# Inferring model Parameters in Systems Biology

Instructor(s): Samuel Bernard bernard@math.univ-lyon1.fr

link to the latest version: http://math.univ-lyon1.fr/homes-www/bernard/teach/systems_biology/mi.pdf

parameter inference

The problem of **parameter inference** is the problem of estimating the values of model parameter, given data, and the uncertainty on these values.

model function

Let us consider a dataset with $n$ observations $y_i$ at time points $t_i$, $i = 1, ..., n$, and a **model function** $m(t; \theta)$ that depends on time and on a certain number of parameters $\theta = (\theta_1, \theta_2, \theta_k)$. Can we find values for $\theta$ such that the function $m$ will take values close to $y_i$ when evaluated at $t_i$:

$$m(t_i; \theta) \sim y_i$$

interpolation

In other words we are trying to minimize the difference between $m_i = m(t_i; \theta)$ and $y_i$, for all $i = 1, ..., n$. If the number of parameter $k$ equal to or larger than $n$, it is in general possible to match exactly all the data: $m_i = y_i$. This is called **interpolation**. However, when the number of parameters if smaller than $n$, it will be in general not possible to match all data points, and there will be a trade-off to achieve between being far-off for few data points and close to the others, or being a bit further away from all data point.

minimization problem

sum of squares of the errors

The way to resolve this trade-off is to pose a **minimization problem**. That is, we will define a scalar-valued objective-function $f$ that depends on $m$ and $y$ and has the property that $f(y, y) = 0$. By far the most common objective-function used in biology is the **sum of squares of the errors** between $m_i$ and $y_i$:

$$f(m, y) = \sum_{i=1}^{n} (m_i - y_i)^2 = ||m - y||^2$$

Clearly, the smaller the sum of square of the error (or SSE) is, the closer the model is to the data. The SSE can only take values greater or equal to zero, so we are guaranteed to have at least one global minimum. Trying to find the parameters that minimize the SSE is called least-square fitting. Least-square fitting has at least one solution, but in real life there is often many parameters sets that will minimize the SSE.

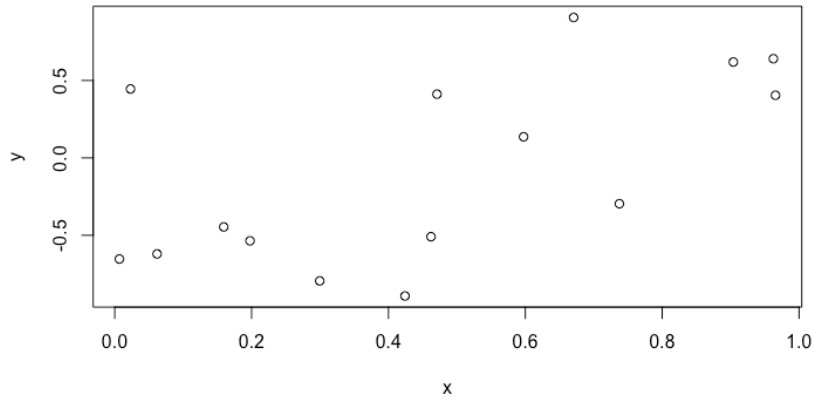


**Figure 1.** raw data

# 1 Small is beautiful

Why bothering with a small number of parameters, when taking $k = n$ is sufficient to fit perfectly (SSE=0) the dataset? The bottom line answer is: models with small numbers of parameters have more explanatory power than models with large numbers of parameters, and they have a better predictive power.

Let see first why too many parameters is not good for prediction. We randomly generate n=15 pairs of points, $x$ in [0,1] and $y$ in [-1,1].

```
n <- 15
x <- runif(n) # generate n random numbers in [0,1]
y <- runif(n, min = -1, 1) # generate response in [-1,1]
plot(x,y)
```

Now we fit a polynomial of degree 14 (it has 15 coefficients/parameters)

```
    interp <- lm(y ~ poly(x,n-1))
^^I^^Iplot(x,y)
^^I^^Isx <- sort(x)
^^I^^Ilines(sx, predict(interp, data.frame(x = sx)))
```

The fit is perfect, the polynomial function goes through all the data points. Suppose we get a new pair of data $(x, y)$. How good will be the prediction?

```
newx <- runif(1)
newy <- runif(1) # add a new data point
plot(c(0,1),c(-1,1), type = 'n', xlab = 'x', ylab = 'y')
```

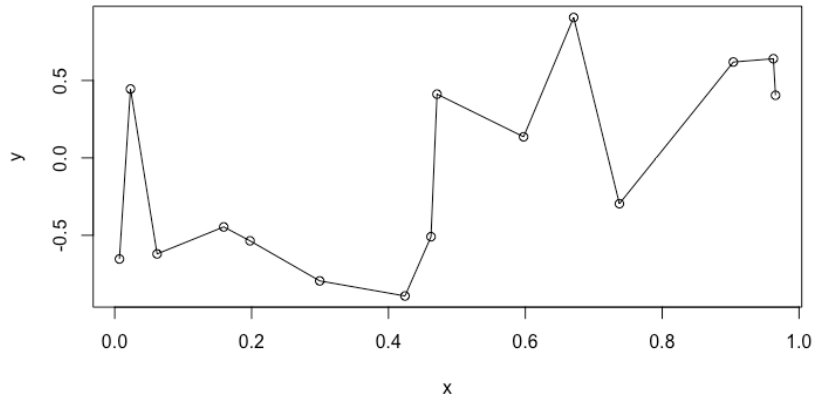


**Figure 2.** interpolated data

```
points(x,y)
points(newx,newy, pch = 19)
lines(sort(c(x,newx)), predict(interp, data.frame(x = sort(c(x,newx)))))
points(sort(c(x,newx)), predict(interp, data.frame(x =
↪  sort(c(x,newx)))), pch = 3)
```

The prediction is terrible! The reason for that is that a polynomial function of degree 14 is 1) quite oscillatory, and 2) quit unbounded. But the problem is not restricted to polynomials; any interpolating function will be bad at predicting the new data point because they fit noise. Given a new data point, the best guess for $y$ is 0, i.e. the mean of the random numbers between -1 and 1. Interpolation will pick noise in the neighbouring data points and propagate is to the new prediction.

Instead of interpolating the data, we now perform a standard linear regression of the data. Prediction is now much better, and more in line with the intuitive understanding of a prediction.

The linear regression has two parameters, (intercept and slope). Between 2 and 15 parameters, there is probably a optimal number of parameters that can be fitted for a given dataset.

## 2 Nonlinear regression with a known model function

We use the pharmacokinetics of Indomethacin dataset `Indometh` from the `datasets` package in R.

```
Indo.1 <- Indometh[Indometh$Subject == 1, ]
with(Indo.1, plot(time,conc))
```

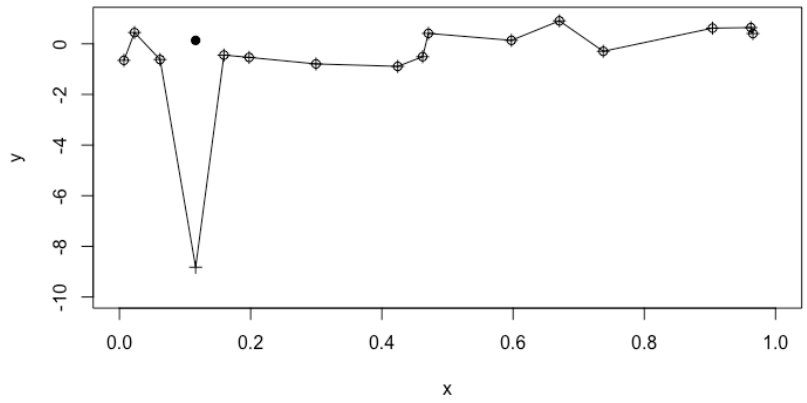


**Figure 3.** Prediction of a new random data point

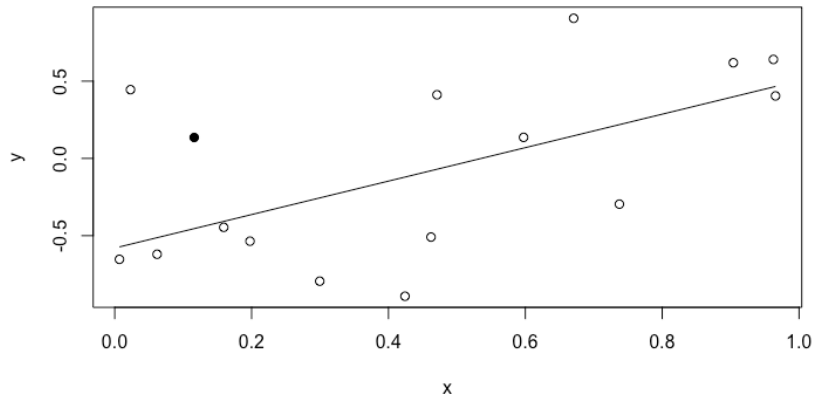


**Figure 4.** Linear regression of the data

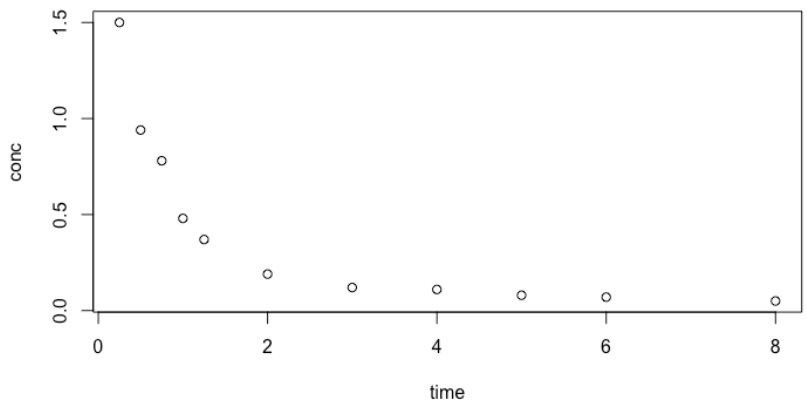


**Figure 5.** Indometh dataset, patient 1

We perform a nonlinear least-square fit with the routine `nls` from R. The response variable is the concentration `conc`, the independent variable is `time`, and the model is the **exponential model**

exponential model

$$m(t) = c_0 \exp(-kt)$$

with two parameters $c_0$ and $k$.

```
indometh.nls <- nls(conc ~ c0 * exp(- k * time),  # model
data = Indo.1,                 # dataset
start = list(c0 = 1, k = 0))  # initial guess for the parameters
lines(seq(0,8, by = 0.1),predict(indometh.nls, list(time = seq(0,8, by =
↪  0.1))))
summary(indometh.nls)
```

The output of the last command shows the estimated values (column `Estimate`) and error on the estimate (column `Std. Error`)

```
Formula: conc ~ c0 * exp(-k * time)

Parameters:
   Estimate Std. Error t value Pr(>|t|)
c0   2.0332     0.1392   14.60 1.42e-07 ***
k    1.3563     0.1232   11.01 1.60e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.07371 on 9 degrees of freedom

Number of iterations to convergence: 10
Achieved convergence tolerance: 8.285e-06
```

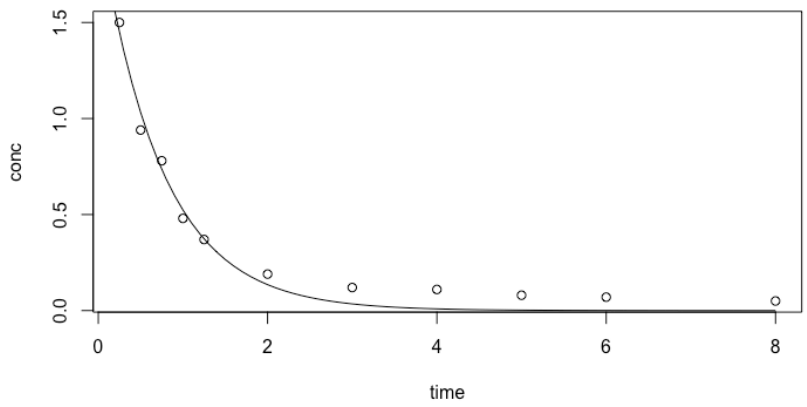


**Figure 6.** Indometh data fitted with `conc ~ c0 * exp(-k * time)`

| Name | Equation |
|---|---|
| Michaelis-Menten | $y = \frac{ax}{1+bx}$ |
| 2-parameter asymptotic exponential | $y = a(1 - e^{-bx})$ |
| 3-parameter asymptotic exponential | $y = a - be^{-cx}$ |
| Gompertz | $y = ae^{-be^{-cx}}$ |
| Bi-exponential | $y = ae^{bx} + ce^{-dx}$ |

**Table 1.** Non-linear functions in biology.

# 3 List of useful non-linear functions

# 4 Nonlinear regression with an ODE model

To have a basis for comparison, we use the same dataset and the same model, except we express the model as an ODE

$$\frac{dC}{dt} = -kC$$

with initial condition $C(0) = c_0$. The R code is a bit more complex

```
rhs <- function(Time, conc, k) {
    return(list(c(- k * conc)))
}

odemodel <- function(Time,c0,k) {
  timespan <- sort(Time)
  addedzero <- FALSE
  if (timespan[1] > 0)
  {
    timespan <- c(0, timespan)
```

```
    addedzero <- TRUE
  }
  expmodel.sol <- ode(c(conc = c0), timespan, rhs, k)
  if (addedzero)
  {
    return( expmodel.sol[-1,2])
  }
  else
  {
    return(expmodel.sol[,2])
  }
}

with(Indo.1, plot(time,conc))
indomethode.nls <- nls(conc ~ odemodel(time, c0, k),  # model
    data = Indo.1,                  # dataset
    start = list(c0 = 5, k = 1.2))  # initial guess for the parameters
summary(indomethode.nls)
```

The results is identical to the exponential model

```
^^I^^IFormula: conc ~ odemodel(time, c0, k)

^^I^^IParameters:
^^I^^I^^I Estimate Std. Error t value Pr(>|t|)
^^I^^Ic0   2.0332     0.1392   14.60 1.42e-07 ***
^^I^^Ik    1.3563     0.1232   11.01 1.60e-06 ***
^^I^^I---
^^I^^ISignif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

^^I^^IResidual standard error: 0.07371 on 9 degrees of freedom

^^I^^INumber of iterations to convergence: 7
^^I^^IAchieved convergence tolerance: 3.348e-06
```

## 5 Refining and selecting a model

The fit with the exponential model is quite good at initial times, but after 2 hours, the concentration decreases more slowly than the fitted exponential. This may be because there is residual amount of drugs in peripheral tissues with slower elimination rates. A **bi-exponential model** can take care of the two sections of the pharmacokinetic data

bi-exponential model

$$b(t) = c_{01}\exp(-k_1 t) + c_{02}\exp(-k_2 t)$$

The R code is

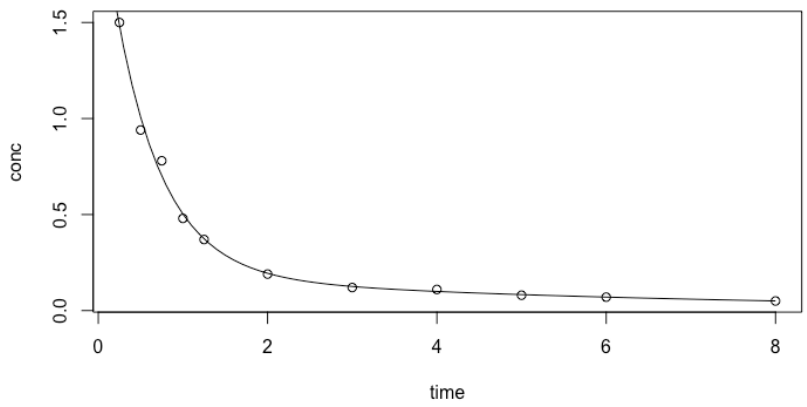


**Figure 7.** Indometh data fitted with a bi-exponential

```
biexpmodel <- function(Time,c01,c02,k1,k2) {
    return(c01*exp(- k1 * Time) + c02*exp(- k2 * Time))
}
with(Indo.1, plot(time,conc))
indometh.biexp <-
        nls(conc ~ biexpmodel(time,c01,c02,k1,k2),  # model
        data = Indo.1,                # dataset
        start = list(c01 = 2,
                 c02 = 1,
                 k1 = 2,
                 k2 = 1))  # initial guess for the parameters
lines(seq(0,8, by = 0.1),predict(indometh.biexp,
        list(time = seq(0,8, by = 0.1))))
```

The fit looks much better! That the fit is better is to be expected, the bi-exponential includes as a subset the exponential model. So, it is really better? One way to test this is with the Akaike Information Criterion. The Akaike Information Criterion (AIC) can be used select the most appropriate model. It takes into account the SSE and the number of model parameters.

```
print(c(AIC_biexp = AIC(indometh.biexp),
        AIC_exp = AIC(indometh.nls)))

AIC_biexp   AIC_exp
-34.01319 -22.35806
```

The lower the AIC, the better. Here the bi-exponential clearly wins over the exponential model.

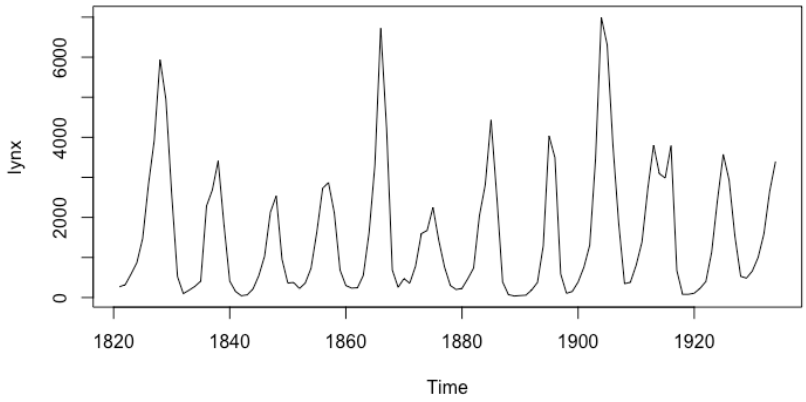


**Figure 8.** Number of lynx trapped in Canada

## 6 Exercises

(a) The bi-exponential model can also be expressed as the solution of a system of ODEs. How could it look likes?

(b) Figure 8 shows the number of lynx in trapped in Canada between 1821 and 1934. If you were a planner, trying to manage the lynx population in a sensible way, how would you do?

## 7 References

MJ Crawley, The R Book, 2012, Wiley

## Index