Estimating parameters from experimental data: The Rust *et al.* [2] model for circadian oscillations

In this practical you will repeat the analysis of Rust *et al.* [2] and estimate a large number of parameters (phosphorylation and de-phosphorylation rates) from their experimental data. We will do this in "R", which is a language to analyze data, and has excellent packages to fit mathematical models to data. R is installed in the UU MyWorkPlace environment, and can easily be installed on your own laptop. We advise you to use the RStudio environment. From this practical you will learn:

- how to fit fairly complicated model to a fairly rich data set
- that one cannot just estimate a large set of parameters from a reasonably large set of data
- that we can repeat the Rust *et al.* [2] analysis by first fitting a subset of parameters on dedicated data, and then fit the others from other dedicated data sets. Parameter estimation therefore requires special design of the experiments.
- even so it remains difficult to find the best parameters
- that estimating the parameters of large systems biology models is highly non-trivial, requiring both the careful design of experiments, as well as making guesses about several parameters of the model.

The Rust *et al.* [2] model. The authors study circadian rhythms of the cyanobacterium *Synechococcus elongatus*. Three adjacent genes at the *kai* locus, KaiA, KaiB and KaiC, that are all regulated by the *kaiBC* operon, are known to be essential for the circadian rhythm in this species [7]. In the Rust *et al.* [2] paper the experimental system is simplified by only considering the three encoded proteins and ATP. Modeling this simple system they thus obtain a minimal model that one can perfectly understand, and which turned out to be a main engine of the circadian rhythm. The KaiC protein can be phosphorylated at two positions, serine-431 and threonine-432, and therefore has four phosphorylation forms, unphosphorylated (U), thReonine (R), serine (S), and double phosphorlylated (D). The sum of the concentrations of these phosphoforms equals the total concentration of KaiC, $\hat{C}$, which is assumed to remain conserved. Studying the regulation of phosphorylation and dephosphorylation by the active form of KaiA, A, they set out to measure the parameters of their mathematical model:

$$
\begin{aligned}
U &= \hat{C} - R - D - S \ , \\
A &= \max(0, \hat{A} - 2S) \ , \\
f_{ij} &= k_{0_{ij}} + k_{A_{ij}} \frac{A}{K_m + A} \ , \quad \text{where} \quad i, j \in \{\mathrm{U}, \mathrm{R}, \mathrm{D}, \mathrm{S}\} \ , \\
\frac{\mathrm{d}D}{\mathrm{d}t} &= f_{\mathrm{RD}} R + f_{\mathrm{SD}} S - f_{\mathrm{DR}} D - f_{\mathrm{DS}} D \ , \\
\frac{\mathrm{d}R}{\mathrm{d}t} &= f_{\mathrm{UR}} U + f_{\mathrm{DR}} D - f_{\mathrm{RU}} R - f_{\mathrm{RD}} R \ , \\
\frac{\mathrm{d}S}{\mathrm{d}t} &= f_{\mathrm{US}} U + f_{\mathrm{DS}} D - f_{\mathrm{SU}} S - f_{\mathrm{SD}} S \ ,
\end{aligned}
$$

where we have three differential equations for: D the concentration of Double phosphorylated KaiC, R the concentration of thReonine phosphorylated KaiC, and S the concentration of Serine phosphorylated KaiC. The concentration of unphosphorylated KaiC, U, is the total concentration of KaiC, $\hat{C}$, minus the concentrations of the three phosphorylated forms. A is the concentration of active KaiA (in μM). Having $\hat{A}$ as the total amount of KaiA (also in μM), active KaiA, A, is the total dimeric KaiA minus that bound by S-KaiC (S) monomers. The max() function guarantees that at high concentrations of S-KaiC the amount of active KaiA cannot drop below zero. The total phosphorylation and dephosphorylation rates follow Michaelis-Menten kinetics, and are composed of a basal rate, k_0, plus or minus a maximum effect of KaiA, k_A, weighed by the KaiA concentration with a Michaelis-Menten constant, K_m. It was assumed that all phosphorylation and dephosphorylation rates in the model obey the same Michaelis-Menten constant, and this constant was fitted numerically in Fig. S4 in the paper, which suggested that $K_m \simeq 0.43\mu$M [2].

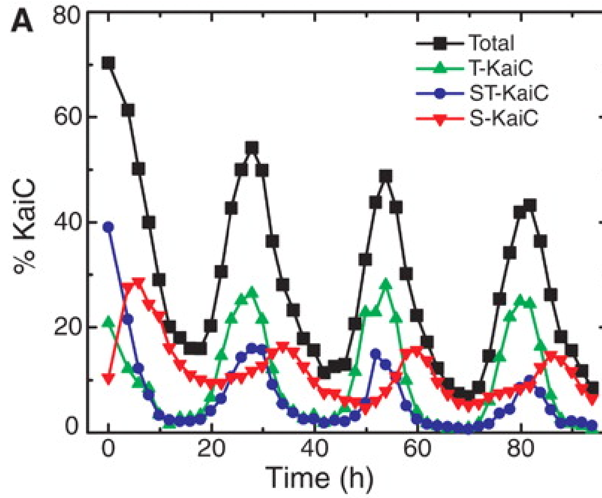

Figure 1: Figure 1a from Rust *et al.* [2]: Phosphorylation of KaiC is cyclically ordered. (A) Decomposition of total KaiC phosphorylation ("Total") into its constituent phosphoforms, measured by SDS-PAGE (used throughout this study unless noted otherwise). The percentage of U-KaiC (not shown) is equal to 100% − Total.

	dephosphorylation				phosphorylation			
Rates	f_{DS}	f_{DR}	f_{RU}	f_{SU}	f_{RD}	f_{SD}	f_{US}	f_{UR}
$k_{0_{ij}}$	0.31	0	0.21	0.11	0	0	0	0
$k_{A_{ij}}$	-0.319	0.173	0.0798	-0.133	0.213	0.506	0.0532	0.479

Table 1: The phosphorylation and dephosphorylation rates of the model, all per hour. For the other parameters we have for the total KaiA: $\hat{A} = 1.3\mu$M, the total KaiC: $\hat{C} = 3.4\mu$M, and $K_m = 0.43\mu$M. Rust *et al.* [2] write that several of the basal rates, k^0_{XY}, were estimated to be very small and could be set to zero without affecting the quality of the fit to the data. In the script `rust.R` we have made all parameters positive by changing f_{ij} into $f_{ij} = k_{0_{ij}} - k_{A_{ij}} \frac{A}{K_m+A}$ for f_{DS} and f_{SU}.

The three differential equations depend on 16 phosphorylation and dephosphorylation rates, which have been estimated in Fig. 2 of the paper [2], that has data in the absence of KaiB (Fig. 2a) or in the absence of KaiA (Fig. 2b). These parameter estimates are given in Table 1 of this handout, and in Table S2 of the paper [2]. Finally, note the modeling on this system is still ongoing as the parameters of this model were recently estimated using a more biophysical approach [1].

Using Grind. Today you will work with an R-script called `grind.R` which is a wrapper around the R-packages `deSolve`, `FME` and `rootSolve` developed by Karline Soetaert and colleagues [3–6]. These packages allow one to solve differential equations, find their steady state, and perform nonlinear parameter estimation. Today you only need three of Grind's easy-to-use functions:

- `run()` integrates a model numerically and provides a time plot or a trajectory in the phase plane,
- `fit()` fits a model to data by estimating its parameters, and depicts the result in a timeplot.
- `timePlot()` plots a data frame.

The full manual of Grind is available on the website `http://tbb.bio.uu.nl/rdb/grindR/`.

We will work in the RStudio environment. To get started perform the following:

- **If you work on your own laptop** you will need to install RStudio, R, and the Soetaert libraries. Check this tutorial on how to install RStudio and R:
  `http://tbb.bio.uu.nl/rdb/bm/Introduction_to_R.pdf`

After that install the three Soetaert libraries by opening RStudio, going to the Tools menu, choosing Install Packages, and then typing `deSolve`, `FME` and `rootSolve`. (Experts may prefer to type `install.packages(c("deSolve", "FME", "rootSolve")` in the R-console).
- **Otherwise**: skip the previous step and go to the University's MyWorkPlace environment.
- Make a local folder (directory) on your system where you will save the following files: the two R-scripts `grind.R` and `rust.R`, and the four data files the four data files `fig1a.txt`, `fig2a.txt`, `fig2b.txt`, and `figS3.txt`. All files can be obtained by downloading the `circadian.zip` file from Blackboard or directly from the `tbb.bio.uu.nl/rdb/practicals/circadian/` website. (In Windows use the right mouse button and use "Save target as", to store a file in a folder).
- Set the working directory of RStudio to the folder where your R-codes and data are stored (`Set working directory` in the `Session` menu of RStudio). Files will then be opened and saved in that directory.
- Use "Open file" to open `grind.R` and **source** Grind to define the functions (button on the right).
- Use "Open file" to open `rust.R` and **run** that script line by line.

Thus, after installing Grind and downloading the files, you first "Source" the `grind.R` file (button in right hand top corner), and then "Run" the model with its parameter and state definitions line by line ("Run" or "Control Enter").

The Rust *et al.* [2] parameters are defined in the parameter vector `p`, by joining three vectors with the default k_0, the k_A, the K_m and the KaiA, KaiB and KaiC concentrations. The concentrations of D (Double phosphorylated KaiC), R (thReonine phosphorylated KaiC), and S (Serine phosphorylated KaiC) determine the state of the system, and are defined in the state vector `s`. Proceed slowly (line-by-line) through the `rust.R` file, by clicking `Run` (or control R), and make sure that you understand what is happening. Make notes!

The Figure 1a data. The data of Fig. 1a and the behavior of the model can be plotted together by running the first lines of the R-script. Take some time to familiarize yourself with using R, and the way the parameters are defined. Questions:

a. What are the main differences between the model behavior and the data of Fig. 1a?
b. What could be a reason for this, and how could this be taken into account?
c. Why do we need to transform the data from the paper (and how does the script do this)?

The Figure 2b data. Next we follow their procedure to fit subsets of the parameters to data collected in under dedicated simplified conditions. In Grind the vector `free` contain the names of the parameters to be fitted, and their initial values are taken from the parameters `p`. Rust *et al.* [2] performed dedicated experiments to estimate subsets of parameters. In Fig. 2b they study the four basal dephosphorylatyion rates by omitting KaiA (KaiC completely dephosphorylates in the absence of KaiA). In the R-script we set `p["KaiA"] <- 0` (make sure to set it back to 1.3 at the end). Repeat their estimation procedure by stepping through the script. Note that parameter estimates are inserted in the parameter vector in the line `p[free] <- par2b`, where `free` contains the names of the fitted parameters, and `par2b` are the estimates taken from the list returned by fit (`par2b <- fit2b$par`). Questions:

a. What is the meaning of the `p[free] <- runif(length(free),0,0.5)` line?
Hint: type `?runif` in the R-console.
b. Do you always get the same values for the four dephosphorylatyion parameters they estimate?
Hint: fit several times by drawing novel random guesses.
c. **Extra question**: what happens if you would allow for some autophosphorylation in this experiment i.e., if you allow all eight k_0 parameters to be fitted (which is achieved by typing `free <- names(k0)`)? Do you get a better fit? Was that to be expected? Do you obtain oscillations for these new parameters?

The Figure 2a data. Having estimated the k_0 parameters, Rust *et al.* [2] proceed by estimating

the k_A parameters. They prevent oscillations by omitting the inhibitor KaiB (in the model we set `p["KaiB"] <- 0`, make sure to set it back to 2 at the end). They found that the data in Fig. 2a is not sufficient to estimate all k_A parameters, and performed another experiment (Fig. S3) with another initial condition to have more data. Repeat their estimation procedure by stepping through the script. Try several initial guesses until you find a reasonable fit to both data sets. Finally, the estimates are inserted in the parameters to test whether the model behavior is periodic. Questions:

a. What is the difference between the Fig. 2a data and the data in Fig. S3?

b. Do you get the same values for the eight k_A parameters they estimate?

c. How do these estimates depend on the k_0 values that you use?

d. Do you get oscillations?

e. Why do the authors first fit parameters to Fig. 2b and then to Fig. 2a? Why do they use Fig. 2b to fit the k_0 and Fig. 2a to fit the k_A parameters?

f. In the last call to `fit()` we set `scaleVar=TRUE` to weigh each data set equally. The Fig. 2a data contains many more data points than the Fig. S3 data. What happens if you don't weigh the two datasets similarly?

The Michaelis-Menten constant. If you still have time you might want to study the dependence of the oscillations on the Michaelis-Menten parameter K_m. On page 15 of the SI the authors write that their model shows circadian oscillations for $0.092 < K_m < 0.93\mu$M, and that their standard value of $K_m = 0.43\mu$M lies well in this range. Thus, it should not matter if their parameter values were slightly incorrect, as the qualitative behavior of the model should stay the same. Questions:

a. Check their statement by setting K_m, e.g., `p["Km"] <- 0.9` and running the model.

b. Study how the period of the oscillation depends on the value of K_m.

October 21, 2018, Rob J. de Boer and Kirsten ten Tusscher, Utrecht University

References

[1] **Paijmans, J., Lubensky, D. K., and Ten Wolde, P. R.**, 2017. A thermodynamically consistent model of the post-translational Kai circadian clock. PLoS. Comput. Biol. **13**:e1005415.

[2] **Rust, M. J., Markson, J. S., Lane, W. S., Fisher, D. S., and O'Shea, E. K.**, 2007. Ordered phosphorylation governs oscillation of a three-protein circadian clock. Science **318**:809–812.

[3] **Soetaert, K.**, 2009. rootSolve: Nonlinear root finding, equilibrium and steady-state analysis of ordinary differential equations. R package 1.6.

[4] **Soetaert, K. and Herman, P. M.**, 2009. A Practical Guide to Ecological Modelling. Using R as a Simulation Platform. Springer. ISBN 978-1-4020-8623-6.

[5] **Soetaert, K. and Petzoldt, T.**, 2010. Inverse modelling, sensitivity and monte carlo analysis in R using package FME. Journal of Statistical Software **33**:1–28.

[6] **Soetaert, K., Petzoldt, T., and Setzer, R. W.**, 2010. Solving differential equations in R: Package desolve. Journal of Statistical Software **33**:1–25.

[7] **Tomita, J., Nakajima, M., Kondo, T., and Iwasaki, H.**, 2005. No transcription-translation feedback in circadian rhythm of KaiC phosphorylation. Science **307**:251–254.

rust.R

```
model <- function(t,state,parms) {
  with(as.list(c(state,parms)),{
    U = max(0,KaiC - R - D - S)
    A = max(0,KaiA - KaiB*S)
    fA = A/(Km + A)
    fUR = k0UR + kaUR*fA
    fDR = k0DR + kaDR*fA
    fRU = k0RU + kaRU*fA
    fRD = k0RD + kaRD*fA
    fSD = k0SD + kaSD*fA
    fDS = k0DS - kaDS*fA
    fUS = k0US + kaUS*fA
    fSU = k0SU - kaSU*fA
    dD = fRD*R + fSD*S - fDR*D - fDS*D
```

```
    dR = fUR*U + fDR*D - fRU*R - fRD*R
    dS = fUS*U + fDS*D - fSU*S - fSD*S
    return (list(c(dD,dR,dS)))
  })
}

# Set some parameters for nicer graphics (keeping the default in opar)
opar <- par
par(mar=c(2.6,2.6,1.6,0.2),mgp=c(1.5,0.5,0))       # mar=c(B,L,T,R)

# The parameters from the paper:
k0 <- c(k0DS=0.31,k0DR=0,k0RU=0.21,k0SU=0.11,k0RD=0,k0SD=0,k0US=0,k0UR=0) # Table S2
ka <- c(kaDS=0.319385,kaDR=0.173,kaRU=0.0798462,kaSU=0.133077,            # Table S2
        kaRD=0.212923,kaSD=0.505692,kaUS=0.0532308,kaUR=0.479077)
K <- c(KaiA=1.3,KaiB=2,KaiC=3.4,Km=0.43)  # the protein concentrations
p <- c(k0,ka,K)                 # the complete parameter vector p
s <- c(D=1,R=0.5,S=0.2)         # the state vector s

# Here the session starts

run()
data1a <- read.table("fig1a.txt",header=TRUE)    # Read the data of Fig1a
data1a[,2:4] <- p["KaiC"]*data1a[,2:4]/100       # Transform the data
timePlot(data1a,add=TRUE,draw=points)

run(state=unlist(data1a[1,2:4]))                # Take initial condition from data
timePlot(data1a,add=TRUE,draw=points)

# Fit dephophorylation rates from Fig 2b data
p["KaiA"] <- 0                                  # In Fig 2b KaiA=0
data2b <- read.table("fig2b.txt",header=TRUE)
data2b[,2:4] <- p["KaiC"]*data2b[,2:4]/100      # Transform the data
timePlot(data2b,draw=points,main="2b")
free <- c("k0DS","k0DR","k0RU","k0SU")          # Fit 4 basal k0 rates
p[free] <- runif(length(free),0,0.5)            # Provide a random initial guess
fit2b <- fit(data2b,free=free,ymax=1.5,lower=0,upper=1,initial=TRUE,main="2b")
summary(fit2b)
par2b <- fit2b$par                              # Save estimated parameters
par2b <- ifelse(par2b<1e-3,0,par2b)             # Set small parameters to zero
p["KaiA"] <- 1.3                                # Reset KaiA
p[free] <- par2b                                # Include estimates in parameters
run()                                           # Inspect new model behavior

# Now fit phophorylation rates from Fig 2a data
p["KaiB"] <- 0                                  # In Fig 2a there is no KaiB
data2a <- read.table("fig2a.txt",header=TRUE)
data2a[,2:4] <- p["KaiC"]*data2a[,2:4]/100      # Transform the data
dataS3 <- read.table("figS3.txt",header=TRUE)
dataS3[,2:4] <- p["KaiC"]*dataS3[,2:4]/100      # Transform the data
par(mfrow=c(2,1))                               # Show data with a run():
timePlot(data2a,legend=FALSE,draw=points,main="2a")
run(20,state=unlist(data2a[1,2:4]),add=TRUE,legend=FALSE)
timePlot(dataS3,legend=FALSE,draw=points,main="3S")
run(10,state=unlist(dataS3[1,2:4]),add=TRUE,legend=FALSE)
par(mfrow=c(1,1))

free <- c("kaDS","kaDR","kaRU","kaSU","kaRD","kaSD","kaUS","kaUR")
p[free] <- runif(length(free),0,0.5) # Random initial guess
par(mfrow=c(2,1))
fit2as3 <- fit(list(data2a,dataS3),free=free,ymax=2,lower=0,upper=1,initial=TRUE,
    scaleVar=TRUE,main=c("2a","3S"),legend=FALSE)
par(mfrow=c(1,1))         # scaleVar scales both data sets equally!
par2as3 <- fit2as3$par
p["KaiB"] <- 2            # Reset KaiB to its correct value
p[free] <- par2as3        # Insert estimates in parameter vector p
run()                     # Inspect model behavior
```

```
p                              # and the estimated parameters

# Plot the Figure 1 data to test if parameters describe that data
timePlot(data1a,draw=points)
run(state=unlist(data1a[1,2:4]),add=TRUE,legend=FALSE)

# Run the model for the Rust parameters for various of Km:
p <- c(k0,ka,K)               # Revert to the original parameters
s <- c(D=1,R=0.5,S=0.2)       # and state
p["Km"] <- 0.9; f <- run()

newton(f)                             # For the "experts": find steady state
continue(f,x="Km",step=0.01)          # and follow it for several of Km
```