Session 3: Basics of R

math operations; using variables

Al Cooper

RAF Sessions on R and RStudio

R as a Calculator

Calculator-like operations

- Standard interactive R (type "R")
- RStudio console provides some conveniences
- Can do some simple programming interactively – see below
- lines below starting ## [1] are output from preceding line

```
sin(45*pi/180)  ## pi is a known variable
## [1] 0.7071068
exp(-1)         ## exponential
## [1] 0.3678794
pnorm(1)        ## cumulative normal PDF > 1 std dev
## [1] 0.8413447
## simple functions: the quadratic formula (one branch)
quadForm1 <- function(a, b, c)
    (-b+sqrt(b^2-4*a*c))/(2*a)
quadForm1 (1,4,4)  ## x^2+4x+4=0: a=1,b=4,c=4
## [1] -2
```

Another Example: Roll angle for a 3-min turn

Equation to solve for roll angle ϕ:

For an aircraft flying at airspeed v, for a turn radius r,

$$\frac{v^2}{r} = g \tan\phi$$

justification: Lift $L = g\cos\phi$,
centrifugal force $v^2/r = L\sin\phi = g\tan\phi$.
Use $r = vT/(2\pi)$ where T is the time for a 360°turn:

$$\phi = \arctan\left(\frac{2\pi v}{gT}\right)$$

```
TAS <- 150        ## assumed airspeed, m/s
gravity <- 9.8    ## m/(s^2)
Time <- 60 * 3    ## time in seconds
## the following prints the required roll angle in degrees
atan (2 * pi * TAS / (gravity * Time)) * 180 / pi
## [1] 28.1149
```

Math Conventions

focus on what might seem different

Operator precedence:

- :: $ [ ] PEU: %x% (MD)(AS) and L-to-R priority if equal

R input and response:

```
:: package ref. [ggplot2::...]
$ named column [e.g., DF$X]
[] indexing -- e.g., DF[3,6]
P parentheses (above math ops)
E exponentiation [x^y]
U unary operators; e.g., [-x]
%..% special ops, incl modulus
(MD) [*/] equal priority
(AS) [+-] equal priority
```

Math Conventions

focus on what might seem different

Operator precedence:

- :: $ [ ] PEU: %x% (MD)(AS) and L-to-R priority if equal
- "1:5 * 2" ":" has precedence

R input and response:

```
1:5 * 2 # 1:10 or 2,4,6...?
## [1]  2  4  6  8 10
```

Math Conventions

focus on what might seem different

Operator precedence:

- :: $ [ ] PEU: %x% (MD)(AS) and L-to-R priority if equal
- "1:5 * 2" ":" has precedence
- logic: ! (& &&) (| ||) xor

R input and response:

```
## R uses TRUE and FALSE, and
## recognizes abbrev. T and F.
## "||" is "OR"; "&&" is "AND"
T || F && F   # && has precedence
## [1] TRUE
```

Math Conventions

focus on what might seem different

Operator precedence:

- :: $ [ ] PEU: %x% (MD)(AS) and L-to-R priority if equal
- "1:5 * 2" ":" has precedence
- logic: ! (& &&) (| ||) xor
- (& is vectorized "AND"; && is single-valued "AND")

R input and response:

```
## R uses TRUE and FALSE, and
## recognizes abbrev. T and F.
## "||" is "OR"; "&&" is "AND"
T || F && F  # && has precedence
## [1] TRUE
```

Math Conventions

focus on what might seem different

Operator precedence:

- :: $ [ ] PEU: %x% (MD)(AS) and L-to-R priority if equal
- "1:5 * 2" ":" has precedence
- logic: ! (& &&) (| ||) xor
- (& is vectorized "AND"; && is single-valued "AND")

Operators to note:

exponentiation: ^ (accepts **)

R input and response:

```
3^2; 3**2
## [1] 9
## [1] 9
```

Math Conventions

focus on what might seem different

Operator precedence:

- :: $ [ ] PEU: %x% (MD)(AS) and L-to-R priority if equal
- "1:5 * 2" ":" has precedence
- logic: ! (& &&) (| ||) xor
- (& is vectorized "AND"; && is single-valued "AND")

Operators to note:

exponentiation: ^ (accepts **)

modulus: %% (matrix: %x%)

R input and response:

```
27 %% 6
## [1] 3
```

Math Conventions

focus on what might seem different

Operator precedence:

- :: $ [ ] PEU: %x% (MD)(AS) and L-to-R priority if equal
- "1:5 * 2" ":" has precedence
- logic: ! (& &&) (| ||) xor
- (& is vectorized "AND"; && is single-valued "AND")

Operators to note:

exponentiation: ^ (accepts **)
modulus: %% (matrix: %x%)
integer division: %/%

R input and response:

```
b <- 5.3 %/% 2.6; print(b)
## [1] 2
## weird: b is not an integer!
is.integer(b); b == as.integer(2)
## [1] FALSE
## [1] TRUE
```

Math Conventions

focus on what might seem different

Operator precedence:

- :: $ [ ] PEU: %x% (MD)(AS) and L-to-R priority if equal
- "1:5 * 2" ":" has precedence
- logic: ! (& &&) (| ||) xor
- (& is vectorized "AND"; && is single-valued "AND")

Operators to note:

exponentiation: ^ (accepts **)
modulus: %% (matrix: %x%)
integer division: %/%
define vector: c(...)

R input and response:

```
x <- c(1,2,3); print(x)
## [1] 1 2 3
```

Math Conventions

focus on what might seem different

Operator precedence:

- :: $ [ ] PEU: %x% (MD)(AS) and L-to-R priority if equal
- "1:5 * 2" ":" has precedence
- logic: ! (& &&) (| ||) xor
- (& is vectorized "AND"; && is single-valued "AND")

Operators to note:

exponentiation: ^ (accepts **)
modulus: %% (matrix: %x%)
integer division: %/%
define vector: c(...)
test if element present: %in%

R input and response:

```
a <- c("alpha", "beta", "gamma")
c("gamma", "eta") %in% a
## [1]  TRUE FALSE
```

Math Conventions

focus on what might seem different

Operator precedence:

- :: $ [ ] PEU: %x% (MD)(AS) and L-to-R priority if equal
- "1:5 * 2" ":" has precedence
- logic: ! (& &&) (| ||) xor
- (& is vectorized "AND"; && is single-valued "AND")

Operators to note:

exponentiation: ^ (accepts **)
modulus: %% (matrix: %x%)
integer division: %/%
define vector: c(...)
test if element present: %in%
equality test: '==', not '='

R input and response:

```
## for assignment,
## avoid a = b; use a <- b
## <- has precedence over =,
## Can lead to problems.
## Exception: Use = for assigning
## in functions: plot(col='red')
```

Math Conventions

focus on what might seem different

Operator precedence:

- :: $ [ ] PEU: %x% (MD)(AS) and L-to-R priority if equal
- "1:5 * 2" ":" has precedence
- logic: ! (& &&) (| ||) xor
- (& is vectorized "AND"; && is single-valued "AND")

Operators to note:

exponentiation: ^ (accepts **)
modulus: %% (matrix: %x%)
integer division: %/%
define vector: c(...)
test if element present: %in%
equality test: '==', not '='
missing: '+=', '++', etc.

R input and response:

```
## must use forms like:
i <- i + 1
```

VECTOR OPERATIONS

Vector Arithmetic:

- Loops are seldom needed: Most functions work vectorized.

R input and response:

```
round(sin(1:4 / 8 * 2 * pi), 3)  # round to 3 digits
## [1] 0.707 1.000 0.707 0.000
```

VECTOR OPERATIONS

Vector Arithmetic:

- Loops are seldom needed: Most functions work vectorized.
- If vector operations use different-length vectors, the shorter one will be recycled.

R input and response:

```
a <- 2*1:10; a <- a + 1:2; print (a)
##  [1]  3  6  7 10 11 14 15 18 19 22
```

VECTOR OPERATIONS

Vector Arithmetic:

- Loops are seldom needed: Most functions work vectorized.
- If vector operations use different-length vectors, the shorter one will be recycled.
- Vector logic is very useful:
  As indices
  (vectors, data.frames)
  or to replace selected values:

  Data[Data$TASX < 130,] <- NA

  E.g, print each 10 s in sequence:
  a[a %% 10 == 0]

R input and response:

```
## use the data.frame RAFdata in "Ranadu"
## library(Ranadu) -- called earlier
summary(RAFdata$GGALT)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    7215    8011    8820    8657    9417    9426
## Select measurements from altitude > 9000 m
Data9000 <- RAFdata[RAFdata$GGALT > 9000, ]
summary(Data9000$GGALT)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9001    9327    9419    9354    9423    9426
```

VECTOR OPERATIONS

Vector Arithmetic:

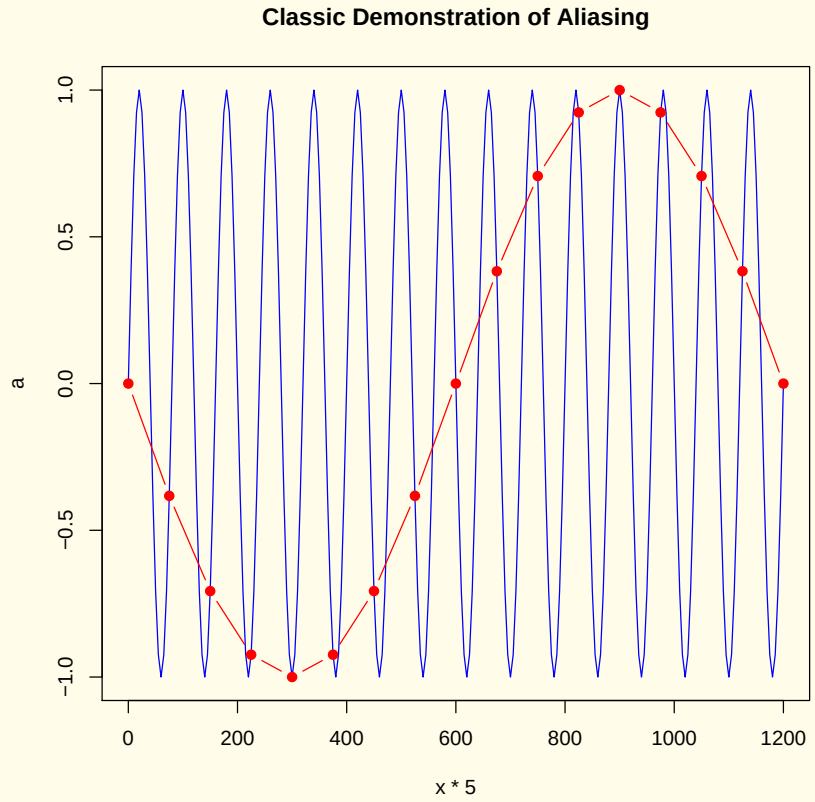
- Loops are seldom needed: Most functions work vectorized.
- If vector operations use different-length vectors, the shorter one will be recycled.
- Vector logic is very useful:
  As indices
  (vectors, data.frames)
  or to replace selected values:

  `Data[Data$TASX < 130, ] <- NA`

  E.g, print each 10 s in sequence:
  `a[a %% 10 == 0]`

R input and response:

```
a <- sin((x <- 0:240) * pi/8)   # period is 80 s
r <- (x*5) %% 75 == 0           # sample at 75 s
plot(x*5, a, type='l', col='blue')
lines(x[r]*5, a[r], type='b', pch=19, col='red')
title("Classic Demonstration of Aliasing")
```



USING VARIABLES

Variables can hold many things, allowing you to organize your work

- text, vectors, data-frames, arrays, matrices, lists, ...
- fit results
- plot characteristics

R input and response:

```
## some examples: (try these)
Title <- "Plot #1"
xlimits <- c(-1, 1)
Data <- data.frame(Time = c(1, 2, 3, 4, 5),
    Pressure = c(700, 600, 500, 400, 300))
## can put many different things in a
## list; e.g., items needed for a plot
plot1List <- list("Fig1", Data, xlimits,
    Title)  ## try printing this
```

USING VARIABLES

Variables can hold many things, allowing you to organize your work

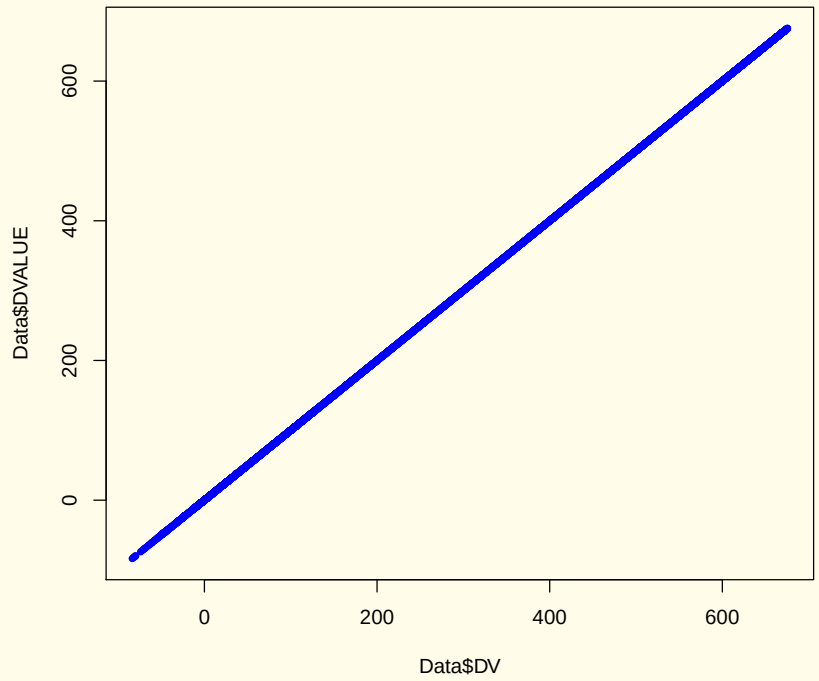
- text, vectors, data-frames, arrays, matrices, lists, ...
- fit results
- plot characteristics

Examples

- Create data-frames to hold data for plots.
- Include new variables in the relevant data-frames.
- When fitting, save the results in unique variables.

R input and response:

```
Data <- getNetCDF(fname, varNames)
## define alternate DVALUE using GGALTB,
## alt to GGALT
Data["DV"] <- Data$GGALTB - Data$PALT
fname
## [1] "/Data/CONTRAST/CONTRASTrf17.nc"
names(Data)[2:6]
## [1] "GGALTB" "GGALT"  "PALT"   "DVALUE" "DV"
plot(Data$DV, Data$DVALUE, type = "p", pch = 20,
    col = "blue")
```



USING VARIABLES

Variables can hold many things, allowing you to organize your work

- ▶text, vectors, data-frames, arrays, matrices, lists, ...
- ▶fit results
- ▶plot characteristics

Examples

- ▶Create data-frames to hold data for plots.
- ▶Include new variables in the relevant data-frames.
- ▶When fitting, save the results in unique variables.

R input and response:

Exercise: Partition the data by GGQUAL

This will show that the standard deviation for GGQUAL == 5 (highest quality) is much smaller.

USING VARIABLES

Variables can hold many things, allowing you to organize your work

- ▶text, vectors, data-frames, arrays, matrices, lists, ...
- ▶fit results
- ▶plot characteristics

Examples

- ▶Create data-frames to hold data for plots.
- ▶Include new variables in the relevant data-frames.
- ▶When fitting, save the results in unique variables.

R input and response:

```
## lm() is linear-model fit
fit1 <- lm (GGALTB ~ GGALT, data=Data)
summary(fit1)
##
## Call:
## lm(formula = GGALTB ~ GGALT, data = Data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.77737 -0.00101 -0.00004  0.00094  0.37749
##
## Coefficients:
##               Estimate Std. Error    t value Pr(>|t|
## (Intercept) -1.936e-03  4.312e-04 -4.491e+00 7.14e-0
## GGALT        1.000e+00  3.539e-08  2.826e+07  < 2e-1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'
##
## Residual standard error: 0.01321 on 21359 degrees of
## Multiple R-squared:      1,Adjusted R-squared:
## F-statistic: 7.986e+14 on 1 and 21359 DF,  p-value:
coefficients(fit1)  # or summary(fit1)$coefficients
## (Intercept)       GGALT
## -0.00193629  1.00000016
```

USING VARIABLES

Variables can hold many things, allowing you to organize your work

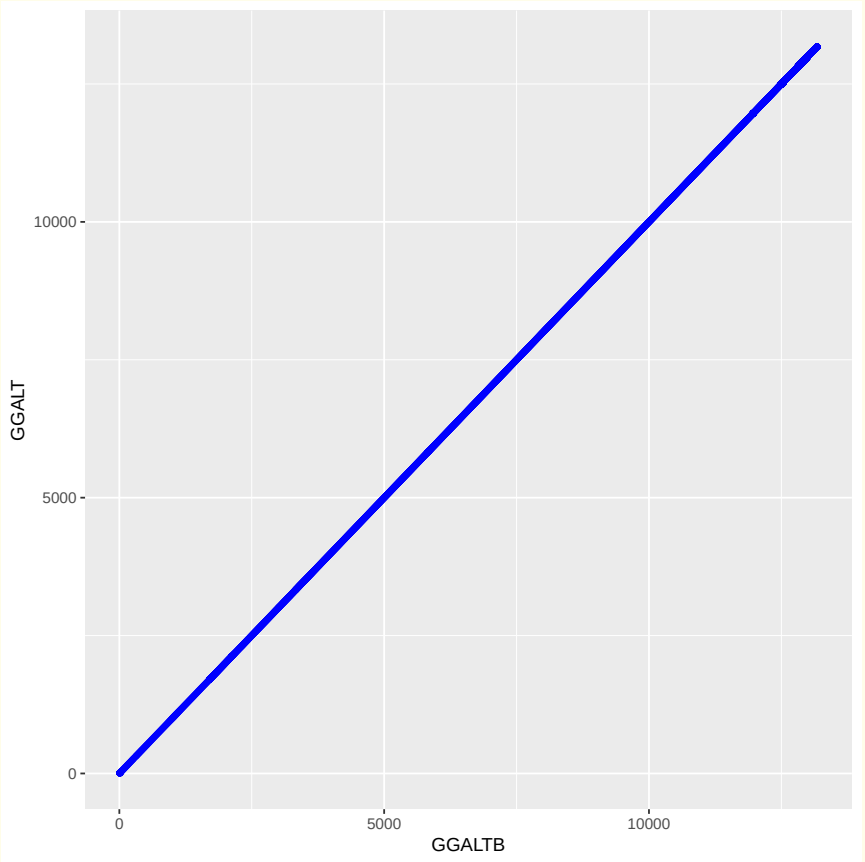
- ▶text, vectors, data-frames, arrays, matrices, lists, ...
- ▶fit results
- ▶plot characteristics

Examples

- ▶Create data-frames to hold data for plots.
- ▶Include new variables in the relevant data-frames.
- ▶When fitting, save the results in unique variables.

R input and response:

```
# nicer plot, using 'grammar of graphics'
# 'g' will be container for plot characteristics
require(ggplot2)
g <- ggplot(data=Data, aes(x=GGALTB, y=GGALT))
g <- g + geom_point(size=2, color='blue', shape=20)
# g <- g + theme_WAC()
print(g)
```



READY FOR NEXT TAB: Guide to 'Ranadu'

Also:

- Review
- Suggestions re 'style' and 'traps'