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
- RStudio console provides some conveniences
- Can do some simple programming interactively

Example: Roll angle for a 4-min turn

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

$$2\pi r = vT$$

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

In RStudio Console:

The Equation:

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

```
TAS <- 200
gravity <- 9.8
atan (2 * pi * TAS / (gravity * 240)) * 180 / pi
## [1] 28.11
```

Math Conventions
focus on what might seem different

Operator precedence:

- >:: $ [ ] PEU:(MD)(AS)
  "1:5 * 2" : has precedence

```r
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:(MD)(AS)
  "1:5 * 2" : has precedence
- ! (& &&) (| ||) xor

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

```r
T | F & F  # & has priority
## [1] TRUE
```

Math Conventions

focus on what might seem different

Operator precedence:

- >:: $ [ ] PEU:(MD)(AS)
  "1:5 * 2" : has precedence
- ! (& &&) (| ||) xor
- & vectorized;
  && single-valued, efficient

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

```
T | F & F  # & has priority
## [1] TRUE
```

Math Conventions
focus on what might seem different

Operator precedence:

- >:: $ [ ] PEU:(MD)(AS)
  "1:5 * 2" : has precedence
- ! (& &&) (| ||) xor
- & vectorized;
  && single-valued, efficient

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

```
T | F & F  # & has priority
## [1] TRUE
```

Operators to note:

exponentiation: ^ (accepts **)

Math Conventions
focus on what might seem different

Operator precedence:

- >:: $ [ ] PEU:(MD)(AS)
  "1:5 * 2" : has precedence
- ! (& &&) (| ||) xor
- & vectorized;
  && single-valued, efficient

Operators to note:

exponentiation: ^ (accepts **)
modulus: %% (mention %x%)

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

```
T | F & F   # & has priority
## [1] TRUE
```

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

Math Conventions

focus on what might seem different

Operator precedence:

- >:: $ [ ] PEU:(MD)(AS)
  "1:5 * 2" : has precedence
- ! (& &&) (| ||) xor
- & vectorized;
  && single-valued, efficient

Operators to note:

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

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

```
T | F & F   # & has priority
## [1] TRUE
```

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

```
b <- 5.3 %/% 2.6; b
## [1] 2
is.integer(b); as.integer (b)
## [1] FALSE
## [1] 2
```

Math Conventions

focus on what might seem different

Operator precedence:

- >:: $ [ ] PEU:(MD)(AS)
  "1:5 * 2" : has precedence
- ! (& &&) (| ||) xor
- & vectorized;
  && single-valued, efficient

Operators to note:

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

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

```
T | F & F   # & has priority
## [1] TRUE
```

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

```
b <- 5.3 %/% 2.6; b
## [1] 2
is.integer(b); as.integer (b)
## [1] FALSE
## [1] 2
```

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

Math Conventions

focus on what might seem different

Operator precedence:

- >:: $ [ ] PEU:(MD)(AS)
  "1:5 * 2" : has precedence
- ! (& &&) (| ||) xor
- & vectorized;
  && single-valued, efficient

Operators to note:

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

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

```r
T | F & F   # & has priority
## [1] TRUE
```

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

```r
b <- 5.3 %/% 2.6; b
## [1] 2
is.integer(b); as.integer (b)
## [1] FALSE
## [1] 2
```

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

Math Conventions

focus on what might seem different

Operator precedence:

- >:: $ [ ] PEU:(MD)(AS)
  "1:5 * 2" : has precedence
- ! (& &&) (| ||) xor
- & vectorized;
  && single-valued, efficient

Operators to note:

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

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

```r
T | F & F   # & has priority
## [1] TRUE
```

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

```r
b <- 5.3 %/% 2.6; b
## [1] 2
is.integer(b); as.integer (b)
## [1] FALSE
## [1] 2
```

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

VECTOR OPERATIONS

Vector Arithmetic:

- Loops seldom needed: Most functions work vectorized. Very useful; cf. Ranadu/R/AirTemperature.R

R input and response:

```
a <- 1:10; a[1:5] <- a[6:10]; a
##  [1]  6  7  8  9 10  6  7  8  9 10
```

VECTOR OPERATIONS

Vector Arithmetic:

- Loops seldom needed: Most functions work vectorized. Very useful; cf. Ranadu/R/AirTemperature.R
- If vector operations use different-length vectors, the shorter one will be recycled.

R input and response:

```
a <- 1:10; a[1:5] <- a[6:10]; a
##  [1]  6  7  8  9 10  6  7  8  9 10
```

```
2*a; a <- a + 1:2; print (a)
##  [1] 12 14 16 18 20 12 14 16 18 20
##  [1]  7  9  9 11 11  8  8 10 10 12
```

VECTOR OPERATIONS

Vector Arithmetic:

- Loops seldom needed: Most functions work vectorized. Very useful; cf. Ranadu/R/AirTemperature.R
- If vector operations use different-length vectors, the shorter one will be recycled.
- Logical tests are very useful:
  As indices
  (vectors, data.frames)
  To replace select values:

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

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

R input and response:

```
a <- 1:10; a[1:5] <- a[6:10]; a
##  [1]  6  7  8  9 10  6  7  8  9 10
```

```
2*a; a <- a + 1:2; print (a)
##  [1] 12 14 16 18 20 12 14 16 18 20
##  [1]  7  9  9 11 11  8  8 10 10 12
```

```
Data <- data.frame("Time"=1:4)
Data["ATX"]=c(10.3, 10.6, 10.9, 11.2)
Data["TASX"] <- c(131.3, 129.8, 132.9, 135.6)
Valid <- (Data$TASX > 130.); Valid
## [1]  TRUE FALSE  TRUE  TRUE
DataValid <- Data[Valid, ]; DataValid
##   Time  ATX  TASX
## 1    1 10.3 131.3
## 3    3 10.9 132.9
## 4    4 11.2 135.6
```

VECTOR OPERATIONS

Vector Arithmetic:

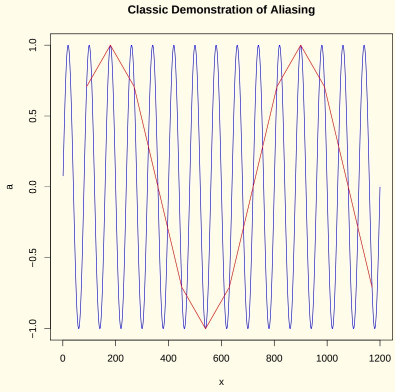
- Loops seldom needed: Most functions work vectorized. Very useful; cf. Ranadu/R/AirTemperature.R
- If vector operations use different-length vectors, the shorter one will be recycled.
- Logical tests are very useful:
  As indices
  (vectors, data.frames)
  To replace select values:

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

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

R input and response:

```
a <- sin((x<-1:1200)*pi/40)  # period is 80 s
r <- 1:1200%%90==0           # sample at 90 s
plot(x,a,type='l', col='blue')
lines(x[r],a[r], col='red')
title("Classic Demonstration of Aliasing")
```



USING VARIABLES

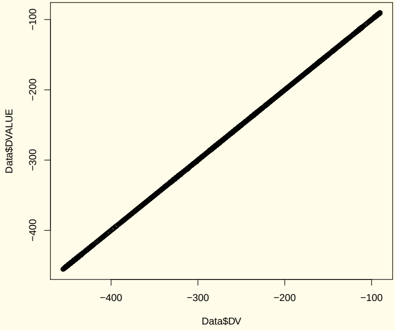
- Variables can hold many things, allowing you to organize your work:
  ▶text, vectors, data-frames, arrays, matrices, lists, ...
  ▶fit results
  ▶plot characteristics
- Suggestion: Make use of this wherever possible
  ▶Create data-frames to hold data for plots.
  ▶Include new variables in the relevant data-frames.
  ▶When fitting, save the results in unique variables.

USING VARIABLES

- Variables can hold many things, allowing you to organize your work:
  ▶text, vectors, data-frames, arrays, matrices, lists, ...
  ▶fit results
  ▶plot characteristics
- Suggestion: Make use of this wherever possible
  ▶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:

```
## [1] "/home/Data/DEEPWAVE/DEEPWAVErf15.nc"
Data <- getNetCDF(fname, varNames)
Data["DV"] <- Data$GGALTB - Data$PALT
names(Data)[2:6]
## [1] "GGALTB" "GGALT"  "PALT"   "DVALUE" "DV"
mean(Data$GGALTB - Data$GGALT, na.rm = TRUE)
## [1] 9.539e-05
sd(Data$GGALTB - Data$GGALT, na.rm = TRUE)
## [1] 0.01711
plot(Data$DV, Data$DVALUE)
```



USING VARIABLES

- Variables can hold many things, allowing you to organize your work:
  ▶text, vectors, data-frames, arrays, matrices, lists, ...
  ▶fit results
  ▶plot characteristics
- Suggestion: Make use of this wherever possible
  ▶Create data-frames to hold data for plots.
  ▶Include new variables in the relevant data-frames.
  ▶When fitting, save the results in unique variables.

Exercise: Partition the data by GGQUAL

This will show that the difference for GGQUAL != 5 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
- Suggestion: Make use of this wherever possible
  ▶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:

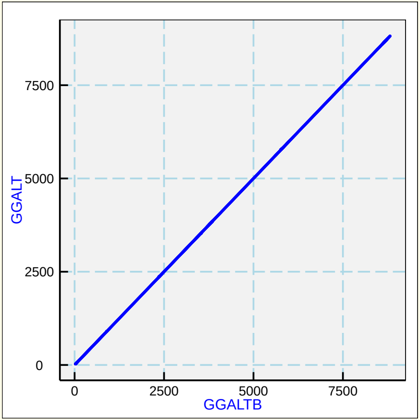
```
fit1 <- lm (GGALTB ~ GGALT, data=Data)
names(fit1)
##  [1] "coefficients"  "residuals"     "effects"
##  [5] "fitted.values" "assign"        "qr"
##  [9] "xlevels"       "call"          "terms"
summary(fit1)
##
## Call:
## lm(formula = GGALTB ~ GGALT, data = Data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.6181 -0.0010 -0.0001  0.0009  0.9111
##
## Coefficients:
##             Estimate Std. Error  t value Pr(>|t|)
## (Intercept) 1.99e-04   2.21e-04 9.00e-01     0.37
## GGALT       1.00e+00   4.28e-08 2.34e+07   <2e-16 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'
##
## Residual standard error: 0.0171 on 16022 degrees of
## Multiple R-squared:     1, Adjusted R-squared:     1
## F-statistic: 5.47e+14 on 1 and 16022 DF,  p-value: <
coefficients(fit1)    #or summary(fit1)$coefficients
## (Intercept)       GGALT
##   0.0001992   1.0000000
```

USING VARIABLES

- Variables can hold many things, allowing you to organize your work:
  ▶text, vectors, data-frames, arrays, matrices, lists, ...
  ▶fit results
  ▶plot characteristics
- Suggestion: Make use of this wherever possible
  ▶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)
```



NEXT TIME: Guide to 'Ranadu'

Also:

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