STA 250/MTH 342 Intro to Mathematical Statistics
Lab Session 1 / Jan 12, 2015 / Handout

This lab work is intended to be an introduction to the software **R**. What follows is a description of the basic functionalities of **R**, along with a series of tasks that you'd have to perform.

See: `https://stat.duke.edu/courses/Spring15/sta250/labs/` for links to source code and data. Submit lab solutions via email to: `sta250@stat.duke.edu`. Any plots should be included in postscript form as attachments. The email subject must be "STA250 ..." with "..." replaced by your name.

1: What is R. **R** is a programming language and a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX(-like) platforms, Windows and MacOS.

- An interpreter-based programming, graphics and statistics package.
- Free, stable, can be extended.
- Can easily perform standard statistical and numerical analysis.
- Can be programmed to handle non-standard cases.
- For complex tasks, it is often used as a first step to interface with `C` or `FORTRAN`.
- Almost all new statistical methodology is published with a ready-to-use package built with **R**.

2: Launching R.

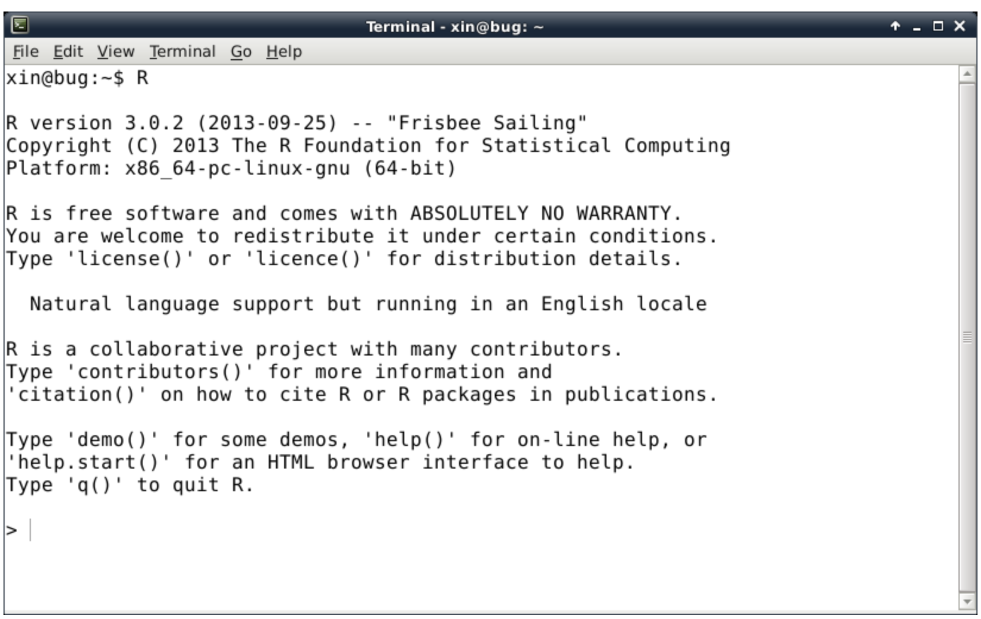


Figure 1: The welcome message of R environment on Linux system.

- On `UNIX` type workstations, simply type **R** at the prompt.
- On windows open through `Start -> All Programs -> R -> R 3.1.x`
- We will talk about the `UNIX` version, and the Windows version is almost identical.
- Once launched, **R** prints a welcome message (see Figure 1) and gives a command prompt `>`.

3: Entering Commands.

- You can directly type commands at the prompt (followed by Enter):

```
> 2 + 3
[1] 5
> 5 * pi
[1] 15.70796
> rnorm(20)
 [1]  0.04003418 -1.41518671  0.34850029  0.49295274  0.53115520 -0.30467104
 [7]  0.60536058  0.04259377 -0.63919160 -0.72791944  0.27232650  1.25095977
[13] -0.35750336  1.28684279 -2.85139690 -0.07714409 -0.27231562  0.08774009
[19] -0.90930971  0.46362665
```

- If you hit enter before completely entering a command, you will get a `+` continuation prompt. You must complete the command or type `^C` (`Esc` in MSW) to continue.

```
> 3 +
+ 5;
[1] 8
```

  It's recommended (but not required) that you end each statement with a semicolon, to make your intentions clear to yourself and other human readers.

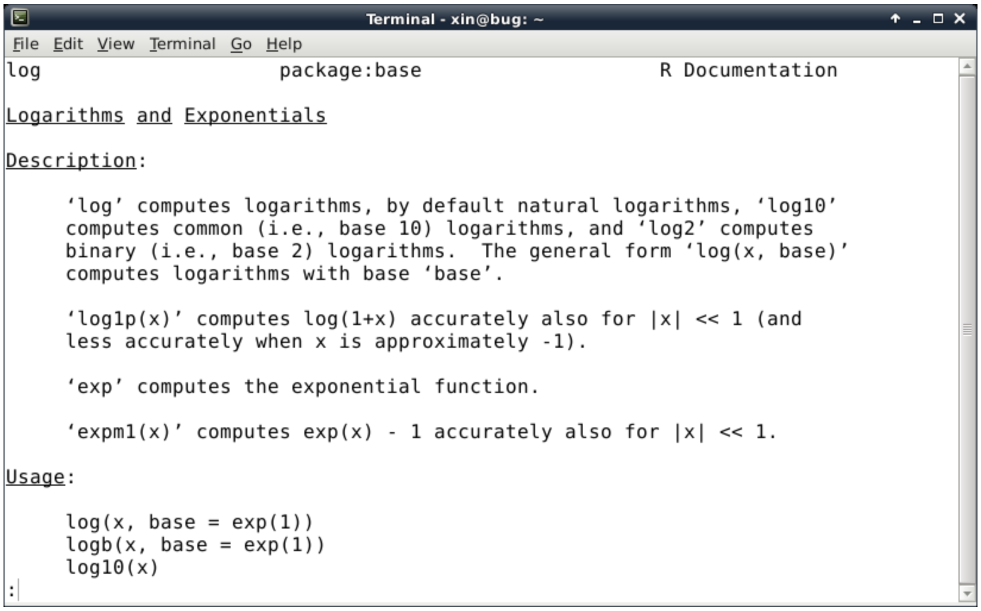
- All arithmetic operations are represented via standard symbols (`+ - * /`) and have the usual order of precedence.

```
> 3 + 4 * 2;
[1] 11
```

- All common functions are represented by their usual names.

```
> sin(pi / 6);
[1] 0.5
> exp(log(2) + log(3));
[1] 6
> atan(Inf)/pi;
[1] 0.5
```

4: Reading Documentation. Documentation is important for learning **R**. Type `?log`, or `?"log"`, for example, to see the help page of the function `log` (see Fig. 2). Why quoting? Because sometimes we need the help page of operators. Try `?"+"` and `?+` separately. Press `q` to quit the help page.



```
log                    package:base                    R Documentation

Logarithms and Exponentials

Description:

     'log' computes logarithms, by default natural logarithms, 'log10'
     computes common (i.e., base 10) logarithms, and 'log2' computes
     binary (i.e., base 2) logarithms.  The general form 'log(x, base)'
     computes logarithms with base 'base'.

     'log1p(x)' computes log(1+x) accurately also for |x| << 1 (and
     less accurately when x is approximately -1).

     'exp' computes the exponential function.

     'expm1(x)' computes exp(x) - 1 accurately also for |x| << 1.

Usage:

     log(x, base = exp(1))
     logb(x, base = exp(1))
     log10(x)
:
```

Figure 2: Help page of the logarithm functions.

TASK 1. The function call `dbinom(x, n, p)` evaluates the **Binomial**(n, p) pdf at the value x. Evaluate **Binomial**$(165, 0.9)$ at $x = 155$.

What should you do if you forget the order of the arguments?

5: Object Type. **R** is an objected oriented environment – everything in **R** is an object, named or unnamed. The objects have a variety of types:

- `double`, e.g., `3.14`, `0`, `-2.71828`, `6.022e23`, `Inf`, `NaN`;
- `complex`, e.g., `1+3i`, `0i`;
- `character`, e.g., `"Duke University"`;
- `logical`, `T` (or equivalently, `TRUE`) and `F` (or equivalently, `FALSE`);
- `integer`, e.g., 3L;
- `list`;
- many others.

R does not distinguish atomic types (namely: double, complex, integer, logical, and character) and the corresponding vectors; the atomic type is simply a vector of length one. Vectors must have their values all of the same type. Vectors may be constructed using the "catonate" function `c()`, as shown below.

Note assignment is traditionally done in R with an arrow made up of the less-than and minus signs, like `a <- 2.3`; the same effect can be obtained with a single equals sign `a = 2.3`. The double equals sign is a test for equality, so depending on the current value of a, "`a == 2.3`" will return TRUE or FALSE (or perhaps NA... what does *that* mean?).

```
> a <- 2.3;
> a;
[1] 2.3
> typeof(a);
[1] "double"
> length(a);
[1] 1
> b <- "Duke University";
> typeof(b);
[1] "character"
> length(b);
[1] 1
> nchar(b);
[1] 15
> "Yale University" == b;
[1] FALSE
```

```
> x <- c("2.33","3.14159");
> typeof(x);
[1] "character"
> length(x);
[1] 2
> nchar(x);
[1] 4 7
> as.double(x);
[1] 2.33000 3.14159
> y <- seq(0,20,3)/10;
> y;
[1] 0.0 0.3 0.6 0.9 1.2 1.5 1.8
> as.character(y);
[1] "0"   "0.3" "0.6" "0.9" "1.2" "1.5" "1.8"
> rep(pi,3);
[1] 3.141593 3.141593 3.141593
```

TASK 2. Create a vector p with elements $0.0, 0.01, 0.02, \cdots, 1.00$. What's the function of the command `p[42]`? Of `p[54:90]`? Of `p[-11]`?

6: Matrices. A matrix is a long vector stored as a rectangular array.

```
> a <- matrix(1:12, ncol = 3, nrow = 4);
> a;
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> typeof(a);
[1] "integer"
> dim(a);
[1] 4 3
> length(a);
[1] 12
> str(a);
 int [1:4, 1:3] 1 2 3 4 5 6 7 8 9 10 ...
> dim(a) <- NULL;
```

```
> a;
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> str(a);
 int [1:12] 1 2 3 4 5 6 7 8 9 10 ...
> dim(a);
NULL
> dim(a) <- c(3,4);
> a;
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> a[1,];
[1]  1  4  7 10
> a[1,1] <- 0;
> t(a);
```

```
     [,1] [,2] [,3]
[1,]    0    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> a %*% t(a);
     [,1] [,2] [,3]
[1,]  165  186  207
[2,]  186  214  240
[3,]  207  240  270
> solve(a %*% t(a));
          [,1]      [,2]       [,3]
[1,]  3.333333 -10.00000   6.333333
[2,]-10.000000  31.50000 -20.333333
[3,]  6.333333 -20.33333  13.222222
> # matrix multiplication is associative
> solve(a %*% t(a)) %*% a %*% t(a);
              [,1]         [,2]          [,3]
[1,]  1.000000e+00 1.207923e-12  1.342926e-12
[2,] -6.252776e-13 1.000000e+00 -8.952838e-13
[3,]  5.542233e-13 6.963319e-13  1.000000e+00
> # This is a comment.
> # R ignores anything after "#".
>
```

7: Basic For-Loops.

```
> a <- 0;
> for(i in 1:100) a <- a+i;
> a;
[1] 5050
```

TASK 3. Store the first 100 elements of the vector `p` you created above as a 10×10 matrix `pp`, with the first 10 elements of `p` appearing on the first column of `pp`.

What's the function of the command `pp[42]`? Or of `pp[, 4]`? Or of `pp[8, ]`? Or of `pp[8, 4]`? Or `pp[,-1]`? Or `diag(pp)`?

8: Plotting. `plot` is the generic function to plot variables. The default use is `plot(x, y)` which plots a vector `y` against another vector `x` provided they have the same length.

```
> a <- (1:50) / 50 * 2 * pi;
> b <- sin(a);
> plot(a,b);
> plot(a,b,type="b");
> plot(a,b,type="b",col="red");
> plot(a,b,type="o",col="blue",pch=23);
> pie(1:5);
> hist(rnorm(10000),breaks=30);
```

For more options, see `?plot` and `?plot.default`. One may save the plot into files.

```
> setEPS();postscript("a.eps");
> pie(1:5);
> dev.off();
null device
          1
> bmp("a.bmp");
```

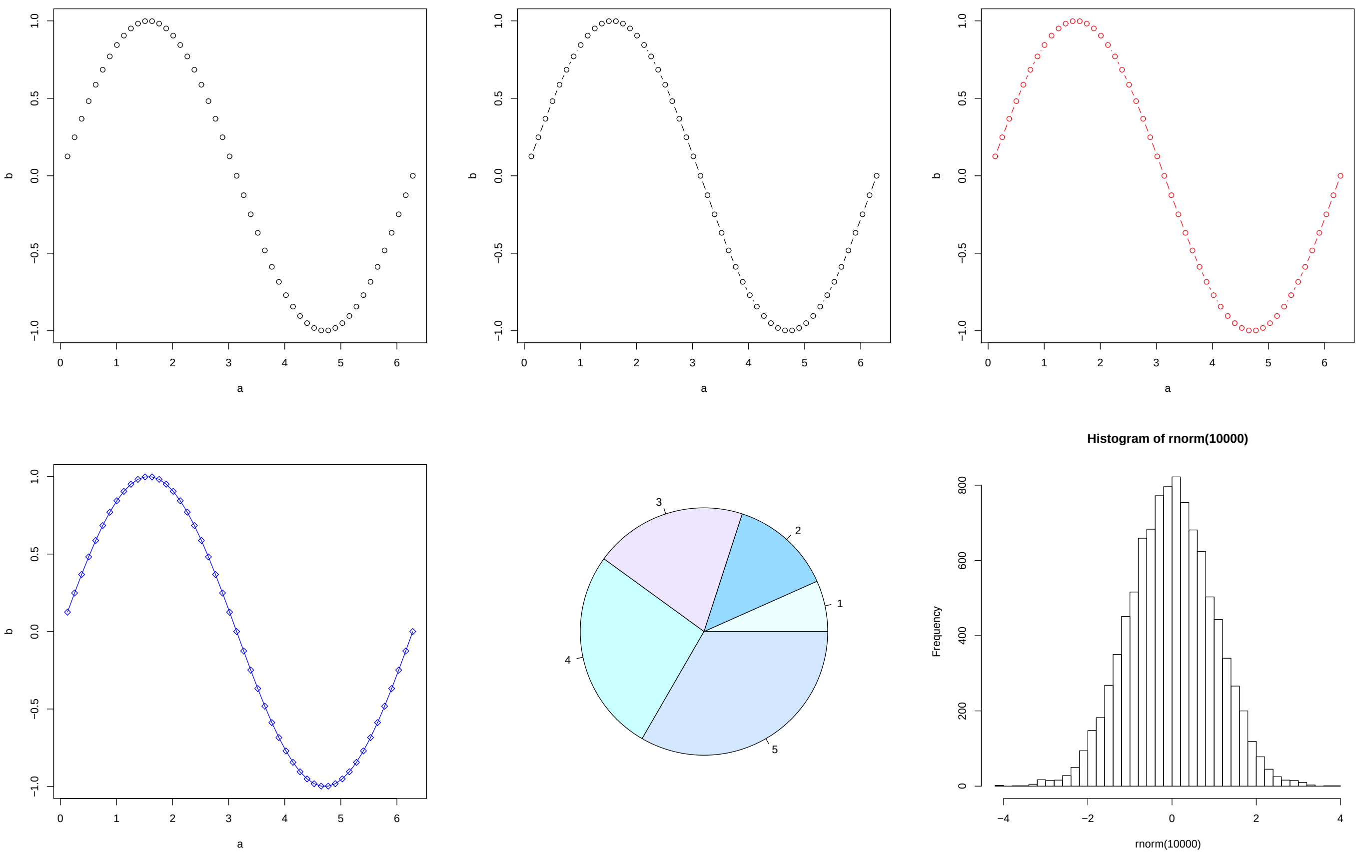


Figure 3: Some sample plots.

```
> pie(1:6);                                              1
> dev.off();                                   >
null device
```

TASK 4. Create a vector `product` that is the product of two random variables $X_1 \sim \mathsf{Bi}(309, p)$ and $X_2 \sim \mathsf{Bi}(289, p)$ for every element p of the vector `p` you created before, with observation $X_1 = 193, X_2 = 203$.

Plot `product` versus `p`.

9: Functions. A function is a special kind of **R** object that takes arguments and outputs other objects. The syntax of a function is as follows:

```
f <- function(args){
  body ...
  ...
  return(ans)
 }
```

For example,

```
Likelihood <- function(p1, p2=p1){
  L1 <- dbinom(193, 309, p1)
  L2 <- dbinom(203, 289, p2)
  return(L1 * L2)
 }
```

evaluates that general likelihood of the model in **TASK 4** at any parameter value `(p1, p2)`. What happens if you evaluate `Likelihood(0.5)` with only one argument?

TASK 5. Evaluate the above function at (0.62, 0.70), (0.66, 0.66) and (0.70, 0.62).

~~END~~