?? find new reference for RQ times ?? put in eigen ?? clean up warnings

```
## Warning in dep_auto(): corrupt dependency files?
## try remove cache/__objects; cache/__globals
```

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 $\sum \mathbf{x} = 1$

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 paramters, we will refer to them as **sumscale** optimization problems.

2 Difficulties using general optimization with sumscale problems

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

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).

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

## try loading optimrx

require(optimx, 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
  st<-1:(n-1)/(n*n)
   ans<-opm(st, pr, gr="grcentral", control=list(trace=0))
   ao<-summary(ans,order=value)
print(ao)

##                    p1        p2        p3        p4    value fevals gevals
## BFGS        0.2000000 0.1999986 0.2000037 0.1999986 -0.00032    102     96
## Nelder-Mead 0.2000034 0.1999983 0.2000017 0.2000021 -0.00032    331     NA
##             convergence kkt1 kkt2 xtime
## BFGS                  0 TRUE TRUE 0.006
## Nelder-Mead           0 TRUE TRUE 0.002
```

While these codes work fine for small n, it is fairly easy to see that there are computational problems 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.

3 Other formulations

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. This converts 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. As programs that try to find the minimum may change the parameters so that logarithms of non-positive numbers are attempted, we have put some safeguards in the function `nll`. At this point we have assumed the gradient calculation is only attempted if the function can be computed satisfactorily, so we have not put safeguards in the gradient.

```
nll <- function(y) {
  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 `optimx` 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"` as we found that some of the methods, in particular those using Powell's quadratic approximation methods, seem to get "stuck".

```
require(optimrx, quietly=TRUE)
n<-5
mset<-c("L-BFGS-B", "BFGS", "CG", "spg", "ucminf", "nlm", "nlminb", "Rvmmin", "Rcgmin")
a5<-opm(2:n/n^2, nll, gr="grfwd", method=mset, control=list(dowarn=FALSE))

## Warning in Rvmminu(par = spar, fn = efn, gr = egr, control = mcontrol, ...):  Too many
gradient evaluations

a5g<-opm(2:n/n^2, nll, nll.g, method=mset, control=list(dowarn=FALSE))

## Warning in Rvmminu(par = spar, fn = efn, gr = egr, control = mcontrol, ...):  Too many
gradient evaluations

a5gb<-opm(2:n/n^2, nll, nll.g, lower=0, upper=1, method=mset, control=list(dowarn=FALSE))

## Warning in optimr(par, fn, gr, method = meth, lower = lower, upper = upper, :  optimr:
optim() with bounds ONLY uses L-BFGS-B
## Warning in optimr(par, fn, gr, method = meth, lower = lower, upper = upper, :  optimr:
optim() with bounds ONLY uses L-BFGS-B

#- 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
## spg      0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     17
## ucminf   0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     14
## Rvmmin   0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     28
## CG       0.2000000 0.2000000 0.2000000 0.2000001  8.047190e+00     59
## nlm      0.2000005 0.1999995 0.2000000 0.2000000  8.047190e+00     NA
## BFGS     0.2000007 0.1999989 0.2000011 0.1999981  8.047190e+00     33
```

```
## 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 0.003
## spg          13           0 TRUE TRUE 0.042
## ucminf       14           0 TRUE TRUE 0.002
## Rvmmin     1001           1 TRUE TRUE 0.091
## CG           21           0 TRUE TRUE 0.002
## nlm          11           0 TRUE TRUE 0.001
## BFGS          9           0 TRUE TRUE 0.001
## nlminb       12           0 TRUE TRUE 0.002
## L-BFGS-B     NA        9999   NA   NA 0.002
```

```r
summary(a5g,order=value)
```

```
##                 p1        p2        p3        p4         value fevals
## ucminf   0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     14
## 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
## Rvmmin   0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     32
## CG       0.2000000 0.2000000 0.2000000 0.2000000  8.047190e+00     59
## nlm      0.2000006 0.1999995 0.2000000 0.2000000  8.047190e+00     NA
## BFGS     0.2000007 0.1999989 0.2000012 0.1999981  8.047190e+00     33
## 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
## ucminf       14           0 TRUE TRUE 0.000
## spg          13           0 TRUE TRUE 0.041
## Rcgmin       12           0 TRUE TRUE 0.001
## Rvmmin     1001           1 TRUE TRUE 0.040
## CG           21           0 TRUE TRUE 0.001
## nlm          11           0 TRUE TRUE 0.001
## BFGS          9           0 TRUE TRUE 0.000
## nlminb       12           0 TRUE TRUE 0.001
## L-BFGS-B     NA        9999   NA   NA 0.000
```

```r
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
## BFGS            NA       NA        NA        NA 8.988466e+307     NA
## CG              NA       NA        NA        NA 8.988466e+307     NA
## ucminf          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 0.002
## spg          13           0 TRUE TRUE 0.043
## nlminb       12           0 TRUE TRUE 0.001
## L-BFGS-B     NA        9999   NA   NA 0.001
## BFGS         NA        9999   NA   NA 0.001
```

```
## CG            NA        9999   NA   NA 0.001
## ucminf        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. 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 with the $n - 1$ parameters near zero. Negative values will cause difficulties.

Numerical gradient approximations will 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.

4 Transformed problems or parameters

When problems give difficulties, it is common to re-formulate them by transformations of the function or the parameters.

4.1 Using a projection

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. Thanks to Ravi Varadhan for the suggestion.

```
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, 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 = 0.044    using numerical approx.= 0.042

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$par-pbest)),"   num. approx.=",max(abs(aspg$par-pbest)),"\n")

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

Here the projection `proj` is the key to success of method `spg`. Other methods 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.

4.2 $log()$ transformation of parameters

A common method to ensure parameters are positive is to transform them. 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? 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 `optimx` and displays the results of `all.methods`. 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
a5x<-opm(st, enll, enll.g, method="ALL", control=list(trace=0))
a5xbyvalue<-summary(a5x, order=value)
xnor<-a5xbyvalue[1, 1:5] # get the 5 parameters of "best" solution
xnor<-xnor/sum(xnor)
cat("normalized parameters:")

## normalized parameters:

print(xnor)

##       p1  p2  p3  p4  p5
## BFGS 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-12)`.

Let us try a larger problem in 100 parameters.

```
require(Rcgmin, quietly=TRUE)
st<-1:100/1e3 # large
stenll<-enll(st)
cat("Initial function value =",stenll,"\n")

## Initial function value = 460.5587

tym<-system.time(acgbig<-Rcgmin(st, enll, enll.g, control=list(trace=0, tol=1e-32)))[[3]]
cat("Time = ",tym,"  fval=",acgbig$value,"\n")

## Time =  0.081   fval= 460.517

xnor<-acgbig$par/sum(acgbig$par)
print(xnor)

##   [1] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
##  [15] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
```

```
## [29] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
## [43] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
## [57] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
## [71] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
## [85] 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
## [99] 0.01 0.01
```

One worrying aspect of the solution is that the objective function at the start and end differ by a tiny amount.

4.3 Another transformation

A slightly different transformation or projection is inspired by spherical coordinates.

```
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)

## [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] 0.009999999 0.010000001 0.010000000 0.010000000 0.009999999
##   [6] 0.010000003 0.010000000 0.010000001 0.010000000 0.010000000
##  [11] 0.010000001 0.010000000 0.010000000 0.010000002 0.010000002
##  [16] 0.010000001 0.009999999 0.010000002 0.010000000 0.010000002
##  [21] 0.010000000 0.010000000 0.010000002 0.010000001 0.010000001
##  [26] 0.010000001 0.010000000 0.010000002 0.010000001 0.010000001
##  [31] 0.010000000 0.010000001 0.010000002 0.010000002 0.010000001
##  [36] 0.010000002 0.010000001 0.010000000 0.010000001 0.010000001
##  [41] 0.009999999 0.009999996 0.010000002 0.010000002 0.010000002
##  [46] 0.009999996 0.009999997 0.009999999 0.010000000 0.009999999
##  [51] 0.009999998 0.010000002 0.010000000 0.009999999 0.010000000
##  [56] 0.009999996 0.010000002 0.010000002 0.009999998 0.010000000
##  [61] 0.010000000 0.009999999 0.010000000 0.010000000 0.010000001
##  [66] 0.009999999 0.010000000 0.010000000 0.010000000 0.010000000
##  [71] 0.009999999 0.010000000 0.009999999 0.010000000 0.009999999
```

```
##  [76] 0.010000000 0.010000000 0.009999999 0.010000000 0.009999999
##  [81] 0.009999999 0.009999999 0.010000000 0.010000001 0.009999999
##  [86] 0.009999999 0.010000000 0.010000001 0.010000000 0.009999998
##  [91] 0.009999998 0.009999999 0.010000000 0.010000000 0.010000000
##  [96] 0.009999997 0.010000038 0.009999935 0.010000003 0.010000023
```

Since this transformation is embedded into the objective function, we could run all the optimizers in `optimx` as follows. This takes some time, as the derivative-free methods appear to have more difficulty with this formulation. Moreover, `Rcgmin` and `Rvmmin` are not recommended when an analytic gradient is not provided.

```
allans<- opm(seq(n-1), obj, gr="grfwd", method="ALL", control=list(dowarn=FALSE))

## Warning in Rvmminu(par = spar, fn = efn, gr = egr, control = mcontrol, ...):  Too many
gradient evaluations
## Warning in commonArgs(par + 0, fn, control, environment()):  maxfun < 10 * length(par)^2
is not recommended.
## Warning in commonArgs(par, fn, control, environment()):  maxfun < 10 * length(par)^2
is not recommended.
## Warning in nmk(par = spar, fn = efn, control = mcontrol, ...):  Nelder-Mead should
not be used for high-dimensional optimization
## Warning in optimr(par, fn, gr, method = meth, lower = lower, upper = upper, :  Maximum
number of fevals exceeded  Restarts for stagnation =0

summary(allans, order = "list(round(value, 3), fevals)", par.select = FALSE)

##                     value fevals gevals convergence  kkt1  kkt2 xtime
## spg          4.605170e+02    230    212           0  TRUE  TRUE 0.440
## lbfgsb3      4.605170e+02    267    267           0  TRUE  TRUE 0.662
## L-BFGS-B     4.605170e+02    280    280           0  TRUE  TRUE 0.540
## ucminf       4.605170e+02    310    310           0  TRUE  TRUE 0.556
## BFGS         4.605170e+02    312    217           0  TRUE  TRUE 0.401
## Rvmmin       4.605170e+02    353   5001           1  TRUE  TRUE 9.748
## Rcgmin       4.605170e+02   1612   1023           0  TRUE  TRUE 1.800
## CG           4.605170e+02   4039   1723           0  TRUE  TRUE 3.132
## hjkb         4.605170e+02  20414     NA           0  TRUE  TRUE 0.417
## bobyqa       4.605170e+02  21493     NA           0  TRUE  TRUE 7.635
## hjn          4.605170e+02  23292     NA           0  TRUE  TRUE 0.583
## nlm          4.605170e+02     NA    203           0  TRUE  TRUE 0.661
## Rtnmin       4.756232e+02    926    926           3  TRUE FALSE 2.579
## nlminb       4.822372e+02    189    151           1 FALSE FALSE 0.302
## nmkb         7.202616e+02   5045     NA           1  TRUE FALSE 2.548
## lbfgs        7.326367e+02     NA     NA       -1001  TRUE FALSE 0.026
## Nelder-Mead  7.439745e+02   5002     NA           1  TRUE FALSE 0.115
## newuoa      8.988466e+307     NA     NA        9999    NA    NA 0.003
```

4.4 Use the gradient equations

Another approach is to "solve" the gradient equations. We can 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.

5 The Rayleigh Quotient

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`.

```
molerbuild<-function(n){ # Create the moler matrix of order n
   # A[i,j] = i for i=j, min(i,j)-2 otherwise
   A <- matrix(0, nrow = n, ncol = n)
   j <- 1:n
   for (i in 1:n) {
      A[i, 1:i] <- pmin(i, 1:i) - 2
   }
   A <- A + t(A)
   diag(A) <- 1:n
   A
}

raynum<-function(x, A){
```

```
   rayquo<-as.numeric((t(x)%*%A)%*%x)
}

proj<-function(x) { x/sqrt(crossprod(x)) }

require(BB, quietly=TRUE)
n<-10
x<-rep(1,n)
A<-molerbuild(n)
tmin<-system.time(asprqmin<-spg(x, fn=raynum, project=proj, A=A))[[3]]

## iter:  0  f-value:  205  pgrad:  3.089431e-09

## Warning in spg(x, fn = raynum, project = proj, A = A): convergence tolerance satisified
at intial parameter values.

tmax<-system.time(asprqmax<-spg(x, fn=raynum, project=proj, A=-A))[[3]]

## iter:  0  f-value:  -205  pgrad:  0.6324555

## Warning in spg(x, fn = raynum, project = proj, A = -A): Unsuccessful convergence.

cat("maximal eigensolution: Value=",asprqmax$value,"in time ",tmax,"\n")

## maximal eigensolution: Value= -205 in time  0.286

print(asprqmax$par)

##  [1] 0.3162278 0.3162278 0.3162278 0.3162278 0.3162278 0.3162278 0.3162278
##  [8] 0.3162278 0.3162278 0.3162278

cat("minimal eigensolution: Value=",asprqmin$value,"in time ",tmin,"\n")

## minimal eigensolution: Value= 205 in time  0.043

print(asprqmin$par)

##  [1] 0.3162278 0.3162278 0.3162278 0.3162278 0.3162278 0.3162278 0.3162278
##  [8] 0.3162278 0.3162278 0.3162278
```

For the record, these results compare well with eigenvalues from eigen().

If we ignore the constraint, and simply perform the optimization, we can get satisfactory solutions, though comparisons require that we normalize the parameters post-optimization. We can check if the scale of the eigenvectors is becoming large by computing the norm of the final parameter vector. In tests on the Moler matrix up to dimension 100, none grew to a worrying size.

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).

6 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)
}

cat("Try penalized sum\n")

## Try penalized sum

require(optimx)
st<-runif(3)
aos<-opm(st, ssums, gr="grcentral", method="ALL")

## Warning in Rvmminu(par = spar, fn = efn, gr = egr, control = mcontrol, ...):  Too many
gradient evaluations
## Warning in optimr(par, fn, gr, method = meth, lower = lower, upper = upper, :  Successful
convergence  Restarts for stagnation =0

# rescale the parameters
nsol<-dim(aos)[1]
for (i in 1:nsol){
  tpar<-aos[i,1:3]
  ntpar<-sum(tpar)
  tpar<-tpar/ntpar
#  cat("Method ",aos[i, "meth"]," gives fval =", ssums(tpar))
  aos[i, 1:3]<-tpar

}

summary(aos,order=value)[1:5,]

##               p1        p2        p3     value fevals gevals convergence
## newuoa 0.5454545 0.2727273 0.1818182 0.5454545     51     NA           0
## hjn    0.5454545 0.2727273 0.1818182 0.5454545    228     NA           0
## Rvmmin 0.5454545 0.2727273 0.1818182 0.5454545     10   1001           1
## nlminb 0.5454545 0.2727273 0.1818182 0.5454545      9      9           0
## bobyqa 0.5454546 0.2727272 0.1818182 0.5454545     55     NA           0
##        kkt1  kkt2 xtime
```

```
## newuoa TRUE FALSE 0.002
## hjn    TRUE FALSE 0.004
## Rvmmin TRUE FALSE 0.092
## nlminb TRUE FALSE 0.000
## bobyqa TRUE FALSE 0.001
```

```
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:  3.246022  pgrad:  0.7459304
## iter:  10  f-value:  0.5454545  pgrad:  8.601083e-05

cat("Using project.simplex with spg: fmin=",as$value," at \n")

## Using project.simplex with spg: fmin= 0.5454545  at

print(as$par)

## [1] 0.5454524 0.2727289 0.1818186
```

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.

```
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
```

7 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 and rescaling the parameters before reporting them. Another choice is to use the projected gradient via `spg` from package `BB`.

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 `http://macnash.telfer.uottawa.ca/~nashjc/RQtimes.pdf` and `http://macnash.telfer.uottawa.ca/~nashjc/RQtimes.Rnw`.

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.