Tuesday, September 29, 2020

# Fix non positive semi-definite covariance matrix in R

In [1], the following R code is proposed to solve a Mean-Variance portfolio optimization problem:

```
set.seed(123)

nO <- 10L  ## number of observations
nA <- 100L  ## number of assets
mData <- array(rnorm(nO * nA, sd = 0.05),
               dim = c(nO, nA)) #Creating sample stock observations

library("quadprog")
aMat <- array(1, dim = c(1,nA))
bVec <- 1
zeros <- array(0, dim = c(nA,1))

solQP <- solve.QP(cov(mData), zeros, t(aMat), bVec, meq = 1) #Minimize optimization
solQP$solution
```

I added the line that explicitly sets the seed to make the runs reproducible.

The model is a bit difficult to decipher in this R code, but it is a simplified traditional MV model:

**Mean-Variance Portfolio Optimization Model**

$$\begin{aligned}\min\ & x'Qx - \lambda\bar{r}'x\\ & \sum_i x_i = 1\\ & x_i \ge 0\end{aligned}$$

Instead of having the expected returns in the objective, we also see versions of this model with a linear constraint that puts a minimum value on th expected return of the portfolio.

In the example, we have $\lambda = 0$, so the objective only has a quadratic term to minimize the risk. We don't care about returns.

Note that the data only has 10 observations but 100 instruments. This situation can often lead to some problems. Indeed when we run this, we see

```
> solQP <- solve.QP(cov(mData), zeros, t(aMat), bVec, meq = 1) #Minimize optimization
Error in solve.QP(cov(mData), zeros, t(aMat), bVec, meq = 1) :
  matrix D in quadratic function is not positive definite!
>
```

As a side note: the data is just completely random, so we cannot expect useful solutions. However, at least, we should be able to produce some sol

## Solution #1

The covariance matrix is in theory positive-semi definite. We can see this from using the definition of the covariance [2]. However, due to floating-p inaccuracies, we can end up with a covariance matrix that is just slightly non-positive semi-definite. We can see this by inspecting the eigenvalues:

```
> Q <- cov(mData)
> eigen(Q)$values
  [1]  3.977635e-02  3.789523e-02  3.258164e-02  3.093782e-02  2.605389e-02  2.466463e-02  2.127899e-02
  [8]  1.897506e-02  1.627900e-02  1.871945e-17  1.454315e-17  8.114384e-18  5.109196e-18  4.392427e-18
```

```
[15]  3.680766e-18  2.728485e-18  2.649302e-18  2.192073e-18  1.937634e-18  1.914487e-18  1.856811e-18
[22]  1.449321e-18  1.391434e-18  1.211216e-18  1.196721e-18  1.127119e-18  9.776146e-19  9.670197e-19
[29]  8.257462e-19  7.572554e-19  6.283522e-19  6.030820e-19  4.056541e-19  3.610128e-19  3.604800e-19
[36]  2.770106e-19  2.530797e-19  2.263314e-19  2.133854e-19  2.089700e-19  2.024130e-19  2.006560e-19
[43]  1.688435e-19  9.412675e-20  7.033666e-20  7.012531e-20  5.942729e-20  5.673501e-20  4.444257e-20
[50]  4.259787e-20  3.124233e-21 -2.017011e-20 -2.269857e-20 -3.034668e-20 -3.075493e-20 -3.262642e-20
[57] -4.345710e-20 -6.351674e-20 -7.347496e-20 -7.739535e-20 -8.065131e-20 -9.318415e-20 -1.004645e-19
[64] -1.048992e-19 -1.185271e-19 -1.315539e-19 -1.362383e-19 -1.376705e-19 -1.456396e-19 -1.697860e-19
[71] -1.700317e-19 -1.745068e-19 -1.794480e-19 -1.835296e-19 -1.960933e-19 -2.017457e-19 -2.332259e-19
[78] -2.377082e-19 -3.232830e-19 -3.468225e-19 -3.844707e-19 -4.310383e-19 -4.467493e-19 -4.540707e-19
[85] -4.977900e-19 -6.159103e-19 -6.338311e-19 -6.484161e-19 -6.489508e-19 -6.551306e-19 -6.664641e-19
[92] -7.276688e-19 -7.422977e-19 -7.927766e-19 -1.106209e-18 -1.707077e-18 -1.742870e-18 -2.465438e-18
[99] -1.859346e-17 -3.473638e-17
```

Most of the eigenvalues are basically zero with quite a few just a bit negative. A positive-definite matrix would have only positive values. A positive definite (PSD) matrix would allow positive and zero eigenvalues but no negative ones.

There are algorithms to find a close-by positive-definite matrix. In R this is available as `nearPD()` in the `Matrix` package. Let's try this out:

```
> Q <- nearPD(cov(mData))$mat
> solQP <- solve.QP(Q, zeros, t(aMat), bVec, meq = 1)
> solQP$solution
  [1]  0.007861808  0.004649581  0.012677143  0.009135964  0.011926439  0.013624409  0.007916747  0.004307772
  [9]  0.011808486  0.011085302  0.010229029  0.013080240  0.010085504  0.010998879  0.011099784  0.008520520
 [17]  0.007451730  0.012424371  0.012677330  0.007013515  0.009070013  0.010422999  0.011041407  0.008768456
 [25]  0.007594051  0.010845828  0.008091472  0.010219700  0.017100764  0.005583947  0.011981802  0.007366402
 [33]  0.013770522  0.009063828  0.010744898  0.007836608  0.009580356  0.007065588  0.008219370  0.010592597
 [41]  0.011775598  0.011112051  0.005555992  0.014596684  0.016019481  0.008870919  0.010940604  0.005662200
 [49]  0.007056997  0.012274299  0.006510091  0.012764160  0.008258651  0.009945869  0.009891680  0.007318763
 [57]  0.014960558  0.008255708  0.009655097  0.010054760  0.007309219  0.008761360  0.011616297  0.010300254
 [65]  0.012043733  0.011963495  0.009991699  0.010353212  0.013896332  0.011619403  0.007160011  0.007202327
 [73]  0.008530819  0.010918658  0.004754215  0.008877062  0.008526937  0.010251250  0.008190446  0.006754629
 [81]  0.013209816  0.014849748  0.012213860  0.008888433  0.018188014  0.009756690  0.014786487  0.014893706
 [89]  0.009737739  0.013256532  0.009990737 -0.001155889  0.009264992  0.007733719  0.014505131  0.011040198
 [97]  0.012859850  0.008987104  0.008458713  0.004497735
```

Much better. When printing the eigenvalues of the new Q matrix, we see:

```
> eigen(Q)$values
  [1] 3.977635e-02 3.789522e-02 3.258164e-02 3.093782e-02 2.605389e-02 2.466462e-02 2.127899e-02 1.897506e-02
  [9] 1.627900e-02 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10
 [17] 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10
 [25] 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10
 [33] 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10
 [41] 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10
 [49] 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10
 [57] 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10
 [65] 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10
 [73] 3.977635e-10 3.977635e-10 3.977635e-10 3.977635e-10 3.977634e-10 3.977634e-10 3.977634e-10 3.977634e-10
 [81] 3.977634e-10 3.977634e-10 3.977634e-10 3.977634e-10 3.977634e-10 3.977634e-10 3.977634e-10 3.977634e-10
 [89] 3.977634e-10 3.977634e-10 3.977634e-10 3.977634e-10 3.977634e-10 3.977634e-10 3.977634e-10 3.977634e-10
 [97] 3.977634e-10 3.977634e-10 3.977633e-10 3.977633e-10
```

All small (and negative) eigenvalues are now mapped to some tolerance. This means the Q matrix is now numerically positive definite.

### Solution #2

A simple DIY approach would be to fix the Q matrix by adding a small number to the diagonal. The following code shows a small perturbation is eno

```
> Q <- cov(mData) + diag(1e-6,nA)
```

Some solvers do something like this automatically.

### Fixing the model

Let's first fix our model a bit:

- We want to pay attention to returns. So let's add a constraint that the expected portfolio returns are at least 0.035. This also has the effect tha just a few assets will be in our solution. Which makes them easier to compare. The constraint can be stated as

  $$\sum_i \bar{r}_i \geq 0.035$$

  where

  $$\bar{r}_i = \frac{\sum_t r_{t,i}}{T}$$

  are the mean returns. Here $T$ indicates the number of observations, and $r_{t,i}$ are the returns.
- We also want to explicitly state that variables are non-negative: $x_i \geq 0$. This is called "no short-selling",

The portfolio model becomes:

| Portfolio Model algebraic formulation | matrix notation |
|---|---|
| $\begin{aligned} \min \sum_{i,j} q_{i,j} x_i x_j \\ \sum_i x_i = 1 \\ \sum_i \bar{r}_i x_i \geq 0.035 \\ x_i \geq 0 \end{aligned}$ | $\begin{aligned} \min\ & x'Qx \\ & e'x = 1 \\ & \bar{r}'x \geq 0.035 \\ & x \geq 0 \end{aligned}$ |

**QuadProg** does not have facilities for bounds on the variables so we need to form explicit constraints.  Its input-model looks like:

$$\begin{aligned} \min\ & -d'b + \frac{1}{2} b'Db \\ & A'b \geq b_0 \end{aligned}$$

where the first `meq` constraints are equalities. The factor 0.5 is a relic from quadratic optimization (it makes the gradient a bit easier). If the model includes linear objective coefficients, it is important to take this 0.5 factor into account. Our model has only quadratic objective coefficients, so we d care in this case. All in all, this is often a bit of an inconvenient input format.

**R trick.** In the examples below we want to display a sparse vector. A simple trick using names can help here.

```
> v <- c(0,0,3,0,0,0,1,0,0,0,2,0,0,0,0)
> v
 [1] 0 0 3 0 0 0 1 0 0 0 2 0 0 0 0
> names(v) <- paste("v",1:length(v),sep="")
> v[v>0]
 v3  v7 v11 
  3   1   2 
```

The R code for our optimization model can look like:

```
> Q <- nearPD(cov(mData))$mat
> aMat <- rbind(
+     c(rep(1,nA)),     # sum(x)=1
+     colMeans(mData),  # sum(mu(i)*x(i))>=0.035
+     diag(1,nA))       # for bounds x(i)>=0
> bVec <- c(1,
+          0.035,
+          rep(0,nA))
> solQP <- solve.QP(Q, zeros, t(aMat), bVec, meq = 1)
> xsol <- solQP$solution
> names(xsol) = paste("x",1:100,sep="")
> xsol[xsol>1e-4]
      x30       x75       x88 
0.1465990 0.5314489 0.3219522 
```

Another issue with QuadProg is that requires a strictly positive definite matrix. This also means it does not allow linear variables: all variables need to quadratically in the objective (and without all zero quadratic coefficients). After all, if the quadratic coefficient matrix would look like

$$D = \left[ \begin{array}{c|c} Q & 0 \\ \hline 0 & 0 \end{array} \right]$$

we would end up with a matrix that is not strictly positive-definite.

Often, for these types of models, CVXR is a more agreeable tool. Here is the same model in CVXR, and solved with the OSQP QP solver:

```
> library(CVXR)
> x <- Variable(nA)
> w <- Variable(nO)
> prob <- Problem(
+     Minimize(quad_form(x,cov(mData))),
+     list(sum(x) == 1,
+          t(colMeans(mData)) %*% x >= 0.035,
+          x >= 0)
+     )
> result <- solve(prob, verbose=T)
-----------------------------------------------------------------
           OSQP v0.6.0  -  Operator Splitting QP Solver
              (c) Bartolomeo Stellato,  Goran Banjac
        University of Oxford  -  Stanford University 2019
-----------------------------------------------------------------
problem:  variables n = 100, constraints m = 102
          nnz(P) + nnz(A) = 5350
settings: linear system solver = qdldl,
          eps_abs = 1.0e-05, eps_rel = 1.0e-05,
          eps_prim_inf = 1.0e-04, eps_dual_inf = 1.0e-04,
          rho = 1.00e-01 (adaptive),
          sigma = 1.00e-06, alpha = 1.60, max_iter = 10000
          check_termination: on (interval 25),
          scaling: on, scaled_termination: off
          warm start: on, polish: on, time_limit: off

iter  objective    pri res    dua res    rho        time
   1   0.0000e+00   1.00e+00   1.00e+02   1.00e-01   2.88e-03s
 200   8.7535e-04   5.96e-04   1.36e-06   1.75e-02   9.38e-03s
 400   9.2707e-04   5.09e-04   1.16e-06   1.75e-02   1.41e-02s
 600   9.7703e-04   4.36e-04   9.94e-07   1.75e-02   1.99e-02s
 800   1.0239e-03   3.72e-04   8.50e-07   1.75e-02   2.58e-02s
1000   1.0671e-03   3.18e-04   7.27e-07   1.75e-02   3.11e-02s
1200   1.1062e-03   2.72e-04   6.21e-07   1.75e-02   3.76e-02s
1400   1.1413e-03   2.33e-04   5.31e-07   1.75e-02   4.35e-02s
1600   1.1726e-03   1.99e-04   4.54e-07   1.75e-02   4.90e-02s
1800   1.2001e-03   1.70e-04   3.88e-07   1.75e-02   5.51e-02s
2000   1.2243e-03   1.45e-04   3.32e-07   1.75e-02   6.20e-02s
2200   1.2455e-03   1.24e-04   2.84e-07   1.75e-02   6.81e-02s
2400   1.2639e-03   1.06e-04   2.43e-07   1.75e-02   7.33e-02s
2600   1.2799e-03   9.09e-05   2.07e-07   1.75e-02   7.83e-02s
2800   1.2938e-03   7.77e-05   1.77e-07   1.75e-02   8.51e-02s
3000   1.3058e-03   6.65e-05   1.52e-07   1.75e-02   9.20e-02s
3200   1.3161e-03   5.68e-05   1.30e-07   1.75e-02   1.00e-01s
3400   1.3251e-03   4.86e-05   1.11e-07   1.75e-02   1.08e-01s
3600   1.3327e-03   4.15e-05   9.48e-08   1.75e-02   1.21e-01s
3800   1.3394e-03   3.55e-05   8.11e-08   1.75e-02   1.28e-01s
4000   1.3450e-03   3.04e-05   6.93e-08   1.75e-02   1.35e-01s
4200   1.3499e-03   2.60e-05   5.92e-08   1.75e-02   1.43e-01s
4400   1.3541e-03   2.22e-05   5.07e-08   1.75e-02   1.50e-01s
4550   1.3568e-03   1.97e-05   4.50e-08   1.75e-02   1.56e-01s
plsh   1.3790e-03   3.33e-16   1.21e-15   ---------   1.58e-01s

status:               solved
solution polish:      successful
number of iterations: 4550
optimal objective:    0.0014
run time:             1.58e-01s
```

```
optimal rho estimate: 2.58e-02

> xsol <- result$getValue(x)
> names(xsol) = paste("x",1:100,sep="")
> xsol[xsol>1e-4]
      x30       x75       x88 
0.1465989 0.5314489 0.3219522 
>
```

Here we use our covariance matrix directly. Both CVXR and OSQP accept it, even though it has tiny negative numbers. However, the OSQP docume warns that this may be a bit dangerous [3]:

> We recommend the user to **check the convexity of their problem before passing it to OSQP**! If the user passes a non-convex problem we do not assure the solver will be able to detect it.
>
> OSQP will try to detect **non-convex** problems by checking if the residuals diverge or if there are any issues in the initial factorization (if a direct method is used). It will detect non-convex problems when one or more of the eigenvalues of `P` are "clearly" negative, i.e., when `P + sigma * I` is not positive semidefinite. However, it might fail to detect non-convexity when `P` has slightly negative eigenvalues, i.e., when `P + sigma * I` is positive semidefinite and `P` is not.

Notice that CVXR will complain if the Q matrix is really non-positive semi-definite.

## Solution #3

In [2] a different model is proposed for these situations where we have more instruments than observations.

**Means-adjusted-returns Portfolio Optimization Model**

$$\begin{aligned}
\min \sum_t & w_t^2 \\
& w_t = \sum_i (r_{t,i} - \bar{r}_i) x_i \\
& \sum_i x_i = 1 \\
& \sum_i \bar{r}_i x_i \geq 0.035 \\
& x_i \geq 0
\end{aligned}$$

Some of the advantages of this model are:

- The quadratic objective is benign. It is sparse and safely positive definite.
- The data is smaller than a full-blown covariance matrix given that the number of observations is smaller than the number of instruments.
- The number of quadratic variables is smaller (again, because $T < N$).

This model is easily implemented in CVXR:

```
> library(CVXR)
> x <- Variable(nA)
> w <- Variable(nO)
> adjRet <- mData - matrix(colMeans(mData),nrow=nO,ncol=nA,byrow=T)
> prob <- Problem(
+     Minimize(sum_squares(w)),
+     list(w == adjRet %*% x,
+          sum(x)==1,
+          t(colMeans(mData)) %*% x >= 0.035,
+          x >= 0)
+     )
> result <- solve(prob, verbose=T)
-----------------------------------------------------------------
           OSQP v0.6.0  -  Operator Splitting QP Solver
              (c) Bartolomeo Stellato,  Goran Banjac
```

```
        University of Oxford  -  Stanford University 2019
-----------------------------------------------------------------
problem:  variables n = 110, constraints m = 112
          nnz(P) + nnz(A) = 1320
settings: linear system solver = qdldl,
          eps_abs = 1.0e-05, eps_rel = 1.0e-05,
          eps_prim_inf = 1.0e-04, eps_dual_inf = 1.0e-04,
          rho = 1.00e-01 (adaptive),
          sigma = 1.00e-06, alpha = 1.60, max_iter = 10000
          check_termination: on (interval 25),
          scaling: on, scaled_termination: off
          warm start: on, polish: on, time_limit: off

iter  objective    pri res    dua res    rho        time
   1   0.0000e+00   1.00e+00   1.00e+02   1.00e-01   1.15e-03s
 200   5.3515e-03   1.49e-03   4.04e-05   1.00e-01   4.33e-03s
 400   7.0787e-03   7.87e-04   6.50e-06   1.00e-01   8.41e-03s
 600   7.3521e-03   7.13e-04   5.90e-06   1.00e-01   1.23e-02s
 800   7.6413e-03   6.45e-04   5.34e-06   1.00e-01   1.61e-02s
1000   7.9373e-03   5.84e-04   4.84e-06   1.00e-01   1.93e-02s
1200   8.2334e-03   5.29e-04   4.38e-06   1.00e-01   2.27e-02s
1400   8.5244e-03   4.79e-04   3.96e-06   1.00e-01   2.64e-02s
1600   8.8067e-03   4.33e-04   3.59e-06   1.00e-01   3.00e-02s
1800   9.0778e-03   3.92e-04   3.25e-06   1.00e-01   3.34e-02s
2000   9.3358e-03   3.55e-04   2.94e-06   1.00e-01   3.69e-02s
2200   9.5798e-03   3.22e-04   2.66e-06   1.00e-01   4.05e-02s
2400   9.8091e-03   2.91e-04   2.41e-06   1.00e-01   4.43e-02s
2600   1.0024e-02   2.64e-04   2.18e-06   1.00e-01   4.81e-02s
2800   1.0224e-02   2.39e-04   1.98e-06   1.00e-01   5.12e-02s
3000   1.0410e-02   2.16e-04   1.79e-06   1.00e-01   5.48e-02s
3200   1.0582e-02   1.96e-04   1.62e-06   1.00e-01   5.84e-02s
3400   1.0741e-02   1.77e-04   1.47e-06   1.00e-01   6.19e-02s
3600   1.0887e-02   1.60e-04   1.33e-06   1.00e-01   6.55e-02s
3800   1.1022e-02   1.45e-04   1.20e-06   1.00e-01   6.90e-02s
4000   1.1145e-02   1.31e-04   1.09e-06   1.00e-01   7.53e-02s
4200   1.1259e-02   1.19e-04   9.85e-07   1.00e-01   8.18e-02s
4400   1.1362e-02   1.08e-04   8.92e-07   1.00e-01   8.95e-02s
4600   1.1457e-02   9.75e-05   8.07e-07   1.00e-01   9.44e-02s
4800   1.1544e-02   8.83e-05   7.31e-07   1.00e-01   1.04e-01s
5000   1.1623e-02   7.99e-05   6.62e-07   1.00e-01   1.09e-01s
5200   1.1695e-02   7.24e-05   5.99e-07   1.00e-01   1.13e-01s
5400   1.1761e-02   6.55e-05   5.42e-07   1.00e-01   1.20e-01s
5600   1.1821e-02   5.93e-05   4.91e-07   1.00e-01   1.26e-01s
5800   1.1876e-02   5.37e-05   4.45e-07   1.00e-01   1.31e-01s
6000   1.1925e-02   4.86e-05   4.02e-07   1.00e-01   1.37e-01s
6200   1.1970e-02   4.40e-05   3.64e-07   1.00e-01   1.45e-01s
6400   1.2011e-02   3.98e-05   3.30e-07   1.00e-01   1.58e-01s
6600   1.2049e-02   3.61e-05   2.99e-07   1.00e-01   1.72e-01s
6800   1.2082e-02   3.27e-05   2.70e-07   1.00e-01   1.81e-01s
7000   1.2113e-02   2.96e-05   2.45e-07   1.00e-01   1.90e-01s
7200   1.2141e-02   2.68e-05   2.22e-07   1.00e-01   1.98e-01s
7400   1.2166e-02   2.42e-05   2.01e-07   1.00e-01   2.01e-01s
7600   1.2189e-02   2.19e-05   1.82e-07   1.00e-01   2.05e-01s
7800   1.2210e-02   1.99e-05   1.64e-07   1.00e-01   2.08e-01s
plsh   1.2411e-02   6.66e-16   1.28e-15   ---------   2.09e-01s

status:               solved
solution polish:      successful
number of iterations: 7800
optimal objective:    0.0124
run time:             2.09e-01s
optimal rho estimate: 3.27e-01

> xsol <- result$getValue(x)
> names(xsol) = paste("x",1:100,sep="")
> xsol[xsol>1e-4]
      x30       x75       x88
0.1465989 0.5314489 0.3219522
```

Using QuadProg this is a much more difficult exercise. The quadratic coefficient matrix $D$ will look like:

$$D = \left[ \begin{array}{c|c} Q & 0 \\ \hline 0 & \varepsilon I \end{array} \right]$$

The R code can look like:

```
> Q <- nearPD(cov(mData))$mat
> D <- rbind(
+   cbind(Q,matrix(0,nrow=nA,ncol=nO)),
+   cbind(matrix(0,nrow=nO,ncol=nA),diag(1e-6,nO))
+ )
> aMat <- rbind(
+   cbind(adjRet,diag(-1,nO)),     # sum(i,adjRet(t,i)*x(i))-w(t)=0
+   c(rep(1,nA),rep(0,nO)),        # sum(x)=1
+   c(colMeans(mData),rep(0,nO)),  # sum(mu(i)*x(i))>=0.035
+   cbind(diag(1,nA),matrix(0,nrow=nA,ncol=nO)) # for bounds x(i)>=0
+ )
> bVec <- c(rep(0,nO),
+           1,
+           0.035,
+           rep(0,nA))
> zeros <- rep(0,nA+nO)
> solQP <- solve.QP(D, zeros, t(aMat), bVec, meq = nO+1)
> xsol <- solQP$solution[1:nA]
> names(xsol) = paste("x",1:100,sep="")
> xsol[xsol>1e-4]
      x30       x75       x88
0.1465990 0.5314489 0.3219522
```

## Conclusion

**QuadProg** is a well-known QP solver for R. Unfortunately it has some drawbacks:

- It is not very suited for more complex models: the matrix oriented input format is just too cumbersome.
- The quadratic coefficient matrix must be strictly positive definite.
- All variables are quadratic, linear variables are not available.

For quick experimentation with these types of models, **CVXR** is likely a better environment. This exercise showed:

- The models are more readable, assuming you are familiar with matrix notation.
- We can formulate models with linear variables without a problem.
- Models with a quadratic coefficient matrix that are borderline positive semi-definite are accepted by both CVXR and the solver I used: OSQP.

Two useful **R tricks** are discussed:

- `nearPD(A)` to find a positive-definite matrix close to A.
- Printing a sparse vector using names.

## References


1. R minimum variance portfolio: solve non invertible matrix, https://stackoverflow.com/questions/64067591/r-minimum-variance-portfolio-solv invertible-matrix
2. Covariance Matrix not positive definite in portfolio models, https://yetanothermathprogrammingconsultant.blogspot.com/2018/04/covariance-r not-positive-definite.html
3. OSQP, Status values and errors, https://osqp.org/docs/interfaces/status_values.html


Posted by Erwin Kalvelagen at 6:17PM

Labels: Convex Optimization, Portfolio optimization

## No comments:

## Post a Comment

Enter Comment

Newer Post Home Olde

Subscribe to: Post Comments (Atom)