4 Lecture 4: Jan 22

Last time

- Linear algebra: vector and vector space, rank of a matrix, Column space (JM Appendix A)

Today

- Nullspace
- Intro to R

Null space

Definition: The null space of a matrix, denoted by $\mathcal{N}(\mathbf{A})$, is $\mathcal{N}(\mathbf{A}) = \{\mathbf{y} : \mathbf{A}\mathbf{y} = \mathbf{0}\}$.

Result A.3

If $\mathbf{A}$ has full-column rank, then $\mathcal{N}(\mathbf{A}) = \{\mathbf{0}\}$.

proof:

Theorem A.1

Assume $\mathbf{A} \in \mathbb{R}^{m\times n}$, then $\dim(\mathcal{C}(\mathbf{A})) = r$ and $\dim(\mathcal{N}(\mathbf{A})) = n - r$, where $r = \text{rank}(\mathbf{A})$.

See JM Appendix Theorem A.1 for the proof.
proof: Denote $\dim(\mathcal{N}(\mathbf{A}))$ by k, to be determined, and construct a set of basis vectors for $\mathcal{N}(\mathbf{A})$: $\left\{\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \ldots, \mathbf{u}^{(k)}\right\}$, so that $\mathbf{A}\mathbf{u}^{(i)} = \mathbf{0}$, for $i = 1, 2, \ldots, k$. Now, construct a basis for $\mathbb{R}^n$ by adding the vectors $\left\{\mathbf{u}^{(k+1)}, \ldots, \mathbf{u}^{(n)}\right\}$, which are not in $\mathcal{N}(\mathbf{A})$. Clearly, $\mathbf{A}\mathbf{u}^{(i)} \in \mathcal{C}(\mathbf{A})$ for $i = k+1, \ldots, n$, and so the span of these vectors form a subspace of $\mathcal{C}(\mathbf{A})$. These vectors $\left\{\mathbf{A}\mathbf{u}^{(i)}, i = k+1, \ldots, n\right\}$ are also linearly independent from the following argument: suppose $\sum_{i=k+1}^{n} c_i \mathbf{A}\mathbf{u}^{(i)} = \mathbf{0}$; then $\sum_{i=k+1}^{n} c_i \mathbf{A}\mathbf{u}^{(i)} = \mathbf{A}\left[\sum_{i=k+1}^{n} c_i \mathbf{u}^{(i)}\right] = \mathbf{0}$, and hence $\sum_{i=k+1}^{n} c_i \mathbf{u}^{(i)}$ is a vector in $\mathcal{N}(\mathbf{A})$. Therefore, there exist b_i such that $\sum_{i=k+1}^{n} c_i \mathbf{u}^{(i)} = \sum_{i=1}^{k} b_i \mathbf{u}^{(i)}$, or $\sum_{i=1}^{k} b_i \mathbf{u}^{(i)} - \sum_{i=k+1}^{n} c_i \mathbf{u}^{(i)} = \mathbf{0}$. Since $\left\{\mathbf{u}^{(i)}\right\}$ form a basis for $\mathbb{R}^n$, c_i must all be zero. Therefore $\mathbf{A}\mathbf{u}^{(i)}, i = k+1, \ldots, n$ are linearly independent. At this point, since $\text{span}\{\mathbf{A}\mathbf{u}^{(k+1)}, \ldots, \mathbf{A}\mathbf{u}^{(n)}\} \subseteq \mathcal{C}(\mathbf{A})$, $\dim(\mathcal{C}(\mathbf{A}))$ is at least $n - k$. Suppose there is a vector $\mathbf{y}$ that is in $\mathcal{C}(\mathbf{A})$, but not in the span; then there exists $\mathbf{u}^{(n+1)}$ so that $\mathbf{y} = \mathbf{A}\mathbf{u}^{(n+1)}$ and $\mathbf{u}^{(n+1)}$ is linearly independent of $\{\mathbf{u}^{(k+1)}, \ldots, \mathbf{u}^{(n)}\}$ (and clearly not in $\mathcal{N}(\mathbf{A})$), making $n+1$ linearly independent vectors in $\mathbb{R}^n$. Since that is not possible, the span is equal to $\mathcal{C}(\mathbf{A})$ and $\dim(\mathcal{C}(\mathbf{A})) = n - k = r = \text{rank}(\mathbf{A})$, so that $k = \dim(\mathcal{N}(\mathbf{A})) = n - r$.

Interpretation: "dimension of column space + dimension of null space = # columns"
Mis-Interpretation: Columns space and null space are orthogonal complement to each other. They are of different orders in general!

R Basics

MATH-7260 Linear Models

Dr. Xiang Ji @ Tulane University

January 22, 2025

Contents

Announcement

- Just stay warm.

R basics

styles

(reading assignment)

Checkout Style guide in Advanced R and the tidyverse style guide.

Arithmetic

R can do any basic mathematical computations.

symbol	use
+	addition
-	subtraction
*	multiplication
/	division
^	power

symbol	use
%%	modulus
exp()	exponent
log()	natural logarithm
sqrt()	square root
round()	rounding
floor()	flooring
ceiling()	ceiling

Objects

You can create an R object to save results of a computation or other command.

Example 1

```
x <- 3 + 5
x
```

```
## [1] 8
```

- In most languages, the direction of passing through the value into the object goes from right to left (e.g. with "="). However, R allows both directions (which is actually bad!). In this course, we encourage the use of "<-" or "=". There are people liking "=" over "<-" for the reason that "<-" sometimes break into two operators "< -".

Example 2

```
x < - 3 + 5
```

```
## [1] FALSE
```

```
x
```

```
## [1] 8
```

- For naming conventions, stick with either "." or "_" (refer to the style guide).

Example 3

```
sum.result <- x + 5
sum.result
```

```
## [1] 13
```

- *important*: many names are already taken for built-in R functions. Make sure that you don't override them.

Example 4

```
sum(2:5)
```

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

```
sum
```

```
## function (..., na.rm = FALSE)  .Primitive("sum")
```

```
sum <- 3 + 4 + 5
sum(5:8)
```

```
## [1] 26
```

```
sum
```

```
## [1] 12
```

- R is case-sensitive. “Math.7260” is different from “math.7260”.

Locating and deleting objects:

The commands “objects()” and “ls()” will provide a list of every object that you’ve created in a session.

```
objects()
```

```
## [1] "sum"        "sum.result" "x"
```

```
ls()
```

```
## [1] "sum"        "sum.result" "x"
```

The “rm()” and “remove()” commands let you delete objects (tip: always clearn-up your workspace as the first command)

```
rm(list=ls())  # clean up workspace
```

Vectors

Many commands in R generate a vector of output, rather than a single number.

The “c()” command: creates a vector containing a list of specific elements.

Example 1

```
c(7, 3, 6, 0)
```

```
## [1] 7 3 6 0
```

```
c(73:60)
```

```
##  [1] 73 72 71 70 69 68 67 66 65 64 63 62 61 60
```

```
c(7:3, 6:0)
```

```
##  [1] 7 6 5 4 3 6 5 4 3 2 1 0
```

```
c(rep(7:3, 6), 0)
```

```
##  [1] 7 6 5 4 3 7 6 5 4 3 7 6 5 4 3 7 6 5 4 3 7 6 5 4 3 0
```

Example 2 The command “seq()” creates a sequence of numbers.

```
seq(7)
```

```
## [1] 1 2 3 4 5 6 7
```

```
seq(3, 70, by = 6)
```

```
##  [1]  3  9 15 21 27 33 39 45 51 57 63 69
```

```
seq(3, 70, length = 6)
```

```
## [1]  3.0 16.4 29.8 43.2 56.6 70.0
```

Operations on vectors

Use brackets to select element of a vector.

```
x <- 73:60
x[2]
```

```
## [1] 72
```

```r
x[2:5]
```

```
## [1] 72 71 70 69
```

```r
x[-(2:5)]
```

```
##  [1] 73 68 67 66 65 64 63 62 61 60
```

Can access by "name" (safe with column/row order changes)

```r
y <- 1:3
names(y) <- c("do", "re", "mi")
y[3]
```

```
## mi
##  3
```

```r
y["mi"]
```

```
## mi
##  3
```

R commands on vectors

command	usage
sum()	sum over elements in vector
mean()	compute average value
sort()	sort elements in a vector
min(), max()	min and max values of a vector
length()	length of a vector
summary()	returns the min, Q1, median, mean, Q3, and max values of a vector
sample(x, size, replace = FALSE, prob = NULL)	takes a random sample from a vector with or without replacement

Exercise Write a command to generate a random permutation of the numbers between 1 and 5 and save it to an object.

Matrix

matrix() command creates a matrix from the given set of values

```r
matrix.example <- matrix(rnorm(100), nrow = 10, ncol = 10, byrow = TRUE)
matrix.example
```

```
##             [,1]        [,2]         [,3]       [,4]        [,5]        [,6]
##  [1,]  0.7085600  0.45192726 -1.003319749  0.1423933  0.02689608  0.29426937
##  [2,]  0.7923849  2.06510949  1.649801041  0.8676237  0.35619381 -0.38472953
##  [3,] -0.4306156 -1.20374669  0.006408992 -0.4624155  1.37097762  1.95097502
##  [4,] -0.5609673 -0.15844595  2.118308274 -0.2858027  0.12286302  0.78303984
##  [5,]  0.2611649  0.85248428 -0.088204361 -1.2360259  0.10968954 -0.01665417
##  [6,] -0.8074358  1.24710264 -1.057458954 -0.8122926  0.76168790 -0.64026974
##  [7,] -1.6987343  1.29180951  1.356032706 -0.1167454  0.72640807 -0.01960615
##  [8,]  1.0857579 -0.58004441  0.516984986  0.9718219 -0.52717205  0.71149957
##  [9,]  0.9201054  0.09870030 -0.363956709  0.7876533  0.69938916 -1.00800723
## [10,] -0.8099918 -0.05265558  0.888416514 -0.5722519  1.55230991 -0.74722253
##              [,7]        [,8]        [,9]      [,10]
```

```
##  [1,] -0.42980335  0.99600581 -0.17342945  2.1574548
##  [2,] -0.03436166  0.48525730  0.21522078 -0.8532752
##  [3,] -0.36446732 -1.19891787  0.59749240 -0.1557247
##  [4,]  0.21571899 -1.67023445  0.79153489  0.3170291
##  [5,]  0.26161545 -0.04543697 -0.02049376  0.8593349
##  [6,] -1.57391952  0.13575105  0.58540460  0.1644719
##  [7,] -1.45332193 -0.03309575  0.21665725 -0.3105336
##  [8,] -2.26311215 -1.46326873 -0.23711252 -0.8892875
##  [9,] -0.84247251 -0.33531167  0.49825814 -0.6812993
## [10,]  0.49761061  1.09562810 -0.13426534 -0.2678570
```

R commands on vector/matrix

command	usage
sum()	sum over elements in vector/matrix
mean()	compute average value
sort()	sort all elements in a vector/matrix
min(), max()	min and max values of a vector/matrix
length()	length of a vector/matrix
summary()	returns the min, Q1, median, mean, Q3, and max values of a vector
dim()	dimension of a matrix
cbind()	combine a sequence of vector, matrix or data-frame arguments and combine by columns
rbind()	combine a sequence of vector, matrix or data-frame arguments and combine by rows
names()	get or set names of an object
colnames()	get or set column names of a matrix-like object
rownames()	get or set row names of a matrix-like object

```
sum(matrix.example)
```

```
## [1] 7.561421
```

```
mean(matrix.example)
```

```
## [1] 0.07561421
```

```
sort(matrix.example)
```

```
##   [1] -2.263112154 -1.698734340 -1.670234449 -1.573919519 -1.463268735
##   [6] -1.453321930 -1.236025933 -1.203746691 -1.198917865 -1.057458954
##  [11] -1.008007235 -1.003319749 -0.889287483 -0.853275187 -0.842472513
##  [16] -0.812292631 -0.809991805 -0.807435817 -0.747222531 -0.681299297
##  [21] -0.640269737 -0.580044411 -0.572251851 -0.560967276 -0.527172048
##  [26] -0.462415550 -0.430615603 -0.429803352 -0.384729527 -0.364467322
##  [31] -0.363956709 -0.335311667 -0.310533647 -0.285802707 -0.267857030
##  [36] -0.237112522 -0.173429451 -0.158445947 -0.155724741 -0.134265338
##  [41] -0.116745417 -0.088204361 -0.052655583 -0.045436971 -0.034361664
##  [46] -0.033095745 -0.020493759 -0.019606146 -0.016654169  0.006408992
##  [51]  0.026896079  0.098700305  0.109689539  0.122863019  0.135751052
##  [56]  0.142393334  0.164471895  0.215220782  0.215718987  0.216657250
```

```
##  [61]  0.261164929  0.261615450  0.294269371  0.317029063  0.356193814
##  [66]  0.451927262  0.485257305  0.497610609  0.498258140  0.516984986
##  [71]  0.585404598  0.597492395  0.699389160  0.708559959  0.711499568
##  [76]  0.726408068  0.761687901  0.783039841  0.787653339  0.791534886
##  [81]  0.792384919  0.852484281  0.859334915  0.867623726  0.888416514
##  [86]  0.920105364  0.971821869  0.996005807  1.085757907  1.095628102
##  [91]  1.247102641  1.291809509  1.356032706  1.370977622  1.552309908
##  [96]  1.649801041  1.950975018  2.065109487  2.118308274  2.157454799
```

```
summary(matrix.example)
```

```
##        V1                  V2                  V3                  V4
##  Min.   :-1.69873   Min.   :-1.2037   Min.   :-1.0575   Min.   :-1.2360
##  1st Qu.:-0.74582   1st Qu.:-0.1320   1st Qu.:-0.2950   1st Qu.:-0.5448
##  Median :-0.08473   Median : 0.2753   Median : 0.2617   Median :-0.2013
##  Mean   :-0.05398   Mean   : 0.4012   Mean   : 0.4023   Mean   :-0.0716
##  3rd Qu.: 0.77143   3rd Qu.: 1.1484   3rd Qu.: 1.2391   3rd Qu.: 0.6263
##  Max.   : 1.08576   Max.   : 2.0651   Max.   : 2.1183   Max.   : 0.9718
##        V5                  V6                  V7                  V8
##  Min.   :-0.5272   Min.   :-1.00801   Min.   :-2.2631   Min.   :-1.67023
##  1st Qu.: 0.1130   1st Qu.:-0.57638   1st Qu.:-1.3006   1st Qu.:-0.98302
##  Median : 0.5278   Median :-0.01813   Median :-0.3971   Median :-0.03927
##  Mean   : 0.5199   Mean   : 0.09233   Mean   :-0.5987   Mean   :-0.20336
##  3rd Qu.: 0.7529   3rd Qu.: 0.60719   3rd Qu.: 0.1532   3rd Qu.: 0.39788
##  Max.   : 1.5523   Max.   : 1.95097   Max.   : 0.4976   Max.   : 1.09563
##        V9                 V10
##  Min.   :-0.2371   Min.   :-0.88929
##  1st Qu.:-0.1058   1st Qu.:-0.58861
##  Median : 0.2159   Median :-0.21179
##  Mean   : 0.2339   Mean   : 0.03403
##  3rd Qu.: 0.5636   3rd Qu.: 0.27889
##  Max.   : 0.7915   Max.   : 2.15745
```

Exercise Write a command to generate a random permutation of the numbers between 1 and 5 and save it to an object.

Comparison (logic operator)

symbol	use
!=	not equal
==	equal
>	greater
>=	greater or equal
<	smaller
<=	smaller or equal
is.na	is it "Not Available"/Missing
complete.cases	returns a logical vector specifying which observations/rows have no missing values
is.finite	if the value is finite
all	are all values in a logical vector true?
any	any value in a logical vector is true?

```r
test.vec <- 73:68
test.vec
```

```
## [1] 73 72 71 70 69 68
```

```r
test.vec < 70
```

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

```r
test.vec > 70
```

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

```r
test.vec[3] <- NA
test.vec
```

```
## [1] 73 72 NA 70 69 68
```

```r
is.na(test.vec)
```

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

```r
complete.cases(test.vec)
```

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

```r
all(is.na(test.vec))
```

```
## [1] FALSE
```

```r
any(is.na(test.vec))
```

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

Now let's do a test of accuracy for doubles in R. Recall that for *Double* precision, we get approximately $\log_{10}(2^{52}) \approx 16$ decimal point for precision.

```r
test.exponent <- -(7:18)
10^test.exponent == 0
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```r
1 - 10^test.exponent == 1
```

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

```r
7360 - 10^test.exponent == 7360
```

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

```r
73600 - 10^test.exponent == 73600
```

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

Other operators

%in%, match

```r
test.vec
```

```
## [1] 73 72 NA 70 69 68
```

```r
66 %in% test.vec
```

```
## [1] FALSE
```

```r
match(66, test.vec, nomatch = 0)
```

```
## [1] 0
```

```r
70 %in% test.vec
```

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

```r
match(70, test.vec, nomatch = 0)
```

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

```r
match(70, test.vec, nomatch = 0) > 0 # the implementation of %in%
```

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

Control flow

These are the basic control-flow constructs of the R language. They function in much the same way as control statements in any Algol-like (Algol short for "Algorithmic Language") language. They are all reserved words.

keyword	usage
if	**if**(*cond*) *expr*
if-else	**if**(*cond*) *cons.expr* **else** *alt.expr*
for	**for**(*var* **in** *seq*) *expr*
while	**while**(*cond*) *expr*
break	breaks out of a *for* loop
next	halts the processing of the current iteration and advances the looping index

Define a function

Read Function section from Advanced R by Hadley Wickham. We will visit functions in more details.

```r
DoNothing <- function() {
  return(invisible(NULL))
}
DoNothing()
```

In general, try to avoid using loops (vectorize your code) in R. If you have to loop, try using **for** loops first. Sometimes, **while** loops can be dangerous (however, a smart *compiler* should detect this).

```r
DoBadThing <- function() {
  result <- NULL
  while(TRUE) {
    result <- c(result, rnorm(100))
  }
  return(result)
}
# DoBadThing()
```

Install packages

You can install R packages from several places (reference):

- Comprehensive R Archive Network (CRAN)
  - Official R packages repository

  - Some levels of code checks (cross platform support, version control etc)
  - Most common place you will install packages
  - Pick a mirror location near you
  - `install.packages("packge_name")`
- GitHub
  - May get development version of a package
  - Almost zero level of code checks
  - Common place to develop a package before submitting to CRAN

```
install.packages("devtools")
library(devtools)
install_github("tidyverse/ggplot2")
```

Load packages

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
require(tidyverse)
```