Additional Material for Operations on Matrices and Data Frames

Calculating the inverse of a matrix

The standard R function to calculate the inverse of a matrix is `solve`, which comes with the **R Base** package. This generic function (`solve`) solves the linear system `a %*% x = b` for `x`. In this equation, `a` is a square numeric or complex matrix containing the coefficients of the linear system. `b` is a numeric or complex vector or matrix giving the right-hand side of the linear system (Documentation 2018). If missing, `b` is taken to be an identity matrix and `solve` function will return the inverse of `a`.

Suppose we want to find the inverse of the following matrix `matA`.

```
matAdata <- c(5, 1, 0, 3, -1, 2, 4, 0, -1)
matA <- matrix(matAdata, nrow=3, byrow=TRUE)
matA
```

```
##      [,1] [,2] [,3]
## [1,]    5    1    0
## [2,]    3   -1    2
## [3,]    4    0   -1
```

We know that the inverse exists only for non-singular matrices. Thus, we need to make sure that the determinant of `matA` is a non-zero value. Accordingly, we use `solve` function to calculate the inverse of the matrix `matA`.

```
if(det(matA)!=0) {
  invA <- solve(matA)
  invA
} else {
    print("matA is a singular matrix!")
}
```

```
##        [,1]    [,2]   [,3]
## [1,] 0.0625  0.0625  0.125
## [2,] 0.6875 -0.3125 -0.625
## [3,] 0.2500  0.2500 -0.500
```

The inverse of `matA` (`invA`), when multiplied by `matA`, gives the identity matrix. Thus, we can also conclude that the inverse of an inverse of a matrix is the matrix itself.

```
invA %*% matA
```

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

There are other functions also like `inv`, `Inverse` in R which can be used to calculate the inverse of a matrix (Friendly 2018). These functions are available in the library named `matlib`.

Calculating sum of rows or columns of a matrix (or a data frame)

To calculate sums across all rows in a matrix or dataframe, we use `rowSums` function.

Consider three vectors as given below.

```
vec1 <- c(1, 2, 3)
vec2 <- c(4, 5, NA)
vec3 <- c(6, 7, 8)
vec1; vec2; vec3
```

```
## [1] 1 2 3
```

```
## [1]  4  5 NA
```

```
## [1] 6 7 8
```

In `vec2`, there is one element `NA`. It is a logical constant of length 1 which contains a missing value indicator. Now, we create a 3×3 matrix (`matB`) from these three vectors.

```
matBdata <- c(vec1, vec2, vec3)
matB <- matrix(matBdata, nrow = 3)
matB
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    6
## [2,]    2    5    7
## [3,]    3   NA    8
```

Suppose we want to find the sums across all rows in `matB`. For this, we use `rowSums` function.

```
rowSums(matB)
```

```
## [1] 11 14 NA
```

For `matB`, the sums across first row and second row are evaluated to be 11 and 14 respectively. However, the sum across third row is evaluated to be `NA`. This is due to the fact that the third row of `matB` contains `NA`. Thus, we need to remove this `NA`, if we want to add the remaining elements in this row. For this, we use another logical argument `na.rm` in `rowSums` function. By default, `na.rm` is set to `FALSE`.

```
rowSums(matB, na.rm = TRUE)
```

```
## [1] 11 14 11
```

Similarly, we use `colSums` function to calculate sums across all columns in a matrix or dataframe (Phillips 2018).

```
colSums(matB, na.rm = TRUE)
```

```
## [1]  6  9 21
```

Suppose we want to find the mean across all columns in `matB`. For this, we use `colMeans` function.

```
colMeans(matB, na.rm = TRUE)
```

```
## [1] 2.0 4.5 7.0
```

Similarly, we can use `rowMeans` function to find the mean across all rows in a matrix.

```
rowMeans(matB, na.rm = TRUE)
```

```
## [1] 3.666667 4.666667 5.500000
```

Adding row(s) or column(s) to a matrix

Consider two different vectors as given below.

```
vec4 <- 9:12
vec5 <- 13:16
vec4; vec5
```

```
## [1]  9 10 11 12
```

```
## [1] 13 14 15 16
```

Now, we create a 4×2 matrix using these two vectors `vec4` and `vec5`.

```
matCdata <- c(vec4, vec5)
matC <- matrix(matCdata, nrow = 4, ncol = 2)
matC
```

```
##      [,1] [,2]
## [1,]    9   13
## [2,]   10   14
## [3,]   11   15
## [4,]   12   16
```

Now, suppose we want to add a vector (with elements as 17, 18, 19, and 20) as a column to the matrix `matC`. For this, we use `cbind` function.

```
matC <- cbind(matC, 17:20)
matC
```

```
##      [,1] [,2] [,3]
## [1,]    9   13   17
## [2,]   10   14   18
## [3,]   11   15   19
## [4,]   12   16   20
```

Thus, the updated matrix `matC` contains 4 rows and 3 columns. This can also be verified by using `dim` function.

```
dim(matC)
```

```
## [1] 4 3
```

References

Documentation, R. 2018. "R: Solve a System of Equations." https://stat.ethz.ch/R-manual/R-devel/library/base/html/solve.html.

Friendly, Michael. 2018. "Inverse of a matrix." https://cran.r-project.org/web/packages/matlib/vignettes/inv-ex1.html.

Phillips, Nathaniel D. 2018. "YaRrr! The Pirate's Guide to R." https://bookdown.org/ndphillips/YaRrr/additional-aggregation-functions.html.