**Category**

* [Data Management](https://datascienceplus.com/category/data-management/)

**Tags**

* [Data Manipulation](https://datascienceplus.com/tag/data-manipulation/)
* [Linear Regression](https://datascienceplus.com/tag/linear-regression/)
* [R Programming](https://datascienceplus.com/tag/rstats/)
* [Tips & Tricks](https://datascienceplus.com/tag/tips-tricks/)

In this article, you learn how to do linear algebra for Machine Learning and Deep Learning in R. In particular, I will discuss: Matrix Multiplication, Solve System of Linear Equations, Identity Matrix, Matrix Inverse, Solve System of Linear Equations Revisited, Finding the Determinant, Matrix Norm, Frobenius Norm, Special Matrices and Vectors, Eigendecomposition, Singular Value Decomposition, Moore-Penrose Pseudoinverse, and Matrix Trace.

**Introduction**

Linear algebra is a branch of mathematics that is widely used throughout data science. Yet because linear algebra is a form of continuous rather than discrete mathematics, many data scientists have little experience with it. A good understanding of linear algebra is essential for understanding and working with machine learning and deep learning algorithms. This article is particularly aimed at linear algebra for these two econometrical/statistical disciplines. Let us dive into the world of linear algebra for machine learning and deep learning with R:

**Matrix Multiplication**

Let us start by creating a Matrix Multiplication in R:

A <- matrix(data = 1:36, nrow = 6)

A

*[,1] [,2] [,3] [,4] [,5] [,6]*

*[1,] 1 7 13 19 25 31*

*[2,] 2 8 14 20 26 32*

*[3,] 3 9 15 21 27 33*

*[4,] 4 10 16 22 28 34*

*[5,] 5 11 17 23 29 35*

*[6,] 6 12 18 24 30 36*

B <- matrix(data = 1:30, nrow = 6)

B

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1 7 13 19 25*

*[2,] 2 8 14 20 26*

*[3,] 3 9 15 21 27*

*[4,] 4 10 16 22 28*

*[5,] 5 11 17 23 29*

*[6,] 6 12 18 24 30*

A %\*% B

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 441 1017 1593 2169 2745*

*[2,] 462 1074 1686 2298 2910*

*[3,] 483 1131 1779 2427 3075*

*[4,] 504 1188 1872 2556 3240*

*[5,] 525 1245 1965 2685 3405*

*[6,] 546 1302 2058 2814 3570*

**Hadamard Multiplication**

Let us try creating a Hadamard Multiplication in R:

A <- matrix(data = 1:36, nrow = 6)

A

*[,1] [,2] [,3] [,4] [,5] [,6]*

*[1,] 1 7 13 19 25 31*

*[2,] 2 8 14 20 26 32*

*[3,] 3 9 15 21 27 33*

*[4,] 4 10 16 22 28 34*

*[5,] 5 11 17 23 29 35*

*[6,] 6 12 18 24 30 36*

B <- matrix(data = 11:46, nrow = 6)

*B*

*[,1] [,2] [,3] [,4] [,5] [,6]*

*[1,] 11 17 23 29 35 41*

*[2,] 12 18 24 30 36 42*

*[3,] 13 19 25 31 37 43*

*[4,] 14 20 26 32 38 44*

*[5,] 15 21 27 33 39 45*

*[6,] 16 22 28 34 40 46*

A \* B

*[,1] [,2] [,3] [,4] [,5] [,6]*

*[1,] 11 119 299 551 875 1271*

*[2,] 24 144 336 600 936 1344*

*[3,] 39 171 375 651 999 1419*

*[4,] 56 200 416 704 1064 1496*

*[5,] 75 231 459 759 1131 1575*

*[6,] 96 264 504 816 1200 1656*

**Dot Product**

Let us now create Dot Product in R:

X <- matrix(data = 1:10, nrow = 10)

X

*[,1]*

*[1,] 1*

*[2,] 2*

*[3,] 3*

*[4,] 4*

*[5,] 5*

*[6,] 6*

*[7,] 7*

*[8,] 8*

*[9,] 9*

*[10,] 10*

Y <- matrix(data = 11:20, nrow = 10)

Y

*[,1]*

*[1,] 11*

*[2,] 12*

*[3,] 13*

*[4,] 14*

*[5,] 15*

*[6,] 16*

*[7,] 17*

*[8,] 18*

*[9,] 19*

*[10,] 20*

Let us now create Dot Product function in R:

dotProduct <- function(X, Y) {

as.vector(t(X) %\*% Y)

}

dotProduct(X, Y)

*[1] 935*

**Properties of Matrix Multiplication**

Let us look at Properties of Matrix Multiplication in R:

#1 Matrix Property: It is Distributive Matrix

*A <- matrix(data = 1:25, nrow = 5)*

*B <- matrix(data = 26:50, nrow = 5)*

*C <- matrix(data = 51:75, nrow = 5)*

A %\*% (B + C)

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 4555 5105 5655 6205 6755*

*[2,] 4960 5560 6160 6760 7360*

*[3,] 5365 6015 6665 7315 7965*

*[4,] 5770 6470 7170 7870 8570*

*[5,] 6175 6925 7675 8425 9175*

A %\*% B + A %\*% C

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 4555 5105 5655 6205 6755*

*[2,] 4960 5560 6160 6760 7360*

*[3,] 5365 6015 6665 7315 7965*

*[4,] 5770 6470 7170 7870 8570*

*[5,] 6175 6925 7675 8425 9175*

#2 Matrix Property: It is Associative Matrix

A <- matrix(data = 1:25, nrow = 5)

B <- matrix(data = 26:50, nrow = 5)

C <- matrix(data = 51:75, nrow = 5)

A %\*% B) %\*% C

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 569850 623350 676850 730350 783850*

*[2,] 620450 678700 736950 795200 853450*

*[3,] 671050 734050 797050 860050 923050*

*[4,] 721650 789400 857150 924900 992650*

*[5,] 772250 844750 917250 989750 1062250*

A %\*% (B %\*% C)

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 569850 623350 676850 730350 783850*

*[2,] 620450 678700 736950 795200 853450*

*[3,] 671050 734050 797050 860050 923050*

*[4,] 721650 789400 857150 924900 992650*

*[5,] 772250 844750 917250 989750 1062250*

#3 Matrix Property: It is Not Commutative Matrix

A <- matrix(data = 1:25, nrow = 5)

B <- matrix(data = 26:50, nrow = 5)

A %\*% B

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1590 1865 2140 2415 2690*

*[2,] 1730 2030 2330 2630 2930*

*[3,] 1870 2195 2520 2845 3170*

*[4,] 2010 2360 2710 3060 3410*

*[5,] 2150 2525 2900 3275 3650*

B %\*% A

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 590 1490 2390 3290 4190*

*[2,] 605 1530 2455 3380 4305*

*[3,] 620 1570 2520 3470 4420*

*[4,] 635 1610 2585 3560 4535*

*[5,] 650 1650 2650 3650 4650*

**Matrix Transpose**

Let us look at Matrix Transpose in R:

A <- matrix(data = 1:25, nrow = 5, ncol = 5, byrow = TRUE)

A

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1 2 3 4 5*

*[2,] 6 7 8 9 10*

*[3,] 11 12 13 14 15*

*[4,] 16 17 18 19 20*

*[5,] 21 22 23 24 25*

t(A)

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1 6 11 16 21*

*[2,] 2 7 12 17 22*

*[3,] 3 8 13 18 23*

*[4,] 4 9 14 19 24*

*[5,] 5 10 15 20 25*

Let us look at Matrix Transpose Property in R:

A <- matrix(data = 1:25, nrow = 5)

B <- matrix(data = 25:49, nrow = 5)

t(A %\*% B)

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1535 1670 1805 1940 2075*

*[2,] 1810 1970 2130 2290 2450*

*[3,] 2085 2270 2455 2640 2825*

*[4,] 2360 2570 2780 2990 3200*

*[5,] 2635 2870 3105 3340 3575*

t(B) %\*% t(A)

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1535 1670 1805 1940 2075*

*[2,] 1810 1970 2130 2290 2450*

*[3,] 2085 2270 2455 2640 2825*

*[4,] 2360 2570 2780 2990 3200*

*[5,] 2635 2870 3105 3340 3575*

**Solve System of Linear Equations**

Now let us Solve System of Linear Equations in R:

Ax = B

A <- matrix(data = c(1, 3, 2, 4, 2, 4, 3, 5, 1, 6, 7, 2, 1, 5, 6, 7), nrow = 4, byrow = TRUE)

A

*[,1] [,2] [,3] [,4]*

*[1,] 1 3 2 4*

*[2,] 2 4 3 5*

*[3,] 1 6 7 2*

*[4,] 1 5 6 7*

B <- matrix(data = c(1, 2, 3, 4), nrow = 4)

B

*[,1]*

*[1,] 1*

*[2,] 2*

*[3,] 3*

*[4,] 4*

solve(a = A, b = B)

*[,1]*

*[1,] 0.6153846*

*[2,] -0.8461538*

*[3,] 1.0000000*

*[4,] 0.2307692*

**Identity Matrix**

Now let us Solve Identity Matrix in R:

I <- diag(x = 1, nrow = 5, ncol = 5)

I

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1 0 0 0 0*

*[2,] 0 1 0 0 0*

*[3,] 0 0 1 0 0*

*[4,] 0 0 0 1 0*

*[5,] 0 0 0 0 1*

A <- matrix(data = 1:25, nrow = 5)

A %\*% I

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1 6 11 16 21*

*[2,] 2 7 12 17 22*

*[3,] 3 8 13 18 23*

*[4,] 4 9 14 19 24*

*[5,] 5 10 15 20 25*

I %\*% A

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1 6 11 16 21*

*[2,] 2 7 12 17 22*

*[3,] 3 8 13 18 23*

*[4,] 4 9 14 19 24*

*[5,] 5 10 15 20 25*

**Matrix Inverse**

Now let us Solve Matrix Inverse in R:

A <- matrix(data = c(1, 2, 3, 1, 2, 3, 4, 5, 6, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 3), nrow = 5)

A

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1 3 3 8 4*

*[2,] 2 4 4 9 5*

*[3,] 3 5 5 1 6*

*[4,] 1 6 6 2 7*

*[5,] 2 2 7 3 3*

library(MASS)

ginv(A)

*[,1] [,2] [,3] [,4] [,5]*

*[1,] -0.3333333 0.3333333 0.3333333 -3.333333e-01 1.040834e-16*

*[2,] -4.0888889 3.6444444 -1.2222222 8.666667e-01 -2.000000e-01*

*[3,] -0.3555556 0.2444444 -0.2222222 1.333333e-01 2.000000e-01*

*[4,] -0.1111111 0.2222222 -0.1111111 -6.938894e-18 2.602085e-18*

*[5,] 3.8888889 -3.4444444 1.2222222 -6.666667e-01 -2.664535e-15*

ginv(A) %\*% A

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1.000000e+00 -6.800116e-16 -1.595946e-16 9.020562e-17 -1.020017e-15*

*[2,] 8.881784e-16 1.000000e+00 -5.329071e-15 -1.287859e-14 -2.464695e-14*

*[3,] -1.665335e-16 -1.387779e-15 1.000000e+00 -1.332268e-15 -1.998401e-15*

*[4,] -2.237793e-16 -8.135853e-16 -8.005749e-16 1.000000e+00 -1.262011e-15*

*[5,] 0.000000e+00 1.953993e-14 6.217249e-15 1.265654e-14 1.000000e+00*

A %\*% ginv(A)

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1.000000e+00 1.776357e-15 -1.776357e-15 2.220446e-15 -1.200429e-15*

*[2,] -7.105427e-15 1.000000e+00 -1.776357e-15 1.776357e-15 -5.316927e-16*

*[3,] -3.552714e-15 0.000000e+00 1.000000e+00 1.776357e-15 1.136244e-16*

*[4,] 0.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00 5.204170e-18*

*[5,] -5.329071e-15 5.329071e-15 -8.881784e-16 1.998401e-15 1.000000e+00*

**Solve System of Linear Equations Revisited**

Let us look at Solve System of Linear Equations Revisited in R:

A <- matrix(data = c(1, 3, 2, 4, 2, 4, 3, 5, 1, 6, 7, 2, 1, 5, 6, 7), nrow = 4, byrow = TRUE)

A

*[,1] [,2] [,3] [,4]*

*[1,] 1 3 2 4*

*[2,] 2 4 3 5*

*[3,] 1 6 7 2*

*[4,] 1 5 6 7*

B <- matrix(data = c(1, 2, 3, 4), nrow = 4)

B

*[,1]*

*[1,] 1*

*[2,] 2*

*[3,] 3*

*[4,] 4*

library(MASS)

X <- ginv(A) %\*% B

X

*[,1]*

*[1,] 0.6153846*

*[2,] -0.8461538*

*[3,] 1.0000000*

*[4,] 0.2307692*

**Determinant**

Let us find the Determinant Matrix in R:

A <- matrix(data = c(1, 3, 2, 4, 2, 4, 3, 5, 1, 6, 7, 2, 1, 5, 6, 7), nrow = 4, byrow = TRUE)

A

*[,1] [,2] [,3] [,4]*

*[1,] 1 3 2 4*

*[2,] 2 4 3 5*

*[3,] 1 6 7 2*

*[4,] 1 5 6 7*

det(A)

*[1] -39*

**Matrix Norm**

Let us now look at the Matrix Norm:

lpNorm = 1 & dim(A)[[2]] == 1 && is.infinite(p) == FALSE) {

sum((apply(X = A, MARGIN = 1, FUN = abs)) \*\* p) \*\* (1 / p)

} else if (p >= 1 & dim(A)[[2]] == 1 & is.infinite(p)) {

max(apply(X = A, MARGIN = 1, FUN = abs)) Max Norm

} else {

invisible(NULL)

}

}

lpNorm(A = matrix(data = 1:10), p = 1)

*[1] 55*

lpNorm(A = matrix(data = 1:10), p = 2)

#Euclidean Distance

*[1] 19.62142*

lpNorm(A = matrix(data = 1:10), p = 3)

*[1] 14.46245*

lpNorm(A = matrix(data = -100:10), p = Inf)

*[1] 100*

Let us find Properties between Matrix Norm and the Determinant Matrix in R:

lpNorm(A = matrix(data = rep(0, 10)), p = 1) == 0

*[1] TRUE*

lpNorm(A = matrix(data = 1:10) + matrix(data = 11:20), p = 1) <= lpNorm(A = matrix(data = 1:10), p = 1) + lpNorm(A = matrix(data = 11:20), p = 1)

*[1] TRUE*

tempFunc <- function(i) {

lpNorm(A = i \* matrix(data = 1:10), p = 1) == abs(i) \* lpNorm(A = matrix(data = 1:10), p = 1)

}

all(sapply(X = -10:10, FUN = tempFunc))

*[1] TRUE*

**Frobenius Norm**

The Frobenius norm, sometimes also called the Euclidean norm (a term unfortunately also used for the vector L^2-norm), is matrix norm of an m×n matrix A defined as the square root of the sum of the absolute squares of its elements. The Frobenius norm is the only one out of the above three matrix norms that is unitary invariant, i.e., it is conserved or invariant under a unitary transformation.  
Let us find Frobenius Norm Matrix in R:

frobeniusNorm <- function(A) {

(sum((as.numeric(A)) \*\* 2)) \*\* (1 / 2)

}

frobeniusNorm(A = matrix(data = 1:25, nrow = 5))

*[1] 74.33034*

**Special Matrices and Vectors**

Let us look at Special Matrices and Vectors in R:

#1 Special Matrix: The Diagonal Matrix

A <- diag(x = c(1:5, 6, 1, 2, 3, 4), nrow = 10)

*A*

*[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]*

*[1,] 1 0 0 0 0 0 0 0 0 0*

*[2,] 0 2 0 0 0 0 0 0 0 0*

*[3,] 0 0 3 0 0 0 0 0 0 0*

*[4,] 0 0 0 4 0 0 0 0 0 0*

*[5,] 0 0 0 0 5 0 0 0 0 0*

*[6,] 0 0 0 0 0 6 0 0 0 0*

*[7,] 0 0 0 0 0 0 1 0 0 0*

*[8,] 0 0 0 0 0 0 0 2 0 0*

*[9,] 0 0 0 0 0 0 0 0 3 0*

*[10,] 0 0 0 0 0 0 0 0 0 4*

X <- matrix(data = 21:30)

X

*[,1]*

*[1,] 21*

*[2,] 22*

*[3,] 23*

*[4,] 24*

*[5,] 25*

*[6,] 26*

*[7,] 27*

*[8,] 28*

*[9,] 29*

*[10,] 30*

A %\*% X

*[,1]*

*[1,] 21*

*[2,] 44*

*[3,] 69*

*[4,] 96*

*[5,] 125*

*[6,] 156*

*[7,] 27*

*[8,] 56*

*[9,] 87*

*[10,] 120*

library(MASS)

ginv(A)

*[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]*

*[1,] 1 0.0 0.0000000 0.00 0.0 0.0000000 0 0.0 0.0000000 0.00*

*[2,] 0 0.5 0.0000000 0.00 0.0 0.0000000 0 0.0 0.0000000 0.00*

*[3,] 0 0.0 0.3333333 0.00 0.0 0.0000000 0 0.0 0.0000000 0.00*

*[4,] 0 0.0 0.0000000 0.25 0.0 0.0000000 0 0.0 0.0000000 0.00*

*[5,] 0 0.0 0.0000000 0.00 0.2 0.0000000 0 0.0 0.0000000 0.00*

*[6,] 0 0.0 0.0000000 0.00 0.0 0.1666667 0 0.0 0.0000000 0.00*

*[7,] 0 0.0 0.0000000 0.00 0.0 0.0000000 1 0.0 0.0000000 0.00*

*[8,] 0 0.0 0.0000000 0.00 0.0 0.0000000 0 0.5 0.0000000 0.00*

*[9,] 0 0.0 0.0000000 0.00 0.0 0.0000000 0 0.0 0.3333333 0.00*

*[10,] 0 0.0 0.0000000 0.00 0.0 0.0000000 0 0.0 0.0000000 0.25*

#2 Special Matrix: The Symmetric Matrix

A <- matrix(data = c(1, 2, 2, 1), nrow = 2)

A

*[,1] [,2]*

*[1,] 1 2*

*[2,] 2 1*

all(A == t(A))

*[1] TRUE*

#3 Special Matrix: The Unit Vector

lpNorm(A = matrix(data = c(1, 0, 0, 0)), p = 2)

*[1] 1*

#4 Special Matrix: Orthogonal Vectors

X <- matrix(data = c(11, 0, 0, 0))

Y <- matrix(data = c(0, 11, 0, 0))

all(t(X) %\*% Y == 0)

*[1] TRUE*

#4 Special Matrix: More Orthogonal Vectors

X <- matrix(data = c(1, 0, 0, 0))

Y <- matrix(data = c(0, 1, 0, 0))

lpNorm(A = X, p = 2) == 1

*[1] TRUE*

lpNorm(A = Y, p = 2) == 1

*[1] TRUE*

all(t(X) %\*% Y == 0)

*[1] TRUE*

#4 Special Matrix: Still More Orthogonal Vectors

A <- matrix(data = c(1, 0, 0, 0, 1, 0, 0, 0, 1), nrow = 3, byrow = TRUE)

A

*[,1] [,2] [,3]*

*[1,] 1 0 0*

*[2,] 0 1 0*

*[3,] 0 0 1*

all(t(A) %\*% A == A %\*% t(A))

*[1] TRUE*

all(t(A) %\*% A == diag(x = 1, nrow = 3))

*[1] TRUE*

library(MASS)

all(t(A) == ginv(A))

*[1] TRUE*

**Eigendecomposition**

Let us Now look at Eigendecomposition in R:

A <- matrix(data = 1:25, nrow = 5, byrow = TRUE)

A

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1 2 3 4 5*

*[2,] 6 7 8 9 10*

*[3,] 11 12 13 14 15*

*[4,] 16 17 18 19 20*

*[5,] 21 22 23 24 25*

y <- eigen(x = A)

library(MASS)

all.equal(y$vectors %\*% diag(y$values) %\*% ginv(y$vectors), A)

*[1] TRUE*

**Singular Value Decomposition**

Now let us look at Singular Value Decomposition in R:

A <- matrix(data = 1:36, nrow = 6, byrow = TRUE)

A

*[,1] [,2] [,3] [,4] [,5] [,6]*

*[1,] 1 2 3 4 5 6*

*[2,] 7 8 9 10 11 12*

*[3,] 13 14 15 16 17 18*

*[4,] 19 20 21 22 23 24*

*[5,] 25 26 27 28 29 30*

*[6,] 31 32 33 34 35 36*

y <- svd(x = A)

y

*$d*

*[1] 1.272064e+02 4.952580e+00 1.068280e-14 3.258502e-15 9.240498e-16*

*[6] 6.865073e-16*

*$u*

*[,1] [,2] [,3] [,4] [,5]*

*[1,] -0.06954892 -0.72039744 0.6716423 -0.11924367 0.08965916*

*[2,] -0.18479698 -0.51096788 -0.6087484 -0.06762569 0.44007566*

*[3,] -0.30004504 -0.30153832 -0.3722328 -0.09266448 -0.41109295*

*[4,] -0.41529310 -0.09210875 0.0313011 0.21692481 -0.67511264*

*[5,] -0.53054116 0.11732081 0.1308779 0.71086492 0.37490569*

*[6,] -0.64578922 0.32675037 0.1471598 -0.64825589 0.18156508*

*[,6]*

*[1,] -0.05319067*

*[2,] 0.36871061*

*[3,] -0.70915885*

*[4,] 0.56145739*

*[5,] -0.20432734*

*[6,] 0.03650885*

*$v*

*[,1] [,2] [,3] [,4] [,5] [,6]*

*[1,] -0.3650545 0.62493577 0.54215504 0.08199306 -0.1033873 -0.4060131*

*[2,] -0.3819249 0.38648609 -0.23874067 -0.40371901 0.5758949 0.3913066*

*[3,] -0.3987952 0.14803642 -0.75665994 0.29137287 -0.2722608 -0.2957858*

*[4,] -0.4156655 -0.09041326 0.14938782 -0.18121587 -0.6652579 0.5668542*

*[5,] -0.4325358 -0.32886294 0.21539167 0.69322385 0.3606549 0.2184882*

*[6,] -0.4494062 -0.56731262 0.08846608 -0.48165491 0.1043562 -0.4748500*

all.equal(y$u %\*% diag(y$d) %\*% t(y$v), A)

*[1] TRUE*

**Moore-Penrose Pseudoinverse**

Now let us look at Moore-Penrose Pseudoinverse in R:

A <- matrix(data = 1:25, nrow = 5)

A

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1 6 11 16 21*

*[2,] 2 7 12 17 22*

*[3,] 3 8 13 18 23*

*[4,] 4 9 14 19 24*

*[5,] 5 10 15 20 25*

B <- ginv(A)

B

*[,1] [,2] [,3] [,4] [,5]*

*[1,] -0.152 -0.08 -8.00000e-03 0.064 0.136*

*[2,] -0.096 -0.05 -4.00000e-03 0.042 0.088*

*[3,] -0.040 -0.02 -9.97466e-18 0.020 0.040*

*[4,] 0.016 0.01 4.00000e-03 -0.002 -0.008*

*[5,] 0.072 0.04 8.00000e-03 -0.024 -0.056*

y <- svd(A)

all.equal(y$v %\*% ginv(diag(y$d)) %\*% t(y$u), B)

*[1] TRUE*

**Trace Matrix in R**

Lastly, let us look at Trace Matrix in R:

A <- diag(x = 1:10)

A

*[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]*

*[1,] 1 0 0 0 0 0 0 0 0 0*

*[2,] 0 2 0 0 0 0 0 0 0 0*

*[3,] 0 0 3 0 0 0 0 0 0 0*

*[4,] 0 0 0 4 0 0 0 0 0 0*

*[5,] 0 0 0 0 5 0 0 0 0 0*

*[6,] 0 0 0 0 0 6 0 0 0 0*

*[7,] 0 0 0 0 0 0 7 0 0 0*

*[8,] 0 0 0 0 0 0 0 8 0 0*

*[9,] 0 0 0 0 0 0 0 0 9 0*

*[10,] 0 0 0 0 0 0 0 0 0 10*

library(psych)

tr(A)

*[1] 55*

We can also code a function that calculates the Trace Matrix of the Frobenius Norm Matrix:

alternativeFrobeniusNorm <- function(A) {

sqrt(tr(t(A) %\*% A))

}

alternativeFrobeniusNorm(A)

*[1] 19.62142*

frobeniusNorm(A)

*[1] 19.62142*

all.equal(tr(A), tr(t(A)))

*[1] TRUE*

Let us look at the diagonals (1:5) of the Trace Matrix of the Frobenius Norm Trace Matrix:

A <- diag(x = 1:5)

A

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 1 0 0 0 0*

*[2,] 0 2 0 0 0*

*[3,] 0 0 3 0 0*

*[4,] 0 0 0 4 0*

*[5,] 0 0 0 0 5*

Let us also look at the diagonals (6:10) of the Trace Matrix of the Frobenius Norm Trace Matrix:

B <- diag(x = 6:10)

B

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 6 0 0 0 0*

*[2,] 0 7 0 0 0*

*[3,] 0 0 8 0 0*

*[4,] 0 0 0 9 0*

*[5,] 0 0 0 0 10*

Let us anow look at the diagonals (11:15) of the Trace Matrix of the Frobenius Norm Trace Matrix:

C <- diag(x = 11:15)

C

*[,1] [,2] [,3] [,4] [,5]*

*[1,] 11 0 0 0 0*

*[2,] 0 12 0 0 0*

*[3,] 0 0 13 0 0*

*[4,] 0 0 0 14 0*

*[5,] 0 0 0 0 15*

Let us alsos see if all conditions of the Frobenius Norm Trace Matrix are true:

all.equal(tr(A %\*% B %\*% C), tr(C %\*% A %\*% B))

*[1] TRUE*

all.equal(tr(C %\*% A %\*% B), tr(B %\*% C %\*% A))

*[1] TRUE*