

Lecture 2

Matrices and arrays

A matrix is just like a vector. (In fact, a matrix is a vector!)

It is easy to create a matrix:

```
> matrix(1:12,3,4)
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Notice the order in which the numbers go into the matrix: down the first column, then down the second, and so on.

```
> a=matrix(1:12,3,4)
> a[2,2]
[1] 5
```

>

Just like with a vector, it is easy to address submatrices of matrices:

```
> a[1:2,2:3]
      [,1] [,2]
[1,]    4    7
[2,]    5    8
> a[,1]
[1] 1 2 3
> a[1,]
[1] 1 4 7 10
```

If we leave one of the indexes out, it is assumed that we mean all the dimensions.

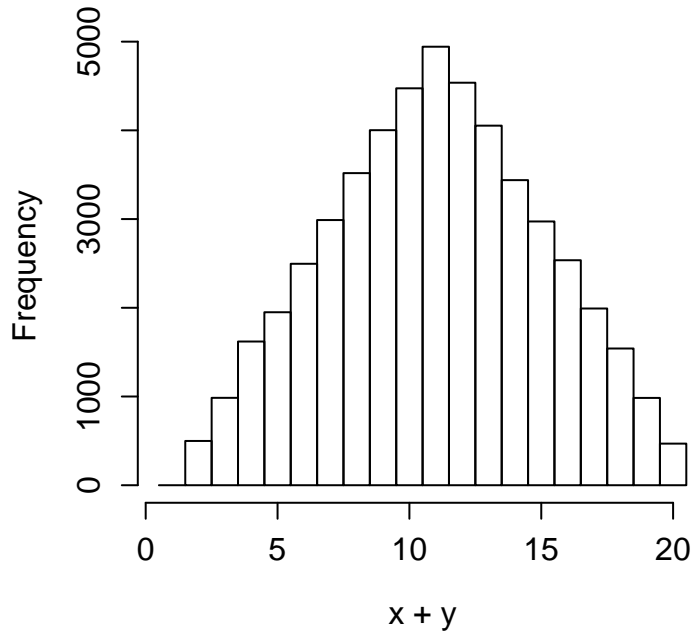
```
> a[1:2,1:2]=0
> a
      [,1] [,2] [,3] [,4]
[1,]    0    0    7   10
[2,]    0    0    8   11
[3,]    3    6    9   12
> a>8
      [,1] [,2] [,3] [,4]
[1,] FALSE FALSE FALSE TRUE
[2,] FALSE FALSE FALSE TRUE
[3,] FALSE FALSE  TRUE  TRUE
> a[a>8]=a[a>8]*3
> a
      [,1] [,2] [,3] [,4]
[1,]    0    0    7   30
[2,]    0    0    8   33
[3,]    3    6   27   36
```

>

In the exercise we saw the distribution of adding up two numbers - it was humped, but not really normal. Let us see this again:

```
> x=sample(10,50000,rep=T);y=sample(10,50000,rep=T)
> hist(x+y,breaks=0.5:20.5);v()
```

Histogram of $x + y$



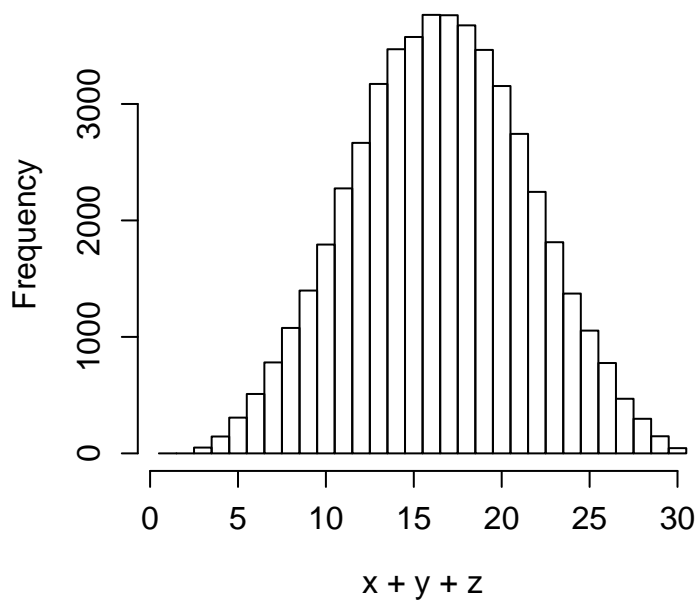
>

Very triangular! What happens if we add 3 numbers?

```
> z=sample(10,50000,rep=T)
```

```
> hist(x+y+z,breaks=0.5:30.5);v()
```

Histogram of $x + y + z$

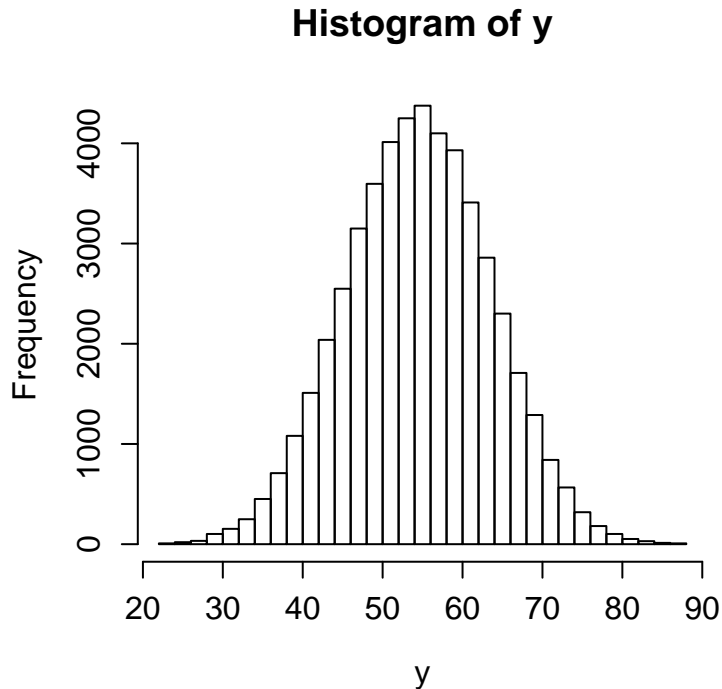


>

Somewhat more bell-shaped. What happens when we look at more numbers?

There is an easier way to do this than to redefine variables and add them up:

```
> x=matrix( sample(1:10,50000*10,rep=T), 50000,10 )
> y=apply(x,1,sum)
> length(y)
[1] 50000
> hist(y,n=30);v()
```



>

So, adding 10 numbers gives something that has normal distribution. This is actually called the central limit theorem: if we sample numbers (independently), and add them up, the distribution of the sum will approach a normal as the number of numbers goes up.

This is pretty nice: the average of the numbers is just the sum divided by n, so that means that no matter what distribution we take, the average is normally distributed.

(The distribution itself is not normal, but the mean of a sample is normally distributed.)

But let us go back to what we did:

```
> a=matrix(1:3,3,3)
> a
      [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
[3,]    3    3    3
> apply(a,1,sum)
[1] 3 6 9
> apply(a,2,sum)
[1] 6 6 6
> c(sum(a[1,]),sum(a[2,]),sum(a[3,]))
```

```
[1] 3 6 9
```

```
>
```

`apply()` takes a matrix, and applies a function to each row, or each column. We can use any function we want, as long as it can be applied to a vector.

```
> apply(a,1,mean)
```

```
[1] 1 2 3
```

```
> apply(a,2,min)
```

```
[1] 1 1 1
```

```
> apply(a,1,cumsum)
```

```
  [,1] [,2] [,3]
[1,]   1   2   3
[2,]   2   4   6
[3,]   3   6   9
```

```
> a
```

```
  [,1] [,2] [,3]
[1,]   1   1   1
[2,]   2   2   2
[3,]   3   3   3
```

```
>
```

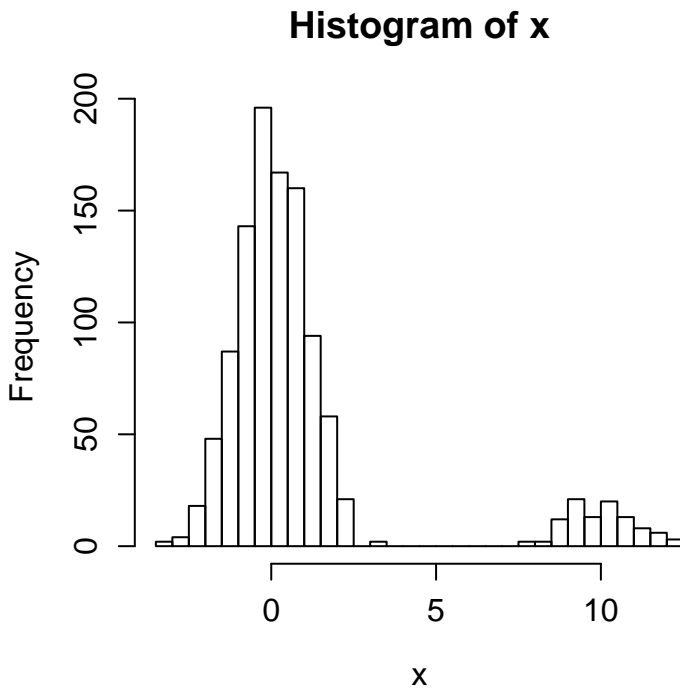
The last example shows that the result of the function can be a vector, instead of simply a number. In that case, we get a matrix back.

So, to calculate a sum of 10 numbers 50000 times, we created a matrix of 50000*10 random numbers, and then summed each 10 up.

Back to the central limit theorem. Let us try a really strange distribution. How about one that has a small hump very far away?

```
> x=c( rnorm(1000), rnorm(100,mean=10) )
```

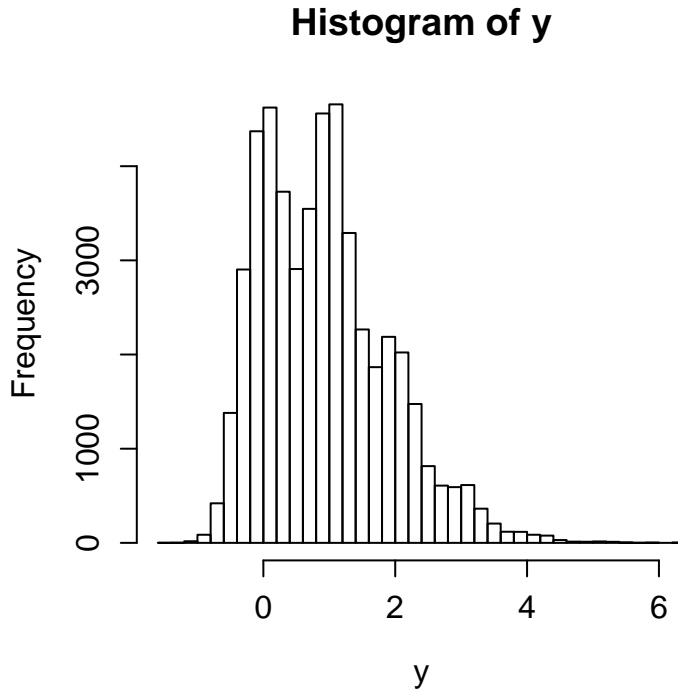
```
> hist(x,breaks=30);v()
```



>

What will the distribution of the mean of 10 numbers drawn from this distribution look like?

```
> a=matrix( sample(x,50000*10,rep=T), 50000, 10)
> y=apply(a,1,mean)
> hist(y,br=30);v()
```



>

Not very normal, yet. How many numbers does one need to add up to get a normal distribution?

Functions

How is it possible to apply something more complicated than a predefined function?

Let us try to solve the following problem: throwing 10 die, what is the chance to get 2 equal in a row?

First let us throw 10 die once:

```
> x=sample(1:6,10,rep=T);x
[1] 2 3 5 6 5 4 1 5 2 5
```

>

How would we check is two in a row are equal?

```
> x[-1]==x[-10]
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> sum(x[-1]==x[-10])
[1] 0
```

>

So, given x, the above expression is 0 if none are equal, and bigger than 0 if some are equal.

To apply it on an array we do the following:

```
> a=matrix( sample(1:6, 10*1000, rep=T), 1000, 10)
> x=apply(a,1, function(x) { sum(x[-1]==x[-10]) } )
> table(x)
```

```
 x
  0  1  2  3  4  5  6
215 342 257 144 35 6 1
```

>

So, the form is 'function(x) { an expression on x that gives the desired result }'.

Reading data

Let us look at data that I got from Philipp, saved in csv format:

```
Shell session inside TeXmacs
shell] cat Example1.csv
id,C_det,c1cCEL,c2cCEL,c3cCEL,h1cCEL,h2cCEL,h3cCEL,c,chan,
38396_at,38396_at,7.16081254,7.290660424,7.392737297,6.952853711,7.292026313,7.412756193,0.003868,
49409_i_at,49409_i_at,10.13546109,10.62638901,10.25252325,10.25231658,10.18956957,10.30671591,0.00
53038_at,53038_at,7.702094201,7.618807349,7.67603345,7.515808309,7.596549958,7.604193007,0.008735,
53434_at,,2.876379498,2.824038624,2.827237258,2.885637592,2.815227189,3.026897002,,n,
59043_at,59043_at,7.966912097,7.96113163,8.126213606,7.871402099,7.966892224,7.937320193,0.008627,
59455_at,59455_at,4.366778851,4.225955724,4.543832778,3.980990847,4.310665357,4.236237567,0.041165
61679_at,61679_at,5.049982992,5.374783109,5.268838032,5.948013417,6.596032038,5.979160843,0.889628
61971_at,61971_at,4.142098062,4.121471299,4.302961649,4.125880936,4.156195764,4.050269331,0.006094
68376_at,,4.595836702,4.650914887,4.665653487,4.606630406,4.520145401,4.541359268,,n,
80627_at,80627_at,4.911620543,4.939508403,5.216403763,4.976140892,4.98236699,5.042316751,0.000494,
81035_i_at,81035_i_at,5.906465854,5.576539595,5.462219328,4.813760076,5.008206281,5.866770711,0.17
85462_at,85462_at,4.5502712,4.529353758,4.666960935,4.738686195,4.548837862,4.561258314,0.00116,n,
88170_at,88170_at,5.678974331,5.433884455,5.604304954,5.45987842,5.427371795,5.654569802,0.003416,
89639_at,89639_at,8.379944752,8.350986947,8.380638937,8.424241477,8.385903768,8.301314198,0,n,
91383_at,91383_at,2.507656878,2.612207403,2.619023484,3.533030418,3.805764366,3.669470322,1.187648
32436_at,32436_at,8.739317645,9.187020063,8.777907612,9.126521153,9.008978032,9.307131573,0.060579
49937_at,,5.586517151,6.146771071,5.498159076,6.136033905,6.008704221,6.140507136,,n,
81837_at,,6.141651244,6.200506461,6.159321513,5.490323476,5.224234723,5.639440434,,y,
85075_at,,6.479857121,6.593030444,6.545424085,6.432517662,6.854616025,6.613686102,,y,
34099_f_at,34099_f_at,4.558208637,4.80513002,4.505734211,5.251655915,5.200594346,5.289451867,0.389
35376_f_at,,3.713291278,3.634191811,3.712004514,3.613298263,3.691586806,3.912949963,,n,
49888_f_at,49888_f_at,11.10310726,11.07483456,11.14825469,10.97350419,10.91908758,10.80596038,0.04
Interrupted TeXmacs shell
shell]
```

It has a header, and the values are comma-separated. The first column is the name of the gene, and the second column is a string.

```
> phil=read.table("Example1.csv",sep=",",head=T,row.names=1,as.is=2)
> phil[1:5,]
      C.det      c1cCEL      c2cCEL      c3cCEL      h1cCEL      h2cCEL
```

```

38396_at      38396_at  7.160813  7.290660  7.392737  6.952854  7.292026
49409_i_at  49409_i_at  10.135461 10.626389 10.252523 10.252317 10.189570
53038_at      53038_at  7.702094  7.618807  7.676033  7.515808  7.596550
53434_at      2.876379  2.824039  2.827237  2.885638  2.815227
59043_at      59043_at  7.966912  7.961132  8.126214  7.871402  7.966892
              h3cCEL      c chan
38396_at      7.412756 0.003868   y
49409_i_at    10.306716 0.007848   n
53038_at      7.604193 0.008735   n
53434_at      3.026897      NA   n
59043_at      7.937320 0.008627   y

```

>

Here are the parameters we used:

- `sep=',','` - The separator between fields. Use slash-t for tab.
- `head=T` - Use the first row as a header for the table
- `row.names=1` - Use the 1st column as a name for the rows
- `as.is=2` - Leave the 2nd column as is - it contains strings.

What is the NA on the 4th row in column c? Not Available. In some cases data is not available. R enables us to deal with these cases.

```

> a=1:10
> a[3]=NA
> a
      [1]  1  2 NA  4  5  6  7  8  9 10
> min(a)
      [1] NA
> min(a,na.rm=T)
      [1]  1
> is.na(a)
      [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE

```

>

What did we get when we read the file? A data.frame

data.frame are very much like arrays. (Later we'll see that they are actually lists)

Arrays, like vectors, contain only one data type - number, string, boolean, categorical.

A data frame can contain several types of vectors, in a table. All have to have the same length.

```

> phil$chan
      [1] y n n n y n n n y n n n n y n y y n n n n
Levels: n y

```

>

This is categorical data. It is like a string, but is stored as integers.

```

> phil[,1]

```

```

[1] "38396_at" "49409_i_at" "53038_at" "" "59043_at"
[6] "59455_at" "61679_at" "61971_at" "" "80627_at"
[11] "81035_i_at" "85462_at" "88170_at" "89639_at" "91383_at"
[16] "32436_at" "" "" "" "34099_f_at"
[21] "" "49888_f_at" "87506_at"

```

>

This is string data.

You can see that data in a data.frame can be addressed like data in a table:

> `phil[1:3,2:4]`

```

      c1cCEL  c2cCEL  c3cCEL
38396_at  7.160813  7.290660  7.392737
49409_i_at 10.135461 10.626389 10.252523
53038_at   7.702094  7.618807  7.676033

```

> `colnames(phil)`

```

[1] "C.det" "c1cCEL" "c2cCEL" "c3cCEL" "h1cCEL" "h2cCEL" "h3cCEL" "c"
[9] "chan"

```

> `rownames(phil)`

```

[1] "38396_at" "49409_i_at" "53038_at" "53434_at" "59043_at"
[6] "59455_at" "61679_at" "61971_at" "68376_at" "80627_at"
[11] "81035_i_at" "85462_at" "88170_at" "89639_at" "91383_at"
[16] "32436_at" "49937_at" "81837_at" "85075_at" "34099_f_at"
[21] "35376_f_at" "49888_f_at" "87506_at"

```

>

It is possible to add data to the data.frame:

> `phil$c3c`

```

[1] 7.392737 10.252523 7.676033 2.827237 8.126214 4.543833 5.268838
[8] 4.302962 4.665653 5.216404 5.462219 4.666961 5.604305 8.380639
[15] 2.619023 8.777908 5.498159 6.159322 6.545424 4.505734 3.712005
[22] 11.148255 9.341933

```

> `phil$big = phil$c3c > 5`

> `phil[1:3,]`

```

      C.det  c1cCEL  c2cCEL  c3cCEL  h1cCEL  h2cCEL
38396_at  38396_at  7.160813  7.290660  7.392737  6.952854  7.292026
49409_i_at 49409_i_at 10.135461 10.626389 10.252523 10.252317 10.189570
53038_at   53038_at  7.702094  7.618807  7.676033  7.515808  7.596550
      h3cCEL  c chan  big
38396_at  7.412756 0.003868  y TRUE
49409_i_at 10.306716 0.007848  n TRUE
53038_at   7.604193 0.008735  n TRUE

```

>

We can also make a matrix out of part of a data.frame:

> `a=phil[,2:4]`

> `a`

```

      c1cCEL  c2cCEL  c3cCEL
38396_at  7.160813  7.290660  7.392737
49409_i_at 10.135461 10.626389 10.252523

```

```

53038_at      7.702094  7.618807  7.676033
53434_at      2.876379  2.824039  2.827237
59043_at      7.966912  7.961132  8.126214
59455_at      4.366779  4.225956  4.543833
61679_at      5.049983  5.374783  5.268838
61971_at      4.142098  4.121471  4.302962
68376_at      4.595837  4.650915  4.665653
80627_at      4.911621  4.939508  5.216404
81035_i_at    5.906466  5.576540  5.462219
85462_at      4.550271  4.529354  4.666961
88170_at      5.678974  5.433884  5.604305
89639_at      8.379945  8.350987  8.380639
91383_at      2.507657  2.612207  2.619023
32436_at      8.739318  9.187020  8.777908
49937_at      5.586517  6.146771  5.498159
81837_at      6.141651  6.200506  6.159322
85075_at      6.479857  6.593030  6.545424
34099_f_at    4.558209  4.805130  4.505734
35376_f_at    3.713291  3.634192  3.712005
49888_f_at    11.103107 11.074835 11.148255
87506_at      9.545659  9.573302  9.341933

```

```
> apply(a,1,mean)
```

```

38396_at 49409_i_at 53038_at 53434_at 59043_at 59455_at
7.281403 10.338124 7.665645 2.842552 8.018086 4.378856
61679_at 61971_at 68376_at 80627_at 81035_i_at 85462_at
5.231201 4.188844 4.637468 5.022511 5.648408 4.582195
88170_at 89639_at 91383_at 32436_at 49937_at 81837_at
5.572388 8.370524 2.579629 8.901415 5.743816 6.167160
85075_at 34099_f_at 35376_f_at 49888_f_at 87506_at
6.539437 4.623024 3.686496 11.108732 9.486965

```

```
> apply(a,2,mean)
```

```

c1cCEL c2cCEL c3cCEL
6.165170 6.232670 6.204101

```

```
>
```

We can not `apply()` on the whole frame, because it would first be converted all to strings.

We can easily make a data.frame:

```
> a=11:20
```

```
> b=a^2
```

```
> data.frame(a,b)
```

```

  a  b
1 11 121
2 12 144
3 13 169
4 14 196
5 15 225
6 16 256
7 17 289
8 18 324
9 19 361
10 20 400

```

```
>
```