

# Rolexa: Probabilistic Base Calling of Solexa Sequencing Data

Jacques Rougemont  
Bioinformatics & Biostatistics Core Facility,  
EPFL School of Life Sciences,  
Lausanne, Switzerland

March 30, 2012

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Environment variables</b>	<b>1</b>
<b>3</b>	<b>Loading data</b>	<b>3</b>
<b>4</b>	<b>Data transforms</b>	<b>4</b>
<b>5</b>	<b>Base calling</b>	<b>4</b>
<b>6</b>	<b>Filtering and saving</b>	<b>5</b>
<b>7</b>	<b>Batch execution</b>	<b>7</b>
<b>8</b>	<b>Diagnostic plots</b>	<b>7</b>
<b>9</b>	<b>Session Information</b>	<b>12</b>

## 1 Introduction

This package provides an alternative base calling algorithm using model-based clustering (*mclust*) and probability theory to identify ambiguous bases and code them with IUPAC symbols. We also select optimal sub-tags using a score based on information content to remove uncertain bases towards the ends of the reads. There are also a few diagnostic plots functionalities. Details of the algorithms were published in [1].

## 2 Environment variables

The *Rolexa* package uses a `RolexaRun` object to store the various parameters of the run, and uses the *ShortRead* for manipulating data, in particular many *Rolexa* functions take a `SolexaPath` object as argument.

We load the library and create a configuration with default parameters except for the `idsep` variable:

```
> library(Rolexa)
> rolenv = SetModel(idsep="_")
> GetModel(rolenv)
```

```
$MinimumTagLength
[1] 15
```

```
$SequencingLength
[1] 36
```

```
$Barcode
[1] 0
```

```
$HThresholds
[1] 0.5849625 1.3219281 1.8073549
```

```
$IThresholds
 [1] 2.058894 2.115477 2.169925 2.222392 2.273018 2.321928 2.369234 2.415037
 [9] 2.459432 2.502500 2.544321 2.584963 2.624491 2.662965 2.700440 2.736966
[17] 2.772590 2.807355 2.841302 2.874469 2.906891 2.938599 2.969626 3.000000
[25] 3.029747 3.058894 3.087463 3.115477 3.142958 3.169925 3.196397 3.222392
[33] 3.247928 3.273018 3.297681 3.321928
```

```
$PET
[1] FALSE
```

```
$fit
[1] FALSE
```

```
$normal
[1] TRUE
```

```
$decorrelate
[1] "both"
```

```
$verbose
[1] 0
```

```
$colors
 [1] "black"      "green"      "blue"      "chocolate3" "red"
 [6] "#007F7F"   "#66B20E"   "#7F7F00"   "#66338E"     "#7F007F"
[11] "#E6330E"   "#7F464E"   "#7F6035"   "#6C5649"     "#685F4C"
[16] "gray"
```

```
$idsep
[1] "_"
```

The meaning of these parameters is as follows:

**MinimumTagLength** tags shorter than this will not be saved

**SequencingLength** number of sequencing cycles, used to calculate the number of columns in files

**Barcode** number of bases used as barcode at the beginning of the tag

**HThresholds** entropy thresholds between 1 and 2-base ambiguities, 2 and 3-base ambiguities and 3-base ambiguity or undecided (the default is  $\log_2(c(1.5, 2.5, 3.5))$ )

**IThresholds** total entropy thresholds, as a function of tag length (the default is  $\log_2(4 + 1 : 36/6)$ )

**PET** paired-end sequencing run

**fit** use full EM convergence instead of only one-step optimization if TRUE

**normal** use tile-level normalization before base-calling if TRUE

**decorrelate** use 'cycle'-level decorrelation procedure, 'channel'-level, 'both' or 'none'

**idsep** character separating coordinate fields in sequence headers (default is ":",")

**verbose** print debug information if > 0

### 3 Loading data

Loading data is done using the *ShortRead* utilities (in particular the `SolexaPath` class) with two additional wrappers `CombineReads` and `CombineFastQ`:

```
> path = SolexaPath(system.file("extdata", package="ShortRead"))
```

Then use the loading functions to read a selection of those files:

```
> (int = readIntensities(path,pattern="s_1_0001",withVariability=FALSE))
```

```
class: SolexaIntensity
dim: 256 4 36
readInfo: SolexaIntensityInfo
intensity: ArrayIntensity
measurementError: not available
```

```
> (seq = CombineReads(run=rolenv,path=path,pattern="s_1_0001_seq*"))
```

```
class: ShortRead
length: 256 reads; width: 36 cycles
```

```
> (seq_fastq = readFastq(path))
```

```
class: ShortReadQ
length: 256 reads; width: 36 cycles
```

## 4 Data transforms

Before going into the base calling itself, we can perform several data transformations to remove some of the systematic biases:

1. Reduce cross-talk between color channels

```
> (theta=OptimizeAngle(int=int))[1:10,]  
  
      [,1]      [,2]      [,3]      [,4]  
[1,] 0.7767119 1.375080 0.4721182 1.557188  
[2,] 0.7653824 1.377907 0.5618510 1.570796  
[3,] 0.7276859 1.367992 0.5290140 1.570796  
[4,] 0.7551378 1.384266 0.6453509 1.570796  
[5,] 0.7349694 1.377229 0.6220983 1.570796  
[6,] 0.7377151 1.383378 0.6556697 1.564773  
[7,] 0.7213154 1.377866 0.6412864 1.570796  
[8,] 0.7685749 1.384597 0.6472642 1.570796  
[9,] 0.7681729 1.387350 0.5537521 1.570796  
[10,] 0.7710965 1.379977 0.6961033 1.570796  
  
> int=DeCorrelateChannels(int=int,theta=theta)
```

2. Reduce dephasing along cycles

```
> (rate=OptimizeRate(int=int))  
  
[1] 0.01760222  
  
> int=DeCorrelateCycles(int=int,rate=rate)
```

3. Reduce position-dependent bias within each tile

```
> int2=TileNormalize(run=rolenv,int=int)
```

## 5 Base calling

The base calling algorithm fits a gaussian mixture model to the four-dimensional intensity values from each cycle. Sequences from a previous base calling, if available, are used to seed the algorithm:

```
> (res=SeqScore(run=rolenv,int=int,seqInit=seq,cycles=1:36))$sread
```

A DNASTringSet instance of length 256

```
width seq  
[1] 36 TTGTTTTTCATGTGATTTTAAAAATGTATTTGTTTGT  
[2] 36 TCCAAACTGGTAGACAATACAAACATTCTCAAATCT  
[3] 36 TGCACCTGATAGGGTCTCTGCTCTGAGAGAGDAAGK  
[4] 36 TATGAGAGTAGCYAATGCCACAAAGWSGRKGTGKBY
```

```

[5] 36 TAGTAGGTGTCCTATTCTGATGCYCAGCACGCCAAG
[6] 36 GAGAGAACTGAAAATCACAGAATATGAGAAATAGAC
[7] 36 GCAGAGACCCACAASCCAGCCAAGCGGCTCCWGACC
[8] 36 GAGATATTTATTGAACACTAACACTCTGTCATGCAA
[9] 36 GGTGAAAAGGAAGCAYCCCSYTYTCYGCTTAYAT
... ..
[248] 36 TGGGGAGMYGKGGGMYMTGGCKGGMYRTHHRWVVDK
[249] 36 GTGGAGGCTAGCACCTGTTTGTGGCBTTGTGARGBA
[250] 36 GATTTTCAAAGTTAAGGGTAAAAATGTTATCACCCG
[251] 36 GAAAATGAGAAACATACAATTGACGACTTGAAAAAT
[252] 36 GGYATTTTCCTTTTGTTTTATTTMRCTTTGKWGBDH
[253] 36 GGTAGGRAGAGCTCGTGKCCGTCTTCTGCTTRAW
[254] 36 GAAAAACGWAAAAATGAGAAWTGCACACTGTAGRA
[255] 36 GATTCCTTATGTGGTAATGAAAAATAATATTTTCATC
[256] 36 GGATGAGAAGAATAGTATATTACATCTCTAGCCACA

```

## 6 Filtering and saving

The base calling results consist of a full-length tag with base quality entropy scores, which can then be filtered to extract the best sequence tag for each colony. This is where the parameters `IThresholds` comes into play:

```

> rolenv@MinimumTagLength = as.integer(1)
> (res2 = FilterResults(run=rolenv,results=res))$sread

```

```

A DNAStringSet instance of length 256
  width seq
[1] 36 TTGTTTTTCATGTGATTTTAAAAATGTATTTGTTTGT
[2] 36 TCCAAACTGGTAGACAATACAAACATTCTCAAATCT
[3] 36 TGCACCTGATAGGGTCTCTGCTCTGAGAGAGDAAGK
[4] 28 TATGAGAGTAGCYAATGCCACAAAGWSG
[5] 36 TAGTAGGTGTCCTATTCTGATGCYCAGCACGCCAAG
[6] 36 GAGAGAACTGAAAATCACAGAATATGAGAAATAGAC
[7] 36 GCAGAGACCCACAASCCAGCCAAGCGGCTCCWGACC
[8] 36 GAGATATTTATTGAACACTAACACTCTGTCATGCAA
[9] 21 GGTGAAAAGGAAGCAYCCC
... ..
[248] 10 TGGGGAGMYG
[249] 34 GTGGAGGCTAGCACCTGTTTGTGGCBTTGTGARG
[250] 36 GATTTTCAAAGTTAAGGGTAAAAATGTTATCACCCG
[251] 36 GAAAATGAGAAACATACAATTGACGACTTGAAAAAT
[252] 30 GGYATTTTCCTTTTGTTTTATTTMRCTTTG
[253] 33 GGTAGGRAGAGCTCGTGKCCGTCTTCTGCTTR
[254] 36 GAAAAACGWAAAAATGAGAAWTGCACACTGTAGRA
[255] 36 GATTCCTTATGTGGTAATGAAAAATAATATTTTCATC
[256] 36 GGATGAGAAGAATAGTATATTACATCTCTAGCCACA

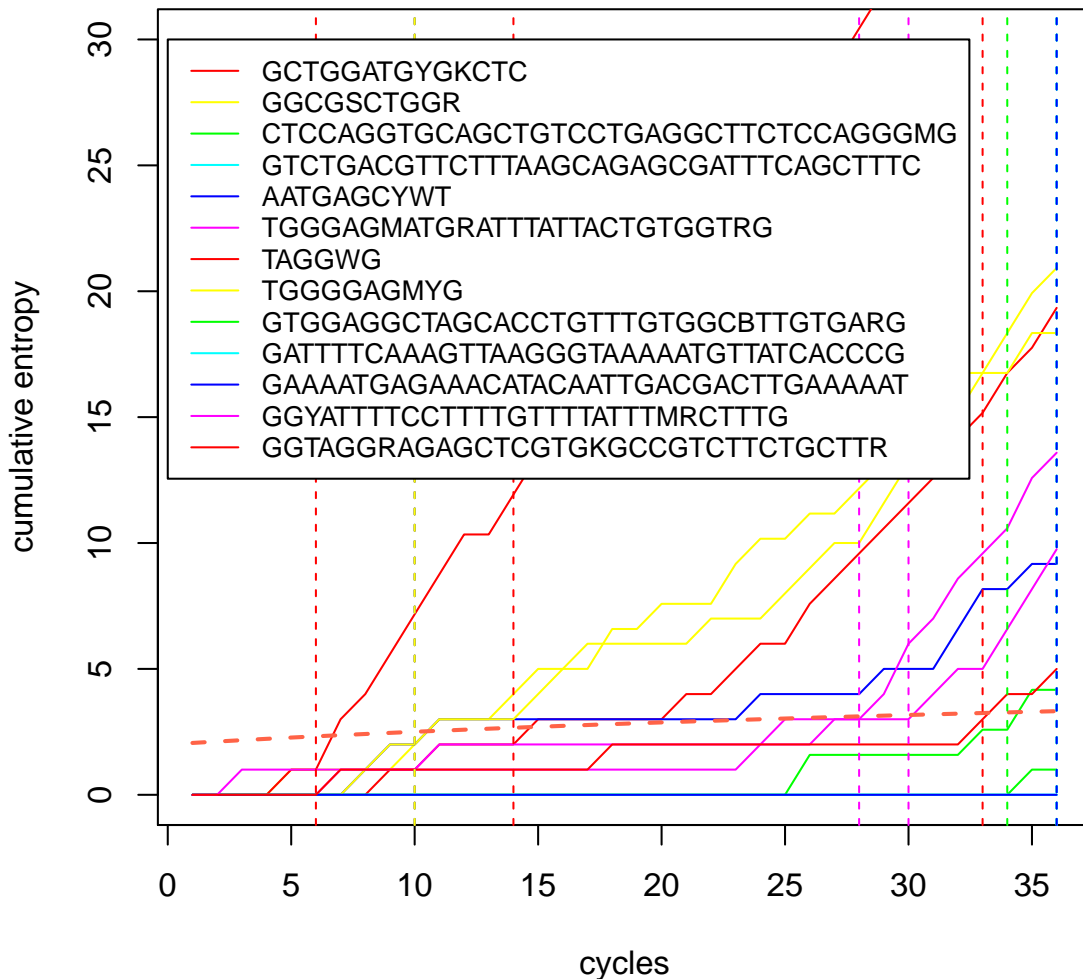
```

```

> str = as.matrix(res$sread[241:253])
> nt = DNA_ALPHABET
> post.entropy = matrix(0,nrow=nrow(str),ncol=36)
> post.entropy[which(str %in% nt[5:10])] = 1
> post.entropy[which(str %in% nt[11:14])] = log2(3)
> post.entropy[which(str == 'N')] = 2
> matplot(1:36,y=apply(post.entropy,1,cumsum),t='l',lty=1,col=rainbow(6),ylim=c(0,30),xlim=
> lines(1:36,rolenv@IThresholds,t='l',lty=2,lwd=2,col="tomato")
> abline(v=nchar(res2$sread[241:253]),col=rainbow(6),lty=2)
> legend(x=0,y=30,res2$sread[241:253],col=rainbow(6),lty=1,bg="white",cex=.8)

```

### Tag length cutoff



The final step is to save results:

```

> SaveResults(run=rolenv,results=res2,outpath="./")

```

## 7 Batch execution

The whole sequence of operations needed to load, analyse, filter and save a sequencing run can be performed in parallel (using calls to the *fork* package) via the function `ForkBatch`:

```
> library(fork)
> ForkBatch(run=rolenv,path=path,outpath="./",prefix="rs_",nthreads=2,nfiles=5,lane=1,tile
```

Each of the `nthreads` threads will execute a call to

```
> OneBatch(run=rolenv,path=path,lane=1,tiles=tiles[n:m],outpath="./",prefix="rs_")
```

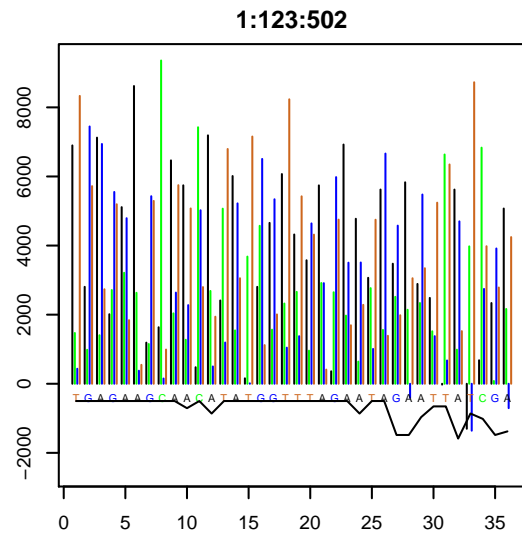
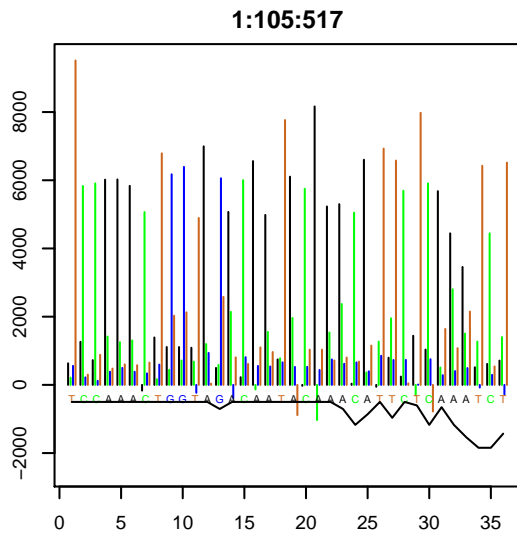
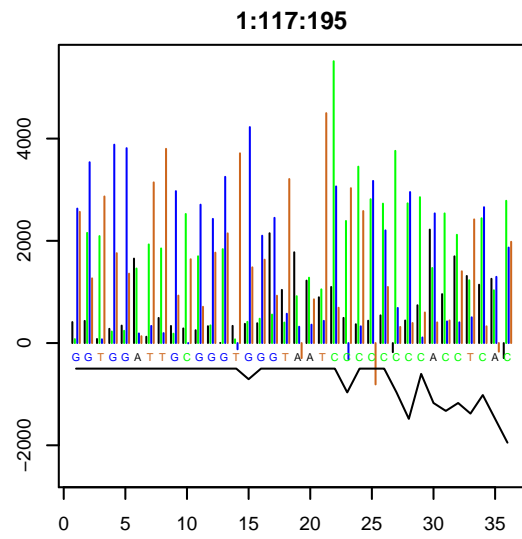
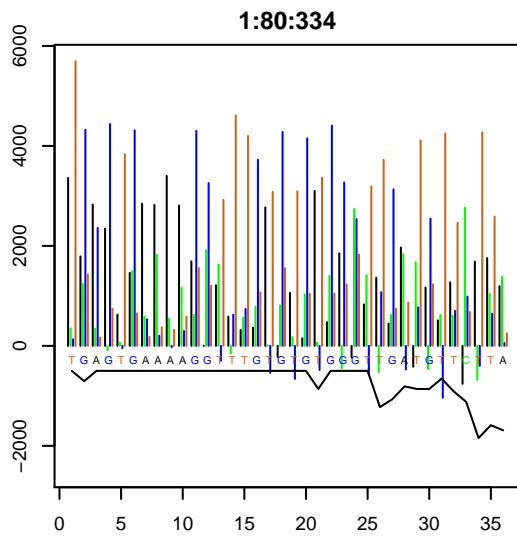
This function can be used in a loop on single-processor systems or in independent jobs distributed on a computing cluster.

## 8 Diagnostic plots

There are multiple possibilities for evaluating the quality of the base calling, at the level of each sequence, tile or lane.

Given a sequence tag, the corresponding raw intensities and a base quality score, we can use `CombinedPlot`:

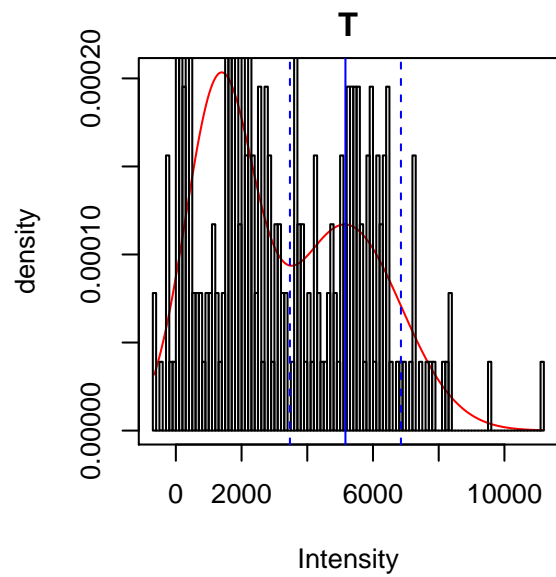
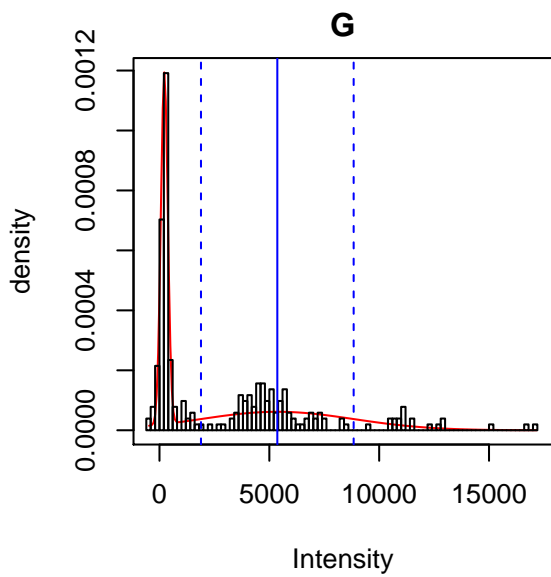
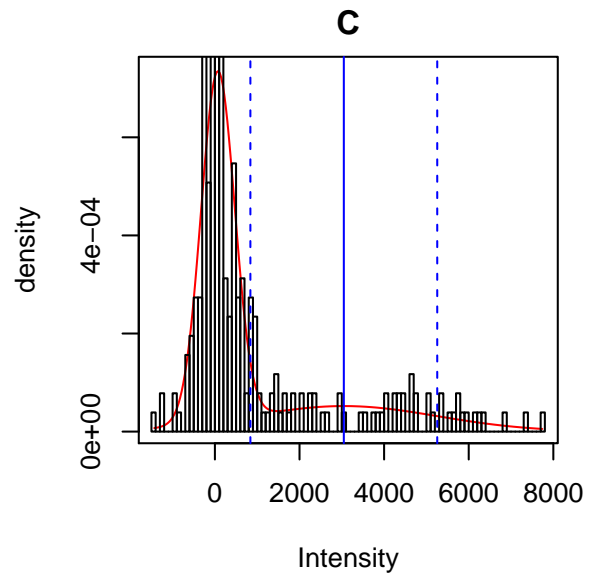
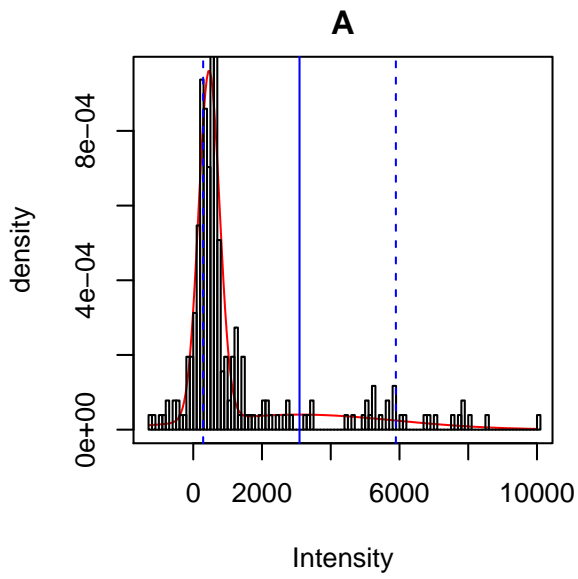
```
> CombinedPlot(run=rolenv,int=int,seq=seq,scores=as(quality(seq_fastq),"matrix"),colonies=
```



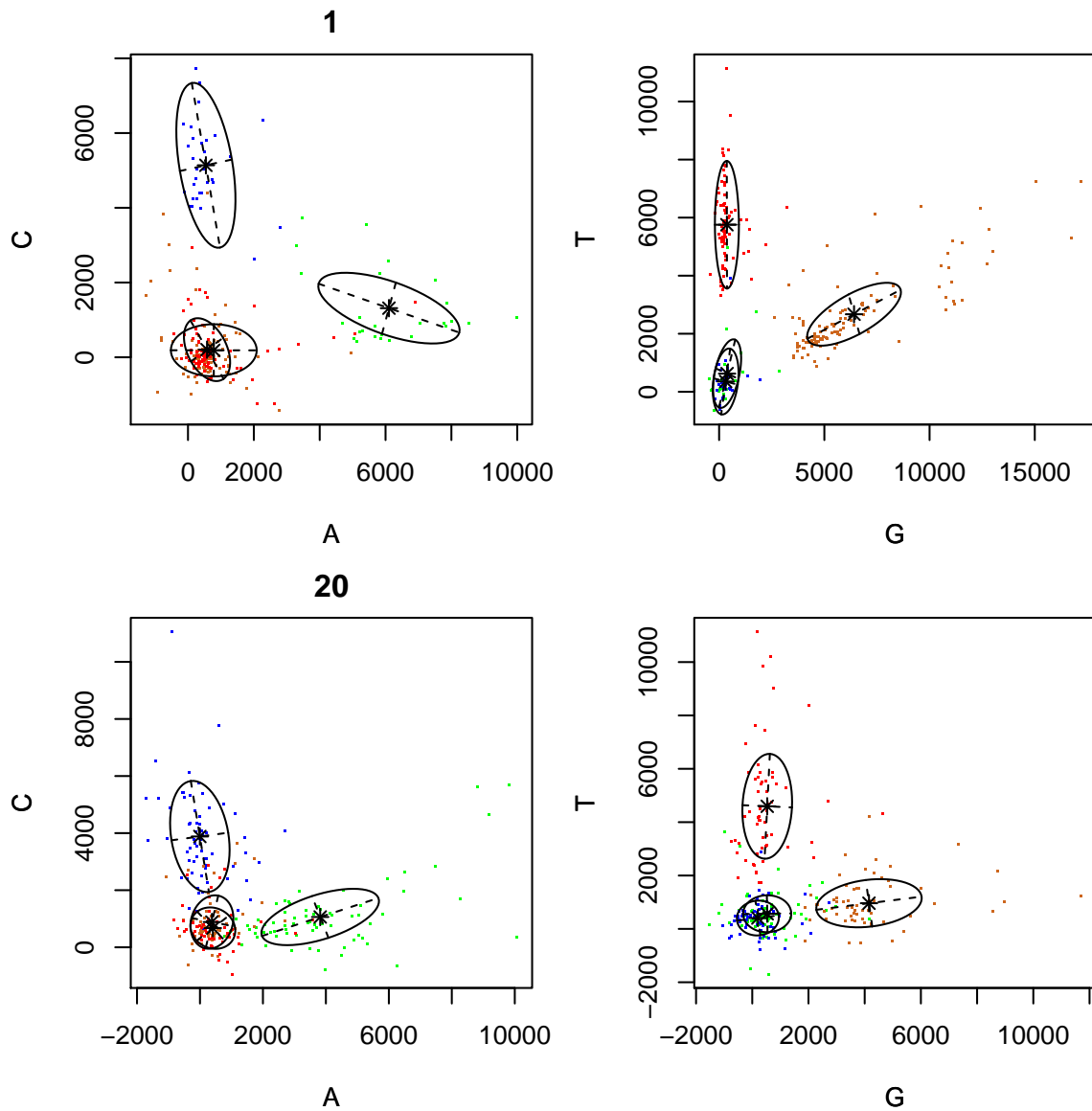
we can also evaluate the distribution of intensity values at selected cycles via 1- and 2-dimensional projections:

```
> ChannelHistogram(int=int,cycles=1,par=list(mfrow=c(2,2),mar=c(4, 4, 2, 1)+.1))
```



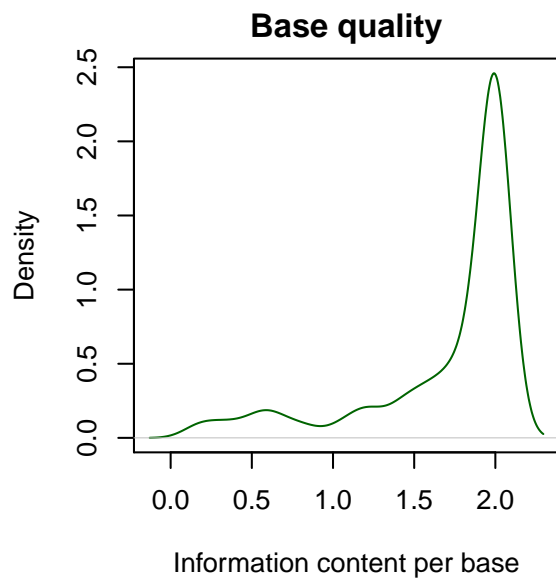
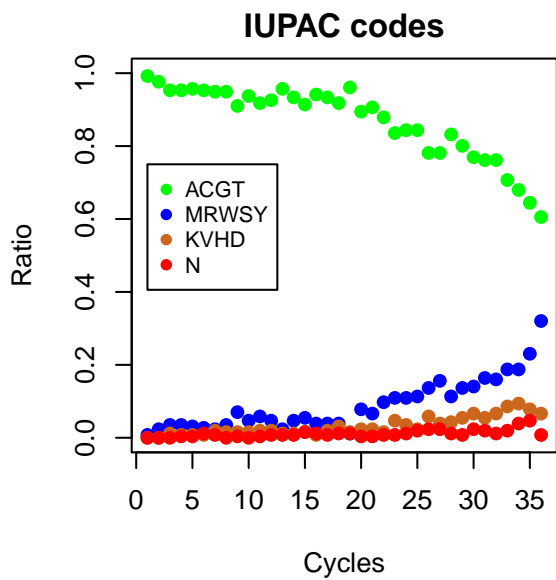
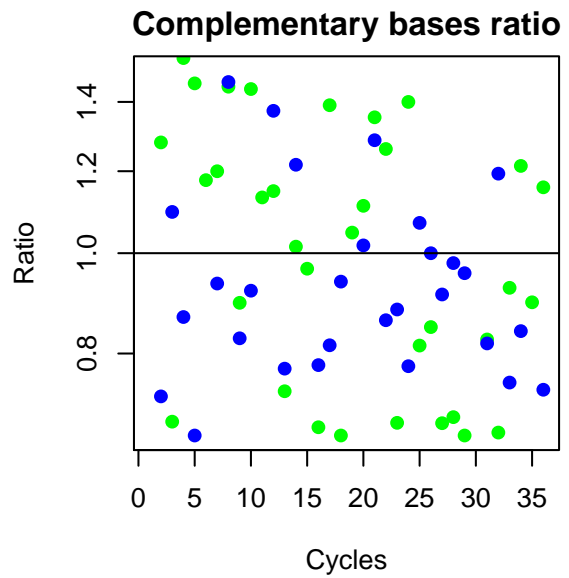
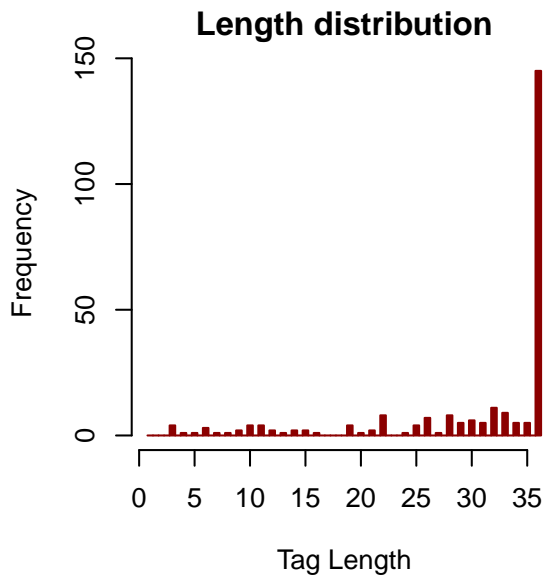


```
> par(mfrow=c(2,2),mar=c(4, 4, 2, 1)+.1)
> PlotCycles(run=rolenv,int=int,seq=seq,cycles=c(1,20))
```



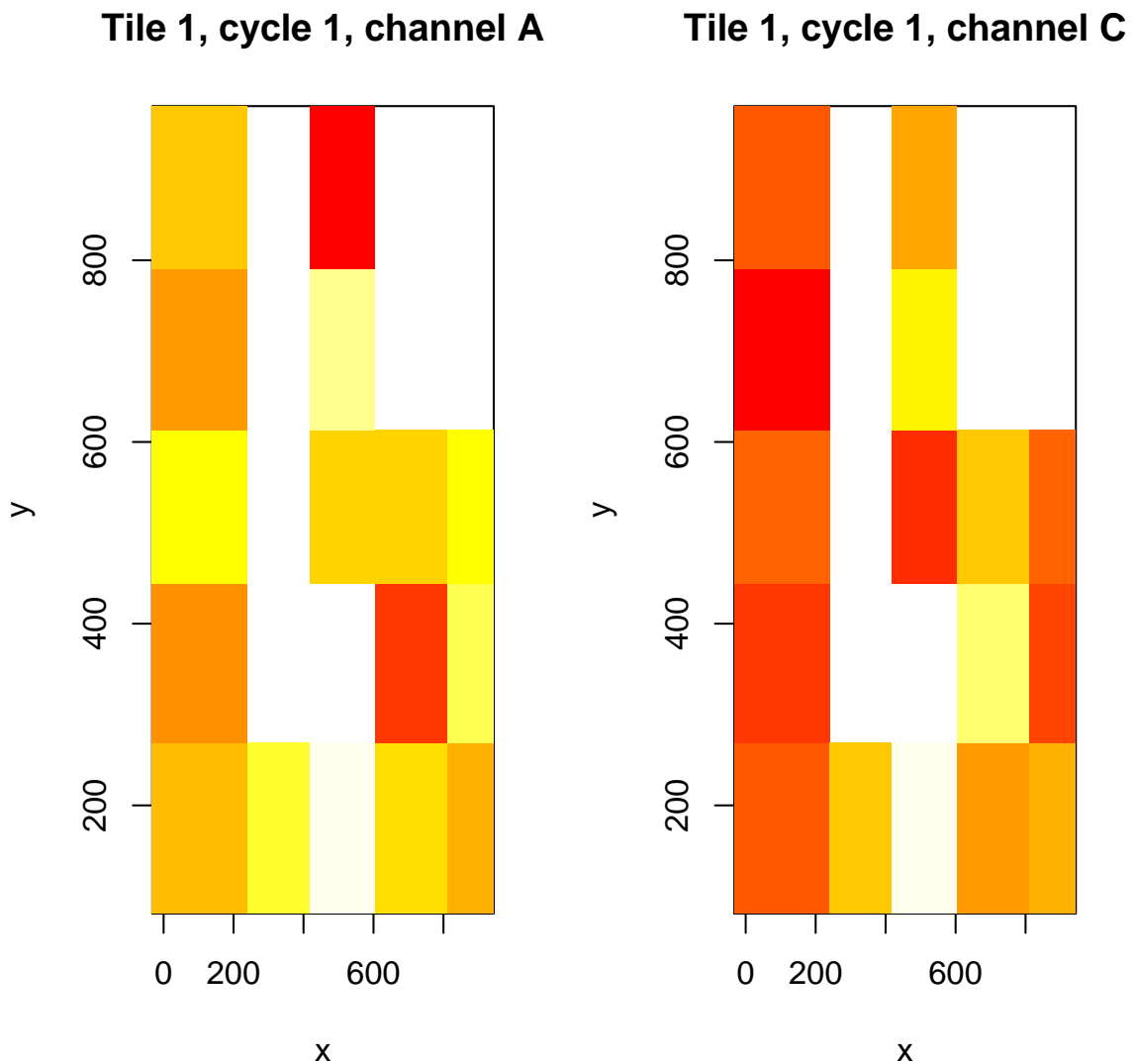
and look at global statistics of a base-calling:

```
> par(mfrow=c(2,2),cex=.8,mar=c(4, 4, 2, 1)+.1)
> BatchAnalysis(run=rolenv,seq=res2$sread,scores=res2$entropy,what="length",main="Length d
> BatchAnalysis(run=rolenv,seq=res$sread,scores=res$entropy,what="ratio",main="Complementa
> BatchAnalysis(run=rolenv,seq=res$sread,scores=res$entropy,what="iupac",main="IUPAC codes
> BatchAnalysis(run=rolenv,seq=res2$sread,scores=res2$entropy,what="information",main="Bas
```



and visualize the positional bias over a tile by

```
> par(mfrow=c(1,2))
> TileImage(int=int,cycle=1,tile=readInfo(int)$tile[1],ncell=5,channel='A')
> TileImage(int=int,cycle=1,tile=readInfo(int)$tile[1],ncell=5,channel='C' )
```



## 9 Session Information

The version number of R and packages loaded for generating the vignette were:

```
> toLatex(sessionInfo())
```

- R version 2.15.0 (2012-03-30), x86\_64-unknown-linux-gnu
- Locale: LC\_CTYPE=en\_US.UTF-8, LC\_NUMERIC=C, LC\_TIME=en\_US.UTF-8, LC\_COLLATE=C, LC\_MONETARY=en\_US.UTF-8, LC\_MESSAGES=en\_US.UTF-8, LC\_PAPER=C, LC\_NAME=C, LC\_ADDRESS=C, LC\_TELEPHONE=C, LC\_MEASUREMENT=en\_US.UTF-8, LC\_IDENTIFICATION=C
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils

- Other packages: BiocGenerics 0.2.0, Biostrings 2.24.0, GenomicRanges 1.8.0, IRanges 1.14.0, RColorBrewer 1.0-5, Rolexa 1.12.0, Rsamtools 1.8.0, ShortRead 1.14.0, lattice 0.20-6, latticeExtra 0.6-19, mclust 3.4.11
- Loaded via a namespace (and not attached): Biobase 2.16.0, bitops 1.0-4.1, grid 2.15.0, hwriter 1.3, stats4 2.15.0, tools 2.15.0, zlibbioc 1.2.0

## References

- [1] Jacques Rougemont, Arnaud Amzallag, Christian Iseli, Laurent Farinelli, Ioannis Xenarios, and Felix Naef. Probabilistic base calling of Solexa sequencing data. *BMC Bioinformatics*, **9**:431, 2008.