README for the R code to the paper "The dynamic linear epigenome"

Tobias Ahsendorf[1,2]

[1]DKFZ, D-69120 Heidelberg, Germany
[2]Department of Systems Biology, Harvard Medical School, Boston, MA, USA

August 26, 2016

We assume to have input files in a bedgraph format style, i.e., like chromosome, start position, end position (and value), so it might look like (the data are assumed to be tab-separated)

```
chr1 1 1000 4
chr1 500 2000 13
...
```

or like

```
chr1 1 200
chr1 50 250
...,
```

where in the latter case we assume the value to be 1 (might make sense for aligned reads, when each line represents a read). In case we start with BAM files (or similar), conversion tools like `bamtobed` from bedtools might be used in first place. When we want to assign enrichment values to each bin on a genome-wide scale (or a subset of chromosomes) one has to call the R script `Enrichments_per_bin_processing.R`, for instance, via

```
Rscript Enrichments_per_bin_processing.R /home/input
    .bg /home/output.RData BSgenome.Hsapiens.UCSC.
    hg19 4 1000 chr1 chr2 chr3
```

which outputs an RData file at the desired position, see the R script file for further information. Though the content of that R script could be formulated as an R function, we decided not to do so as according to our own experience it is most useful when calling that R script file on a cluster by submitting several jobs each dealing with one or a few files at a time.
After that has been done for each mark at each time point, we have to combine all processed data for each mark. For that purpose we use a function from the collection `functions.R`.

```
#load all functions from functions.R
source('/home/functions.R')

#paths to the individual RData files for the epigenetic marks
#at a particular time point (the output of
#Enrichments_per_bin_processing.R), e.g.,
paths = c("/home/test.RData","/home/test2.RData")
#marks at the respective positions of paths, e.g.,
marks =  c("H3K4me3","H3K27me3")

#to obtain the pasted data frame at that time point we call
X = paste_processed_data(paths,marks)
#or with a different quantile cutoff
X = paste_processed_data(paths,marks,quantile_cutoff=0.05)
```

Now assume we have a data frame to two different time points, respectively, where the rows correspond to the same bins. Here we can use the example data which correspond to 10000 randomly selected loci of Koike et al. data at 1000 bp resolution at time point 0 hours and time point 4 hours. These can be loaded with something like

```
load("/home/Koike_0hours_sample.RData")
start_data = dataset
load("/home/Koike_4hours_sample.RData")
target_data=dataset
```

If we want to predict the changes at each bin for all possible marks from start_data to target_data by using 10-fold CV and a homogeneous system of linear ODEs of first order, we call

```
X = CV_ODE_prediction_change(start_data,target_data)
```

For further information regarding that function, particularly what parameters we can modify (e.g., k-fold CV, not necessarily homogeneous system of linear ODEs of first order), we refer to `functions.R`. In particular, since for large datasets it might be challenging to calculate everything for all marks at once, we can select individual marks by

```
#select one or a few marks
marks_sel = c("H3K4me3","H3K9ac")
X = CV_ODE_prediction_change(start_data,target_data,marks=marks_sel)
```

In addition, when assuming a system of linear ODEs of first order with constant coefficients, $\frac{d}{dt}x = A \cdot x(+b)$, we might want to get out A (and b).
So if we start with the transition from start_data to target_data, at time points t_1 and t_2, respectively, where $t_1 < t_2$, then we first calculate the rows of the matrix $B = \exp(A \cdot (t_2 - t_1))$ in the homogeneous case. In the general case we want to get out the rows $B = \exp\left(\begin{bmatrix} A & b \\ 0 & 0 \end{bmatrix} \cdot (t_2 - t_1)\right)$, from which can reconstruct A and b. For the former we first we call

```
r = ode_exponential_matrix_rows(start_data,target_data)
```

and for the ladder we call

```
r = ode_exponential_matrix_rows(start_data,target_data,ODE_mode = "with intercept")
```

and get out a list of the rows of B and the names of that list correspond to the marks that are present in both start_data and target_data (and the constant value in the not necessarily homogeneous case). Just as above, the computation for large datasets might be challenging for all marks at once, here we can select individual marks by

```
r = ode_exponential_matrix_rows(start_data,target_data,marks=marks_sel)
```

in the homogeneous case and with the obvious variation in the general case and get out a list of rows of B, but just for those marks that have been called. In the latter option, it is necessary to paste together the output of all individual calculations into a list, such that we do have a list covering all marks that are present in both start_data and target_data (and the constant value in the not necessarily homogeneous case). Now, since in the matrix logarithm complex values might arise and all matrix logarithm functions in R (at least those we are aware of) run into trouble then, we call Matlab using the function `system()`. We suggest to test that first, for instance, by entering the toy code

```
system('matlab -nodisplay -r "a=1; display(a); exit"')
```

into the R console.
If that does work, we can call

```
#to ensure that we have the rows for all marks
#in the variable r we call again
r = ode_exponential_matrix_rows(start_data,target_data)
#set time points (in minutes)
t_1 = 0
t_2 = 240
#path for a short term intermediate csv file storage
#directory has to be existent
temporal_path = "/home/intermediate.csv"
output = ode_logarithm_matrix(r,t_1,t_2,temporal_path)
```

and get out a list with the matrix A (and the vector b).