Linear Projection as Optimization Problem

Hans W. Borchers

2018-04-11

Introduction

Linear Projection as Optimization Task

Given an affine subspace S of R^n, defined through a matrix equation $Ax = b$ with a matrix A of size (m, n) and a vector b of length m, the task is to find the *projection* of a point p_0 onto this subspace. We could solve this problem as a pure Linear Algebra task. Then we would look at a decomposition of R^n as an orthogonal sum $A \oplus U$ of A and another subspace U. Then represent p_0 in a basis of U.

Here we will solve the problem as an optimization task. If p is the projection of p_0 on A, then the distance between p_0 and p is minimal for all points in S, that is p with $Ap = b$. We will solve this for an example with $n = 1000$ and $m = 100$.

Generating Example Data

```
set.seed(65537)
n = 1000; m = 100
p0 <- rep(0, n)
Aeq <- matrix(runif(m*n), nrow = m)
beq <- rep(1, m)
```

Solving the Optimization Task

Solution with *quadprog*

```
library(quadprog)

Q = diag(n)
d = as.matrix(p0)
A = t(Aeq)
b = as.matrix(beq)
```

```r
system.time( sol <- solve.QP(Q, d, A, b, meq=m) )
```

```
##    user  system elapsed
##   1.304   0.000   1.304
```

```r
p = sol$solution
d = sqrt(sum((p - p0)^2))
cat("The minimal distance is:", d)
```

```
## The minimal distance is: 0.06597077
```

Solution with *CVXR*

```r
library(CVXR)
```

```
##
## Attaching package: 'CVXR'

## The following object is masked from 'package:stats':
##
##     power
```

```r
x   = Variable(n)
obj = Minimize( sum((p0 - x)^2) )
con = list(Aeq %*% x == beq)
pro = Problem(obj, con)

system.time( sol <- solve(pro) )
```

```
##    user  system elapsed
##   0.473   0.024   0.497
```

```r
cat("The minimal distance is:", sqrt(sol$value))
```

```
## The minimal distance is: 0.06597077
```

We can also try out the other available solver, SCS:

```r
system.time( sol <- solve(pro, solver = "SCS") )
```

```
##    user  system elapsed
##   0.209   0.000   0.209
```

Use the ECOS Solver Directly

Instead of employing *CVXR* we can call the underlying ECOS solver directly. First we define the `mindist` function that builds the necessary matrices and then sends them to `ECOS_csolve`.

```r
library(ECOSolveR)

mindist <- function(x0, Aeq, beq) {
    n <- length(x0)
    cc <- c(rep(0, n), 1)
    A_eq <- Matrix::Matrix(cbind(Aeq, 0), sparse = TRUE)
    b_eq <- beq
    A2 <- rbind(c(rep(0, n), -1),
                cbind(diag(1, n), 0))
    G <- Matrix::Matrix(rbind(A2), sparse = TRUE)
    h <- c(0, x0)
    sol <- ECOS_csolve(cc, G = G, h = h,
                       dims = list(q = as.integer(n+1)),
                       A = A_eq, b = b_eq)
    return(list(point = sol$x[1:n], dist = sol$x[n+1],
                message = sol$infostring))
}
```

```r
system.time(sol <- mindist(p0, Aeq, beq))
```

```
##    user  system elapsed
##   0.611   0.016   0.627
```

```r
cat("The minimal distance is:", sol$dist)
```

```
## The minimal distance is: 0.06597077
```

Prepare for Several Projections

Or we set up the matrices needed before we call the solver:

```r
n = length(p0)
cc = c(rep(0, n), 1)
A_eq = Matrix::Matrix(cbind(Aeq, 0), sparse = TRUE)
b_eq = beq
A2 = rbind(c(rep(0, n), -1),
           cbind(diag(1, n), 0))
G <- Matrix::Matrix(rbind(A2), sparse = TRUE)
```

```r
h <- c(0, p0)
system.time(
    sol <- ECOS_csolve(cc, G = G, h = h,
                        dims = list(q = as.integer(n+1)),
                        A = A_eq, b = b_eq)
)
```

```
##    user  system elapsed
##   0.241   0.004   0.245
```

```r
dist = sol$x[n+1]
cat("The minimal distance is:", d)
```

```
## The minimal distance is: 0.06597077
```