GS01 0163
Analysis of Microarray Data

Keith Baggerly and Kevin Coombes
Section of Bioinformatics
Department of Biostatistics and Applied Mathematics
UT M. D. Anderson Cancer Center
kabagg@mdanderson.org
kcoombes@mdanderson.org

31 August 2004

Lecture 1: Introduction to Microarrays

- Administrative Matters
- What do microarrays measure?
- Introduction to R

Administrative Matters

- Course web site
  http://bioinformatics.mdanderson.org/MicroarrayCourse
- Office hours: T, Th, 3:30-4:30, Faculty Center
  - Dr. Baggerly: FC2.2060
  - Dr. Coombes: FC2.3014
- Grading:
  - 70% Homeworks (every two weeks)
  - 30% Final project

  1. Homeworks and final projects can be worked on jointly.
  2. Assignments submitted electronically, in a single file.
  3. Submissions should include figures, R code, and text.

Textbooks

Required: Dalgaard P. *Introductory Statistics with R.* Springer-Verlag, New York, 2002.

Recommended: Simon RM, Korn EL, McShane LM, Radmacher MD, Wright GW, Zhao Y. *Design and Analysis of DNA Microarray Investigations.* Springer-Verlag, New York, 2003.

Optional: Speed T (ed). *Statistical Analysis of Gene Expression Microarray Data.* Chapman and Hall, New York, 2003.

Optional: Parmigiani G, Garrett ES, Irizarry RA, Zeger SL (eds). *The Analysis of Gene Expression Data.* Springer-Verlag, New York, 2003.

Course Outline

Week 1: Introduction to microarray technologies

Weeks 2 and 3: Image analysis and quantification for cDNA arrays and Affymetrix arrays

Weeks 4 and 5: Normalization methods. Implications for downstream analysis of low-level processing choices.

Week 6: Methods for selecting differentially expressed genes

Week 7: Theory and practice of multiple comparisons

Weeks 8 and 9: Interpreting gene lists with reference to public databases: IMAGE clone ids, Affymetrix probe sets, GenBank, UniGene, LocusLink

Course Outline

Week 10: Experimental design, sample size, and power for microarray experiments

Week 11: Clustering microarray data

Week 12: Classification of samples using microarray data

Week 13: Validation and cross-validation of results

Week 14: Incorporating clinical information: more advanced uses of microarrays

Week 15: Open problems: meta-analysis, time course experiments, etc.

Course objectives

Microarrays are important for the study of gene expression. This technology changes the way biologists approach problems and introduces new challenges for statisticians. The literature now contains more than 7000 papers using microarrays; biologists should understand how the data is processed in order to evaluate these publications. Statisticians need to understand where the data comes from, in order analyze it appropriately. After taking this course, students should be able to:

- Understand how microarrays work and how they are analyzed.
- Evaluate the analysis of microarray data in a published paper.
- Perform some basic analyses of microarrays.

What do microarrays measure?

Short answer: Gene expression.

Longer answer (also known as “biology in ten minutes”) follows...

What do microarrays measure?

Short answer: Gene expression.

Longer answer (also known as “biology in ten minutes”) follows...

All living creatures are made of cells.

What do microarrays measure?

Short answer: Gene expression.

Longer answer (also known as “biology in ten minutes”) follows...

All living creatures are made of cells. (Except for viruses...)

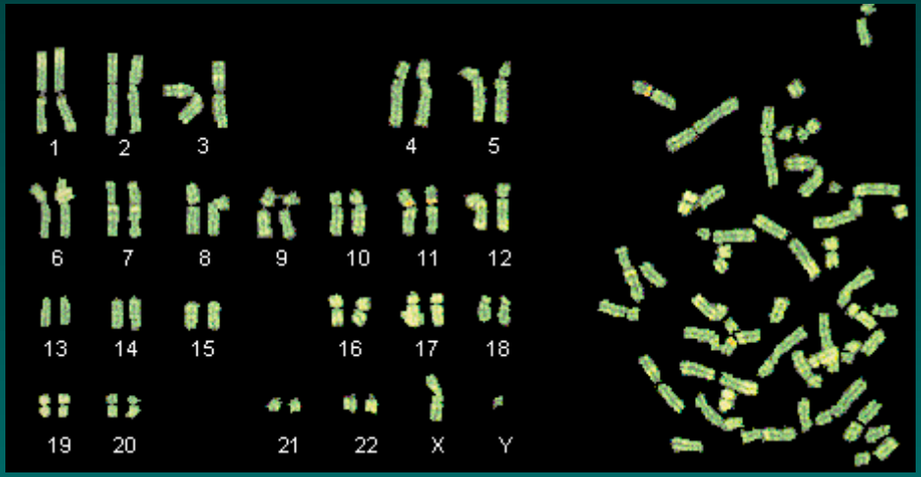
What do microarrays measure?

Short answer: Gene expression.

Longer answer (also known as “biology in ten minutes”) follows...

All living creatures are made of cells. (Except for viruses...)

Chromosomes are the carriers of inherited information. Normal human cells contain 23 pairs of chromosomes. (Figure courtesy of Robert J. Huskey at the University of Virginia.)



The role of gene expression

Each chromosome consists of a pair of DNA molecules held together by complementary nucleotide base pairs (in total, about 3×10^9 base pairs). The structure of DNA provides an explanation for heredity, by copying individual strands and maintaining complementarity.

A normal individual is composed of bazillions of cells all containing identical chromosomes. Since the cells contain the same genetic information, what makes skin cells different from liver cells or kidney cells or brain cells?

The role of gene expression

Each chromosome consists of a pair of DNA molecules held together by complementary nucleotide base pairs (in total, about 3×10^9 base pairs). The structure of DNA provides an explanation for heredity, by copying individual strands and maintaining complementarity.

A normal individual is composed of bazillions of cells all containing identical chromosomes. Since the cells contain the same genetic information, what makes skin cells different from liver cells or kidney cells or brain cells?

Short answer: Gene expression ....

What is a gene?

Traditional definition: the fundamental unit of heredity.

> "Old geneticists knew what they were talking about when they used the term 'gene', but it seems to have become corrupted by modern genomics to mean any piece of expressed sequence...."
>
> – Sydney Brenner, *Science*. 2000; 287: 2173

> "[Gene] is a highly nuanced noun like 'truth'. Ten years ago, it commonly meant 'genetic locus'.... Over time biologists became more comfortable thinking of a gene as a transcribed region of the genome that results in a functional molecular product."
>
> – Nat Goodman, *Genome Technology*. 2001; April: 55-58.

The Central Dogma

Information flows from DNA to RNA to proteins.

The information flow typically occurs in three steps:

Transcription: Portions of DNA sequences are copied into RNA molecules that guide protein synthesis.

Splicing: Eucaryotic RNA molecules are spliced to remove intron sequences.

Translation: Sequences of nucleotides in mRNA are read in sets of three and translated into amino acids to produce a protein.

Definitions

A gene is a contiguous segment of a DNA molecule that gets transcribed into RNA in some cells.

A protein-coding gene is a gene whose mRNA is translated into at least one protein in some cells.

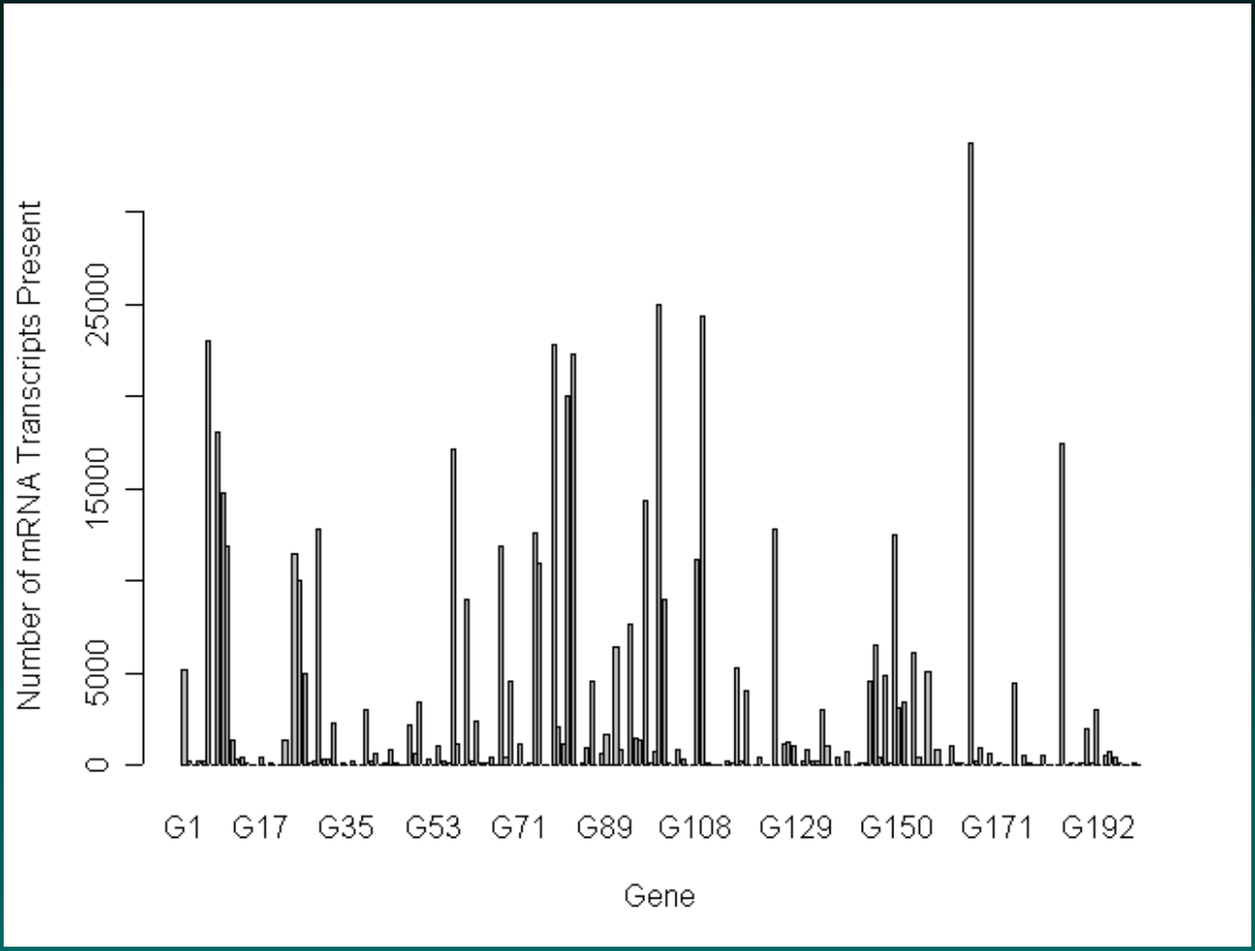
We say that a gene is expressed in a cell if its gene product, in the form of mRNA or protein, is present.

Theoretically, the goal of a microarray experiment is simultaneously to quantify the amount of expression of thousands of genes in a collection of cells; i.e., to measure gene expression.

Microarrays measure mRNA expression, not protein.

An idealized expression profile

If we could count the number of mRNA molecules from each gene in a single cell at a particular time, we might get this:



How do microarrays work?

The biological principle involved is the same one that allows DNA double helices to provide the basis for heredity: Sequences of DNA or RNA molecules containing complementary base pairs have a natural tendency to bind together.

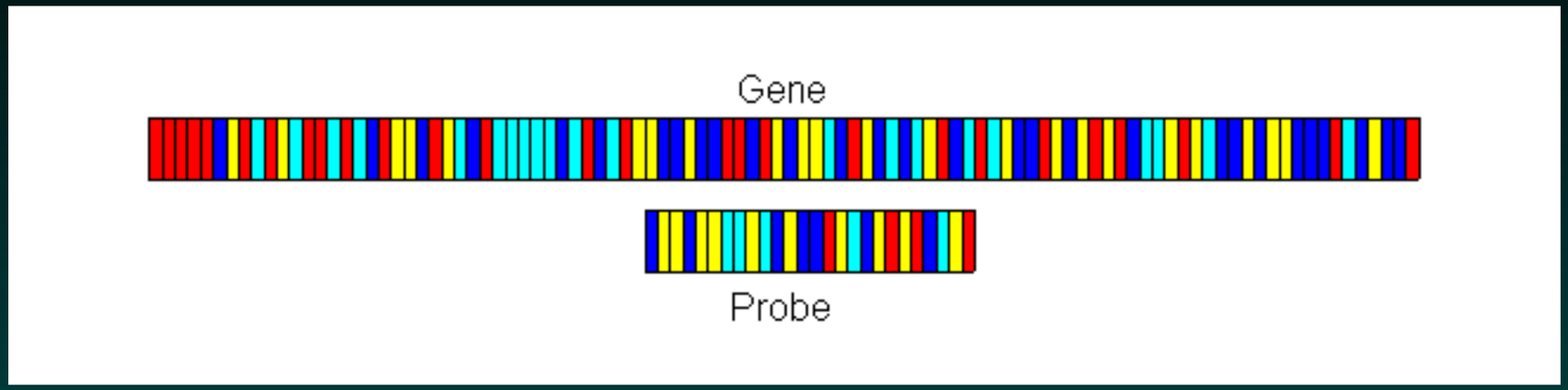
```
...AAAAAGCTAGTCGATGCTAG...
...TTTTTCGATCAGCTACGATC...
```

If we know the mRNA sequence, we can build a probe for it using the complementary sequence. Two possibilities:

- Direct synthesis of a short sequence (oligo)
- Reverse transcription from mRNA to cDNA

How do microarrays work?

In general, the probes are shorter than the genes.



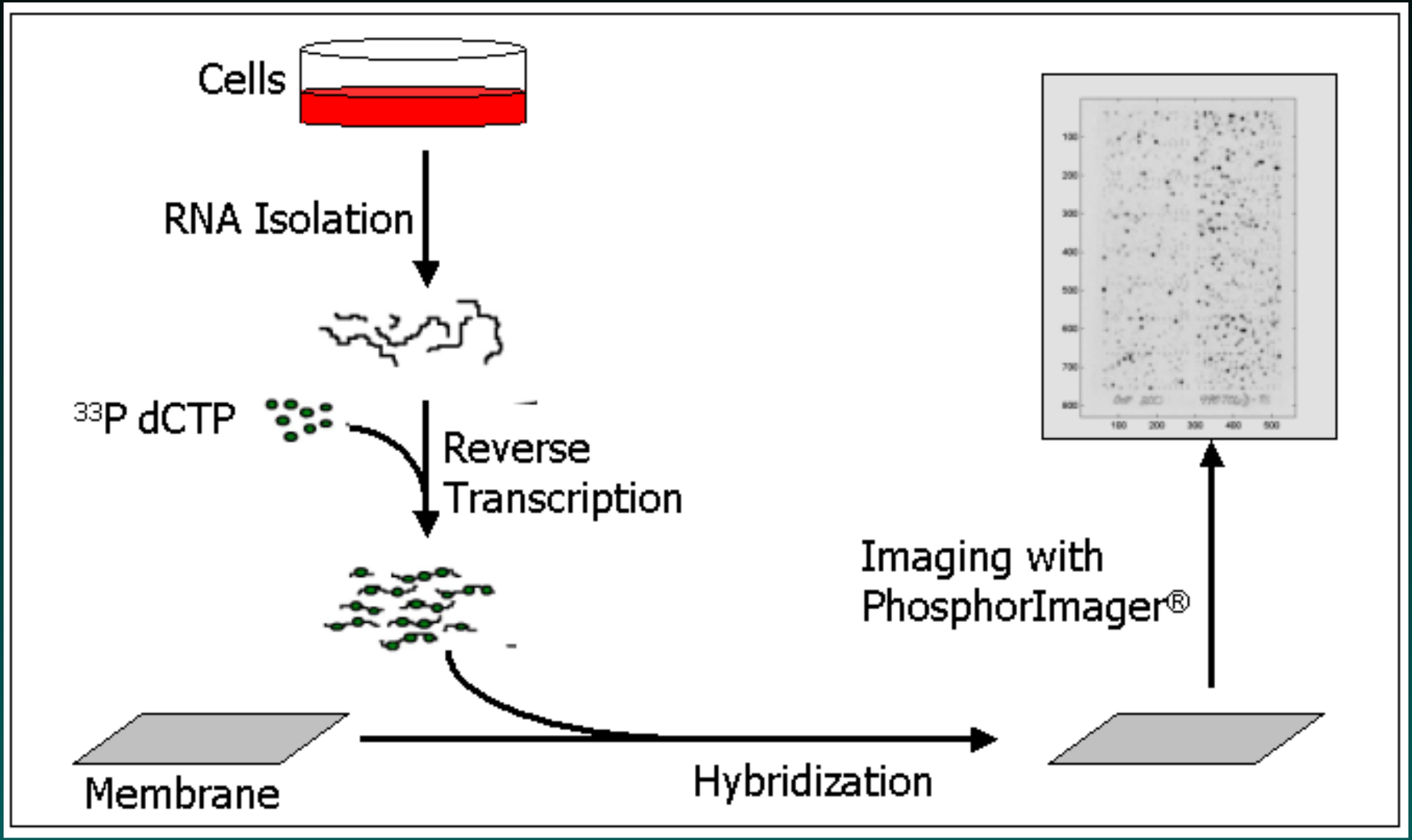
Critical note: Different probes for the same gene have different binding affinities. Since the affinities are unknown, microarrays produce relative measurements of gene expression.

After selecting the desired probes for all genes of interest, they are attached to a solid substrate. Samples containing the target genes are labeled with a fluorescent dye or radiaoactive particle, hybridized to the array, and scanned. *The image is the data.*

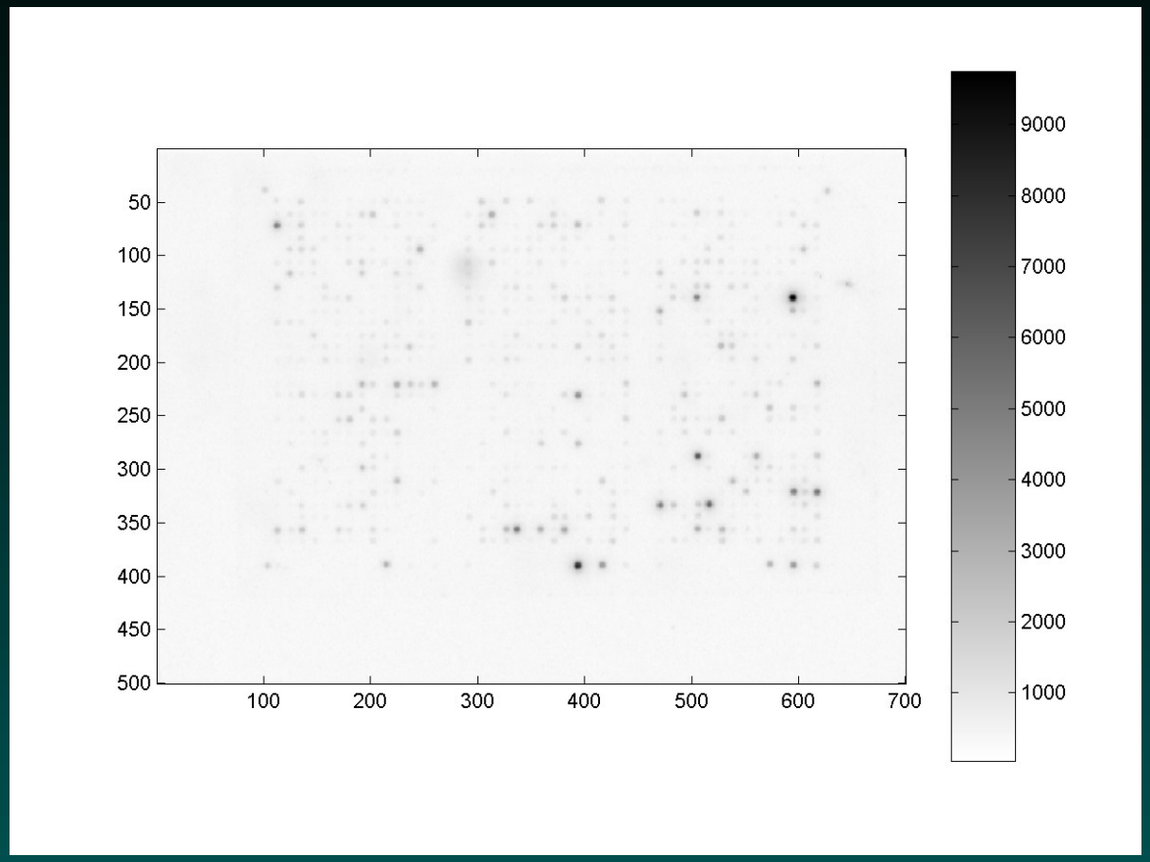
Microarray Platforms

- Spotted cDNA on nylon membranes (obsolete)
  - Commerically produced: Research Genetics, Clontech
  - Radioactive labeling, single channel
- Multiple synthesized short oligos (25-mers) on silicon
  - Commerically produced: Affymetrix
  - Single channel fluorescent labeling
  - Between 11 and 20 probes per gene target
- Spotted cDNA or long oligos (60- or 70-mers) on glass slides
  - Home-grown or commercial
  - Two-channel: simultaneous co-hybridization of two samples
  - Two-color fluorescent labeling

Overview: Nylon cDNA Microarrays

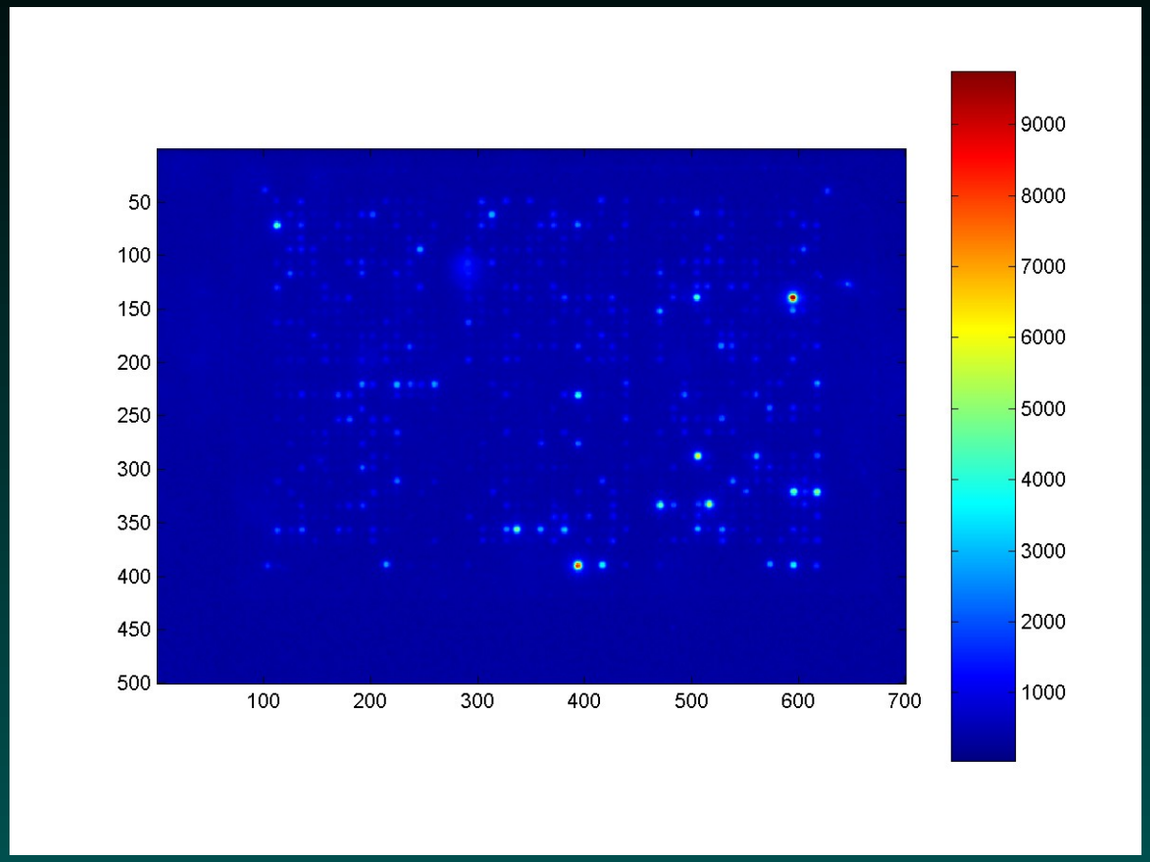


Nylon cDNA Microarrays



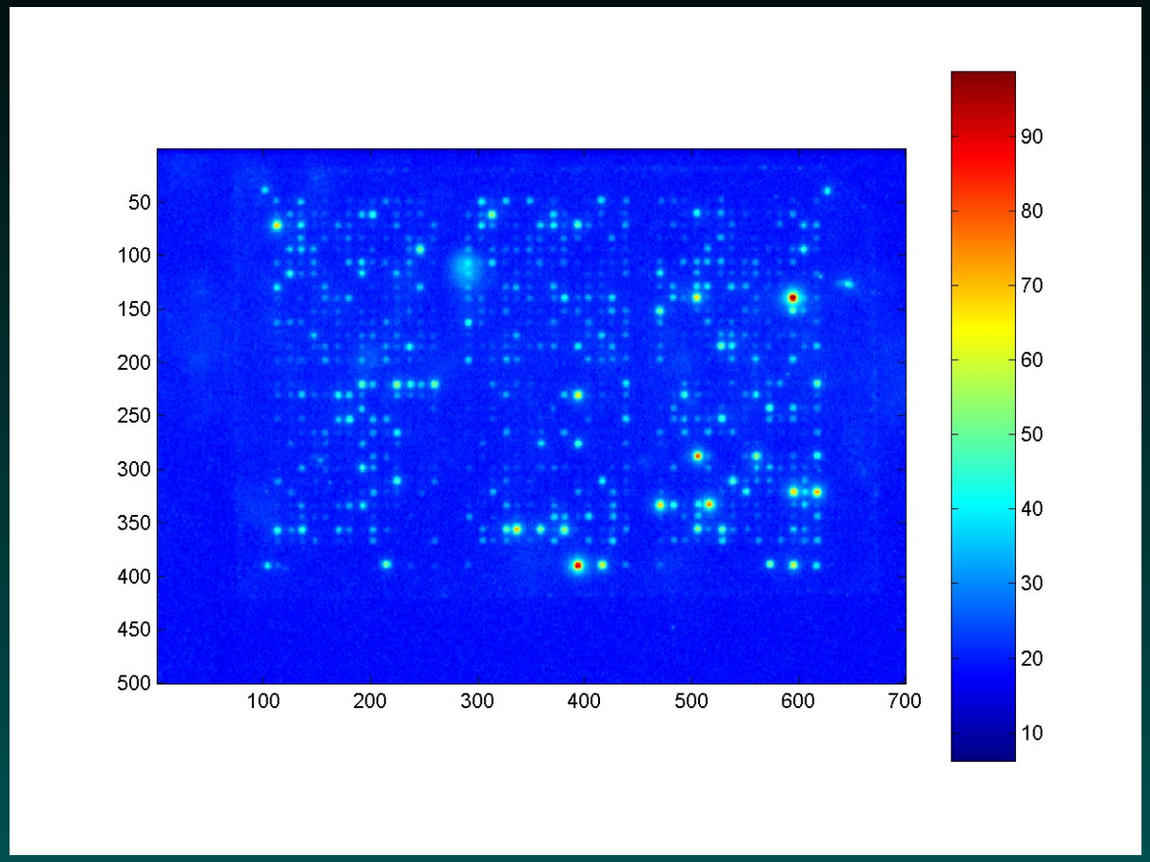
The raw data is a 16-bit, gray-scale, TIFF image.

Nylon cDNA Microarrays



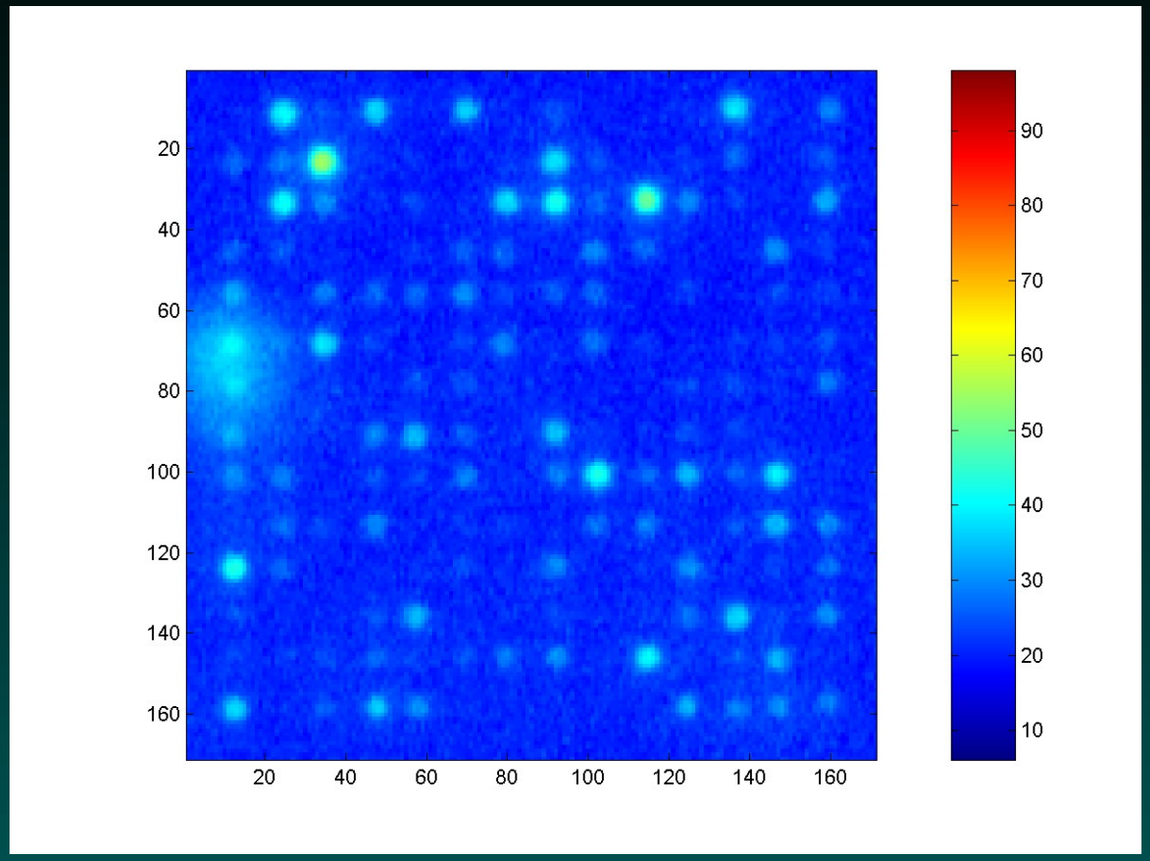
Array images are often viewed in color by changing the color map; this does not change the actual data.

Nylon cDNA Microarrays



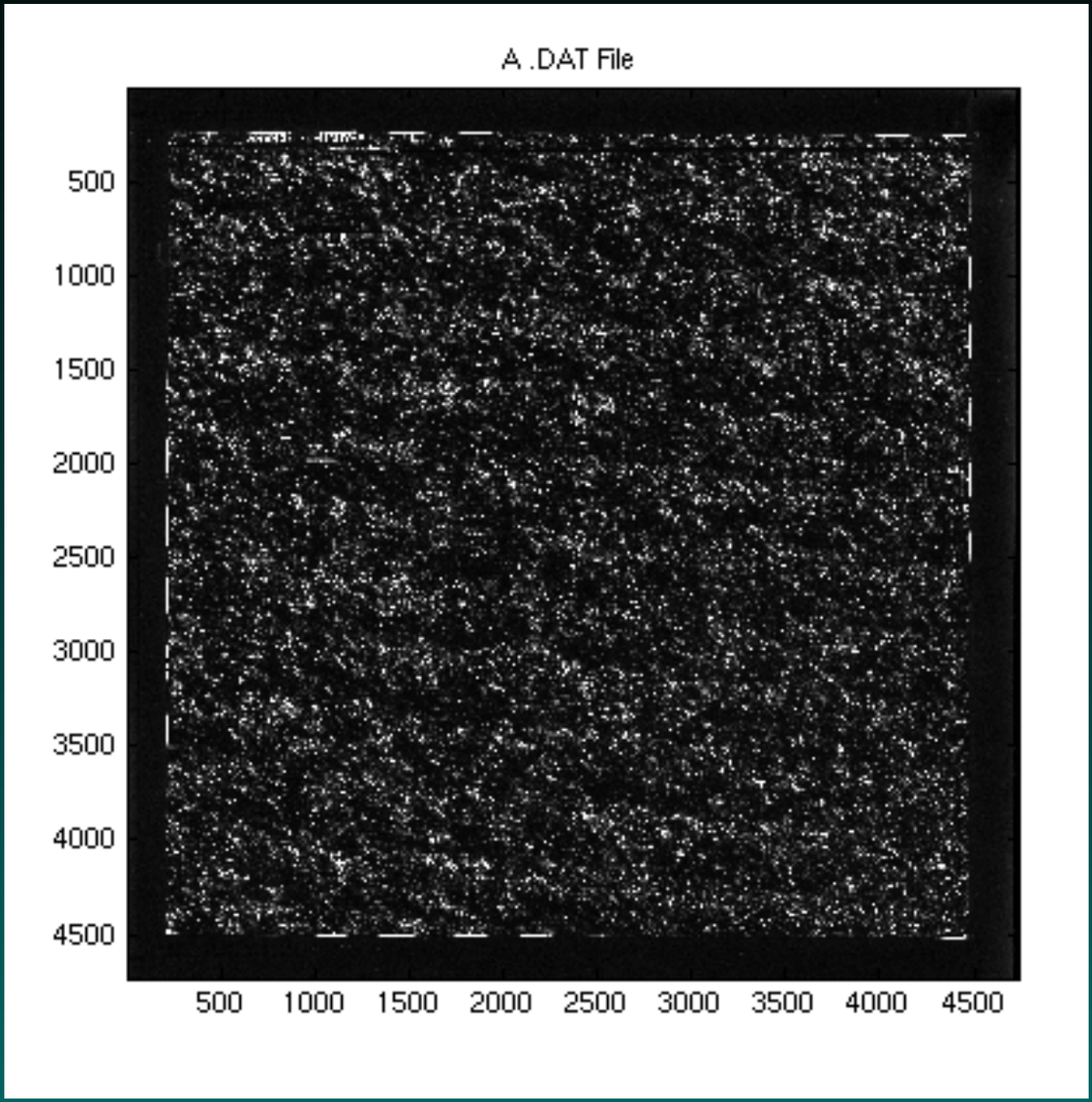
Numerical operations (like this square-root transform) can make certain features more visible. Transformations must only be used for visualization, since they would distort the quantifications.

Nylon cDNA Microarrays

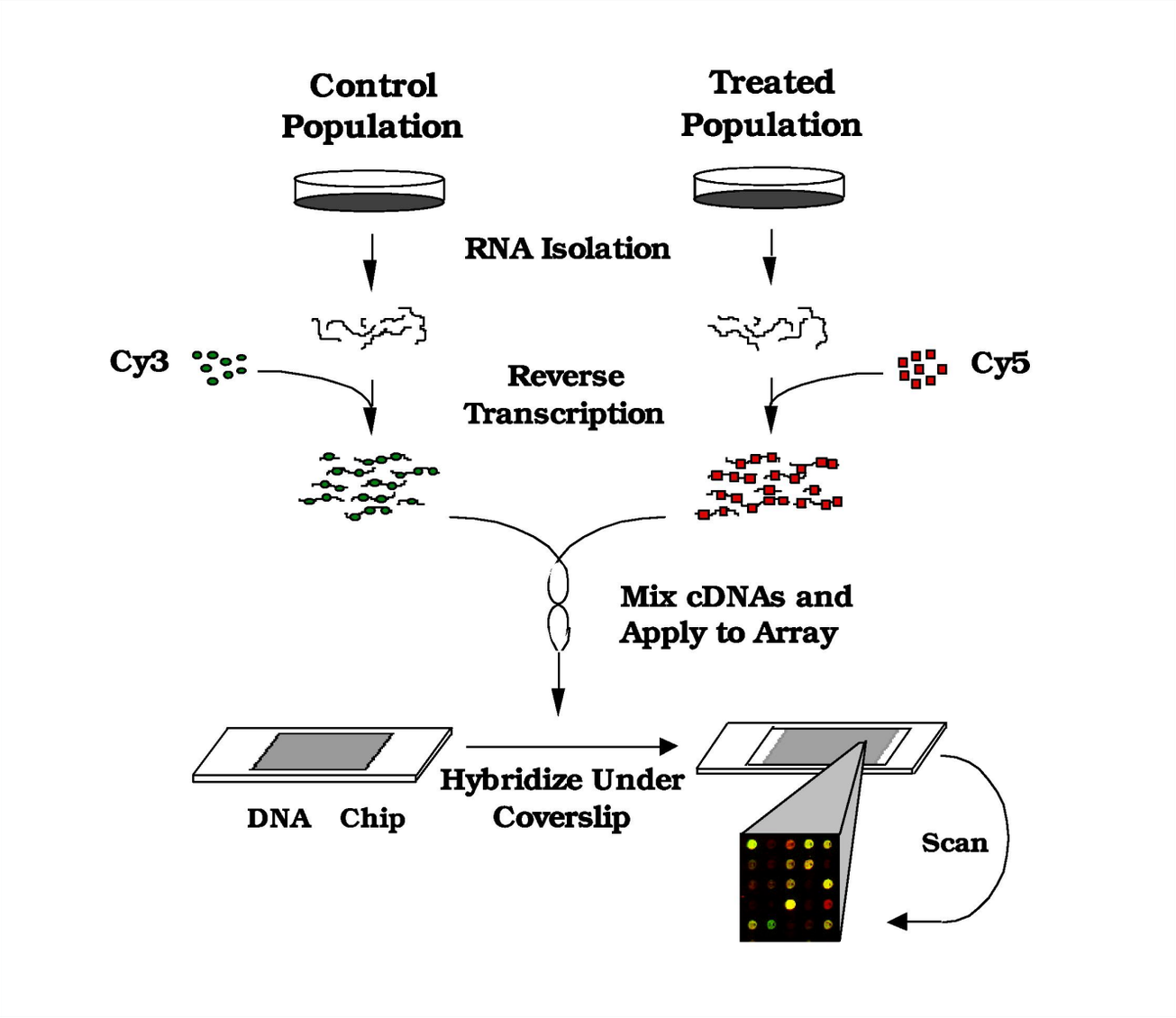


This is a close-up of the same microarray. Note the general blurriness, along with the blotchy artifact along the left edge.

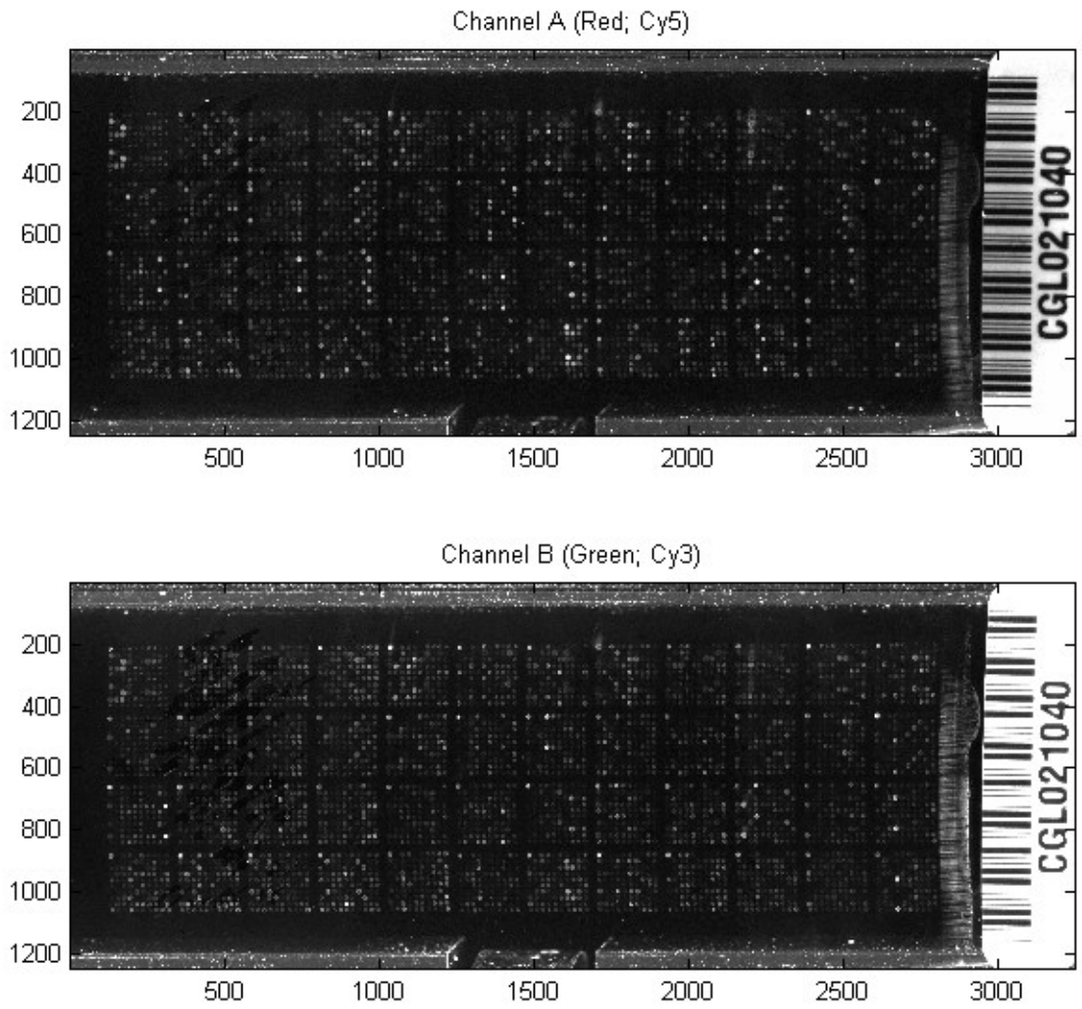
Affymetrix Microarrays



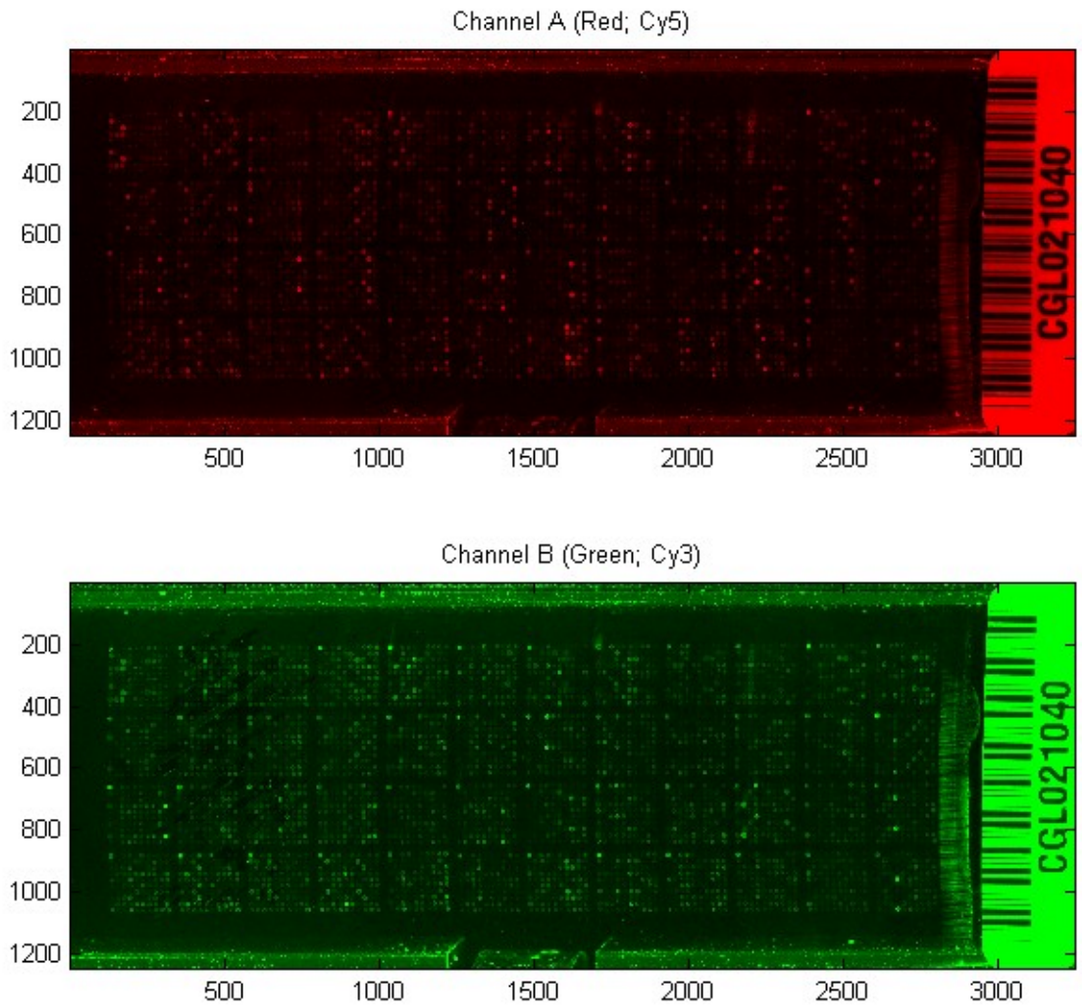
Overview: Two-color Spotted Microarrays



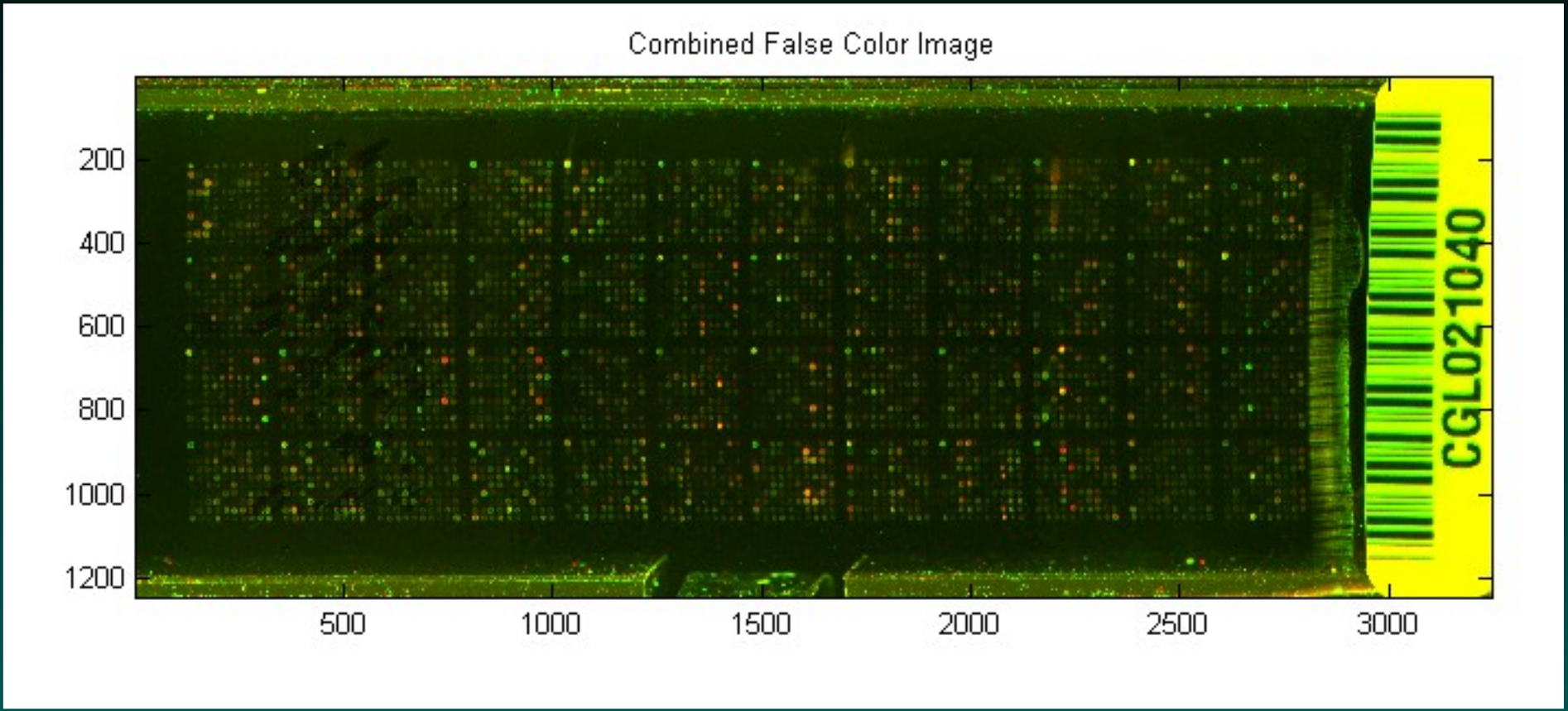
Two-color Spotted Microarrays



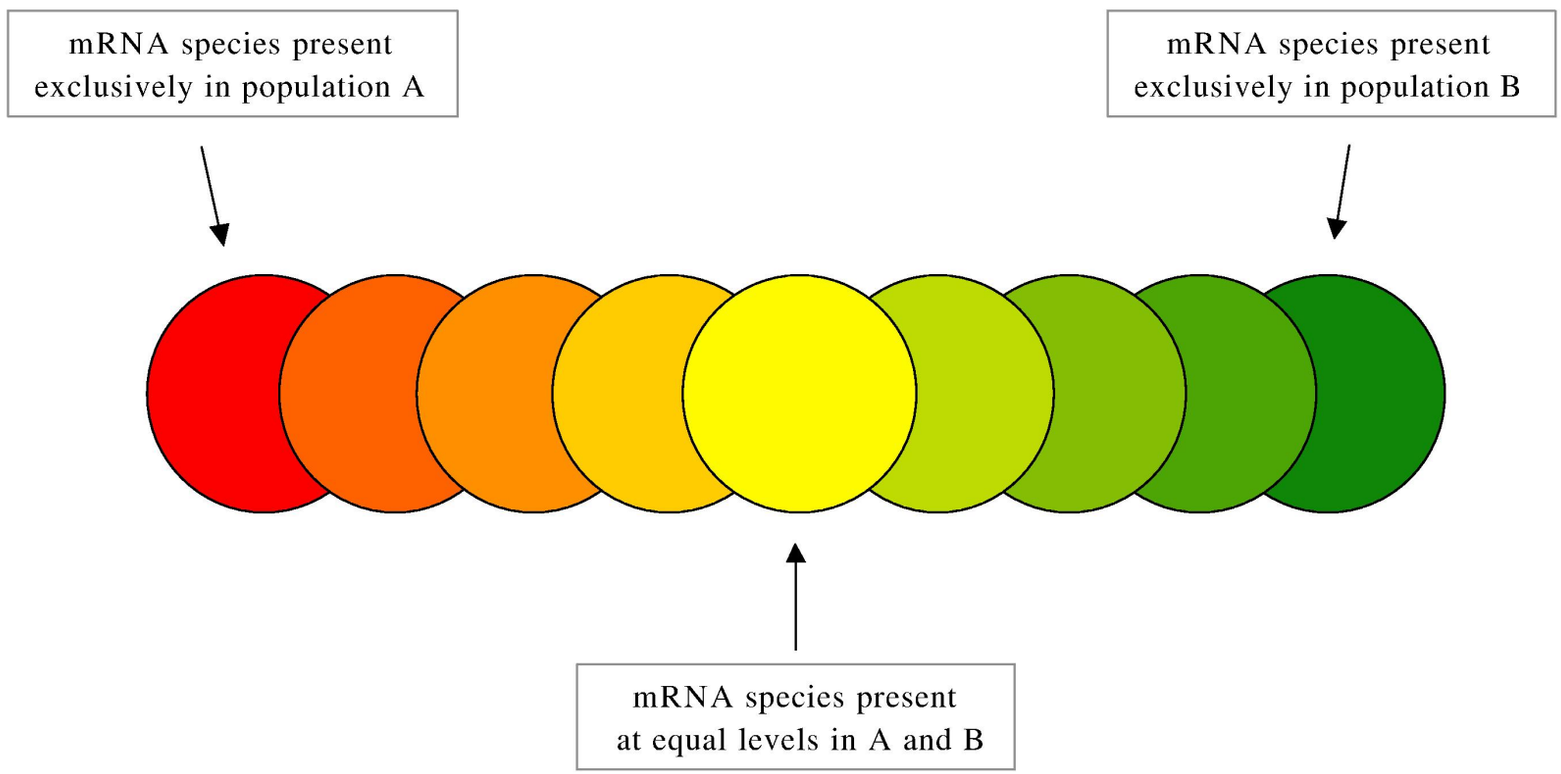
Two-color Spotted Microarrays



Two-color Spotted Microarrays



Two-color Spotted Microarrays



The Images Are The Data

A common feature of all microarray platforms is that the primary data produced by an experiment is in the form of a gray-scale image. The rest of this course will discuss

- First, how to get from those images to useful (semi-)quantifications of gene expression, and
- Second, how to interpret those quantifications to learn something about the underlying biology.

Why R?

- R is a powerful, general purpose language and software environment for statistical computing and graphics.

Why R?

- R is a powerful, general purpose language and software environment for statistical computing and graphics.
- R runs on any modern computer system (including Windows, Macintosh, and UNIX).

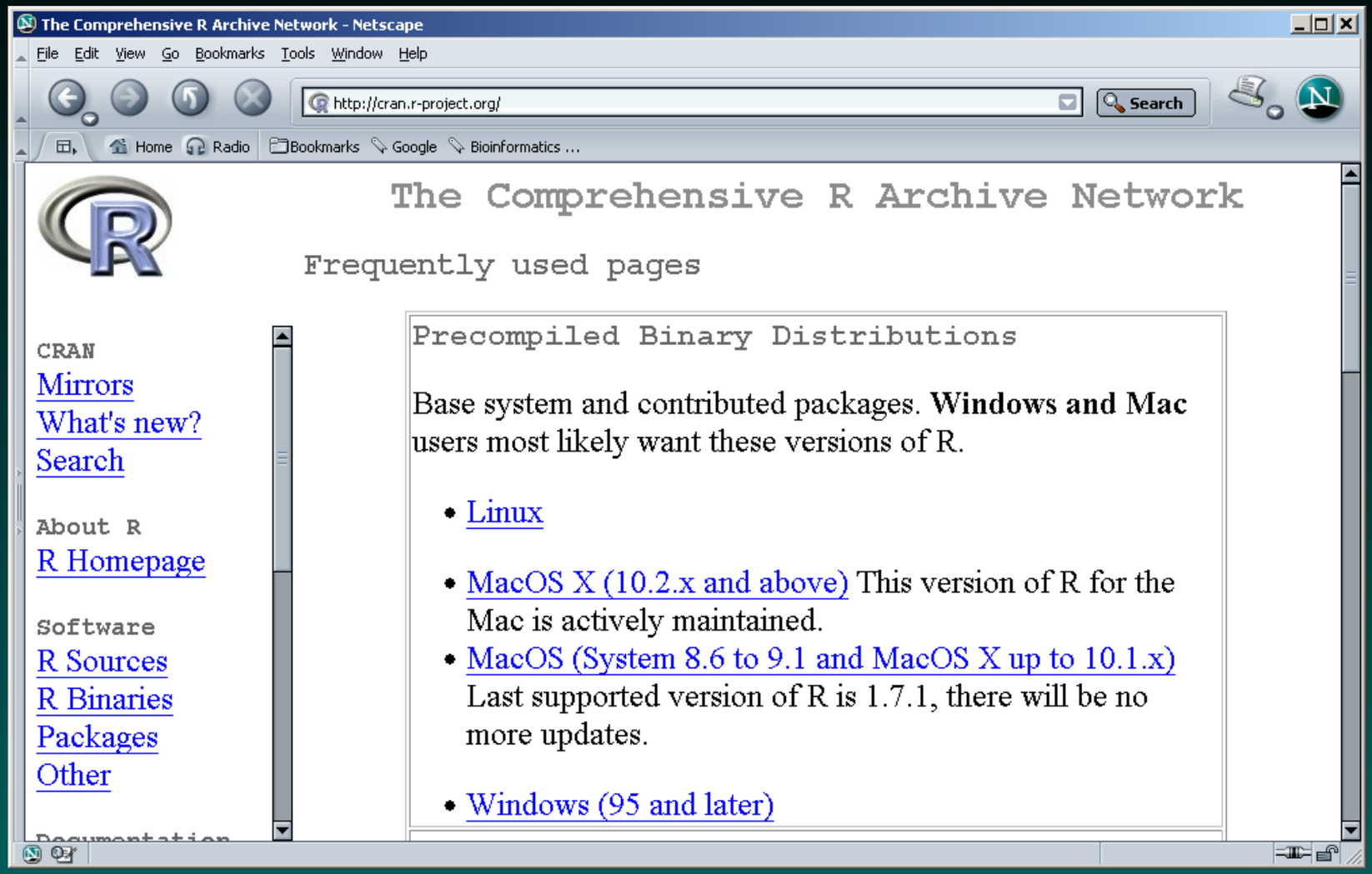
Why R?

- R is a powerful, general purpose language and software environment for statistical computing and graphics.
- R runs on any modern computer system (including Windows, Macintosh, and UNIX).
- There already exists an extensive package of microarray analysis tools, called BioConductor, written in R.

Why R?

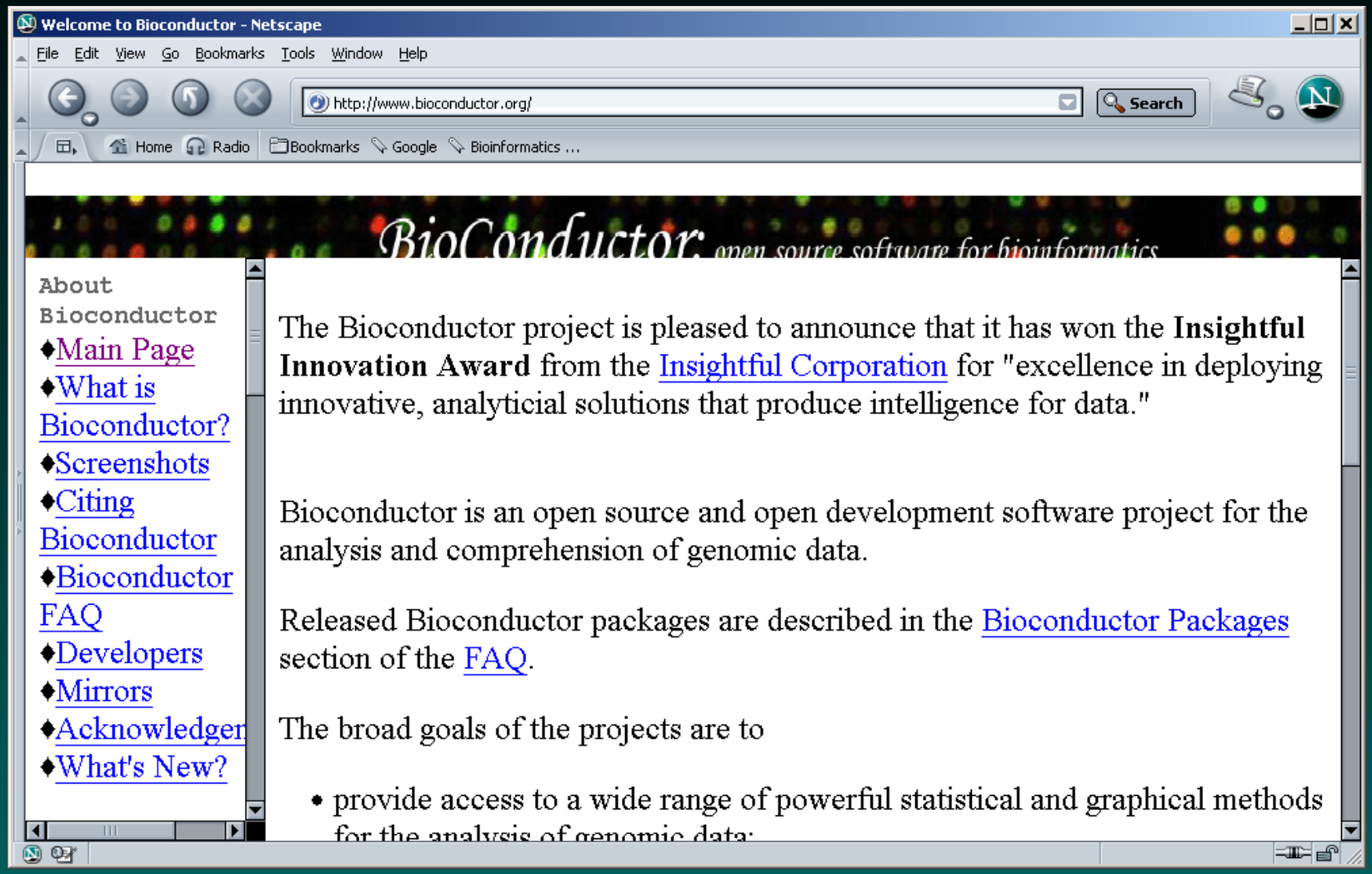
- R is a powerful, general purpose language and software environment for statistical computing and graphics.
- R runs on any modern computer system (including Windows, Macintosh, and UNIX).
- There already exists an extensive package of microarray analysis tools, called BioConductor, written in R.
- R and BioConductor are open source and free.

The Comprehensive R Archive Network



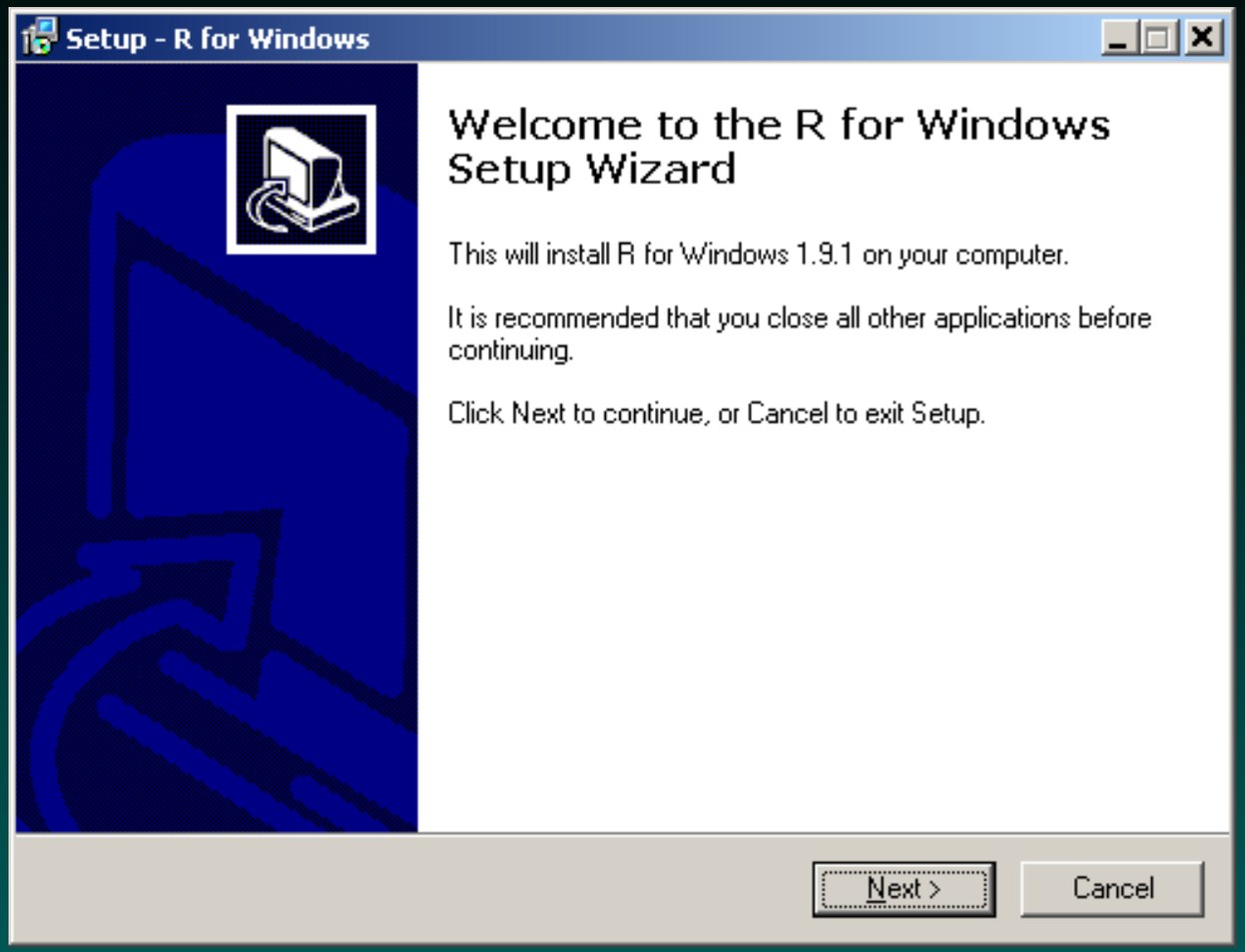
http://cran.r-project.org

The BioConductor Project



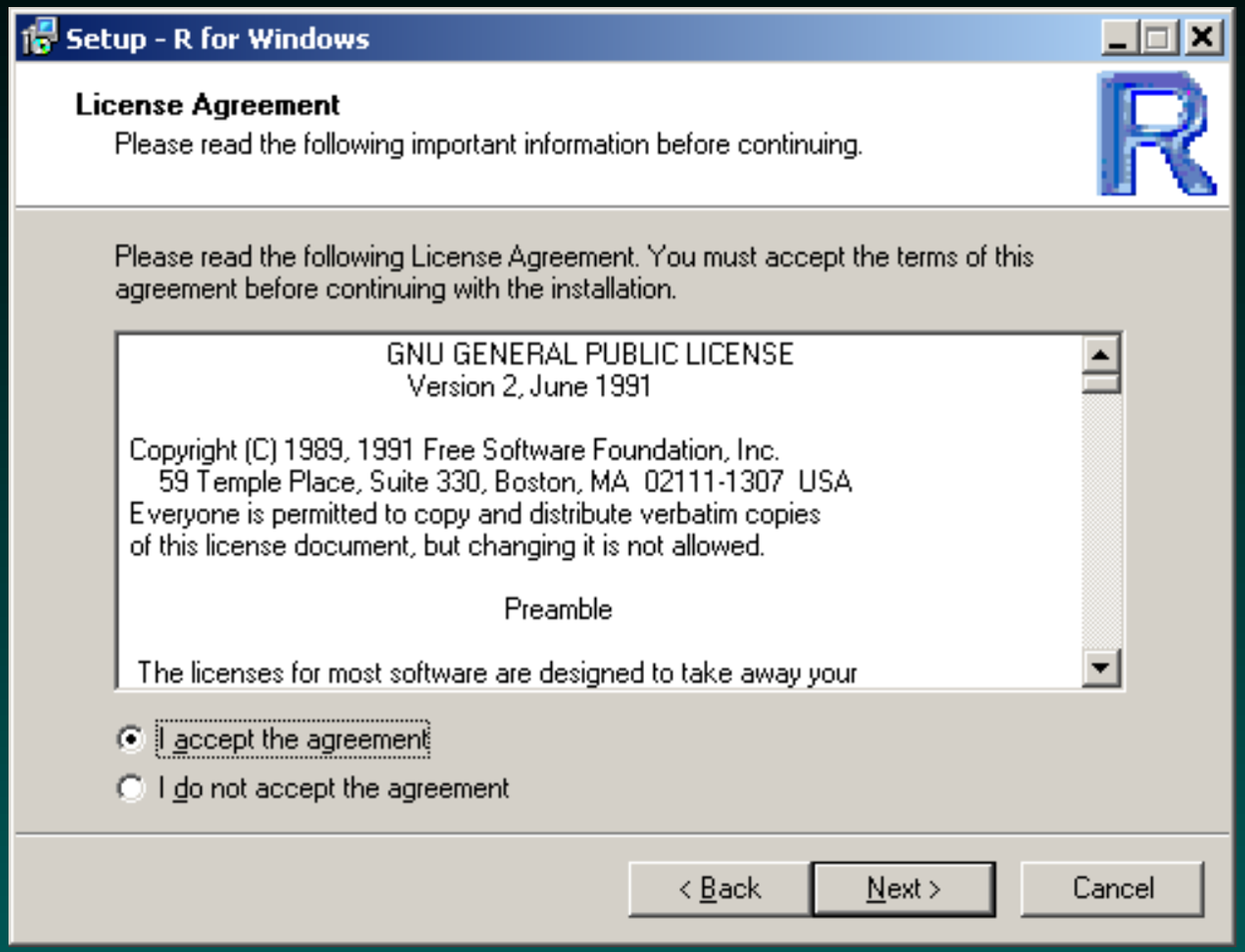
http://www.bioconductor.org

R Installation



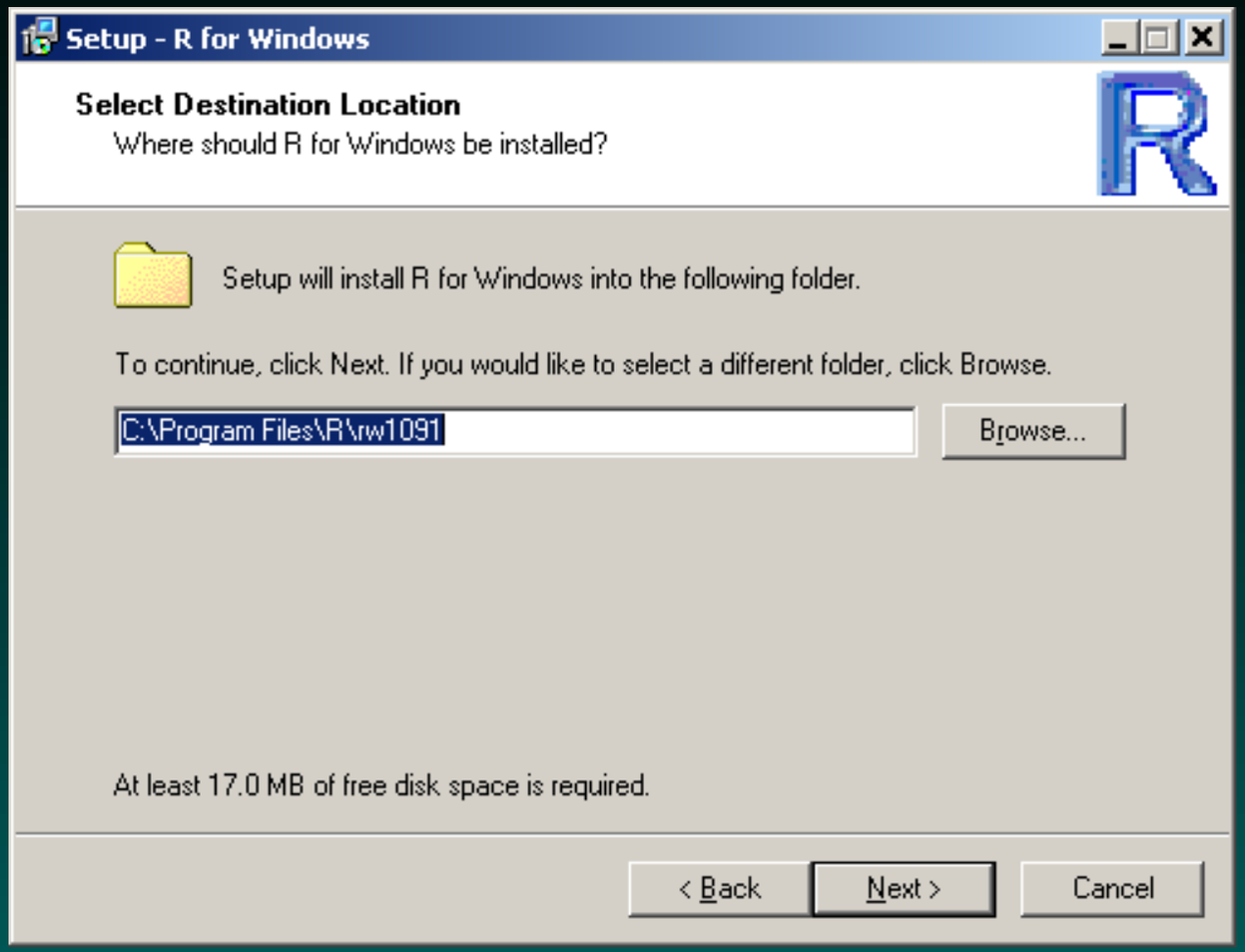
After downloading R from CRAN, you start the installation program and see this screen. Press “Next”.

R Installation



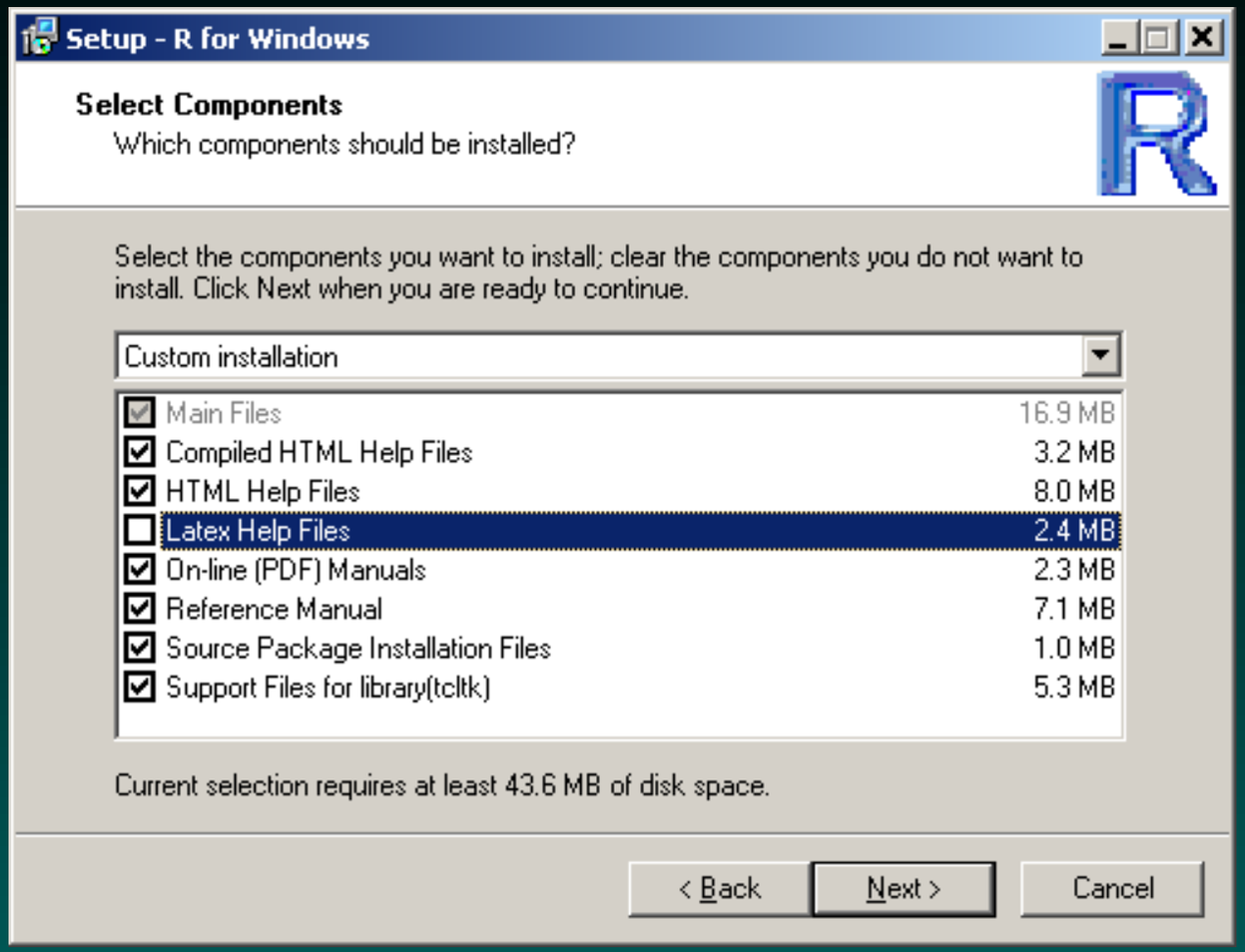
You must click to accept the license agreement before you can proceed.

R Installation



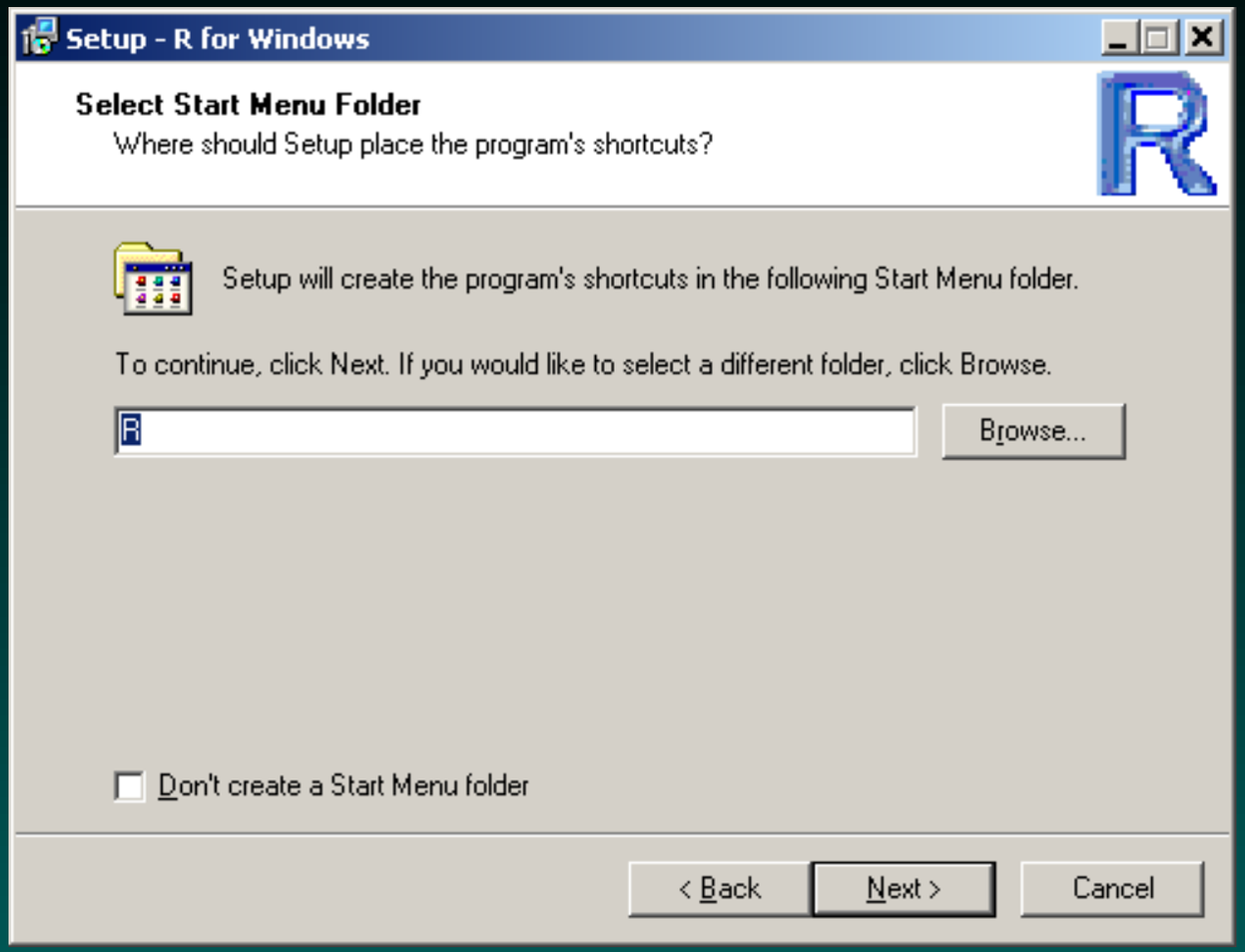
You can change the installation path. It may be a good idea to choose a path name that does not include any spaces.

R Installation



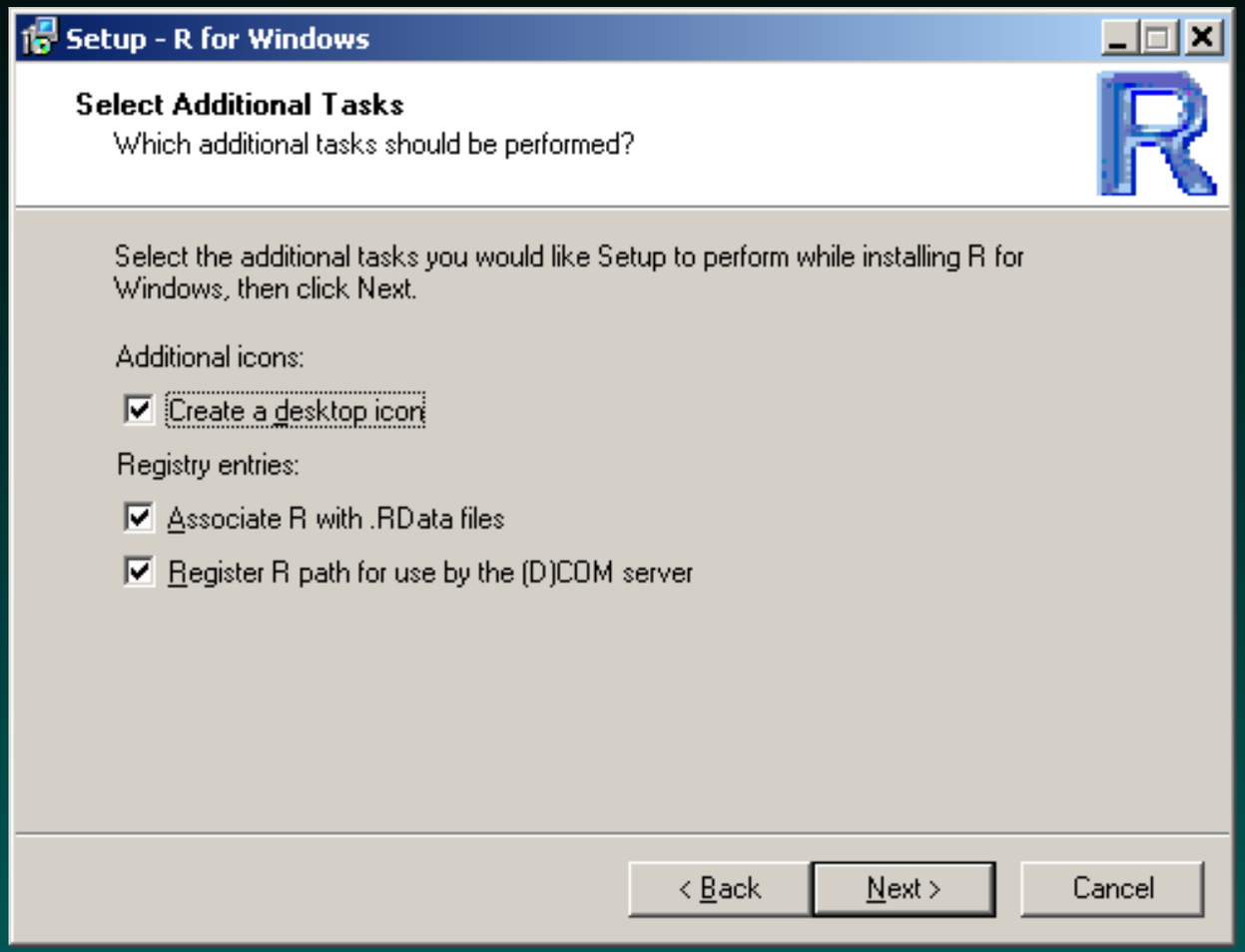
You can choose which pieces to install. In general, installing documentation and help files is a good idea.

R Installation



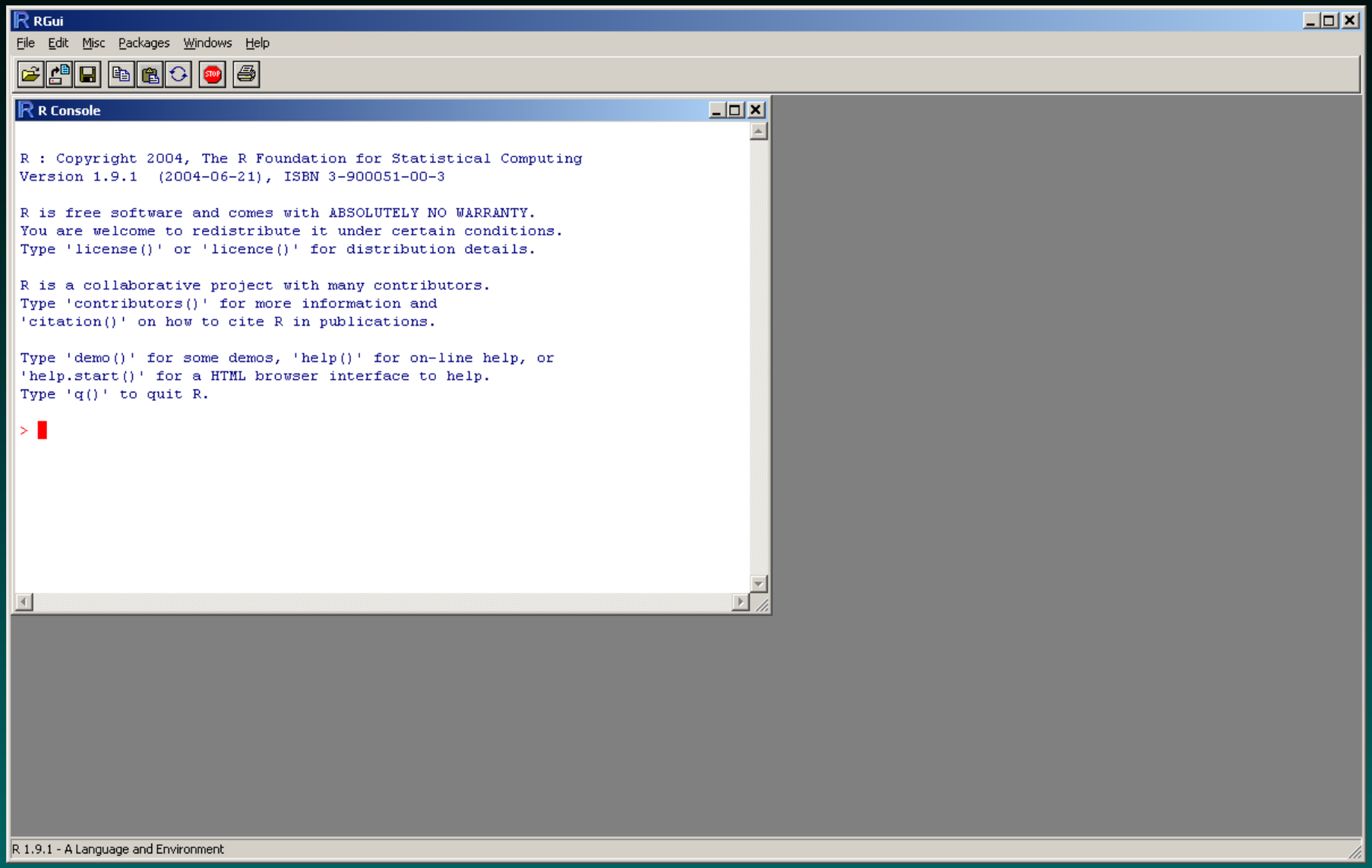
Decide whether to make a folder on the start menu.

R Installation

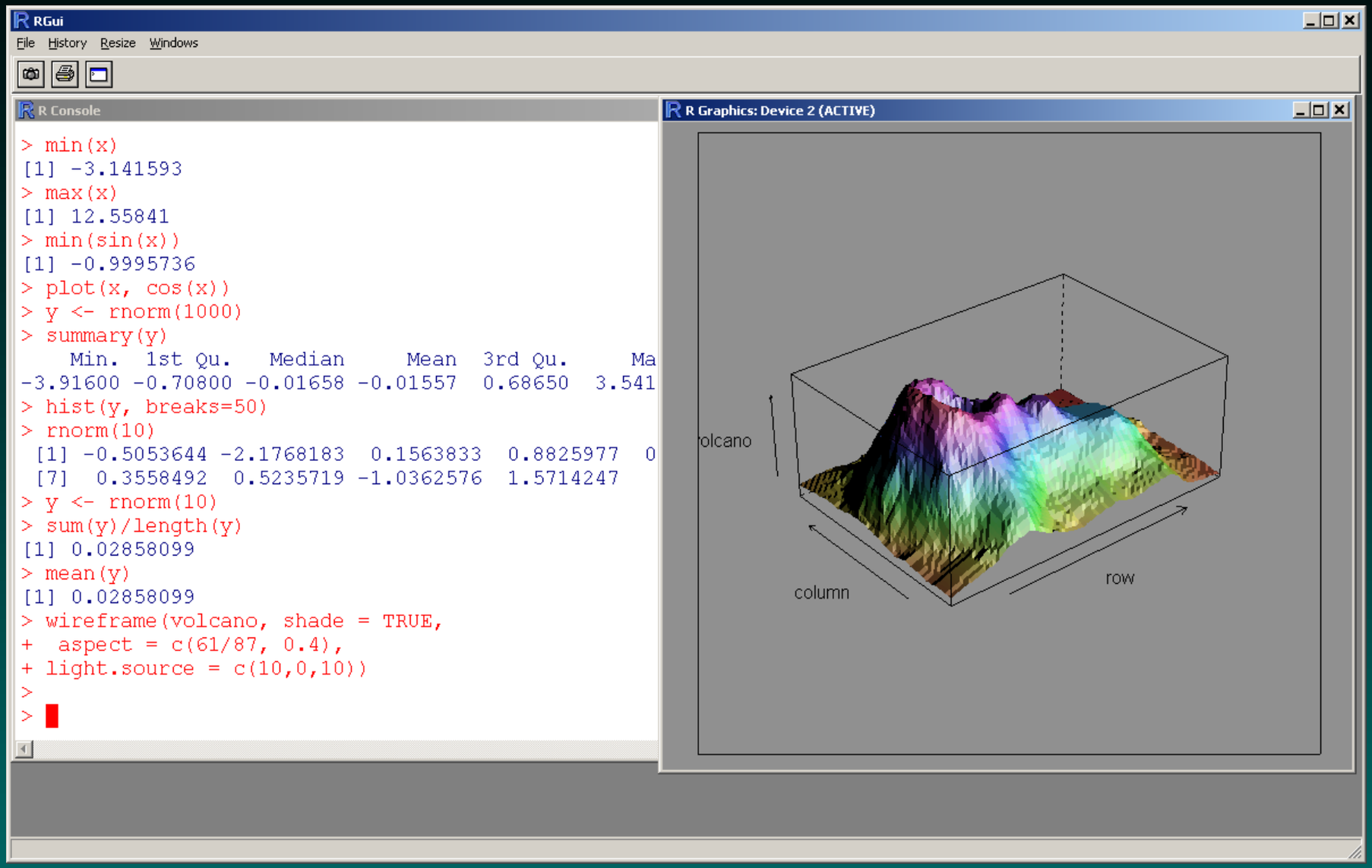


Decide whether to put an icon on the desktop. After this step, the program installs itself fairly quickly.

The R Gui


RGui
File Edit Misc Packages Windows Help
R Console
R : Copyright 2004, The R Foundation for Statistical Computing
Version 1.9.1 (2004-06-21), ISBN 3-900051-00-3
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.
>
R 1.9.1 - A Language and Environment

The R Gui


RGui
File History Resize Windows
R Console
> min(x)
[1] -3.141593
> max(x)
[1] 12.55841
> min(sin(x))
[1] -0.9995736
> plot(x, cos(x))
> y <- rnorm(1000)
> summary(y)
Min. 1st Qu. Median Mean 3rd Qu. Ma
-3.91600 -0.70800 -0.01658 -0.01557 0.68650 3.541
> hist(y, breaks=50)
> rnorm(10)
[1] -0.5053644 -2.1768183 0.1563833 0.8825977 0
[7] 0.3558492 0.5235719 -1.0362576 1.5714247
> y <- rnorm(10)
> sum(y)/length(y)
[1] 0.02858099
> mean(y)
[1] 0.02858099
> wireframe(volcano, shade = TRUE,
+ aspect = c(61/87, 0.4),
+ light.source = c(10,0,10))
>
R Graphics: Device 2 (ACTIVE)
olcano
column
row

Notes on R

At heart, R is a command line program. You type commands in the console window. Results are displayed there, and plots appear in associated graphics windows.

R always prints a prompt (usually $>$) where you can type commands. If a line does not contain a complete command, then R prints a continuation prompt (usually $+$).

To assign the value of a command to a variable, you use a "left arrow", made by typing $<$ (less than) followed by $-$ (minus), as in

```
x <- 2
```

This command produces no output; it simply stores the value "2" under the name "x". To retrieve the value, type the name of the variable.

```
> x
[1] 2
```

Note that the output is prefaced by the number “1” in brackets. Output often consists of vectors, and R tells you which item of the vector starts the output.

```
> y <- rnorm(10)
> y
 [1] -1.07521929 -1.15549677 -1.88800876 -0.89362362
 [5]  0.60838354 -2.11006124  0.41604637  0.52506983
 [9] -0.06416302 -0.22610929
```

The `rnorm` function generates random variables from the normal distribution.

R has lots of built-in functions.

```
> sum(y)
[1] -5.863182
> sum(y)/length(y)
[1] -0.5863182
> mean(y)
[1] -0.5863182
> sd(y)
[1] 0.9856325
```

You can get help on functions using (surprise) the `help` command. For example,

```
> help(rnorm)
```

will open a separate help window:

R Help on rnorm

```
RGui
File  Edit  Windows

R Console
> mean(y)
[1]
> s
[1]
> s
Err
> s
[1]
> m
[1]
> s
[1]
> s
[1]
[5]
> h
> h
> l
cha
> h
Err
> h
> h
> h
>

`rnorm' help
Normal                    package:stats                    R Documentation

The Normal Distribution

Description:

     Density, distribution function, quantile function and random
     generation for the normal distribution with mean equal to 'mean'
     and standard deviation equal to 'sd'.

Usage:

     dnorm(x, mean=0, sd=1, log = FALSE)
     pnorm(q, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
     qnorm(p, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
     rnorm(n, mean=0, sd=1)

Arguments:

     x,q: vector of quantiles.

       p: vector of probabilities.

       n: number of observations. If 'length(n) > 1', the length is
          taken to be the number required.
```

Packages

How do you find out which functions are available? Every function in R is in a package, and packages come with documentation. To get help on the “stats” package, you would type

```
help(package=stats)
```

This will open a help window containing one-line descriptions of all functions in the package.

When R starts, it loads the packages “base”, “utils”, “graphics”, and “stats”. Other packages must be loaded using the `library` command. Alternatively, you can use the menu item “Packages”, then “Load packages...”, which is available when the cursor is in the console window. (Note: Menu items in the R GUI change depending on the active subwindow.) You get a dialog box with a list of packages.

GUI Loading Library Packages

RGui

File Edit Misc Packages Windows Help

Select one

base
boot
class
cluster
ctest
eda
foreign
graphics
grid
KernSmooth
lattice
lqs
MASS
methods
mgcv
mle
modreg
mva
nlme
nls
nnet
rpart

OK Cancel

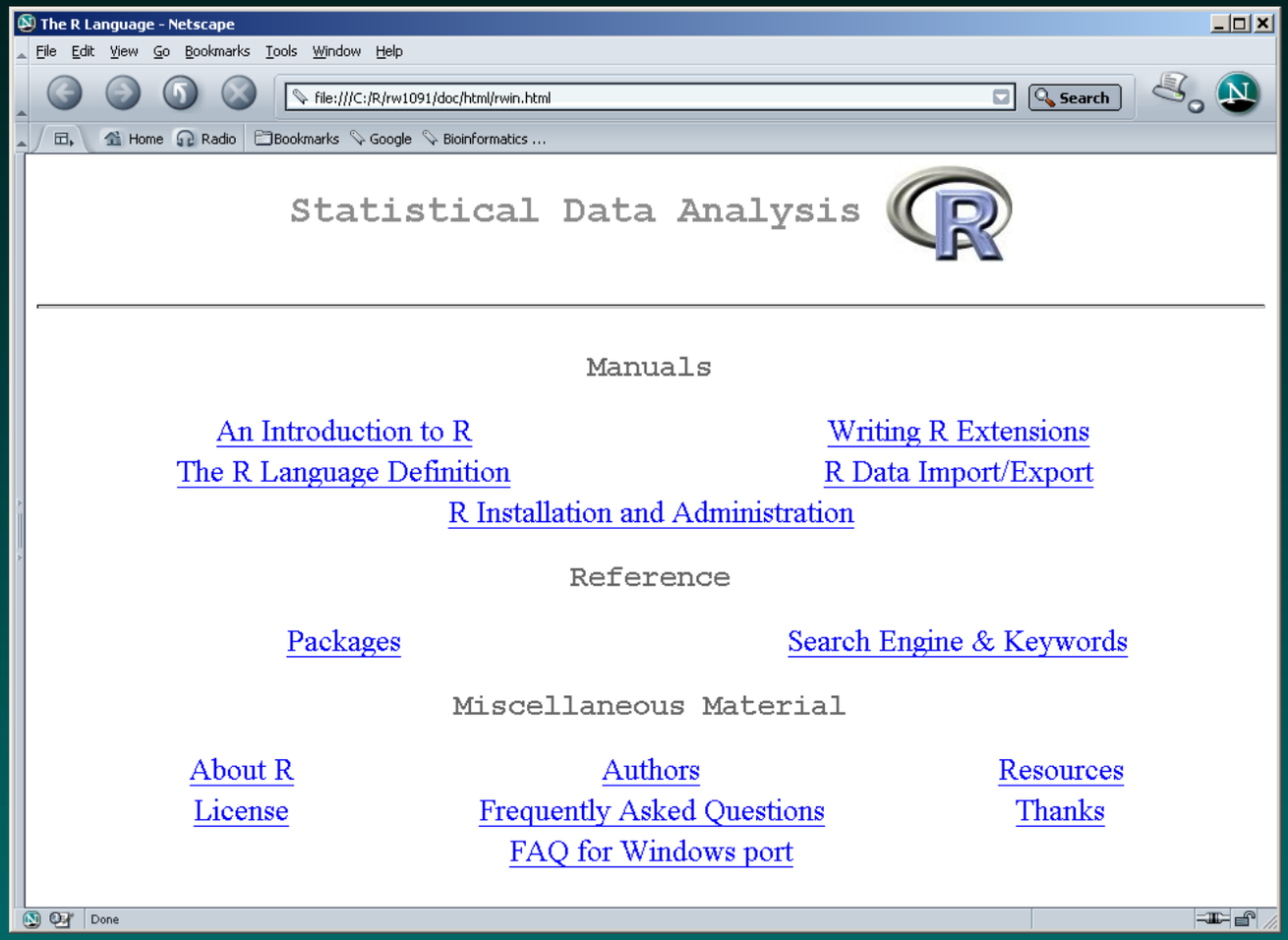
R Console

```
> sum(y)
[1] -5.863182
> sum(y)/length(y))
Error: syntax error
> sum(y)/length(y)
[1] -0.5863182
> mean(y)
[1] -0.5863182
> sd(y)
[1] 0.9856325
> search()
[1] ".GlobalEnv"        "pa                    "package:methods"  "package:stats"
[5] "package:graphics" "pa                    "Autoloads"        "package:base"
> help.search('autoload')
> help(autoload)
> ls('Autoloads')
character(0)
> help(stats)
Error in help(stats) : Object "stats" not found
> help(help)
> help(package='stats')
> help(rnorm)
> help(package=stats)
> local({pkg <- select.list(sort(.packages(all.available = TRUE)))
+ if(nchar(pkg)) library(pkg, character.only=TRUE)})
```

R 1.9.1 - A Language and Environment

Browser-based R Help

You can also use the GUI menu item “Help” followed by “Html help” to open a web browser with help information.



Homework Submission

There are both good and bad aspects of R's interactive command-line interface. On the good side, it is very flexible. It encourages exploration, allowing you to try things out and get rapid feedback on what works and what doesn't.

On the bad side, record-keeping and documentation are difficult. As a result, you may have a hard time reconstructing exactly how you solved a problem. You'll need to devise a method for keeping better records than are possible just by typing things at the command line.

The critical R command that makes this possible is `source`.

R Help for source

RGui - [`source' help]

File Edit Windows

```
source                  package:base                R Documentation

Read R Code from a File or a Connection

Description:

     'source' causes R to accept its input from the named file (the
     name must be quoted). Input is read from that file until the end
     of the file is reached.   'parse' is used to scan the expressions
     in, they are then evaluated sequentially in the chosen
     environment.

Usage:

     source(file, local = FALSE, echo = verbose, print.eval = echo,
            verbose = getOption("verbose"),  prompt.echo = getOption("prompt"$
            max.deparse.length = 150, chdir = FALSE)

Arguments:

    file: a connection or a character string giving the name of the
          file or URL to read from.

   local: if 'local' is 'FALSE', the statements scanned are evaluated
          in the user's workspace (the global environment), otherwise
          in the environment calling 'source'.
```

Using source

One method for keeping track of how you solve a problem is to create a file containing the commands with the solution. You then `source` this file to produce the answer. You can use a “plain text” editor (like Notepad, but not Microsoft Word) to modify the commands if they don’t work correctly the first time around.

Comments can be include in the file by prefacing them with a “hash” or “pound” sign (#), as in the example on the next slide.

```
# Partial solution to problem 1 of Assignment 1

# Create a vector of x-values
x <- seq(0, 3*pi, by=0.1)

# Plot the sine of x as a curve instead of as a
# bunch of unconnected points.
plot(x, sin(x), type='l')
```