3.154998e-02  3.376184e-02  3.596511e-02  3.815924e-02  4.034366e-02
##  [21]  4.251783e-02  4.468118e-02  4.683316e-02  4.897324e-02  5.110086e-02
##  [26]  5.321549e-02  5.531659e-02  5.740362e-02  5.947605e-02  6.153335e-02
##  [31]  6.357501e-02  6.560050e-02  6.760930e-02  6.960092e-02  7.157483e-02
##  [36]  7.353054e-02  7.546755e-02  7.738537e-02  7.928351e-02  8.116149e-02
##  [41]  8.301883e-02  8.485505e-02  8.666970e-02  8.846231e-02  9.023242e-02
##  [46]  9.197958e-02  9.370335e-02  9.540329e-02  9.707898e-02  9.872997e-02
##  [51]  1.003559e-01  1.019562e-01  1.035307e-01  1.050788e-01  1.066002e-01
##  [56]  1.080944e-01  1.095612e-01  1.110001e-01  1.124108e-01  1.137930e-01
##  [61]  1.151461e-01  1.164700e-01  1.177643e-01  1.190286e-01  1.202627e-01
##  [66]  1.214661e-01  1.226387e-01  1.237801e-01  1.248900e-01  1.259682e-01
##  [71]  1.270143e-01  1.280281e-01  1.290094e-01  1.299578e-01  1.308733e-01
##  [76]  1.317554e-01  1.326040e-01  1.334189e-01  1.341999e-01  1.349467e-01
##  [81]  1.356592e-01  1.363372e-01  1.369806e-01  1.375891e-01  1.381626e-01
##  [86]  1.387010e-01  1.392041e-01  1.396719e-01  1.401041e-01  1.405006e-01
##  [91]  1.408615e-01  1.411865e-01  1.414756e-01  1.417287e-01  1.419458e-01
##  [96]  1.421268e-01  1.422717e-01  1.423803e-01  1.424528e-01  1.424890e-01
## Eigenvalue solution test: 0.0001959908    Normtest: 0
## [1] "OK"
```

For the record, these results compare well with eigenvalues from eigen(), but the timings are **slower** than the `eigen` function that computes all the solutions for the order 100 matrix. The main saving of the optimization approach is space if we compute the Rayleigh Quotient without actually forming the matrix.

For comparison, we also ran a specialized Geradin routine as implemented in R by one of us (JN). This gave equivalent answers, albeit more efficiently. For those interested, the Geradin routine is available as referenced in (Nash, 2012).

4 The R-help example

As a final example, let us use our present techniques to solve the problem posed by Lanh Tran on R-help. We will use only a method that scales the parameters directly inside the objective function and not bother with gradients for this small problem.

```
ssums<-function(x){
  n<-length(x)
  tt<-sum(x)
  ss<-1:n
  xx<-(x/tt)*(x/tt)
  sum(ss*xx)
}
## Try penalized sum
st<-runif(3)
aos<-opm(st, ssums, gr="grcentral", method=mset)
# rescale the parameters
nsol<-dim(aos)[1]
for (i in 1:nsol){
  tpar<-aos[i,1:3]
  ntpar<-sum(tpar)
  tpar<-tpar/ntpar
  aos[i, 1:3]<-tpar
}
summary(aos,order=value)

##                 p1        p2        p3     value fevals gevals convergence
## Rvmmin   0.5454546 0.2727273 0.1818182 0.5454545     14     14           2
## L-BFGS-B 0.5454546 0.2727273 0.1818182 0.5454545      8      8           0
## nlm      0.5454545 0.2727273 0.1818182 0.5454545     NA     12           0
## nlminb   0.5454546 0.2727273 0.1818181 0.5454545      8      8           0
## spg      0.5454546 0.2727272 0.1818182 0.5454545      9      8           0
## Rcgmin   0.5454547 0.2727272 0.1818181 0.5454545     13      9           0
##          kkt1  kkt2 xtime
## Rvmmin   TRUE FALSE 0.004
## L-BFGS-B TRUE FALSE 0.000
## nlm      TRUE FALSE 0.000
## nlminb   TRUE FALSE 0.004
## spg      TRUE FALSE 0.040
## Rcgmin   TRUE FALSE 0.004
```

We can also use a projection method.

```r
ssum<-function(x){
  n<-length(x)
  ss<-1:n
  xx<-x*x
  sum(ss*xx)
}
proj.simplex <- function(y) {
# project an n-dim vector y to the simplex Dn
# Dn = { x : x n-dim, 1 >= x >= 0, sum(x) = 1}
# Ravi Varadhan, Johns Hopkins University
# August 8, 2012

n <- length(y)
sy <- sort(y, decreasing=TRUE)
csy <- cumsum(sy)
rho <- max(which(sy > (csy - 1)/(1:n)))
theta <- (csy[rho] - 1) / rho
return(pmax(0, y - theta))
}
as<-spg(st, ssum, project=proj.simplex)

## iter:  0  f-value:  0.8755125  pgrad:  0.9248068
## iter:  10  f-value:  0.5454545  pgrad:  8.16606e-06
```

```
## Using project.simplex with spg: fmin= 0.5454545  at
## [1] 0.5454517 0.2727294 0.1818189
```

Apart from the parameter rescaling, this is an entirely "doable" problem. Note that we can also solve the problem as a Quadratic Program using the `quadprog` package.

```r
library(quadprog)
Dmat<-diag(c(1,2,3))
Amat<-matrix(c(1, 1, 1), ncol=1)
bvec<-c(1)
meq=1
dvec<-c(0, 0, 0)
ans<-solve.QP(Dmat, dvec, Amat, bvec, meq=0, factorized=FALSE)
ans

## $solution
## [1] 0.5454545 0.2727273 0.1818182
##
## $value
## [1] 0.2727273
##
## $unconstrained.solution
## [1] 0 0 0
##
## $iterations
## [1] 2 0
##
```

```
## $Lagrangian
## [1] 0.5454545
##
## $iact
## [1] 1
```

5 Recommendations

Despite the relatively limited experience above, it is nevertheless incumbent on us to recommend to R users how to approach these problems.

Overall, the approaches that gave the least trouble computationally were those in which the indeterminacy caused by the sumscale constraint was explicitly removed. That can be done either with a solution for one parameter, or else by fixing one parameter.

When it can be used in a straightforward manner, the gradient projection method `spg` is quite effective. The main concern is to get the specification of the projection correct. The ideas can be extended to programs customized to particular types of problems, as in the Geradin approach to Rayleigh Quotient minimization, but the effort is clearly only justified when many such problems must be solved, and solved quickly.

If parameters are required to be positive, then working with their logarithms is a sensible transformation.

5.1 Application of the ideas to a new example

Let us try to solve the following problem:

Maximize the product of n parameters such that their weighted sum of squares is fixed to value A. The particular scaling is

$\sum_{i=1}^{n} i * {x_i}^2 = A$

We will set $A = 1$, but other values will simply scale the objective function. Defining

$scale(\mathbf{x}) = \sqrt{(\sum_{i=1}^{n} i * {x_i}^2)}$

we want to maximize

$\prod \mathbf{x}/scale(\mathbf{x})$

As before, we should **minimize**

$-\sum_{i=1}^{n} log(x_i) + log(scale(\mathbf{x}))$

so that we can avoid the product becoming too large or too small. Thus we create the functions

```
scale <- function(x) {sum( (1:n) * x^2)}
enew <- function(x){
  n <- length(x)
##  x <- exp(lx)
  sc <- sqrt(scale(x))
  obj <- - sum(log(x/sc))
#  obj <- n*log(sc)-sum(log(x))
}
```

These functions involve all the parameters, so we apply them with solvers that can fix one parameter. Note that we set lower and upper bounds to the other parameters, and must scale the results before reporting them. The specification of fixed parameters trips a warning that we have put lower and upper bounds equal. At the time of writing, this warning cannot be suppressed. Indeed, it may be inadvisable to do so, though users could easily modify function `bmchk` in package **optextras** .

```
mmth <- c("Rvmmin", "Rcgmin")
library(optimrx)
n <- 5
mmth<-c("Rvmmin", "Rcgmin")
mset<-c("Rcgmin", "ucminf", "Nelder-Mead")
strt <- (1:n)/(2*n)
lo <- c(rep(1e-15, (n-1)),strt[n])
up <- c(rep(100, (n-1)),strt[n])
amsk2 <- opm(strt, enew, gr="grcentral", lower=lo, upper=up, method=mmth)

## Warning in bmchk(par, lower = lower, upper = upper):  Masks (fixed parameters) set
by bmchk due to tight bounds.  CAUTION!!
## Warning in bmchk(par, lower = lower, upper = upper):  Masks (fixed parameters) set
by bmchk due to tight bounds.  CAUTION!!
## Warning in bmchk(par, lower = lower, upper = upper):  Masks (fixed parameters) set
by bmchk due to tight bounds.  CAUTION!!

## amsk2 <- opm(strt, enew, gr="grcentral", method=mmth)
amsk2 <- summary(amsk2, order=value)
print(amsk2)

##              p1        p2        p3       p4  p5    value fevals gevals
## Rvmmin 1.118034 0.7905694 0.6454972 0.559017 0.5 6.417341     20     17
## Rcgmin 1.118034 0.7905695 0.6454973 0.559017 0.5 6.417341     45     23
##        convergence kkt1  kkt2 xtime
## Rvmmin           2 TRUE FALSE 0.008
## Rcgmin           0 TRUE FALSE 0.008

parm2 <- amsk2[1, 1:n]
print(parm2)
```

```
##              p1        p2        p3       p4  p5
## Rvmmin 1.118034 0.7905694 0.6454972 0.559017 0.5

parm2 <- as.numeric(parm2/sqrt(scale(parm2)))
## parm2e<-as.numeric(exp(parm2))
print(parm2)

## [1] 0.4472136 0.3162278 0.2581989 0.2236068 0.2000000

cat("enew(parm2)=", enew(parm2),"\n")

## enew(parm2)= 6.417341

print(scale(parm2))

## [1] 1

areg<- opm(strt, enew, gr="grnd", method=mset)

## Warning in log(x/sc):  NaNs produced
## Warning in Rcgminu(par = spar, fn = efn, gr = egr, control = mcontrol, ...):  Rcgmin
- undefined function
## Warning in log(x/sc):  NaNs produced
## Warning in Rcgminu(par = spar, fn = efn, gr = egr, control = mcontrol, ...):  Rcgmin
- undefined function

areg

##                    p1        p2        p3        p4        p5    value
## Rcgmin      0.5707154 0.4035568 0.3295027 0.2853577 0.2552317 6.417341
## ucminf      1.1141731 0.7878393 0.6432681 0.5570865 0.4982733 6.417341
## Nelder-Mead 0.6300243 0.4449968 0.3629333 0.3145654 0.2812454 6.417344
##             fevals gevals convergence kkt1  kkt2 xtime
## Rcgmin          32     15           0 TRUE FALSE 0.012
## ucminf          15     15           0 TRUE FALSE 0.008
## Nelder-Mead    216     NA           0 TRUE FALSE 0.000

for (ii in 1:dim(areg)[1]){
    prm <- areg[ii,1:5]
#    print(prm)
    prm<-as.numeric(prm/sqrt(scale(prm)))
    cat(names(areg)[ii]," scaled params:")
    print(prm)
    cat("new scale=", scale(prm),"\n")
}

## p1  scaled params:[1] 0.4472136 0.3162278 0.2581989 0.2236068 0.2000000
## new scale= 1
## p2  scaled params:[1] 0.4472136 0.3162278 0.2581989 0.2236068 0.2000000
## new scale= 1
## p3  scaled params:[1] 0.4478024 0.3162904 0.2579621 0.2235837 0.1999008
## new scale= 1
```

Moreover we could work with logarithms of parameters to avoid non-positive inputs to the objective function.

```
scalel <- function(lx){
  x<-exp(lx)
  scale(x)
}
enewl <- function(lx){
  x<-exp(lx)
  enew(x)
}
```

```
## Warning in bmchk(par, lower = lower, upper = upper):  Masks (fixed parameters) set
by bmchk due to tight bounds.  CAUTION!!
## Warning in bmchk(par, lower = lower, upper = upper):  Masks (fixed parameters) set
by bmchk due to tight bounds.  CAUTION!!
## Warning in bmchk(par, lower = lower, upper = upper):  Masks (fixed parameters) set
by bmchk due to tight bounds.  CAUTION!!

##              p1        p2        p3       p4  p5    value fevals gevals
## Rvmmin 1.118034 0.7905694 0.6454972 0.559017 0.5 6.417341     20     17
## Rcgmin 1.118034 0.7905695 0.6454973 0.559017 0.5 6.417341     45     23
##        convergence kkt1  kkt2 xtime
## Rvmmin           2 TRUE FALSE 0.008
## Rcgmin           0 TRUE FALSE 0.008
## [1] 0.4094367 0.2951010 0.2552505 0.2341040 0.2206876
## enew(parm2)= 6.417341
## [1] 1
##                     p1        p2        p3        p4        p5    value
## Rcgmin      -0.8663376 -1.212911 -1.415644 -1.559485 -1.671057 6.417341
## ucminf      -0.8663376 -1.212911 -1.415644 -1.559485 -1.671057 6.417341
## Nelder-Mead -0.9447184 -1.291208 -1.493720 -1.637800 -1.749406 6.417341
##             fevals gevals convergence kkt1  kkt2 xtime
## Rcgmin          13      8           0 TRUE FALSE 0.008
## ucminf          12     12           0 TRUE FALSE 0.008
## Nelder-Mead    282     NA           0 TRUE FALSE 0.004
## Rcgmin  scaled params:[1] 0.4472136 0.3162278 0.2581989 0.2236068 0.2000000
## new scale= 1
## ucminf  scaled params:[1] 0.4472136 0.3162277 0.2581989 0.2236068 0.2000000
## new scale= 1
## Nelder-Mead  scaled params:[1] 0.4471702 0.3162237 0.2582526 0.2235996 0.1999868
## new scale= 1
```

The "solve for one parameter" approach requires slightly more work. We find x_1 as

$x_1 = sqrt(A - \sum_{i=2}^{n} {x_i}^2)$

It is fairly easy to set up the objective using this expression, but we note that the starting parameters need to obey the scaling, or we may not be able to proceed. We also must note that the names of the parameters reported by the optimization are all shifted one place, so `p1` is really the original `p2`. To properly report the results, we should solve for x_1 and put the value as the first element of an augmented solution vector.

```r
es1 <- function(x){ ## safeguarded objective
  n <- length(x)+1
  sx <- sum((2:n)*x^2)
  if (sx >= 1) {obj <- 1e70}
  else {obj <- - sum(log(x)) - log(sqrt(1-sx)) }
}
n <- 5
mset<-c("Rcgmin", "ucminf", "Nelder-Mead")
strt <- (1:n)/(2*n)
strt<-strt/sqrt(scale(strt))
as1 <- opm(strt[2:n], es1, gr="grcentral", method=mset)
## amsk2 <- opm(strt, enew, gr="grcentral", method=mmth)
as1 <- summary(as1, order=value)
as1

##                    p1        p2        p3        p4    value fevals gevals
## Rcgmin      0.3162278 0.2581989 0.2236068 0.2000000 6.417341     50     26
## ucminf      0.3162278 0.2581989 0.2236068 0.2000000 6.417341     24     24
## Nelder-Mead 0.3162849 0.2582493 0.2235563 0.1999534 6.417341    295     NA
##             convergence kkt1 kkt2 xtime
## Rcgmin                0 TRUE TRUE 0.004
## ucminf                0 TRUE TRUE 0.004
## Nelder-Mead           0 TRUE TRUE 0.004
```

The projection approach was less successful with this problem.

```r
proj5 <- function(x) { x/sqrt(scale(x))}
proj5l <- function(lx) { exp(lx)/scale(exp(lx))}
n <- 5
strt <- (1:n)/(2*n)
strt <- strt/sqrt(scale(strt))
aspg2 <- try(spg(strt, enew, project=proj5))

## iter:  0  f-value:  8.752759  pgrad:  0.6334095

## Warning in log(x/sc):  NaNs produced
## Warning in spg(strt, enew, project = proj5):  Unsuccessful convergence.

if ((class(aspg2) != "try-error") && (aspg2$convergence == 0)) {
   print(aspg2)
} else { cat("spg failed\n") }

## spg failed

strtl <- log(strt)
aspg2l <- try(spg(strtl, enewl, project=proj5l))
if ((class(aspg2l) != "try-error") && (aspg2l$convergence == 0)) {
   print(aspg2l)
} else { cat("spg failed\n") }

## spg failed
```

6 Conclusion

Sumscale problems can present difficulties for optimization (or function minimization) codes. These difficulties are by no means insurmountable, but they do require some attention.

While specialized approaches are "best" for speed and correctness, a general user is more likely to benefit from a simpler approach of embedding the scaling in the objective function by a "solve for one parameter" approach or by fixing one parameter. It is often important to transform parameters to ensure positivity, and it may be necessary to rescale parameters before reporting them.

When a projection is well-understood, the projected gradient via `spg` from package `BB` may be helpful.

References

M. Geradin. The computational efficiency of a new minimization algorithm for eigenvalue analysis. *J. Sound Vib.*, 19:319–331, 1971.

J. C. Nash. *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation.* Adam Hilger, Bristol, 1979. Second Edition, 1990, Bristol: Institute of Physics Publications.

J. C. Nash. Timing Rayleigh Quotient minimization in R , July 2012. This is a 'tip' on http://rwiki.sciviews.org/doku.php?id=tips:rqcasestudy. The files related to this study are `https://r-forge.r-project.org/scm/viewvc.php/*checkout*/RQdoc/RQtimes.pdf?revision=1322&root=optimizer` and `https://r-forge.r-project.org/scm/viewvc.php/*checkout*/RQdoc/RQtimes.Rnw?revision=1322&root=optimizer`.

J. C. Nash. *optextras: Tools to Support Optimization Possibly with Bounds and Masks*, 2013. URL `https://CRAN.R-project.org/package=optextras`. R package version 2013-10.27.