Homework 1 Solutions

You can think of vectors representing many dimensions of related information. For instance, Net ix might store all the ratings a user gives to movies in a vector. This is clearly a vector of very large dimensions (in the millions) and very sparse as the user might have rated only a few movies. Similarly, Amazon might store the items purchased by a user in a vector, with each slot or dimension representing a unique product and the value of the slot, the number of such items the user bought. One task that is frequently done in these settings is to find similarities between users. And, we can use dot-product between vectors to do just that. As you know, the dot-product is proportional to the length of two vectors and to the angle between them. In fact, the dot-product between two vectors, normalized by their lengths is called as the cosine distance and is frequently used in recommendation engines.

1. Calculate the dot product for u=[.5,.5] and v = [3, -4].

```
u=c(.5,.5)
v=c(3,4)
udotv=t(u)%*%v
udotv
```

```
##      [,1]
## [1,]  3.5
```

2. What are the lengths of u and v?

```
lenu=sqrt(t(u)%*%u)
lenv=sqrt(t(v)%*%v)
lenu
```

```
##           [,1]
## [1,] 0.7071068
```

```
lenv
```

```
##      [,1]
## [1,]    5
```

3. What is the linear combination of 3u-2v?

```
3*u-2*v
```

```
## [1] -4.5 -6.5
```

4. What is the angle between u and v?

```
costheta=udotv/(lenu*lenv)
acos(costheta)
```

```
##           [,1]
## [1,] 0.1418971
```

Next, you are asked to build a simple elimination program for a 3x3 LHS with no error checking. Here is a very quick and dirty solution.

```
myfun=function(A)
{
failure="singular matrix"  #check only for singularity
if(det(A[,1:3])==0){return(failure)}

else{


if(A[1,1]==0){     #check for zero on A[1,1]
    tmprow=A[1,]
    if(A[2,2]!=0) {
                A[1,]=A[2,]
                A[2,]=tmprow
               }
    else           {
               A[1,]=A[3,]
               A[3,]=tmprow}
}

for (i in 2:3){   #create first column of triangular matrix
if(A[i,1]!=0){
    tmp=A[1,1]/A[i,1]
    A[i,]=A[i,]*tmp
    A[i,]=A[i,]-A[1,]
           }
}

if(A[2,2]==0){    #ensure A[2,2]!=0
tmprow=A[2,]
A[2,]=A[3,]
A[3,]=tmprow
}

tmp=A[2,2]/A[3,2]  #finish upper triangular matrix
A[3,]=A[3,]*tmp
A[3,]=A[3,]-A[2,]

x1=A[3,4]/A[3,3]   #solve upper triangular matrix
x2=(A[2,4]-A[2,3]*x1)/A[2,2]
x3=(A[1,4]-A[1,3]*x1-A[1,2]*x2)/A[1,1]

X=c(x1,x2,x3)      #return solution
names(X)=c("x1","x2","x3")
return(X)
}
}

#build invertible matrix
a1=c(-1,-2,-2)  #build LHS example
a2=c(-5,5,5)
a3=c(0,2,-1)
```

```r
b=c(3,4,5)      #build RHS example

A=matrix(rbind(a1,a2,a3),nrow=3)
AUG=cbind(A,b)  #Augmented matrix
myfun(AUG)
```

```
##         x1         x2         x3
## -2.155556  1.422222 -1.533333
```

```r
#build singular matrix
c1=a1
c2=a1*2
c3=a3
A2=matrix(rbind(c1,c2,c3),nrow=3)
AUG2=cbind(A2,b)
myfun(AUG2)
```

```
## [1] "singular matrix"
```