zad1

June 12, 2019

0.1 A library of operation of addition, scalar multiplication, dot product of vectors.

[1]:
```
vector.addition <- function(a, b){
    a + b
}

vector.scalar.multiplication <- function(a, b){
    a * b
}

vector.dot.product <- function(a, b){
    a %*% b
}
```

[2]:
```
vector.addition(c(1,1,1), c(1,1,1))
```

1. 2 2. 2 3. 2

[3]:
```
vector.scalar.multiplication(4, c(1,1,1,1))
```

1. 4 2. 4 3. 4 4. 4

[4]:
```
vector.dot.product(c(1,2,3), c(1,2,4))
```

17

0.2 The same but very simple

[5]:
```
addition <- function(a, b){
    if (length(a) != length(b)) 0
    my_list <- c()
    for (i in 1:length(a)) {
        my_list[i] <- a[i] + b[i]
    }
    my_list
}
```

[6]:
```
scalar.mult <- function(a, b) {
    result = c()
    for (i in 1:length(b)) {
        result[i] <- a * b[i]
    }
```

```
    result
}
```

[7]:
```
dot.product <- function(a, b) {
    result = 0
    for (i in 1:length(a)) {
        result <- result + a[i] * b[i]
    }
    result
}
```

[8]:
```
c1 <- c(1,2,3,4,5)
c2 <- c(1,1,1,1,1)
x <- addition(c1,c2); x
```

1. 2 2. 3 3. 4 4. 5 5. 6

[9]:
```
x <- scalar.mult(2, c1); x
```

1. 2 2. 4 3. 6 4. 8 5. 10

[10]:
```
x <- dot.product(c1,c2); x
```

15

0.3 A library of operation of addition, multiplication, transposition of matrices.

[11]:
```
m.add <- function(a, b){
    a + b
}
```

[12]:
```
m.mult <- function(a, b){
    a %*% b
}
```

[13]:
```
m.trans <- function(a){
    t(a)
}
```

0.4 The same but simple

[14]:
```r
matrix.addition <- function(a, b){
    long.c <- c()
    counter <- 1
    for (row in 1:nrow(a)) {
        for (col in 1:ncol(a)) {
            long.c[counter] <- a[row, col] + b[row, col]
            counter <- counter + 1
        }
    }
    matrix(long.c, nrow(a), ncol(a), TRUE)
}
```

[15]:
```r
matrix.mul <- function(a, b){
    if (!ncol(a) == nrow(b)) {
        0
    } else {
        result <- matrix(0, nrow(a), ncol(b))

        for (row in 1:nrow(a)) {

            for (b_col in 1:ncol(b)) {

                 for (col in 1:ncol(a)) {
                     result[row, b_col] <- result[row, b_col] + (a[row, col] *
 ↪b[col, b_col])
                }

            }

        }
        result
    }
}
```

[16]:
```r
matrix.transpose <- function(a){
    result = matrix(0, ncol(a), nrow(a))
    for (col in 1:ncol(a)){
        result[col,] = a[,col]
    }
    result
}
```

[17]:
```r
m1 <- matrix(c(1,2,3,4,5,6), 3); m1
m2 <- matrix(c(1,2,3,4,5,6), 3);

x <- matrix.addition(m1, m2); x
matrix.addition(m1, m2) == m.add(m1, m2)
```

```
x <- matrix.mul(m1, matrix.transpose(m2)); x
matrix.mul(m1, matrix.transpose(m2)) == m.mult(m1, m.trans(m2))
```

1	4
2	5
3	6
2	8
4	10
6	12

TRUE	TRUE
TRUE	TRUE
TRUE	TRUE

17	22	27
22	29	36
27	36	45

TRUE	TRUE	TRUE
TRUE	TRUE	TRUE
TRUE	TRUE	TRUE

0.5 Matricies for tests to gaussian elimination

[18]:
```
m <- matrix(c(3,1,2,6,4,4, 3,4,7), 3, 3);
m1 <- matrix(c(3,1,2,6,4,4,3,4,7,1,2,3), 3, 4);
m2 <- matrix(c(3,1,2,6,4,4,3,4,7,1,2,3), 4, 3);
m.zeros <- matrix(c(0,1,2,1,0,2,1,2,0), 3, 3)
print(m.zeros)
```

```
     [,1] [,2] [,3]
[1,]    0    1    1
[2,]    1    0    2
[3,]    2    2    0
```

0.6 A library of elementary column and row operations.

[19]:
```
#multiplies r-th row of M by factor
matrix.row.divide <- function( M, r, factor) {
    M[r,] = M[r,] * factor
    M #return
}
```

[20]:
```
#adds to i-th row tje j-th row multiplied by factor
matrix.add.raw <- function(i, j, factor, M) {
    M[i,] <- M[i,] + M[j,] * factor
    M # return
}
```

[21]:
```
# swap the i-th row with the j-th raw
matrix.swap.raw <- function(i, j, M) {
```

```
    tmp <- M[i,]
    M[i,] <- M[j,]
    M[j,] <- tmp
    M
}
```

[22]:
```
m.swaped <- matrix.swap.raw(1,2,m)
m
m.swaped
```

3	6	3
1	4	4
2	4	7

1	4	4
3	6	3
2	4	7

[23]:
```
#general Gauss row reduction algorithm
matrix.eliminate <- function(m){
    border = min(c(nrow(m), ncol(m)))
    for (ic in 1:border){
        # find the index of max absolute value in column and swap rows
        to.swap <- which(m[ic:nrow(m),ic] == max(abs(m[ic:nrow(m),ic]))) + ic -␣
 ↪1
        m <- matrix.swap.raw(ic, to.swap[1], m)
        # makes one on the diagonal
        m <- matrix.row.divide(m, ic, 1/m[ic,ic])
        if (ic + 1 > nrow(m)) break
        for (ir in (ic+1):nrow(m)){
            m <- matrix.add.raw(ir, ic,-m[ir,ic], m)
        }
    }
    m
}
```

0.7 Explain Gaussian method of computing an inverse of a matrix.

[24]:
```
matrix.solve <- function(m){
    # check if invertible
    if (det(m) == 0){
        0
    } else {
    # append identity matrix to m
    mI <- cbind(m, diag(nrow(m)))
    # gaussion elimination we get ones on diagonal
    mI <- matrix.eliminate(mI)
    for (i in nrow(mI):2) {
        for (j in (i-1):1) {
```

```
            mI <- matrix.add.raw(j, i, -mI[j,i], mI)
        }
    }
    mI[,(nrow(m) + 1):ncol(mI)]
    }
}
```

[25]:
```
print("m original")
m
me <- matrix.eliminate(m)
me

print("m1 original")
m1
m1e <- matrix.eliminate(m1)
m1e
print("m2 original")
m2
m2e <- matrix.eliminate(m2)
m2e

print("m.zeros original")
m.zeros
m.zeros.e <- matrix.eliminate(m.zeros)
m.zeros.e
```

```
[1] "m original"
```

3	6	3
1	4	4
2	4	7

1	2	1.0
0	1	1.5
0	0	1.0

```
[1] "m1 original"
```

3	6	3	1
1	4	4	2
2	4	7	3

1	2	1.0	0.3333333
0	1	1.5	0.8333333
0	0	1.0	0.4666667

```
[1] "m2 original"
```

3	4	7
1	4	1
2	3	2
6	4	3

1	0.6666667	0.50
0	1.0000000	0.15
0	0.0000000	1.00
0	0.0000000	0.00

[1] "m.zeros original"

0	1	1
1	0	2
2	2	0
1	1	0
0	1	1
0	0	1

[26]:
```
m
matrix.solve(m)
solve(m)
```

3	6	3
1	4	4
2	4	7

0.40000000	-1.0	0.4
0.03333333	0.5	-0.3
-0.13333333	0.0	0.2

0.40000000	-1.0	0.4
0.03333333	0.5	-0.3
-0.13333333	0.0	0.2

0.8 Explain Gaussian method of computing the rank of a matrix.

[27]:
```
matrix.rank <- function(m){
    border = min(c(nrow(m), ncol(m)))
    if (ncol(m) > nrow(m)) {
        m <- t(m)
    }
    rank <- border
    for (ic in 1:border){
        if (m[ic, ic] == 0){
            non.zero.index <- which(m[ic:nrow(m), ic] != 0) + ic -1
            # if exists non zero value in the column
            if(length(non.zero.index) > 0) {
                m <- matrix.swap.raw(non.zero.index[1], ic, m)
            } else {
                # there is no value different than zero
                # whole column is 0
                rank <- rank - 1
                next
            }
        }
```

```
        for (ir in 1:nrow(m)){
            if (ir != ic){
                m <- matrix.add.raw(ir, ic, -m[ir, ic]/m[ic,ic], m)
            }
        }
    }
    rank
}
```

[28]:
```
m <- matrix(c(0,0,0,1,2,3,2,3,0,1,1,1,10,8,5,3,4,5),6)
m
matrix.rank(m)
qr(m)$rank
matrix.rank(m1) == qr(m1)$rank
matrix.rank(m2) == qr(m2)$rank
matrix.rank(m.zeros) == qr(m.zeros)$rank
```

0	2	10
0	3	8
0	0	5
1	1	3
2	1	4
3	1	5

3
3
TRUE
TRUE
TRUE

0.9 Propose a method of calculation of a dimension of linear span $Span(A)$ of given finite set of vectors ARn

We need to find all lineary independent vectors from A, which will be the base of $Span(A)$. Cardinality of this base will be dimension of $Span(A)$. Vectors must be lineary independent not just non collinear. Number of lineary independent vectors from A is the definiotion of $rank(A)$. We just need to compute rank on the given matrix A.