Introduction to R, Part IV

Rebecca C. Steorts, STA 360

Agenda

- Matrices
- Matrix Operations
- Examples

Simple example: resource allocation ("mathematical programming")

Factory makes cars and trucks, using labor and steel

- a car takes 40 hours of labor and 1 ton of steel
- a truck takes 60 hours and 3 tons of steel
- resources: 1600 hours of labor and 70 tons of steel each week

Matrices

In R, a matrix is a specialization of a 2D array

```
factory <- matrix(c(40,1,60,3),nrow=2)
is.array(factory)
```

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

```
is.matrix(factory)
```

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

could also specify `ncol`, and/or `byrow=TRUE` to fill by rows.

Element-wise operations proceed as usual (e.g., `factory/5`)

Matrix multiplication

Gets a special operator

```
six.sevens <- matrix(rep(7,6),ncol=3)
six.sevens
```

```
##      [,1] [,2] [,3]
## [1,]    7    7    7
## [2,]    7    7    7
```

```
factory %*% six.sevens # [2x2] * [2x3]
```

```
##      [,1] [,2] [,3]
## [1,]  700  700  700
## [2,]   28   28   28
```

Exercise: What if you try `six.sevens %*% factory`?

Multiplying matrices and vectors

Numeric vectors can act like proper vectors:

```
output <- c(10,20)
factory %*% output
```

```
##      [,1]
## [1,] 1600
## [2,]   70
```

```
output %*% factory
```

```
##      [,1] [,2]
## [1,]  420  660
```

R silently casts the vector as either a row or a column matrix

Matrix operators

Transpose:

```
t(factory)
```

```
##      [,1] [,2]
## [1,]   40    1
## [2,]   60    3
```

Determinant:

```
det(factory)
```

```
## [1] 60
```

The diagonal

The `diag()` function can extract the diagonal entries of a matrix:

```
diag(factory)
```

```
## [1] 40  3
```

Creating a diagonal or identity matrix

```
diag(c(3,4))
```

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

```
diag(2)
```

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

```
diag(10)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1    0    0    0    0    0    0    0    0     0
##  [2,]    0    1    0    0    0    0    0    0    0     0
##  [3,]    0    0    1    0    0    0    0    0    0     0
##  [4,]    0    0    0    1    0    0    0    0    0     0
##  [5,]    0    0    0    0    1    0    0    0    0     0
##  [6,]    0    0    0    0    0    1    0    0    0     0
##  [7,]    0    0    0    0    0    0    1    0    0     0
##  [8,]    0    0    0    0    0    0    0    1    0     0
##  [9,]    0    0    0    0    0    0    0    0    1     0
## [10,]    0    0    0    0    0    0    0    0    0     1
```

Inverting a matrix

```
solve(factory)
```

```
##             [,1]       [,2]
## [1,]  0.05000000 -1.0000000
## [2,] -0.01666667  0.6666667
```

```
factory %*% solve(factory)
```

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

Why's it called "solve"" anyway?

Solving the linear system $\mathbf{A}\vec{x} = \vec{b}$ for $\vec{x}$:

```
# this is acting like b in our linear system
# factory is behaving like A
# goal is to solve for x
available <- c(1600,70)
solve(factory,available)
```

```
## [1] 10 20
```

```r
factory %*% solve(factory,available)
```

```
##      [,1]
## [1,] 1600
## [2,]   70
```

Names in matrices

We can name either rows or columns or both, with `rownames()` and `colnames()`

These are character vectors

We use the same function to get and to set their respective values

Names are useful since they help us keep track of what we are working with

Example

```r
rownames(factory) <- c("labor","steel")
colnames(factory) <- c("cars","trucks")
factory
```

```
##       cars trucks
## labor   40     60
## steel    1      3
```

```r
available <- c(1600,70)
names(available) <- c("labor","steel")
```

Example (Continued)

```r
output <- c(20,10)
names(output) <- c("cars","trucks")
factory %*% output
```

```
##       [,1]
## labor 1400
## steel   50
```

```r
factory %*% output[colnames(factory)]
```

```
##       [,1]
## labor 1400
## steel   50
```

```r
all(factory %*% output[colnames(factory)] <= available[rownames(factory)])
```

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

Summaries

Take the mean: `rowMeans()`, `colMeans()`: input is matrix, output is vector. Also `rowSums()`, etc.

`summary()`: vector-style summary of column

```
colMeans(factory)
```

```
##   cars trucks
##   20.5   31.5
```

```
summary(factory)
```

```
##       cars           trucks     
##  Min.   : 1.00   Min.   : 3.00  
##  1st Qu.:10.75   1st Qu.:17.25  
##  Median :20.50   Median :31.50  
##  Mean   :20.50   Mean   :31.50  
##  3rd Qu.:30.25   3rd Qu.:45.75  
##  Max.   :40.00   Max.   :60.00  
```

Apply function

`apply()`, takes 3 arguments: the array or matrix, then 1 for rows and 2 for columns, then name of the function to apply to each

```
rowMeans(factory)
```

```
## labor steel 
##    50     2
```

```
apply(factory,1,mean)
```

```
## labor steel 
##    50     2
```

What would `apply(factory,1,sd)` do?