User’s manual for GCAT’s *abcd* package

- 1 Overview
- 2 Installation of package *abcd*
- 3 Skew
  - 3.1 Skew for single nucleotides
  - 3.2 Skew for n-plets
- 4 Boxplot for n-plet skews
- 5 Standard deviation of skews
- 6 Normalized skews
- 7 n-plet skews of sequences with different lengths
- 8 n-plet skews in partitions of sequences
- 9 Random sequences

Markus Gumbel, Ali Karpuzoglu

Mannheim University of Applied Sciences

This tutorial is for *abcd* version 0.9-x.

1 Overview

This document introduces the *abcd* (**A**nalysis of **B**ase **C**omposition of long **D**NA sequences) package which belongs to a family of R packages known as GCAT R packages (Genetic Code Analysis Toolkit). The R package *abcd* can be used for the analysis of n-plets and the skew. More information about GCAT can be found at http://www.gcat.bio (http://www.gcat.bio).

2 Installation of package *abcd*

**Note:** We are working on a simpler installation procedure - this is work in progress.

The R package *abcd* requires

- **R** (in version 3.1 or higher)
- packages *rJava* and *seqinr*
- Java virtual machine (Java 8 or higher)

We recommend Rstudio as a workbench.

*abcd* internally calls Java routines; before the package itself is going to be installed, you need to install *rJava*. Type in the R console:

```
install.packages("rJava")
```

Next install *seqinr*, the package for accessing genetic sequences and more.

```
install.packages("seqinr")
```
The steps above only need to be done once. Anytime you update the *abcd* package these steps are not required anymore.

Finally, we need to prepare the JVM anytime we start a R session, e.g. after launching Rstudio. **Important**: The JVM must be of the same architecture than R itself. If R is 32 bit (x86) you also need a x86 JVM. On the other hand, if R is 64 bit (x64) the JVM has to be a x64 version, too.

Run the following command to indicate R where the JVM is or set `JAVA_HOME` as an environment variable in your operating system. This differs from computer to computer and operating system to operating system. Under windows it looks something like this where `<NUMBER>` indicates a specific JVM version:

```
Sys.setenv(JAVA_HOME="C:/Program Files/Java/jdk<NUMBER>")
```

Under Mac it is typically:

```
Sys.setenv(JAVA_HOME=""/Library/Java/JavaVirtualMachines/jdk<NUMBER>")
```

Now the *abcd* package itself can be installed or updated. You might skip this step if *abcd* is up-to-date. We assume that *abcd_0.9-0.tar.gz* is the package - certainly there will be other versions in the future. Type:

```
install.packages("abcd_0.9-0.tar.gz",
    INSTALL_opts = "--no-multiarch", # Ignore architecture
    repos = NULL) # allows us to use a file; no download
```

We are almost done. Just type

```
library("abcd")
```

to load the library. You should now be ready to go!

Note that you can browse the manual where each function is described in detail. In RStudio: go to Packages, click on the library *abcd*.

### 3 Skew

#### 3.1 Skew for single nucleotides

Let us use the following short sequence as an example:

```
seq = "AGTACGCAGTGCCATA"
```

The base distribution over a sequence can be calculated with:
It returns a vector with labels for each base.

The skews $S_{A\sim T}(1, 1)$ and $S_{C\sim G}(1, 1)$ is calculated with:

```
skew(seq)
```

Internally, the base distribution is calculated first.

### 3.2 Skew for n-plets

Next we want to calculate the skew for n-plet. The following example gives us the skews for a n-plet size of $n = 3$.

```
nplet.skew(seq, 3)
```

\[
\begin{array}{c}
\text{at} \\
1 & 2 & 3 \\
0.5000000 & 0.0000000 & 0.3333333
\end{array}
\]

\[
\begin{array}{c}
\text{cg} \\
1 & 2 & 3 \\
1 & 0 & -1
\end{array}
\]

An n-plet of size 3 has 3 positions ($k = 1, 2, 3$) where the skew is calculated. This is returned by this function. The names of the skew (header) are the positions.

There is also an alternative implementation of the skew function that is based on Java. It computes the skew much faster than the R version. Before Java can be used we need to prepare R to work with Java. The abcd package will automatically do this for you at start time.

The Java based skew is calculated with:

```
nplet.skewj(seq, 3)
```
Clearly, we obtain the same result.

The `nplet.skew` function can be generalized by passing a vector of n-plet sizes to the `nplet.skews` function (note “skews” not “skew” in the function name).

```r
skews123 = nplet.skews(seq, c(1,2,3))
```

As we can see, the return value (data structure) is more complex: it consists of two nested lists:

- A-T skew and then the skew for all n-plet sizes (here 1 to 3).
- C-G skew and then again the skew for all n-plet sizes.

If we want to access the C-G skews for n-plet size 2 we simply can query:
By default, this function uses the Java-based version. The parameter `skewF` can be set to change the function: `skewF = nplet.skew` or `skewF = nplet.skewj`.

### 4 Boxplot for n-plet skews

Let us load a slightly larger DNA sequence of about 2800 bases in the fasta format that is stored in `data/demo.fasta`. The function `seq.fastaread` reads a fasta file and returns the first entry.

```r
library("seqinr") # This library provides a read fasta function.
filename = "/data/demo.fasta"
demoseq = sequence.fastaread(filename)
demoseq

a t c g
960 780 532 567
```

In the following sections we use the n-plet sizes `nsizes = c(2, 3, 10, 30, 50)` by default.

```r
nsizes = c(2, 3, 10, 30, 50) # default n-plet sizes
demoskews = nplet.skews(demoseq, nsizes)
demoskews
```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.06682028</td>
<td>0.13990826</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.19710669</td>
<td>0.05031447</td>
<td>0.07078040</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.08771930</td>
<td>0.29069767</td>
<td>0.12568306</td>
<td>0.08379888</td>
<td>0.11111111</td>
<td>0.21505376</td>
</tr>
<tr>
<td>-0.02325581</td>
<td>0.06358382</td>
<td>0.03333333</td>
<td>0.03703704</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.11864407</td>
<td>0.28125000</td>
<td>0.07142857</td>
<td>0.09803922</td>
<td>-0.01886792</td>
<td>0.12903226</td>
</tr>
<tr>
<td>0.14814815</td>
<td>0.10000000</td>
<td>0.00000000</td>
<td>0.16000000</td>
<td>0.01639344</td>
<td>0.14285714</td>
</tr>
<tr>
<td>0.24590164</td>
<td>0.16216216</td>
<td>0.19298246</td>
<td>0.26666667</td>
<td>-0.15625000</td>
<td>0.05263158</td>
</tr>
<tr>
<td>0.27586207</td>
<td>-0.05882353</td>
<td>0.13725490</td>
<td>0.46153846</td>
<td>0.06060606</td>
<td>-0.03703704</td>
</tr>
<tr>
<td>0.15384615</td>
<td>0.25000000</td>
<td>-0.03703704</td>
<td>0.03571429</td>
<td>-0.16129032</td>
<td>0.04545455</td>
</tr>
<tr>
<td>0.17647059</td>
<td>0.35135135</td>
<td>0.51351351</td>
<td>0.03030303</td>
<td>0.09090909</td>
<td>0.21951220</td>
</tr>
<tr>
<td>-0.09523810</td>
<td>0.27272727</td>
<td>0.13513514</td>
<td>0.05555556</td>
<td>-0.07692308</td>
<td>0.21212121</td>
</tr>
<tr>
<td>-0.02857143</td>
<td>-0.05882353</td>
<td>0.39393939</td>
<td>0.21052632</td>
<td>-0.11764706</td>
<td>0.03030303</td>
</tr>
<tr>
<td>0.14814815</td>
<td>0.10000000</td>
<td>0.00000000</td>
<td>0.16000000</td>
<td>0.01639344</td>
<td>0.14285714</td>
</tr>
<tr>
<td>-0.09523810</td>
<td>0.27272727</td>
<td>0.13513514</td>
<td>0.05555556</td>
<td>-0.07692308</td>
<td>0.21212121</td>
</tr>
<tr>
<td>-0.02857143</td>
<td>-0.05882353</td>
<td>0.39393939</td>
<td>0.21052632</td>
<td>-0.11764706</td>
<td>0.03030303</td>
</tr>
<tr>
<td>0.10000000</td>
<td>0.00000000</td>
<td>0.00000000</td>
<td>0.12500000</td>
<td>0.08571429</td>
<td>-0.12820513</td>
</tr>
<tr>
<td>0.00000000</td>
<td>0.29729730</td>
<td>0.12500000</td>
<td>-0.02857143</td>
<td>-0.12820513</td>
<td>0.13333333</td>
</tr>
<tr>
<td>0.10526316</td>
<td>0.39393939</td>
<td>-0.05263158</td>
<td>0.25000000</td>
<td>0.08571429</td>
<td>0.20000000</td>
</tr>
<tr>
<td>-0.12500000</td>
<td>-0.25000000</td>
<td>0.06250000</td>
<td>-0.03225806</td>
<td>0.21052632</td>
<td>0.24324324</td>
</tr>
<tr>
<td>0.13513514</td>
<td>0.11111111</td>
<td>-0.03448276</td>
<td>0.13333333</td>
<td>0.12500000</td>
<td>0.25000000</td>
</tr>
<tr>
<td>0.00000000</td>
<td>0.25000000</td>
<td>-0.15151515</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\text{cg}$

$\text{cg}$`^2`

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

$\text{cg}^2$
There is a convenience command `nplet.skewplot` which plots the skews for different n-plet sizes.

```
nplet.skewplot(demoskews,
    seqId = "demo sequence",
    yMax = .5)        # Range of y-axis
```

5 Standard deviation of skews

The following R command calculates the standard deviation of the skews in different positions for given n-plet sizes.

```r
skews = nplet.skews(demoseq, nSizes = nsizes)
unlist(lapply(skews$at, sd)) # std. deviation of AT skews
```

<table>
<thead>
<tr>
<th>n-plet size</th>
<th>AT std. deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.05168101</td>
</tr>
<tr>
<td>3</td>
<td>0.07950382</td>
</tr>
<tr>
<td>10</td>
<td>0.09169087</td>
</tr>
<tr>
<td>30</td>
<td>0.13528175</td>
</tr>
<tr>
<td>50</td>
<td>0.16120547</td>
</tr>
</tbody>
</table>

```r
unlist(lapply(skews$cg, sd)) # std. deviation of CG skews
```

<table>
<thead>
<tr>
<th>n-plet size</th>
<th>CG std. deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.02980235</td>
</tr>
<tr>
<td>3</td>
<td>0.07382358</td>
</tr>
<tr>
<td>10</td>
<td>0.12330676</td>
</tr>
<tr>
<td>30</td>
<td>0.17623767</td>
</tr>
<tr>
<td>50</td>
<td>0.21427798</td>
</tr>
</tbody>
</table>

The list of standard deviations can be plotted with the function `nplet.sdskewplot`. Note that this function calculates the standard deviation, i.e. a parameter is the skews:

```r
nplet.sdskewplot(skews, nSizes = nsizes, seqId = "demo sequence", yMax = .2)
```
6 Normalized skews

Calculate the sample-size normalized skews with the function `nplet.sizenormskews` and plot them with `nplet.skewplot`. Note that we use the optional parameter `ylabPrefix` in `nplet.skewplot` to modify the y-axis label. A sufficient cover factor is $cf > 50$, optimal is $c > 100$. (It takes more than a couple of minutes if $cf = 25$, so we set $cf = 10$.)

```r
ndemoskews = nplet.sizenormskews(demoseq, nsizes, cf = 10)
nplet.skewplot(ndemoskews, seqId = "demo sequence", yMax = 0.5, ylabPrefix = "size norm. ")
```
Also plot the standard deviations; again with the `ylabPrefix` parameter:

```r
nplet.sdskewplot(ndemoskews, nSizes = nsizes, seqId = "demo sequence",
                  yMax = .25, ylabPrefix = "size norm.")
```
7 n-plet skews of sequences with different lengths

We’d like to analyze now how the n-plet skews differ if we have sequences of different lengths. Let us take the demo sequence again and calculate the skews for n-plets of size 20 for length 100 and 1000. The demo sequence contains about 2800 bases:

```r
sdemoskews = nplet.nskewpersize(demoseq, sizes = c(500, 1000), n = 20)
sdemoskews
```
Now we calculate the standard deviation of these values with the convenience function `nplet.stddevpersize` and plot them with `nplet.stddevplotpersize`. We add “n = 20” to the plot’s title to indicate that the data is for a n-plet size of 20:

```
sdsdemoskews = nplet.stddevpersize(sdemoskews)
sdsdemoskews
```
As we can see, the A-T skews of the short sequence ($|X|=100$) is not a number.

8 n-plet skews in partitions of sequences

n-plet skews may not only be calculated for the entire sequence but also for a partition of a sequence, i.e. a subsequence. First, we introduce the `subseqpart` function which simply divides a sequence into $m$ partitions of the same size $N/m$ where $N$ is the length of the sequence. Example:
The function `nplet.partskew` calculates the skews for all partitions. Let us use the longer sequence `demoseq`:

```r
part = nplet.partskew(demoseq,
    nSize = 3,  # n-plet size
    pCount = 4) # Number of partitions
part
```
The result is (again) a complex data structure:

- The first entry encapsulated the A-T skews
- then there is a list of partitions (here 1 to 4)
- followed by the number of skews for each position (here 1,2,3) in the n-plet of size 3
- The second entry is the same for the C-G skew.

Finally, we can plot the results as a boxplot with `nplet.partskewplot`:
Note that the x-axis shows now the partition (not the n-plet size). Once again with two times more partitions:

```r
nplet.partskewplot(demoseq,
    nSize = 4,  # n-plet size
    pCount = 2*4,  # Number of partitions
    seqId = "demo sequence",
    yMax = .5)
```
sequence.condbaseprob calculates the conditional probabilities of bases for a DNA or RNA sequence. The following example does this for the demo sequence. As output we calculate the values in percentage and round them to one digit.

```r
cp = sequence.condbaseprob(demoseq)
round(cp * 100, digits = 1)
```

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>36.7</td>
<td>24.6</td>
<td>16.6</td>
<td>22.1</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>26.0</td>
<td>29.6</td>
<td>17.7</td>
<td>26.7</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>37.6</td>
<td>34.0</td>
<td>20.3</td>
<td>8.1</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>36.2</td>
<td>23.3</td>
<td>22.2</td>
<td>18.3</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

Note that there is a special base \(\text{N}\) that represents an unknown base.

The function `sequence.rndbycondprob` generates a random DNA sequence of a given length using conditional probabilities:

```r
seqrnd = sequence.rndbycondprob(50, cp)
seqrnd
```
[1] "GTGCATCACAGAAATCTCTCTCGGAATCAGCTTAATTCCCTTAGCC"

sequence.basedist(seqrnd)

<table>
<thead>
<tr>
<th>a</th>
<th>t</th>
<th>c</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>15</td>
<td>15</td>
<td>8</td>
</tr>
</tbody>
</table>

sequence.basedist(seqrnd) / sum(sequence.basedist(seqrnd))

<table>
<thead>
<tr>
<th>a</th>
<th>t</th>
<th>c</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.24</td>
<td>0.30</td>
<td>0.30</td>
<td>0.16</td>
</tr>
</tbody>
</table>

or for a longer random sequence:

seqrnd2 = sequence.rndbycondprob(100000, cp)
sequence.basedist(seqrnd2)

<table>
<thead>
<tr>
<th>a</th>
<th>t</th>
<th>c</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>33601</td>
<td>27513</td>
<td>18677</td>
<td>20209</td>
</tr>
</tbody>
</table>

sequence.basedist(seqrnd2) / sum(sequence.basedist(seqrnd2))

<table>
<thead>
<tr>
<th>a</th>
<th>t</th>
<th>c</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.33601</td>
<td>0.27513</td>
<td>0.18677</td>
<td>0.20209</td>
</tr>
</tbody>
</table>