# Aljebra, laborategia R-en

Aitor Saiz Telleria

May 21, 2015

## 1 1. Praktika

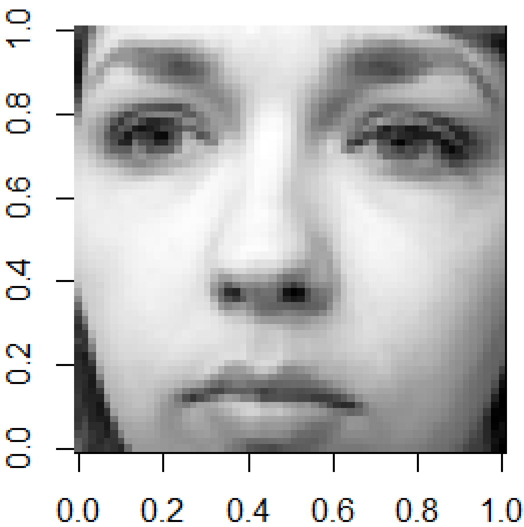
"Aztertu JPEG kodeketa duen irudi artxibo bat. R erabiliz, kalkulatu irudi txiki baten 2D Fourier transformatu diskretua (DFT), baita alderantzizko transformazioa ere. Koefiziente matrizea konprimatu eta berreraiki irudia konpresio maila desberdinak aztertuz."

Lehen praktikarako Fourier-en transformatuaren bitartez irudiak konprimatzen ikasi genuen, eta segidan konprimatutako artxibo hori bera deskonprimitzen. Gainera, probak egin genituen konpresio maila ezberdinekin.

Hasteko, "face.R" fitxategiko koefizienteak 60x60-ko matrize batean gorde genituen, koefiziente hauek irudi bat irudikatzen dutelarik 256 koloreko gris-en eskala bat erabiliz, eta matrize horri f_irudia deitu genion.

Irudi hau ikusteko proba egin genuen image funtzioari deitzen, horrela:

```
image(f_irudia,col=gray((0:255)/255))
```




Matrizea Fourier-en transformatuaren bitartez konprimatzeko, DFT2d funtzioa erabili genuen:

DFT2d:

```
F_irudia <- matrix(0, M, M)
```

```
for (u in 0:(M-1)) {
  for (v in 0:(M-1)) {
    for (x in 0:(M-1)) {
      for (y in 0:(M-1)) {
        F_irudia[u+1,v+1] <- F_irudia[u+1,v+1] + f_irudia[x+1,y+1] * exp(-2i*pi*(u*x+v*y)/
      }
    }
  }
}
u_koord <- ( v_koord <- 1:(M-1) )
```

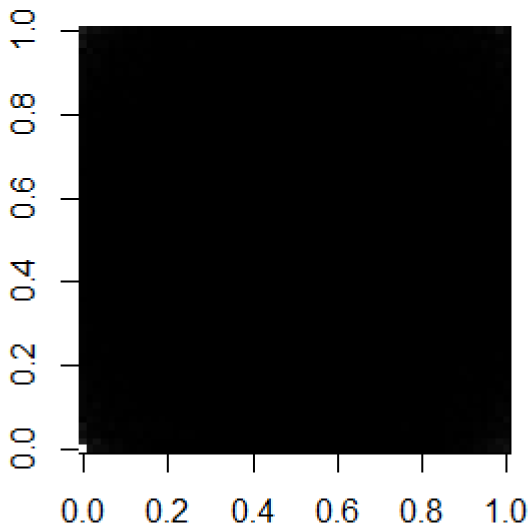
F_irudia matrize berriaren zati errealekin image funtzioari dei eginez gero ez dugu ezer antzemango irudian, baina atzera matrize hori deskonprimituz gero invDFT2d funtzioaren bidez, eta image funtzioarekin dei eginez, irudia berreskura dezakegu.

invDFT2d:

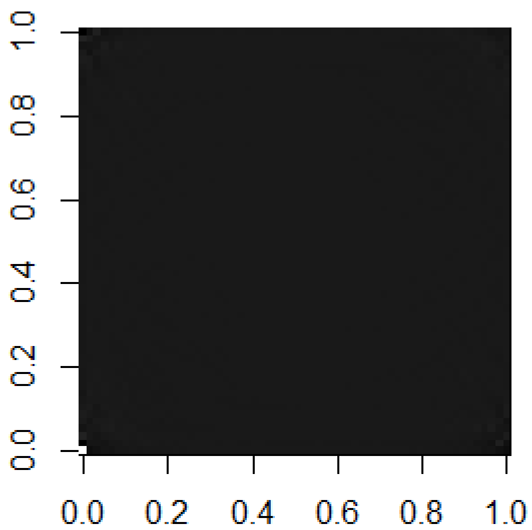
```
f_berria <- matrix(0, M, M)
for (x in 0:(M-1)) {
  for (y in 0:(M-1)) {
    for (u in 0:(M-1)) {
      for (v in 0:(M-1)) {
        f_berria[x+1,y+1] <- f_berria[x+1,y+1] + F_irudia[u+1,v+1] * exp(2i*pi*(u*x+v*y)/M
      }
    }
  }
}
f_berria <- f_berria/(M^2)
```

Honelakoa litzateke kodearen itxura:

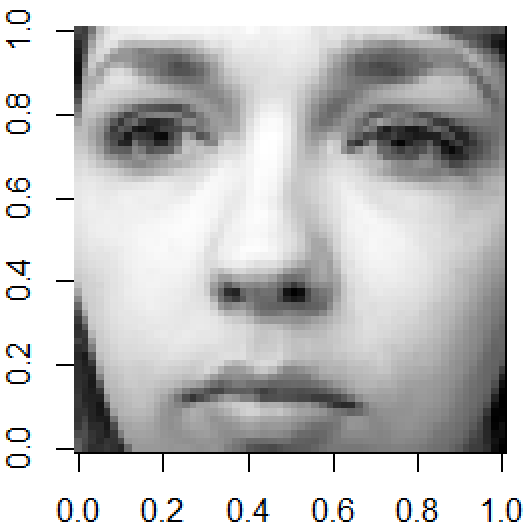
```
> source("face.R")
> M <- 60
> f_irudia <- matrix(face, M, M, byrow=T)
> image(f_irudia, col = gray((0:255)/255))
> source("DFT2d.R")
> image(Mod(F_irudia), col = gray((0:255)/255))
```




```
> image(Re(F_irudia), col = gray((0:255)/255))
```




```
> source("invDFT2d.R")
> image(Re(f_berria), col = gray((0:255)/255))
```




Ikus dezakegunez, hasierakoaren berdina lortu dugu.

```
Konpresio ezberdinak:
> konpr_tartea <- c(0:5, 55:59)
> source("konpresioa.R")
> image(Re(f_berria), col = gray((0:255)/255))
> konpr_tartea <- c(0:10, 50:59)
> source("konpresioa.R")
> image(Re(f_berria), col = gray((0:255)/255))
> konpr_tartea <- c(0:15, 45:59)
> source("konpresioa.R")
> image(Re(f_berria), col = gray((0:255)/255))
```

Azken parte hori konpresio maila ezberdinekin probak egiteko soilik da.

## 2 2. Praktika

"Kalkulatu plano geometrikoan bektore baten gaineko proiekzioari dagokion transformazio linealaren Tproj matrizea. Programatu R-ko lortuTproj() funtzioa, argumentu bezala b bektorea (vector motakoa) hartuko duena eta emaitza bezala Tproj itzuliko duena, b bektoreak definitutako zuzenaren gainerako proiekzioaren matrizea. Matrizea lortzeko hartu oinarri kanonikoaren unitate bektoreak eta hauen proiekzioak kalkulatu, ondoren transformazioaren matrizea eraikiz proiekzio hauek zutabeka ipinita.

Funtzioa definitu ondoren, hartu b bektore bat eta poligono itxi bat, adibidez triangelua, idatzi V matrizean erpinen koordenatuak zutabeka. Proiektatutako puntuen koordenatuak izango ditugu zutabeka lortuTproj(b) %*% V matrizean. R-ko grafikoan marraztu: triangelua, b bektorea, proiektatutako puntuak."

2. praktikako laborategian matrizeen eskalaketa, biraketa eta proiekzioak landu genituen transformazio linealei dagokienez, eta linealak ez direnei dagokienez, desplazamenduarekin jardun genuen.

Baina hauetatik proiekzioetan zentratu gara. Honen oinarrian biderketa eskalarraren printzipioa dago, honexegatik hain zuzen ere:

$(a.b) = (|a|.|b|).cos@$; beraz, $(a.b)/(|a|.|b|) = cos@$; $cos@$ =pr_luzera/$|a|$ denez, pr_luzera= $(a.b)/|b|$

Atal honetan egindako guztia horretan oinarritzen da.

Hasteko, b bektorea emanik Tproj matrizea itzuliko duen lortuTproj() funtzioa egin dugu, b bektoreak definitutako zuzenaren gainerako proiekzioaren matrizea hain zuzen ere.

Behin hori dakigunean, lortu dugun Tproj matrizea eta proiektatu nahi dugun arteko bektorearen biderketa matriziala egin beharko dugu, eta horrela lortuko ditugu proiekzioaren koordenatuak. Gainera, hasiera batean 2 dimentsioko bektoreekin soilik funtzionatzen zuen funtzioa inplementatu arren, moldatzea lortu dut edozein dimentsiotakoekin funtzionatzeko. Hona hemen biak:

2 dimentsioetarako soilik:

```
lortuTproj <- function(b)
lortuTproj <- function(b){

  e1 <- c(1,0);
  e2 <- c(0,1);
  e1_bidesk_b <- crossprod(e1,b);
  e1_luz <- sqrt(crossprod(e1,e1));
  b_luz <- sqrt(crossprod(b,b));
  kosinua_e1 <- as.vector(e1_bidesk_b/(e1_luz*b_luz));
  pr_e1_b_luz <- e1_bidesk_b/b_luz;
  pr_e1_b <- (b/b_luz)*pr_e1_b_luz;

  e2_bidesk_b <- crossprod(e2,b);
  e2_luz <- sqrt(crossprod(e2,e2));
  kosinua_e2 <- as.vector(e2_bidesk_b/(e2_luz*b_luz));
  pr_e2_b_luz <- e2_bidesk_b/b_luz;
  pr_e2_b <- (b/b_luz)*pr_e2_b_luz;

  Tproj <- cbind(pr_e1_b,pr_e2_b)
  return (Tproj);

}
```

Eta hona hemen beste bertsioa:

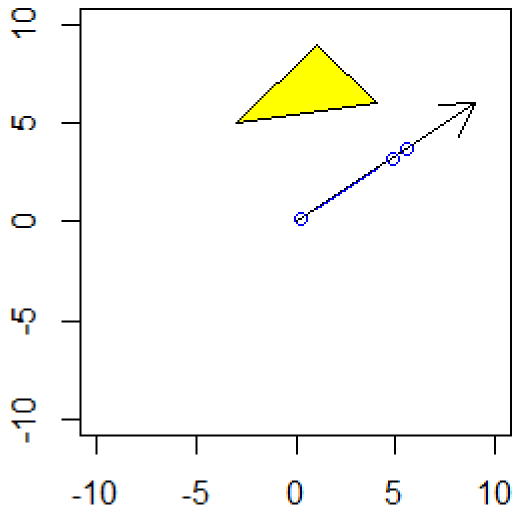
```
lortuTproj <- function(b)
{
  length<-length(b);
  D<-diag(length);
  b_luz<-sqrt(crossprod(b));

  for(i in 1:length)
  {
    c<-D[,i];
    c_bidesk_b<-crossprod(c,b);
```

```
    pr_luz<-c_bidesk_b/b_luz;
    r<-((b/b_luz)*pr_luz);
    D[,i]<-r;
  }

  return(D);
}
```

Orain, b bektore bat eta triangelu bat (triangeluaren koordenatuak V matrizean daude, zutabeka) zehaztu ditugu; segidan b bektorearen (irudian gezi moduan ageri dena) proiekzio matrizea lortu dugu lortuTproj() funtzioaren bitartez, eta Tproj%*%V egitean proiekzioaren koordenatuak lortu ditugu, irudian urdinez ageri direnak:




```
> x<-c(-3,1,4)
> y<-c(5,9,6)
> V<-rbind(x,y)
> V
  [,1] [,2] [,3]
x   -3    1    4
y    5    9    6
> polygon(t(V),col="yellow")
> b<-c(9,6)
> arrows(0,0,b[1],b[2])
> Tproj<-lortuTproj(b)
> Tproj
          [,1]      [,2]
[1,] 0.6923077 0.4615385
[2,] 0.4615385 0.3076923
> Tproj%*%V
          [,1]     [,2]     [,3]
[1,] 0.2307692 4.846154 5.538462
```

```
[2,] 0.1538462 3.230769 3.692308
> points(t(Tproj%*%V),type="b",col="blue")
```

# 3 3. Praktika

"Lehenik aztertu matrize simetrikoen diagonalizazioa eta ondoren, edozein matrizerako, SVD deskonposizioa. Aplikazio bezala hartu 1. praktikako irudi digitalaren matrizea, kalkulatu SVD deskonposizioa eta berreraiki irudia balio singular handienak hartuz, kopuru desberdinekin lortutako emaitzak erakutsi."

Matrizeen diagonalizazioari dagokionez, A matrize bat diagonalizagarria bada, D matrize diagonala existituko da, non $D = P^{-1} * A * P$ baita. Lehenik, proba batzuk egin ditugu autobalio eta autobektoreak erabiliz, eigen funtzioaren bitartez.

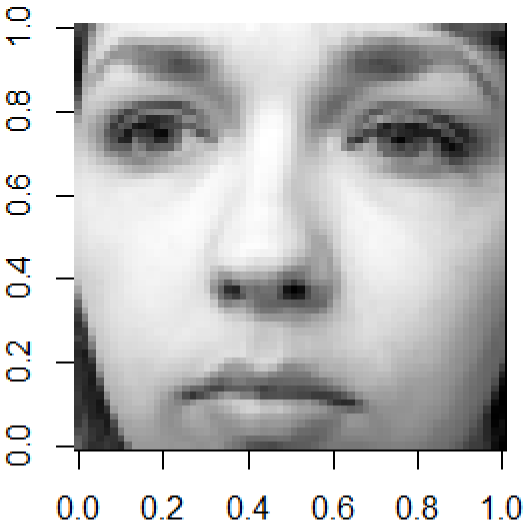
```
> A <- matrix(c(1,-4,3,7,-2,3),3,2)
> A
     [,1] [,2]
[1,]    1    7
[2,]   -4   -2
[3,]    3    3
> Matrizea <- A %*% t(A)
> Matrizea
     [,1] [,2] [,3]
[1,]   50  -18   24
[2,]  -18   20  -18
[3,]   24  -18   18
> eigendesk <- eigen(Matrizea, symmetric = T)
> eigendesk
$values
[1]  7.400000e+01  1.400000e+01 -8.881784e-16

$vectors
           [,1]       [,2]       [,3]
[1,]  0.7798129  0.5976143 -0.1864109
[2,] -0.4159002  0.7171372  0.5592328
[3,]  0.4678877 -0.3585686  0.8077807

> Matrizea %*% eigendesk$vec[,1]
          [,1]
[1,]  57.70615
[2,] -30.77661
[3,]  34.62369
> eigendesk$val[1] * eigendesk$vec[,1]
[1]  57.70615 -30.77661  34.62369
> round(t(eigendesk$vec)%*%eigendesk$vec,4)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

```
> eigendesk$vec %*% diag(eigendesk$val) %*% t(eigendesk$vec)
     [,1] [,2] [,3]
[1,]   50  -18   24
[2,]  -18   20  -18
[3,]   24  -18   18
> Matrizea
     [,1] [,2] [,3]
[1,]   50  -18   24
[2,]  -18   20  -18
[3,]   24  -18   18
```

Eta hauxe da praktika bera. Hasteko, lehen praktikako ”face.R” irudia dugu:



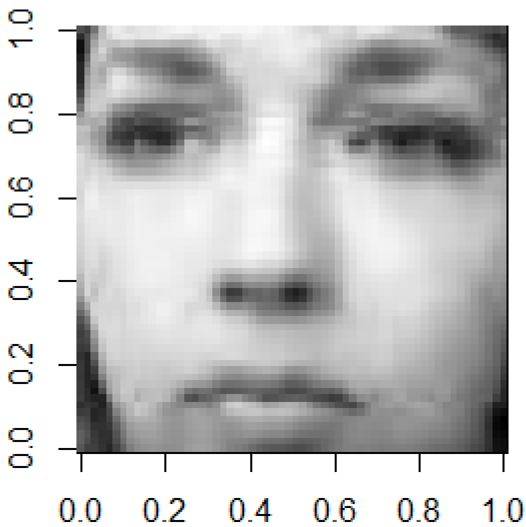

Irudi hau osatzen duen matrizearekin zera egin dugu, lehenengo SVD deskonposizioa aplikatu, eta gero balio singular ezberdinekin berreraikitzean lortutako emaitzak konparatu.

```
> source("Desktop/face.R")
> M<-60
> A<- matrix(face, M, M, byrow=T)
> image(A, col=gray((0:255)/255))
> ?svd
> svdesk <- svd(A)
> svdesk$d
 [1] 1.050705e+04 1.665270e+03 9.627892e+02 7.634238e+02 4.882529e+02 3.030881e+02 2.86541
 [9] 2.501197e+02 2.226648e+02 1.991354e+02 1.864365e+02 1.689899e+02 1.519633e+02 1.42173
[17] 1.227960e+02 1.074830e+02 8.719422e+01 7.843929e+01 7.052358e+01 6.699967e+01 6.54504
[25] 5.248613e+01 4.998368e+01 4.077626e+01 3.689776e+01 3.590459e+01 3.035925e+01 2.86101
[33] 2.280791e+01 2.205190e+01 1.797254e+01 1.777259e+01 1.535746e+01 1.416995e+01 1.23329
[41] 1.059476e+01 9.769924e+00 8.963786e+00 7.550862e+00 6.564840e+00 5.427355e+00 4.99030
[49] 4.260333e+00 3.671096e+00 3.249175e+00 2.813489e+00 2.187839e+00 1.998853e+00 1.35821
[57] 1.063737e+00 6.185763e-01 3.456576e-01 1.203431e-01
```

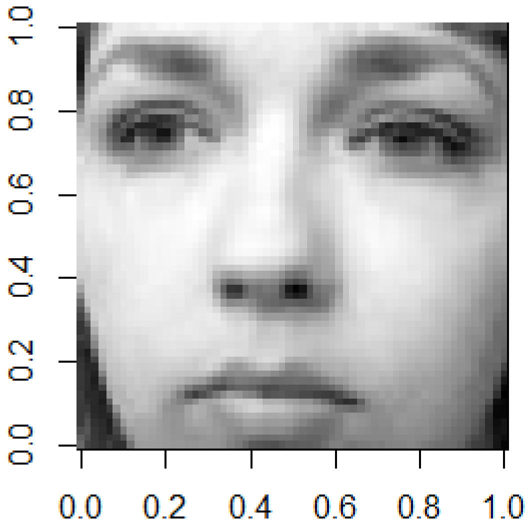
```
> D <- diag(svdesk$d)
> dim(D)
[1] 60 60
> U <- svdesk$u
> V<-svdesk$v
> dif <- A - U %*% D %*% t(V)
> max(abs(dif))
[1] 1.790568e-12
```

Hau da guk egin beharrekoa: N ezberdinekin egin probak eta alderatu emaitzak.

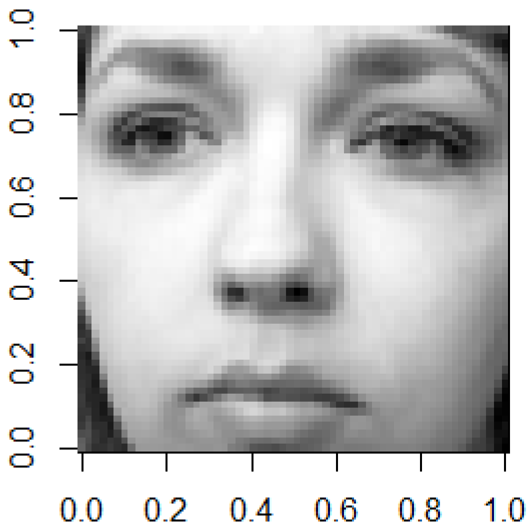
```
> N<-10
> A_ <- U[,1:N] %*% D[1:N,1:N] %*% t(V[,1:N])
> image(A_, col=gray((0:255)/255))
```



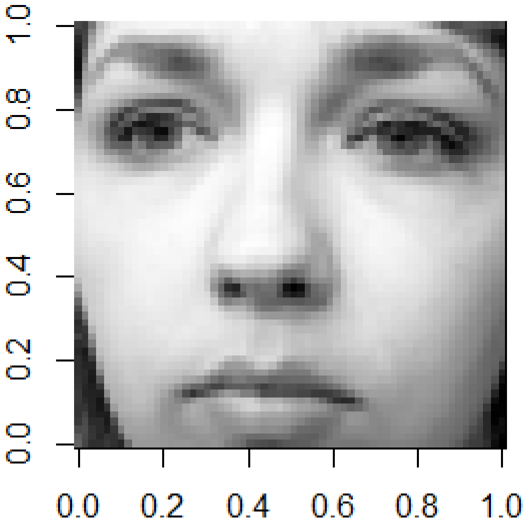

```
> N<-20
> A_ <- U[,1:N] %*% D[1:N,1:N] %*% t(V[,1:N])
> image(A_, col=gray((0:255)/255))
```




```
> N<-30
> A_ <- U[,1:N] %*% D[1:N,1:N] %*% t(V[,1:N])
> image(A_, col=gray((0:255)/255))
```




```
> N<-60
> A_ <- U[,1:N] %*% D[1:N,1:N] %*% t(V[,1:N])
> image(A_, col=gray((0:255)/255))
```




Ikus dezakegunez, azken hau orijinalaren berdin berdina da.