1-05b Matrixes

CSI 500

Course material derived from:
An Introduction to R. Notes on R: A Programming Environment for Data Analysis and Graphics
Version 3.4.3 (2017-11-30)
https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

Matrixes in R

- A matrix is a 2-dimensional set of vectors
  - has rows
  - has columns
  - byrow indicates row major ordering (default is column major)
  - all data must be the same base type (numeric, character, logical)

```
> x = matrix(nrow=2, ncol=3, byrow=TRUE, data=c(1,3,5,2,4,6))
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> y = matrix(nrow=2, ncol=3, byrow=FALSE, data=c(1,3,5,2,4,6))
> y
     [,1] [,2] [,3]
[1,]    1    5    4
[2,]    3    2    6
>
```

Inner and outer products

- R supports vector arithmetic
- Outer product is formed using the %o% operator
  - alternatively, you can also use the more general form **outer(x, y, "*")**
- Inner product is formed using the %*% operator

```
# inner and outer product example
> x = 2:4
> x
[1] 2 3 4

> y = 6:8
> y
[1] 6 7 8

> x %o% y
     [,1] [,2] [,3]
[1,]   12   14   16
[2,]   18   21   24
[3,]   24   28   32

> x %*% y
     [,1]
[1,]   65
>
```

Matrix multiplication

- R supports matrix arithmetic
- Multiplication is performed using the %*% operator
  - can be used for conforming matrixes or vectors

```
# matrix multiplication example
> x = matrix(nrow=2,ncol=2,data=1:4)
> y = matrix(nrow=2,ncol=2,data=5:8)

> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> y
     [,1] [,2]
[1,]    5    7
[2,]    6    8

> x %*% y
     [,1] [,2]
[1,]   23   31
[2,]   34   46
>
```

More multiplication

- You can multiply vectors and matrixes
  - using the %*% operator
  - using crossprod(A, b), which is shorthand for t(A) %*% b
- diag() has two meanings:
  - for vectors, creates a matrix where the vector is the diagonal
  - for matrixes, returns the diagonal of the matrix
  - Compatible syntax with MATLAB ™ products

```
# matrix multiplication example
> X = matrix(nrow=2,ncol=2,data=1:4)
> X
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> z = 10:11

> X %*% z
     [,1]
[1,]   43
[2,]   64

> z %*% X
     [,1] [,2]
[1,]   32   74

> crossprod(X, z)
     [,1]
[1,]   32
[2,]   74
```

```
> diag(z)
     [,1] [,2]
[1,]   10    0
[2,]    0   11
> diag(X)
[1] 1 4
>
```

Matrix transposition

- R supports matrix operations
- To compute the transpose of a vector or matrix, use the t() function

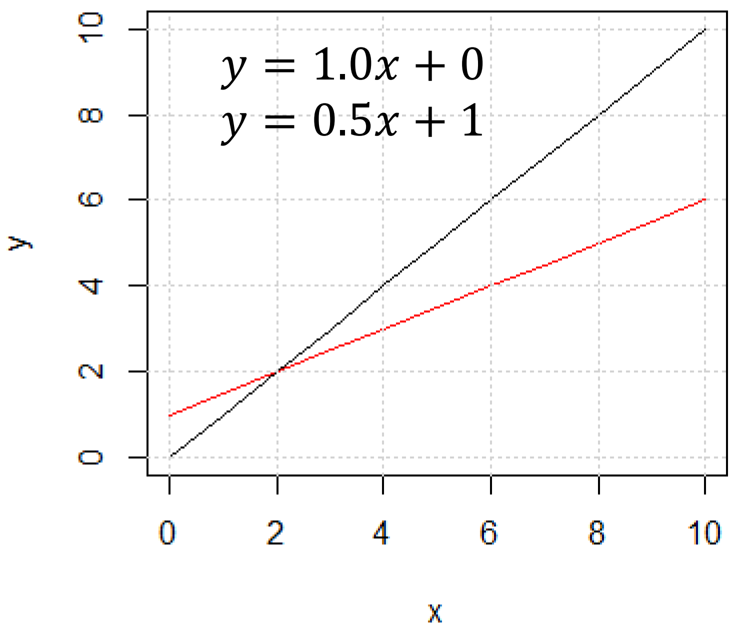
```
# matrix transposition example
> x = matrix(nrow=2, ncol=4, data=1:8)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8

> t(x)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8

> y = 1:8
> y
[1] 1 2 3 4 5 6 7 8
> t(y)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    2    3    4    5    6    7    8
>
```

Example: Linear Equations

- R can be used to solve linear equation problems
  - consider this application with two linear equations



Rearranging in Matrix form…

$$\begin{bmatrix} -1.0 & 1 \\ -0.5 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Recalling matrix algebra…

$$Ax = b$$

$$A^{-1}Ax = A^{-1}b$$

$$x = A^{-1}b$$

```
# linear equation example
> A = matrix(nrow=2,ncol=2,
> + data=c(-1,1,-0.5,1),byrow=TRUE)
> A
      [,1] [,2]
[1,] -1.0    1
[2,] -0.5    1
>
> b = c(0,1)
> b
[1] 0 1
>
> # solve(A) computes matrix inv
> solve(A)
     [,1] [,2]
[1,]   -2    2
[2,]   -1    2

> # solve(A,b) computes solution
> x = solve(A,b)
> x
[1] 2 2
>
```

cbind() and rbind()

- use cbind() to assemble a matrix using column vectors
  - c stands for "column-bind"
- use rbind() to assemble a matrix using row vectors
  - r stands for "row-bind"

```
# cbind and rbind examples
> x = cbind( 1:3, 5:7, 11:13)
> x
     [,1] [,2] [,3]
[1,]    1    5   11
[2,]    2    6   12
[3,]    3    7   13
>
> y = rbind( 1:3, 5:7, 11:13)
> y
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    5    6    7
[3,]   11   12   13
```

Flattening matrixes

- as we've seen before, use the c() function to "concatenate" elements into an array
- You can also use c() to "flatten"' a matrix into a vector
  - Note carefully: elements are concatenated in column-major order
- Purists note that the official way to flatten a matrix is using the as.vector() casting function
  - achieves the same results…

```
# cbind and rbind examples
> x = rbind( 1:3, 5:7, 11:13)
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    5    6    7
[3,]   11   12   13
>
> z = c(x)
> z
[1]  1  5 11  2  6 12  3  7 13
>
>
> z = as.vector(x)
> z
[1]  1  5 11  2  6 12  3  7 13
```

Summary

- R provides support for matrixes
  - matrix is a 2-D collection of vectors
  - vector-matrix operations supported
  - matrix-matrix operations supported
- Matrixes built using matrix() command
  - specify nrow, ncol, byrows, and data
- Matrixes built using cbind(), rbind()
  - "glue" together directly from vectors

Advanced Matrix Methods

- R provides tools for advanced matrix computations
  - Eigenvalues and Eigenvectors
  - Singular Value Decomposition
  - Least Squares fitting
- If you need these, check the documentation and your matrix algebra texts…

Frequency tables from factors

- Arrays can be used to compute frequency tables
  - handy for assessing frequency of occurrence in ordinal data set

```
# freq table example
# assume we did a survey and got data in array "d"
# on a scale of 1 to 5, respond to this question: "I think R is better than Python".
> d    # we can generate random test data using d = sample(1:5, 10, replace=TRUE)
 [1] 3 5 1 2 1 4 1 4 2 4
> dset = ordered(d, levels=c(1:5),
+ labels=c("strong_disagree","disagree","indifferent","agree","strong_agree"))
> dset
 [1] indifferent     strong_agree    strong_disagree disagree
 [5] strong_disagree agree           strong_disagree agree
 [9] disagree        agree
Levels: strong_disagree < disagree < indifferent < agree < strong_agree
> df = table(dset)
> df
dset
strong_disagree        disagree     indifferent           agree    strong_agree
              3               2               1               3               1
```

Index matrixes

- The vector indexing concept also works for matrixes
- In this case, the index is itself a matrix, not just a scalar or a vector
  - index matrix contains row:column pairs to extract from main matrix
- Example: lets extract the N,E,S, and W elements from a 3x3 matrix
  - idx is our index matrix
  - has 4 rows, 2 columns
  - data represent row:col indexes

```
# index matrixes
> x = sample(1:9, 9)
> x
[1] 8 9 5 4 3 1 6 7 2
>
> y = matrix(nrow=3,ncol=3,data=x)
> y
     [,1] [,2] [,3]
[1,]    8    4    6
[2,]    9    3    7
[3,]    5    1    2
>
> idx = matrix(nrow=4,ncol=2,
+ data=c(1,2,2,3, 2,1,3,2))
>
> idx
     [,1] [,2]
[1,]    1    2
[2,]    2    1
[3,]    2    3
[4,]    3    2
>
> y[idx]
[1] 4 9 7 1
```