Stat 154 HW 01

Mokssh Surve

2/3/2020

```r
library('ramify')
```

```
##
## Attaching package: 'ramify'

## The following object is masked from 'package:graphics':
##
##     clip
```

Problem 5

5a) - inner_prod()

```r
inner_prod <- function(a,b) {

  if(length(a) != length(b))
  { stop("Vector lengths must be the same") }

  prod <- as.numeric(t(a) %*% b)
  prod
}

y <- c(1, 1, 1, 1, 1)
z <- c(2, 4, 6, 8, 10)

inner_prod(y,y)
```

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

```r
inner_prod(z,z)
```

```
## [1] 220
```

```r
inner_prod(y,z)
```

```
## [1] 30
```

5b) - vnorm()

```r
vnorm <- function(a) {

  eu_norm <- sqrt(inner_prod(a,a))
  eu_norm
}

b <- c(1,2,3,4,5)
vnorm(b)
```

```
## [1] 7.416198
```

5c) - unit_norm()

```r
unit_norm <- function(a) {

  unit <- a / vnorm(a)
  unit
}

x <- c(1,2,3)
y <- c(1,1,1)
z <- c(1,0,0)

unit_norm(x)
```

```
## [1] 0.2672612 0.5345225 0.8017837
```

```r
unit_norm(y)
```

```
## [1] 0.5773503 0.5773503 0.5773503
```

```r
unit_norm(z)
```

```
## [1] 1 0 0
```

5d) - vector_proj()

```r
vector_proj <- function(a,b) {

  if(length(a) != length(b))
  { stop("Vector lengths must be the same") }

  vec_proj <- (inner_prod(a,b)/inner_prod(b,b)) * b
  vec_proj
}
```

```r
x <- c(1,2,3)
y <- c(1,1,1)

vector_proj(x,y)
```

```
## [1] 2 2 2
```

5e) - scalar_proj()

```r
scalar_proj <- function(a,b) {

  if(length(a) != length(b))
  { stop("Vector lengths must be the same") }

  scal_proj <- inner_prod(a,b) / vnorm(b)
  scal_proj
}

x <- c(1,2,3)
y <- c(1,1,1)

scalar_proj(x,y)
```

```
## [1] 3.464102
```

Problem 6

6a) Implementing Power Method

```r
power_method <- function(A, n) {

  # initialising the initial random vector
  rows <- nrow(A)
  w_0 <- randn(rows,1)

  w_old <- w_0

  for(k in 1:n)
  {
    w_new <- (A %*% w_old)
    s_new <- max(abs(w_new))
    w_new <- w_new / s_new

    # appending new column to matrix of approximated eigenvectors
    w_0 <- cbind(w_0, w_new)

    w_old <- w_new
  }
```

```r
  # converting vector so as to have unit norm
  w_new <- unit_norm(w_new)

  #creating list to output dominant eigen vector w_k+1 and eigen value
  out <- list(dom_vec=w_new, dom_value=s_new)
  out

}

A <- matrix(c(5,-4,3,-14,4,6,11,-4,-3), nrow=3, ncol=3)
power_method(A, 10)
```

```
## $dom_vec
##                [,1]
## [1,]  0.8943051739
## [2,] -0.4474571319
## [3,]  0.0006090899
##
## $dom_value
## [1] 11.97553
```

```r
eigen(A)
```

```
## eigen() decomposition
## $values
## [1]  1.200000e+01 -6.000000e+00  4.930713e-16
##
## $vectors
##                [,1]          [,2]       [,3]
## [1,] -8.944272e-01  7.071068e-01 -0.2672612
## [2,]  4.472136e-01  1.040834e-16  0.5345225
## [3,] -5.945103e-17 -7.071068e-01  0.8017837
```

```r
#eigenvalue given by Rayleigh Quotient (4)
vec <- power_method(A, 10)$dom_vec
rayleigh <- (t(vec) %*% A %*% vec) / (t(vec) %*% vec)
rayleigh
```

```
##          [,1]
## [1,] 12.01793
```

```r
#taking unit_norm() (5)
unit_norm(vec)
```

```
##              [,1]
## [1,]  0.89420343
## [2,] -0.44765945
## [3,]  0.00111546
```

```r
#checking if euclidean norm is 1
vnorm(vec)
```

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

From this, it can be seen that the dominant (largest) eigenvalues & eigenvectors are almost the same for both functions.

- **power_method()** outputs 12.05289
- **eigen() outputs** 12
- **Raleigh Quotient** gives us 11.97889

Note: The values (except for eigen()) might be a bit different since each time the function is run, a pseudo-random vector is created

6b) Other Scaling Options

```r
#A - Matrix
#n - no. of iterations
#p - L_p norm parameter
power_method_p <- function(A, n, p) {

  # initialising the initial random vector
  rows <- nrow(A)
  w_0 <- randn(rows,1)
  w_old <- w_0

  for(k in 1:n)
  {
    w_new <- (A %*% w_old)
    s_new <- (sum(abs(w_new))) ^(1/p)
    w_new <- w_new / s_new

    # appending new column to matrix of approximated eigenvectors
    w_0 <- cbind(w_0, w_new)

    w_old <- w_new
  }

  # converting vector so as to have unit norm
  w_new <- unit_norm(w_new)

  out <- list(dom_vec=w_new, dom_value=s_new)
  out

}

power_method_p(A, 20, 2)
```

```
## $dom_vec
##               [,1]
## [1,]  8.944271e-01
## [2,] -4.472138e-01
```

```
## [3,]  4.547727e-07
##
## $dom_value
## [1] 11.99994
```

```
eigen(A)
```

```
## eigen() decomposition
## $values
## [1]  1.200000e+01 -6.000000e+00  4.930713e-16
##
## $vectors
##                [,1]          [,2]       [,3]
## [1,] -8.944272e-01  7.071068e-01 -0.2672612
## [2,]  4.472136e-01  1.040834e-16  0.5345225
## [3,] -5.945103e-17 -7.071068e-01  0.8017837
```

From this, it can be confirmed that the dominant (largest) eigenvalues & eigenvectors are almost the same for both functions.

- **power_method()** outputs 11.99988
- **eigen()** outputs 12

Note: The values (except for eigen()) might be a bit different since each time the function is run, a pseudo-random vector is created

6c) Deflation & More eigenvectors

```
B <- cbind(c(5, 1), c(1, 5))

#applying Power Method to get first eigenvector & eigenvalue
v_1 <- power_method(B, 20)$dom_vec
lambda_1 <- power_method(B, 20)$dom_value


#deflating matrix B
B_1 <- B - lambda_1 * ((v_1) %*% t(v_1))

#getting the next eigenvector & eigenvalue
v_2 <- power_method(B_1, 20)$dom_vec
lambda_2 <- power_method(B_1, 20)$dom_value

#displaying first 2 eigenvectors
cbind(v_1, v_2)
```

```
##             [,1]       [,2]
## [1,] -0.7089504  0.7043321
## [2,] -0.7052584 -0.7098706
```

```
#displaying their respective eigenvalues
cbind(lambda_1, lambda_2)
```

```
##      lambda_1 lambda_2
## [1,] 5.998786  4.00002
```

```r
eigen(B)
```

```
## eigen() decomposition
## $values
## [1] 6 4
##
## $vectors
##           [,1]       [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

Thus, it can be confirmed, that Deflation works as eigen() gives the same eigenvalues & eigenvectors.