deltaGseg

Diana HP Low[1], Efthymios Motakis[2]

Institute of Molecular and Cell Biology[1], Bioinformatics Institute[2]
Agency for Science, Technology and Research (A*STAR), Singapore
dlow@imcb.a-star.edu.sg[1]
emotakis@hotmail.com[2]

October 24, 2023

Contents

1 Introduction

deltaGseg aims to identify subpopulations within time series data, and here we have applied it to molecular dynamic (MD) simulation free binding energies to provide a descriptive free energy landscape where different macrostates can coexist [1,2]. The theoretical and methodological considerations are analytically discussed in the main paper [1]. Here, we will demonstrate how to perform macrostate identification analysis with *deltaGseg*.

2 Installing and loading the deltaGseg package

We recommend that users install the package via Bioconductor, since this will automatically detect and install all required dependencies. The Bioconductor installation procedure is described at http://www.bioconductor.org/docs/install/.To install *deltaGseg*, launch a new R session, and in a command terminal either type or copy/paste:

```
if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
BiocManager::install("deltaGseg")
```

3 Using deltaGseg

3.1 Loading the deltaGseg

To load the deltaGseg package, simply:-

```
library(deltaGseg)
```

This package also includes all the variables generated in this vignette, and they can be loaded by:

```
data(deltaGseg)
```

3.2 Step 1: Data loading, visualization and testing

deltaGseg takes in 2-column, space-separated files of timepoints (column 1) and time-series measurement (column 2). For our example, the binding energy of a system (e.g. between a protein and its ligand) can be extracted from a MD trajectory as a time series measurement. Details on how to extract binding energies can be found at http://ambermd.org/tutorials/advanced/tutorial4/py_script. Typically, replicates are carried out for a system by running multiple simulations with slightly different starting conditions (eg. initial configuration or seed number). We will aim to identify different macrostates from this data using time series analysis, by estimating the significance of existence of multiple subpopulations.

We load three replicated series, i.e. the tab delimited 2-column data files `D_GBTOT1, D_GBTOT2, D_GBTOT3` where the first column contains the time data t = 1, 2, ..., 5000 and the second column the free binding energy values B_t. The variable `path` defines the directory where the files reside, and in this case, we will look in the directory where *deltaGseg* has been installed. The variable `files` defines the filenames to be read. If left blank, it will read the entire directory's content.

```
dir<-system.file("extdata",package="deltaGseg")
traj1<-parseTraj(path=dir, files=c("D_GBTOT1","D_GBTOT2","D_GBTOT3"))
```

Typing the name of the object (i.e. `traj1`) will give a brief description of its contents.

```
traj1

## class: Trajectories
##  Source: /tmp/RtmpnYKHMK/Rinst22a7919a720f7/deltaGseg/extdata/
##  Names: D_GBTOT1 D_GBTOT2 D_GBTOT3
##  Trajectories: 3
##  Points per trajectory: 5000 5000 5000
##  adf p-values: 0.01 0.01 0.07872805
##
## Available slots: path filenames trajlist avd
```

Here it shows that there were 3 files loaded, each with 5000 time points. Next, we visualize the series and test the weak-stationarity assumption. Formally, the null hypothesis "H_0: the series is not weakly-stationary" is rejected if adf p-value ≤ 0.05 [3]. We notice that the adf p-value of the D_GBTOT3 series is higher than 0.05 indicating that we do not have enough evidence to reject H_0 of non-stationarity. An initial plot (Figure 1) can be made to view the time series. `D_GBTOT3` exhibits a significant shift around the midpoint (approx t=12775) resulting in the high adf p-value.

```
plot(traj1,name='all')
```

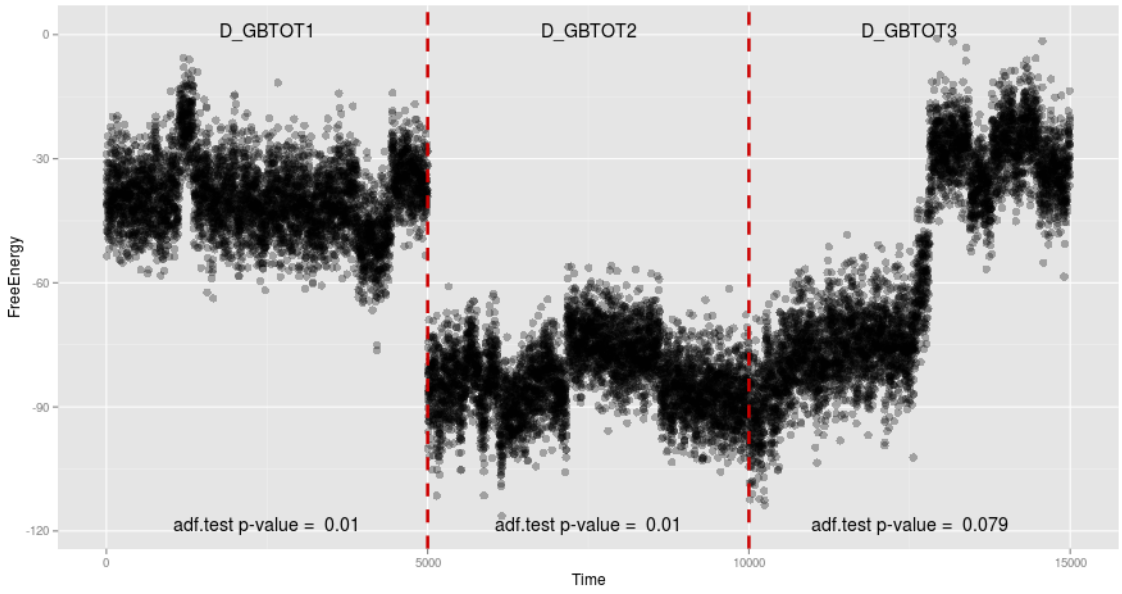

Figure 1: The complete `traj1` series

The `transformSeries` function identifies which series is "inappropriate" (i.e. `D_GBTOT3`) and splits it into two weakly-stationary subseries (the parameter specifying the number of splits is decided after data visualization), which are segmented (and subsequently modeled) independently. We can see here that `D_GBTOT3` has now been split (`D_GBTOT3_1`, `D_GBTOT3_2`) and the resulting subseries has adf pvalues ≤ 0.05. Plotting the transformed series will now indicate the new breakpoint.

```
traj1.tr <- transformSeries(object = traj1, method = "splitting", breakpoints = 1)
traj1.tr

## class: TransTrajectories
##  Method: splitting
##  Names: D_GBTOT1 D_GBTOT2 D_GBTOT3_1 D_GBTOT3_2
##  Trajectories: 4
##  Points per trajectory: 5000 5000 2775 2225
##  adf p-values: 0.01 0.01 0.01 0.01
##  Segment splits per trajectory: 1
##
## class: Trajectories
##  Source: /tmp/RtmpnYKHMK/Rinst22a7919a720f7/deltaGseg/extdata/
##  Names: D_GBTOT1 D_GBTOT2 D_GBTOT3
```

```
##  Trajectories: 3
##  Points per trajectory: 5000 5000 5000
##  adf p-values: 0.01 0.01 0.07872805
##
## Available slots: tmethod breakpoints tavd ttrajlist tfilenames difftraj path filenames trajlist av
```

```
plot(traj1.tr)
```

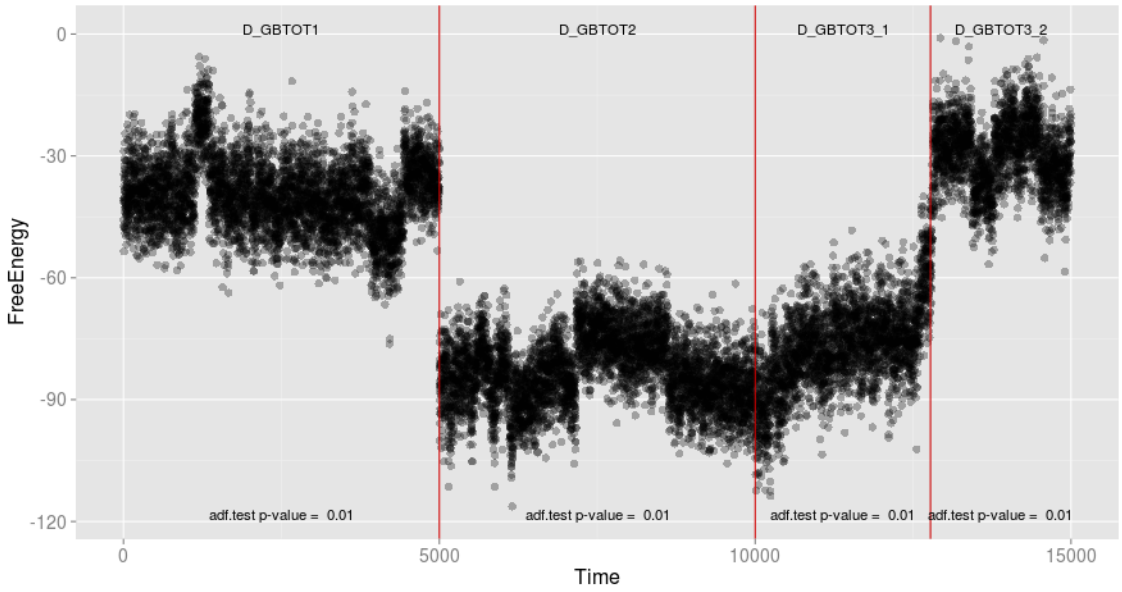

Figure 2: The transformed series, with newly defined breakpoint.

Another way to derive weakly stationary (sub)series is by data differentiation B_t-B_{t-1}. This technique is suitable for series with a significant trend-like behavior, causing the weak stationarity assumption to fail. Differentiation removes the trend and returns a transformed series that can be safely segmented. We show such an example in the Appendix.

3.2.1 Other possible reasons for splitting a time-series

Long series of e.g. more than 50000 time points could also be split into smaller subseries to take advantage of R's computation time. `splitTraj` identifies appropriate breakpoints:

```
all_breakpoints <- splitTraj(traj1, segsplits = c(5, 5, 5))
all_breakpoints
```

```
## $D_GBTOT1
## [1] 1124 1353 1573 3912 4429 5000
##
## $D_GBTOT2
## [1]  584 1095 1570 2166 3618 5000
##
## $D_GBTOT3
## [1]  496 2603 2775 2812 4516 5000
```

Here, `splitTraj` has identified 5 possible breakpoints (number depends on user input). The original series can now be plot with all determined breakpoints.

```
plot(traj1,breakpoints=all_breakpoints)
```

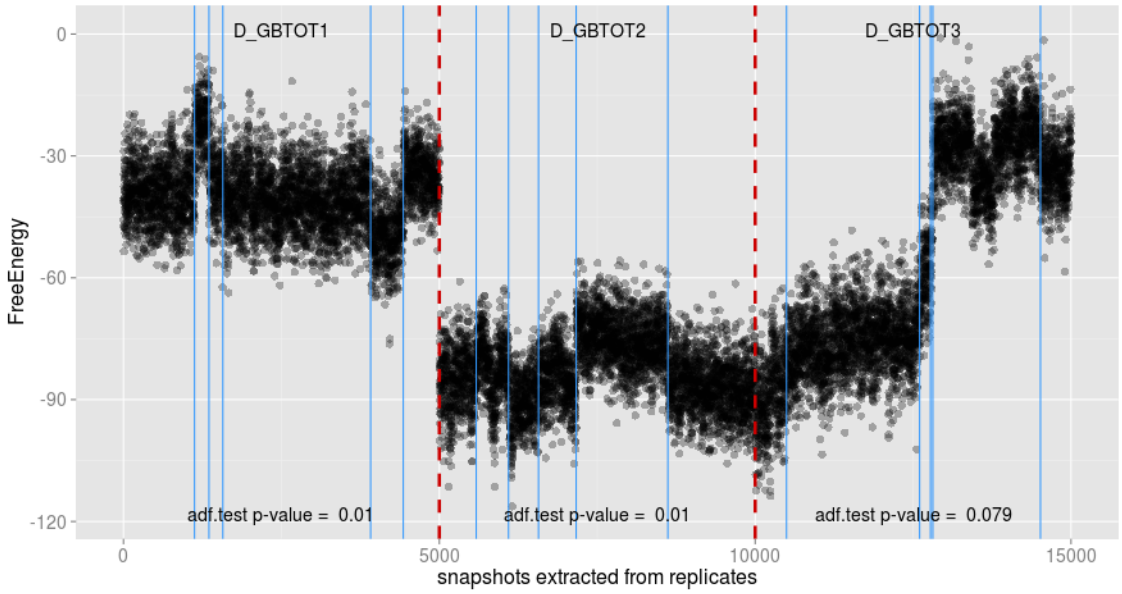

Figure 3: Trajectories with breakpoints

Often, there is little use in taking all defined breakpoints, so here we elect to pick 3 breakpoints (for each series). `chooseBreaks` selects evenly-spaced breakpoints (this is optional, and users can choose the split points manually, as a list of lists).

```
mybreaks <- chooseBreaks(all_breakpoints, numbreaks = 3)
mybreaks

## $D_GBTOT1
## [1] 1124 1573 4429
##
## $D_GBTOT2
## [1]  584 1570 3618
##
## $D_GBTOT3
## [1]  496 2775 4516
```

`transformSeries` (together with `method="override_splitting"`) generates the new subseries.

```
traj1.sp.tr <- transformSeries(object = traj1, method = "override_splitting", breakpoints = mybreaks)
traj1.sp.tr

## class: TransTrajectories
##  Method: override_splitting
##  Names: D_GBTOT1_1 D_GBTOT1_2 D_GBTOT1_3 D_GBTOT1_4 D_GBTOT2_1 D_GBTOT2_2 D_GBTOT2_3 D_GBTOT2_4 D_
```

```
##  Trajectories: 12
##  Points per trajectory: 1124 449 2856 571 584 986 2048 1382 496 2279 1741 484
##  adf p-values: 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01
##  Segment splits per trajectory: 3 3 3
##
## class: Trajectories
##  Source: /tmp/RtmpnYKHMK/Rinst22a7919a720f7/deltaGseg/extdata/
##  Names: D_GBTOT1 D_GBTOT2 D_GBTOT3
##  Trajectories: 3
##  Points per trajectory: 5000 5000 5000
##  adf p-values: 0.01 0.01 0.07872805
##
## Available slots: tmethod breakpoints tavd ttrajlist tfilenames difftraj path filenames trajlist av
```

Another useful application of `method="override_splitting"` is the manual split of the series. For example, if the user is not satisfied with the automatic split that `method="splitting"` offers, he can override it by setting the `breakpoints` parameter the desired splits. In the above example, the transformation of `traj1` with 9 chosen breakpoints resulted in `traj1.sp.tr`, that has 12 trajectories (subseries). Note that calling the transformed object will always report the original object as well, for tracking purposes.

3.3 Step 2: Segmentation and denoising

In this step, each series (or subseries if `transformSeries` was performed) is segmented by the Segment Neighborhoods (SegNeigh) method [4]. Each segment $q = 1, ..., Q$ (Q is the total number of segments estimated in all series) is subsequently smoothed by wavelet decomposition and shrinkage [5]. The segments will be used for the subpopulation estimation (sets of clustered segments define a subpopulation). The denoising removes the data autocorrelation and generates an "identity" vector for each segment that is used in clustering.

```
traj1.denoise<-denoiseSegments(object=traj1.tr,seg_method="SegNeigh",maxQ=15,fn=1,factor=0.8,thresh_l
```

3.4 Step 3: Subpopulation estimation

The data from each denoised segment are summarized (in a vector of quantiles) and used in hierarchical clustering that assesses segments similarity (Euclidean distances) and groups similar segments together in a hierarchical tree fashion. The user inspects the tree structure, the time series plots and the significance of the clusters (if *pvclust* algorithm [6] is used) to decide how many and which subpopulations to derive. Here we illustrate subpopulation identification using the *pvclust* algorithm. The alternative, simple hierarchical clustering option will also discussed later. To use *pvclust* clustering, we first estimate the *multi-scale bootstrap* p-values:

```
pvals<-clusterPV(object=traj1.denoise,bootstrap=500)
```

We will now use the p-values in the hierarchical clustering to aid our grouping of segments.

```
traj1.ss <- clusterSegments(object = traj1.denoise, intervention = "pvclust", pv = pvals)
## Segment grouping. Click on the root of the groups you want clustered. Please
## ensure that ALL segments are grouped (boxed). Otherwise, function will not
## exit. To exit, click Esc (Windows/Linux) or Ctrl-click (Mac)
```

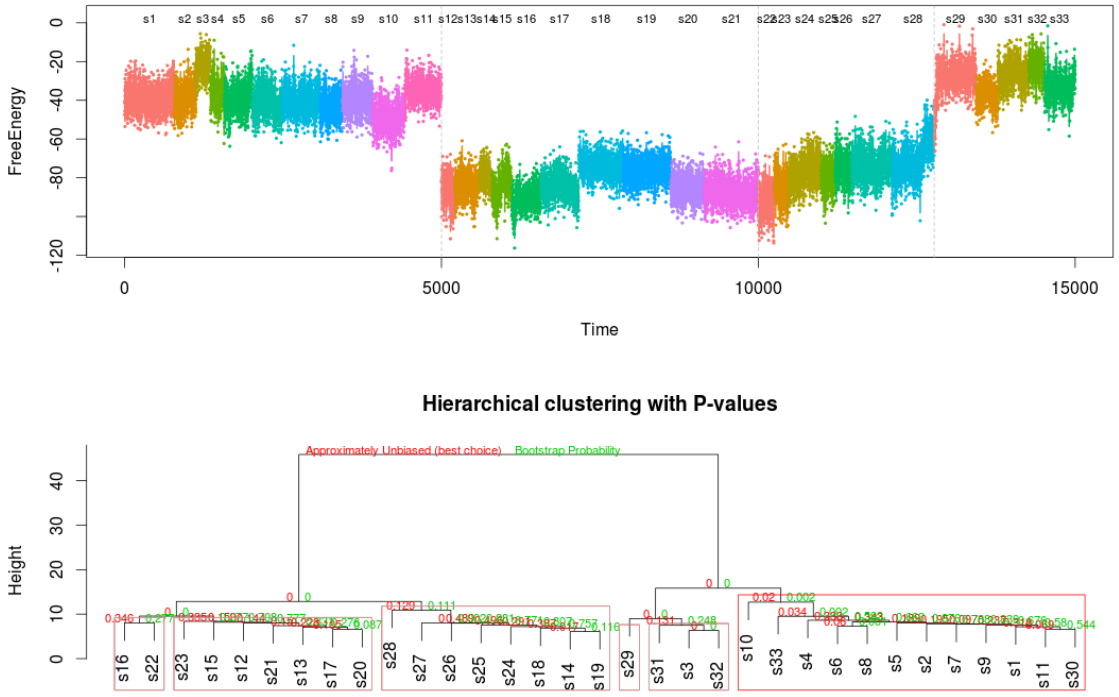

Figure 4: `clusterSegments` run with *pvclust*. The top figure shows the initial segments (product of `denoiseSegment`) and the bottom figure shows the resulting clustering using *pvclust*. Here, the segments have been grouped into 6 subpopulations (boxed).

The top plot shows the segmented data for each (sub)series. The vertical lines separate each (sub)series, within which each segment has a unique color. The bottom plot shows the result of the *pvclust* algorithm, consisting of a simple hierarchical tree (Euclidean distances and average linkage; obtained by the *hclust* function) and the two kinds of estimated p-values at each node. The ones in red (Approximately Unbiased; AU) are more reliable than the ones in green (Bootstrap Probability; BP)[6].

To describe how *pvclust* inference is made, assume two disjoint segment sets $q_i \cap q_j = 0$ for two vectors $i, j \in [1, ..., Q]$. The null hypothesis of "H_0: q_i and q_j are clustered together" is rejected if the p-value of the respective node is lower than a threshold. Here, we avoid explicitly specifying the threshold (e.g. 0.05) because (1) it depends on the user's assumptions on how many clusters he wants to identify (a priori assumption aided by data visualization) and (2) the p-values are meant to be provide evidence of possible subpopulations and only help the user decide which subpopulations can be meaningful.

Practically, the user wishes to identify a small number of ergodic subpopulation, so the focus is on the p-values of the top nodes. In this example, based on the estimated AU p-values we estimate 6 subpopulations which are subsequently plotted in Figure 6.

```
plot(traj1.ss)
```

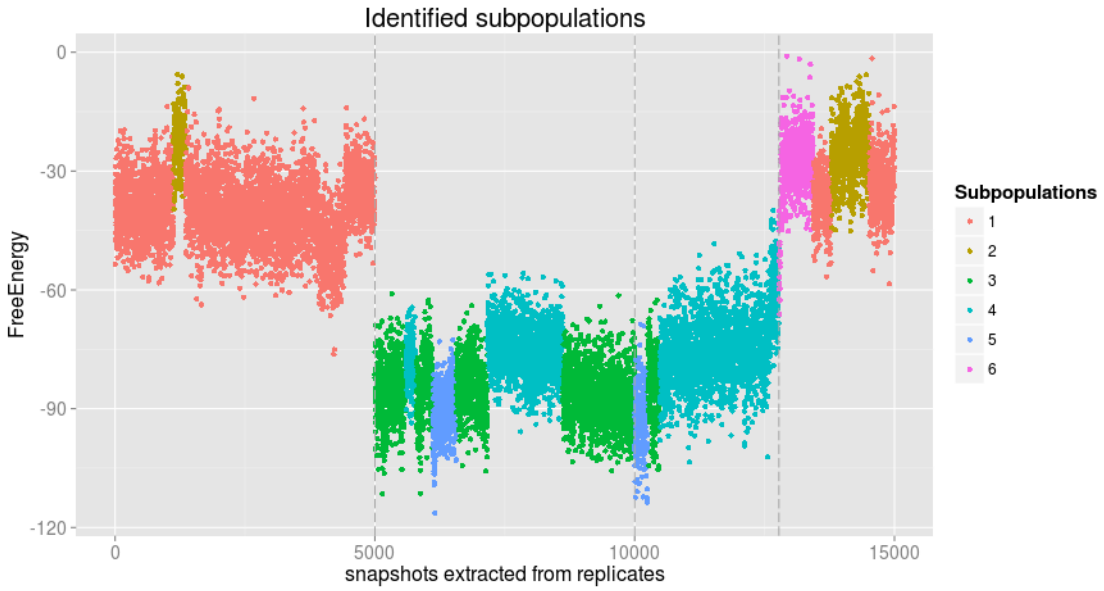

Figure 5: Result of the `clusterSegments grouping`, with segments coloured according to their respective subpopulations.

Instead of *pvclust* the user can simply use hierarchical clustering to derive the subpopulations of interest. This is done in two alternative ways: (1) by simply specifying the number of subpopulations to be identified (the algorithm requests a value and splits automatically) and (2) by interactive, point and click user intervention on the plot according to which the clusters are separated based on height (see ?hclust). These are more trivial but ultimately effective subpopulation identification options.

3.4.1 Obtaining the subpopulation intervals

The resulted subpopulations and the respective snapshot intervals they occur can be obtained by the following convenience function:

```
getIntervals(traj1.ss)

## $subpopulation
##  [1] 1 2 1 3 4 3 5 3 4 3 5 3 4 6 1 2 1
##
## $interval.start
##  [1]     1  1125  1354  5001  5597  5799  6113  6571  7167  8619 10001 10252
## [13] 10480 12776 13435 13780 14517
```

3.5 Step 4: Diagnostic plots

This post-processing step should always performed to assess the validity of the results and especially of the wavelet modeling. We assume that the residuals of the model are approximately normally distributed N(0,σ^2) or, at least, symmetrically distributed around 0. Also, the residuals should not

exhibit significant autocorrelation coefficients. We run a series of statistical tests whose results we visualize in terms of autocorrelation plots, histograms and p-values for the significance of the null hypothesis "H_0: the residuals are N(0,σ^2) distributed".

```
diagnosticPlots(object = traj1.ss, norm.test = "KS", single.series = TRUE)
```

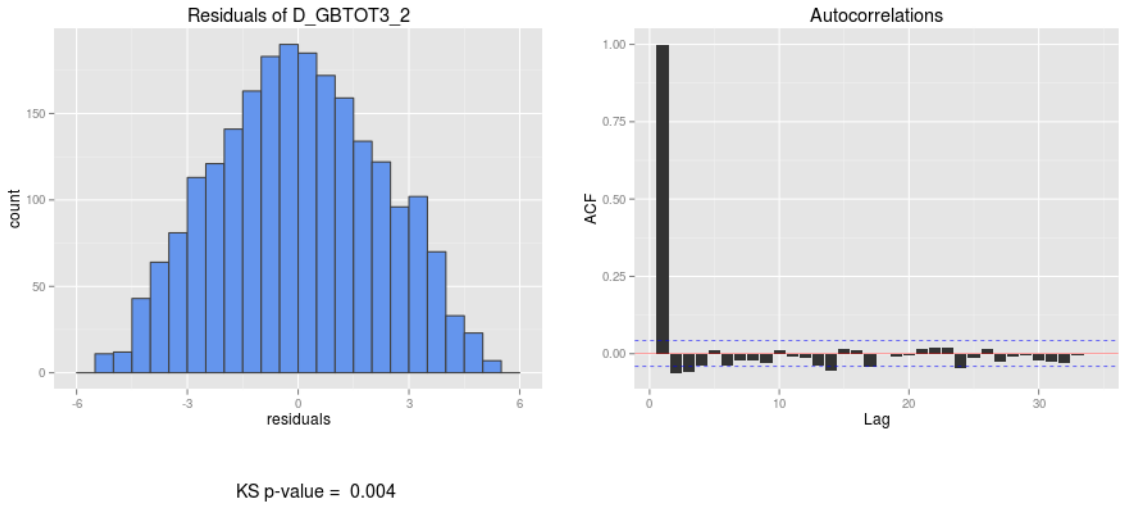

Figure 6: diagnosticPlots

4 Appendix

In this section we will describe the case of series differentiation, which is performed by function `transformedSeries`. As mentioned earlier, differentiation is suitable for series exhibiting a trend-like behavior causing the weak stationarity assumption to fail. Below, we simulate a set of data that could be appropriately analyzed with this method.

```
x<-getTraj(traj1.tr)[['D_GBTOT3_1']]
y<-x[,2]-35; y1<-y[1:2000]; y2<-y[2001:length(y)]+17
ss<-c(seq(50,length(y2),100),length(y2))
for(i in 1:(length(ss)-1)) y2[ss[i]:ss[(i+1)]]<-y2[ss[i]:ss[(i+1)]]+i^1.85
y<-cbind(x[,1],c(y1,y2))
simtraj<-parseTraj(files=list(y),fromfile=FALSE)
simtraj

## class: Trajectories
##  Source: /tmp/RtmpnYKHMK/Rbuild22a797ea30ce5/deltaGseg/vignettes/
##  Names: 1
##  Trajectories: 1
##  Points per trajectory: 2775
##  adf p-values: 0.8402461
##
## Available slots: path filenames trajlist avd
```

```
plot(simtraj)
```

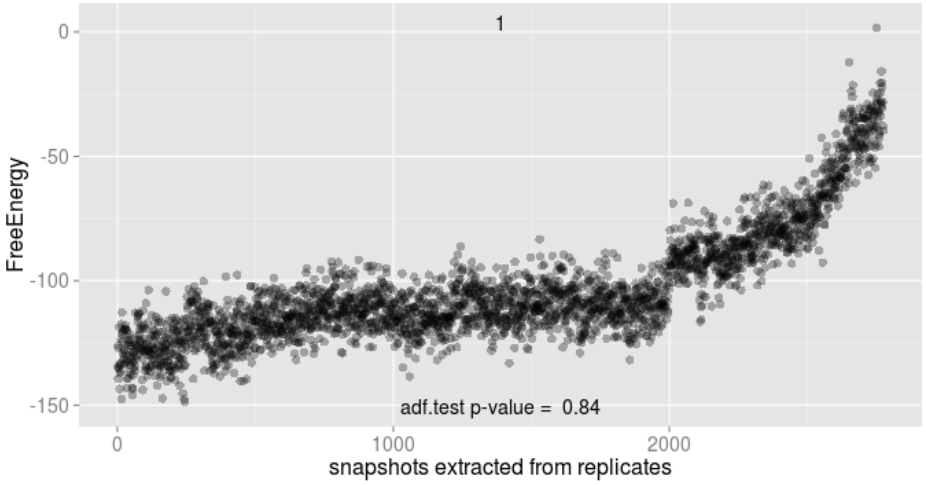

Figure 7: The simulated series from `D_GBTOT3_1`

The plot above presents our simulated series, obtained by shifting downwards the `D_GBTOT3_1` data by 35 and altering, after t = 2000 the increase rate as $B_t = k \times B_t^{1.85}$, where for the first 50 observations we use initial value k = 1 and increase k = k + 1 for subsequent disjoint segments of 100 observations each. Additionally, at t = 2000 we shifted the series upwards by 17. In this way the simulated data (`simtraj`)resemble a real, more variable than `D_GBTOT3_1`.

In the first step, we will use `transformSeries` with `method="splitting"` to divide the series into two subseries, *s1* and *s2*.

```
simtraj.tr <- transformSeries(object = simtraj, method = "splitting", breakpoints = 1)
simtraj.tr

## class: TransTrajectories
##  Method: splitting
##  Names: 1_1 1_2
##  Trajectories: 2
##  Points per trajectory: 2000 775
##  adf p-values: 0.01 0.07739078
##  Segment splits per trajectory: 1
##
## class: Trajectories
##  Source: /tmp/RtmpnYKHMK/Rbuild22a797ea30ce5/deltaGseg/vignettes/
##  Names: 1
##  Trajectories: 1
##  Points per trajectory: 2775
##  adf p-values: 0.8402461
##
## Available slots: tmethod breakpoints tavd ttrajlist tfilenames difftraj path filenames trajlist av
```

```
plot(simtraj.tr)
```

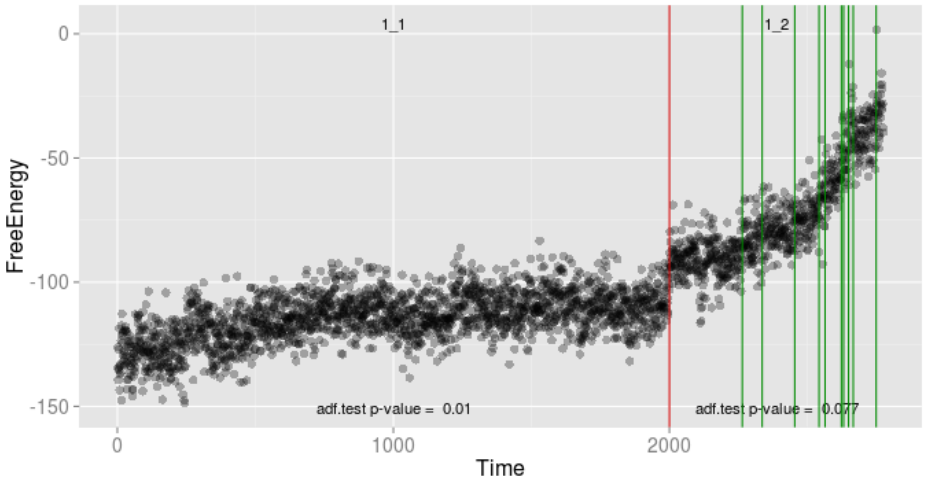

Figure 8: The transformed series

The BinSeg algorithm successfully identified the splitting point at t = 2000 (red line). We run the adf test which for *s2* estimates adf p-value = 0.077, implying that we do not have enough evidence to reject the null hypothesis of non-stationarity. If we split *s2* again, we take very small segments (green lines), which by default our methodology does not accept (the minimum length of a segment is arbitrarily set to 200). If we consider a subset of these segments to split *s2*, the adf test will fail for certain series (not shown). Accepting these segmentation results is not methodologically correct [4].

For such cases we suggest running `transformSeries` with `method="differentiation"`.

```
simtraj.tr2 <- transformSeries(object = simtraj.tr, method = "differentiation")
simtraj.tr2
```

```
plotDiff(simtraj.tr2,name="1_2")
```

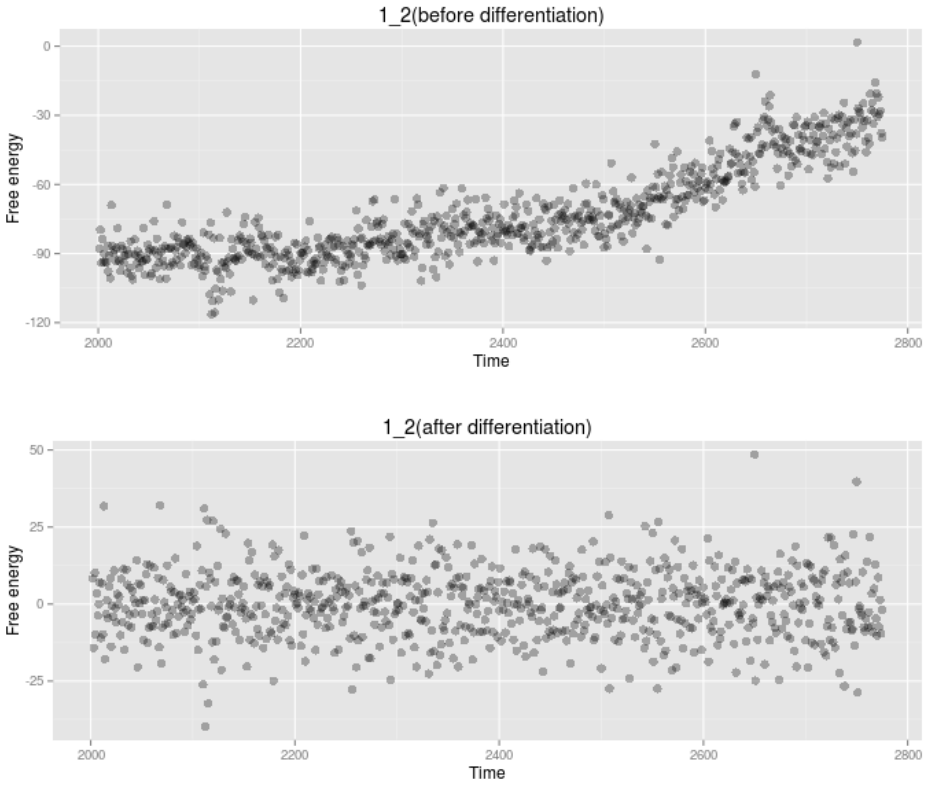

Figure 9: Second segment before and after differentiation

The plot shows the differentiated data of *s2*, $D = B_t$ - B_{t-1}, and the segments identified. Evidently, no significantly different segments are present (the one at the end is much smaller than 200 observations). In practice (long series like the one we studied above), we might conclude that the whole *s2* belongs to a transition-like period. Notice that the differentiated data fluctuates around 0 and thus they are not comparable to the original free binding energies. For this reason D is used for segmentation only. Once the segments have been estimated, they are applied to the original, B_t, data that are subsequently denoised and clustered.

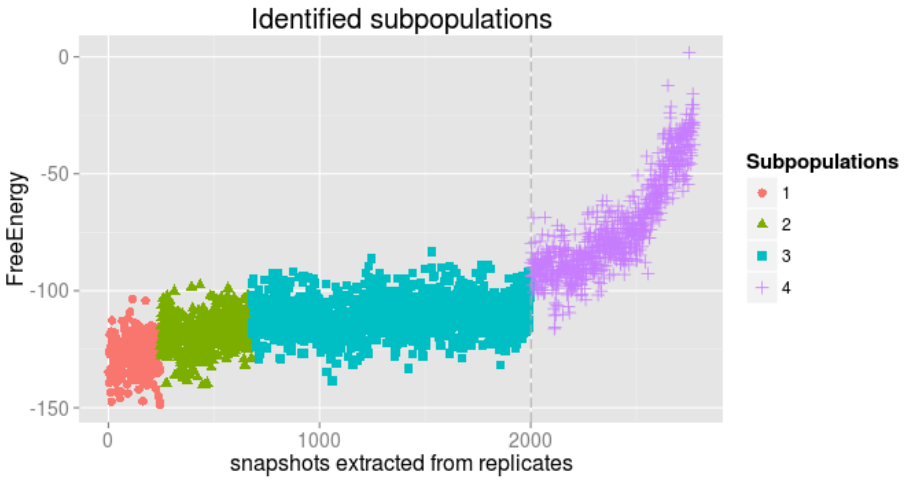

Figure 10: Final segmentation results of the simulated series

References

[1] Zhou W., Motakis E., Fuentes G. and Verma S.C. (2012).*Macrostate Identification from Biomolecular Simulations through Time Series Analysis.*.J. Chem. Inf. Model., 2012, 52 (9), pp 2319-2324

[2] Low, D.H.P, Motakis E. deltaGseg: identifying distinct subpopulations though multiscale time series analysis, to be submitted in Bioinformatics

[3] Dickey, D.A. and W.A. Fuller (1979) *Distribution of the Estimators for Autoregressive Time Series with a Unit Root,* Journal of the American Statistical Association, 74, p. 427-431.

[4] Auger, I. E.; Lawrence, C. E. *Algorithms for the optimal identification of segment neighborhoods.* Bull. Math. Biol. 1989, 51(1), 39-54.

[5] Nason, G.P. (2008) *Wavelet methods in Statistics with R.* Springer, New York.

[6] Shimodaira, H. (2004) *Approximately unbiased tests of regions using multistep-multiscale bootstrap resampling,*Annals of Statistics, 32, 2616-2641.

[7] D'Agostino R.B., Pearson E.S. (1973) *Tests for Departure from Normality*, Biometrika 60, 613-22.


```
sessionInfo()
```

```
## R Under development (unstable) (2023-10-22 r85388)
## Platform: x86_64-pc-linux-gnu
## Running under: Ubuntu 22.04.3 LTS
##
## Matrix products: default
## BLAS:   /home/biocbuild/bbs-3.19-bioc/R/lib/libRblas.so
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: America/New_York
## tzcode source: system (glibc)
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] deltaGseg_1.43.0  scales_1.2.1      reshape_0.8.9     fBasics_4031.95
##  [5] pvclust_2.2-0     tseries_0.10-54   wavethresh_4.7.2  MASS_7.3-60.1
##  [9] changepoint_2.2.4 zoo_1.8-12        ggplot2_3.4.4
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.4        dplyr_1.1.3         compiler_4.4.0
```

```
##  [4] highr_0.10          Rcpp_1.0.11         tidyselect_1.2.0
##  [7] timeSeries_4031.107 lattice_0.22-5      plyr_1.8.9
## [10] R6_2.5.1            generics_0.1.3      curl_5.1.0
## [13] knitr_1.44          tibble_3.2.1        spatial_7.3-17
## [16] munsell_0.5.0       timeDate_4022.108   pillar_1.9.0
## [19] rlang_1.1.1         quantmod_0.4.25     utf8_1.2.4
## [22] xfun_0.40           quadprog_1.5-8      cli_3.6.1
## [25] formatR_1.14        withr_2.5.1         magrittr_2.0.3
## [28] xts_0.13.1          lifecycle_1.0.3     vctrs_0.6.4
## [31] evaluate_0.22       glue_1.6.2          fansi_1.0.5
## [34] colorspace_2.1-0    TTR_0.24.3          tools_4.4.0
## [37] pkgconfig_2.0.3
```