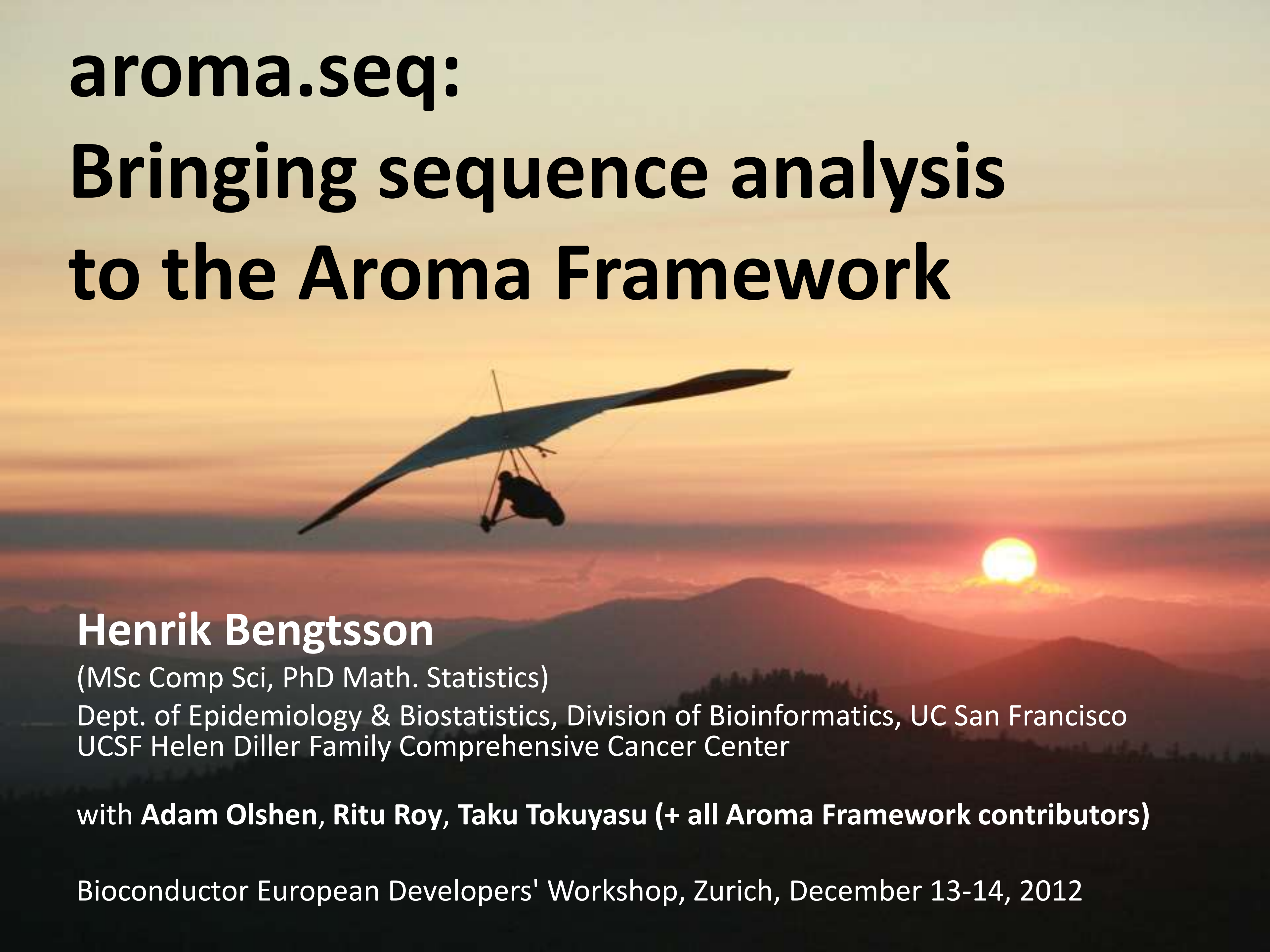
aroma.seq: Bringing sequence analysis to the Aroma Framework



Henrik Bengtsson
(MSc Comp Sci, PhD Math. Statistics)
Dept. of Epidemiology & Biostatistics, Division of Bioinformatics, UC San Francisco
UCSF Helen Diller Family Comprehensive Cancer Center

with **Adam Olshen**, **Ritu Roy**, **Taku Tokuyasu (+ all Aroma Framework contributors)**

Bioconductor European Developers' Workshop, Zurich, December 13-14, 2012

Thank you all!

- Organizers Mark Robinson and Michal Okoniewski.
- Irene Hofmann et al.
- Institute of Molecular Life Sciences,
  ETH Zurich, and University of Zurich.
- The Bioconductor Project/Team & its developers.
- All presenters and participants!

Outline

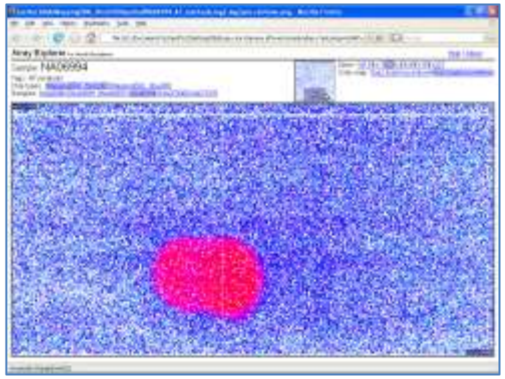
- **Overview of the Aroma Framework.**
- aroma.seq: proof-of-concept DNAseq analysis.
- My tips and tricks for large data analysis.

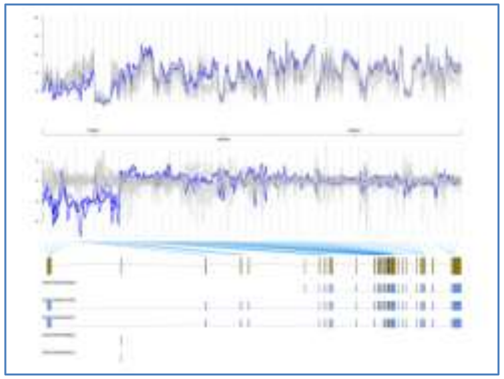
This is a 25-minute presentation, where the first two parts take 20 minutes and the last part 5 minutes.

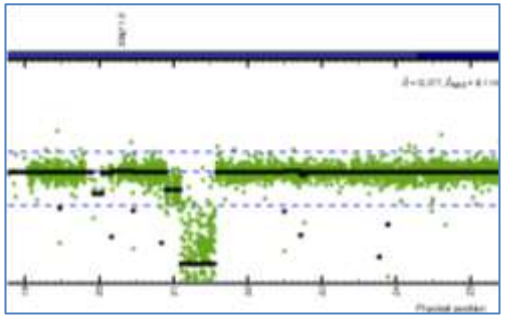
The Aroma Framework

The Aroma Framework
- Worry-free large-scale analysis in R

- **Unlimited data sizes**, e.g. 10,000 Affymetrix microarrays.
- **Persistent memory**, results live beyond R's quit().
- **Fault tolerant**, e.g. recovery even from power failures.
- **Portable / shareable**, i.e. same script works everywhere.
- **Cross platform**, e.g. Unix, OS X, Windows.
- Leverages **CRAN and Bioconductor packages**.
- **Reproducible research**.
- **Extendable**, i.e. add your own methods.
- aroma-project.org







Some numbers:

Since 2006. ~500 installs last month. ~800 on mailing list.

100+ citations. 100,000+ lines (excl. comments)

R.filesets is the core and knows about files

```
setA/
  fileA,20100112.csv
  fileB,other,tags.tsv
  fileC,inverted.csv
  fileD,3cols.csv

> library(R.filesets)
> df <- GenericDataFile("setA/fileA,20100112.csv")
> df
GenericDataFile:
Name: fileA
Tags: 20100112
Full name: fileA,20100112
Pathname: setA/fileA,20100112.csv
File size: 2.88 MB (2,949,102 bytes)
RAM: 0.00 MB
> getChecksum(df)
[1] "fcb889d29d51c600409d242e03d7d779"
```

```
> df <- TabularTextFile("setA/fileA,20100112.csv")
> df
TabularTextFile:
Name: fileA
Tags: 20100112
Full name: fileA,20100112
Pathname: setA/fileA,20100112.csv
File size: 2.88 MB (2,949,102 bytes)
RAM: 0.00 MB
Number of data rows: 17987
Columns [4]: 'x', 'y', 'fac', 'char'
Number of text lines: 18004
> readDataFrame(df, rows=c(5,4,1),
                         colClasses=c("(x|y)"="integer"))
   x  y
5 10 5
4 12 4
1 19 1
```

R.filesets makes it easy to handle large sets of files of any size and any type

```
> ds <- GenericDataFileSet$byPath("setA/")
> ds
GenericDataFileSet:
Name: setA
Number of files: 4
Names: fileA, fileB, fileC, fileD [4]
Path (to the first file): setA/
Total file size: 10.00 MB
RAM: 0.01MB

> lapply(ds, FUN=getChecksum)
$`fileA,20100112`
[1] "fcb889d29d51c600409d242e03d7d779"
$`fileB,other,tags`
[1] "e0e0d2750626df38cedab8796cfa6459"
…
```

```
> ds <- TabularTextFileSet$byPath("setA/")
> ds
TabularTextFileSet:
Name: setA
Number of files: 4
Names: fileA, fileB, fileC, fileD [4]
Path (to the first file): setA/
Total file size: 10.00 MB
RAM: 0.01MB

> readDataFrame(ds, rows=c(1,5),
        colClasses=c("(x|y)"="integer"))
     x  y
1.1 19 1
1.5 10 5
2.1 15 4
2.5 32 9
…
```

aroma.affymetrix: Analyzing small and large Affymetrix data sets

Standardized and strict file structure:

annotationData/chipTypes/**HG-U133_Plus_2**/**HG-U133_Plus_2**.CDF
rawData/**GSE13159**/**HG-U133_Plus_2**/*.CEL (2096 files)

```
> library(aroma.affymetrix)
> dsR <- AffymetrixCelSet$byName("GSE13159",  chipType="HG-U133_Plus_2")
> dsR

AffymetrixCelSet:
Name: GSE13159
Path: rawData/GSE13159/HG-U133_Plus_2
Chip type: HG-U133_Plus_2
Number of arrays: 2096
Names: GSM329407, GSM329408, GSM329409, ..., GSM331732 [2096]
Total file size: 27.09 GB
RAM: 2.19MB
```

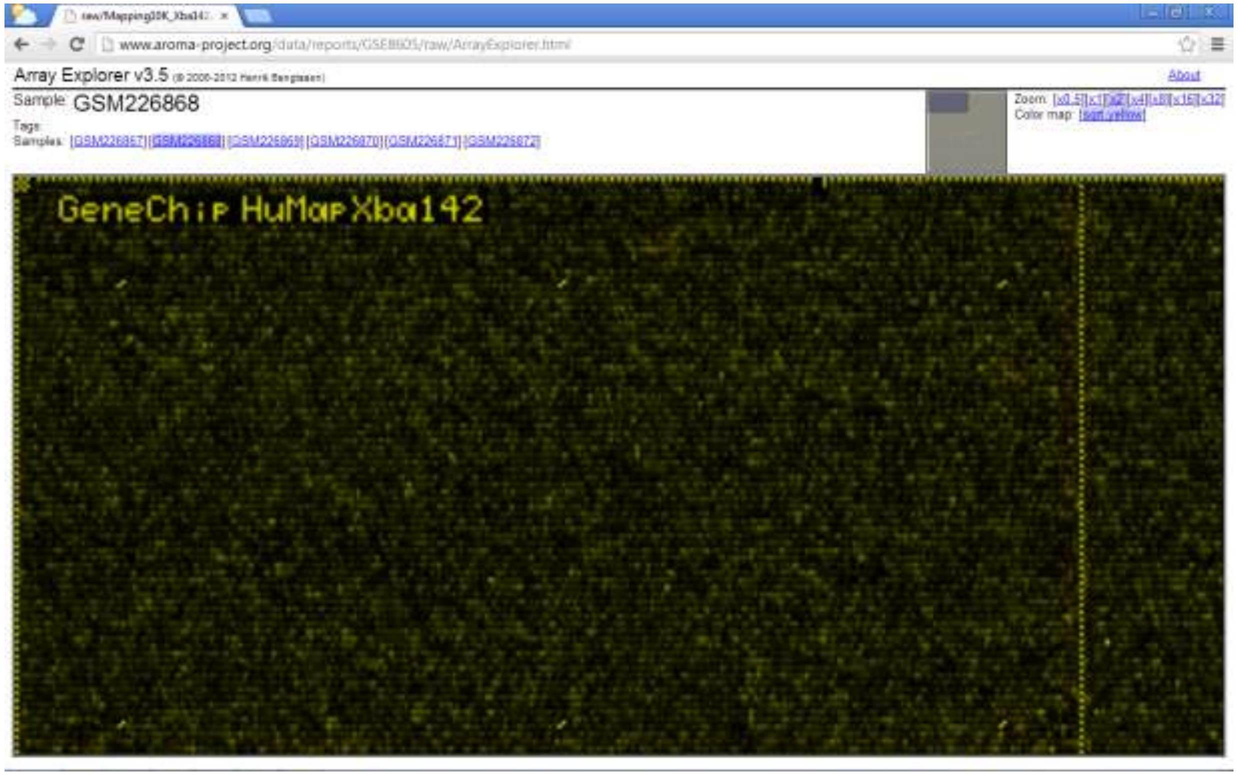
Example: RMA on 2,096 arrays

```
> dsR <- AffymetrixCelSet$byName(“GSE13159”, chipType=“HG-U133_Plus_2”)
> ces <- doRMA(dsR)
> eset <- extractExpressionSet(ces)
> eset

ExpressionSet (storageMode: lockedEnvironment)
assayData: 54675 features, 2096 samples
  element names: exprs
protocolData: none
phenoData: none
featureData: none
experimentData: use 'experimentData(object)‘
Annotation: hgu133plus2
```

Example: Spatial visualization of arrays

```
> dsR <- AffymetrixCelSet$byName("GSE8605", chipType="Mapping10K_Xba142")
> ex <- ArrayExplorer(dsR)
> process(ex)
```



raw/Mapping10K_Xba142
www.aroma-project.org/data/reports/GSE8605/raw/ArrayExplorer.html
About
GeneChip HuMapXba142

Array Explorer v3.5 (© 2006-2012 Henrik Bengtsson)

Sample: GSM226868

Tags:

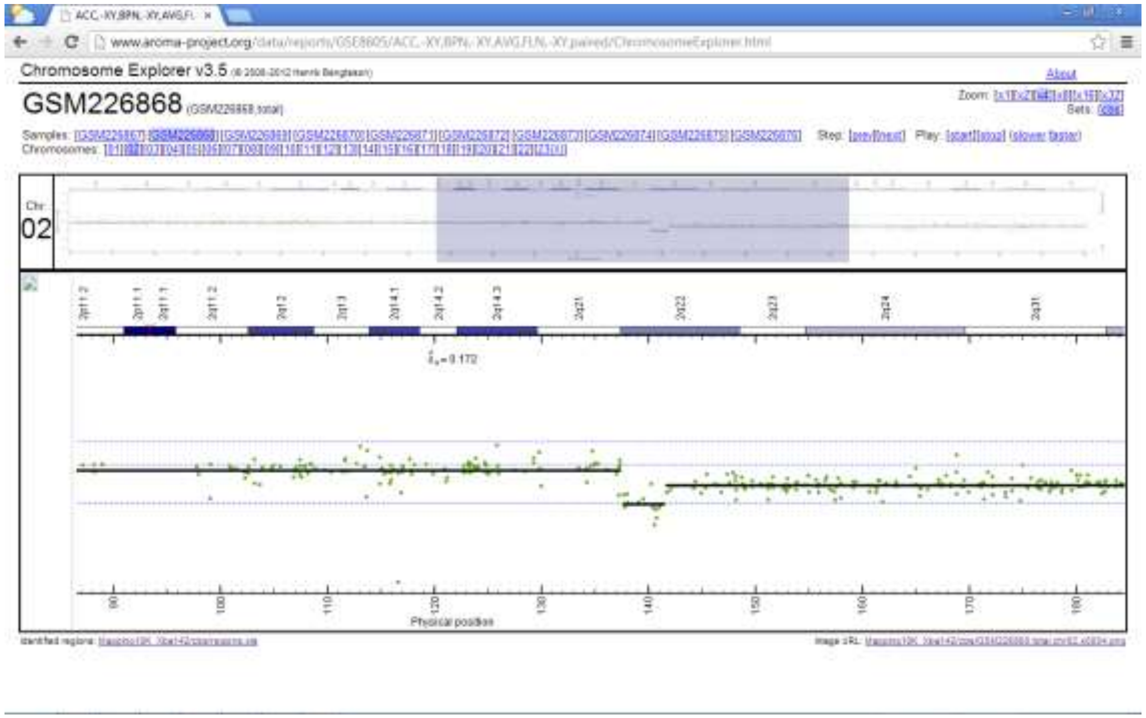
Samples: [GSM226867][GSM226868][GSM226869][GSM226870][GSM226871][GSM226872]

Zoom: [x0.5][x1][x2][x4][x8][x16][x32]

Color map: [sqrt,yellow]

Example: DNA copy number segmentation

> dsR <- AffymetrixCelSet$byName("GSE8605", chipType="Mapping10K_Xba142")
>
> dsN <- **doCRMAv2(dsR)**
>
> seg <- **CbsModel(dsN)**
>
> ex <- **ChromosomeExplorer(seg)**
>
> process(ex)


ACC,-XY,BPN,-XY,AVG,FL
www.aroma-project.org/data/reports/GSE8605/ACC,-XY,BPN,-XY,AVG,FLN,-XY,paired/ChromosomeExplorer.html
Chromosome Explorer v3.5
About
GSM226868
Zoom:
Sets:
Samples:
Step: [prev][next]
Play: [start][stop] (slower faster)
Chromosomes:
Chr
02
Physical position

Software Engineering

Software Design:

- All in R ("**R is the glue**").
- **Cross platform**, e.g. Unix, OS X, Windows.
- Leverages **CRAN and Bioconductor packages**.
- **Standardization**, e.g. file & directory structure.
- "Functional in the small, OO in the large" [Luke Hoban (F#) via John D. Cook (The Endeavour blog)]

Software Quality: [code base is 100,000+ lines (excl. comments)]

- Rich set of system, redundancy and reproducibility tests (> 24 CPU hours).
- All releases are validated so they don't break any downstream packages.
- Embrace bug/error reports.
- Software robustness, e.g. asserting arguments and results.

Outline

- Overview of the Aroma Framework.
- **aroma.seq: proof-of-concept DNAseq analysis**
- My tips and tricks for large data analysis.

aroma.seq

aroma.seq: Start-to-end NGS analysis in R

Currently (before bringing it into BioC):

- Sequence analysis is done with a variety of software via the command line.
- Error prone, e.g. manual file handling and lots of tedious parameter specifications.
- Highly specific to a given computer environment/setup.
- Complicated to share script.

Objectives aroma.seq:

- Everything available at the **R prompt**.
- Utilize **Bioconductor tools** and external tools such as Bowtie, BWA, TopHat and Cufflink.
- Reproducible research, e.g. **easy to share scripts**.
- **Automate tedious tasks**, e.g. sorting and indexing of BAM files, handling SAM Read Groups.
- Provide **standardized pipelines**, e.g. DNAseq copy number analysis with strong quality control.
- Transparent utilizing of **compute clusters.**
  => Same script for single-thread as compute cluster processing.
- **Availability**: Early 2013 by request. Mid/late 2013 publicly.

Classical total copy-number analysis with low-coverage DNAseq

Data

- DNASeq: Illumina
- Multiplex: 20 samples per lane
- Low depth: **0.2x** coverage per sample

Acknowledgements and original method approach

- Ilari Scheinin, Daoud Sie, Bauke Ylstra (VUMC, Amsterdam)

Classical total copy-number analysis with low-coverage DNAseq (in 7 steps)

1. Load R package

```
library(aroma.seq)
capabilitiesOf(aroma.seq)
```

=> bowtie2, bwa, gatk, picard, samtools …

2. Setup DNAseq data

```
# Setup FASTQ files
dsR <- FastqDataSet$byName("SCC", “Solexa")
```

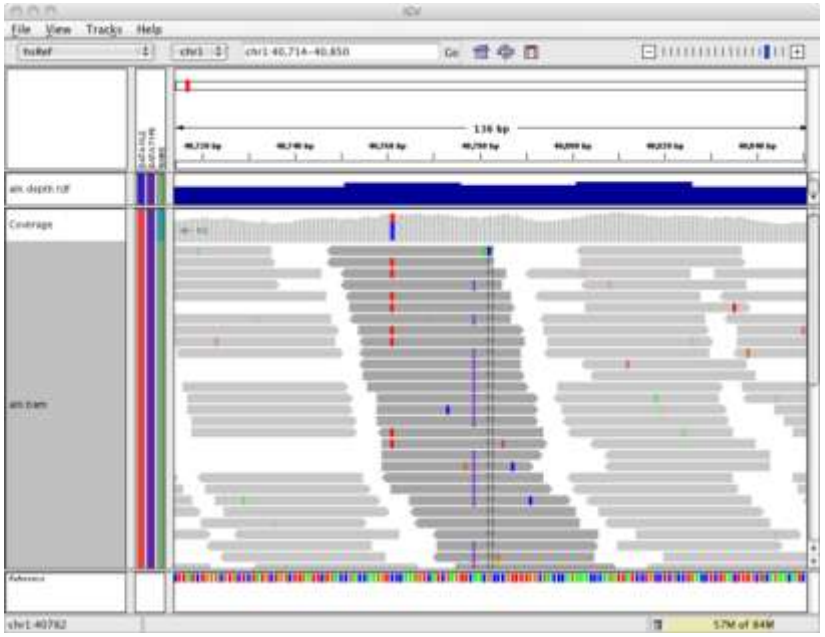
Unlimited number of samples can be loaded even on small computers, e.g. 1 or 10,000.

Classical total copy-number analysis with low-coverage DNAseq (in 7 steps)

3. Align reads to genome

```
# Setup (FASTA) genome reference
fa <- FastaReferenceFile$byName("human_g1k_v37")

# Burrows-Wheeler Alignment (FASTQ -> BAM)
alg <- BwaAlignment(dsR, ref=fa, n=2, q=40)
bs <- process(alg)
```

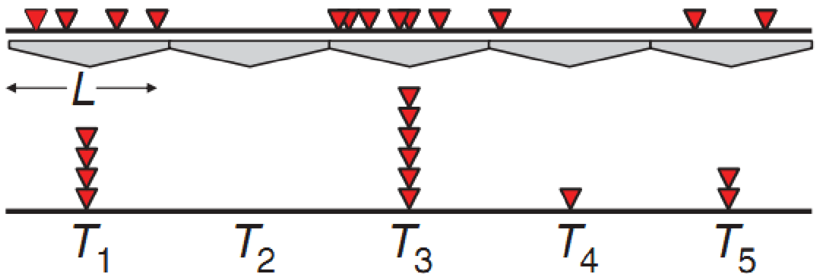


Internal validation detects common user mistakes and data errors so they are not propagated in the analysis. User do not have to deal with tedious details (e.g. SAM header groups).

Classical total copy-number analysis with low-coverage DNAseq (in 7 steps)

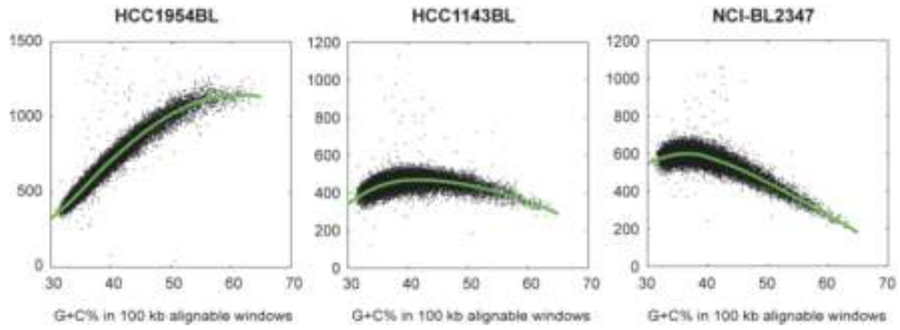
4. Bin and count reads

```
# (BAM -> Aroma count files)
ugp <- getAromaUgpFile(fa, “50kb")
bc <- TotalCnBinnedCounting(bs, targetUgp=ugp)
dsB <- process(bc)
```



5. Normalize for GC content

```
bgn <- BinnedGcNormalization(dsB)
dsG <- process(bgn)
```



Removing GC content effects makes it possible to estimate copy numbers without a reference.

Image courtesy: Chiang et al. (2009)

Classical total copy-number analysis with low-coverage DNAseq (in 7 steps)

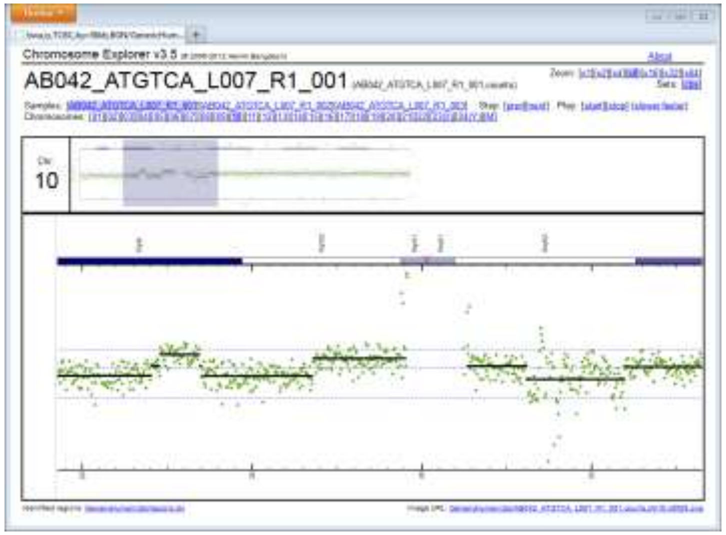
6. Segmenting total CNs

seg <- CbsModel(dsG)
fit(seg)

The aroma.seq package leverages highly specialized sequencing and statistical tools.

7. Chromosome Explorer

ce <- ChromosomeExplorer(seg)
process(ce)



A Chromosome Explorer report can be viewed in any modern web browser (offline and online).

Outline

- Overview of the Aroma Framework.
- aroma.seq: proof of concept DNAseq analysis.
- **My tips and tricks for large data analysis.**

Constant Memory Utilization

“Even if it works for you today, assume that tomorrow there will be no machine in Universe that can fit all of your data into RAM.”

Also as a non-programming statistician you can help out a lot

- Already **from day #1**, design your method (statistical model and/or algorithm) such that only a **fixed-size subset** of the data needs to be in memory at any time.

- **Load data** into memory **only when needed** and discard as soon as possible.

- This will also make it **much easier to parallelize** your methods later.

Classical example: Rank-based Quantile Normalization

- The **naive approach requires all samples** to be loaded into memory from start, but…
- …with a **two-pass read of the data, only two samples** need to be kept in memory at any time.

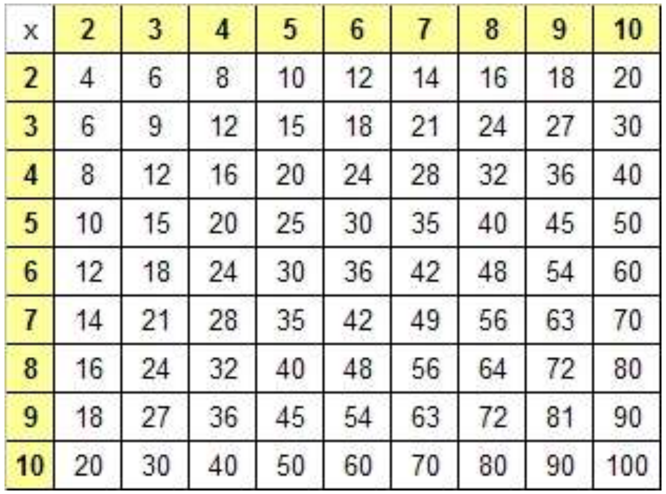
Memoization

“Memorize the results of repetitive computationally expensive tasks”

Each kid learn memoization in school

Question: What is 7 times 8?

1. Multiply(7, 8) = 8 + 8 + 8 + 8 + 8 + 8 + 8 = … = 56
2. Memorize (multiplication table):



x	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

3. Multiply(7, 8) = { “look up memoized result” } = 56

R.cache memoizes to file

```
getbdry <- function(nperm, beta, aux=NA) {
  # 1. Already calculated?
  key <- list("getbdry", nperm=nperm, beta=beta)  <= FULL CONTROL
  if (!is.null(res <- loadCache(key))) return(res)

  # 2. Calculate (takes a long time)
  res <- DNAcopy::getbdry(nperm=nperm, beta=beta)

  # 3. Store result (across R sessions)
  saveCache(res, key=key)


  res
}

getbdry(1000, 0.5)  # <= Slow!
getbdry(1000, 0.5)  # <= Instant from cache.
```

Related packages:

- digest
- Biobase::cache()
- memoise
- cacher
- filehash
- …

Software Robustness

"Errors WILL occur one way or the other!

–

write your code so
the impact of errors is minimal and
make sure they don't pass undetected"

Long-running analyzes needs fault tolerant software

Typical errors:

- Software bugs.
- User passes non-expected argument values.
- Corrupt data files.
- Session interrupts, e.g. sysadm reboot a computer.
- Hardware failures, e.g. power outage and network failures.

Don’t let errors propagate - catch them ASAP

Pre- and post-condition contracts; each function asserts that:

- the arguments received, and
- the returned values

are of proper types and have proper values, otherwise an exception is thrown. For instance, if a function returns a p-value, assert that it is indeed in [0,1] before returning.

Example:

```
stopifnot(length(p) == 1 && 0 <= p && p <= 1)
```

```
library(R.utils)
p <- Arguments$getNumeric(p, range=c(0,1))
```

Atomicity
- Don’t generate incomplete results

```
png("myPlot.png", width=640, height=480)
curve(dnorm, from=-3, to=+3)
abline(v=log("1"))
dev.off()
```

Use on.exit() whenever possible

```
myPlot <- function() {
  png("myPlot.png", width=640, height=480)
  on.exit(dev.off())
  curve(dnorm, from=-3, to=+3)
  abline(v=log("1"))
}
myPlot()
```

R.devices generates image files atomically

```
library("R.devices")

toPNG("myPlot", aspectRatio=3/4, {
  curve(dnorm, from=-3, to=+3)
  abline(v=log("1"))
})
```

The default behavior of toPNG() is to generate either complete image files or none (atomic). This is achieved by:

1. Write to a temporary file
2. Rename file only iff code complete successfully

This strategy also works with more serious software interrupts (e.g. power failures) and not only for image files.

Distributed processing

"…is awesome, R helps you a lot,
but it's not business as usual."

Also advanced developers run into unexpected problems with parallelized computing

- **Time outs and errors WILL occur** and compute nodes will go down, leaving unfinished/corrupt results. In other words, write fault-tolerant code.
- **Do NOT assume that file updates are instantaneous**, e.g. it can take up to 30 seconds for one machine to see a file modification of another machine.
- **SQLite does NOT guarantee atomic updates across machines** - you will eventually corrupt your database if you assume that.
(It’s only a valid assumption on a single machines with properly setup)
- **Do NOT assume your processes are automagically synchronized** - when scaling up such mistakes will come back and bite you (…and hopefully you notice).
- **Above errors are hard to troubleshoot**, because they only occur once in a while.

Thank you!