Three variable data and hierarchical model

Hugo Maruri-Aguilar

08/04/2021

This document has a small example to illustrate the lasso methodology and the `R` code I previously prepared. I run first standard lasso from library `lars` and then with in-house programs replicate the analysis. Then I run the hierarchical versions (H, W, S) of lasso analysis. An Appendix describes the main functions used.

Data preparation

This is the initial synthetic data set, stored in variable `DAT`. The column headers indicate the exponent term, i.e. `010` is x_2. Note that data columns are already centered around the mean. This centering of columns and response will remove the need to consider intercept in the rest of the analysis.

```
DAT
```

```
##      100 010 001  y
## [1,]   0  -1  -1 -2
## [2,]  -1   0   0  0
## [3,]  -1  -1  -1  1
## [4,]  -1   0   1  1
## [5,]  -3  -1   1 -1
## [6,]  -1   0   1 -1
## [7,]   7   3  -1  2
```

Now add columns with interactions x_1x_2 and x_1x_3, which are also centered around their means. The matrix `DATX` has the regressors, while `DATY` has the response values. Column names of `DATX` are updated.

```
cbind(DATX,DATY)
```

```
##      100 010 001        110        101  y
## [1,]   0  -1  -1 -3.5714286  1.5714286 -2
## [2,]  -1   0   0 -3.5714286  1.5714286  0
## [3,]  -1  -1  -1 -2.5714286  2.5714286  1
## [4,]  -1   0   1 -3.5714286  0.5714286  1
## [5,]  -3  -1   1 -0.5714286 -1.4285714 -1
## [6,]  -1   0   1 -3.5714286  0.5714286 -1
## [7,]   7   3  -1 17.4285714 -5.4285714  2
```

This is the model, to be read as exponents of term in each row. Note absence of intercept.

```
LD
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
## [4,]    1    1    0
## [5,]    1    0    1
```

Initial LASSO analysis

Standard lasso analysis with function `lars` from the same library.

```
library(lars)
A<-lars(x=DATX,y=DATY,type="lasso",normalize=FALSE,intercept=FALSE)
```

The rows in the table below correspond to breakpoints determined by values of λ at which there is a change in the piecewise linear trajectories of coefficients. The first column in the table are these breakpoint values of λ; then the column entries are lasso estimates $\beta^L(\lambda)$, that is coefficients of model terms `100, 010, 001, 110, 101`; the last column is the criterion $L = ||Y - X\beta||_2^2 + \lambda||\beta||_1$. I did some post processing of code outputs `A$beta` and `A$lambda` before showing the table.

```
##     lambda     100     010    001    110    101      L
## 7  0.0000 -0.0937 -0.0938 2.0000 0.9063 2.0000 1.0000
## 6  0.0117 -0.1307  0.0000 1.8927 0.8816 1.9448 1.0583
## 5  0.4286  0.0000  0.0000 1.1654 0.5599 1.2168 2.6822
## 4  0.4407  0.0000  0.0000 1.1438 0.5527 1.1981 2.7176
## 3  0.9327  0.0000  0.3051 0.0000 0.1348 0.2394 3.5969
## 2  1.7733  0.0000  0.0000 0.0000 0.1274 0.0680 3.9646
## 1  2.1185  0.0000  0.0000 0.0000 0.1047 0.0000 4.0164
## 0 40.0000  0.0000  0.0000 0.0000 0.0000 0.0000 6.0000
```

In standard Lasso like above, the maximum value of λ for the analysis is determined by $\max_i |X^T Y|_i$. This is the value at which all the coefficients shrink to zero. For these data, this is the maximum of {16, 8, 2, 40, 10}. The breakpoints of λ above are determined automatically by `lars` and note that the analysis does not consider the hierarchical structure of the model terms.

Lasso with `quadprog`, single orthant

In preparation for later analyses, here is the least squares fit $\hat{\beta}$, corresponding to the coefficients in the first row in the table above. We also compute the vector of signs of $\hat{\beta}$, to be used in quadrant optimization.

```
bols<-solve(t(DATX)%*%DATX)%*%t(DATX)%*%DATY; t(bols) ## LSE estimate
```

```
##         100      010 001     110 101
## y -0.09375 -0.09375   2 0.90625   2
```

```
sign(bols)->c0; t(c0) ## the quadrant of the LSE estimate
```

```
##   100 010 001 110 101
## y  -1  -1   1   1   1
```

The lasso using the quadrant function. The output of the function is the lasso estimate $\hat{\beta}^L(\lambda)$ and the achieved value L. Here we do two computations of the table above for $\lambda = 0$ and for $\lambda = 0.0117273$.

```
minconoineq(CM=diag(c(c0)),XM=DATX,YM=DATY,lamm=0)
```

```
## [1] -0.09375 -0.09375  2.00000  0.90625  2.00000  1.00000
```

```
minconoineq(CM=diag(c(c0)),XM=DATX,YM=DATY,lamm=A$lambda[length(A$lambda)])
```

```
## [1] -0.1307415  0.0000000  1.8927132  0.8816272  1.9447532  1.0583059
```

Lasso with `quadprog`, list of λ and neighboring orthants

Here we use the values of λ={0, 0.0117, 0.4286, 0.4407, 0.9327, 1.7733, 2.1185, 40} from the `lars` previous analysis. From the initial orthant (-1, -1, 1, 1, 1), the function below computes neighbors by switching one orthant at a time:

```
vecinos(c(c0))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   -1   -1    1    1    1
## [2,]    1   -1    1    1    1
## [3,]   -1    1    1    1    1
## [4,]   -1   -1   -1    1    1
## [5,]   -1   -1    1   -1    1
## [6,]   -1   -1    1    1   -1
```

In below, there are $|LM|+1$ optimizations for every value of λ. This is the size of the entry given in `GG`. Note that the absence of the parameter `AA` means the optimization is performed without constraints apart from the orthants.

```
round(lassocono(XM=DATX, YM=DATY, GG=vecinos(c(c0)),
                 lambdaval=sort(c(A$lambda,0))  )[,-c(2:6)],5)
```

```
##          [,1]     [,2]     [,3]    [,4]    [,5]    [,6]    [,7]
## [1,]  0.00000 -0.09375 -0.09375 2.00000 0.90625 2.00000 1.00000
## [2,]  0.01173 -0.13074  0.00000 1.89271 0.88163 1.94475 1.05831
## [3,]  0.42855  0.00000  0.00000 1.16542 0.55989 1.21684 2.68225
## [4,]  0.44067  0.00000  0.00000 1.14380 0.55271 1.19806 2.71762
## [5,]  0.93270  0.00000  0.30509 0.00000 0.13484 0.23944 3.59686
## [6,]  1.77331  0.00000  0.00000 0.00000 0.12745 0.06804 3.96456
## [7,]  2.11849  0.00000  0.00000 0.00000 0.10473 0.00000 4.01638
## [8,] 40.00000  0.00000  0.00000 0.00000 0.00000 0.00000 6.00000
```

Lasso with `quadprog`, list of λ and all orthants

Using the same function, the search below uses all orthants. This is achieved by not specifying orthants `GG` and thus for every value of λ there are $2^{|LM|}$ optimizations. As above, the parameter `AA` was not used.

```
round(lassocono(XM=DATX, YM=DATY,
                 lambdaval=sort(c(A$lambda,0))  )[,-c(2:6)],5)
```

```
##          [,1]     [,2]     [,3]    [,4]    [,5]    [,6]    [,7]
## [1,]  0.00000 -0.09375 -0.09375 2.00000 0.90625 2.00000 1.00000
## [2,]  0.01173 -0.13074  0.00000 1.89271 0.88163 1.94475 1.05831
## [3,]  0.42855  0.00000  0.00000 1.16542 0.55989 1.21684 2.68225
## [4,]  0.44067  0.00000  0.00000 1.14380 0.55271 1.19806 2.71762
## [5,]  0.93270  0.00000  0.30509 0.00000 0.13484 0.23944 3.59686
## [6,]  1.77331  0.00000  0.00000 0.00000 0.12745 0.06804 3.96456
## [7,]  2.11849  0.00000  0.00000 0.00000 0.10473 0.00000 4.01638
## [8,] 40.00000  0.00000  0.00000 0.00000 0.00000 0.00000 6.00000
```

Note that without specifying constraints matrix `AA`, both approaches give exactly the same results.

Timings for both calls

This initial example is by no means an expensive computation. Nevertheless, as there are 8 values of λ, the first call (neighbor) does 48 optimizations which is compared against 256 optimizations of the second call (full orthant). Thus for this example, the full orthant search is 5.333 times more expensive than neighbor orthant search.

```
library(tictoc)
tic("Neighboring orthants")
B0<-lassocono(XM=DATX, YM=DATY, GG=vecinos(c(c0)), lambdaval=sort(c(A$lambda,0)))
toc()
```

```
## Neighboring orthants: 0.02 sec elapsed
```

```
tic("All orthants")
B0<-lassocono(XM=DATX, YM=DATY, lambdaval=sort(c(A$lambda,0)))
toc()
```

```
## All orthants: 0.01 sec elapsed
```

Hierarchical lasso (H)

The Hasse diagram

The function `hassediagram` retrieves the dependences between terms. Each row corresponds to model term; the columns are indexed by variables and. In a given row, a nonzero entry indexes which term is divisible by the corresponding row, and the position in the column is the remainder. In other words, every non zero value corresponds to an edge in the diagram, and the edge is drawn between the term indexed by the entry and the term inding the row.

```
HD<-hassediagram(LM=LD); row.names(HD)<-nombres(LM=LD); HD
```

```
##     [,1] [,2] [,3]
## 100    0    4    5
## 010    4    0    0
## 001    5    0    0
## 110    0    0    0
## 101    0    0    0
```

For the model of this example, the Hasse diagram has 5 nodes (number of rows) and 4 edges (number of non-zero entries). Of the nodes, 1 node (100) has two descendant terms ; other 2 nodes (010, 001) have only one descendant each and 2 nodes (110, 101) have no descendant nodes.

Matrix of constraints for hierarchy H

We now build the analysis using the H set of constraints, stemming directly from the edges of the Hasse diagram. The function `restrict` gives the S type of hierarchy using option `type=1`; the type W using option `type=2` and the type H using option `type=3`; it calls internally `hassediagram` so no need to compute the diagram beforehand.

```
AA<-restrict(LM=LD,type=3,weight=FALSE); colnames(AA)<-nombres(LM = LD); AA
```

```
##      100 010 001 110 101
## [1,]   1   0   0  -1   0
## [2,]   1   0   0   0  -1
## [3,]   0   1   0  -1   0
## [4,]   0   0   1   0  -1
```

Constrained least squares orthant

The first approach uses only the orthant from least squares (-1, -1, 1, 1, 1).

```
bcc<-minconoineq(CM=diag(c(c0)),XM = DATX, YM = DATY,AA = AA,lamm = 0)
Ball<-matrix(nrow=1,c(0,bcc)); colnames(Ball)<-nombracoeffsmatrix(LM = LD)
round(Ball,5);
```

```
##      lambda      100      010 001     110 101       L
## [1,]      0 -0.17073 -0.17073   0 0.17073   0 4.63415
```

Constrained least squares neighbors of ols orthant

We use the less expensive search over neighboring orthants of the ordinary least squares (-1, -1, 1, 1, 1).

```
matrix(nrow=1,lassocono(XM=DATX, YM=DATY,AA=AA,GG=vecinos(c(c0)),lambdaval=0  )[,-c(2:6)])->BN
colnames(BN)<-nombracoeffsmatrix(LM = LD); round(BN,4);
```

```
##      lambda     100    010    001    110    101     L
## [1,]      0 -0.3671 0.8804 0.3671 0.2427 0.3671 2.763
```

Constrained least squares all orthants

Here we perform the expensive search over all the space composed of 2^5 orthants.

```
matrix(nrow=1,lassocono(XM=DATX, YM=DATY, AA=AA, lambdaval=0  )[,-c(2:6)])->BA
colnames(BA)<-nombracoeffsmatrix(LM = LD); round(BA,4);
```

```
##      lambda     100    010     001    110    101      L
## [1,]      0 -0.9951 2.0576 -0.6316 0.3207 0.6316 1.8421
```

From the earlier computations, this example is an instance where the initial least squares' orthant or neighboring orthants are not neccesarily a good starting point as it is evident when compared against the full search.

Constrained lasso

We now build lasso paths for the cases of neighbor orthants and full orthant.

Searching neighbors of least squares orthant

We check the last result using the less expensive search over neighboring orthants of the ordinary least squares orthant (-1, -1, 1, 1, 1).

```
round(lassocono(XM=DATX, YM=DATY, AA=AA, GG=vecinos(c(c0)),
                nlambda=7,lmax=16   )[,-c(2:6)],5)->LNE
colnames(LNE)<-nombracoeffsmatrix(LM = LD);  LNE
```

```
##        lambda      100      010    001     110    101       L
## [1,]  0.00000 -0.36710  0.88045 0.3671 0.24274 0.3671 2.76295
## [2,]  2.66667  0.12774 -0.04833 0.0000 0.04833 0.0000 4.50394
## [3,]  5.33333  0.10219 -0.03867 0.0000 0.03867 0.0000 5.04252
## [4,]  8.00000  0.07664 -0.02900 0.0000 0.02900 0.0000 5.46142
## [5,] 10.66667  0.05110 -0.01933 0.0000 0.01933 0.0000 5.76063
## [6,] 13.33333  0.02555 -0.00967 0.0000 0.00967 0.0000 5.94016
## [7,] 16.00000  0.00000  0.00000 0.0000 0.00000 0.0000 6.00000
```

Searching all orthants

This is the same call of `lassocono` as above except that when the argument `GG` is not given, the search is over the space composed of 2^5 orthants.

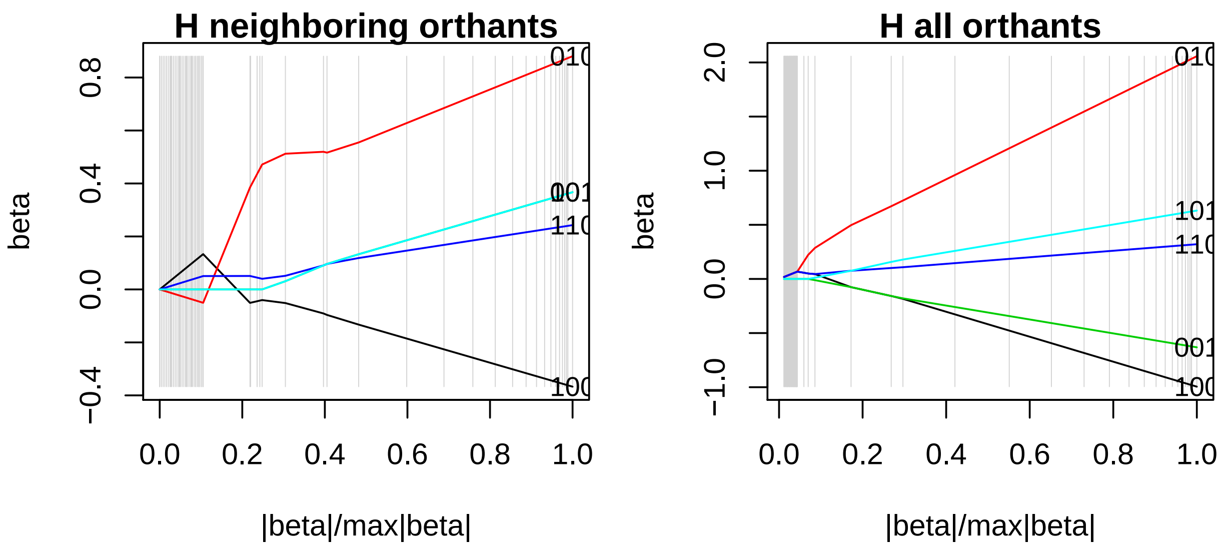
```
round(lassocono(XM=DATX, YM=DATY, AA=AA,
                 nlambda=7, lmax=22  )[,-c(2:6)],5)->LAL
colnames(LAL)<-nombracoeffsmatrix(LM = LD);  LAL
```

```
##        lambda      100     010      001     110     101       L
## [1,]  0.00000 -0.99507 2.05757 -0.63158 0.32072 0.63158 1.84211
## [2,]  3.66667  0.06038 0.06038  0.00000 0.06038 0.00000 4.39982
## [3,]  7.33333  0.04785 0.04785  0.00000 0.04785 0.00000 4.99512
## [4,] 11.00000  0.03532 0.03532  0.00000 0.03532 0.00000 5.45256
## [5,] 14.66667  0.02279 0.02279  0.00000 0.02279 0.00000 5.77214
## [6,] 18.33333  0.01025 0.01025  0.00000 0.01025 0.00000 5.95386
## [7,] 22.00000  0.00000 0.00000  0.00000 0.00000 0.00000 6.00000
```

Plotting the results

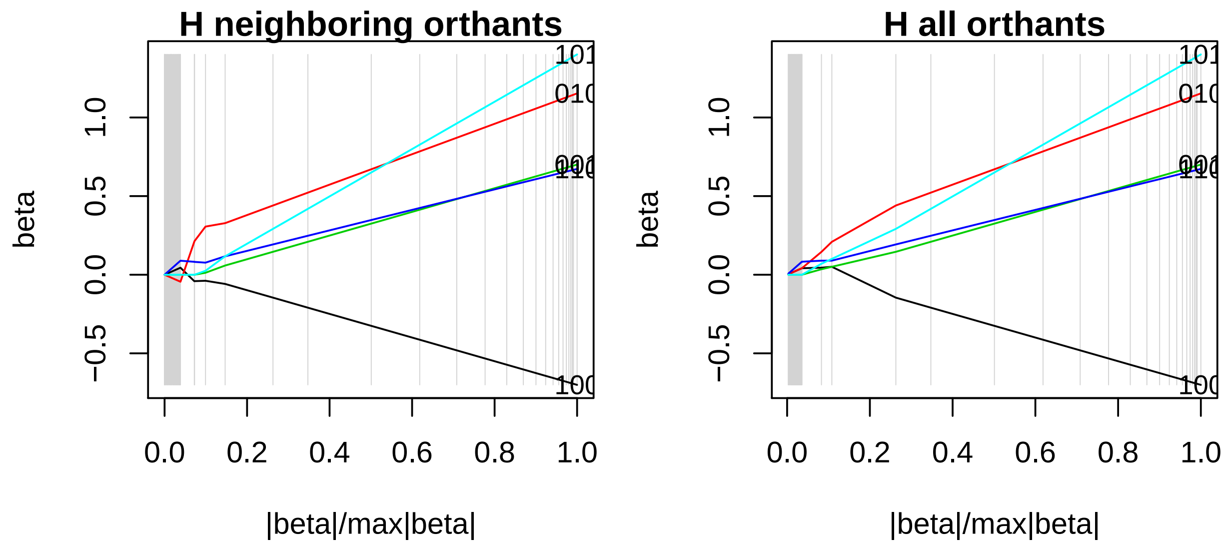
The custom built function `graficalo` is used to plot the paths for the two cases, only neighboring orthants and all orthants. To improve the path plotted, we use more values of λ than earlier. These are computed with function `seqq`.

```
par(mar=c(4,4,1,1),mfrow=c(1,2))
graficalo(Coeffs=lassocono(XM=DATX, YM=DATY, AA=AA, GG=vecinos(c(c0)),
                  lambdaval=seqq(maximo=16,cuantos=30)   )[,-c(2:6)], LM=LD,
                  titulo="H neighboring orthants")
graficalo(Coeffs=lassocono(XM=DATX, YM=DATY, AA=AA,
                lambdaval=seqq(maximo=16,cuantos=30)   )[,-c(2:6)], LM=LD,
                 titulo="H all orthants")
```

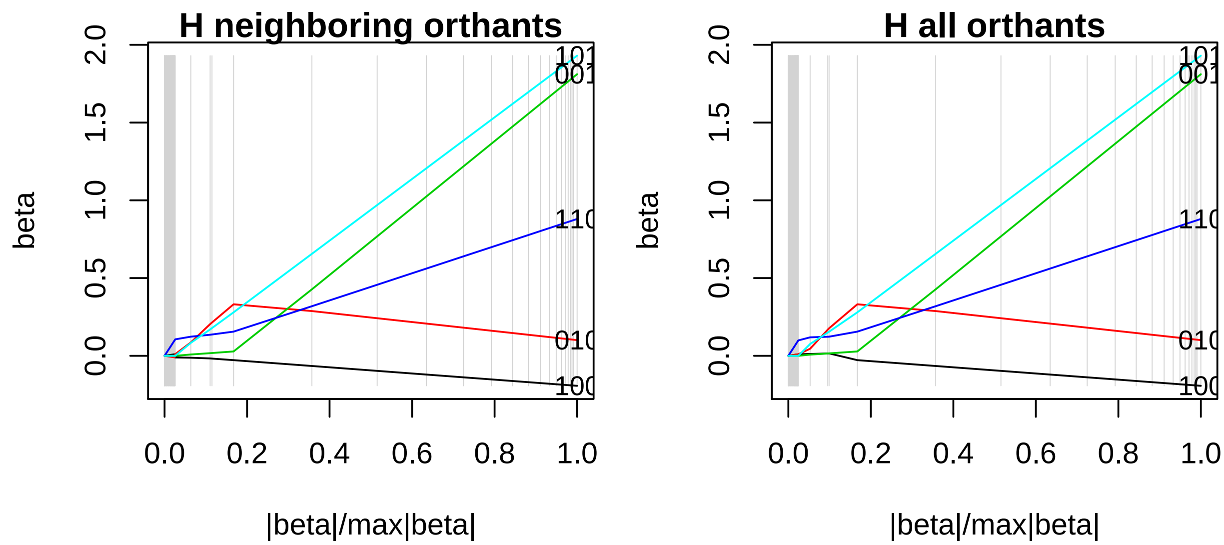


We can rescale the restriction matrix by multiplying its positive entries by a factor. Using a factor bigger than one enlarges the search region and we would expect less differences between searching in neighboring orthants against searching over all orthants. This is done below for the factor equal to two and to ten.

```
AA2<-rescale(MAT=AA,factor=2); AA10<-rescale(MAT=AA,factor=10);
par(mar=c(4,4,1,1),mfrow=c(1,2))
graficalo(Coeffs=lassocono(XM=DATX, YM=DATY,  AA=AA2, GG=vecinos(c(c0)),
          lambdaval=seqq(maximo=24,cuantos=30)   )[,-c(2:6)], LM=LD,
          titulo="H neighboring orthants")
graficalo(Coeffs=lassocono(XM=DATX, YM=DATY,   AA=AA2,
           lambdaval=seqq(maximo=24,cuantos=30)   )[,-c(2:6)], LM=LD,
           titulo="H all orthants")
```




```r
par(mar=c(4,4,1,1),mfrow=c(1,2))
graficalo(Coeffs=lassocono(XM=DATX, YM=DATY, AA=AA10, GG=vecinos(c(c0)),
          lambdaval=seqq(maximo=36,cuantos=30)   )[,-c(2:6)], LM=LD,
          titulo="H neighboring orthants")
graficalo(Coeffs=lassocono(XM=DATX, YM=DATY, AA=AA10,
          lambdaval=seqq(maximo=36,cuantos=30)   )[,-c(2:6)], LM=LD,
           titulo="H all orthants")
```



In this latter case with factor equal to ten the differences between trajectories in the two cases become negligible.

Below we exhibit the constraint matrix for this latter case with factor equal to ten,

```r
AA10
```

```
##      100 010 001 110 101
## [1,]  10   0   0  -1   0
## [2,]  10   0   0   0  -1
## [3,]   0  10   0  -1   0
## [4,]   0   0  10   0  -1
```

Here are the Lasso paths for this last scaling factor of ten and the two cases (neighboring orthants and all orthants). The paths are largely similar although they are not equal.

```r
lassocono(XM=DATX, YM=DATY, AA=AA10, GG=vecinos(c(c0)),
                lambdaval=seqq(maximo=36,cuantos=4)   )[,-c(2:6)]->BBN
colnames(BBN)<-nombracoeffsmatrix(LM = LD);
lassocono(XM=DATX, YM=DATY, AA=AA10,
                lambdaval=seqq(maximo=36,cuantos=4)   )[,-c(2:6)]->BBA
colnames(BBA)<-nombracoeffsmatrix(LM = LD);
round(BBN,5) ## Neighboring orthants
```

```
##        lambda      100      010     001     110     101       L
## [1,]  0.00000 -0.19303  0.10084 1.81102 0.87990 1.93034 1.00557
## [2,]  0.01000 -0.19119  0.10355 1.79100 0.87178 1.91189 1.05450
## [3,]  0.15325 -0.16477  0.14229 1.50417 0.75546 1.64766 1.70511
## [4,]  2.34849 -0.01050  0.01050 0.00000 0.10504 0.00000 4.08919
## [5,] 12.00000  0.00699 -0.00699 0.00000 0.06992 0.00000 5.07701
## [6,] 24.00000  0.00318 -0.00318 0.00000 0.03178 0.00000 5.80930
## [7,] 35.99000  0.00000  0.00000 0.00000 0.00000 0.00000 6.00000
## [8,] 36.00000  0.00000  0.00000 0.00000 0.00000 0.00000 6.00000
```

```r
round(BBA,5) ## ALL orthants
```

```
##        lambda      100     010     001     110     101       L
## [1,]  0.00000 -0.19303 0.10084 1.81102 0.87990 1.93034 1.00557
## [2,]  0.01000 -0.19119 0.10355 1.79100 0.87178 1.91189 1.05450
## [3,]  0.15325 -0.16477 0.14229 1.50417 0.75546 1.64766 1.70511
## [4,]  2.34849  0.00984 0.00984 0.00000 0.09842 0.00000 4.05209
## [5,] 12.00000  0.00696 0.00696 0.00000 0.06963 0.00000 5.02525
## [6,] 24.00000  0.00338 0.00338 0.00000 0.03382 0.00000 5.77004
## [7,] 35.99000  0.00000 0.00000 0.00000 0.00000 0.00000 6.00000
## [8,] 36.00000  0.00000 0.00000 0.00000 0.00000 0.00000 6.00000
```

Weak hierarchical Lasso (W)

Here is the constraint matrix for the same model and weak hierarchy. This matrix is obtained with the function `restrict` and setting `type=2`. In this case, the flag `weight=TRUE` gives matrix with non zero entries being ± 1. We also give the scaled version of it.

```
restrict(LM=LD,type=2,weight=TRUE)->AA; colnames(AA)<-nombres(LM=LD); AA
```

```
##      100 010 001 110 101
## [1,]   1   1   0  -1   0
## [2,]   1   0   1   0  -1
```

```
rescale(MAT=restrict(LM=LD,type=2,weight=TRUE),factor=10)->AA10; colnames(AA10)<-nombres(LM=LD); AA10
```

```
##      100 010 001 110 101
## [1,]  10  10   0  -1   0
## [2,]  10   0  10   0  -1
```

The following two paths are for neighboring orthants and for computation using all orthants. The analyses coincide.

```
lassocono(XM=DATX, YM=DATY, GG=vecinos(c(c0)), AA=AA,
                lambdaval=seqq(maximo=38,cuantos=4)   )[,-c(2:6)]->BBN
lassocono(XM=DATX, YM=DATY,  AA=AA,
                lambdaval=seqq(maximo=38,cuantos=4)   )[,-c(2:6)]->BBA
colnames(BBN)<-nombracoeffsmatrix(LM = LD); colnames(BBA)<-nombracoeffsmatrix(LM = LD);
round(BBN,5) ## neighbor orthant
```

```
##        lambda      100     010     001     110     101       L
## [1,]  0.00000 -0.34230 0.46596 1.45816 0.80825 1.77452 1.04024
## [2,]  0.01000 -0.33790 0.46314 1.44348 0.80104 1.75823 1.08850
## [3,]  0.15604 -0.27693 0.42144 1.24511 0.69838 1.52205 1.74235
## [4,]  2.43470  0.00000 0.08771 0.00000 0.08771 0.00000 4.10841
## [5,] 12.66667  0.04395 0.00000 0.00000 0.04395 0.00000 5.32605
## [6,] 25.33333  0.00764 0.00000 0.00000 0.00764 0.00000 5.97962
## [7,] 37.99000  0.00000 0.00000 0.00000 0.00000 0.00000 6.00000
## [8,] 38.00000  0.00000 0.00000 0.00000 0.00000 0.00000 6.00000
```

```
round(BBA,5) ## all orthants
```

```
##        lambda      100     010     001     110     101       L
## [1,]  0.00000 -0.34230 0.46596 1.45816 0.80825 1.77452 1.04024
## [2,]  0.01000 -0.33790 0.46314 1.44348 0.80104 1.75823 1.08850
## [3,]  0.15604 -0.27693 0.42144 1.24511 0.69838 1.52205 1.74235
## [4,]  2.43470  0.00251 0.08474 0.00000 0.08725 0.00000 4.10838
## [5,] 12.66667  0.04395 0.00000 0.00000 0.04395 0.00000 5.32605
## [6,] 25.33333  0.00764 0.00000 0.00000 0.00764 0.00000 5.97962
## [7,] 37.99000  0.00000 0.00000 0.00000 0.00000 0.00000 6.00000
## [8,] 38.00000  0.00000 0.00000 0.00000 0.00000 0.00000 6.00000
```

Strong hierarchical Lasso (S)

This is achieved with the constraint matrix built with `restrict` and `type=1`. The flag `weight=TRUE` is used as in the weak hierarchy case.

```
restrict(LM=LD,type=1,weight=TRUE)->AA; colnames(AA)<-nombres(LM=LD); AA
```

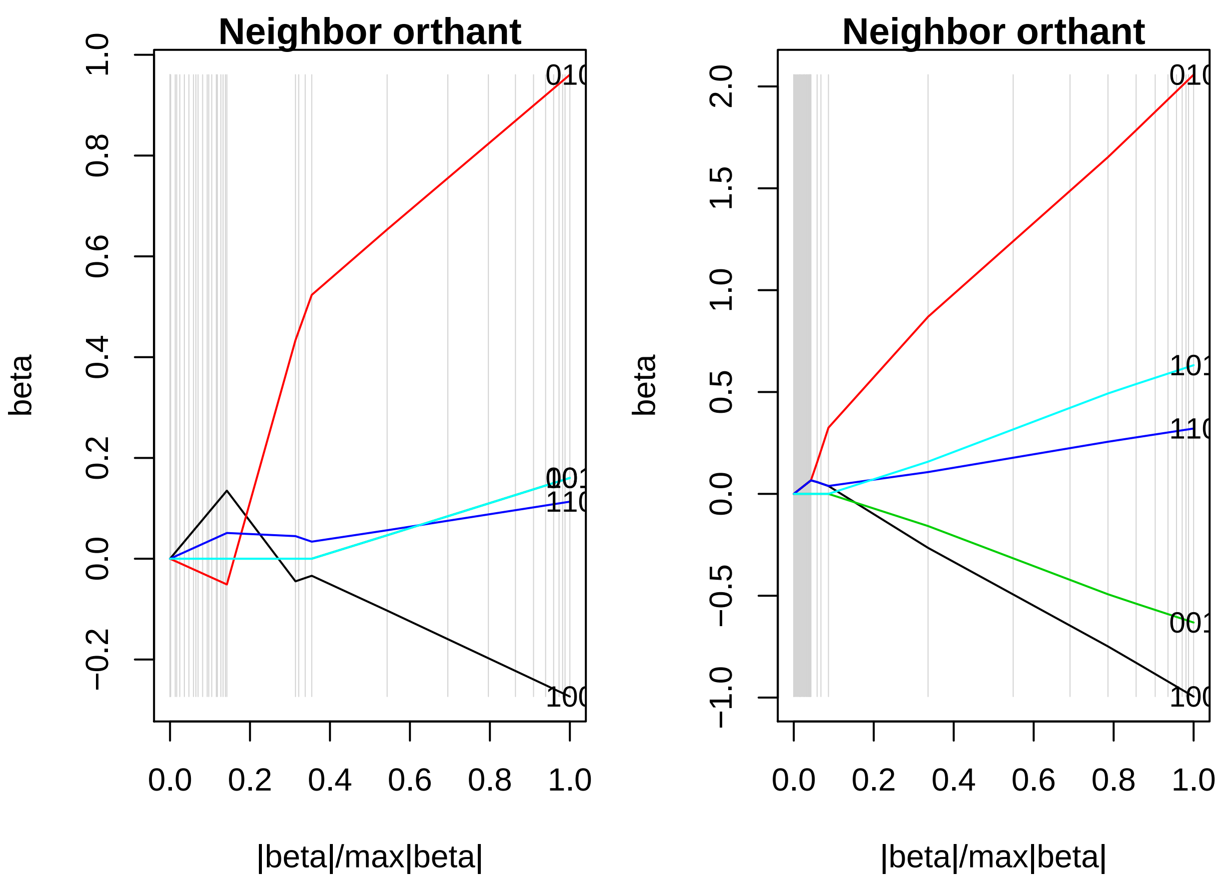
```
##      100 010 001 110 101
## [1,]   1   0   0  -1  -1
## [2,]   0   1   0  -1   0
## [3,]   0   0   1   0  -1
```

```
rescale(MAT=restrict(LM=LD,type=1,weight=TRUE),factor=100)->AA10; colnames(AA10)<-nombres(LM=LD); AA10
```

```
##      100 010 001 110 101
## [1,] 100   0   0  -1  -1
## [2,]   0 100   0  -1   0
## [3,]   0   0 100   0  -1
```

Below are two paths built using the matrix `AA` and the two searches over "neighbor orthants" and over "all orthants". It is clear in the plots the suboptimality of neighbor search.

```
lassocono(XM=DATX, YM=DATY, GG=vecinos(c(c0)), AA=AA,
                lambdaval=seqq(maximo=21.5,cuantos=20)   )[,-c(2:6)]->BBN
lassocono(XM=DATX, YM=DATY,  AA=AA,
                lambdaval=seqq(maximo=21.5,cuantos=20)   )[,-c(2:6)]->BBA
par(mar=c(4,4,1,1),mfrow=c(1,2))
graficalo(Coeffs=BBN,titulo="Neighbor orthant",LM=LD)
graficalo(Coeffs=BBA,titulo="Neighbor orthant",LM=LD)
```



Appendix

Data format

Assume that there are n observations in which a response variable was collected and there are measurements of d covariates available for every value of the response variable. The data for this analysis consists of three objects.

The **response** vector Y is a column vector of n rows; there is a **design-model matrix** X of n rows and with p columns. The columns of X are computed by evaluating polynomial terms at each of covariate values. These p columns of X correspond to the terms in the polynomial **model** LM. This model is allocated in a matrix of p rows and with d columns, where d is the number of covariates, and the entries of this matrix are integer numbers which are the exponents that create the polynomial model terms.

Main functions from the file `codigofuente.r`

The file has many different utilities, some for creating and handling polynomial models, other functions are for handling the model terms, other for plotting analyses. Here I describe the main functions used for analysis.

`minconoineq`

This is the main function, based upon the function `solve.QP` from library `quadprog`. For a given model data matrix `XM` with response vector `YM`, and choice of regularization parameter `lamm` (i.e. λ), the function `minconoineq` performs a quadratic minimization over the orthant given by diagonal of matrix `CM`. The objective function is that orthant part of standard lasso criterion

$$L = ||Y - X\beta||_2^2 + \lambda||\beta||_1,$$

subject to constraints in β imposed by the constraint matrix `AA` and to the orthant of `CM`. The entries of `CM` that identify orthant may be not just ± 1 but could be zero as well, hence the function has steps that prevent the matrices involved having zero rows/columns.

The function retrieves the coefficient $\hat{\beta}^L(\lambda)$ over the specified orthant, together with the minimal objective value achieved.

`minconoineqquadr`

This function is a wrapper that runs `minconoineq` over several quadrants specified by the matrix `GG`. The other entries are `XM`, `YM`, `lamm` and `AA`. If `GG` is not specified, this function performs a computation over all possible orthants which are $2^{|LM|}$ (recall that columns of `XM` correspond to terms in LM).

If the flag `minimize` is set to `TRUE`, then the function retrieves the best result of `minconoineq` over the specified set of orthants `GG`. By default, `minimize=FALSE` and the function retrieves all the results, one for each orthant.

`lassocono`

This is a wrapper that runs `minconoineqquadr` (and consequently `minconoineq`) over a choice of values of λ specified by one of two ways. The user can give a list of specific values in `lambdaval`; alternatively the user specifies `lmin`, `lmax` and `nlambda` to construct a uniformly spaced sequence of values to use. Other entries `XM`, `YM`, `AA` and `GG` are as per inputs of `minconoineqquadr`.

The output is the collection of coefficients of the model that minimizes over choice of orthants `GG`. The output is slightly verbose as for every λ, it includes the value of λ, which orthant was the minimizer, the estimate $\hat{\beta}^L(\lambda)$ and the minimal objective value achieved.

`hassediagram`

This function creates the simplified Hasse diagram. Its input is `LM`, the matrix in which each row is a term and entries are exponents of the model.

Each row of the output corresponds to a term (row) in `LM`. In each row, nonzero numbers are pointers (indexes) which other model terms depend on the current row, via simple product of unit exponents and thus the number of columns equals the number of variables (in turn, number of columns in `LM`).

`restrict`

This function generates the matrix of restrictions to be used as input `AA` in the above functions. The following are inputs for this function, `LM` which is the specification of polynomial terms in the model, and `type` that determines the hierarchy to be used. If `type=1` strong hierarchy (S) adding over descendants is done, if `type=2` then return weak hierarchy (W) adding over parents and otherwise `type=3` return general strong hierarchy (H). The other input is the flag `weight`. By default `weight=TRUE` and all non-zero entries are ± 1. This flag does not matter for `type=3`. This function calls automatically `hassediagram` so no need to create a Hasse diagram separately.

The output is a matrix with the weights for the inequalities. It has as many columns as the terms (rows) in `LM` and as many rows as the type of hierarchy. If (S, `type=1`) the number of rows is the number of nodes with are parents; for (W, `type=2`) it is the number of nodes which are descendants, and for (H, `type=3`) it is the number of edges in the Hasse diagram.

`rescale`

This function is to scale the positive entries of matrix `MAT`. Depending on the value of input `factor`, it is possible to make the search region less constrained for `factor` bigger than one and more constrained for such value less than one. Usually we want to make the analysis less constrained. A logical flag `override` controls whether this output will be forced to be exactly equal to `factor`, otherwise it is the product of the maximum positive entry by `factor`.

The output is the scaled matrix `MAT` that will be used as matrix of restrictions `AA`.

`vecinos`

This function takes as input `CC` a vector of entries taking values of $\{0, \pm 1\}$.

The output is a matrix output with as many columns as entries in `CC` and the rows are sign changes of `CC` one at a time, including the original `CC`. This is to be used in a simple, low cost orthant search. This is the method refered to as the 'neighbor orthant' method.

`graficalo`

This is a call to plot the lasso path, stemming from the output of `lassocono`. Its main input is `Coeffs` which by default (controlled by flag `extras`) is assumed to have λ in the first column then columns for the coefficients $\hat{\beta}^L(\lambda)$ and in the final column the criterion L. If the table for `Coeffs` only has the coefficients $\hat{\beta}^L(\lambda)$ available, the flag `extras` should be set to `FALSE`.

By default the path are plotted against percentage of shrinkage $s = s(\lambda)$, but they could be plotted against λ by setting the flag `plotlambda=TRUE`. The flag `simplito` by default allows for very simple axis labels but could create more complex labels if desired by switching to `simplito=TRUE`. If model `LM` is given, the plot adds labels to each path.