Inference of interaction networks using generalized Lotka Volterra equations on time-series data with package gLVInterNetworks

Lukas Hirsch

2017-01-03

Installation

To download the package from my github account use:

```
devtools::install_github("lkshrsch/gLVInterNetworks", build_vignettes = TRUE)
```

Note: After installing the package for the first time, or updating to a new version, restart the RStudio session before usage.

To load the package into the R environment

```
library(gLVInterNetworks)
```

Generation of in silico data

To generate in silico data sets we use the following command with the following mandatory arguments:

- species: The amount of variables in the system
- number_of_interactions: The amount of interactions between different variables
- timepoints: Starting from 0, the number of measurements the output will have
- noise: The standard deviation of the stochasticity added to the data (variance of the gaussian noise)
- testData: The percentage of ending datapoints left out for validation purposes.

```
data <- gLVgenerateData(species = 2,
                        number_of_interactions = 2,
                        timepoints = 40,
                        noise = 0.01,
                        testData = 20)
```

Note that depending on the size of the system to simulate (number of species, etc) this command can output messages which can be ignored regarding the difficulty of the numerical integration of the solution.

```
data("exampleData1")
```

The variable data is a list containing the following attributes

```
names(exampleData1)
```

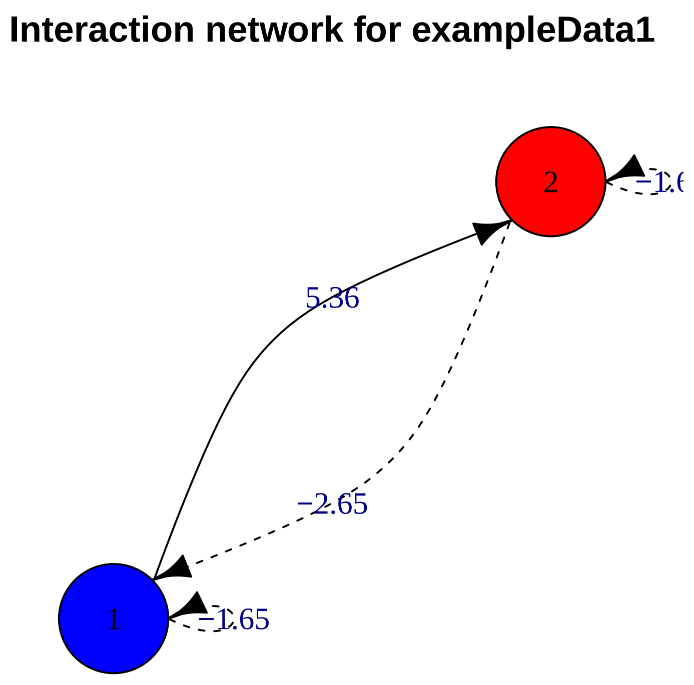
```
## [1] "species"    "timepoints" "Parms"      "noise"      "sparsity"
## [6] "obs"        "testData"
```

From which *species*, *timepoints*, *noise*, *sparsity*, and *testData* correspond to the input arguments.

- Parms : Is the randomly generated parameter matrix under the given input constraints. It contains one row per variables / species in the system and the parameters correspond to :
- obs : Corresponds to the solution using the estimated model parameters for the time interval given. This are the values plotted when using

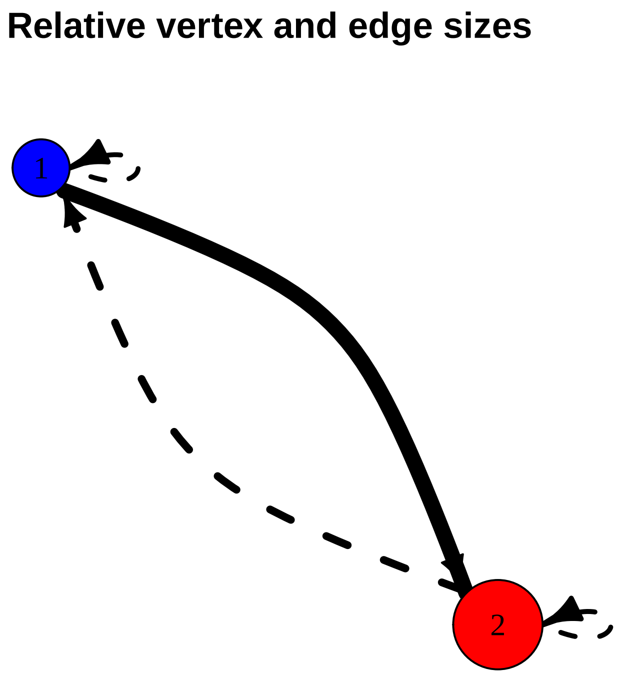
To visualize the representation of the system as an interaction network, use the plotGraph() function over the object. Arguments vsize and edgeSize adjust the size of the vertices and edges respectively and must be tuned by the user when visualizing different sized networks. The "verbose" argument or flag prints also the numeric value of the interaction parameters over the respective edges connecting the variables.

```
plotGraph(x = exampleData1, vsize = 50, edgeSize = 1, main="Interaction network for exampleData1", verb
```



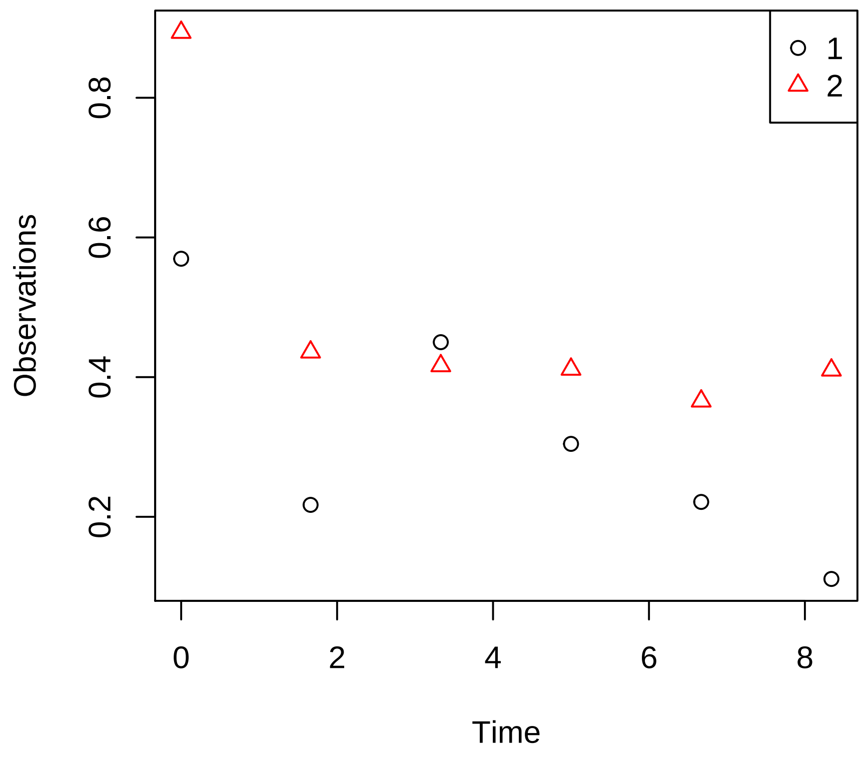
Optionally, the network can be plotted with sizes of the vertices and edges relative to the magnitude they represent by activating the relativeVertexSize and relativeEdgeSize flags (setting them to TRUE, with default value FALSE). In this case, the vertex sizes are proportional to the mean abundance of the respective variable they represent, weighted by an arbitrary and user set scaling factor given by the argument "vsize". The "relativeEdgeSize" flag sets the width of the edges relative to the magnitude of the interaction parameter they represent.

```r
plotGraph(x = exampleData1, relativeVSize = TRUE ,vsize = 25, relativeEdgeSize = TRUE, main="Relative ve
```



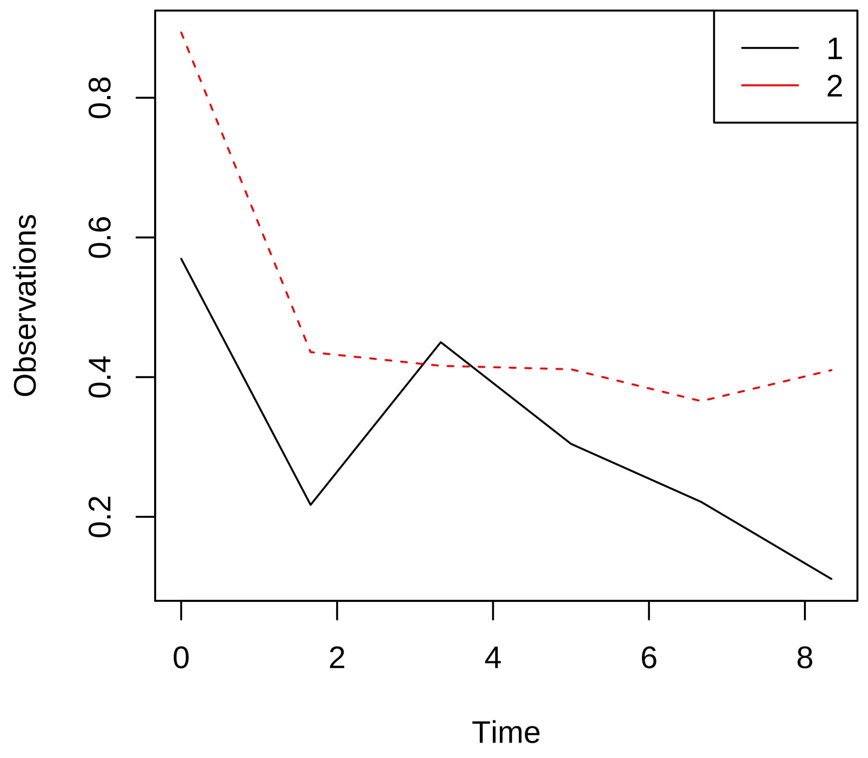
Basic plotting of the time series data:

```r
plot(exampleData1, pch = 1:exampleData1$species)
legend("topright",
       legend = colnames(exampleData1$obs[,-1]),
       pch = 1:exampleData1$species,
       col=1:exampleData1$species)
```



Or alternatively with the legend flag set to TRUE for an easy plot generation

```
plot(exampleData1, legend = T)
```



Loading data sets

The package comes with a small example of raw data stored in a csv file with a time column with the dates when the observations were taken. The .csv file is a text file with entries sepparated by semicolons ";". To get the absolute path of the file 'example1.csv' installed in the package directory 'rawdata/' use

```
fpath <- system.file("rawdata", "example1.csv", package="gLVInterNetworks")
```

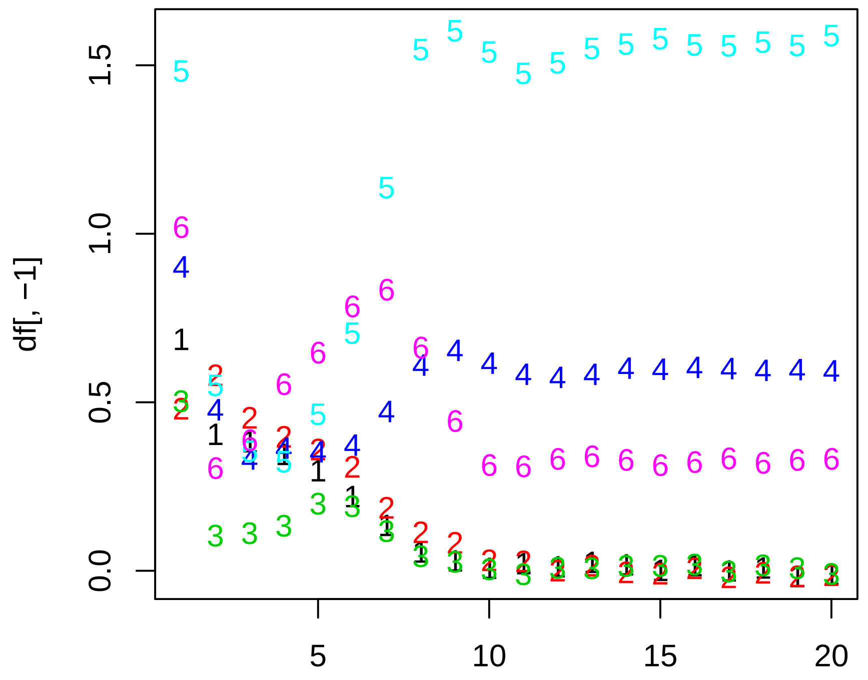
The load the file

```
df <- read.csv2(file = fpath, header = TRUE)
```

```
head(df)
```

```
##       time     Gate1     Gate2     Gate3    Gate.C     Gate5    Gate.6
## 1 01. Feb 0.6873188 0.4814508 0.5028041 0.9024577 1.4820795 1.0192897
## 2 02. Feb 0.4056058 0.5804809 0.1040970 0.4784449 0.5495137 0.3043241
## 3 03. Feb 0.3836464 0.4544713 0.1112002 0.3323488 0.3541356 0.3869106
## 4 04. Feb 0.3459959 0.3978596 0.1333045 0.3648185 0.3228151 0.5533130
## 5 05. Feb 0.2974755 0.3602963 0.1988122 0.3538549 0.4635254 0.6470609
## 6 06. Feb 0.2209130 0.3089127 0.1913458 0.3735745 0.7044591 0.7841929
```

```
matplot(df[,-1])
```



We can convert the data.frame to a numerical matrix using

```
exampleData <- data.matrix(df)
```

With which we have transformed the dates into numerical values. The original measurement time rate (per day) will be the changing rate for the estimated parameters after fitting the model to the data.

```
head(exampleData)
```

```
##      time     Gate1     Gate2     Gate3    Gate.C     Gate5    Gate.6
## [1,]    1 0.6873188 0.4814508 0.5028041 0.9024577 1.4820795 1.0192897
## [2,]    2 0.4056058 0.5804809 0.1040970 0.4784449 0.5495137 0.3043241
## [3,]    3 0.3836464 0.4544713 0.1112002 0.3323488 0.3541356 0.3869106
## [4,]    4 0.3459959 0.3978596 0.1333045 0.3648185 0.3228151 0.5533130
## [5,]    5 0.2974755 0.3602963 0.1988122 0.3538549 0.4635254 0.6470609
## [6,]    6 0.2209130 0.3089127 0.1913458 0.3735745 0.7044591 0.7841929
```

Fitting the generalized Lotka Volterra model to the data

The generalized Lotka Volterra model consists in a set of differential equations, nonlinear in the variables x. It describes the rate of change in time of the abundance of a variable (species, or taxonomical/functional unit)

$$\frac{dx_i}{dt} = \alpha_i x_i + \sum \beta_{ij} x_i x_j$$

Or written simpler

$$\dot{x_i} = x_i(\alpha_i + \sum \beta_{ij} x_j)$$

Written in matrix form for a two species system:

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \times \left[\begin{pmatrix} \alpha_1 & \beta_{11} & \beta_{12} \\ \alpha_2 & \beta_{21} & \beta_{22} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \right]$$

This matrix form for the parameter matrix with the growth term in the first column and interaction parameters in the right-side quadratic matrix corresponds to the output of the parameter in the R code:

```r
data("exampleData1")
print(exampleData1$Parms)
```

```
##          growth
## [1,]  1.5334121 -1.653507 -2.652309
## [2,] -0.7558098  5.358952 -1.692742
```

$$= \begin{pmatrix} \alpha_1 & \beta_{11} & \beta_{12} \\ \alpha_2 & \beta_{21} & \beta_{22} \end{pmatrix}$$

And for bigger systems such as the six species simulation of "exampleData2"

```r
data("exampleData2")
print(exampleData2$Parms)
```

```
##          growth
## [1,]  2.1118070 -1.297752  0.000000  3.997974 -2.9417880  0.000000
## [2,] -0.7378096  0.000000 -1.867226  1.589395  5.4730475 -2.216080
## [3,]  3.9810420  2.710835 -6.945811 -1.537990  0.0000000 -3.820136
## [4,]  0.1564243  3.865123 -3.941819 -3.679745 -1.3240106  0.000000
## [5,]  1.9821135 -3.323225 -3.122314  0.000000  0.8075217 -1.989126
## [6,]  3.6807751  0.000000  0.000000  0.000000 -8.9701972  1.306043
##
## [1,] -2.625256
## [2,]  0.000000
## [3,]  0.000000
## [4,]  1.906246
## [5,]  2.062700
## [6,] -1.251722
```

Model parameters for variables x_1 to x_6

$$= \begin{pmatrix} \alpha_1 & \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} & \beta_{15} & \beta_{16} \\ \alpha_2 & \beta_{21} & \beta_{22} & \beta_{23} & \beta_{24} & \beta_{25} & \beta_{26} \\ \alpha_3 & \beta_{31} & \beta_{32} & \beta_{33} & \beta_{34} & \beta_{35} & \beta_{36} \\ \alpha_4 & \beta_{41} & \beta_{42} & \beta_{43} & \beta_{44} & \beta_{45} & \beta_{46} \\ \alpha_5 & \beta_{51} & \beta_{52} & \beta_{53} & \beta_{54} & \beta_{55} & \beta_{56} \\ \alpha_6 & \beta_{61} & \beta_{62} & \beta_{63} & \beta_{64} & \beta_{65} & \beta_{66} \end{pmatrix}$$

Nonlinear Regression to estimate values for the model parameters

To estimate the parameter matrix from the equations above, use

```r
nlrData2 <- gLVnonlinearRegression(data = exampleData2)
```

The resulting list containts 12 objects, named Parms, SSR, residual_SD, SE, residuals_t.test, message, obs, Fit, df, quantitative, qualitative1, qualitative2.

- **SSR** : The value of the sum of squared residuals of the found solution
- **residual_SD** : The estimated standard deviation of the data stochasticity in respect to the solution found
- **SE** : A matrix contatining the standard errors of each estimated parameter
- **residuals_t.test** : The result of a t test with the null hypothesis of the residuals from the model solution being normally distributed with standard error 'residual_SD'
- **message** : A text message related to the reason why the optimization algorithm stopped
- **obs** : A matrix with the calculated abundances on each timestep according to the estimated parameters
- **Fit** : The object returned by the optimization algorithm. It is a list containing further details of the optimization algorithm run
- **df** : The degrees of freedom
- **quantitative** : Only when validating against in Silico data : The ratio of estimated parameters that contain the original value within the 95% confidence interval given by their standard error (displayed in 'SE')
- **qualitative1** : Only when validating against in Silico data : The ratio of correctly retrieved edges from the original interaction network
- **qualititative2** : Only when validating against in Silico data : The ratio of correctly retrieved edges from the original interaction network after setting estimated parameters that are not significantly different from zero to zero (i.e. parameters that contain the zero in their 95% confidence interval as given by their standard error)

To see explanations for each input argument and output values at any time during the R session use the help function

```r
?gLVnonlinearRegression.
```

Interpretation of results and model identifiability

Without knowing the original mechanisms and causes that generated the data, the only reasonable interpretation of these results is the goodness/usefulness of the approximation given by the solution found and the uniqueness of this solution.

In this case, the algorithm achieved a fit with a total sum of squared residuals of

```
print(nlrData2$SSR)
```

```
## [1] 0.008053986
```

Note that this value as a standalone value does not have any objective meaning, only relative, when comparing different solutions.

The estimated parameter matrix is

```
print(nlrData2$Parms)
```

```
##             [,1]          [,2]        [,3]        [,4]       [,5]
## [1,]  1.79223499 -1.6710904970 -0.04198621  3.53269933 -1.4976110
## [2,] -1.14900477  4.4813918003 -3.63452284  2.71661123  2.3632050
## [3,]  5.68942535 -0.0005482157 -8.21869778  1.05303570  1.7927713
## [4,]  0.08948867  5.5097018443 -5.47304844 -4.05040893 -0.7140135
## [5,]  1.81274708 -3.4934328640 -3.30201388 -0.08410942  1.6976769
## [6,]  3.57599179  2.9824639899 -1.69752061  0.48877777 -9.7911085
##            [,6]       [,7]
## [1,] -0.3371221 -2.4281236
## [2,] -0.7749937 -0.1214798
## [3,] -3.9802883 -2.2094892
## [4,] -0.2387796  2.2040078
## [5,] -2.2471820  2.2061703
## [6,]  1.7306970 -1.4043691
```

With standard deviations

```
nlrData2$SE
```

```
##           [,1]     [,2]     [,3]     [,4]     [,5]      [,6]      [,7]
## [1,] 0.4696158 3.647585 1.827694 2.214321 3.083912 1.3035065 0.7403814
## [2,] 0.5487080 3.914280 1.952428 2.245579 2.896472 1.2864679 0.7365580
## [3,] 0.8597807 8.854572 4.916268 3.562578 5.865719 2.5456396 1.0907289
## [4,] 0.4105454 2.885802 1.686134 1.552633 1.139284 0.6868723 0.3240578
## [5,] 0.5413763 3.286180 1.938462 1.247560 1.204180 0.6803302 0.2733233
## [6,] 0.8933475 4.988918 2.743110 1.947661 2.138584 1.1344472 0.3993421
```

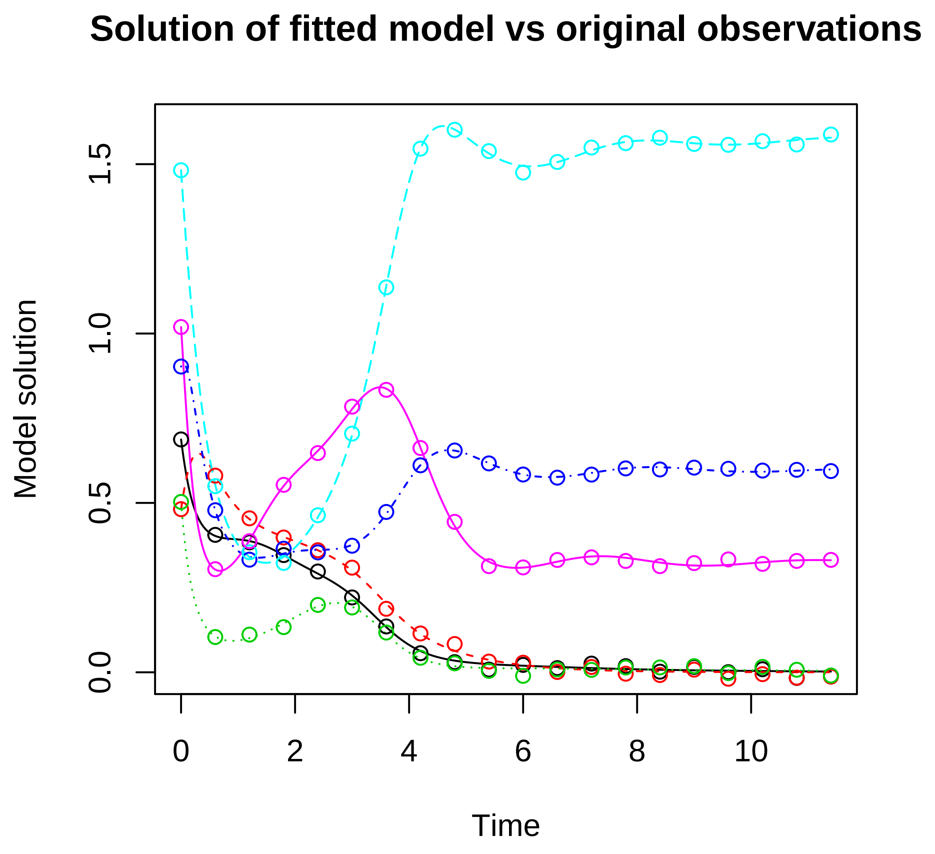
With the standard errors and degrees of freedom we can compute standard t tests with a null hypothesis claiming that the parameter value is equal to zero (i.e. no interaction). This is the output of the summary function on the regression result:

```r
summary(nlrData2)
```

```
##          Estimate     StdErr t.value   p.value    
##  [1,]  1.79223499  0.46961585  3.8164 0.0002698 ***
##  [2,] -1.14900477  0.54870800 -2.0940 0.0395064 *  
##  [3,]  5.68942535  0.85978069  6.6173 4.173e-09 ***
##  [4,]  0.08948867  0.41054537  0.2180 0.8280178    
##  [5,]  1.81274708  0.54137626  3.3484 0.0012536 ** 
##  [6,]  3.57599179  0.89334751  4.0029 0.0001416 ***
##  [7,] -1.67109050  3.64758521 -0.4581 0.6481285    
##  [8,]  4.48139180  3.91428001  1.1449 0.2557586    
##  [9,] -0.00054822  8.85457181 -0.0001 0.9999508    
## [10,]  5.50970184  2.88580155  1.9092 0.0599081 .  
## [11,] -3.49343286  3.28617960 -1.0631 0.2910312    
## [12,]  2.98246399  4.98891756  0.5978 0.5516933    
## [13,] -0.04198621  1.82769429 -0.0230 0.9817311    
## [14,] -3.63452284  1.95242783 -1.8615 0.0664351 .  
## [15,] -8.21869778  4.91626804 -1.6717 0.0985839 .  
## [16,] -5.47304844  1.68613405 -3.2459 0.0017261 ** 
## [17,] -3.30201388  1.93846243 -1.7034 0.0924712 .  
## [18,] -1.69752061  2.74310962 -0.6188 0.5378310    
## [19,]  3.53269933  2.21432123  1.5954 0.1146698    
## [20,]  2.71661123  2.24557865  1.2098 0.2300247    
## [21,]  1.05303570  3.56257752  0.2956 0.7683344    
## [22,] -4.05040893  1.55263263 -2.6087 0.0108885 *  
## [23,] -0.08410942  1.24756004 -0.0674 0.9464206    
## [24,]  0.48877777  1.94766130  0.2510 0.8025079    
## [25,] -1.49761104  3.08391217 -0.4856 0.6285976    
## [26,]  2.36320500  2.89647244  0.8159 0.4170473    
## [27,]  1.79277129  5.86571855  0.3056 0.7606965    
## [28,] -0.71401352  1.13928425 -0.6267 0.5326721    
## [29,]  1.69767687  1.20417987  1.4098 0.1625679    
## [30,] -9.79110852  2.13858372 -4.5783 1.745e-05 ***
## [31,] -0.33712206  1.30350650 -0.2586 0.7966043    
## [32,] -0.77499375  1.28646795 -0.6024 0.5486421    
## [33,] -3.98028829  2.54563956 -1.5636 0.1219667    
## [34,] -0.23877962  0.68687232 -0.3476 0.7290520    
## [35,] -2.24718204  0.68033025 -3.3031 0.0014452 ** 
## [36,]  1.73069704  1.13444722  1.5256 0.1311577    
## [37,] -2.42812362  0.74038135 -3.2796 0.0015552 ** 
## [38,] -0.12147983  0.73655803 -0.1649 0.8694265    
## [39,] -2.20948918  1.09072892 -2.0257 0.0462156 *  
## [40,]  2.20400779  0.32405784  6.8013 1.877e-09 ***
## [41,]  2.20617034  0.27332327  8.0717 6.836e-12 ***
## [42,] -1.40436911  0.39934207 -3.5167 0.0007317 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

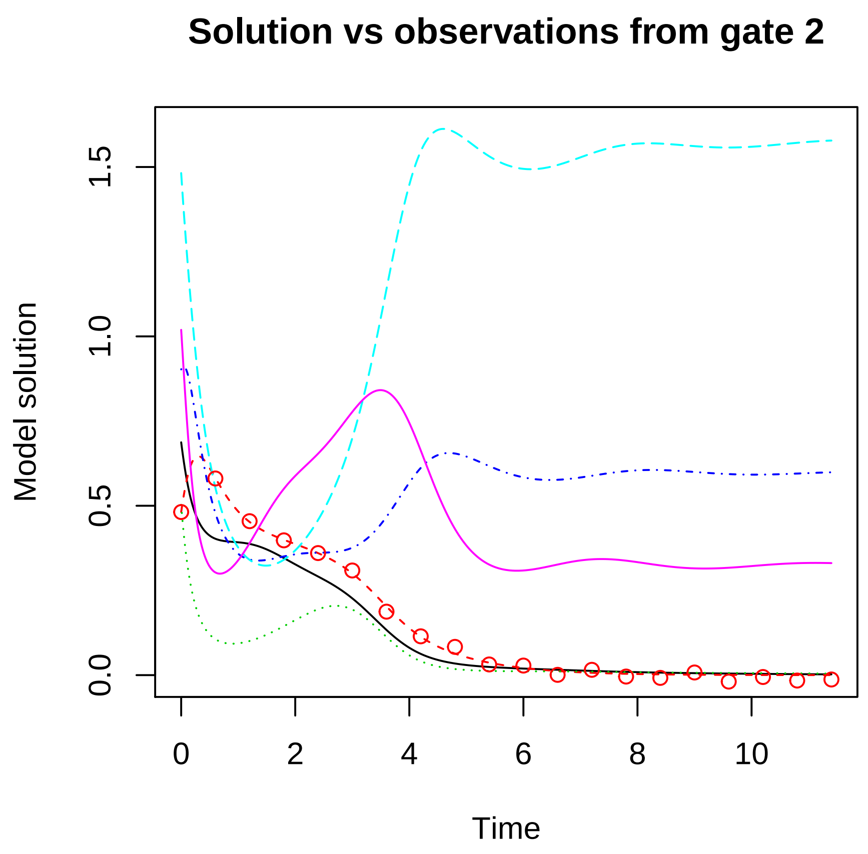
We can plot the solution of the fitted model with estimated parameters against the original observations using the points() function

```r
plot(nlrData2, type="l", main="Solution of fitted model vs original observations")
points(exampleData2)
```



The points function can also be used to compare single trajectories:

```r
plot(nlrData2, type="l", main="Solution vs observations from gate 2")
points(exampleData2, index = 2)
```



Sensitivity analysis

We can perform a sensitivity analysis on the estimated parameter set with the function sensitivityAnalysis(). This function requires the original observations saved under the name "data".

```
data <- exampleData2
ident <- sensitivityAnalysis(Parms = nlrData2$Parms)
```

This function produces two large tables, saved under the names "sens" and "coll"

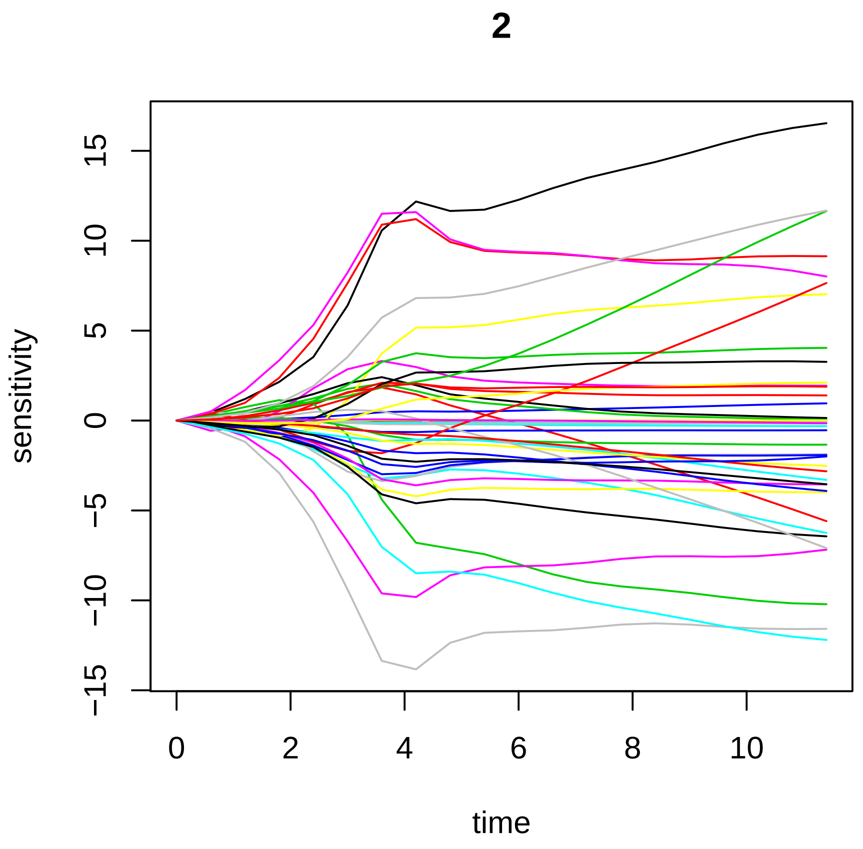
```
names(ident)
```

```
## [1] "sens" "coll"
```

Sens corresponds to a sensitivity matrix. This evaluates how much the output of the model varies by changing the value of a single parameter, in other words, how sensitive the model is to each single parameter. Parameters that have a large effect in the model output are usually easier to identify, while parameter that have little effect in the model output are often harder to identify to a precise value. This evaluation only evaluates parameters single-handedly and thus does not account for correlations and multicollinearity issues between parameters.

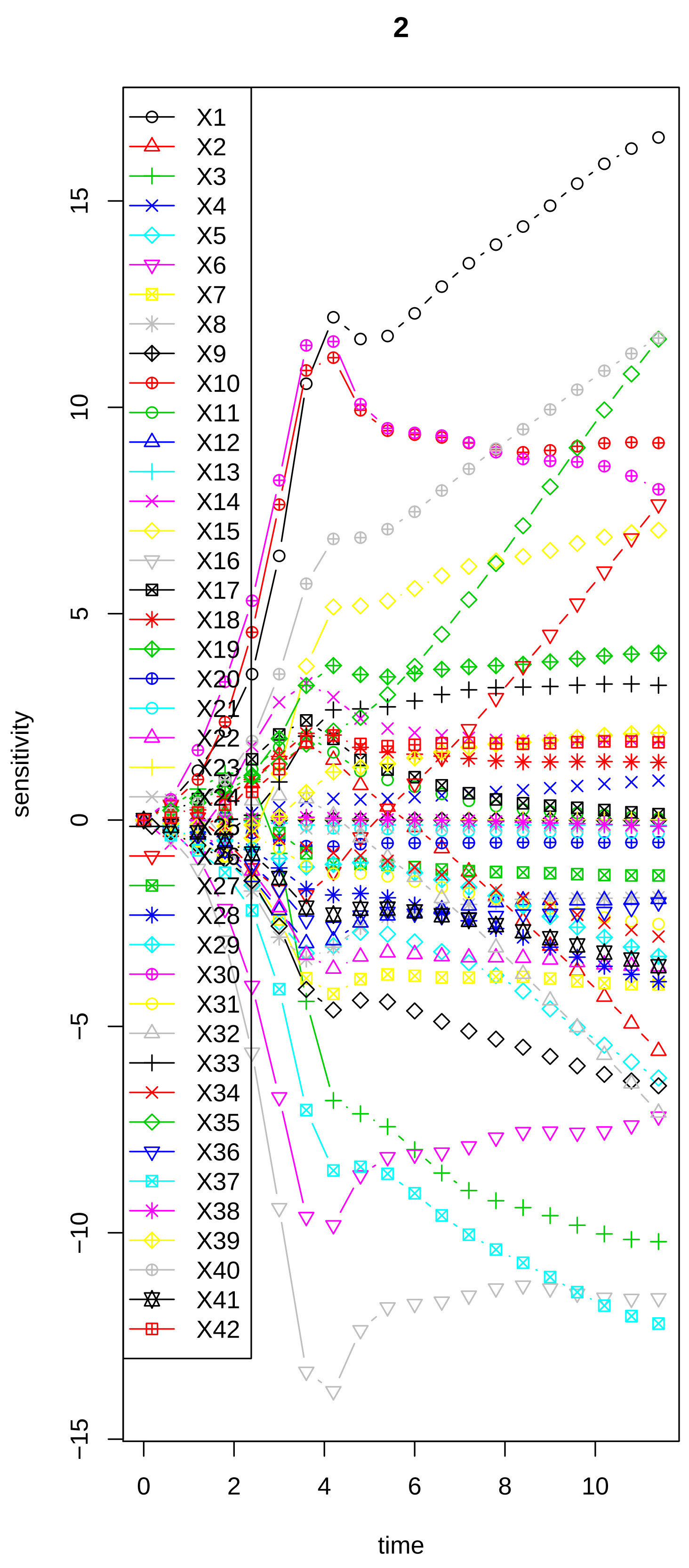
```
head(ident$sens)
```

An overview plot showing the sensitivity of the parameters for variable 2 can be generated with the plot function on ident$sens



And if we wanted to add a legend to try to identify the corresponding parameter to which sensitivity trajectory:

```
plot(ident$sens, which=2, type="b" , pch=c(1:10,1:10,1:10,1:14), col=1:44)
```


2
sensitivity
time
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
X16
X17
X18
X19
X20
X21
X22
X23
X24
X25
X26
X27
X28
X29
X30
X31
X32
X33
X34
X35
X36
X37
X38
X39
X40
X41
X42
15
10
5
0
−5
−10
−15
0
2
4
6
8
10

In this overpacked and difficult to visualize plot we see that most parameters have little effect of the output for variable 2 (all lines near the x-axis zero), and some parameters have a larger effect in the abundance of variable 2 (all lines further away from the x-axis zero, for example the gray line of parameter 16 , which correspond to β_{42} i.e. the effect of variable 4 on variable 2). Trajectories that are x-axis symmetric correspond to parameters that have a reciprocal sensitivity to the model output, meaning that they are potentialy correlated and changing the value of them simultaneously can produce a similar fitting solution.

When dealing with large systems with a lot of parameters, visualization of parameter sensitivities like above can be confusing and hard to evaluate. The main message we can retrieve is that a most parameters seem to have similar shaped trajectories of sensitivites to model output, which can be troublesome for finding unique solutions to the estimation problem.

Less sensitivity (standard: lower L2 value) means that the parameter is harder to identify and the result is more correlated to the values that other parameter obtain. This is seen in the summary statistic of the parameter matrix with each standard deviation. Note that only more sensitive parameters (parameters) are the ones with narrower standard deviation (significantly different from zero):

```
sSens <- summary(ident$sens)
sSens[order(sSens$L2, decreasing = TRUE),c("value","L2")]
```

```
##       value      L2
## 35 -2.24718 9.9e-01
## 1   1.79223 7.6e-01
## 30 -9.79111 7.4e-01
## 33 -3.98029 5.8e-01
## 40  2.20401 5.7e-01
## 37 -2.42812 5.6e-01
## 3   5.68943 5.5e-01
## 16 -5.47305 5.4e-01
## 6   3.57599 5.3e-01
## 5   1.81275 4.4e-01
## 10  5.50970 4.4e-01
## 36  1.73070 3.3e-01
## 15 -8.21870 3.2e-01
## 25 -1.49761 3.0e-01
## 29  1.69768 2.6e-01
## 41  2.20617 2.2e-01
## 28 -0.71401 2.1e-01
## 7  -1.67109 1.8e-01
## 34 -0.23878 1.8e-01
## 19  3.53270 1.7e-01
## 26  2.36321 1.6e-01
## 22 -4.05041 1.5e-01
## 32 -0.77499 1.5e-01
## 2  -1.14900 1.4e-01
## 31 -0.33712 1.4e-01
## 17 -3.30201 1.4e-01
## 8   4.48139 1.3e-01
## 14 -3.63452 1.3e-01
## 11 -3.49343 1.2e-01
## 12  2.98246 1.2e-01
## 42 -1.40437 1.1e-01
## 27  1.79277 1.0e-01
## 39 -2.20949 9.5e-02
## 18 -1.69752 8.5e-02
## 4   0.08949 4.8e-02
## 20  2.71661 3.5e-02
## 21  1.05304 1.5e-02
## 38 -0.12148 8.7e-03
## 24  0.48878 8.5e-03
## 13 -0.04199 5.5e-03
## 23 -0.08411 1.0e-03
## 9  -0.00055 1.8e-05
```

Most sensitive parameters: 35, 1, 30, 33, 40, 37, 3, 16, 6, 5 …

```
s <- cbind(1:42,summary(nlrData2))
s[order(s[,5]),]
```

```
##                Estimate    StdErr       t.value      p.value
##  [1,] 41  2.2061703360 0.2733233  8.0716520129 6.835962e-12
##  [2,] 40  2.2040077859 0.3240578  6.8012790095 1.877011e-09
##  [3,]  3  5.6894253515 0.8597807  6.6172983502 4.173052e-09
##  [4,] 30 -9.7911085211 2.1385837 -4.5783143524 1.745095e-05
##  [5,]  6  3.5759917927 0.8933475  4.0029123744 1.415518e-04
##  [6,]  1  1.7922349925 0.4696158  3.8163852354 2.697588e-04
##  [7,] 42 -1.4043691082 0.3993421 -3.5167071615 7.316644e-04
##  [8,]  5  1.8127470798 0.5413763  3.3484051899 1.253632e-03
##  [9,] 35 -2.2471820363 0.6803302 -3.3030753039 1.445232e-03
## [10,] 37 -2.4281236237 0.7403814 -3.2795580464 1.555161e-03
## [11,] 16 -5.4730484445 1.6861340 -3.2459153866 1.726125e-03
## [12,] 22 -4.0504089278 1.5526326 -2.6087361921 1.088848e-02
## [13,]  2 -1.1490047656 0.5487080 -2.0940186289 3.950636e-02
## [14,] 39 -2.2094891841 1.0907289 -2.0256996439 4.621558e-02
## [15,] 10  5.5097018443 2.8858016  1.9092448804 5.990811e-02
## [16,] 14 -3.6345228407 1.9524278 -1.8615401719 6.643507e-02
## [17,] 17 -3.3020138850 1.9384624 -1.7034190816 9.247116e-02
## [18,] 15 -8.2186977797 4.9162680 -1.6717350855 9.858394e-02
## [19,] 19  3.5326993267 2.2143212  1.5953870106 1.146698e-01
## [20,] 33 -3.9802882882 2.5456396 -1.5635710359 1.219667e-01
## [21,] 36  1.7306970414 1.1344472  1.5255862186 1.311577e-01
## [22,] 29  1.6976768727 1.2041799  1.4098200039 1.625679e-01
## [23,] 20  2.7166112334 2.2455786  1.2097600031 2.300247e-01
## [24,]  8  4.4813918003 3.9142800  1.1448827839 2.557586e-01
## [25,] 11 -3.4934328640 3.2861796 -1.0630681475 2.910312e-01
## [26,] 26  2.3632050028 2.8964724  0.8158907254 4.170473e-01
## [27,] 28 -0.7140135230 1.1392842 -0.6267211407 5.326721e-01
## [28,] 18 -1.6975206079 2.7431096 -0.6188307588 5.378310e-01
## [29,] 32 -0.7749937479 1.2864679 -0.6024197876 5.486421e-01
## [30,] 12  2.9824639899 4.9889176  0.5978178541 5.516933e-01
## [31,] 25 -1.4976110385 3.0839122 -0.4856205221 6.285976e-01
## [32,]  7 -1.6710904970 3.6475852 -0.4581361096 6.481285e-01
## [33,] 34 -0.2387796205 0.6868723 -0.3476331994 7.290520e-01
## [34,] 27  1.7927712948 5.8657186  0.3056354100 7.606965e-01
## [35,] 21  1.0530356982 3.5625775  0.2955825360 7.683344e-01
## [36,] 31 -0.3371220579 1.3035065 -0.2586270623 7.966043e-01
## [37,] 24  0.4887777705 1.9476613  0.2509562466 8.025079e-01
## [38,]  4  0.0894886708 0.4105454  0.2179751075 8.280178e-01
## [39,] 38 -0.1214798279 0.7365580 -0.1649290666 8.694265e-01
## [40,] 23 -0.0841094158 1.2475600 -0.0674191328 9.464206e-01
## [41,] 13 -0.0419862101 1.8276943 -0.0229722280 9.817311e-01
## [42,]  9 -0.0005482157 8.8545718 -0.0000619133 9.999508e-01
```

Most certain parameter values: 41, 40, 3, 30, 6, 1, 42, 5, 35, 37 . . .

In these two list, eight out of ten parameters coincide (40, 3, 30, 6, 1, 5, 35, 37)

Parameter collinearities

In order to evaluate the parameter collinearities and problems which may arise from similar sensitivity trajectories, we can take a look at the second output from the sensitivity() function, namely the "coll" table:

```
head(ident$coll)
```

```
##   X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15 X16 X17 X18 X19 X20
## 1  1  1  0  0  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0   0
## 2  1  0  1  0  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0   0
## 3  1  0  0  1  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0   0
## 4  1  0  0  0  1  0  0  0  0   0   0   0   0   0   0   0   0   0   0   0
## 5  1  0  0  0  0  1  0  0  0   0   0   0   0   0   0   0   0   0   0   0
## 6  1  0  0  0  0  0  1  0  0   0   0   0   0   0   0   0   0   0   0   0
##   X21 X22 X23 X24 X25 X26 X27 X28 X29 X30 X31 X32 X33 X34 X35 X36 X37 X38
## 1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##   X39 X40 X41 X42 N collinearity
## 1   0   0   0   0 2          1.2
## 2   0   0   0   0 2          1.5
## 3   0   0   0   0 2          2.9
## 4   0   0   0   0 2          1.7
## 5   0   0   0   0 2          4.5
## 6   0   0   0   0 2          2.7
```

In contrast to the "sens" table, here groups of parameters are analyzed simultaneously to evalute the "grade of linear dependency" between them, in other words, how a similar output of the model (and thus solution to the fitting problem) can be achieved using different values for the parameters involved, and at which precision level.

Each column corresponds to a parameter in the model which are either present in the subset to evaluate (= 1) or absent (= 0). The second to last column named "N" displays how large the subset is (i.e. how many parameters are contained in the subset to evaluate the collinearity). The last column named "collinearity" displays the collinearity index for the subset of selected parameters. For example: Row 6 evaluates the subset of size 2 containing parameters 1 and 7, which have a collinearity index of "2.7". The collinearity index shows the precision we can approximate the solution given by the model, by simultaneously changing the parameters in the subset. In this case it means that we can compensate a change in the value of parameter 1 by changing the value of parameter 7, and doing this appropiately we can reach a fraction of 1/2.7 of the original model solution.

The higher the collinearity index is, the nearer we can approximate a model solution. Two linear dependent parameter (for example α and β in the model $y = x(\alpha + \beta)$ would have a collinearity index of infinite, as we can achieve exactly the same solution with different values for α and β). As a rule of thumb, collinearity indexes of over 30 are considered troublesome for estimation algorithms.

The function gLVnonlinearRegression() attempts to estimate all parameters simultaneously. Therefore, relevant for this is the collinearity index for the full set of parameters:

```
ident$coll[ident$coll[,"N"]== length(nlrData2$Parms),]
```

```
##   X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15 X16 X17 X18 X19 X20
## 1  1  1  1  1  1  1  1  1  1   1   1   1   1   1   1   1   1   1   1   1
##   X21 X22 X23 X24 X25 X26 X27 X28 X29 X30 X31 X32 X33 X34 X35 X36 X37 X38
## 1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##   X39 X40 X41 X42  N collinearity
## 1   1   1   1   1 42         2519
```

We can retrieve this value

```
coll_index <- ident$coll[ident$coll[,"N"]==42,"collinearity"]
paste0("Collinearity index for the complete set of 42 parameters: ", coll_index)
```

```
## [1] "Collinearity index for the complete set of 42 parameters: 2519.48813036233"
```

Thus the least level of precision required for a unique solution with this parameterization is 1/coll_index $\sim 3.9 * 10^{-4}$. This can be compared to the level of precision that the model solution fits the original data, described by the standard deviation of the residuals left between them:

Level of precision achieved by the solution:

```
nlrData2$residual_SD
```

```
## [1] 0.008185737
```

Level of precision achievable by different parameterizations (different numeric solutions for the whole parameter matrix):

```
1/coll_index
```

```
## [1] 0.000396906
```

Is the solution unique?

```
nlrData2$residual_SD < 1/coll_index
```

```
## [1] FALSE
```

Linear regression

A linearized algebraic and discrete version of the model equations is possible through decoupling of the variables

$$log(x_i(t+1)) = \alpha_i + \sum \beta_{ij} x_j(t)$$

An estimate for the parameters in this form is quickly and easy achievable through linear regression

```
lrData1 <- gLVlinearRegression(data = exampleData1)
```

We can inspect the solution just like we did before with the results from the nonlinear regression.

In contrast to the nonlinear regression, the parameter for the linear model are a unique solution as the sum of squared error has a global optimum.

Due to errors introduced in the discretization, slope approximation and due to the neglection of the time-correlation between residuals, the solution to the linear regression is distorted and only accurate in very small and simple systems. Nevertheless, it is often useful to compute the linear regression parameter estimates to provide them as alternative starting values (parms0 argument) for the nonlinear regression when it fails to converge from the zero start vector

```
nlrData1 <- gLVnonlinearRegression(data = exampleData1, parms0 = lrData1$Parms)
```

Often in large systems the linear regression returns very high parameter estimates, which are not useful as start vectors for the nonlinear regression nor have realistic biological interpretations. For these cases it is useful to perform shrinkage of the parameter matrix to avoid numerical instabilites. This can be done through regularization:

```
lrData2 <- gLVlinearRegression(data = exampleData2)
lrData2$Parms
```

```
##           [,1]       [,2]        [,3]       [,4]       [,5]       [,6]
## [1,] 2.1875364  5.9237858  -9.3629157  -3.211697   5.586053 -3.3372798
## [2,] 3.3160375  0.3180209   0.1691785  18.732517 -14.850432  4.6608749
## [3,] 3.7788314 10.3455179 -12.4982440 -10.370104   4.831088 -4.1910615
## [4,] 0.9519292  1.2810992  -0.6250050  -1.056990  -5.051056  1.1736605
## [5,] 2.0550439 -4.1894790  -2.1775565   1.249551   1.953121 -2.3153159
## [6,] 2.6369452 -1.2376828   0.6139137   2.461730  -4.044600  0.2757798
##            [,7]
## [1,] -1.2709429
## [2,] -5.9353681
## [3,] -0.5140691
## [4,]  0.7444985
## [5,]  1.2413160
## [6,] -2.0581190
```

```
## [1] "range of estimated parameters: [-14.9 , 18.7]"
```

```
lrData2_regularized <- gLVlinearRegression(data = exampleData2, regularization = TRUE, alpha = 0.5)
lrData2_regularized$Parms
```

```
##             [,1]       [,2]       [,3]       [,4]      [,5]        [,6]
## [1,] -0.3697540  0.0000000  0.0000000  0.0000000  0.000000  0.00000000
## [2,] -0.3175587  0.0000000  0.0000000  0.0000000  0.000000  0.00000000
## [3,] -0.3464287  0.0000000  0.0000000  0.0000000  0.000000  0.00000000
## [4,]  0.9480365  0.1185183 -0.5437521 -0.4812906 -3.591136  0.62276827
## [5,]  2.0509603 -3.8194904 -2.0854705  1.1446155  1.278050 -2.04862558
## [6,]  2.6114013 -1.0286878  0.2804677  1.9775898 -3.544639  0.07236599
##             [,7]
## [1,]  0.0000000
## [2,]  0.0000000
## [3,]  0.0000000
## [4,]  0.7141698
## [5,]  1.2151594
## [6,] -1.9216549

## [1] "range of estimated parameters after regularization: [-3.8 , 2.6]"
```

Note that regularization is computed through an optimization method involving cross-validation with random partitions of the data, so the solution is not unique and likely to change in different runs.

Comparing different Network Structures

In order to compare explicit network structures based on prior knowledge of the system or to test for different solutions, the functions networkStructures() and compareStructures() can be used.

A network structure or topology is defined here as the qualitative values that the edges connecting the nodes take, in this case differentiating between possitive "+" , negative "-", and nonexistent interactions "0". Because the optimization algorithms never estimate a parameter value to be exactly zero, the results of a fitted model will always contain just "+" or "-" edges. Further inspection of the estimated values can give hints on values that approximate zero.

The codification of a network structure is then a vector derived from the parameter matrix.

$$\begin{pmatrix} \alpha_1 & \beta_{11} & \beta_{12} \\ \alpha_2 & \beta_{21} & \beta_{22} \end{pmatrix}$$

As a vector will look like the following:

$$(\alpha_1, \alpha_2, \beta_{11}, \beta_{21}, \beta_{21}, , \beta_{22})$$

For example, the following parameter matrix from "exampleData1":

```r
print(exampleData1$Parms)
```

```
##          growth
## [1,]  1.5334121 -1.653507 -2.652309
## [2,] -0.7558098  5.358952 -1.692742
```

Has the following structure:" + - - + - - "

The function **networkStructures()** generates either a complete list containing all possible network topologies for n species, or generates a list based on a pre-specified list of network structures to test.

```r
## generate all possible structures for a network containing 2 species:
s <- networkStructures(n = 2)
head(s$bounds)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] "-"  "-"  "-"  "-"  "-"  "-"
## [2,] "-"  "-"  "-"  "-"  "-"  "+"
## [3,] "-"  "-"  "-"  "-"  "+"  "-"
## [4,] "-"  "-"  "-"  "-"  "+"  "+"
## [5,] "-"  "-"  "-"  "+"  "-"  "-"
## [6,] "-"  "-"  "-"  "+"  "-"  "+"
```

```r
head(s$ub)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0   10
## [3,]    0    0    0    0   10    0
## [4,]    0    0    0    0   10   10
## [5,]    0    0    0   10    0    0
## [6,]    0    0    0   10    0   10
```

```r
head(s$lb)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  -10  -10  -10  -10  -10  -10
## [2,]  -10  -10  -10  -10  -10    0
## [3,]  -10  -10  -10  -10    0  -10
## [4,]  -10  -10  -10  -10    0    0
## [5,]  -10  -10  -10    0  -10  -10
## [6,]  -10  -10  -10    0  -10    0
```

After having generated the list of the structures to test, and their respective lower and upper bounds for the parameter estimation process (described by s$ub and s$lb), we can give this as an input to the function compareStructures().

iterations Number of times the regression algorithm should be performed on each structure. The default is 1 for only one parameter estimation run per structure. If set to integers x > 1 the algorithm runs x times and keeps the best solution according to the goodness of fit. The number of different solutions found and the number of times the optimal solution was achieved is kept in the regression attributes in order to evaluate how many local maximas were found, and how certain the user can be that the kept optimal could approximate a global optimum point.

Having no prior knowledge on characteristics of this system which could allow us to test qualititative constraints on specific interactions, we can first test all 64 possible structures and see which ones perform better for explaining the data. We set iteration = 2 to perform two parameter estimation steps on each structure each time with different randomly generated starting parameters (obeying the qualitative structure of the network to test). In general it is not practical to perform these evaluations, as it takes a lot of time.

In practice this takes 28 minutes to complete (DELL - OS: Windows7 - intel core i3-2330M CPU 2.20GHz)

```
comparisonAllStructuresData1 <- compareStructures(data = exampleData1, structures = s, iterations = 2)
```

The result contains a list with all network structures tested, and their goodness of fit described by the sum of squared residuals of the best solution, and the estimated variance of the data stochasticity:

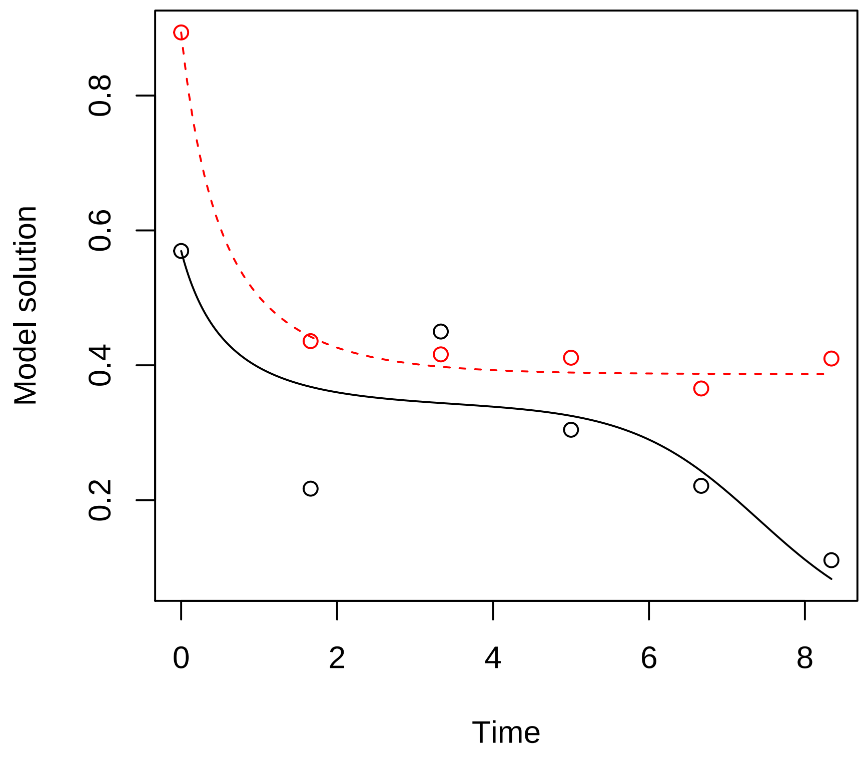
```
comparisonAllStructuresData1$networks
```

```
##        Index Network   SSR    SD
## 1  Original  +--+--      0   0.1
## 2         29  -+++-- 0.038 0.058
## 3         53  ++-+-- 0.051 0.068
## 4         21  -+-+-- 0.063 0.076
## 5         61  ++++-- 0.063 0.075
## 6         19  -+--+- 0.065 0.077
## 7         37  +--+-- 0.065 0.077
## 8         51  ++--+- 0.065 0.077
## 9         49  ++---- 0.066 0.077
## 10        23  -+-++- 0.067 0.078
## 11         4  ----++ 0.105 0.098
## 12         1  ------ 0.118 0.103
## 13        35  +---+- 0.118 0.103
## 14        36  +---++ 0.118 0.103
## 15         3  ----+- 0.119 0.104
## 16         5  ---+-- 0.119 0.104
## 17        11  --+-+-  0.12 0.105
## 18         2  -----+ 0.349 0.146
## 19        17  -+---- 0.405 0.155
## 20        25  -++--- 0.406 0.155
## 21         9  --+--- 0.425 0.156
## 22        13  --++-- 0.461  0.17
## 23        33  +----- 0.898 0.224
## 24         6  ---+-+  <NA>  <NA>
## 25         7  ---++-  <NA>  <NA>
## 26         8  ---+++  <NA>  <NA>
## 27        10  --+--+  <NA>  <NA>
## 28        12  --+-++  <NA>  <NA>
## 29        14  --++-+  <NA>  <NA>
## 30        15  --+++-  <NA>  <NA>
## 31        16  --++++  <NA>  <NA>
## 32        18  -+---+  <NA>  <NA>
## 33        20  -+--++  <NA>  <NA>
## 34        22  -+-+-+  <NA>  <NA>
## 35        24  -+-+++  <NA>  <NA>
## 36        26  -++--+  <NA>  <NA>
## 37        27  -++-+-  <NA>  <NA>
## 38        28  -++-++  <NA>  <NA>
## 39        30  -+++-+  <NA>  <NA>
## 40        31  -++++-  <NA>  <NA>
## 41        32  -+++++  <NA>  <NA>
## 42        34  +----+  <NA>  <NA>
## 43        38  +--+-+  <NA>  <NA>
## 44        39  +--++-  <NA>  <NA>
## 45        40  +--+++  <NA>  <NA>
## 46        41  +-+---  <NA>  <NA>
## 47        42  +-+--+  <NA>  <NA>
## 48        43  +-+-+-  <NA>  <NA>
## 49        44  +-+-++  <NA>  <NA>
## 50        45  +-++--  <NA>  <NA>
## 51        46  +-++-+  <NA>  <NA>
## 52        47  +-+++-  <NA>  <NA>
## 53        48  +-++++  <NA>  <NA>
## 54        50  ++---+  <NA>  <NA>
## 55        52  ++--++  <NA>  <NA>
## 56        54  ++-+-+  <NA>  <NA>
## 57        55  ++-++-  <NA>  <NA>
## 58        56  ++-+++  <NA>  <NA>
## 59        57  +++---  <NA>  <NA>
## 60        58  +++--+  <NA>  <NA>
## 61        59  +++-+-  <NA>  <NA>
## 62        60  +++-++  <NA>  <NA>
```

First we see that most networks (in total 38) failed to achieve a viable solution for the observations on two tries each. This could either mean that the starting vectors on each try generated too unstable numeric solution to allow for convergence, or that the constraints on the parameters just do not allow any solution which doesnt exponentially grow to infinite for the given timepoints (this is likely to be the reason behind the failing of network structures with a lot of "+" parameters which result in uncontrolled exponentially growing variables).

We can inspect the detailed result from the nonlinear regression by giving the index of the network in question to the saved runs in $runs:

```r
plot(comparisonAllStructuresData1$runs[[29]], type="l")
points(exampleData1)
```



The best 10 networks:

```
as.character(comparisonAllStructuresData1$networks[1:10,"Network"])
```

```
##  [1] "+--+--" "-+++--" "++-+--" "-+-+--" "++++--" "-+--+-" "+--+--"
##  [8] "++--+-" "++----" "-+-++-"
```

Now we use the networkStructures to generate a list of networks to test based directly on two structures given by ourselves. Note that the structures must contain either the characters "+" or "-" and be as long as the number of parameters to estimate.

```
structures4testing <- networkStructures(testStructures = c("++++--","+--+--"))
structures4testing
```

```
## $ub
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   10   10   10   10    0    0
## [2,]   10    0    0   10    0    0
##
## $lb
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0  -10  -10
## [2,]    0  -10  -10    0  -10  -10
```

```
##
## $bounds
##      [,1]
## [1,] "++++--"
## [2,] "+--+--"
```

```
comparisonTwoStructuresData1 <- compareStructures(data = exampleData1, structures = structures4testing,
```

```
comparisonTwoStructuresData1$networks
```

```
##      Index Network   SSR    SD
## 1 Original  +--+--     0   0.1
## 2        2  +--+-- 0.065 0.077
## 3        1  ++++-- 0.066 0.078
```

Here we see how even in a noisy system, with a good prior guess on proper data structure we can focus the evaluation on a subset of structures and even find better solutions with alternative structures which might explain the data better. In this case the “hunch”" for testing the structure “+ - - + - -” came from knowing the original data structure, but this can come from knowledge from biological information, previous literature research, etc. Testing of different hypothesis is a crucial step when using fitted models to noisy data, and for presentation and further usage of the results.