Style analysis

Enrico Schumann

13 September 2013

Style analysis

Style analysis, described in [cite:sharpe1992], uses a linear factor model to describe the returns of a portfolio, with a few constraints added: the factors should be actual asset classes; the factor loadings, which are interpreted as weights, should sum to 100% and should take on reasonable values. For a typical mutual fund, for instance, the weights should be nonnegative.

Data

We generate random `X` and `y`. `X` is a matrix that would hold the observations on indices (rows are observations; columns are indices). All information is collected in a list `Data`.

```
require("NMOF")
Data <- list(ni = 30,     ## number of indices
             no = 100,    ## number of observations
             maxi = 3,    ## maximum number of included indices
             sum2one = FALSE)   ## scale weights

Data$X <- array(rnorm(Data$no * Data$ni),
                dim = c(Data$no, Data$ni))
Data$y <- rnorm(Data$no)
str(Data)
```

```
Loading required package: NMOF
List of 6
 $ ni     : num 30
 $ no     : num 100
 $ maxi   : num 3
 $ sum2one: logi FALSE
 $ X      : num [1:100, 1:30] 1.139 1.936 -0.185 -0.369 0.14 ...
 $ y      : num [1:100] -1.29 -0.619 1.353 1.061 -0.431 ...
```

Least Squares

We will use Differential Evolution to solve the model. As a test, we compute the Least Squares solution to $y = Xw + \epsilon$. To minimise the variance of the residuals, we just need to compute the variance of the columns. Or, perhaps simpler, we just add a column of ones to `X` that is always included as a regressor. In this way, the residuals will have a zero mean and Least Squares is equivalent to minimising variance.

```
ofun <- function(p, Data) {
    tmp <- Data$X %*% p - Data$y
    ##colSums(tmp * tmp)  ## for Least Squares
    apply(tmp, 2, var)
}

settings <- list(min = rep(0, Data$ni),
                 max = rep(1, Data$ni),
                 nG = 2000,  ## number of generations
                 loopOF = FALSE, printBar = FALSE)
sol <- DEopt(ofun, algo = settings, Data = Data)
```

```
Differential Evolution.
Best solution has objective function value 0.6052827 ;
standard deviation of OF in final population is 7.092955e-17 .
```

To see whether we succeeded we also compute the solution with `lm`. The differences between the results are negligible (and could be made as small as machine precision allows us to by increasing `settings$nG`).

```
## for Least Squares, fit 'Data$y ~ -1 + Data$X' instead
summary(abs(coef(lm(Data$y ~ Data$X))[-1] - sol$xbest))
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
5.056e-11 8.426e-10 1.335e-09 1.467e-09 2.033e-09 3.300e-09
```

Constraints

We constrain the model as follows: all weights must be nonnegative; the sum of the weights must equal one; and there must be no more than `maxi` regressors included in the model.

We handle these constraints with a repair function.

```
repair <- function(p, Data) {

    ## remove negative values
    p <- p + abs(p)

    ## reduce cardinality of solutions (if required)
    tmp <- p > 0
    cs <- colSums(tmp)
    for (i in which(cs > Data$maxi)) {
        nneg <- which(tmp[ ,i])
        p[sample(nneg, length(nneg) - Data$maxi), i] <- 0
    }

    ## sum = 1
    if (Data$sum2one)
        for (i in seq_len(ncol(p)))
            p[ ,i] <- p[ ,i]/sum(p[ ,i])

    p
}
```

A test: we create a random population of `nP` solutions and repair it. The first three columns are displayed.

```
nP <- 200
pop <- array(rnorm(Data$ni * nP), dim = c(Data$ni, nP))

repair(pop, Data)[ ,1:3]
Data$sum2one <- TRUE
repair(pop, Data)[ ,1:3]
```

```
            [,1]       [,2]       [,3]
 [1,] 0.00000000 0.00000000 0.00000000
 [2,] 0.00000000 0.00000000 0.00000000
 [3,] 0.00000000 0.00000000 0.00000000
 [4,] 0.00000000 0.00000000 0.09037343
 [5,] 0.00000000 0.00000000 0.00000000
 [6,] 0.00000000 0.00000000 0.00000000
 [7,] 0.00000000 0.00000000 0.00000000
 [8,] 0.00000000 0.00000000 0.00000000
 [9,] 0.01820455 0.00000000 0.11609557
[10,] 0.00000000 0.00000000 0.00000000
[11,] 0.00000000 0.00000000 0.00000000
[12,] 0.14071222 0.00000000 0.00000000
[13,] 0.00000000 0.00000000 0.00000000
[14,] 0.00000000 0.00000000 0.00000000
[15,] 0.00000000 0.00000000 0.00000000
[16,] 0.00000000 3.39099072 0.00000000
[17,] 0.00000000 0.00000000 0.00000000
[18,] 0.00000000 0.00000000 0.00000000
[19,] 0.00000000 0.00000000 0.00000000
[20,] 0.00000000 0.00000000 0.00000000
[21,] 0.00000000 0.03087561 0.00000000
[22,] 0.00000000 0.00000000 0.00000000
[23,] 0.00000000 0.00000000 0.00000000
[24,] 0.00000000 0.63664925 0.00000000
[25,] 0.00000000 0.00000000 0.00000000
[26,] 0.00000000 0.00000000 0.00000000
[27,] 0.00000000 0.00000000 2.61028021
[28,] 0.00000000 0.00000000 0.00000000
[29,] 0.00000000 0.00000000 0.00000000
[30,] 1.19943852 0.00000000 0.00000000
          [,1]      [,2]      [,3]
 [1,] 0.0000000 0.0000000 0.1855297
 [2,] 0.0000000 0.0000000 0.0000000
 [3,] 0.0000000 0.0000000 0.0000000
 [4,] 0.0000000 0.0000000 0.0000000
 [5,] 0.0000000 0.2052219 0.0000000
 [6,] 0.0000000 0.6478631 0.0000000
 [7,] 0.0000000 0.0000000 0.0000000
 [8,] 0.0000000 0.0000000 0.0000000
 [9,] 0.0000000 0.0000000 0.0000000
[10,] 0.4687971 0.0000000 0.0000000
[11,] 0.0000000 0.0000000 0.0000000
```

```
[12,] 0.1524628 0.0000000 0.0000000
[13,] 0.0000000 0.0000000 0.0000000
[14,] 0.3787401 0.0000000 0.4471838
[15,] 0.0000000 0.0000000 0.0000000
[16,] 0.0000000 0.0000000 0.0000000
[17,] 0.0000000 0.0000000 0.0000000
[18,] 0.0000000 0.0000000 0.3672864
[19,] 0.0000000 0.0000000 0.0000000
[20,] 0.0000000 0.0000000 0.0000000
[21,] 0.0000000 0.0000000 0.0000000
[22,] 0.0000000 0.0000000 0.0000000
[23,] 0.0000000 0.0000000 0.0000000
[24,] 0.0000000 0.0000000 0.0000000
[25,] 0.0000000 0.0000000 0.0000000
[26,] 0.0000000 0.0000000 0.0000000
[27,] 0.0000000 0.0000000 0.0000000
[28,] 0.0000000 0.0000000 0.0000000
[29,] 0.0000000 0.1469150 0.0000000
[30,] 0.0000000 0.0000000 0.0000000
```

```r
settings <- list(min = rep(0, Data$ni),
                 max = rep(1, Data$ni),
                 nG = 2000,
                 nP = Data$ni * 5,
                 repair = repair,
                 loopOF = FALSE, loopRepair = FALSE,
                 printBar = FALSE)
sol <- DEopt(ofun, algo = settings, Data = Data)
data.frame(row.names = which(sol$xbest > 0),
           weights = round(sol$xbest[sol$xbest > 0], 2))
```

```
Differential Evolution.
Best solution has objective function value 0.9348385 ;
standard deviation of OF in final population is 3.96455e-17 .
  weights
3    0.37
4    0.35
8    0.28
```

Changing the cardinality constraint.

```r
Data$maxi <- 4
sol <- DEopt(ofun, algo = settings, Data = Data)
data.frame(row.names = which(sol$xbest > 0),
           weights = round(sol$xbest[sol$xbest > 0], 2))
```

```
Differential Evolution.
Best solution has objective function value 0.888393 ;
standard deviation of OF in final population is 0 .
   weights
3     0.32
4     0.25
```

```
8     0.25
26    0.18
```

Literature

Appendix: QP solution

See also GMS, p. 391.

```
settings <- list(min = rep(0, Data$ni),
                 max = rep(1, Data$ni),
                 nG = 2000,
                 nP = Data$ni * 5,
                 repair = repair,
                 loopOF = FALSE, loopRepair = FALSE,
                 printBar = FALSE)
Data$maxi <- Data$ni
sol <- DEopt(ofun, algo = settings, Data = Data)

D <- var(Data$X)
d <- cov(Data$y, Data$X)

min.w <- 0
AT <- rbind(1, diag(Data$ni))
b0 <- c(1, rep(min.w, Data$ni))
cbind(qp = round(solve.QP(D, d, t(AT), b0, meq = 1)$solution, 4),
      de = round(sol$xbest, 4))
```

```
Differential Evolution.
Best solution has objective function value 1.084127 ;
standard deviation of OF in final population is 0 .

         qp     de
 [1,] 0.0090 0.0090
 [2,] 0.0037 0.0037
 [3,] 0.0000 0.0000
 [4,] 0.0421 0.0421
 [5,] 0.0000 0.0000
 [6,] 0.0000 0.0000
 [7,] 0.0961 0.0961
 [8,] 0.1475 0.1475
 [9,] 0.0000 0.0000
[10,] 0.1309 0.1309
[11,] 0.0000 0.0000
[12,] 0.0000 0.0000
[13,] 0.0000 0.0000
[14,] 0.0000 0.0000
[15,] 0.0000 0.0000
[16,] 0.0178 0.0178
[17,] 0.0708 0.0708
[18,] 0.0000 0.0000
[19,] 0.0000 0.0000
[20,] 0.0000 0.0000
```

```
[21,] 0.0368 0.0368
[22,] 0.0000 0.0000
[23,] 0.0000 0.0000
[24,] 0.0000 0.0000
[25,] 0.1355 0.1355
[26,] 0.0000 0.0000
[27,] 0.1020 0.1020
[28,] 0.1684 0.1684
[29,] 0.0394 0.0394
[30,] 0.0000 0.0000
```

Appendix: More on variances and squared residuals

Two examples for quadratic problems that do not require a QP-solver.

No QP needed, example 1: minimising a sum of squares.

In the linear regression model

$$y = X\beta + \epsilon \tag{1}$$

the method of Least Squares minimizes the sum of the squared residuals $(y - X\beta)'(y - X\beta)$. Analytically, we would solve the normal equations

$$X'X\beta = X'y$$

for β. Note that if X contains a constant column, the residuals will have zero mean, and hence minimising the sum of squares is equivalent to minimizing the variance of the residuals (see the next example).

The sum of squares for a sample X and y can be written as:

$$\begin{aligned}(y - X\beta)'(y - X\beta) &= y'y - y'X\beta - (X\beta)'y + (Xb)'X\beta \\ &= y'y - 2y'X\beta + b'X'X\beta\end{aligned}$$

(The product $y'X\beta$ is a scalar and hence equals $\beta'X'y$.) We drop $y'y$ since it does not depend on β and divide by 2 to obtain

$$-\underbrace{y'X}_{c}\beta + \frac{1}{2}\beta'\underbrace{X'X}_{Q}\beta\,, \tag{2}$$

to be minimised. This looks exactly like the type of model that a QP solver could handle.

No QP needed, example 2: minimizing variance.

We stay with the regression model, but now we wish to minimize the variance of the residuals. We regard every row of X as the realization of a random variable. Hence, let x be a random vector, then we want to minimize

$$\begin{aligned}\operatorname{var}(y - x'\beta) &= \operatorname{var}(y) + \quad \operatorname{var}(x'\beta) - 2\operatorname{cov}(y, x'\beta) \\ &= \operatorname{var}(y) + \beta'\operatorname{var}(x')\beta - 2\operatorname{cov}(y, x')\beta\end{aligned}$$

We drop $\operatorname{var}(y)$ since it does not depend on β, divide by 2, and obtain

$$-\operatorname{cov}(y, x')\beta + \frac{1}{2}\beta' \operatorname{var}(x')\beta\,. \tag{3}$$

For a sample of X and y, this becomes

$$-\underbrace{\operatorname{cov}(y, X)}_{c}\beta + \frac{1}{2}\beta' \underbrace{\operatorname{var}(X)}_{Q}\beta \tag{4}$$

with $\operatorname{cov}(y, X)$ a vector of sample covariances between y and the columns of X, and $\operatorname{var}(X)$ the variance–covariance matrix of the columns of X. We an solve it without a QP but with techniques from linear regression. In R, for instance, we can handle such problems with the function `lmlm`}.

```
require("quadprog")

## create data
na <- 10
ns <- 100
X  <- array(rnorm(ns*na), dim = c(ns,na))
y  <- rnorm(ns)

## minimize squares
# variant 1 -- linear regression
coef(lm(y ~ 0 + X))
# variant 2 -- quadprog
Dmat <- crossprod(X)
dvec <- y %*% X
Amat <- as.matrix(rep(0,na))
solve.QP(Dmat = Dmat, dvec = dvec, Amat = Amat)

## minimize variance
# variant 1 -- linear regression
coef(lm(y ~ X))[-1]
# variant 2 -- quadprog
Dmat <- cov(X)
dvec <- cov(X,y)
Amat <- as.matrix(rep(0,na))
solve.QP(Dmat = Dmat, dvec = dvec,Amat = Amat)
```