



Introduction to R

1st lecture

Alessandro FERMI – Giovanna VENUTI



The R environment

R is an integrated suite of software facilities for data manipulation, calculation and graphical display.

- It's open source!
- It has a suite of operators for calculations on arrays, in particular matrices
- It's object-oriented
- It has graphical facilities for data analysis
- It consists of various packages

It has a well developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.

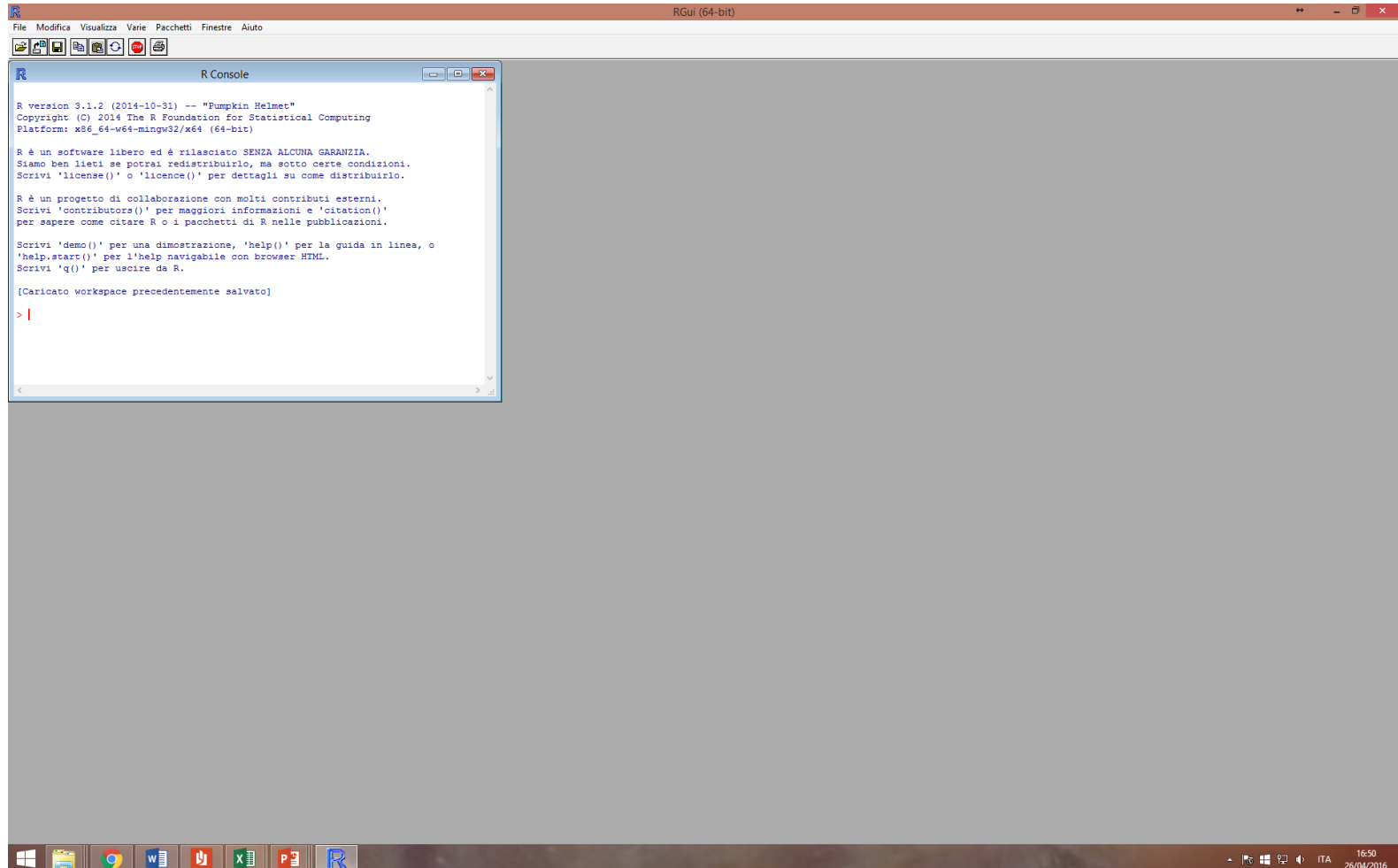


How to download R?

- Google it using R or CRAN
(Comprehensive R Archive Network)
- <http://www.r-project.org>

R Interface

Starting R, the main window (RGui) with a sub window (R Console) will appear. In the `Console' window the cursor is waiting for you to type in some R commands.



The screenshot displays the R GUI (RGui) window. The main window is titled "RGui (64-bit)" and has a menu bar with "File", "Modifica", "Visualizza", "Varie", "Pacchetti", "Finestre", and "Aiuto". A toolbar is visible below the menu bar. A sub-window titled "R Console" is open, showing the R startup sequence. The console text is as follows:

```
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"  
Copyright (C) 2014 The R Foundation for Statistical Computing  
Platform: x86_64-w64-mingw32/x64 (64-bit)  
  
R è un software libero ed è rilasciato SENZA ALCUNA GARANZIA.  
Siamo ben lieti se potrai redistribuirlo, ma sotto certe condizioni.  
Scrivi 'license()' o 'licence()' per dettagli su come distribuirlo.  
  
R è un progetto di collaborazione con molti contributi esterni.  
Scrivi 'contributors()' per maggiori informazioni e 'citation()' per sapere come citare R o i pacchetti di R nelle pubblicazioni.  
  
Scrivi 'demo()' per una dimostrazione, 'help()' per la guida in linea, o 'help.start()' per l'help navigabile con browser HTML.  
Scrivi 'q()' per uscire da R.  
  
[Caricato workspace precedentemente salvato]  
  
> |
```

The Windows taskbar at the bottom shows the system tray with the date and time "25/04/2016 16:50" and the language "ITA".

Literature on R

R version 3.1.2 (2014-10-31) -- "Pumpkin Pie"
 Copyright (C) 2014 The R Foundation for Statistical Computing
 Platform: x86_64-w64-mingw32/x64 (64-bit)

R è un software libero ed è rilasciato sotto la licenza GNU GPL.
 Siamo ben lieti se potrai redistribuirlo e migliorarlo.
 Scrivi 'license()' o 'licence()' per maggiori informazioni.

R è un progetto di collaborazione con molti altri programmi.
 Scrivi 'contributors()' per maggiori informazioni.
 Per sapere come citare R o i pacchetti, scrivi 'citation()'.

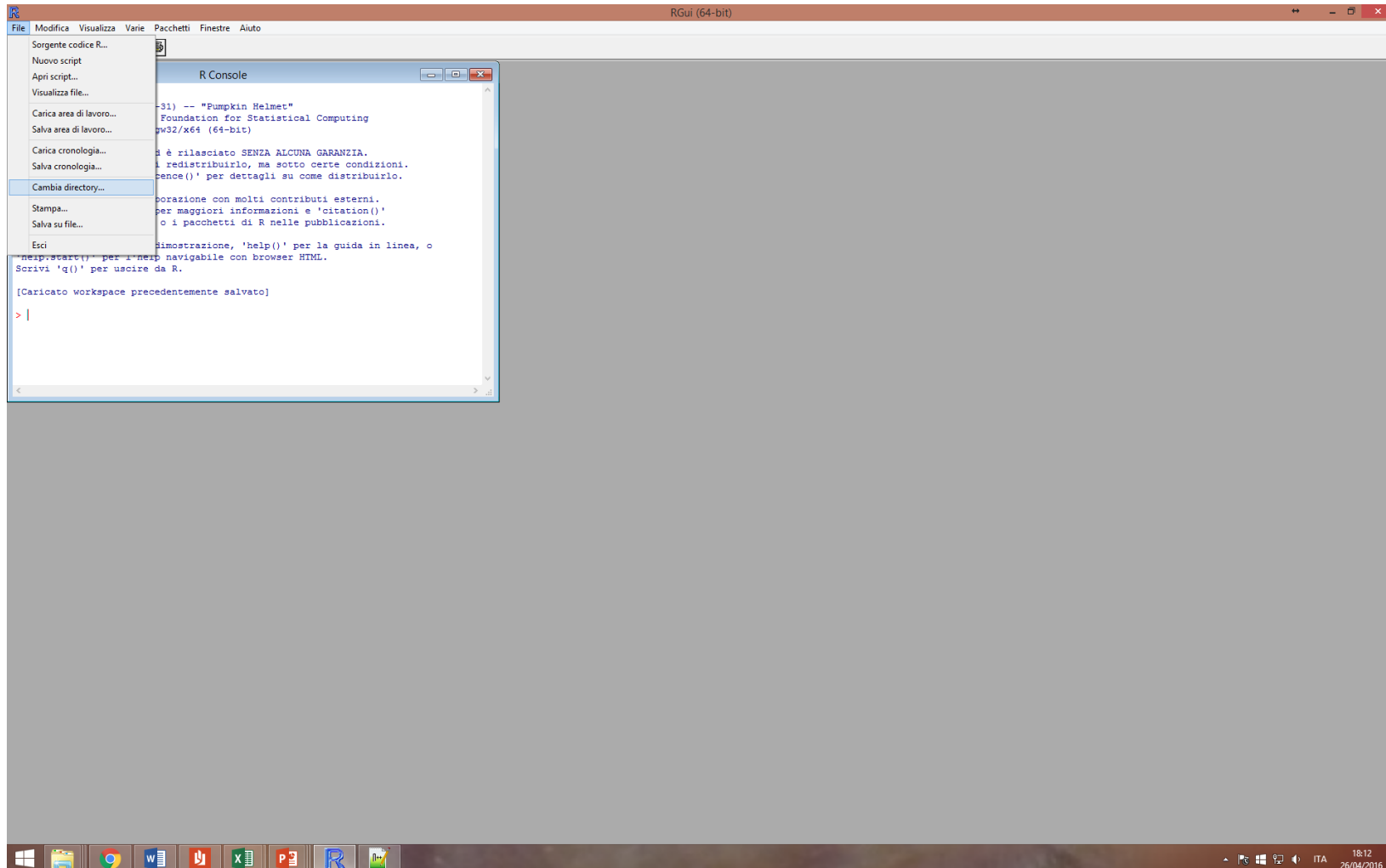
Scrivi 'demo()' per una dimostrazione, 'help.start()' per l'help navigabile con browser HTML.
 Scrivi 'q()' per uscire da R.

[Caricato workspace precedentemente salvato]

```
> library()
> q()
> |
```

Using R interactively

Create a separate folder to hold data files and commands files used in the current session. Then in the interface you can choose this directory as the working directory.





R overview - I

You can enter commands one at a time at the console or run a set of commands from a source file.

There is a variety of data types, including vectors (numerical, logical and character), matrices and higher dimensional arrays, factors, lists and data frames.

To quit R use the command

```
> q()
```

At this point you will be asked whether you want to save the data from your R session.

All object variables and commands are stored in two files “.Rdata” and “.Rhistory” in the working directory.



R overview - II

Most functionalities can be either provided by built-in or user-defined subroutines.

All data objects are kept in memory during a session
The R commands

- `> objects()` or alternatively
- `> ls()`

can be used to display the names of the objects.

The collection of objects currently stored is called the *workspace*.



To remove an object no longer needed, use the R function

- `> rm()`

E.g., if the objects `x`, `foo` and `temp` are in the current workspace,

```
> rm(x, foo, temp)
```

delete them from the workspace.



R overview - IV

R has a built-in help facility for getting more info about specific functions, e.g. solve.

You may use the syntax

- `> ?solve`

or alternatively

- `> help(solve)`

Some further features on R

- it is case-sensitive
- elementary commands consist of either expressions or assignments. If an expression is given as a command, it is evaluated, printed and the value is lost



R overview - V

- commands are separated either by a semi-colon (;), or by a newline. Elementary commands can be grouped together into one compound expression by braces ({ and })
- comments can be put almost everywhere and have to start with a hashmark (#).

All R functions and datasets are stored in *packages*. Only when a package is loaded are its contents available.

To see which packages are installed use the function

- `> library()`

To see which packages are loaded use the function

- `> search()`



R numbers and vectors

The simplest data structures, which R operates on, are numbers and numeric vectors. The latter are ordered collection of numbers.

To define a numeric vector x with specific values, the assignment operator ' $<-$ ' or ' $=$ ' is used; e.g.

```
> x <- c(2.4, 6, 8.9, 1.0, 0.0, 7.3)
```

Alternatively, one can use the `assign()` function

```
> assign("x", c(2.4, 6, 8.9, 1.0, 0.0, 7.3)).
```

The assignment may be done in the other direction as well

```
> c(2.4, 6, 8.9, 1.0, 0.0, 7.3) -> x
```



R numbers and vectors

The further assignment

```
> y <- c(x, 0, x)
```

defines a vector with 13 entries, consisting of two copies of `x` with `0` in the middle.

The length of a vector can be checked through the `length()` function; e.g. the commands

```
> l_y <- length(y); l_x <- length(x)
```

store the lengths of `y` and `x`, respectively, in the variables `l_y` and `l_x`.

If `x` is entered as a command, then its value is printed.



Generate regular sequences

To generate regular sequences, R offers various facilities.

1. If the sequence $c(1,2,3,4\dots,50)$ is needed, one can use the colon operator `:`, i.e.

```
> x1 <- 1:50
```
2. The function `seq()` is a more general facility. It has the following arguments
 1. `'from=value'`, `'to=value'`: define, respectively, the first and end values of the vector.
 2. `'by=value'`, `'length=value'`: define, respectively, the step size and a length for the sequence
 3. `'along=vector'`: normally used as the only argument to create the sequence $1, 2, \dots, \text{length}(\text{vector})$, or the empty sequence if the vector is empty.



Generate regular sequences

For instance

```
> x2 <- seq(from=1, to=50)
```

is the same as x1.

Furthermore

```
> x3 <- seq(by=1, length=50)
```

is the same as x1 and x2.

Not all the names of the arguments are needed. For example

```
> seq(-5, 5, by=.1) -> x4
```

defines the vector $c(-5.0, -4.8, -4.6, \dots, 4.6, 4.8, 5.0)$.

If the names of the arguments are used, then their order is irrelevant.



Vector arithmetic

Vectors can be used in arithmetic expressions, in which case the operations are performed element by element.

Example

```
> y1 <- 1:21
> seq(0, by=.1,2) -> y2
> aux <- length(y1) == length(y2)
> aux
[1] TRUE
> w <- 2*y1+y2^2+1
> w
```

The elementary arithmetic operators are the usual $+$, $-$, $*$, $/$ and $^$. In addition all of the common arithmetic functions are available, e.g. \log , \exp , \sin , \cos , \tan , $\sqrt{}$, \max , \min , range , sum , prod .

The `ls()` will display all vector objects created so far.



Vector arithmetic

Important statistical functions available are

- `mean(x)` : computes the sample mean of the vector `x`
- `var(x)` : computes the sample variance of `x`
- `sd(x)` : it yields the standard deviation of `x`

Of course `mean(x)` is the same as `'sum(x)/length(x)'`, while `var(x)` is equal to

`'sum((x-mean(x))^2)/(length(x)-1)'`

`sort(x)` returns a vector of the same length of `x` with elements in increasing order.

Remark: complex vectors can be defined and complex arithmetic is available.



Vector arithmetic

Example. Assume the following observations of the variables X and Y are given:

(1,1) (1,0) (0,1) (0,2) (0,2) (1,0) (1,1) (0,2) (0,2) (1,0)

compute the table of relative frequencies, the marginal frequencies of X and Y. Moreover, compute the relative frequencies of Y conditional to $X=0$

```
> X <- c(1, 1, 0, 0, 0, 1, 1, 0, 0, 1)
> Y <- c(1, 0, 1, 2, 2, 0, 1, 2, 2, 0)
> v3 <- table(X,Y)
> v3
> v3 <- prop.table(v3) or v3/length(X)
> marg_X <- margin.table(v3,1)
> marg_Y <- margin.table(v3,2)
> Y_X0 <- v3[1,] / marg_X[1]
> Y_X0
  0  1  2
0.0 0.2 0.8
```



Logical vectors

R allows handling and manipulation of logical quantities.

A logical variable can have the values TRUE, FALSE or NA (“Not Available”).

Logical vectors are generated by logical conditions.

The logical operators are `<`, `<=`, `>`, `>=`, `==` for exact equality and `!=` for inequality. In addition if `c1` and `c2` are logical expressions, then “`c1 & c2`” is their intersection (“and”), “`c1 | c2`” is their union (“or”), and “`!c1`” is the negation of `c1`

Example

```
> l1 <- seq(to=3, from=-1)
> l2 <- seq(from=1, to=length(l1), by=.5)
> bool <- length(l1) == length(l2)
> bool
> ind_pos <- (l1 < 0)
```



Character vectors

Character variables and strings are very useful.

They are denoted by a sequence of characters delimited by the double quote character, e.g., "x-values" (or 'x-values')

They use C-style escape sequences, e.g. '\n' for a new line.

For a full list of escape sequences, one can use the help command

```
> ?Quotes
```

The paste() function takes an arbitrary number of arguments and concatenate them to a string.

Example.

```
> paste(c("X", "Y"), seq(1,10))  
> paste(c('X','Y'), 1:10, sep="")
```



Index vectors

Subsets of vector elements may be selected by suitable *index vectors*.
For vectors they can be

- a logical vector: values corresponding to TRUE in the index vector are selected and those corresponding to FALSE are omitted; for instance
 - > (x+1) [is.na(x) & x > 0] -> z
- a vector of positive integers: the meaning is the usual one; e.g.
 - > seq(-5, 5, by=.1) -> x4
 - > i <- 10:20
 - > x4[i]
 - [1] -4.1 -4.0 -3.9 -3.8 -3.7 -3.6 -3.5 -3.4 -3.3 -3.2 -3.1
- a vector of negative integers: here the values corresponding to the index vector are omitted



Index vectors

For example

```
> y <- x4[-(1:51)]
> y
[1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9
[20] 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8
[39] 3.9 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.0
(this is the same as x4 [x4 > 0 ] -> y1 (check it!))
```

- a vector of strings: an example make things clear

```
> incomes <- c(10, 15, 7.5, 9, 12.2)
> names(incomes) <- c('Lazio', 'Lombardia', 'Liguria', 'Sicilia', 'Puglia')
> incomes
Lazio Lombardia Liguria Sicilia Puglia
10.0 15.0 7.5 9.0 12.2
> south_incomes <- incomes[c('Lazio', 'Sicilia', 'Puglia')]
Lazio Sicilia Puglia
10.0 9.0 12.2
```



Arrays and matrices

R can operate on matrices and higher dimensional arrays.

A vector can be handled as an array, if a “dim” attribute is assigned to it.

Example

```
> x5 <- 1:60
> class(x5)
[1] "integer"
> dim(x5) <- c(2,3,10)
> x5
> class(x5)
[1] "array"
> dim(x5) <- c(6,10)
> x5
class(x5)
[1] "matrix"
```



Arrays and matrices

Other functions such as `matrix()` and `array()` are available for simpler and more natural assignments.

- `array()` function: the general syntax is
`Z <- array(data_vector, dim_vector)`
For example

```
> x5 <- seq(by=.5, from=.5, to=12)
> Z <- array(x5, dim=c(4,6))
> Z
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	0.5	2.5	4.5	6.5	8.5	10.5
[2,]	1.0	3.0	5.0	7.0	9.0	11.0
[3,]	1.5	3.5	5.5	7.5	9.5	11.5
[4,]	2.0	4.0	6.0	8.0	10.0	12.0



Arrays and matrices

Remark: if the data vector is shorter than the product of the components of the dimension vector, then its values are recycled to match the given dimension vector

For example

```
> x5 <- seq(by=.5, from=.5, to=12)
> length(x5)
[1] 24
> Z <- array(x5, dim=c(4,7))
> Z
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,] 0.5  2.5  4.5  6.5  8.5 10.5  0.5
[2,] 1.0  3.0  5.0  7.0  9.0 11.0  1.0
[3,] 1.5  3.5  5.5  7.5  9.5 11.5  1.5
[4,] 2.0  4.0  6.0  8.0 10.0 12.0  2.0
```



Arrays and matrices

Instead of using the function `array()`, a matrix may be defined applying the function `matrix()`; its general syntax is

```
M <- matrix(data_vector, n_rows, n_cols)
```

Example

```
> x5 <- seq(by=.5, from=.5, to=12)
> M <- matrix(x5, 4, 6)
> M
> M <- matrix(x5, 4, 7)
```

Warning message:

In `matrix(x5, 4, 7)` :

data length [24] is not a sub-multiple or multiple of the
number of columns [7]

```
> M == Z
```

It is always TRUE!!



Arrays and matrices arithmetic

Arrays and matrices may be used in arithmetic expressions and the result is an array/matrix formed by elementwise operations.

The `dim` attributes of operands generally need to be the same, and this becomes the dimension vector of the result.

Furthermore, again all basic mathematical operations performed element-by-element are available



Arrays and matrices operations

An important operation on matrices and arrays is the outer product.

If M and Z are two numeric arrays, their outer product is an array whose dimension vector is obtained by concatenating their two dimension vectors (order is important), and whose data vector is got by forming all possible products of elements of the data vector of a with those of b .

Its syntax is

```
> MZ <- M %o% Z
```

or

```
> MZ <- outer(M, Z, '*')
```

Remark. The multiplication function can be replaced by an arbitrary function of two variables.



Arrays and matrices operations

For example if we wished to evaluate the function $f(x;y) = \cos(y)/(1 + x^2)$ over a regular grid, then

```
> f <- function(x,y) cos(y)/(1+x^2)
> x <- seq(1,2,by=.1)
> y <- seq(1,3,by=.2)
> val_mesh <- outer(x, y, f)
> image(x,y,val_mesh)
> contour(x,y,val_mesh)
```

(Change function and mesh!)

Example. Consider the determinants of 2 by 2 matrices with entries in the range 0,1,...,9. The problem is to find the determinants, $ad-bc$, of all possible matrices of this form and represent the frequency with which each value occurs as a high density plot.

```
> d <- outer(1:9, 1:9)
> det <- outer(d, d, ' - ')
> dim(det)
```



Matrix operations

- > `fr <- table(det)`
- > `fr <- fr / length(det)`
- > `plot(fr, xlab='Determinants', ylab='rel. freq.')`
- > `Fr <- cumsum(fr)`
- > `dev.new()`
- > `Fr <- plot(Fr)`

All usual matrix operations are available in R. In particular

- Transpose: `t(M)`
- Determinant: `det(M)`
- Number of rows and columns: `nrows(M)` and `ncols(M)`
- Elementwise product: `A * B`
- Matrix multiplication: `A %*% B`



Matrix operations

- A linear system of the form $Ax = b$, where A is a matrix and b is a known vector, may be solved with the `solve()` function

```
> x <- solve(A, b)
```

To compute the inverse use the command

```
> solve(A)
```

- Eigenvalues and eigenvectors: the function `eigen()` computes the eigenvalues and eigenvectors of a square (real and complex) matrix. It returns a list, consisting of two components, namely values and vectors.



Matrix operations

Example

```
> M <- diag(1:5)
> eg <- eigen(M, symmetric=TRUE)
$values
[1] 5 4 3 2 1
```

\$vectors

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	0	0	0	0	1
[2,]	0	0	0	1	0
[3,]	0	0	1	0	0
[4,]	0	1	0	0	0
[5,]	1	0	0	0	0

```
> evals <- eg$values
> eg <- eigen(M, symmetric=TRUE, only.values=TRUE)
> eg
```




**THANK YOU FOR YOUR
ATTENTION!**