# Algoritmo WW

adeandak, slabreu, jhottalengo

## Dynamic Lot Sizing Models with Walter-Within Algorithm

A continuación se encuentra el código realizado para encontrar los inventarios óptimos siguiendo el algoritmo de Walter-Within:

```
WW<-function(lambda, h, c=rep(0,length(lambda)), K){
  #inicio de la funcion
  n=length(lambda) #numero de periodos a considerar
  minAnt=0
  min=0
  CM=matrix(data = 0, nrow = n, ncol = n)
  #aux=matrix(data = 0, nrow = n, ncol = n)

  hc=matrix(data = 0, nrow = n, ncol = n)
  for(ren in 1:(n-1)){
    #aux[ren,ren]=min+K[ren]+c[ren]*lambda[ren]
    for(col in (ren+1):n){
      hc[ren,col]=hc[ren,(col-1)]+sum(h[ren:(col-1)])*lambda[col]
      #aux[ren,col]=hc[ren,col]+min+K[ren]+c[ren]*sum(lambda[ren:col])
    }
    #min=min(aux[1:ren:ren])
  }
  #aux[n,n]=min+K[n]+c[n]*lambda[n]

  for(i in 1:n){  #para recorrer los renglones
    for(j in i:n){ #para recorrer las columnas
      CM[i,j]=minAnt+K[i]+c[i]*sum(lambda[i:j])+hc[i,j]   #guardamos el costo en la matriz
    }
    minAnt=min(CM[1:i,i])  #guardamos el minimo de la columna anterior
  }

  #analisis
  i=n
  Q=rep(0,n)
  auxString<-""
  while(i>0){
    minI=which.min(CM[1:i,i])
    auxString<-paste(auxString,paste("Se pide en ",minI," para ",minI," a ",i),"\n")
    Q[minI]=sum(lambda[minI:i])
    i=minI-1
  }

  #para llenar los vectores necesarios para la ultima matriz
  l<-rep(0,n)
```

```
  for (k in 1:n) {
    if(Q[k]>0){
      l[k]=Q[k]-lambda[k]
    }else{
      l[k]=l[k-1]-lambda[k]
      K[k]=0
    }
  }
  h<-h*l
  cQ<-Q*c
  costo<-cQ+K+h
  matAnalysis<-rbind(lambda,Q,c,l,cQ,K,h,costo)

  #resultados
  cat(auxString)
  WW<-list("cost"=minAnt, "costMatrix"=CM, "analysis"=matAnalysis)
  return(WW)
}
```

Una vez que se implementó el algoritmo, se probó con los ejercicios analizados en clase:

```
lambda = c(60,100,10,200,120,15)
h = c(0.8,0.8,0.8,1.8,2,2)
c = c(5,5,5,5,5,5)
K = c(150,110,120,200,200,200)

WW1<-WW(lambda = lambda, c=c, h=h, K=K)
```

```
##  Se pide en  5  para  5  a  6
##  Se pide en  4  para  4  a  4
##  Se pide en  1  para  1  a  3
```

```
#El costo total sera de:
WW1$cost
```

```
## [1] 3201
```

```
#La matriz de costos es:
WW1$costMatrix
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  450 1030 1096 2576 3680 3848
## [2,]    0 1060 1118 2438 3446 3602
## [3,]    0    0 1200 2360 3272 3416
## [4,]    0    0    0 2296 3112 3244
## [5,]    0    0    0    0 3096 3201
## [6,]    0    0    0    0    0 3371
```

```
#La matriz con el analisis final es:
WW1$analysis
```

```
##        [,1] [,2] [,3] [,4] [,5] [,6]
## lambda   60  100   10  200  120   15
## Q       170    0    0  200  135    0
## c         5    5    5    5    5    5
## l       110   10    0    0   15    0
## cQ      850    0    0 1000  675    0
```

```
## K       150    0    0  200  200    0
## h        88    8    0    0   30    0
## costo  1088    8    0 1200  905    0
```

### Ejercicio de la presentacion

James, the manager of a local computer store, has the following requirements for a particular new brand computer: 100, 100, 50, 50, and 210, respectively. Becase the demand is lumpy James is not keen to use static lot sizing. To place an order, he pays $50 and estimates that holding a computer over a month will cost him $0.5

```
lambda2<-c(100,100,50,50,210)
K2<-rep(50,5)
h2<-rep(0.5,5)
WW2<-WW(lambda=lambda2, h=h2, K=K2)
```

```
##  Se pide en  5  para  5  a  5
##  Se pide en  2  para  2  a  4
##  Se pide en  1  para  1  a  1
```

```
WW2
```

```
## $cost
## [1] 225
##
## $costMatrix
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   50  100  150  225  645
## [2,]    0  100  125  175  490
## [3,]    0    0  150  175  385
## [4,]    0    0    0  175  280
## [5,]    0    0    0    0  225
##
## $analysis
##        [,1] [,2] [,3] [,4] [,5]
## lambda  100  100   50   50  210
## Q       100  200    0    0  210
## c         0    0    0    0    0
## l         0  100   50    0    0
## cQ        0    0    0    0    0
## K        50   50    0    0   50
## h         0   50   25    0    0
## costo    50  100   25    0   50
```