MoPS - Model-based Periodicity Screening

Philipp Eser, Achim Tresch

May 3, 2016

Contents

1 Overview

The detection and characterization of periodic changes of measurements in time is important in many fields of science. This package employs a model-based screening algorithm for the identification of periodic fluctuations in time series datasets. It uses a likelihood ratio statistic to decide whether a time series displays periodic fluctuations or not. Additionally MoPS infers the best fitting periodic time course for each time series and thus allows the characterization of various parameters like period length and phase. The algorithm was originally designed to identify and characterize periodic expression profiles in Microarray time series measurements. Hence, the described case study (section 6) is based on a time series of genome-wide expression measurements during the S.cerevisiae cell cycle [**?**].

2 Installation

To run the MoPS package, the software R ($>$ 3.0.0) has to be installed. For installation of R, refer to `http://www.r-project.org`. To see how to install add-on R-packages, start R and type in `help(INSTALL)`. To install the MoPS package, use the function `biocLite`. Once the package is installed, you can load it by

```
> library(MoPS)
```

3 MoPS Quickstart

The application of MoPS essentially requires only two commands, `fit.periodic` and `predictTimecourses`: Given a (noisy, possibly periodic) time series, say

```
> y = 2*sin(seq(0,6*pi,length.out=50))
> y.noise = y+rnorm(50,sd=1)
```

we can estimate a best periodic fit to the time series y.noise by

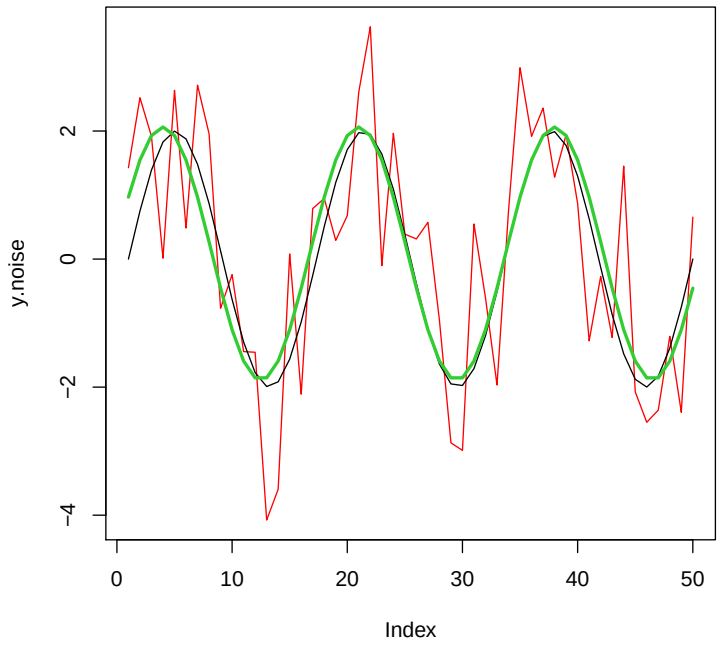
```
> res = fit.periodic(y.noise)
```

the fitted time course can be accessed by

```
> predicted = predictTimecourses(res)
```

We visualize the results in a plot:

```
> plot(y.noise,type="l",col="red")
> points(y,type="l")
> points(predicted,type="l",lwd=2.5,col="limegreen")
```



4 Basic Usage

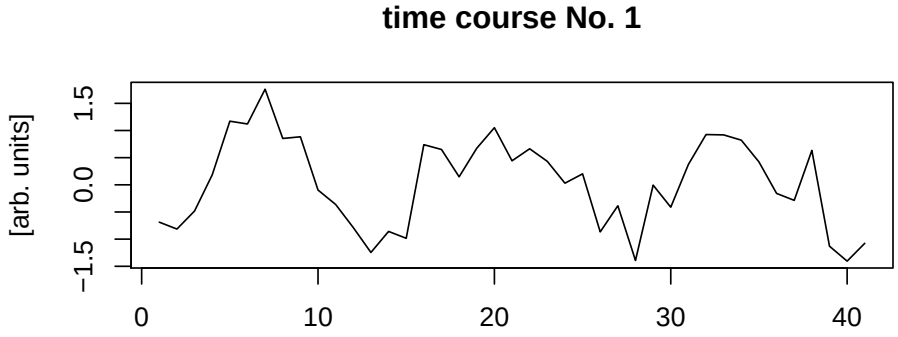
This section illustrates the basic usage of MoPS to fit periodic time courses to time series data. We demonstrate the use of MoPS at the example of 10 time series consisting of 41 consecutive measurements each (see `help(fit.periodic)` for more information on how these data were obtained).

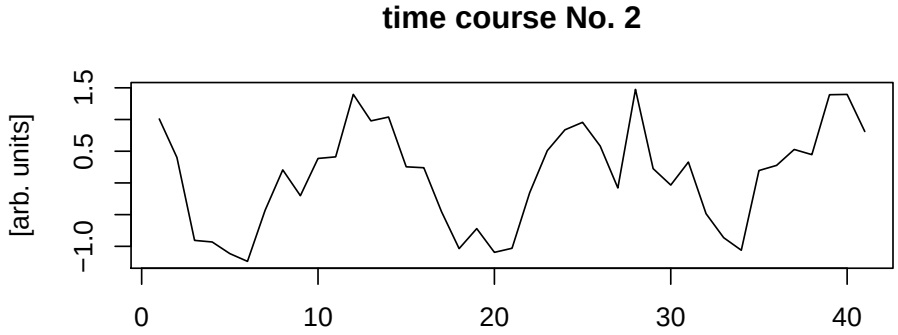
```
> data(basic)
> dim(basic)
```

```
[1] 10 41
```

For illustration, let us plot time course 1 and 10.

```
> par(mfrow=c(2,1))
> plot(basic[1,],type="l",main="time course No. 1",xlab="",ylab="[arb. units]")
> plot(basic[10,],type="l",main="time course No. 2",xlab="",ylab="[arb. units]")
```





The function `fit.periodic` performs the fitting of periodic functions to each time series in the data matrix. The results can be converted to a data.frame that yields the best fitting periodic parameters for each time series.

```
> res = fit.periodic(basic)
> result.as.dataframe(res)
```

```
       ID score phi lambda sigma  mean amplitude
1   ID_1  1.12   7     13     0  0.08      0.95
2   ID_2  1.48   8     14     0 -0.03      0.92
3   ID_3  1.76  11     14     0  0.05      0.97
4   ID_4  1.39  13     14     0  0.06      1.07
5   ID_5  1.06   1     14     0  0.01      0.92
6   ID_6  1.18   3     14     0 -0.03      0.93
7   ID_7  1.72   6     14     0 -0.02      1.01
8   ID_8  1.63   9     13     0  0.10      1.13
9   ID_9  1.26  10     14     0  0.03      1.12
10 ID_10  1.26  12     14     0  0.12      0.96
```

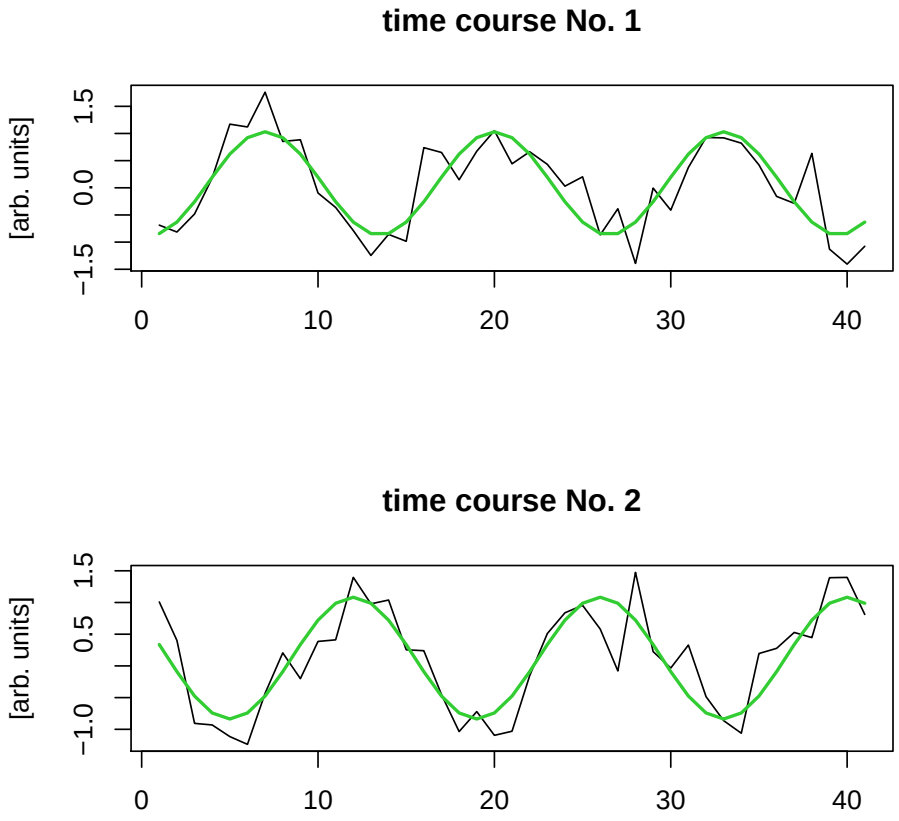
For each time series (black line), using the fitted parameters, we can now calculate and plot the predicted time course (green). By default, `predictTimecourses` returns a matrix of the same dimension as the input matrix basic that was used for learning the parameters.

```
> fitted.mat = predictTimecourses(res)
> dim(fitted.mat)
```

```
[1] 10 41
```

```
> par(mfrow=c(2,1))
> plot(basic[1,],type="l",main="time course No. 1",xlab="",ylab="[arb. units]")
> points(fitted.mat[1,],type="l",col="limegreen",lwd=2)
```

```
> plot(basic[10,],type="l",main="time course No. 2",xlab="",ylab="[arb. units]")
> points(fitted.mat[10,],type="l",col="limegreen",lwd=2)
```



5 Specification

5.1 Formal description

MoPS determines an optimal periodic fit to a given time series. It provides the parameters of the optimal fit, plus a periodicity score that can be used to rank several time courses according to their periodicity. The central idea is to compare the best least squares fit among an exhaustive set of periodic test functions with the best fit of a corresponding set of non-periodic test functions. The log likelihood ratio of these two fits is our periodicity score (see Figure 1).

MoPS uses periodic curves that are parametrized by a 6-tuple (lambda, sigma, phi, psi, mean, amplitude) of parameters. The period length lambda defines the time duration between two consecutive maxima. The parameter phi determines the phase, e.g. the time point at which the time course assumes its maximal values. The parameter sigma determines the magnitude of attenuation of the signal along the complete time course. This can arise if there is increasing synchrony loss with time, leading to a blurred profile with decreasing amplitude. Finally, the parameter psi allows the generation of more flexible periodic test functions by deforming the sine wave curves at variable sampling points (see Figure 2).

Note that we use an equidistant grid of phases which is independent of the period length (lambda). This makes sure that for each lambda the same phase increment is used and thus does not favor test functions with small lambda.

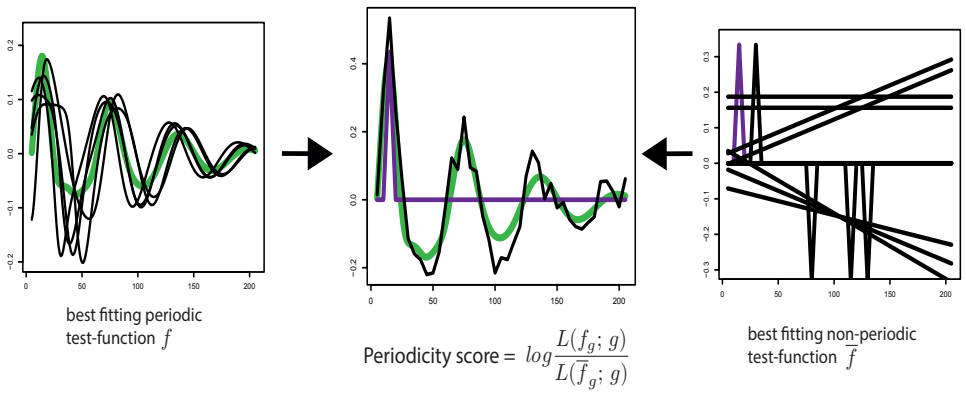


Figure 1: Scoring of time series according to periodicity. Each input time course is fitted to an exhaustive set of periodic (left) and non-periodic (right) test functions. The log likelihood ratio of the best fitting test functions is the periodicity score (taken from [**?**]).

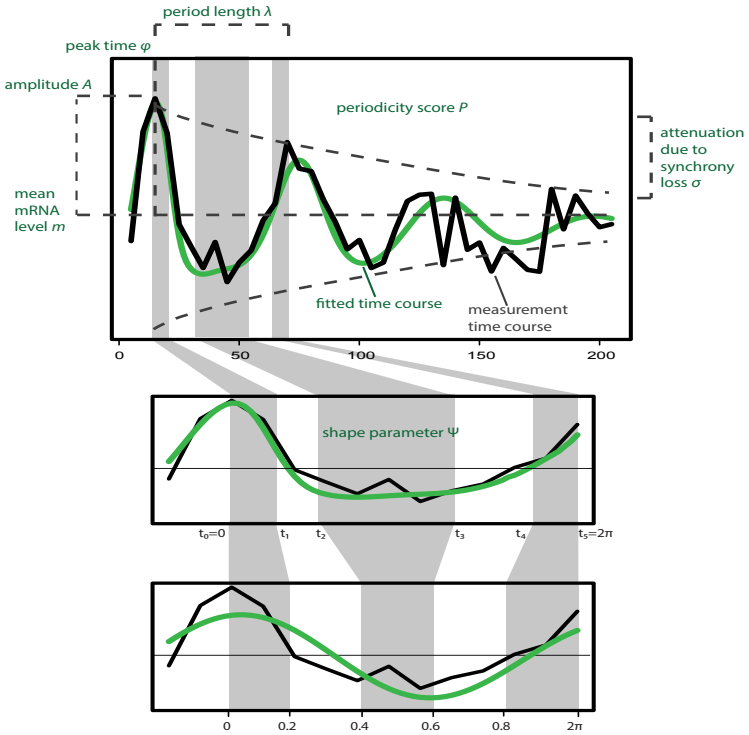


Figure 2: Fitting of a characteristic periodic profile to time course measurements. See text for a description of the parameters (taken from [**?**]).

5.2 Implementation

The main function of the MoPS package is `fit.periodic`, a method that determines an optimal fit of a periodic curve to a given time course. The function takes as input a matrix containing time series measurements and optionally a set of parameters that are used to model the underlying periodicity.

`fit.periodic` automatically creates periodic test functions based on all possible combinations of the input parameters. It further creates non-periodic test functions that represent a diverse set of constant time courses. It then compares each input time course to all periodic and non-periodic test functions with linear regression based on the `lm` function to estimate the likelihood of periodic behaviour. Instead of using plain least squares regression, the user can specify a vector of regression weights for each measurement. This is particularly useful when the magnitude of the measurement error is not constant for all measurements, as, e.g., for gene expression measurements.

The screening result is returned as a list object (for a detailed description, see `help(fit.periodic)`). The first slot contains the screening result for each time series (here only the first is shown):

```
> res$fitted[[1]]
```

```
$ID
[1] "ID_1"

$score
[1] 1.122195

$minLossPeriodic
[1] 7.680137

$minLossNonPeriodic
[1] 23.59022

$phi
[1] 7

$psi
NULL

$lambda
[1] 13

$sigma
[1] 0

$a.coef
[1] 0.9513084

$b.coef
[1] 0.08037059
```

The remaining slots contain information about the screened parameter ranges:

```
> res[2:6]
```

```
$time
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41

$cols.mat
[1] 41

$phi
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41

$lambda
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41

$sigma
[1] 0
```

This result list serves as input to the function `predictTimecourses` and thus contains several parameters that need not necessarily be accessed by the user. The function `result.as.dataframe` can convert the result object to a data.frame which includes only the time series specific best fitting periodic parameters:

```
> head(result.as.dataframe(res))
```

```
    ID score phi lambda sigma  mean amplitude
1 ID_1  1.12   7     13     0  0.08      0.95
2 ID_2  1.48   8     14     0 -0.03      0.92
3 ID_3  1.76  11     14     0  0.05      0.97
4 ID_4  1.39  13     14     0  0.06      1.07
5 ID_5  1.06   1     14     0  0.01      0.92
6 ID_6  1.18   3     14     0 -0.03      0.93
```

The function `predictTimecourses` constructs the best fitting periodic time course for each entry. By default, the resulting matrix of predicted time courses has the same dimensions as the original input matrix. However, the number of columns changes, if the user chooses a phase grid that has more resolution (less spacing) than the measurements points.

6 Case Study - Yeast Cell Cycle

The following section illustrates the workflow for the identification and characterization of cell cycle regulated genes in expression time series data. We start by loading a data matrix that contains normalized mRNA expression time series of 200 genes (rows) (subset of data published in [**?**], ArrayExpress E-MTAB-1908). The values in each column correspond to the measurements taken at different time points after release from cell cycle arrest. It entails 41 samples that are separated by 5 minutes, covering a total of 205 minutes.

```
> data(ccycle)
> timepoints = seq(5,205,5)
```

6.1 Detection of periodically expressed genes

The first step in the MoPS screening procedure is the construction of the exhaustive set of periodic test functions. This is done by specifying a cartesian grid of parameters that define the test functions. I.e., for each parameter, we specify a set of values that will be combined with all other parameter choices. We aim to identify genes that are expressed in a cell cycle regulated manner and thus show periodic expression profiles. We will use the input parameters phi, lambda and sigma to model this periodic expression. Accordingly, lambda corresponds to the cell cycle length, phi is the time at which maximal expression is achieved and sigma corresponds to the magnitude of synchrony loss.
While phi and lambda are basic parameters of periodic functions, sigma is harder to conceive: it determines the magnitude of attenuation along the complete time course. Here, this arises from cell cycle length differences of individual cells in the population which in turn leads to increasing loss of synchrony after release from cell cycle arrest.
By choosing meaningful parameter ranges for lambda, phi and sigma we screen our data for cell cycle regulated genes:

```
> phi = seq(5,80,5)
> lambda = seq(40,80,5)
> sigma = seq(4,8,1)
> res = fit.periodic(ccycle,timepoints=timepoints,phi=phi,lambda=lambda,sigma=sigma)
> res.df = result.as.dataframe(res)
> head(res.df)
```

```
       ID score phi lambda sigma    mean amplitude
1 YPR179C -0.03  15     40     8   90.94     28.97
2 YLR212C  1.28  15     60     5 1288.87    578.22
3 YAR003W  1.14  15     65     7  539.42    182.85
4 YGL148W -0.59  30     40     8 5834.10    530.00
5 YKL068W -0.08  45     55     4 1202.26     49.12
6 YDR019C -0.56   5     50     5 1307.84    449.30
```

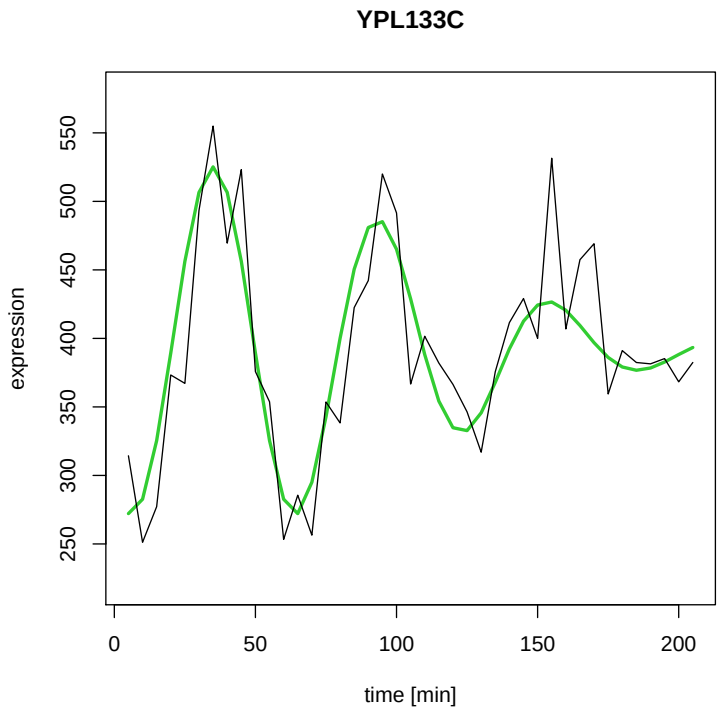
`fit.periodic` returns the best-fitting periodic parameter set and periodicity score for each gene. Genes with a positive score achieve better fits with periodic than non-periodic test functions. However, this does not imply statistical significance, it merely provides a ranking of genes. The fitting results are used to predict the periodic time courses for all genes:

```
> time.courses = predictTimecourses(res)
> dim(time.courses)
```

```
[1] 200  41
```

This results in a numeric matrix containing the fitted characteristic time courses, which can be plotted for individual genes.

```
> id = "YPL133C"
> t = time.courses[id,]
> plot(timepoints,t,type="l",col="limegreen",lwd=2.5,main=id,
+                  xlab="time [min]",ylab="expression",ylim=c(220,580))
> points(timepoints,ccycle[id,],type="l",col="black")
```



Using the gene-specific best fitting parameters of genes with score > 0, we can derive the median cell cycle length and variation in the population.

```
> lambda.global = median(res.df$lambda[res.df$score > 0])
> lambda.global
```

```
[1] 60
```

```
> sigma.global = median(res.df$sigma[res.df$score > 0])
> sigma.global
```

```
[1] 7
```

6.2 Characterisation of cell cycle regulated genes

The following analyses aim to characterize the periodic time courses of cell cycle regulated genes. MoPS provides an optional parameter psi that defines the flexibility to shape the periodic test functions. Using this parameter, the number of created test functions increases greatly and leads to better fits. The larger the number of psi, the more flexible but also more time consuming the calculations.

In this case study, the estimated cell cycle length (lambda.global) and its variation in the cell population (sigma.global) is a common characteristic and should be the same for all genes. Thus we want to fix these global parameters and fit the gene-specific phase and shape (psi) of the gene expression time courses.

First we limit the data matrix to genes that are putatively cell cycle regulated (score > 0) which we can derive from the result of the first screening.

```
> periodic.ids = res.df$ID[res.df$score > 0]
> ccycle = ccycle[periodic.ids,]
```

To achieve better resolution, it makes sense to decrease the spacing between phases in this screening.

```
> phi = seq(5,80,2.5)
> res.shape = fit.periodic(ccycle,timepoints=timepoints,
+         phi=phi,
+         lambda=lambda.global,
+         sigma=sigma.global,
+         psi=2)
> time.courses.shape = predictTimecourses(res.shape)
```

Note that we set the shape-parameter psi to 2 in order to speed up the fitting. The user can set this parameter to higher values to allow a more flexible fitting. We again plot the expression time course of our example gene (YPL133C) together with the fitted time course and extract the phase of its maximal epxression.

```
> t = time.courses.shape[id,]
> predicted.phase = phi[which(t == max(t))]
> plot(timepoints,ccycle[id,],type="l",lwd=2,ylim=c(220,580),
+                 xlab="time [min]",ylab="expression",
+                 main=paste(id,"| peak at",predicted.phase,"min"))
> new.timepoints = seq(5,205,2.5)
> points(new.timepoints,t,type="l",col="limegreen",lwd=3)
> abline(v=predicted.phase,col="limegreen",lwd=3)
```