Matrix Algebra in R Cheatsheet

Update: Jan 2021

IDEA

Rensselaer Institute for Data Exploration and Applications

Creating Rectangular Matrices (random data)

```
# Generate a rectangular matrix with 10 rows, 3 columns
set.seed(222)   # Always set a random seed (for repeatability)
A <- matrix(runif(30), nrow=10, ncol=3)
# Generate a rectangular matrix with 3 rows, 5 columns
B <- matrix(runif(15), nrow=3, ncol=5)
# Generate a rectangular matrix with 4 rows, 4 columns
C <- matrix(runif(16), nrow=4, ncol=4)
```

Examining (Inspecting) Matrices

```
# Is A a matrix?
is.matrix(A)
# Dimensions of matrix A
dim(A)
# Number of rows or columns of A
nrow(A)
ncol(A)
# Assign row and column names to A
rownames(A) <- 1:10
colnames(A) <- c("a1", "a2", "a3")
# Find the class of object 'A'
class(a) # Should be 'Matrix'
# Find the type of 'A'
typeof(A)
# Show the first few rows of 'A'
head(A) # VERY useful!
# Show the last few rows of 'A'
tail(A)
# Summarize 'A'
summary(A)
# Show row '2' of 'A' (only)
A[2,]
# Show columns 2 & 3 of 'A' (only)
A[,2:3]
```

Matrix "Gotchas": Common Problems

```
# Element-wise multiplication vs. matrix multiplication
A * B     # Element-by-element multiplication
A %*% B   # Matrix multiplication
# Avoid `==` when testing equality in floating point objects
isTRUE(all.equal(X, Y)) # Handles nearly-equal numbers
identical(X, Y)  # Safe, reliable way to test two f.p. objects

# Columns or rows extracted from matrices are simple vectors
# You must 'convert" them to matrices for them to behave!
A <- matrix(c(1,2,3,4,5,6,7,8,9), nrow=3) # Make a 3x3 matrix
a <- A[1,] # contents of row 1
b <- A[,2] # contents of column 2
a1 <- matrix(a, nrow=1) # a with correct orientation
b1 <- matrix(b) # b with correct orientation
```

Reference: Basic

Element-wise multiplication
```
A * B
```
Matrix multiplication
```
A %*% B
```
Outer product. AB'
```
A %o% B
```
Dot Product of Vectors
```
dot(a, b)
```
A'B and A'A respectively
```
crossprod(A,B)
crossprod(A)
```
Transpose (Vector or Matrix)
```
t(A)
```
Create diagonal matrix
```
diag(x) # x is a vector
```
Return principal diagonal
```
diag(A) # A is a vector
```
Create kxk identity matrix
```
diag(k) # k is the dimension
```
Solve for x when: x b = Ax
```
solve(A, b)
```
Inverse of A
```
solve(A)
```
Combine matrices (horiz)
```
cbind(A,B,...)
```
Combine matrices (vert)
```
rbind(A,B,...)
```
Create vector of row means
```
rowMeans(A)
```
Create vector of row sums
```
rowSums(A)
```
Create vector of col means
```
colMeans(A)
```
Create vector of col sums
```
colSums(A)
```
Test if object is a matrix
```
is.matrix(A)
```
Change type to Matrix
```
as.matrix(A)
```

Useful Matrix Operations

```
# Matrix Multiplication: AB
A %*% B   # Matrix multiplication
# Transpose of B: B'
t(B)
# Matrix Product: B'A'
t(B) %*% t(A)
# Scalar multiplication
5 * B
B * 5
# Extract diagonal elements of a square matrix
diag(C)  # C is a square matrix
# Trace of a square matrix
sum(diag(C))
# Determinant of a square matrix
det(C)
# Create a 5x5 identity matrix
I <- diag(5)
# Inverse of a square matrix
solve(C)
# Singular value decomposition (SVD)
svd(A)
# Eigendecomposition (of a symmetric matrix)
eigen( C %*% t(C))
```

Functions for Basic Calculations

```
# Sum of elements by rows
rowSums(A)
# Sum of elements by columns
colSums(A)
# Mean of elements by rows
rowMeans(A)
# Mean of elements by columns
colMeans(A)
```

Handy Functions

```
# Center matrix A
scale(A, scale=FALSE) # Centering, no scaling
# Standardize A: variables with mean=0, var=1
scale(A)  # Centering and scaling are defaults
# Elements as fraction of the total sum
prop.table(A)
# Elements as fraction of rows margin
prop.table(A, 1)
# Elements as fraction of columns margin
prop.table(A, 2)
```

NOTE: See ?scale specific details!

Examples of Applying Functions

```
# Sum of elements by rows
apply(A, 1, sum)
# Sum of elements by columns
apply(A, 2, sum)
# Standard deviation of elements by rows
apply(A, 1, sd)
# Standard deviation of elements by rows
apply(A, 2, sd)
# Maximum of elements by rows
apply(A, 1, max)
# Minimum of elements by columns
apply(A, 2, min)
```

Create a Matrix from a CSV (known to be numeric)

```
# Common method: First creates a dataframe using read.csv()
# No row or column names imported
m1 <- as.matrix(read.csv("file.csv", sep=",", header = FALSE))
# Row names in column 1, column names in row 1 (the header)
m2 <- as.matrix(read.csv("file.csv", sep=",", row.names=1))

# NOTE: RStudio uses built-in important functions such as read_csv
# from `readr` package; these produce tibbles (special dataframes)
m3 <- as.matrix(read_csv("file.csv", col_names = FALSE))
```

Principal Component Analysis Basics

```
# prcomp() comes with the default "stats" package, which
# means that you don't have to install anything.

# PCA with function prcomp
pca1 = prcomp(USArrests, scale. = TRUE)

# sqrt of eigenvalues
pca1$sdev

# view the loadings
head(pca1$rotation)

# view the principal components (aka scores)
head(pca1$x)

# biplot (see upper figure, right)
biplot(pca1)

# "scree" or loadings plot (see lower figure, right)
plot(pca1)
```

Reference: Advanced

Moore-Penrose Inverse of A

```
ginv(A)
```

y$val : the eigenvalues of A

y$vec : the eigenvectors of A

```
Y <- eigen(A)
```

Single value decomposition of A

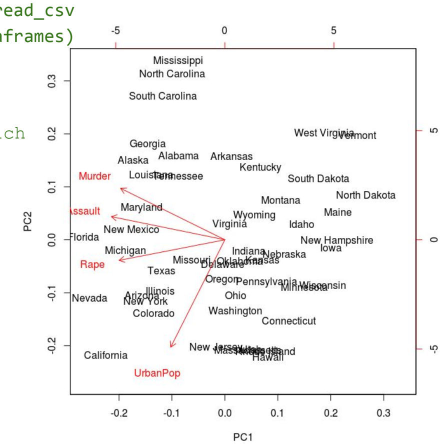
```
Y <- svd(A)
```

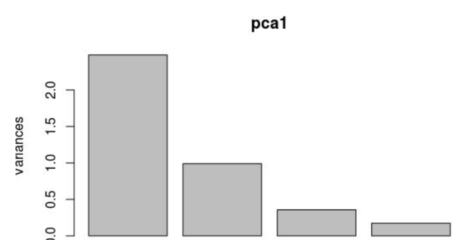
Cholesky factorization of A

```
R <- chol(A)
```

QR decomposition of A

```
y <- qr(A)
```





Create a Matrix from a Data Frame

```
# From ?data.matrix: "Return the matrix obtained by converting all the
# variables in a data # frame to numeric mode and then binding them
# together as the columns of a matrix. Factors and ordered factors are
# replaced by their internal codes. NOTE: Use the usual techniques to
# select a subset of `myDataFrame` if required

data.matrix(myDataFrame)
```

Create Matrices from Vectors...

```
# Given a set of vectors a, b, c
# Treating a, b, c as column vectors
> a <- c(1,2,3); b <- c(4,5,6); c <- c(7,8,9)
> as.matrix(cbind(a,b,c))
     a b c
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
# Treating a, b, c as column vectors
> as.matrix(rbind(a,b,c))
  [,1] [,2] [,3]
a    1    2    3
b    4    5    6
c    7    8    9
```

...or from one vector

```
# Given both dimensions
> mat = matrix(1:12,4,3)
> mat
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
# Given one dimension
# Same result
> mat = matrix(1:12,ncol=3)
> mat = matrix(1:12,nrow=4)
```

Naming rows and columns of a matrix

```
# Naming rows of A, a 3x3 matrix
row.names(A) <- c("R1", "R2", "R3")
# Naming columns of A
colnames(A) <- c("C1", "C2", "C3")
```

Compute the norm of a Vector

```
# Compute a vector norm explicitly
sqrt(sum(x^2))
# Compute vector norm using LAPACK.
# See also "Compute the Norm of a Matrix"
norm(x, type = "2")
```

Compute the norm of a Vector

# Compute a vector norm explicitly
sqrt(sum(x^2))
# Compute vector norm using LAPACK.
# See also "Compute the Norm of a Matrix"
norm(x, type = "2")

Compute the norm of a Matrix

```
# Compute a matrix norm of x using LAPACK. The norm can be the one ("O")
# norm, the infinity ("I") norm, the Frobenius ("F") norm, the maximum
# modulus ("M") among elements of a matrix, or the "spectral" or "2"-norm,
# as determined by the value of type.
norm(x, type = "F")
```

Find other RStudio Cheatsheets at
https://www.rstudio.com/resources/cheatsheets/