Homework 0A - Introduction to R through a Sample Session

CS/HG 124/224

March 31, 2014

The following session is intended to introduce to you some features of the R environment by using them. Many features of the system will be unfamiliar and puzzling at first, but this puzzlement will soon disappear.

1 Course Basics

Instructor Eleazar Eskin

TA Farhad Hormozdiari

Course CS/HG 124/224

Syllabus http://genetics.cs.ucla.edu/cs124/syllabus.html

Website http://genetics.cs.ucla.edu/cs124/

Other Resources http://www.r-project.org/

Homework 0A Do this session on your own (Nothing to turn in)

Homework 0B Posted on the website (Turn in : 14 April 2014)

2 Getting Started

1. Login, start your windowing system.
2. Start R as appropriate for your platform.

   ```
   R
   ```

   The R program begins, with a banner.
3. Start the HTML interface to on-line help (using a web browser available at your machine). You should briefly explore the features of this facility with the mouse. Iconify the help window and move on to the next part.

   ```
   > help.start()
   ```

3 Basic XY Plot

1. Generate two pseudo-random normal vectors of x- and y-coordinates.

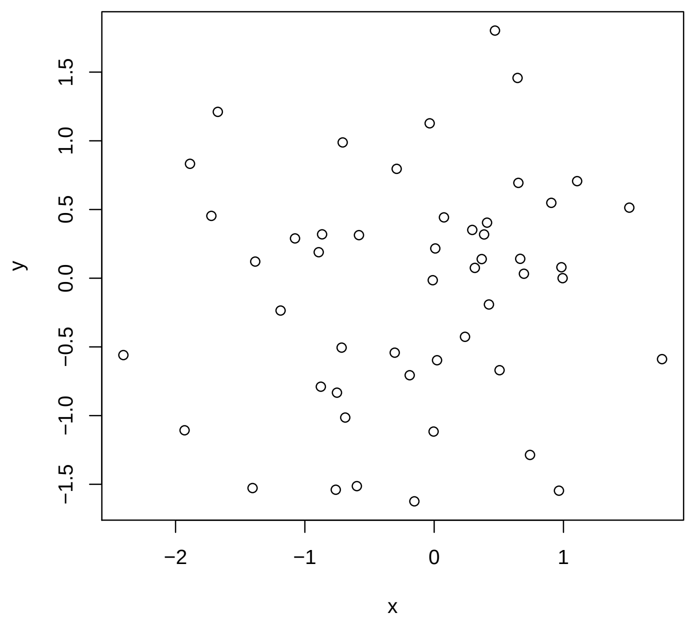
```
> x <- rnorm(50)
> y <- rnorm(x)
```

2. `rnorm` is a function which generates random numbers from normal distribution.

3. Type `?norm` you should get the help for this function and type `q` to exit.

4. Plot the points in the plane. A graphics window will appear automatically.

```
> plot(x, y)
```



5. See which R objects are now in the R workspace.

```
> ls()

[1] "x" "y"
```

6. Now generate a plot where x is drawn from normal distribution with mean of 1 and variant of 0.1, and we have $y = 2x$

7. Remove objects no longer needed. (Clean up).

```
> rm(x, y)
```

4 Simple Linear Regression

1. Make x = (1, 2, ..., 20).

```
> x <- 1:20
```

2. A 'weight' vector of standard deviations.

```
> w <- 1 + sqrt(x)/2
```

3. Make a data frame of two columns, x and y, and look at it.

```
> dummy <- data.frame(x = x, y = x + rnorm(x) * w)
> dummy
```

```
    x          y
1   1  0.4613607
2   2  3.4680682
3   3  1.5669109
4   4  6.3743211
5   5  1.8589891
6   6  4.3844714
7   7  5.1565241
8   8  6.5356371
9   9  9.8101871
10 10  6.7753995
11 11  9.8670983
12 12 12.6596021
13 13 14.8776186
14 14 12.4444900
15 15 13.8145575
16 16 17.6444689
17 17 16.3534848
18 18 17.1738212
19 19 17.3716331
20 20 23.9525071
```

4. Fit a simple linear regression and look at the analysis. With y to the left of the tilde, we are modelling y dependent on x.

```
> fm <- lm(y ~ x, data = dummy)
> summary(fm)
```

```
Call:
lm(formula = y ~ x, data = dummy)

Residuals:
    Min      1Q  Median      3Q     Max 
-2.8223 -1.1268 -0.7262  1.3763  3.7578 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -0.99930    0.88419   -1.13    0.273    
x            1.05970    0.07381   14.36 2.68e-11 ***
---
Signif. codes:  0 â

Residual standard error: 1.903 on 18 degrees of freedom
Multiple R-squared: 0.9197,	Adjusted R-squared: 0.9152 
F-statistic: 206.1 on 1 and 18 DF,  p-value: 2.677e-11
```

5. Since we know the standard deviations, we can do a weighted regression.

```
> fm1 <- lm(y ~ x, data = dummy, weight = 1/w^2)
> summary(fm1)
```

```
Call:
lm(formula = y ~ x, data = dummy, weights = 1/w^2)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.2657 -0.4641 -0.2467  0.5159  1.4269 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -0.55614    0.70572  -0.788    0.441    
x            1.01918    0.07196  14.163 3.35e-11 ***
---
Signif. codes:  0

Residual standard error: 0.7799 on 18 degrees of freedom
Multiple R-squared: 0.9177,	Adjusted R-squared: 0.9131 
F-statistic: 200.6 on 1 and 18 DF,  p-value: 3.355e-11
```

6. Make the columns in the data frame visible as variables.

```
> attach(dummy)
```

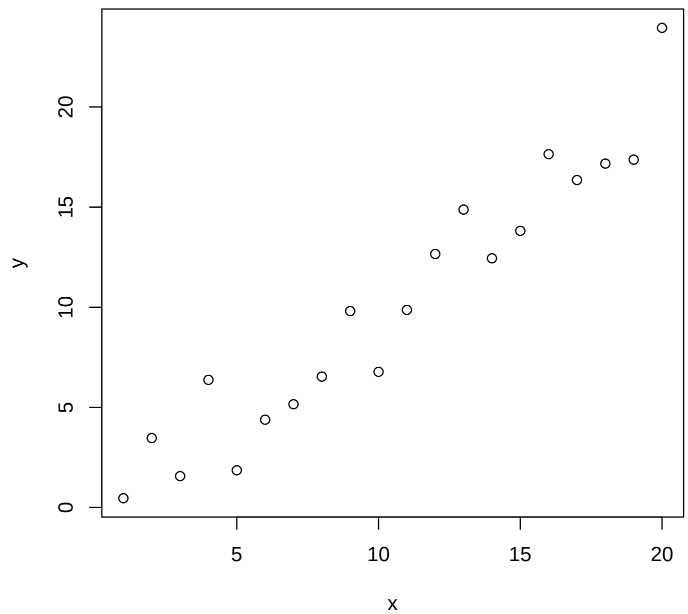
```
        The following object(s) are masked _by_ .GlobalEnv :

         x
```

7. Make a nonparametric local regression function.

```
> lrf <- lowess(x, y)
```

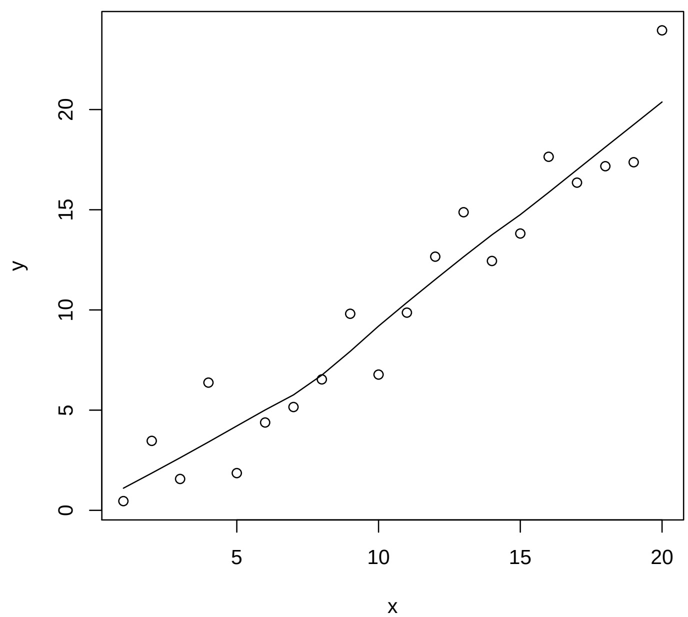
8. Standard point plot.

```
> plot(x, y)
```



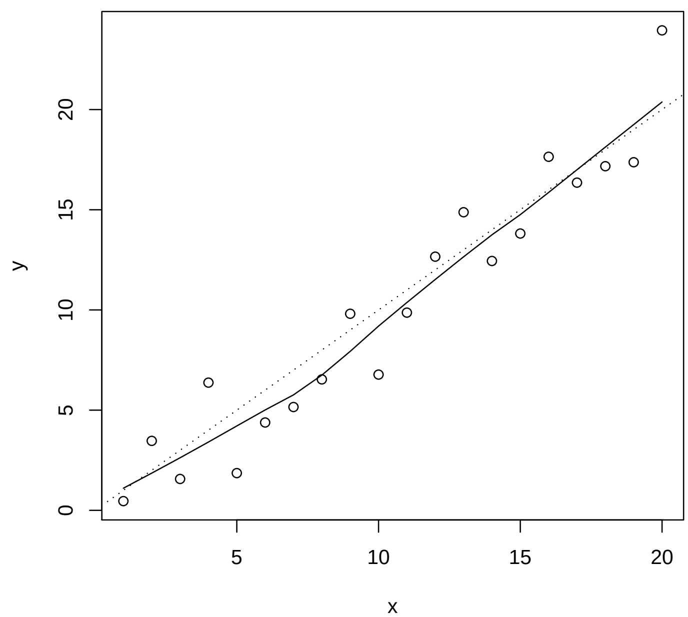
9. Add in the local regression.

```
> plot(x, y)
> lines(x, lrf$y)
```



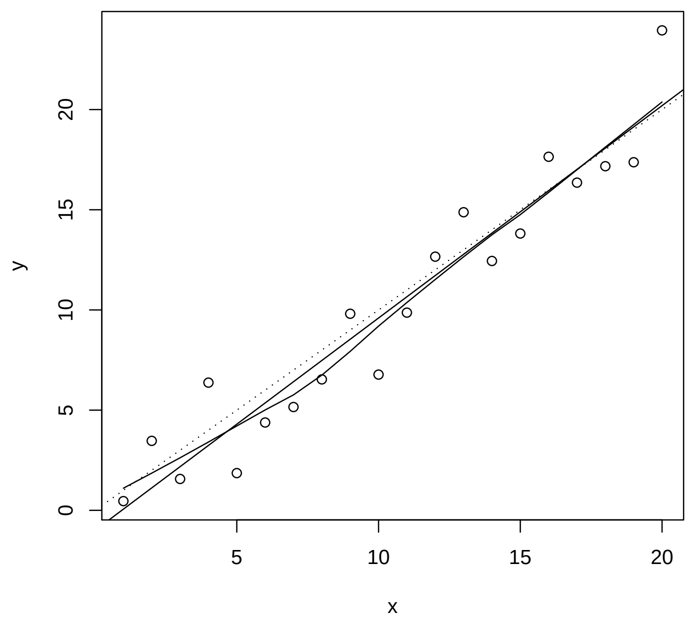
10. The true regression line: (intercept 0, slope 1).

```
> plot(x, y)
> lines(x, lrf$y)
> abline(0, 1, lty = 3)
```



11. Unweighted regression line.

```
> plot(x, y)
> lines(x, lrf$y)
> abline(0, 1, lty = 3)
> abline(coef(fm))
```



12. Weighted regression line.

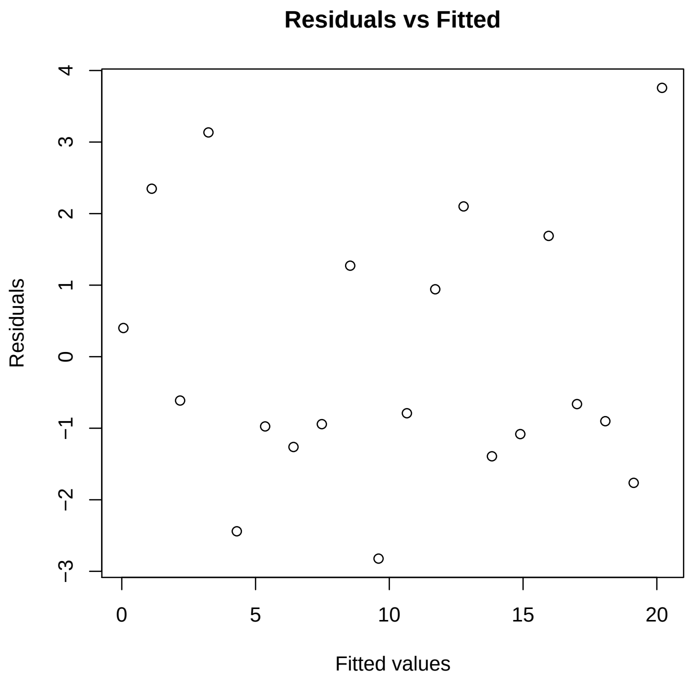
```
> plot(x, y)
> lines(x, lrf$y)
> abline(0, 1, lty = 3)
> abline(coef(fm))
> abline(coef(fm1), col = "red")
```

13. Remove data frame from the search path.

```
> detach()
```

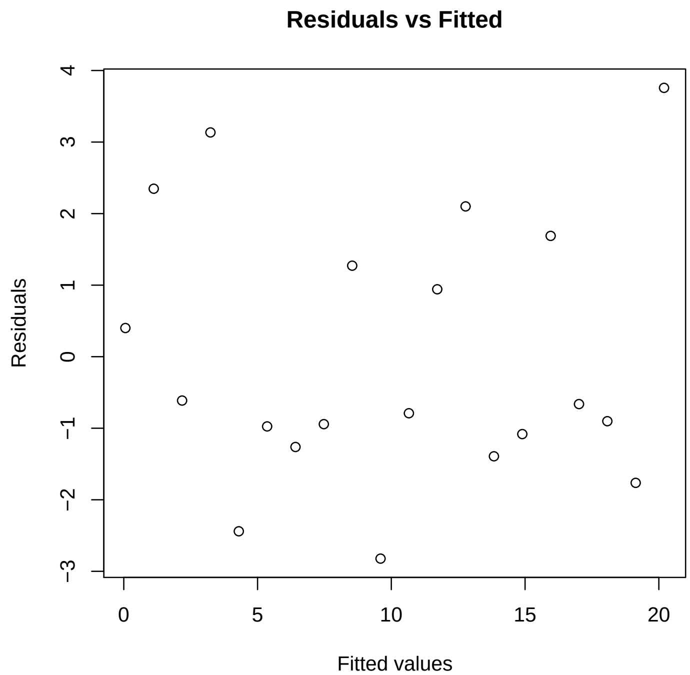
14. A standard regression diagnostic plot to check for heteroscedasticity. Can you see it?

```
> plot(fitted(fm), resid(fm), xlab = "Fitted values", ylab = "Residuals",
+     main = "Residuals vs Fitted")
```



15. A normal scores plot to check for skewness, kurtosis and outliers. (Not very useful here.)

```
> plot(fitted(fm), resid(fm), xlab = "Fitted values", ylab = "Residuals",
+     main = "Residuals vs Fitted")
> qqnorm(resid(fm), main = "Residuals Rankit Plot")
```



16. Clean up again.

```
> rm(fm, fm1, lrf, x, dummy)
```

5 Speed of Light Experiment

The next section will look at data from the classical experiment of Michaelson and Morley to measure the speed of light. This dataset is available in the morley object, but we will read it to illustrate the read.table function.

1. Get the path to the data file.

```
> filepath <- system.file("data", "morley.tab", package = "datasets")
> filepath
```

```
[1] "/usr/local/lib/R/library/datasets/data/morley.tab"
```

2. Optional. Look at the file.

```
> file.show(filepath)
```

3. Read in the Michaelson and Morley data as a data frame, and look at it. There are five experiments (column Expt) and each has 20 runs (column Run) and sl is the recorded speed of light, suitably coded.

```
> mm <- read.table(filepath)
> mm

    Expt Run Speed
001    1   1   850
002    1   2   740
003    1   3   900
004    1   4  1070
005    1   5   930
006    1   6   850
007    1   7   950
008    1   8   980
009    1   9   980
010    1  10   880
011    1  11  1000
012    1  12   980
013    1  13   930
014    1  14   650
015    1  15   760
016    1  16   810
017    1  17  1000
018    1  18  1000
019    1  19   960
020    1  20   960
021    2   1   960
022    2   2   940
023    2   3   960
024    2   4   940
025    2   5   880
026    2   6   800
027    2   7   850
028    2   8   880
029    2   9   900
030    2  10   840
031    2  11   830
032    2  12   790
033    2  13   810
034    2  14   880
035    2  15   880
036    2  16   830
037    2  17   800
038    2  18   790
039    2  19   760
040    2  20   800
041    3   1   880
042    3   2   880
```

043	3	3	880
044	3	4	860
045	3	5	720
046	3	6	720
047	3	7	620
048	3	8	860
049	3	9	970
050	3	10	950
051	3	11	880
052	3	12	910
053	3	13	850
054	3	14	870
055	3	15	840
056	3	16	840
057	3	17	850
058	3	18	840
059	3	19	840
060	3	20	840
061	4	1	890
062	4	2	810
063	4	3	810
064	4	4	820
065	4	5	800
066	4	6	770
067	4	7	760
068	4	8	740
069	4	9	750
070	4	10	760
071	4	11	910
072	4	12	920
073	4	13	890
074	4	14	860
075	4	15	880
076	4	16	720
077	4	17	840
078	4	18	850
079	4	19	850
080	4	20	780
081	5	1	890
082	5	2	840
083	5	3	780
084	5	4	810
085	5	5	760
086	5	6	810
087	5	7	790
088	5	8	810

```
089     5   9   820
090     5  10   850
091     5  11   870
092     5  12   870
093     5  13   810
094     5  14   740
095     5  15   810
096     5  16   940
097     5  17   950
098     5  18   800
099     5  19   810
100     5  20   870
```

4. Change Expt and Run into factors.

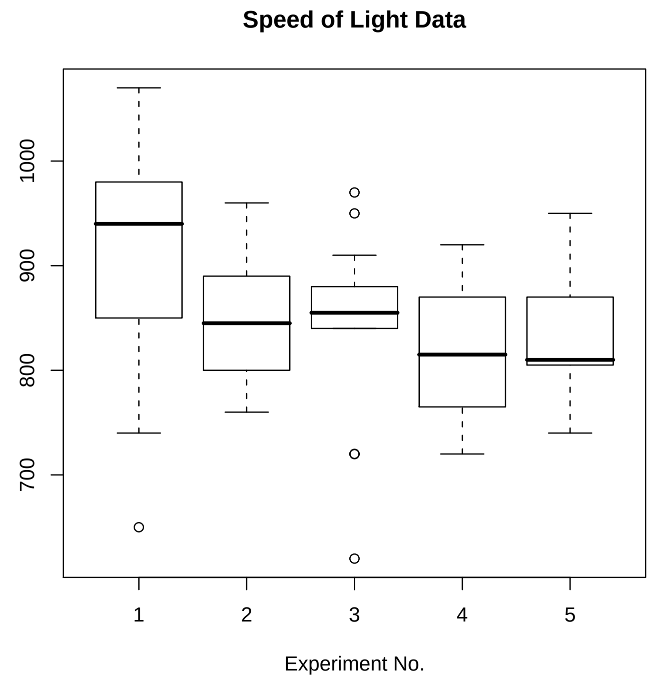
```
> mm$Expt <- factor(mm$Expt)
> mm$Run <- factor(mm$Run)
```

5. Make the data frame visible at position 3 (the default).

```
> attach(mm)
```

6. Compare the five experiments with simple boxplots.

```
> plot(Expt, Speed, main = "Speed of Light Data", xlab = "Experiment No.")
```



7. Analyze as a randomized block, with 'runs' and 'experiments' as factors.

```
> fm <- aov(Speed ~ Run + Expt, data = mm)
> summary(fm)

            Df Sum Sq Mean Sq F value   Pr(>F)
Run         19 113344    5965  1.1053 0.363209
Expt         4  94514   23629  4.3781 0.003071 **
Residuals   76 410166    5397
---
Signif. codes:  0 â
```

8. Fit the sub-model omitting 'runs', and compare using a formal analysis of variance.

```
> fm0 <- update(fm, . ~ . - Run)
> anova(fm0, fm)

Analysis of Variance Table

Model 1: Speed ~ Expt
Model 2: Speed ~ Run + Expt
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
```

```
1      95 523510
2      76 410166 19    113344 1.1053 0.3632
```

9. Clean up before moving on.

```
> detach()
> rm(fm, fm0)
```

6 Graphics in R

10. We now look at some more graphical features: contour and image plots. x is a vector of 50 equally spaced values in the interval [-pi pi]. y is the same.

```
> x <- seq(-pi, pi, len = 50)
> y <- x
```

11. f is a square matrix, with rows and columns indexed by x and y respectively, of values of the function $cos(y)/(1 + x^2)$.

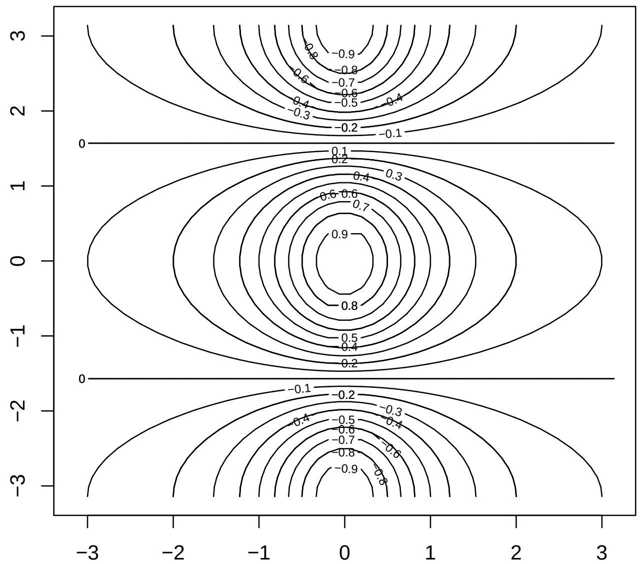
```
> f <- outer(x, y, function(x, y) cos(y)/(1 + x^2))
```

12. Save the plotting parameters and set the plotting region to âĂIJsquareâĂİ.

```
> oldpar <- par(no.readonly = TRUE)
> par(pty = "s")
```

13. Make a contour map of f; add in more lines for more detail.

```
> contour(x, y, f)
> contour(x, y, f, nlevels = 15, add = TRUE)
```

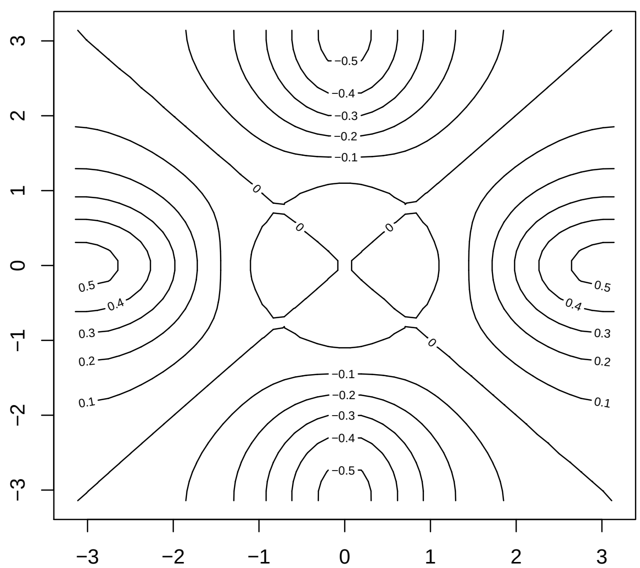


14. fa is the âĂIJasymmetric partâĂİ of f. (t() is transpose).

```
> fa <- (f - t(f))/2
```

15. Make a contour plot, ...

```
> contour(x, y, fa, nlevels = 15)
```

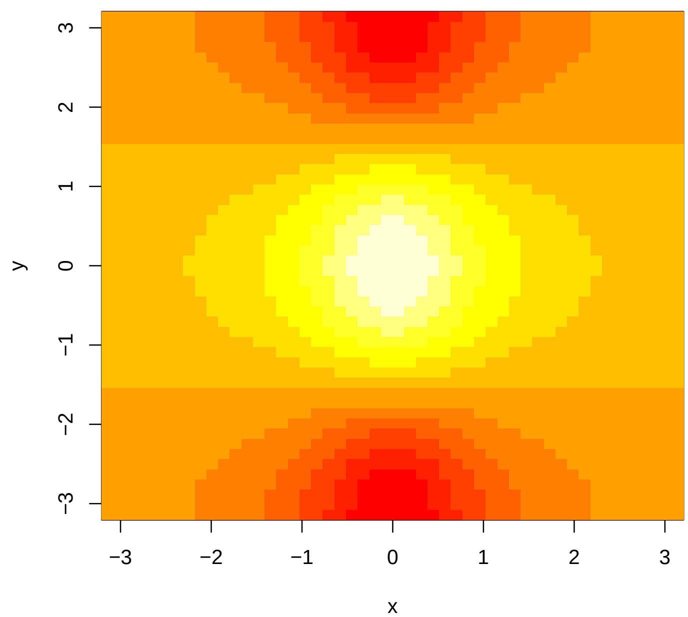


16. ... and restore the old graphics parameters.

```
> par(oldpar)
```

17. Make some high density image plots, (of which you can get hardcopies if you wish), ...

```
> image(x, y, f)
> image(x, y, fa)
```



18. ... and clean up before moving on.

```
> objects()
```

```
[1] "f"        "fa"       "filepath" "mm"       "oldpar"   "w"        "x"
[8] "y"
```

```
> rm(x, y, f, fa)
```

7 Complex Arithmetic

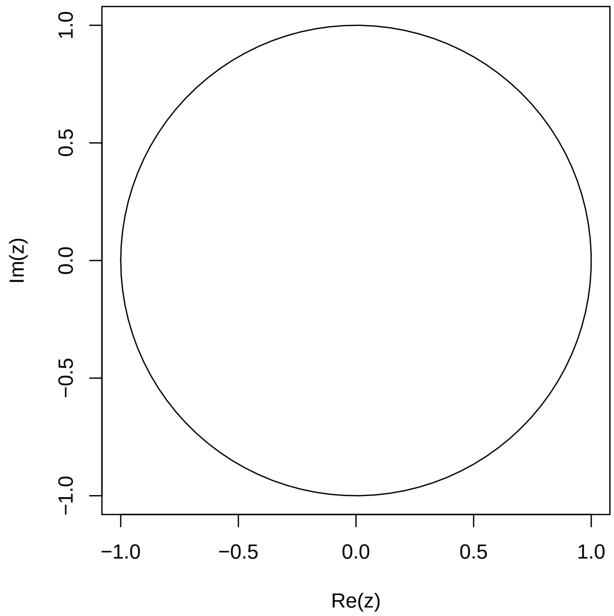
R can do complex arithmetic, also.

1. 1i is used for the complex number i.

```
> th <- seq(-pi, pi, len = 100)
> z <- exp((0+1i) * th)
```

2. Plotting complex arguments means plot imaginary versus real parts. This should be a circle.

```
> par(pty = "s")
> plot(z, type = "l")
```



3. Suppose we want to sample points within the unit circle. One method would be to take complex numbers with standard normal real and imaginary parts ...

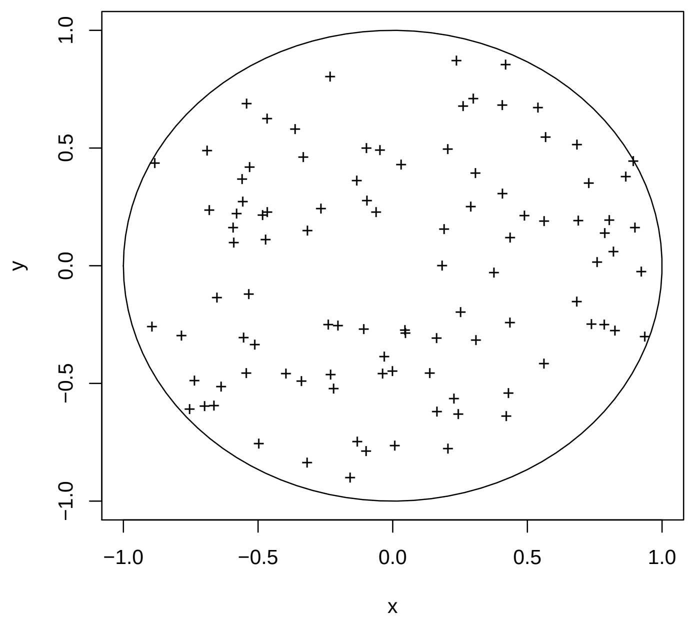
```
> w <- rnorm(100) + rnorm(100) * (0+1i)
```

4. ... and to map any outside the circle onto their reciprocal.

```
> w <- ifelse(Mod(w) > 1, 1/w, w)
```

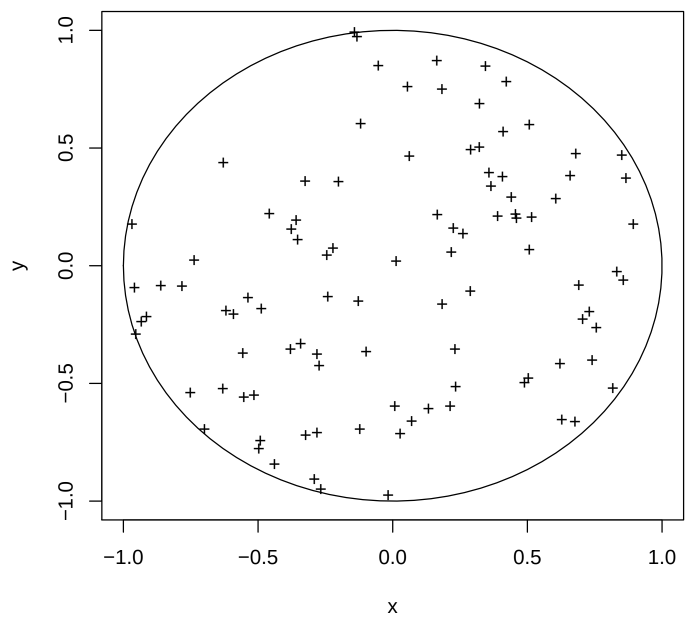
5. All points are inside the unit circle, but the distribution is not uniform.

```
> plot(w, xlim = c(-1, 1), ylim = c(-1, 1), pch = "+", xlab = "x",
+     ylab = "y")
> lines(z)
```



6. The second method uses the uniform distribution. The points should now look more evenly spaced over the disc.

```
> w <- sqrt(runif(100)) * exp(2 * pi * runif(100) * (0+1i))
> plot(w, xlim = c(-1, 1), ylim = c(-1, 1), pch = "+", xlab = "x",
+     ylab = "y")
> lines(z)
```



7. Clean up again.

```
> rm(th, w, z)
```

8. Quit the R program. You will be asked if you want to save the R workspace, and for an exploratory session like this, you probably do not want to save it.

```
> q()
```