



## DNA Microarray Data Analysis

IIRIS HOVATTA, KATJA KIMPPA, ANTTI LEHMUSSOLA, TOMI PASANEN,  
JANNA SAARELA, ILANA SAARIKKO, JUHA SAHARINEN, PEKKA TIIKKAINEN  
TEEMU TOIVANEN, MARTTI TOLVANEN, MAUNO VIHINEN AND GARRY WONG  
EDITORS JARNO TUIMALA AND M. MINNA LAINE

CSC

CSC – Scientific Computing Ltd. is a non-profit organization for high-performance computing and networking in Finland. CSC is owned by the Ministry of Education. CSC runs a national large-scale facility for computational science and engineering and supports the university and research community. CSC is also responsible for the operations of the Finnish University and Research Network (FUNET).

All rights reserved. The PDF version of this book or parts of it can be used in Finnish universities as course material, provided that this copyright notice is included. However, this publication may not be sold or included as part of other publications without permission of the publisher.

© The authors and  
CSC – Scientific Computing Ltd.  
2005

Second edition

ISBN 952-5520-11-0 (print)

ISBN 952-5520-12-9 (PDF)

<http://www.csc.fi/oppaat/siru/>

<http://www.csc.fi/molbio/arraybook/>

Printed at  
Picaset Oy  
Helsinki 2005

## List of Contributors

Iiris Hovatta  
National Public Health Institute  
Haartmaninkatu 8  
FI-00290 Helsinki  
Finland

Katja Kimppa  
PerkinElmer Life Sciences and Analytical Sciences  
- Wallac Oy  
P.O.Box 10  
FI-20101 Turku  
Finland

M. Minna Laine  
CSC, the Finnish IT center for science  
Keilaranta 14  
FI-02101 Espoo  
Finland

Antti Lehmussola  
Tampere University of Technology  
P.O.Box 553  
FI-33101 Tampere  
Finland

Tomi Pasanen  
University of Helsinki  
P.O.Box 68  
FI-00014 University of Helsinki  
Finland

Janna Saarela  
Biomedicum Biochip Center  
Haartmaninkatu 8  
FI-00290 Helsinki  
Finland

Ilana Saarikko  
University of Helsinki  
P.O.Box 68  
FI-00014 University of Helsinki  
Finland

Juha Saharinen  
National Public Health Institute  
Haartmaninkatu 8  
FI-00290 Helsinki  
Finland

Pekka Tiikkainen  
VTT  
P.O.Box 106  
FI-20521 Turku  
Finland

Teemu Toivanen  
Centre for Biotechnology  
Tykistökatu 6  
FI-20521 Turku  
Finland

Martti Tolvanen  
Institute of Medical Technology  
Biokatu 8  
FI-33520 Tampere  
Finland

Jarno Tuimala  
CSC, the Finnish IT center for science  
Keilaranta 14  
FI-02101 Espoo  
Finland

Mauno Vihinen  
Institute of Medical Technology  
Biokatu 8  
FI-33520 Tampere  
Finland

Garry Wong  
A. I. Virtanen -institute  
University of Kuopio  
FI-70211 Kuopio  
Finland

# 3 Web extra: Software issues

Jarno Tuimala and Teemu Toivanen

## 3.1 Programming

When the capacities of the spreadsheet program are not sufficient or many similar files need processing, some programming tools can be used instead for the datafile management purposes or for data analyses. There are actually several programming languages that are especially suited for the data file formatting and other text file manipulations.

### 3.1.1 Perl

One of the most common languages is Perl, which has very powerful and easy to use text manipulation tools. Perl is available for free for UNIX, Linux <http://www.perl.org> and PC machines <http://www.activeperl.com>. Perl is a programming language, which means that getting to know it takes some time and effort. However, if data conversion tools are needed everyday, it would definitely be worthwhile to befriend Perl.

To illustrate how easily text can be manipulated using Perl, we present a short example. The next code produces a complementary DNA sequence from the original sequence, which has been stored into the text string `$sequence`. Both the original sequence and its complement are printed on the computer screen.

The program starts with a line that tells the computer where The Perl software can be found. The next four lines contain the actual commands that manipulate the DNA sequences. Note that every command line has to end with a semicolon. Function `tr` makes a complementary sequence from the original one, and function `print` outputs the result to the screen.

```
#!/usr/bin/perl
$sequence="aaattcgagtaggtcaggcat";
print "Original:          $sequence\n";
$sequence=~ tr/acgtACGT/tgcaTGCA/;
print "Complementary:    $sequence\n";
```

You can use pico editor on CSC's Cedar server to create a similar file and test

the example yourself. Perl programs are started in Cedar with a command `perl filename`.

### 3.1.2 Awk

Awk is a standard UNIX and Linux tool, which is available on CSC's servers. With Awk, individual columns can be easily extracted from tab-delimited text files. Using other standard UNIX tools, these individual columns can be saved into a new text file. For example, the next script takes the first column from a specified datafile and saves it into a new file.

```
Awk '{print$1}' datafile > newfile
```

Two columns can be "awked" into a new file next to each other separated with a space:

```
awk '{print$1, $2}' datafile > newfile
```

or one after another:

```
awk '{print$1}{print$2}' datafile > newfile
```

### 3.1.3 R

R is a free statistical analysis tool and a self-sufficient programming language. It is available for UNIX, Linux, Macintosh and PC platforms. In R, scripts for analyses and data file manipulations can be easily constructed. R has many add-on packages for cluster analysis, self-organizing maps, and neural networks, among others. There are also many packages available that have been specifically tailored for DNA microarray data analysis: Bioconductor project develops and updates many of these packages.

Here is an example on how to read the tab-delimited datafile to R and how to process it into a new table, which is then written out to a new file. The function `read.table` reads in the specified file with the headers. The table is then saved in a variable `data`. Two columns are extracted from the data, and saved into new variable (`x` and `y`). The new variables are used for the creation of a new table (`dataout`), which is then written to a new text file (`filenameout.txt`). Such a script can easily be automated using R, and the analyses can simultaneously be integrated with the datafile conversions.

```
data<-read.table("filename.txt", header=T)
x<-data$greenintensity
y<-data$redintensity
dataout<-cbind(x,y)
sink("filenameout.txt")
dataout
```

```
sink()
```

Many images included in chapters 5–8 have been produced by R using real DNA microarray datasets.

## 3.2 Common code examples

This section covers common problems when manipulating text formatted datafiles that most programs export and import. Problems is that those file formats are program specific and using them for analysis is hard without a lot of editing. These are on Linux systems, but same code should work on all UNIX flavours and even in windows with minor changes.

For the actual code examples, see online material at <http://www.csc.fi/oppaat/siru/>.

### 3.2.1 Read tabular data

This is the basis for rest of the examples (copy/paste this first and then add what you want from the latter examples). Reads tabular files (change variables for sectioned tabular data). Please note that examples are in both Perl and Python languages and cannot be intermixed.

#### Perl

```

1  %%
2  %% This is file '.tex',
3  %% generated with the docstrip utility.
4  %%
5  %% The original source files were:
6  %%
7  %% fileerr.dtx (with options: 'return')
8  %%
9  %% This is a generated file.
10 %%
11 %% Copyright 1993 1994 1995 1996 1997 1998 1999
12 %% The LaTeX3 Project and any individual authors listed elsewhere
13 %% in this file.
14 %%
15 %% This file was generated from file(s) of the Standard LaTeX 'Tools Bundle'.
16 %% -----
17 %%
18 %% It may be distributed and/or modified under the
19 %% conditions of the LaTeX Project Public License, either version 1.2
20 %% of this license or (at your option) any later version.
21 %% The latest version of this license is in
22 %% http://www.latex-project.org/lppl.txt
23 %% and version 1.2 or later is part of all distributions of LaTeX
24 %% version 1999/12/01 or later.
25 %%
26 %% This file may only be distributed together with a copy of the LaTeX

```

```

27 %% 'Tools Bundle'. You may however distribute the LaTeX 'Tools Bundle'
28 %% without such generated files.
29 %%
30 %% The list of all files belonging to the LaTeX 'Tools Bundle' is
31 %% given in the file 'manifest.txt'.
32 %%
33 \message{File ignored}
34 \endinput
35 %%
36 %% End of file '.tex'.
#!/usr/bin/perl -w
#filename to read, get from commandline
my $filename=shift;
#open file (F is the filehandle)
open(F, $filename);
#specify data delimiter \t = tab, \; = semicolon
$delim="\t";
#this will have the data matrix
@data=();
#put this to 1 if first row is a header
$hasHeader=1;
#this includes header
@headerData=();
#use filter to specify when data section starts and ends
#not needed, for normal tabular data
$useFilter=0;
#search for this string in the beginning of the line to start data section
$startFilter="BEGIN DATA";
#search for this string in the beginning of the line to end data section
$endFilter="END DATA";
#internal variable so that we know when we are at data section
$doInsert=0;
#loop while there's data
while($line=<F>) {
  chomp($line);
  if ($useFilter==1) { # go here if we are using filters
    if ($doInsert==0) { # not in data section
      if ($line =~ /$startFilter/) { #search for beginning of data
        $doInsert=1; #go to data phase
      }
      next;
    } else { # we are in data phase
      if ($line =~ /$endFilter/) { #search for the end of data
        $doInsert=0; #exit data phase
        next;
      }
    }
  }
  #data insertion section
  if ($hasHeader==1) {
    $hasHeader=-1; # we come here only once
    #split header row with $delim to a vector
    @headerData=split($delim, $line);
  }
}

```

```

    next; #read next line
}
#add new row to matix data, that has $line split with $delim
#to a vector (which is that row of the matrix)
push(@data, [split($delim, $line)]);
}
#now we have data in @data matrix and possible header info in @haderData

```

## Python

```

#!/usr/bin/python
from sys import argv #for filename
import re,string # regular expressions and strings
#filename to read, get from commandline
filename=argv[1]
#open file (f is the filehandle)
f = open(filename, 'r')
#specify data delimiter \t = tab, \; = semicolon
delim="\t"
#this will have the data matrix
data=[]
#put this to 1 if first row is a header
hasHeader=1
#this includes header
headerData=[]
#use filter to specify when data section starts and ends
#not needed, for normal tabular data
useFilter=0
#search for this string in the beginning of the line to start data section
startFilter="BEGIN DATA"
#search for this string in the beginning of the line to end data section
endFilter="END DATA"
#internal variable so that we know when we are at data section
doInsert=0
#loop while there's data
line=f.readline()
while len(line) != 0:
    next=0
    line=re.sub("\n|\r","",line)
    if (useFilter==1):# go here if we are using filters
        next=1
        if (doInsert==0): # not in data section
            if (re.search(startFilter, line) != None): #search for beginning of data
                doInsert=1 #go to data phase
                next=1 #do not append data
        else: # we are in data phase
            next=0
            if (re.search(endFilter, line) != None): #search for the end of data
                doInsert=0 #exit data phase
                next=1 #do not append data
    #data insertion section
    if (hasHeader==1):

```

```

hasHeader=-1 # we come here only once
#split header row with $delim to a vector
headerData=string.split(line, delim)
next=1 #do not append data

#add new row to matix data, that has $line split with $delim
#to a vector (which is that row of the matrix)
if next != 1:
    data.append(string.split(line, delim))
line = f.readline()

#now we have data in a matrix and possible header info in haderData

```

### 3.2.2 Filtering/Rows

Select what column to filter (column) and what to search (search).

#### Perl

```

#column number to use for filtering (columns nubered from 0,1,2...)
$colnum=0; # first column
#what to search
$search="1";
#check each data row
@filteredData=();
for my $row (@data) {
    #exact match 'eq', numric match '==', numeric greater or eual >=
    # =~ /$search/ for regexp match
    if ($row->[$colnum] eq $search) {
        push(@filteredData, $row);
    }
}
#replace @data with filtered data @ret
@data=@filteredData;

```

#### Python

```

#column number to use for filtering (columns nubered from 0,1,2...)
colnum=0 # first column
#what to search
search="1"
#check each data row
filteredData=[]
for row in (data):
    #exact match '==', numric match '==', numeric greater or eual >=
    # re.search(pattern, string for regexp match
    if (row[colnum] == search):
        filteredData.append(row)
#replace data with filtered data ret
data=filteredData

```

## Filtering/Columns

### Perl

Change "cols" list to reflect wanted columns

```
#column number in a list (rest is filtered out, columns nubered from 0,1,2...)
@cols=(0,2); # first and third column
@filteredData=();
#header cols
if (scalar(@headerData)>0) { #check that header is not empty
    @headerData=@headerData[@cols];
}
for my $r (@data) {
    my @row=@{$r};
    my @tmp;
    @tmp=@row[@cols];
    push(@filteredData, \@tmp);
}
#replace @data with filtered data @ret
@data=@filteredData;
```

### Python

```
#column number in a list (rest is filtered out, columns nubered from 0,1,2...)
cols=[0,2] # first and third column
filteredData=[]
#header cols
if (len(headerData)>0): #check that header is not empty
    tmp=[]
    for x in cols: # append all columns
        tmp.append(headerData[x])
    headerData=tmp
# for each row
for row in (data):
    tmp=[]
    for x in cols:
        tmp.append(row[x]) # append all columns
    filteredData.append(tmp)
#replace data with filtered data ret
data=filteredData
```

### 3.2.3 File conversions/print/write

First print header (if exists) and then rest of the data with selected delimiter. Use > 'filename' to print to a file.

### Perl

```
# open(FILEOUT, "> outfile.txt"); #remove leading hash for file writing
# also change STDOUT to FILEOUT later on this file
#output delim
```

```
$outDelim="";
if (scalar(@headerData)>0) { #check that header is not empty
    #print header with $outDelim as delimiter
    print STDOUT join($outDelim,@headerData);
    print STDOUT "\n"; #new line
}
#for each row
for my $row (@data) {
    #print row vector with $outDelim as delimiter
    print STDOUT join($outDelim,@$row);
    print STDOUT "\n"; #new line
}
# close(FILETOUT); #see open(..) comments
```

### Python

```
#output delim
outDelim="";
if (len(headerData)>0): #check that header is not empty
    #print header with outDelim as delimiter
    print string.join(headerData,outDelim)
#for each row
for row in data:
    #print row vector with outDelim as delimiter
    print string.join(row,outDelim)
```