UNIVERSIDAD POLITÉCNICA DE CATALUNYA

Primal Affine Scaling Algorithm

Ignasi Mañé Bosch and Joaquim Girbau Xalabarder

Professor: Jordi Castro

Contents

1 The Primal Affine Scaling Algorithm

The affine-scaling algorithm was introduced by I.I.Dikin in 1967. It is the simplest interior-point method, provides good computational results, and is easy to implement (this is the goal of the assignment). It is defined as follows:

INITIALIZATIONS

Compute infeasibilities: $r = b - Ae_n$, where $e_l = (1_1, \ldots, 1_l)^\top$
Extend A with additional column r: $A \leftarrow [A\ r]$, $n \leftarrow n + 1$
Extend cost vector: $c \leftarrow [c^\top\ M]^\top$, $M \in \mathbb{R}$ big
$x^0 = e_{n+1}$; x^0 is interior ($x^0 > 0$) and feasible ($Ax^0 = b$)
$D = I$, $y = (ADA^\top)^{-1}ADc$
$k = 0$, $\varepsilon = 10^{-6}$, $\rho \in [0.95, 0.9995]$

ITERATIVE PROCEDURE

while $\frac{|c^\top x^k - b^\top y|}{1 + |c^\top x^k|} \varepsilon$ **do**
    Compute z: $z = c - A^\top y$
    Compute Δx: $\Delta x = -Dz$
    if ($\Delta x \geq 0$) **then** STOP: Unbounded problem
    Compute α: $\alpha = \rho \cdot \min \left\{ -\frac{x_i^k}{\Delta x_i}\ \forall i\ \Delta x_i \leq 0 \right\}$
    Compute x^{k+1}: $x^{k+1} = x^k + \alpha \Delta x$
    $k \leftarrow k + 1$
    Compute X^k: $X^k = \text{diag}(x_1^k, , \ldots, x_n^k)$
    Compute D: $D = (X^k)^2$
    Compute y: $(ADA^\top)y = ADc$
end_while

END

if $x_{n+1}^k \neq 0$ **then**
    STOP: Infeasible problem
else
    STOP: Optimal solution found: $x^* \leftarrow (x_1^k, \ldots, x_n^k)^\top$
end_if

Figure 1: Primal Affine Scaling Algorithm

To guarantee that the algorithm converges to the global optimal solution, the following three assumptions must hold.

Assumption 1. *Matrix $A \in \mathbb{R}^{mxn}$ is a full rank matrix.*

Assumption 2. *The solution of the primal problem is non-degenerate.*

Assumption 3. *The solution of the Dual problem is non-degenerate.*

2 R Implementation

The primal affine scaling algorithm has been implemented in R, an open source language and environment for statistical computing and graphics freely available at http://cran.r-project.org, along with the Matrix package, a library to efficiently operate with matrices and vectors.

2.1 Function affineScaling

Given a numeric vector c, a matrix A and a numeric vector b, the function affineScaling solves the associated Linear Programming Problem in standard form

$$\min_{x \in \mathbb{R}^n} \{c'x : Ax = b, x \geq 0\}$$

using the primal affine scaling algorithm. Where $x \in \mathbb{R}^n$ is the vector of variables, $b \in \mathbb{R}^m$ is the right hand side term and $A \in \mathbb{R}^{mxn}$ is the constraints matrix.

Arguments

- c: a Matrix object containing vector c
- A: a Matrix object containing matrix A
- b: a Matrix object containing vector b
- fM: a numeric value to control the value of the big M. M=fM*max(abs(c))
- scaling: a boolean value indicating whether or not apply scaling
- iter: a numeric value indicating the maximum number of iterations.

Values

- results: a list containing the value of the objective function, a data frame of two variables with each iteration and its relative gap, and the global optima of the problem. If the maximum number of iterations is reached, the function stops and returns the same information until this point.

```
library(Matrix)

affineScaling <- function(c, A, b, fM=100, scaling=TRUE, iter=500) {

# INITIALIZATIONS
    n <- dim(A)[2] # Number of variables in A
    m <- dim(A)[1] # Number of rows in A (equality constraints)
    M <- fM*max(abs(c))

    el <- Matrix(rep(1,n)) # Vector of ones with n columns

    # 1) COMPUTE THE INFEASIBILITIES
    r <- b - A%*%el

    # 2) EXTEND A WITH ADDITIONAL COLUMN
    A <- cbind(A,r)

    # 3) EXTEND COST VECTOR
    c <- rbind(c,M) #costs associated to the new variable

    # 4) x0 interior and feasible
    x <- Matrix(rep(1,n+1)) #initial solution

    # 5) COMPUTE D=I and y
    D <- Diagonal(n+1) #scaling matrix
    y <- solve(A%*%D%*%t(A),A%*%D%*%c,tol = 1e-30)
```

```
    #Cholesky code (not working +semidefinite)
    #y <- solve(Cholesky(as(A%*%D%*%t(A),"symmetricMatrix"),
    #LDL=FALSE),A%*%D%*%c,system="L")

    # 6) Parameters
    k <- 0 # First iteration
    epsi <- 1e-6 # Max relative duality gap
    rho <- 0.95  # Factor to reduce alfa (move away null comp)

# ITERATIVE PROCEDURE

    # 7)
    rel.gap <- abs(t(c)%*%x - t(b)%*%y)/(1+abs(t(c)%*%x))
    results <- data.frame(Iter=0,Gap=as.numeric(rel.gap))

    while ( (as.numeric(rel.gap) > epsi) & (k <= iter) ) {
        #print(k)
        # 8)
        z <- c - t(A)%*%y
        # 9)
        d <- -D%*%z
        # 10)
        if (sum(d>=0)==n+1) {
            print("Unbounded Problem")
            return("Unbounded Problem")
            }
        # 11)
        ratio <- -x/d
        alpha <- rho*min(ratio[d<=0])
        # 12)
        x <- x + alpha*d
        # 13)
        k <- k + 1

        if (scaling==TRUE) {
        # 14)
            X <- Diagonal(x = as.numeric(x))
        # 15)
            D <- X^2
        }
        # 16)
        y <- solve(A%*%D%*%t(A),A%*%D%*%c,tol = 1e-30)

        #Cholesky code (not working +semidefinite)
        #y <- solve(Cholesky(as(A%*%D%*%t(A),"symmetricMatrix"),
        #LDL=FALSE),A%*%D%*%c,system="L")

        rel.gap <- abs(t(c)%*%x - t(b)%*%y)/(1+abs(t(c)%*%x))

        results[k,] <- c(k,round(as.numeric(rel.gap),8))

    } # END
```

```
    if(k>=500 & as.numeric(rel.gap)>epsi) {
      print("No optimal solution found. Max Num iterations reached")
      opt<-round(x[1:n],5) #optima
      Iter<-results #Relative gap per iteration
      Obj <- t(c)%*%x #Objective function value

      return(list(Optim=opt,Iter=Iter,Obj=as.numeric(Obj)))

    }
    # 18)
    if (x[n+1]>epsi) {
    # 19)
       print("Infeasible Problem")
       return("Infeasible Problem")
    }
    # 20) 21) 22)
    else {
      opt<-round(x[1:n],5) #optima
      Iter<-results #Relative gap per iteration
      Obj <- t(c)%*%x #Objective function value

      print("Optimal Solution found")
      return(list(Optim=opt,Iter=Iter,Obj=as.numeric(Obj)))
    }

}
```

3 Testing the Algorithm

In order to test the code, problems 596, 597 and 598 from the Netlib list of linear problems have been attempted to solve using the affineScaling R function. All the information related to them can be found on this link.

3.1 Solving LP Problem 596 in R

We load the matrix A, the right hand side vector b, and the coefficient cost vector c from the problem 596. Then, the problem is solved with and without scaling using the affineScaling function made in the previous section; the result is stored in sol1. The objective function and the number iterations done are reported.

```
A <- Matrix(as.matrix(read.csv("data/P2/ex596A.csv", header=FALSE)))
b <- Matrix(read.table("data/P2/ex596b.csv", quote="\", comment.char="")$V1)
c <- Matrix(read.table("data/P2/ex596c.csv", quote="\", comment.char="")$V1)

sol1 <- affineScaling(c = c,A = A,b = b,fM=100, scaling=TRUE)
sol1$Obj #obective function
max(sol1$Iter[,1]) #iterations

#without scaling

sol1 <- affineScaling(c = c,A = A,b = b,fM=100, scaling=FALSE)
```

```
round(sol1$Obj,4)
max(sol1$Iter[,1])
```

3.2 Solving LP Problem 597 in R

We do the same as before using the data from the problem 597.

```
A <- Matrix(as.matrix(read.csv("data/P1/ex597A.csv", header=FALSE)))
b <- Matrix(read.table("data/P1/ex597b.csv", quote="\", comment.char="")$V1)
c <- Matrix(read.table("data/P1/ex597c.csv", quote="\", comment.char="")$V1)

sol1 <- affineScaling(c = c,A = A,b = b,fM=100, scaling=TRUE)
sol1$Obj #obective function
max(sol1$Iter[,1]) #iterations

#without scaling

sol1 <- affineScaling(c = c,A = A,b = b,fM=100, scaling=FALSE)
round(sol1$Obj,4)
max(sol1$Iter[,1])
```

3.3 Solving LP Problem 598

We do the same as in the section of problem 596 using the data from the problem 598.

```
A <- Matrix(as.matrix(read.csv("data/P3/ex598A.csv", header=FALSE)))
b <- Matrix(read.table("data/P3/ex598b.csv", quote="\", comment.char="")$V1)
c <- Matrix(read.table("data/P3/ex598c.csv", quote="\", comment.char="")$V1)

sol1 <- affineScaling(c = c,A = A,b = b,fM=100, scaling=TRUE)
sol1$Obj #obective function
max(sol1$Iter[,1]) #iterations

#without scaling

sol1 <- affineScaling(c = c,A = A,b = b,fM=100, scaling=FALSE)
round(sol1$Obj,4)
max(sol1$Iter[,1])
```

4 Results

4.1 Scaling = TRUE

The obtained results, both the number of iterations and the value of the objective function, were compared to those obtained using AMPL and two standard solvers such as Minos and CPLEX. Table 1 summarizes the data obtained from the three solutions. As it can be seen, we got the global optimal solution for two of the three linear programming problems. Notice, however, that our algorithm was not able to solve the third problem due to singularity problems of matrix ADA'. Actually, since the LP 598 problem did not satisfy the second assumption presented in the introduction of the document, the convergence were not guaranteed. Probably, due to a degenerate or close to degenerate value

of x, that is with less than m values different from 0, diagonal matrix D became singular or was very close to singularity at some iteration, making the algorithm return an error.

			Dimen		R affineScaling		Minos 5.51		CPLEX 12.8	
Problem	full	deg.	m	n	Iter	Opt. Value	Iter	Opt. Value	Iter	Opt. Value
LP 596	yes	yes	56	138	33	225495.2358	138	225495.1182	81	225495.1182
LP 597	yes	yes	27	51	17	-464.7530	10	-464.7531	7	-464.7531
LP 598	yes	yes	488	615	error	error	108	-35992829.29	92	-35992829.29

Table 1: Summary of the Results Obtained for each of the problems scaling

4.2 Scaling = FALSE

Without scaling, as Table 2 summarizes, the algorithm got stacked at some point unable to improve the value of the objective function and reached the maximum number of 500 iterations allowed in each problem.

			Dimen		R affineScaling		Minos 5.51		CPLEX 12.8	
Problem	full	deg.	m	n	Iter	Opt. Value	Iter	Opt. Value	Iter	Opt. Value
LP 596	yes	yes	56	138	501	311934.7446	138	225495.1182	81	225495.1182
LP 597	yes	yes	27	51	501	997.5175	10	-464.7531	7	-464.7531
LP 598	yes	yes	488	615	501	9368.3793	108	-35992829.29	92	-35992829.29

Table 2: Summary of the Results Obtained for each of the problems without scaling