
# Tutorial on the SBPOP Package

*Efficient Support for Model Based Drug Development:*
*From Mechanistic Models to Complex Trial Simulation*

Henning Schmidt

# Tutorial Outline

- **General introduction to the SBPOP Package**

- **Model definition and simulation**
  - ODE based models
  - Biochemical reaction equation based models
  - Import and export of models (SBML, etc.)
  - High performance computation via MEX simulation functions
  - Simple simulation and analysis of models
  - Commenting of models

# Tutorial Outline

- **Systems Biology relevant topics**
  - **Model analysis**
    - Steady-state analysis and stability
    - Moiety conservations and reduction
    - Parameter sensitivity analysis (steady-state & oscillating systems, MCA, local & global)
    - Localization of complex behaviors
  - **Definition of experiments and measurement data**
    - Excel and CSV measurement representation
    - Experiment descriptions and merging with models
    - Import and export measurements and experiments
  - **Projects, parameter estimation, model reduction**
    - Projects
    - Parameter estimation / manual tuning / parameter fit analysis / parameter identifiability analysis
    - Model reduction
    - Simulation of projects
    - Commenting of projects

Slides with this background color are related to Systems Biology relevant topics

# Tutorial Outline

- **Pharmacometrics / Systems Pharmacology relevant topics**
  - **Model and dosing description**
    - Needed changes in SBmodels to allow for simulation of dosing scenarios
    - Simulation of dosing scenarios
  - **Datasets**
    - Import, export, conversion
    - Analysis (plotting capabilities)
  - **Interface to Monolix**
  - **PopPK/PD workflow**
  - **Clinical trial simulations**
  - **More complex modeling**
    - PBPK
    - Antibody modeling

Independent of systems biology relevant topics

# Tutorial Info

**In large parts you will have the opportunity to get hands-on-experience**

```
>> installSBPOPpackage
```

Commands shown in these boxes should be entered on the MATLAB command line, during the tutorial

Text shown in these boxes should be entered where appropriate (will become clear later)

```
********** MODEL NAME
Simple model
********** MODEL STATES
d/dt(A) = -R
d/dt(B) = R
A(0) = 1
B(0) = 0
********** MODEL PARAMETERS
k1 = 0.5
********** MODEL REACTIONS
R = k1*A
```

# Tutorial Goal: „You should be able to“

- **Systems Biology relevant topics**
  - Set up your own parameter estimation project
    - Model description
    - Measurement data
    - Experiment descriptions
  - Perform parameter estimation, identifiability analysis, etc.
  - Analyze the resulting model

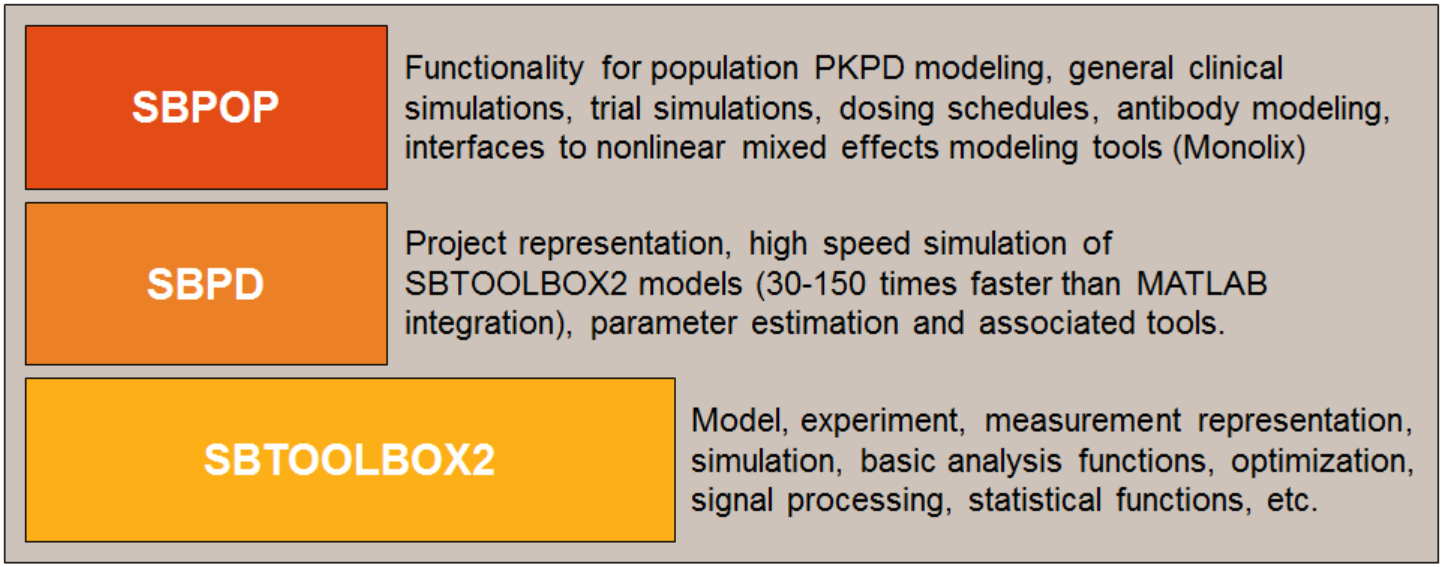
- **Pharmacometrics / Systems Pharmacology relevant topics**
  - Define arbitrary ODE based traditional and/or mechanistic, PK, PKPD, PBPK models
  - Perform
    - NLME estimation of parameters
    - Clinical trial simulations

# Tutorial Outline

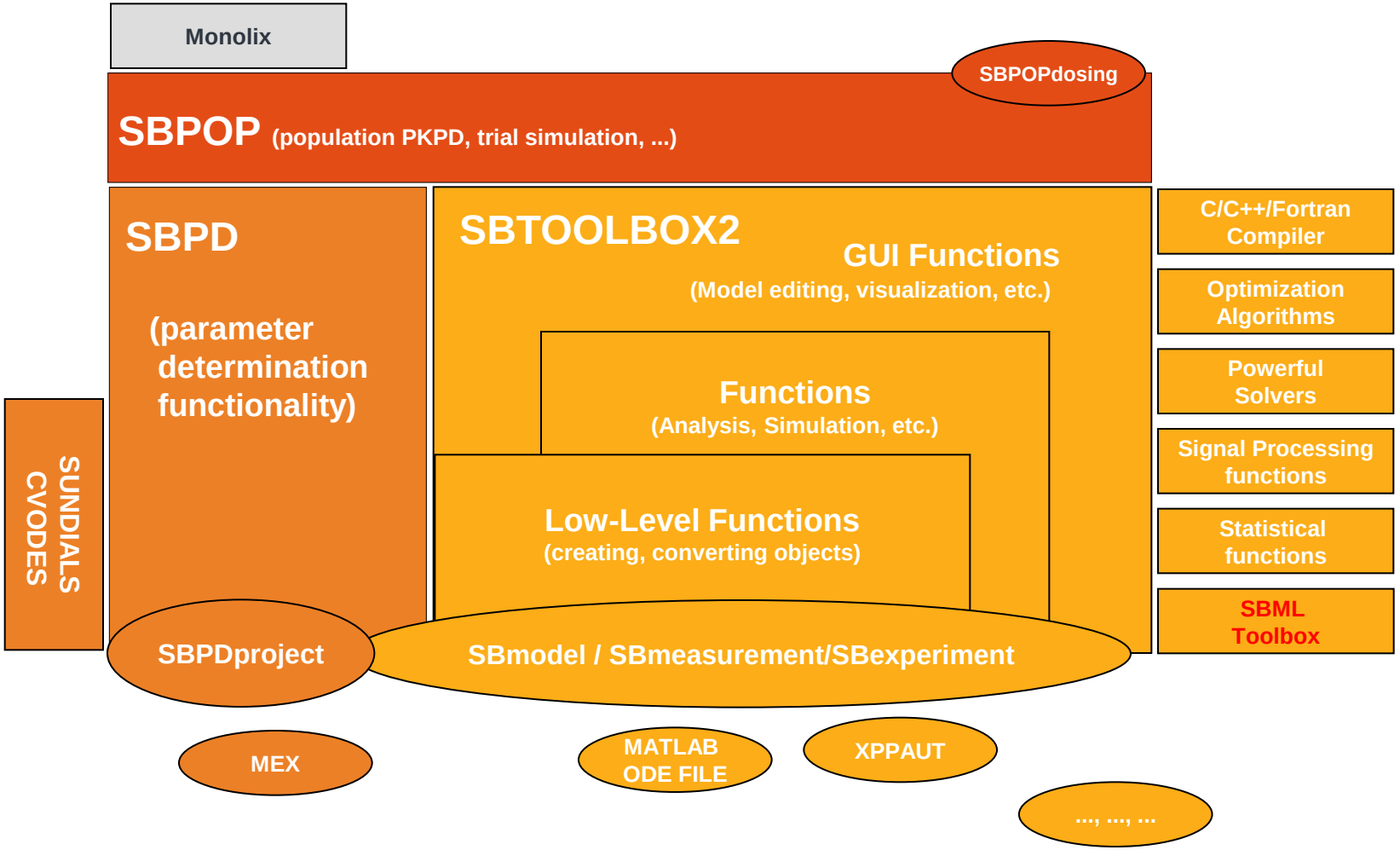
- General introduction to the SBPOP Package
- Model definition and simulation
- Systems Biology relevant topics
- Pharmacometrics / Systems Pharmacology relevant topics

# Introduction to the "SBPOP Package"

- The SBPOP Package is open source and published under a GNU General Public License
- The SBPOP Package consists of 3 parts
- Each part builds on the functionality provided by the packages below
- The whole is based on MATLAB (www.mathworks.com)



# Modular Design of the SBPOP Package



# Requirements

- **MATLAB R2010a (or later)**
  - Model reduction methods require the Symbolic Toolbox
  - The ”pharmacometrics” part requires the Statistics Toolbox

- **Optional 3rd party software**
  - Monolix Version 4.2.1 (or later) (http://www.lixoft.org)
  - SSm Global Optimization Toolbox (http://www.iim.csic.es/~gingproc/ssmGO.html)
  - SBML Toolbox (for SBML import and export)
    - Only needed for Unix/Linux/Mac
    - For Windows it is included in the distribution of the SBPOP Package

# Where to get the SBPOP Package from?

## *Novartis Version*

- The SBPOP Package is available on Subversion in the following repository

```
http://chbslx0132.eu.novartis.net/svn/SBPOP/trunk/SBPOP PACKAGE
```

- You can download it to MODESIM by the following command

```
svn export "http://chbslx0132.eu.novartis.net/svn/SBPOP/trunk/SBPOP PACKAGE"
```

- If you only want to try out SBPOP, please create a "TOOLS" folder in your MODESIM home folder and run the above command from within this "TOOLS" folder
- If you want to use SBPOP on a clinical project, please run the above command from within the GPSII location where you are working

- After export of SBPOP please rename the "SBPOP PACKAGE" folder to "SBPOP_Rev_XYZ", where "XYZ" is the revision number that is shown after you have run the above command
  - This helps to keep track of what revision of SBPOP you are using

# Where to get the SBPOP Package from?

## *General Version*

- The SBPOP Package should have been provided to you prior to this tutorial as a zip file “SBPOP_PACKAGE.zip”

- Unzip the file to a location of your choice

# How to Install the SBPOP Package?

- Start MATLAB
- Change into the "SBPOP PACKAGE" folder
- Execute the "installSBPOPpackageInitial.m" script
  - You only need to do this ONCE

- If your computer system does not allow MATLAB path settings to be stored then, each time you start MATLAB, you need to execute the "installSBPOPpackage.m" script to add SBPOP to the MATLAB path
  - In this case you might want consider the use of a "startup.m" script

# Installation of optional 3rd party software

- **Monolix:** Please have a look at www.lixoft.org

```
% Add Monolix to the MATLAB path
cd('/CHBS/apps/Monolix/latest/Matlab/matlab')
addpath(genpath(pwd));
run_init();
```

- **SBML Toolbox, SSm Toolbox:** Please follow the download and installation instructions on www.sbtoolbox2.org

# Toolbox' Documentation

- **MATLAB style help**

```
>> help SBTOOLBOX2
>> help SBPD
>> help SBPOP

>> help SBsimulate
>> help SBPDsimulate
>> help SBPOPcreateMONOLIXproject
```

- Documentation of SBTOOLBOX2 and SBPD can be found on the webpage www.sbtoolbox2.org

- Detailed documentation of all functions

- Examples to almost all functions

# Tutorial Outline

- **General introduction to the SBPOP Package**
- **Model definition and simulation**
  - ODE based models
  - Biochemical reaction equation based models
  - Import and export of models
  - High performance computation via MEX simulation functions
  - Simple simulation and analysis of models
  - Commenting of models
- **Systems Biology relevant topics**
- **Pharmacometrics / Systems Pharmacology relevant topics**

- ODE based models

# SBmodel

# A First Model

- Creating a first model

A —R—> B  R = k1*A

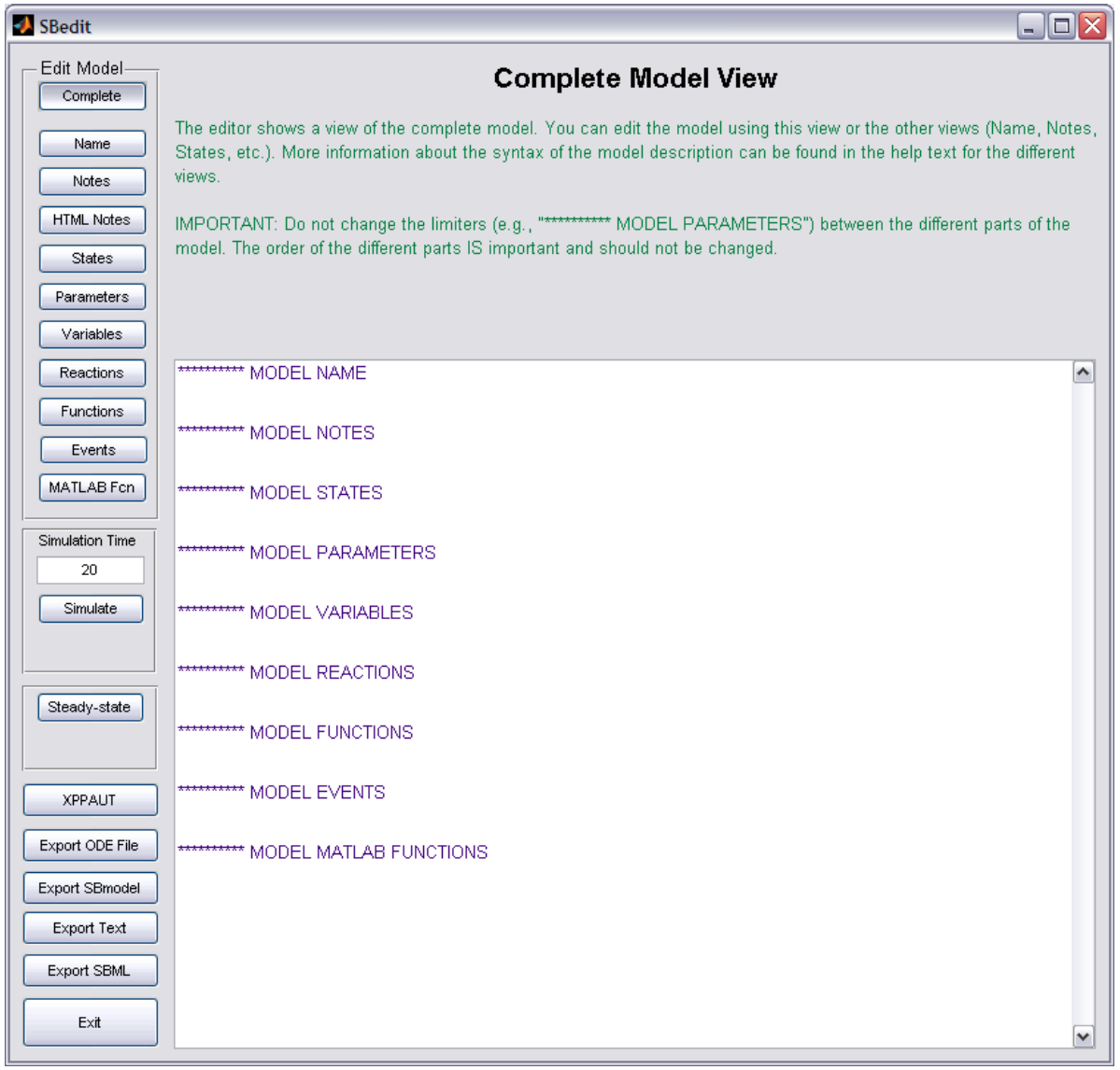
A(0) = 1, B(0) = 0  k1 = 0.5

```
>> model = SBmodel()          % creating empty model
>> model = SBedit(model)      % editing the model
```

```
>> model = SBedit()           % starting the editor with an empty model
```

# SBedit

- Graphical User Interface allowing to edit models
- Click on the ”Edit Model” buttons
- **Each view provides a help text about the model syntax**
- The limiters are important – do not change them



# Simple ODE Model (States, Parameters, Reactions)

- Enter the following information (keep all non-used limiters)

```
********** MODEL NAME
Simple model
********** MODEL STATES
d/dt(A) = -R
d/dt(B) = R
A(0) = 1
B(0) = 0
********** MODEL PARAMETERS
k1 = 0.5
********** MODEL REACTIONS
R = k1*A
```

A –R–> B

A(0) = 1
B(0) = 0

k1 = 0.5

R = k1*A

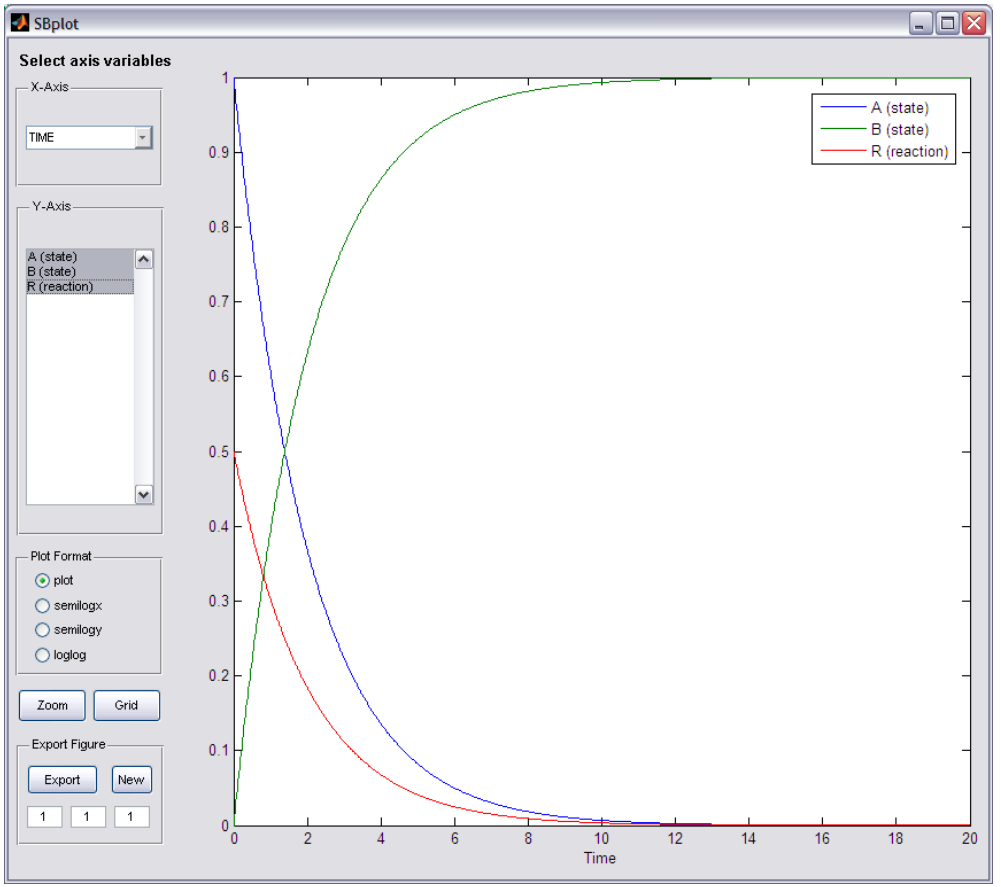
Click ”Simulate”


henning.schmidt@novartis.com

# Simulate Simple Model

- **SBplot**
  Graphical User Interface
  for the display of time series
  type of data

- Play around with the features
  of the plotting window





henning.schmidt@novartis.com

# Model Functions

- Model functions can be used to define often recurring calculations

```
********** MODEL STATES
d/dt(A) = -R
d/dt(B) = R
A(0) = 1
B(0) = 0
********** MODEL PARAMETERS
k1 = 0.5
********** MODEL REACTIONS
R = k1*f(A)
********** MODEL FUNCTIONS
f(x) = x^3
```

A –R–> B

A(0) = 1
B(0) = 0

k1 = 0.5

R = k1***f(A)**

**f(x) = x^3**

Click ”Simulate”


henning.schmidt@novartis.com

# Model Events

- Model events can be used to define discrete state events

```
********** MODEL STATES
d/dt(A) = -R
d/dt(B) = R
A(0) = 1
B(0) = 0
********** MODEL PARAMETERS
k1 = 0.5
********** MODEL REACTIONS
R = k1*f(A)
********** MODEL FUNCTIONS
f(x) = x^3
```

```
********** MODEL EVENTS
event = lt(A,0.3), A, 1, B, 0
```

**If A becomes less than 0.3 then reset A to 1 and B to 0**

Click ”Simulate”

# Model Variables

- Simulation of these two models gives exactly the same results

```
********** MODEL NAME
Simple model
********** MODEL STATES
d/dt(A) = -R
d/dt(B) = R
A(0) = 1
B(0) = 0
********** MODEL PARAMETERS
k1 = 0.5
********** MODEL REACTIONS
R = k1*A
```

```
********** MODEL NAME
Simple model
********** MODEL STATES
d/dt(A) = -R
d/dt(B) = R
A(0) = 1
B(0) = 0
********** MODEL PARAMETERS
k1 = 0.5
********** MODEL VARIABLES
R = k1*A
```

So what is the difference?

# Model Reactions vs. Model Variables

- Reaction rates should be defined under MODEL REACTIONS
- Variables for intermediate calculations (or for monitoring) should be defined under MODEL VARIABLES
- The only cases where it matters for the toolbox is when
  - Determining the stoichiometric matrix
  - Exporting the model to SBML

# Command Line Simulation

- Clicking ”Exit” in the SBedit GUI returns the model to the workspace

```
>> model = SBedit()
SBmodel
=======
Name:  Simple Model
Number States:                2
Number Variables:             0
Number Parameters:            1
Number Reactions:             1
Number Functions:             1
Number Events:                1

>> SBsimulate(model,50)       % simulation over 50 time units
```

# Command Line Simulation, continued

```
>> output = SBsimulate(model,50)
output =
              time: [50x1 double]
            states: {'A'  'B'}
       statevalues: [50x2 double]
         variables: {}
    variablevalues: []
         reactions: {'R'}
    reactionvalues: [50x1 double]


>> help SBsimulate
```

- Array specification of TEXT SBmodels

# Array specification of TEXT SBmodels

- Consider a system that can be described by the following set of differential equations:

$$d/dt\ x[n] = kon*x[1]*x[n-1] + koff*x[n+1] - (kon*x[1]+koff)*x[n],\ n=1\ldots N$$

  - Example:
    - Model for the length distribution of actin filaments, Edelstein-Keshet, Mathematical Biology, 1998

- If N is large it is very messy to type all these differential equations in by hand. Here, the array type model specification of SBmodels helps in setting up such equations

# Array specification of TEXT SBmodels

# Example

```
********** MODEL STATES
d/dt(x<n,0>) = 0
d/dt(x<n,1:N>)= kon<n>*x<1>*(x<n-1>) + koff*x<n+1> - (kon*x<1>+koff)*x<n> + R<n>
d/dt(x<n,N+1>) = 0

x<n,[1:2, 4:N]>(0) = n*N
x<n,3>(0) = 100
x<n,N+1>(0) = 0

********** MODEL PARAMETERS
N = 10
koff = 2
kon = 0.01

********** MODEL VARIABLES
kon<k,1:N> = kon*sqrt(k)

SUMEXAMPLE1 = 5 + arraysumSB(n^2/(x<n,1:N> + n/N))+ 56
SUMEXAMPLE2 = arraysumSB(x<n,1:N>)
SUMEXAMPLE3 = arraysumSB(x<n,[1,3,5]>)
SUMEXAMPLE4 = arraysumSB(x<n,[1:2:N-1]>)

********** MODEL REACTIONS
R<n,1:N> = koff*x<n+1> - kon<n>*x<n>*N*n
```

Simple to use format

ODEs, variables, and reactions can be defined by arrays

Negative indices possible

arraysumSB is a powerful and general construct to determine sums etc. over desired array elements

During import of such a model the array notation is expanded

...

# Array specification of TEXT SBmodels
# Example

- The expansion of the array notation is best understood by running an example (change into the „Example Files" folder):

```
>> edit array.txt                    % opens the model file as seen in the previous slide
>> model = SBmodel('array.txt')      % import the model
>> SBedit(model)                     % look at the model and compare to array.txt
```

- Now change „**N**" in the array.txt file to **N=100** and save the file

```
>> edit array.txt                    % opens the model file as seen in the previous slide
>> model = SBmodel('array.txt')      % import the model
>> SBedit(model)                     % look at the model and compare to array.txt
```

- **You see the difference?**

# Array specification of TEXT SBmodels
# Example

- Array-type notation and standard ODE specification can be combined

- More information about the array notation can be found in the `array_notation_explained.txt` model in the „`Example Files`“ folder

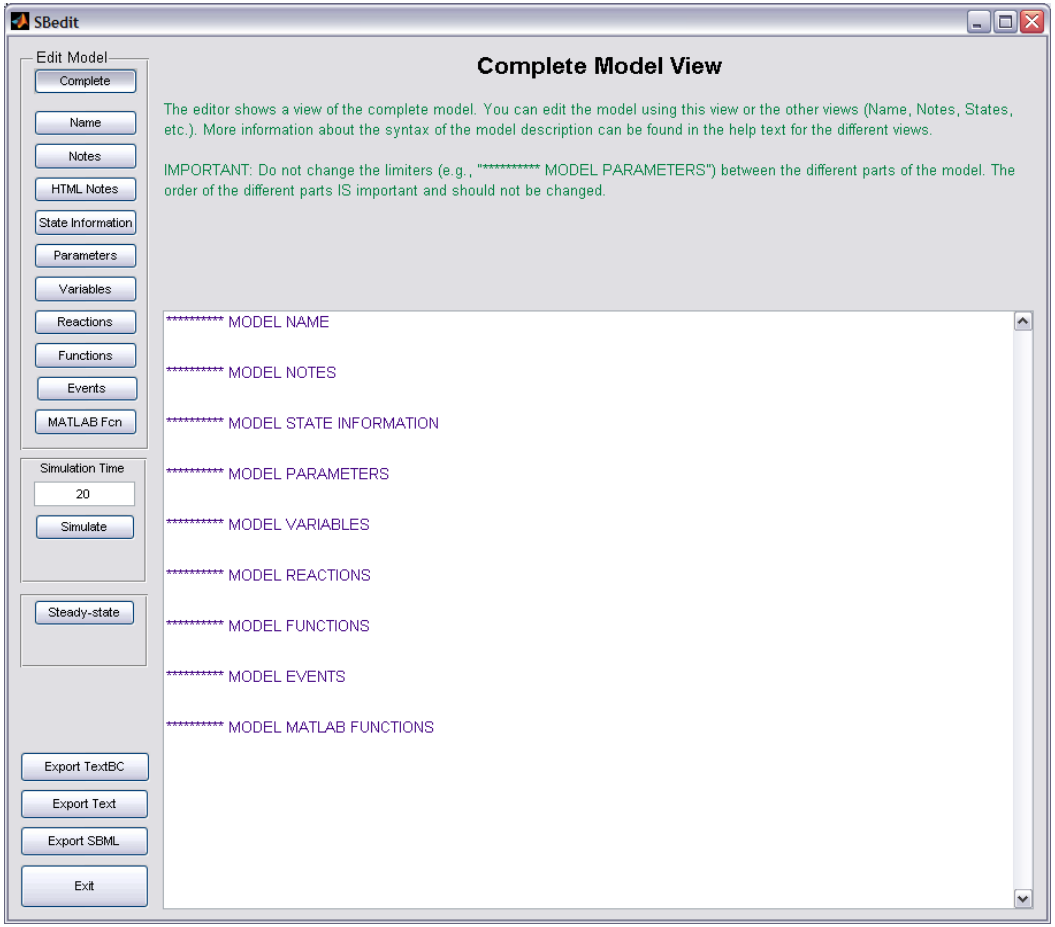
- Biochemical reaction equation based models

# SBmodel

# Modeling Using BioChemical Reaction Equations

```
>> model = SBeditBC()
```

- **SBeditBC**
  Graphical User Interface allowing to edit models based on reaction equations
- Click on the "Edit Model" buttons
- Each view provides a help text about the model syntax
- The limiters are important – do not change them





henning.schmidt@novartis.com

# Simple Model Again …

Enter the following information (keep all non used limiters)

```
********** MODEL NAME
Simple model
********** MODEL STATE INFORMATION
A(0) = 1
B(0) = 0
********** MODEL PARAMETERS
k1 = 0.5
********** MODEL REACTIONS
A => B : R
    vf = k1*A
```

A –R–> B

A(0) = 1
B(0) = 0

k1 = 0.5

R = k1*A

Click ”Simulate”


henning.schmidt@novartis.com

# More Complex Model

```
********** MODEL STATE INFORMATION
B(0) = 1
********** MODEL PARAMETERS
k1 = 0.5
********** MODEL REACTIONS
A+B => 2*C : R1
       vf = k1*A*B
B <=> A+D : R2
       vf = 5.1*B
       vr = 3*A*D
2*A => A2 : R3
       vf = 2.7*A^2
```

- Click ”Simulate”
- Click ”Exit”

```
>> model = SBedit(model)
```

=> Interchangeable formats

- Import/Export of models
  - Textual description
  - SBML

# Export a Model to the Textual Descriptions

```
>> SBedit(model)    % the variable "model" contains a model from the previous work
```

- To export a model to the ODE / Biochemical textual description using SBedit or SBeditBC click **"Export Text"** / **"Export TextBC"**
  - Choose a file name – here: **textmodel.txt** / **textmodel.txtbc**
  - ODE textmodel files are required to have the extension **.txt**
  - Biochemical textmodel files are required to have the extension **.txtbc**
  - Click "Exit"

- Export by command line commands

```
>> SBcreateTEXTfile(model, 'textmodel')       % saves model as "textmodel.txt"

>> SBcreateTEXTBCfile(model, 'textmodel')     % saves model as "textmodel.txtbc"
```

# Import a Textual Description to an SBmodel

- Open the TEXT(BC) model file with the MATLAB editor to see its syntax

```
>> edit textmodel.txt

>> edit textmodel.txtbc
```

- To import the TEXT(BC) model and convert it to an SBmodel write

```
>> model = SBmodel('textmodel.txt')     % the SBmodel command loads a model in txt

>> model = SBmodel('textmodel.txtbc')   % and in txtbc format. The extensions are important!
```

# Import of SBML Models

- SBML Level 1 and 2 models can be imported
- **The SBML Toolbox needs to be present**
- Change into the "**Example Files**" folder

```
>> model = SBmodel('CellCycle.xml')     % SBmodel additionally also import SBML models
    SBmodel
    =======
    Name: CellCycle
    Number States:              13
    Number Variables:           2
    Number Parameters:          41
    Number Reactions:           23
    Number Functions:           0

>> SBedit(model)
```

Increase simulation time to 400 and click "Simulate"


henning.schmidt@novartis.com

## Import of SBML Models

# Import of Incompletely Defined SBML Models

- Per default the SBTOOLBOX2 requires an SBML model to be completely defined before it is correctly imported

- To allow the user to import incompletely defined models the following optional call to SBmodel exists:

```
% File located in the "Example Files" folder
>> model = SBmodel('SBMLfileIncomplete.xml',1);
>> SBedit(model)
```

  - **SBmodel** then does not require that an SBML model is fully defined in terms of parameter values, rate equations, kinetic parameters, etc.

  - However, when trying to „Exit" SBedit or SBeditBC model correctness is checked. To still exit on Windows click the red cross in the upper right corner of the window. On Unix/Linux/Mac also click the corresponding symbol ...

## Import of SBML Models

# Name to ID Conversion

- 'Name' to 'id' conversion
  - E.g.: Not all models in Biomodels.net have informative IDs, and CellDesigner chooses the ids of certain elements (e.g., species and reactions) automatically and the user only can choose names that need not be unique

```
********** MODEL REACTIONS
reaction_0000001 = compartment_0000002 * (parameter_0000027 * parameter_0000021 +
        parameter_0000031)
reaction_0000002 = compartment_0000001 * (parameter_0000028 * parameter_0000020 +
        parameter_0000032)
reaction_0000003 = compartment_0000001 * delay(parameter_0000029 * parameter_0000022 +
        parameter_0000034, parameter_0000039)
reaction_0000004 = compartment_0000001 * (parameter_0000030 * (power(species_0000009, 2) /
        (power(species_0000009, 2) + power(parameter_0000010, 2))) * (power(parameter_0000008, 2) /
        (power(species_0000007, 2) + power(parameter_0000008, 2))) + parameter_0000033)
```

  - In the SBT the ids are used as names for states, variables, etc. (since they are unique)
  - => an optional call to SBmodel (see box below) allows to use the SBML names instead of the IDs, but making them unique by numerical extensions
  - Only for Level 2 SBML models

```
>> model = SBmodel('SBMLfileIncomplete.xml',1); % YES, it is the same flag as on the page before!
```

# Export of SBML Models

- Export only to SBML Level 2
- SBmodels do not necessarily contain all needed information for export
  - Additional information is required

```
>> help SBexportSBML          % for more information on the additional information
```

- Location of additional information (in SBmodel internal data structure)

| | |
|---|---|
| **states.type** | **algebraic.type** |
| **states.compartment** | **algebraic.compartment** |
| **states.unittype** | **algebraic.unittype** |
| | |
| **parameters.type** | **variables.type** |
| **parameters.compartment** | **variables.compartment** |
| **parameters.unittype** | **variables.unittype** |

| | | | |
|---|---|---|---|
| ***.type:** | 'isSpecie' | 'isParameter' | 'isCompartment' |
| ***.compartment:** | compartment | - | outside compartment |
| ***.unittype:** | 'amount' or 'concentration' | - | - |

## Export of SBML Models

# Additional Information in the TEXT Description

- Additional information can be added in the text / textbc format

```
>> help SBedit
>> help SBeditBC
```

- After SBML import the additional information is already present

```
********** MODEL NAME
example

********** MODEL NOTES


********** MODEL STATES
d/dt(s1) = -re1 {isSpecie:cytosol:amount} % comment
d/dt(s2) = +re1 {isSpecie:cytosol:amount}
d/dt(s3) = -re2+re4 {isSpecie:cytosol:amount}
d/dt(s4) = +re2-re5 {isSpecie:cytosol:amount}
d/dt(s5) = (-re3)/nucleus {isSpecie:nucleus:concentration}
d/dt(s6) = +re3 {isSpecie:nucleus:amount}

s1(0) = 1

********** MODEL PARAMETERS
s7 = 1 {isParameter} % comment
s8 = 1 {isParameter}
cytosol = 1 {isCompartment:}
nucleus = 0.1 {isCompartment:cytosol}


********** MODEL VARIABLES


********** MODEL REACTIONS
re1 = 3 * s1 - 2 * s2 {reversible} % comment
re2 = s3
re3 = s4 * (s5 - s6) {reversible}
re4 = 1
re5 = s4
```


henning.schmidt@novartis.com

# Export of SBML Models

# Automatic Determination of Additional Information

- Even if no additional information is given, the toolbox is under certain conditions able to determine the correct information.

novaktyson1.txt

```
d/dt(Cyclin) = R1-R2-R3
d/dt(YT) = R4-R5-R6-R7+R8+R3
d/dt(PYT) = R5-R8-R9-R10+R11
d/dt(PYTP) = R12-R11-R13-R14+R9
d/dt(MPF) = R6-R4-R12-R15+R13
d/dt(Cdc25P) = R16
d/dt(Wee1P) = R17
d/dt(IEP) = R18
d/dt(APCstar) = R19
```

**=> Species in Amount units**

novaktyson2.txt

```
d/dt(Cyclin) = (R1-R2-R3)/compartment
d/dt(YT) = (R4-R5-R6-R7+R8+R3)/compartment
d/dt(PYT) = (R5-R8-R9-R10+R11)/compartment
d/dt(PYTP) = (R12-R11-R13-R14+R9)/compartment
d/dt(MPF) = (R6-R4-R12-R15+R13)/compartment
d/dt(Cdc25P) = (R16)/compartment
d/dt(Wee1P) = (R17)/compartment
d/dt(IEP) = (R18)/compartment
d/dt(APCstar) = (R19)/compartment
```

**=> Species in Concentration units**

- Remember: SBML assumes reaction rates defined in amount/time

- Here it is important that the elements on the RHS are defined under the „**MODEL Reactions**“ header

# Export of SBML Models
# Automatic Determination of Additional Information

- Example

```
>> model = SBmodel('novaktyson1.txt')   % file located in ”example files” folder
>> SBexportSBML(model)                  % choose sbmlmodel1 as name for the file

>> model = SBmodel('novaktyson2.txt’)   % file located in ”example files” folder
>> SBexportSBML(model)                  % choose sbmlmodel2 as name for the file
```

- Have a look at both files (novaktyson1.txt and ...2.txt)

- Have a look at the exported SBML files
  - => species, reactions, compartments and unittypes are correctly determined

- This does not work for all possible model definitions, and thus the manually added information will override the automatically generated one

- High performance computation via MEX simulation functions

# MEX Simulation Functions – Why?

- Parameter estimation and clinical trial simulations require many repeated simulations

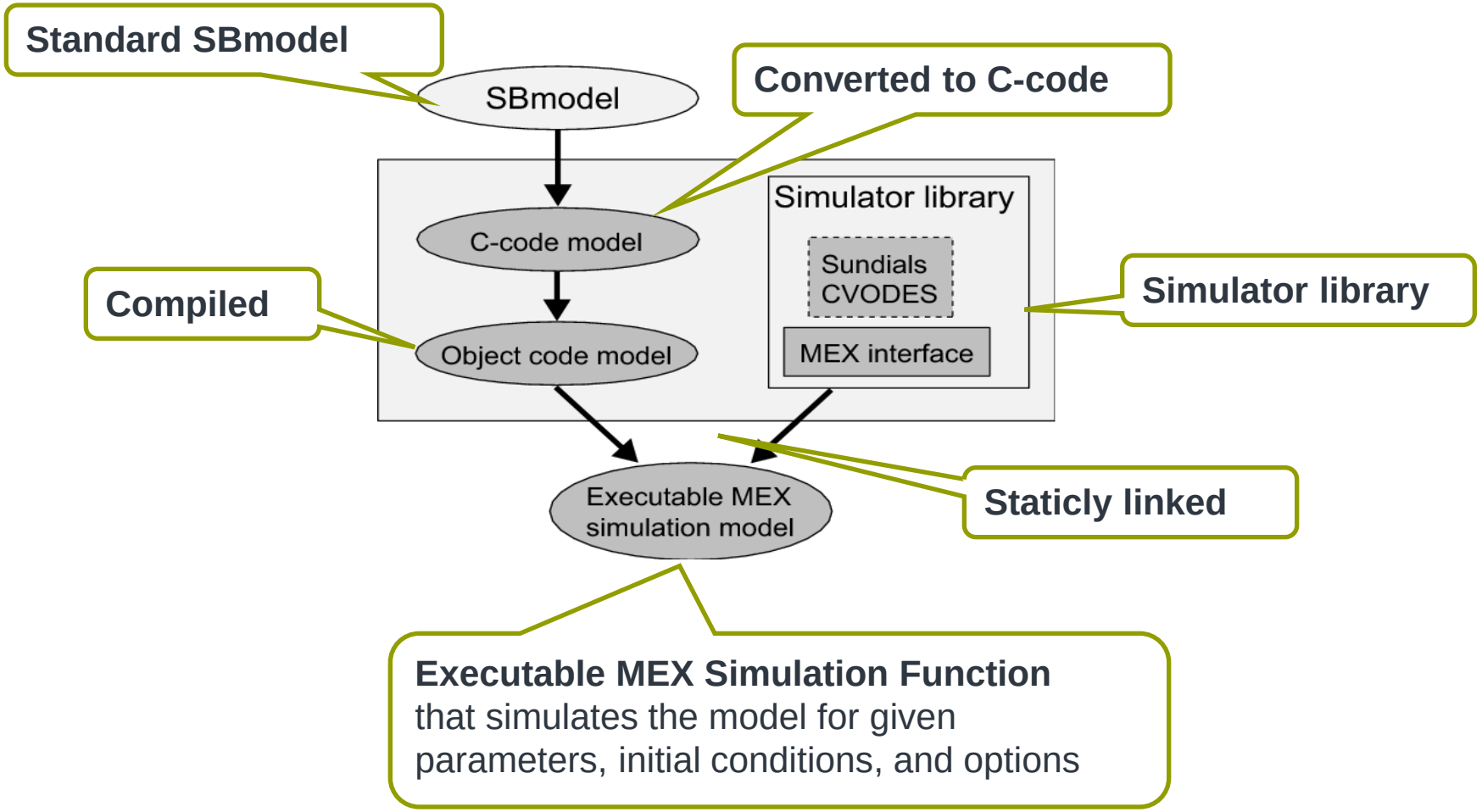
  **=> Simulation speed matters a lot**

- Benchmark (**MEX simulation functions** vs. **ODE15s**)

| | Model 1 | Model 2 | Model 3 |
|---|---|---|---|
| **Model description** | Novak Tyson cell-cycle model | Full-scale model of glycolysis in yeast | Model 14 from the Biomodels.net database |
| **End time (TEND)** | 1000 | 50 | 300 |
| **Number simulation points (NRPOINTS)** | 1000 | 200 | 300 |
| **Average time ODE15S [ms]** | 3442 | 3448 | 743 |
| **Average time SBPD [ms]** | 24 | 68 | 24 |
| **Speedup by SBPD** | **144x** | **51x** | **30x** |

# MEX Simulation Functions

- What are MEX Simulation Functions?



# MEX Simulation Functions - Generation

- A simple example (change into the „**Example Files**“ folder)

```
>> model = SBmodel('novaktyson1.txt')                % load a model
>> SBPDmakeMEXmodel(model)                           % create a MEX simulation function
```

- Simulation of a MEX model

```
>> simdataMEX = NovakTysonModel([0:1:400])
>> simdata = SBsimulate(model,[0:1:400])
```

- The results of both simulations are identical

- C-Conversion only (no compilation) => **WYSIWYC**

```
>> SBPDmakeMEXmodel(model,'MEXmodel',1) % convert to MEXmodel.c and MEXmodel.h
>> edit MEXmodel.c                      % look at the created files
>> edit MEXmodel.h
>> SBPDmakeMEXmodel(model,'MEXmodel')    % Same but with compilation!
```

# MEX Simulation Functions – Syntax

- How a MEX simulation function can be called

```
output = MEXmodel()

output = MEXmodel('states')

output = MEXmodel('parameters')

output = MEXmodel('parametervalues')
```

- With prior definition of `timevector`, `initialconditions`, etc.:

```
output = MEXmodel(timevector)

output = MEXmodel(timevector, initialconditions)

output = MEXmodel(timevector, initialconditions, parametervector)

output = MEXmodel(timevector, initialconditions, parametervector, options)
```

# MEX Simulation Functions – Syntax

- Options for MEX model simulation (more available, see help text)

```
options.abstol:          Absolute tolerance (default: 1e-6)
options.reltol:          Relative tolerance (default: 1e-6)
options.minstep:         Minimal integrator step size (default: 0)
options.maxstep:         Maximal integrator step size (default: inf)
options.maxnumsteps:     Maximal number of steps between two output points (default: 500)
options.xdotcalc:        =0: do integration (default), =1: return RHS of ODEs for
                         given state and parameter values. Time information
                         is neglected and it is assumed that time=0.
```

- More information on **www.sbtoolbox2.org** or:

```
>> help SBPDmakeMEXmodel
```

# Limitations of MEX Simulation Functions

- The MEX models can not handle
  - SBML Algebraic rules
  - The fast reaction flag included in SBML
- Delays and delayed events can be handled, but the simulation performance becomes VERY poor ... In this case the standard MATLAB simulation is faster
- C compatibility necessary (no MATLAB functions can be used that do not exist in C)

- Simple Simulation

# Deterministic Simulation

- Serves also as example for using the Cell-Mode

- Change into the „**Example Files**“ folder

```
>> edit simpleSimulation
```

1. Read the documentation in the opened file
2. Execute the cells sequentially by pressing „**Ctrl+Enter**“

# Stochastic Simulation

- Serves also as example for using the Cell-Mode

- Change into the „**Example Files**“ folder

```
>> edit stochasticSimulation
```

1. Read the documentation in the opened file
2. Execute the cells sequentially by pressing „**Ctrl+Enter**“

- Commenting of models

# WHY COMMENTS AND DOCUMENTATION?

- Models **without** comments and documentation **are even less useful** than software without documentation

# Commenting a Model

- The more complex a model the better it is if there are comments included

- The SBmodels allow 3 types of comments
  - Information in the „MODEL Notes“
  - Optional comment on each state, variable, parameter, etc.
  - Furthermore, .txt and .txtbc files allow to use whole lines for commenting if prefixed with the „%“ character.

- Example:

```
********** MODEL REACTIONS
% This is just a comment about
% the following reaction:
A+B => 2*C : R1   % comment
       vf = k1*A*B
```

# Commenting a Model
# Example

```
********** MODEL NOTES
Unified phototransduction model from Hamer et al., Visual Neuroscience 22, 417-436
This model here uses only 6 phosphorylation states

********** MODEL STATE INFORMATION
% ODEs for the "back-end" model
d/dt(g) = alfamax/(1+power((c/Kc),m)) - (betadark + betasub*PDE_a)*g

% Initial Conditions
R(0) = 3.6e9                % (#) Rhodopsin unactivated (same value as parameter Rtot)
PDE(0) = 2.67e7             % (#) PDE (same value as parameter PDEtot)

********** MODEL PARAMETERS
% Total concentrations or numbers
Rtot = 3.6e9            % (#) total amount of Rhodopsin (same value as initial condition for R)

% R_n bound to RK  pre=>post
kRK3_ATP = 400                      % here it is already multiplied by ATP, see paper xyz

% Unbinding of R_n and RK
kRK4 = 20                           % value measured in paper WXY

********** MODEL REACTIONS
% Inactivation Pathways
% ======================
% R_n (activated Rhodopsin n-times phosphorylated) binding to RK
% RK is in number of molecules not in concentration. (Different to paper but
% taken into account by having scaled kRK1_n with RK in numbers.
R_0 + RK <=> R_0_RK_pre : v_A1a_0  % Comment about reaction
    vf = kRK1_0 * RK * R_0
    vr = kRK2 * R_0_RK_pre
```

# Commenting a Model

- Using MATLAB syntax highlighting comments are displayed in a different color
- Comments make the model more readable
- Modeling process, assumptions, references, etc. **well documented**

- Important:
  - Note that during import of a .txt or .txtbc model the comments shown in **blue** (lines) are removed from the model
  - However, when working directly on .txt and .txtbc models using the MATLAB editor the comments stay

## Commenting a Model

# Using Named Kinetic Rate Laws

- kin_allosteric_inihib_empirical_rev
- kin_allosteric_inihib_mwc_irr
- kin_catalytic_activation_irr
- kin_catalytic_activation_rev
- kin_comp_inihib_irr
- kin_comp_inihib_rev
- kin_constantflux
- kin_degradation
- kin_hill_1_modifier_rev
- kin_hill_2_modifiers_rev
- kin_hill_cooperativity_irr
- **kin_hill_rev**
- kin_hyperbolic_modifier_irr
- kin_hyperbolic_modifier_rev
- kin_iso_uni_uni_rev
- kin_mass_action_irr
- kin_mass_action_rev
- kin_michaelis_menten_irr
- kin_michaelis_menten_rev
- kin_mixed_activation_irr
- kin_mixed_activation_rev
- kin_mixed_inihib_irr
- kin_mixed_inihib_rev
- kin_noncomp_inihib_irr
- kin_noncomp_inihib_rev
- kin_ordered_bi_bi_rev
- kin_ordered_bi_uni_rev
- kin_ordered_uni_bi_rev
- kin_ping_pong_bi_bi_rev
- kin_specific_activation_irr
- kin_specific_activation_rev
- kin_substrate_activation_irr
- kin_substrate_inihib_irr
- kin_substrate_inihib_rev
- kin_uncomp_inihib_irr
- kin_uncomp_inihib_rev
- kin_uni_uni_rev

Instead of

```
********** MODEL REACTIONS
R = Vf*substrate/Shalve*(1-product/(substrate*Keq))*
    (substrate/Shalve+product/Phalve)^(h-1) /
    ( 1+(substrate/Shalve + product/Phalve)^h )
```

You can write

```
********** MODEL REACTIONS
R = kin_hill_rev(Vf,substrate,Shalve,product,Keq,Phalve,h)
```

```
>> help SBPD
```

37 inbuild rate laws (SBPD)
More rate laws can easily be added

**Readability of models is improved**


henning.schmidt@novartis.com

# Tutorial Outline

- **General introduction to the SBPOP Package**
- **Model definition and simulation**
- **Systems Biology relevant topics**
  - **Model analysis**
    - Steady-state analysis and stability
    - Moiety conservations and reduction
    - Parameter sensitivity analysis (steady-state & oscillating systems, MCA, local & global)
    - Localization of complex behaviors
- **Pharmacometrics / Systems Pharmacology relevant topics**

- Steady-State Analysis and Stability

# Steady-state Determination

- Change into the „**Example Files**“ folder
- Determination of the steady-state

```
>> model = SBmodel('CellCycle.txt')
>> SBsteadystate(model)

Steady state could not be found.
Try different options and/or a different starting guess.

>> SBinitialconditions(model)  % almost all zero
```

- Starting conditions are important
- One possibility to get starting conditions is to simulate a short time

```
>> output = SBsimulate(model,20)
>> ss = SBsteadystate(model,output.statevalues(end,:))
```

- Another possibility is the use of SBedit to set new initial conditions

# Jacobian and Stability

- Determination of the Jacobian

```
>> Jacobian = SBjacobian(model,ss)
```

- Determination of stability by considering the Jacobian eigenvalues

```
>> eig(Jacobian)
          0.0413 + 0.1528i
          0.0413 - 0.1528i
          ...
```

- Two complex conjugated eigenvalues with positive real part
  => the considered steady-state is unstable and the system is oscillating around it

- Moiety Conservations and Reduction

# Moiety Conservations

- Determination of moiety conservations

```
>> model = SBmodel('CellCycle.txt')
>> SBmoietyconservations(model)

Cdc25P = 1 - 1 Cdc25
Wee1 = 1 - 1 Wee1P
APC_ = 1 - 1 APC
IEP = 1 - 1 IE
```

- Moiety conservations (linear dependencies between ODEs) are not allowed to be present in a system, e.g., for bifurcation analysis

- Moiety conservations present => The model is singular

# Simple Model Reduction

- Singular model -> Non singular model
- Replacement of linear dependent state variables by static variables

```
>> model = SBmodel('CellCycle.txt')
>> modelred = SBreducemodel(model)

>> SBedit(modelred)
```

- The `modelred` model has 4 states less and 4 variables more
- The simulation results are identical

- Sensitivity Analysis

# Forward Sensitivity Analysis

- Determine the sensitivy trajectories wrt to perturbations in IC and parameters

$$\begin{aligned} \frac{d}{dt}x &= f(x,p) \\ x(0) &= x_0 \end{aligned}$$

$$\begin{aligned} S(t) &= \frac{dx(t)}{dp} \\ \frac{d}{dt}S &= \frac{\partial f(x,p)}{\partial x}S + \frac{\partial f(x,p)}{\partial p} \\ S(0) &= 0 \end{aligned}$$

```
>> model = SBmodel('novaktyson1.txt')
>> output = SBPDsensitivity(model, [0:1:200], {'k1', 'Ka', 'khs'}, {'Cyclin','YT'})
output =

                time: [1x201 double]
              states: {'Cyclin'  'YT'  'PYT'  'PYTP'  'MPF'  'Cdc25P'  'Wee1P'  'IEP'  'APCstar'}
         statevalues: [201x9 double]
           variables: {'k2'  'kwee'  'k25'}
      variablevalues: [201x3 double]
           reactions: {'R1'  'R2'  'R3'  'R4'  'R5'  'R6'  'R7'  'R8'  'R9'  'R10'  'R11'  'R12'  'R13'  'R14'  ... 'R19'}
      reactionvalues: [201x19 double]
      sensparameters: {'k1'  'Ka'  'khs'}
   paramtrajectories: [1x1 struct]
        sensicstates: {'Cyclin'  'YT'}
      ictrajectories: [1x1 struct]


>> output.paramtrajectories
ans =

      states: {[201x9 double]  [201x9 double]  [201x9 double]}
   variables: {[201x3 double]  [201x3 double]  [201x3 double]}
   reactions: {[201x19 double]  [201x19 double]  [201x19 double]}
```

# Local Parameter Sensitivity Analysis

- Steady-state sensitivities (states, reactions)
- Period and amplitude sensitivities for oscillating systems (states, reactions)
- Metabolic Control Analysis

- Two step approach (except for MCA)
  - 1) Generation of data for sensitivity analysis
  - 2) Determining of sensitivities and display of sensitivity data

```
>> help SBsensdatastat
>> help SBsensdataosc
>> help SBmca
```

```
>> help SBsensdataoscevents
```

# Local Parameter Sensitivity Analysis
# Steady-State

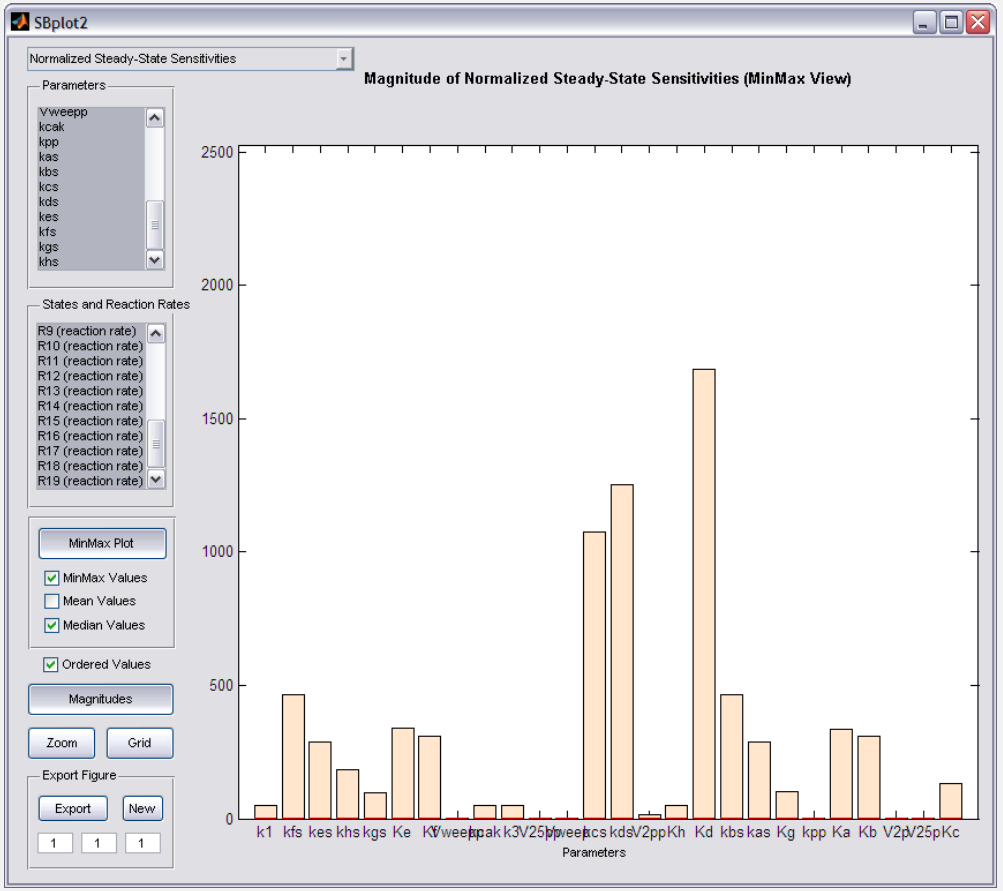
```
>> model = SBmodel('CellCycleRed.txt')
>> output = SBsensdatastat(model)
    ...
          model: [1x1 SBmodel]
         states: {9x1 cell}
         xssnom: [9x1 double]
       xsspert: {1x26 cell}
     reactions: {19x1 cell}
         rssnom: [19x1 double]
       rsspert: {1x26 cell}
    parameters: {26x1 cell}
     nomvalues: [1x26 double]
      pertSize: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
        absRel: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

>> SBsensstat(output)
```

# Local Parameter Sensitivity Analysis
# Steady-State

- **SBplot2**

  Graphical User Interface allowing to display block diagram type of data

- Play around with the GUI to get a feeling for its use

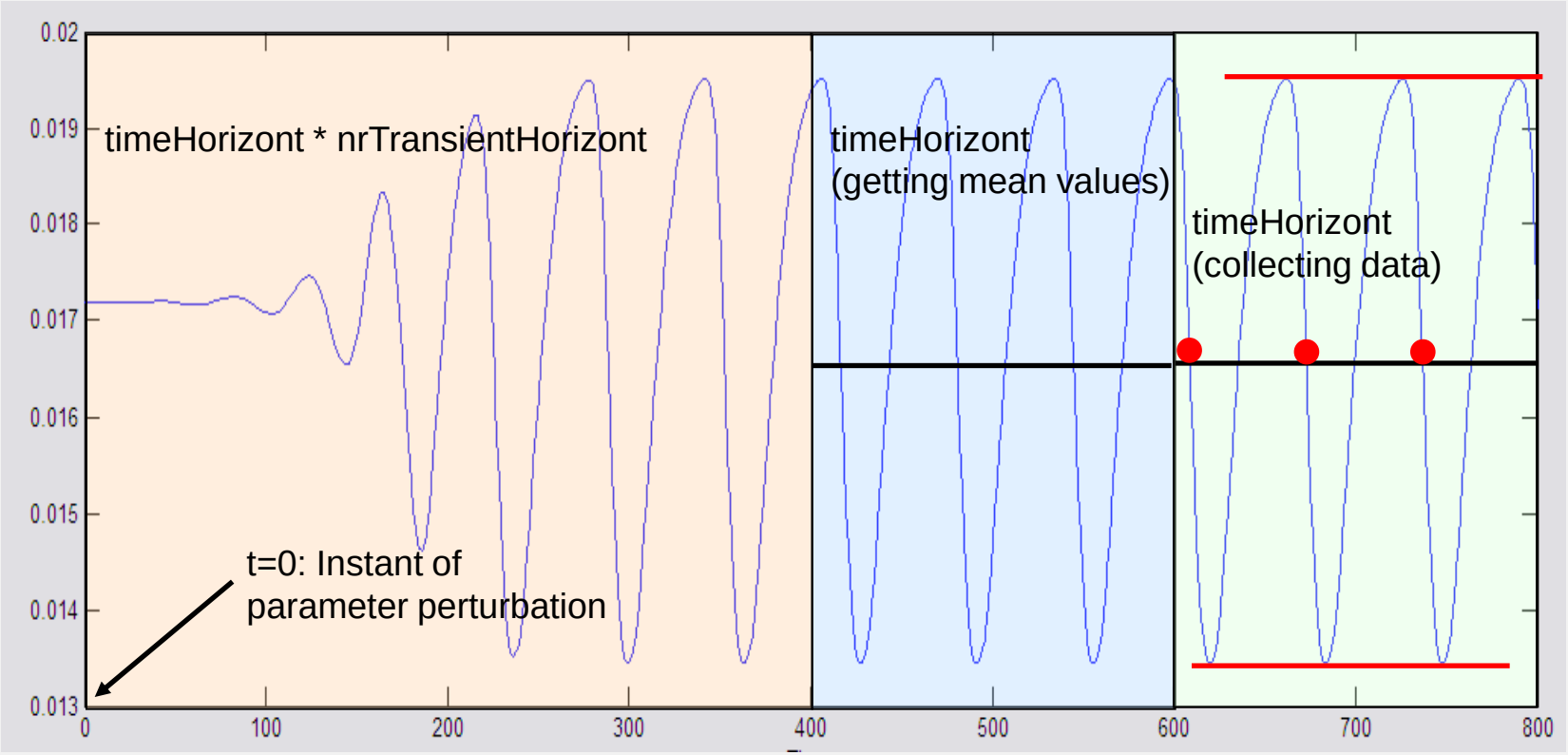


## Local Parameter Sensitivity Analysis

# Oscillating Systems (Period and Amplitude)

- How is it implemented?
- `output = SBsensdataosc(model,timeData)`

  `timeData = [timeHorizont nrTransientHorizont]`





henning.schmidt@novartis.com

## Local Parameter Sensitivity Analysis

# Oscillating Systems (Period and Amplitude)

- Simulate the model to determine the time needed for the transients to die out and the time needed to have several oscillations within the horizont

```
>> model = SBmodel('CellCycleRed.txt')
>> output = SBsensdataosc(model,[200 2])
output =
         model: [1x1 SBmodel]
          time: [1x1001 double]
         tenom: {1x9 cell}
        tepert: {1x26 cell}
        states: {9x1 cell}
          xnom: [1001x9 double]
         xpert: {1x26 cell}
    parameters: {26x1 cell}
     nomvalues: [1x26 double]
      pertSize: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
        absRel: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

## Local Parameter Sensitivity Analysis

# Oscillating Systems (Period and Amplitude)

- Determining and displaying sensitivities

```
>> SBsensperiod(output)     % display parameter sensitivities for the oscillation period
>> SBsensamplitude(output)  % display parameter sensitivities for the oscillation amplitude
```

- Accessing sensitivity data

```
>> sensdata = SBsensperiod(output)
sensdata =
                S: [9x26 double]
          statesS: {9x1 cell}
     parametersS: {26x1 cell}
               Sn: [9x26 double]
         statesSn: {9x1 cell}
    parametersSn: {26x1 cell}
>> sensdata = SBsensamplitude(output)
...
```

# Metabolic Control Analysis

- MCA is a special case of sensitivity analysis
- Function: **SBmca**
  - Flux Control Coefficients
  - Concentration Control Coefficients
  - Elasticity Coefficients

```
>> model = SBmodel('CellCycleRed.txt')  % load model

>> modelIR = SBmakeirreversible(model)  % make reversible reactions irreversible

>> ouput = SBmca(modelIR)               % perform MCA
       states: {9x1 cell}
    reactions: {23x1 cell}
          FCC: [23x23 double]
          CCC: [9x23 double]
           EC: [23x9 double]
```

- The analyzed model needs to contain **irreversible reactions** only!

# Global Sensitivity Analysis

- 4 global sensitivity algorithems implemented
  - **SBsensglobalfast** - Extended FAST
  - **SBsensglobalprcc** - PRCC (Partial Rank Correlation Coefficient)
  - **SBsensglobalsobol** - Sobols method
  - **SBsensglobalwals** - WALS (weighted average of local sensitivities)

```
(output = ) SBsensglobalfast(model,timevector)
(output = ) SBsensglobalfast(model,timevector,paramNames)
(output = ) SBsensglobalfast(model,timevector,paramNames,OPTIONS)
```

Syntax

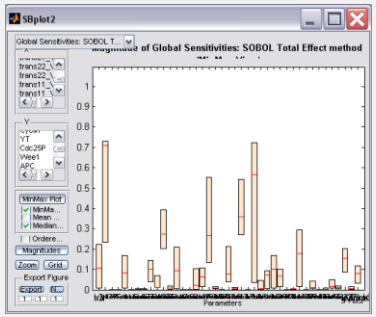
```
OPTIONS.statenames:      cell-array with state names which to consider as model outputs
OPTIONS.variablenames:   cell-array with variable names which to consider as model outputs
OPTIONS.reactionnames:   cell-array with reaction names which to consider as model outputs
OPTIONS.Nsim:            Number of simulation to carry out (approximate value)
OPTIONS.range:           Order of magnitude of parameter perturbations
OPTIONS.firstorder:      =0: use total effect, =1: use first order approx.

OPTIONS.objectivefunction: 'relative' or ' absolute'
OPTIONS.integrator:      Structure with optional settings for the integrator
```

OPTIONS

# Global Sensitivity Analysis
# Example

- Running global sensitivity analysis on the Cell-cycle model

```
>> model = SBmodel('CellCycle.txt');     % load the model
>> time = [0:2:400];                     % define time vector
>> parameters = SBparameters(model);     % define the parameters to consider

>> OPTIONS = [];
>> OPTIONS.statenames = SBstates(model);          % define all states as output variables
>> OPTIONS.Nsim = 1000;                           % around 1000 simulations
>> OPTIONS.range = 1;                             % order of magnitude of perturbation

>> SBsensglobalprcc(model,time,parameters,OPTIONS)          % run PRCC method
>> SBsensglobalsobol(model,time,parameters,OPTIONS)         % run Sobol's method
>> SBsensglobalfast(model,time,parameters,OPTIONS)          % run FAST method
>> SBsensglobalwals(model,time,parameters,OPTIONS)          % run WALS method

>> output = SBsensglobalprcc(model,time,parameters,OPTIONS)  % don't plot, just return data
```





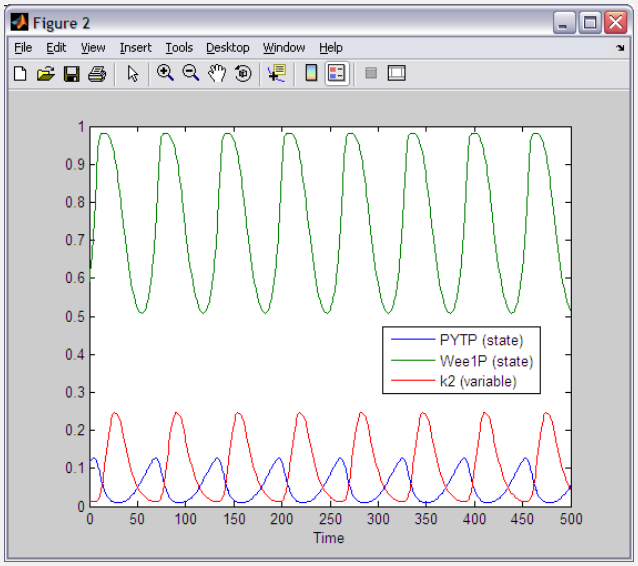
henning.schmidt@novartis.com

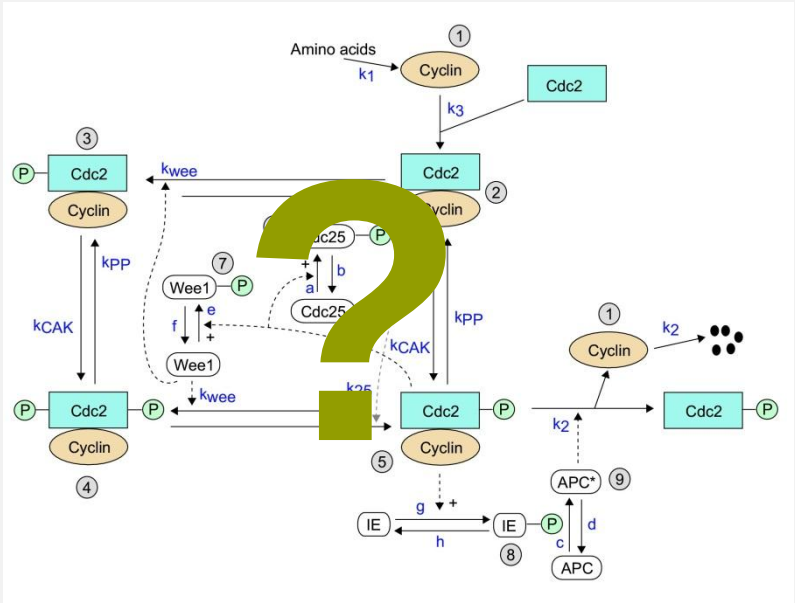
- Localization of Complex Behaviors

Schmidt, H., Jacobsen, E.W. (2004) Linear systems approach to analysis of complex dynamic behaviours in biochemical networks, *IEE Systems Biology*, 1, 149-158

# Localization of mechanisms leading to complex behavior

```
>> model = SBmodel('CellCycleRed.txt')
>> SBsimulate(model,500)
```

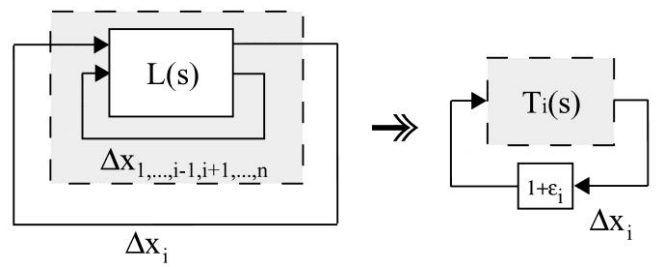


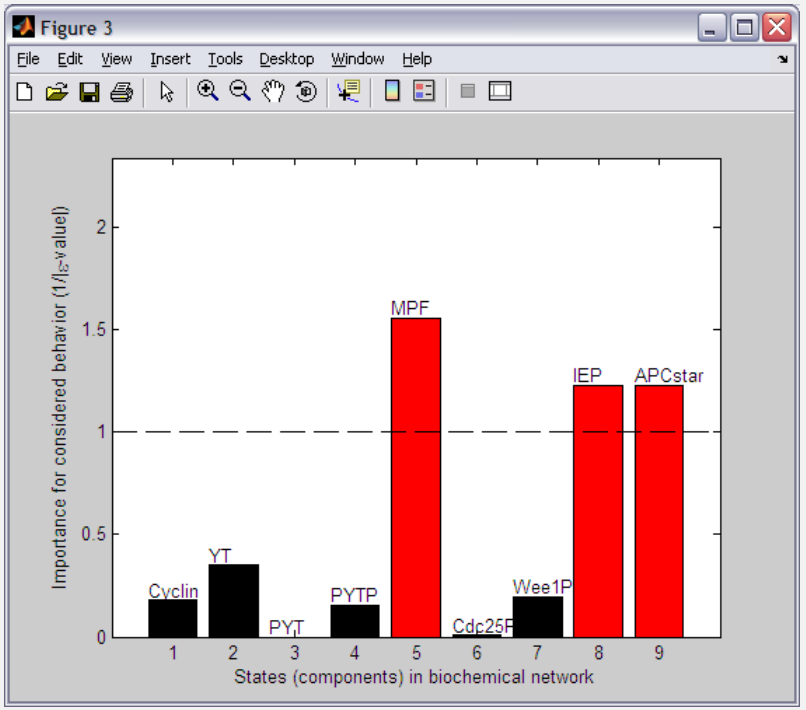

- Which mechanism is the source of the oscillations?

# Localization of mechanisms leading to complex behavior

- Importance of individual components / feedback signals

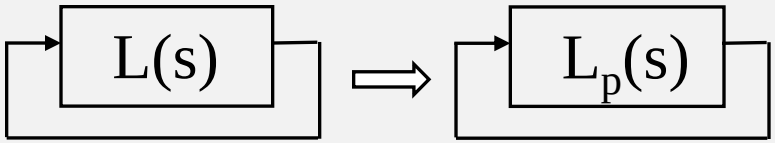



```
>> ss = SBsteadystate(model)
>> SBlocbehavcomp(model,ss)
```



# Localization of mechanisms leading to complex behavior

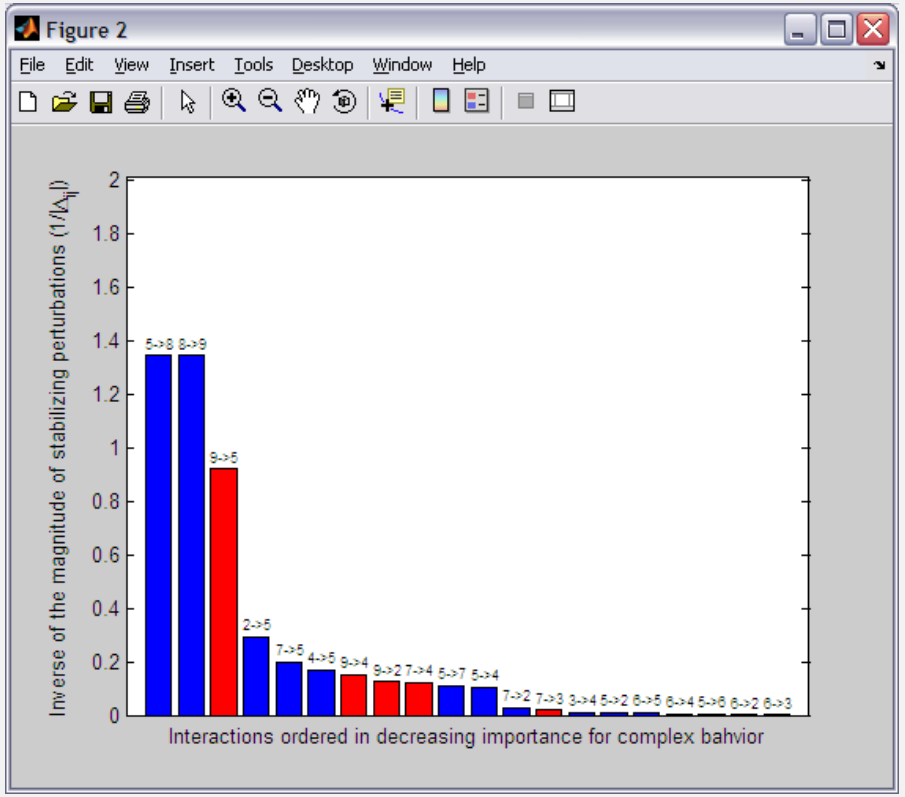
- Importance of pairwise component interactions




$$[L_p]_{ij} = [L]_{ij}\left(1 + \Delta_{ij}\right)$$

$$\Delta_{ij} = -\frac{1}{[RGA(I-L)]_{ij}}$$

```
>> ss = SBsteadystate(model)
>> SBlocbehavinteract(model,ss)
```

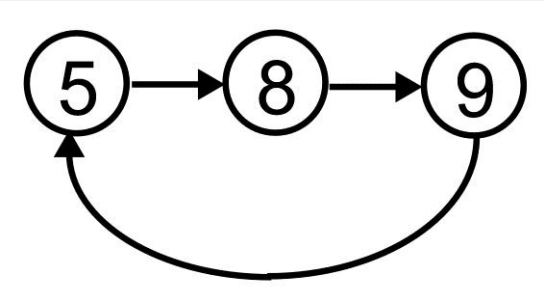


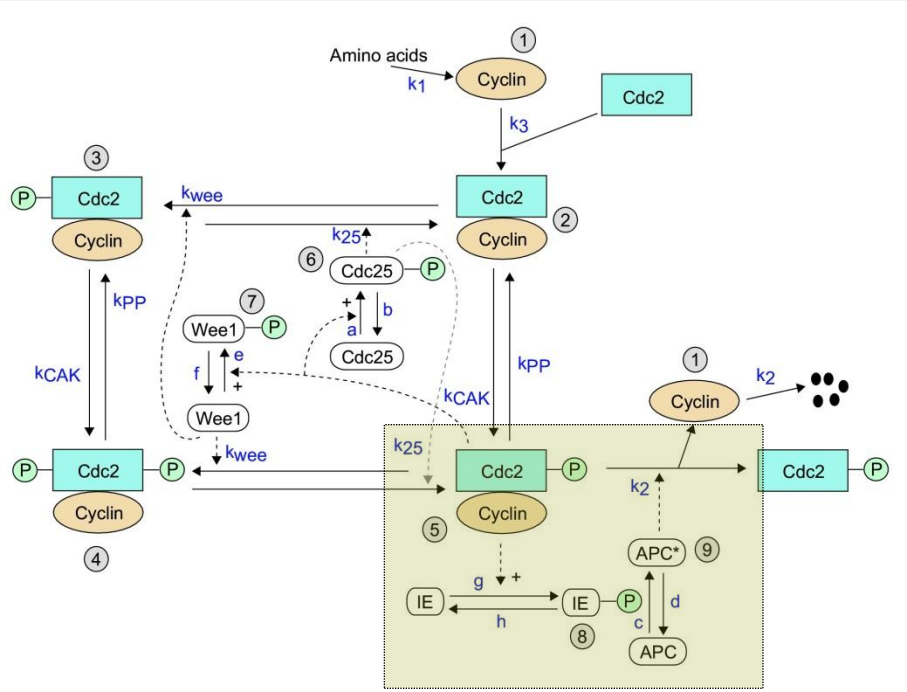


henning.schmidt@novartis.com

# Identified mechanism behind oscillations

- Feedback mechanism involving
  - 5: YTP
  - 8: IEP
  - 9: APC*





henning.schmidt@novartis.com

# Tutorial Outline

- **General introduction to the SBPOP Package**
- **Model definition and simulation**
- **Systems Biology relevant topics**
  - **Definition of experiments and measurement data**
    - Excel and CSV measurement representation
    - Experiment descriptions and merging with models
    - Import and export models of measurements and experiments
- **Pharmacometrics / Systems Pharmacology relevant topics**


henning.schmidt@novartis.com

- Representation of measurement data

# SBmeasurement

# Measurements

- The toolbox can handle 2 representation formats
  - Excel
  - Comma separated values (CSV)
- One Excel file can contain measurements of several experiments
- One CSV file can contain measurements of a single experiment
- Contents
  - Name
  - Notes
  - Components (same names as model components)
  - Componentnotes (additional information, e.g. units)
  - Values

# Measurement Examples and Import

- Measurements in Excel

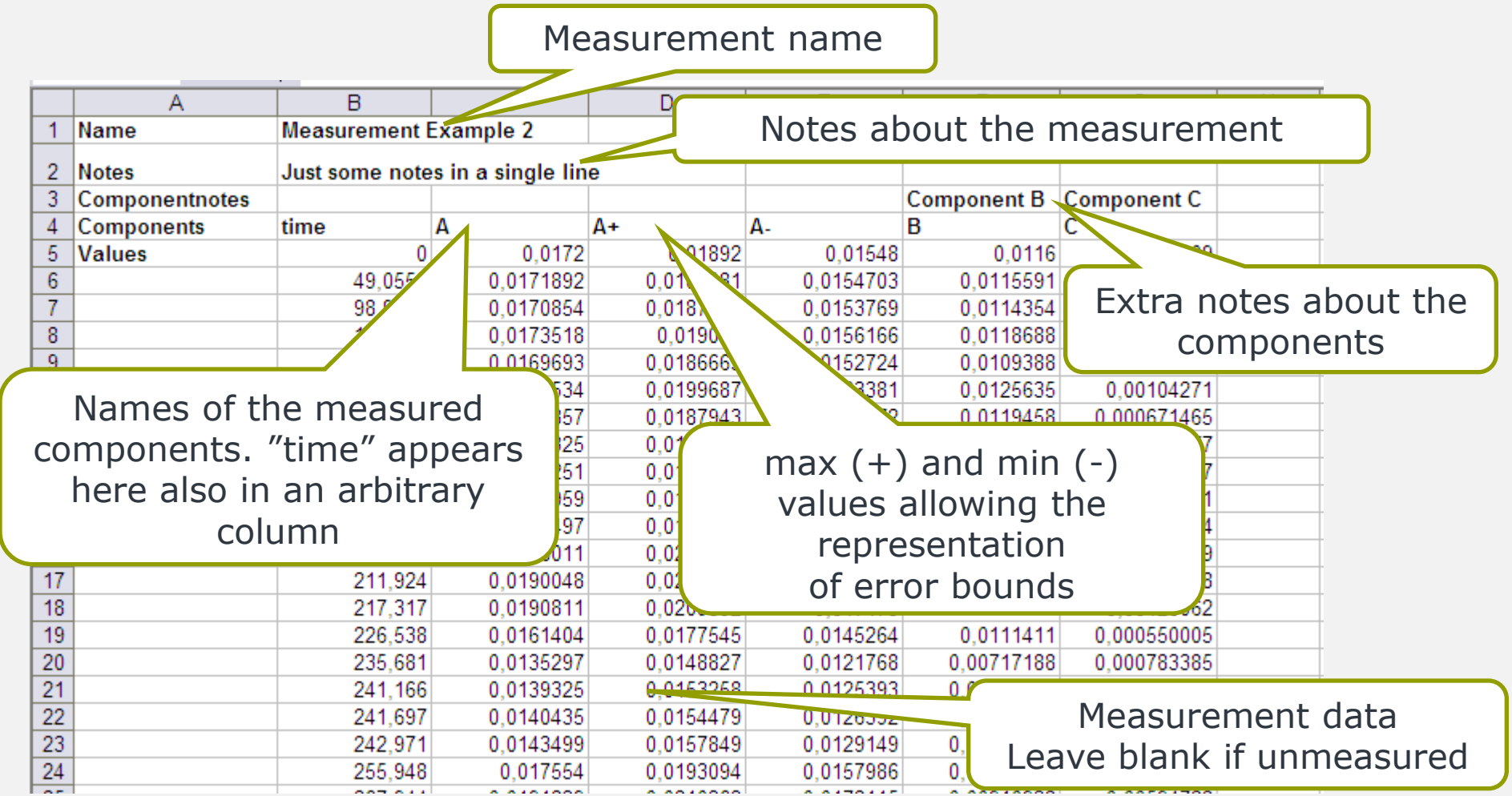
```
Change into the "Example Files" folder
Open the "MeasurementExample.xls" in Excel

>> data = SBmeasurement('MeasurementExample.xls')

data = [1x1 SBmeasurement]    [1x1 SBmeasurement]

>> data = data{1}  % use only first data object in the cell-array
```

WINDOWS ONLY

- Measurements as CSV  (importing CSV is MUCH faster then XLS)

```
>> edit MeasurementExample.csv

>> data = SBmeasurement('MeasurementExample.csv')

    SBmeasurement
    =============
    Name: MeasurementExampleCSV
    Measured components:      3
    Number time points:       42
    Error bound information present at least for one measurement.
    Measurements not present for all time points
```

# Measurements – Excel format




| | A | B | | D | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | Name | Measurement Example 2 | | | | | | |
| 2 | Notes | Just some notes in a single line | | | | | | |
| 3 | Componentnotes | | | | | Component B | Component C | |
| 4 | Components | time | A | A+ | A- | B | C | |
| 5 | Values | 0 | 0,0172 | 01892 | 0,01548 | 0,0116 | | |
| 6 | | 49,055 | 0,0171892 | 0,01 81 | 0,0154703 | 0,0115591 | | |
| 7 | | 98,6 | 0,0170854 | 0,018 | 0,0153769 | 0,0114354 | | |
| 8 | | 1 | 0,0173518 | 0,0190 | 0,0156166 | 0,0118688 | | |
| 9 | | | 0,0169693 | 0,018666 | 0152724 | 0,0109388 | | |
| | | | 534 | 0,0199687 | 3381 | 0,0125635 | 0,00104271 | |
| | | | 357 | 0,0187943 | 2 | 0,0119458 | 0,000671465 | |
| | | | 925 | 0,01 | | | 7 | |
| | | | 251 | 0,01 | | | 7 | |
| | | | 959 | 0,01 | | | 1 | |
| | | | 97 | 0,01 | | | 4 | |
| | | | 011 | 0,02 | | | 9 | |
| 17 | | 211,924 | 0,0190048 | 0,02 | | | 8 | |
| 18 | | 217,317 | 0,0190811 | 0,020 | | | 062 | |
| 19 | | 226,538 | 0,0161404 | 0,0177545 | 0,0145264 | 0,0111411 | 0,000550005 | |
| 20 | | 235,681 | 0,0135297 | 0,0148827 | 0,0121768 | 0,00717188 | 0,000783385 | |
| 21 | | 241,166 | 0,0139325 | 0,0153268 | 0,0125393 | 0,0 | | |
| 22 | | 241,697 | 0,0140435 | 0,0154479 | 0,012 | | | |
| 23 | | 242,971 | 0,0143499 | 0,0157849 | 0,0129149 | 0, | | |
| 24 | | 255,948 | 0,017554 | 0,0193094 | 0,0157986 | 0, | | |

**Worksheets in an Excel book can contain any data. BUT if A1 is set to ”Name” then the above format is expected.**

# Measurements – CSV format

```
[Name]
Measurement Example CSV

[Notes]
Notes about the measurement / experiment

[Components]
time,A,A+,A-,B,C

[Componentnotes]
A: Component A
B: Component B

[Values]
0, 0.0172, 0.01892, 0.01548, 0.0116, 0.0009
49.0552, 0.0171892, 0.0189081,  , 0.0115591, 0.000865671
98.9524, 0.0170854, 0.0187939, NaN , 0.0114354, 0.000858473
128.814, 0.0173518, 0.019087, 0.0156166, 0.0118688, 0.000825422
151.362, 0.0169693, 0.0186663, 0.0152724, 0.0109388, 0.00104578
160.548, 0.0181534, 0.0199687, 0.0163381, 0.0125635, 0.00104271
```

**Good practice**
Give a useful name and document the measurement data with telling notes

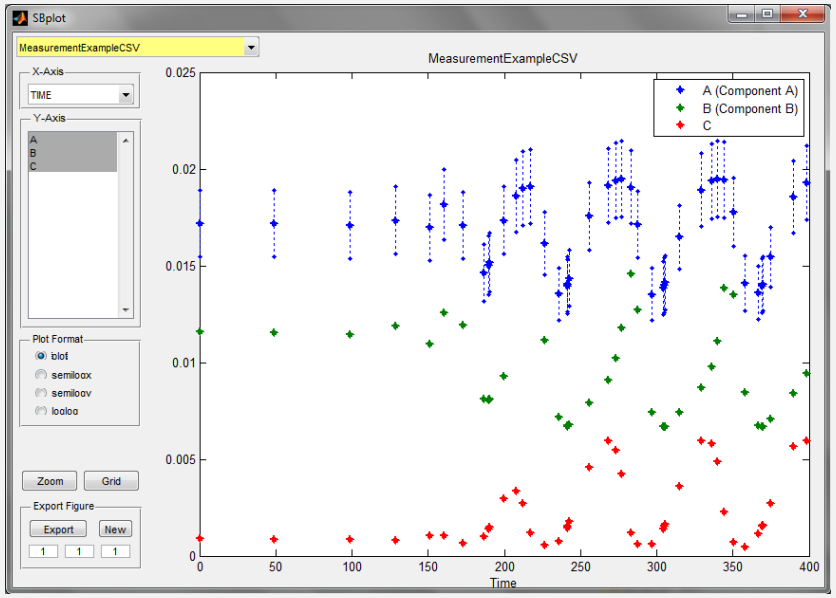
Measurement data
Leave empty or set to NaN if unmeasured

**Important:**
**The measurements should be given in units that are to be used for the models components!**

# Handle Measurement Data

- Visualizing the data

```
>> SBvisualizemeasurement(data)
```




- Extract information

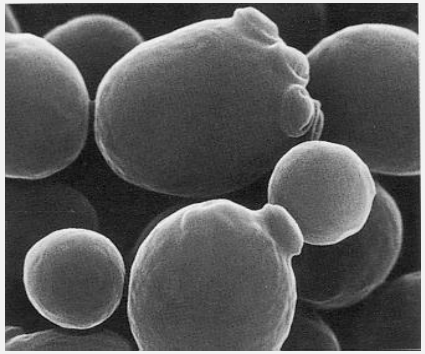
```
>> [time, componentNames, values, minvalues, maxvalues] = SBmeasurementdata(data)
```

- Representation of in silico experiments

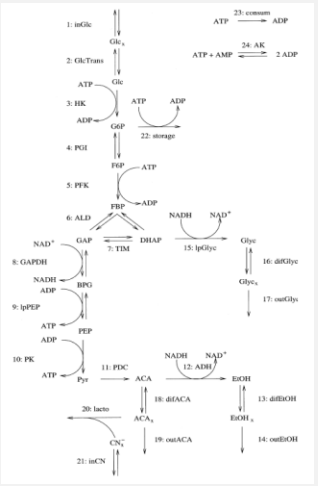
# SBexperiment

# Experiment Descriptions

- The System



The Model



- The Experiments
  - Knockout
  - Overexpression
  - Change of concentration 1
  - Change of concentration 2
  - etc.

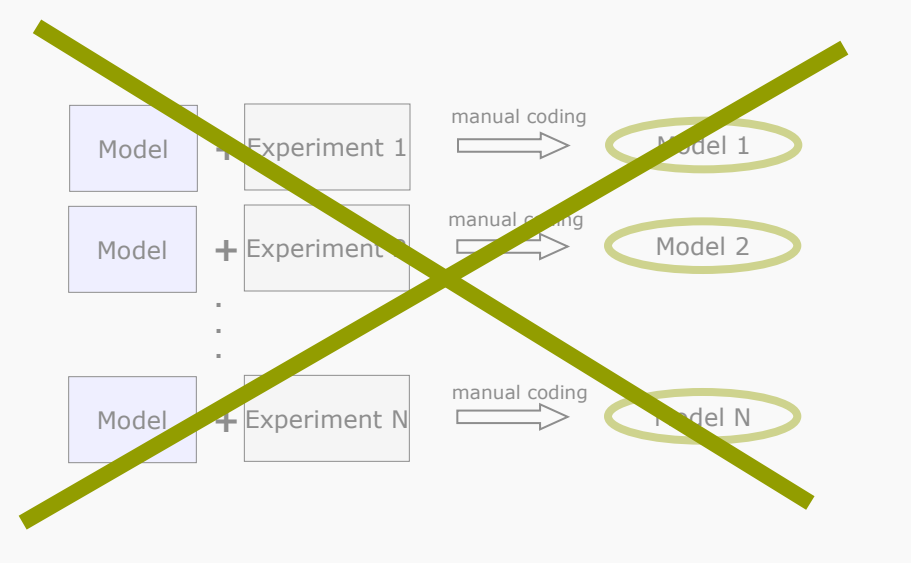
**One slightly different model for each experiment?**

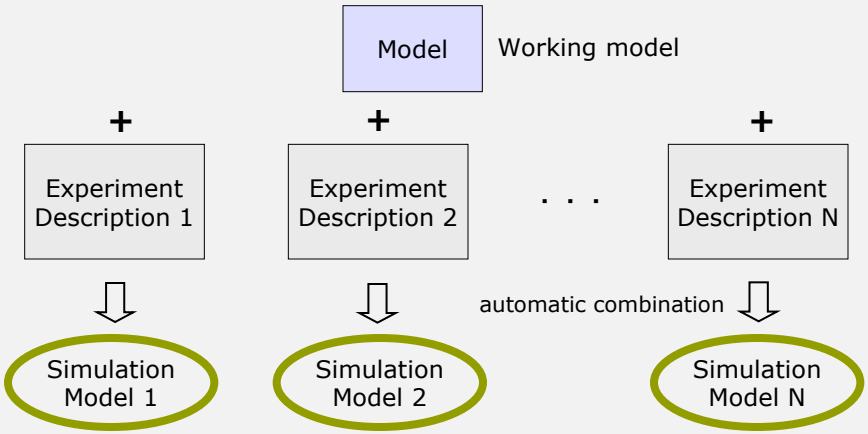

henning.schmidt@novartis.com

# Experiment Descriptions – Why?

- Typically many different biological experiments

- Coding each experimental setting in the model leads to
  - very complicated models or
  - a large number of models to keep track of





henning.schmidt@novartis.com

# Experiment Descriptions

- Experiment descriptions allow to define experimental settings insilico
- Experiment descriptions in the SBTOOLBOX2 allow to
  - change initial conditions
  - change parameter values
  - change parameter values over time
  - change state variables at given time instants
- A simple example

```
>> edit model1.txt
>> model = SBmodel('model1.txt')                    % load the model

>> edit experiment1.exp
>> experiment = SBexperiment('experiment1.exp')     % load the experiment

>> modelexp = SBmergemodexp(model,experiment)       % combine model with experiment

>> SBedit(modelexp)                                 % have a look at the new model
```

Click ”Simulate”


henning.schmidt@novartis.com

# Experiment Descriptions – How does it work?

```
********** MODEL NAME
Simple model
********** MODEL STATES
d/dt(A) = -R
d/dt(B) = R
A(0) = 1
B(0) = 0
********** MODEL PARAMETERS
k1 = 0.5
********** MODEL REACTIONS
R = k1*A
```

```
********** EXPERIMENT NAME
Simple Experiment for simple model
********** EXPERIMENT INITIAL PARAMETER AND STATE SETTINGS
k1 = 2
********** EXPERIMENT PARAMETER CHANGES

********** EXPERIMENT STATE CHANGES
time=10, A=1
```

```
********** MODEL NAME
Simple model
********** MODEL STATES
d/dt(A) = -R
d/dt(B) = R
A(0) = 1
B(0) = 0
********** MODEL PARAMETERS
k1 = 2
********** MODEL REACTIONS
R = k1*A
********** MODEL EVENTS
StateChange_1 = ge(time,10),A,1
```

The result is a new model where the experimental settings have been added


henning.schmidt@novartis.com

# Experiment Descriptions

- The general syntax is described in the `experiment2.exp` file

```
>> edit experiment2.exp
```

- An experiment object is realized as an object of class SBexperiment

# Internal Experiment Data Structure

```
>> experimentstructure = SBstruct(experiment)

experimentstructure =

            name: 'Simple Experiment'
           notes: 'Simple Experiment for Simple Model'
  paramicsettings: [1x1 struct]
 parameterchanges: [0x0 struct]
      stateevents: [1x1 struct]
```

- Import/Export of measurement data
  - CSV
  - Excel

# Measurements Export

- Export a single measurement to an Excel file

```
>> SBexportXLSmeasurement(data, 'filename')        % variable „data“ already defined
                                                   % from previous commands
```

- Exporting several measurements to an Excel file

```
>> SBexportXLSmeasurements({data, data}, 'filename2')   % just export twice the same data
```

- Export to CSV file

```
>> SBexportCSVmeasurement(data, 'filename')
```

- Import of data always by using the **SBmeasurement** command

```
>> data = SBmeasurement('filename.csv') % imports the CSV file
>> datacellarray = SBmeasurement('filename2.xls')   % imports the Excel data file
```

- Import/Export of experiment descriptions

# Experiment Descriptions

- Export of experiment descriptions

```
>> SBcreateEXPfile(experiment, 'experimentfile') % "experiment" defined from before
```

- Import of experiment description files

```
>> exp = SBexperiment('experimentfile.exp')
```


henning.schmidt@novartis.com

# Tutorial Outline

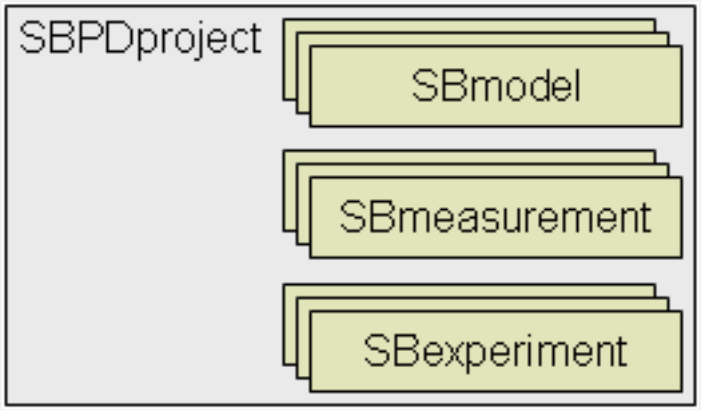
- **General introduction to the SBPOP Package**
- **Model definition and simulation**
- **Systems Biology relevant topics**
  - **Projects, parameter estimation, model reduction**
    - Projects
    - Parameter estimation / manual tuning / parameter fit analysis / parameter identifiability analysis
    - Model reduction
    - Simulation of projects
    - Commenting of projects
- **Pharmacometrics / Systems Pharmacology relevant topics**

- Projects
  - Modeling example

# SBPDproject

# Modeling Projects: SBPDproject

- An `SBPDproject` is a container for
  - Models
  - Measurement data
  - Experiment descriptions
  - Modeling information

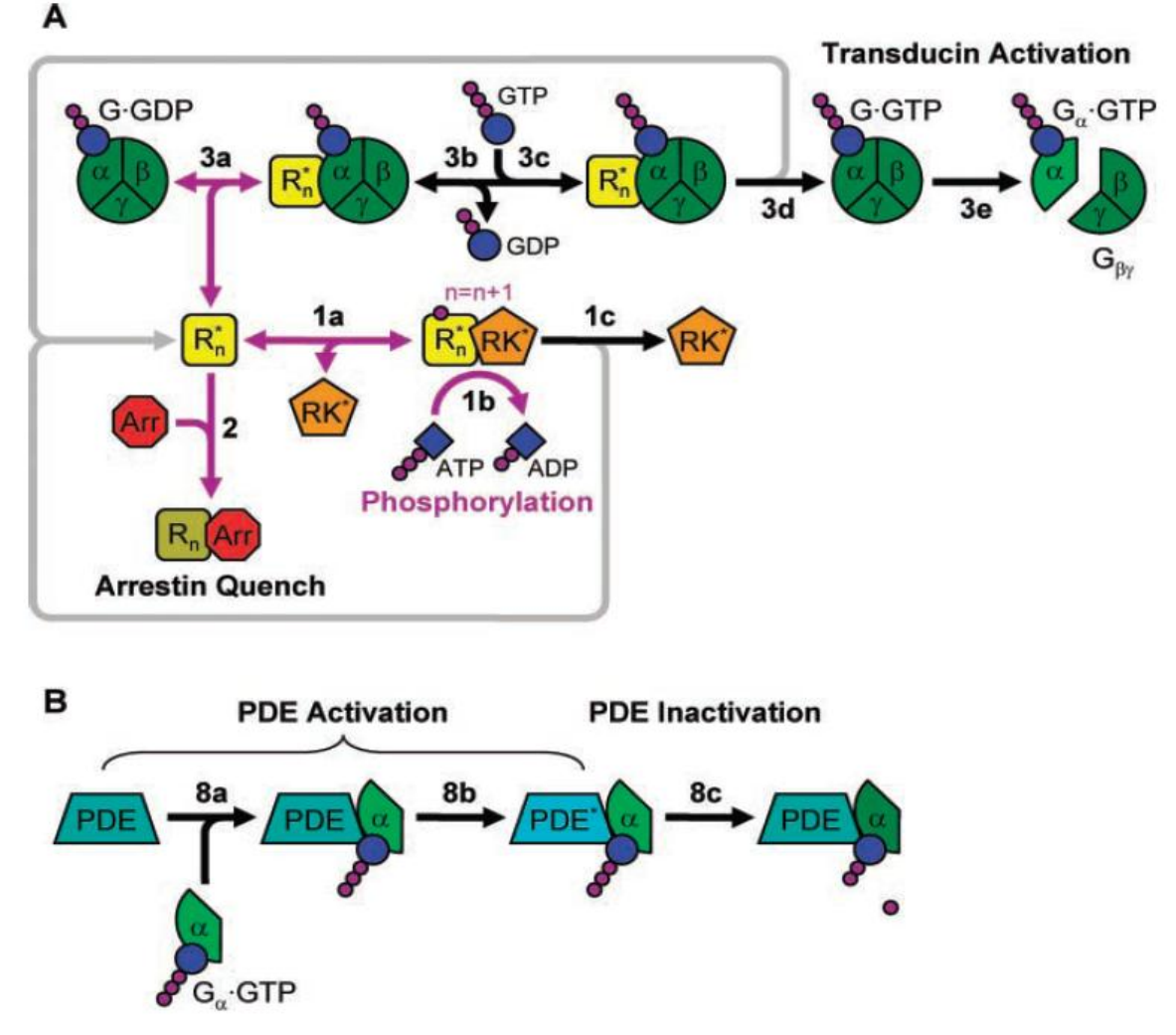



- Parameter estimation, etc., can directly be run on projects
- **When modeling a system we build a project**

**Let‘s do it!**

## The System to be Modeled:

# Photo Transduction in Rod Cells

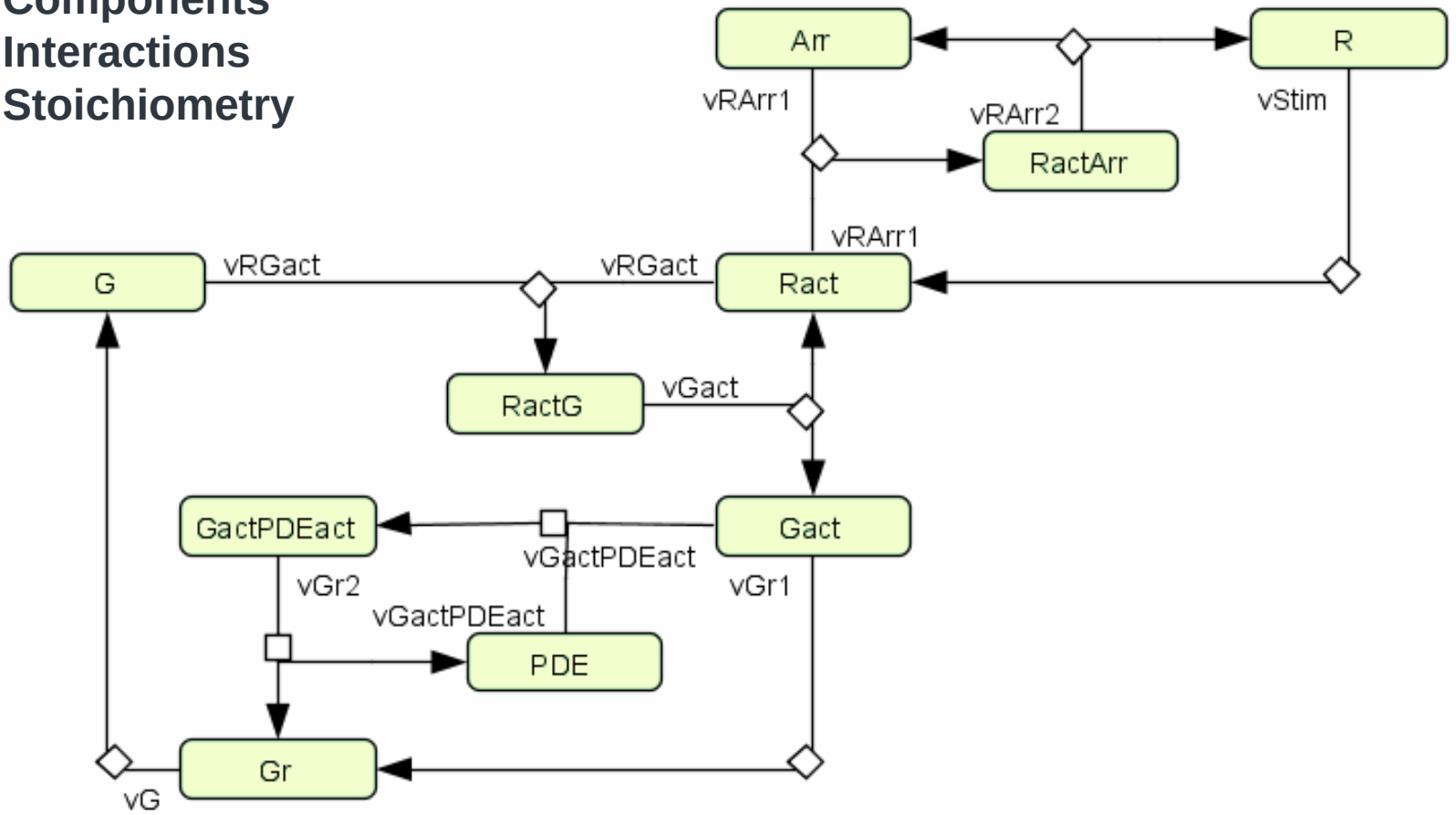



The example shown in the following is adapted for this tutorial


henning.schmidt@novartis.com

# 1) Model Network Structure

**Components**
**Interactions**
**Stoichiometry**




**Adding mathematics in graphical tools is often a pain**


henning.schmidt@novartis.com

# 2) Kinetic Rate Laws

- Export graphical model to SBML => Import to SBTOOLBOX2
- Add kinetic rate expressions in the SBmodel
  - We start simple, by assuming **mass action kinetics**
  - Only in a single reaction we assume Michaelis Menten kinetics
- Initialize parameters with guessed values
  - Nothing known about the parameters => we set all to „1“

# 3) Initial Conditions

- In this example we assume that:
  Carefully undertaken experiments showed that the number of molecules of the involved species (WT) in the inactive state (thus their total amount) is as follows:

  R = 500

  G = 3000

  Arr = 5

  **The number of PDE molecules is unknown but believed to be somewhere between 10 and 1000**

  All remaining species have an initial amount of 0

# The Initial Model

- Based on the previous information we can construct an SBmodel representation

- Change into the folder:
  „**Example Files\projectexample\phototransduction project\models**”

```
>> edit model.txtbc          % or just doubleclick on the model file
```

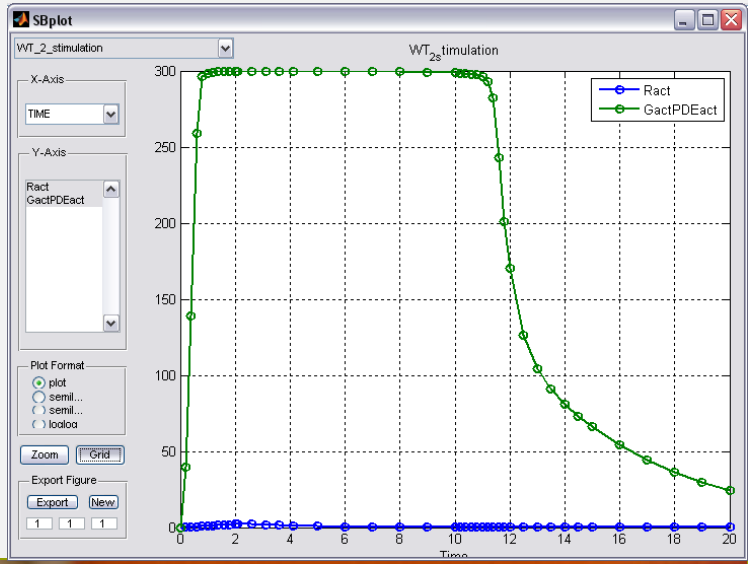
- Have a look at the model

- Realizing the pulse stimulation:

```
stimulus = piecewiseSB(magStim,le(time,durStim),0)
```

# 4) Measurement Data

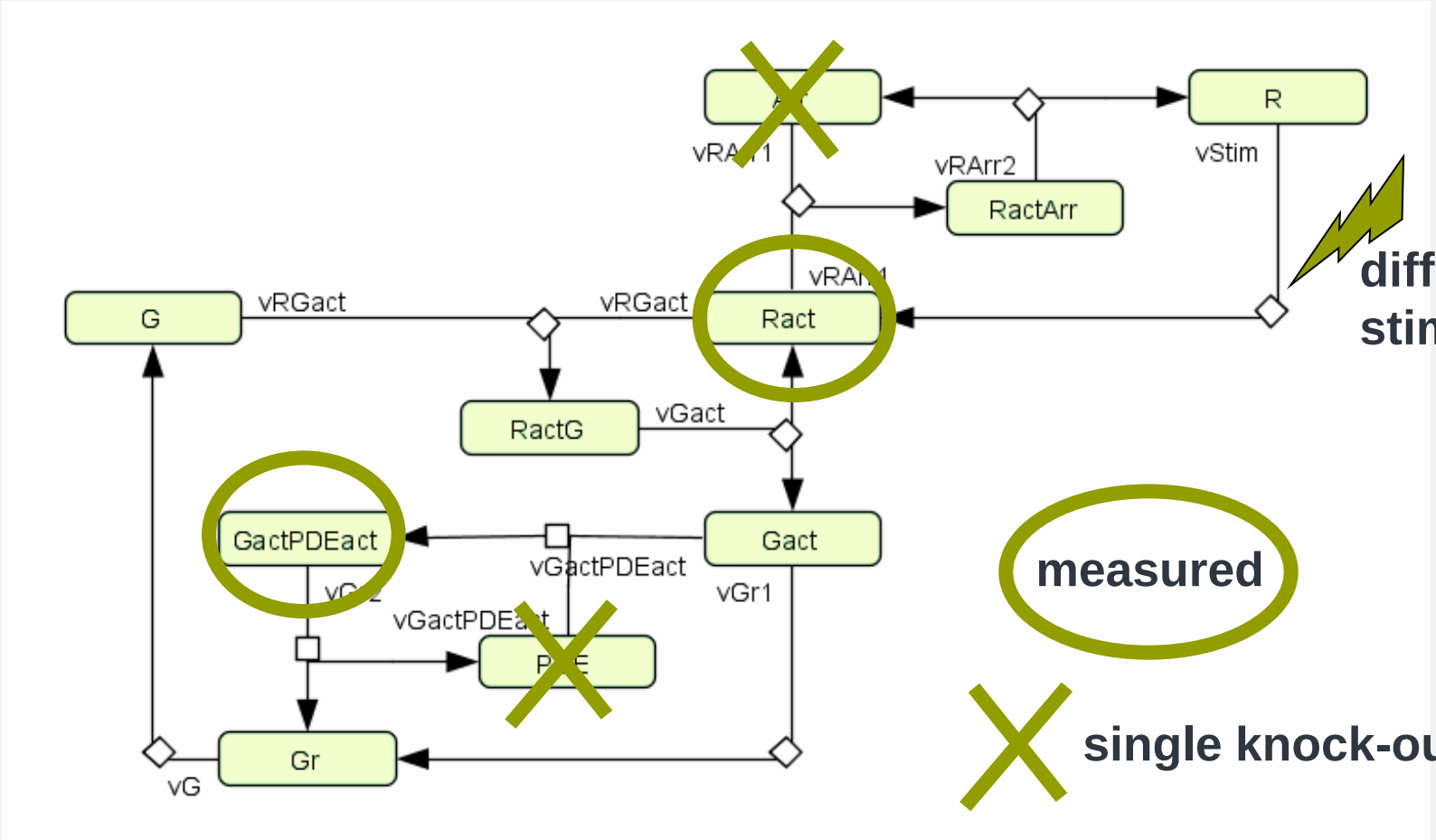
- The experimentalists gave us **4 documented CSV data files**
  - Time-series measurement data of Ract and GactPDEact
  - **Information about the biological experiments** that have been performed to obtain the data
  - The information in the documentation is the base for the specification of the experiment descriptions





henning.schmidt@novartis.com

# Experiments and Measurements




**Time-series data measured**

# 5) Constructing Experiment Descriptions

- Change into the folder:

  „Example Files\projectexample\phototransduction project\experiments\deltaArr_01”

```
>> edit deltaArr_0point1_stimulation.csv  % or right-click => “open as text”
```

- Have a look at the measurement documentation

- Define experiment description:

```
********** EXPERIMENT INITIAL PARAMETER AND STATE SETTINGS
Arr(0) = 0   % knock-out

magStim = 2
durStim = 0.1
```

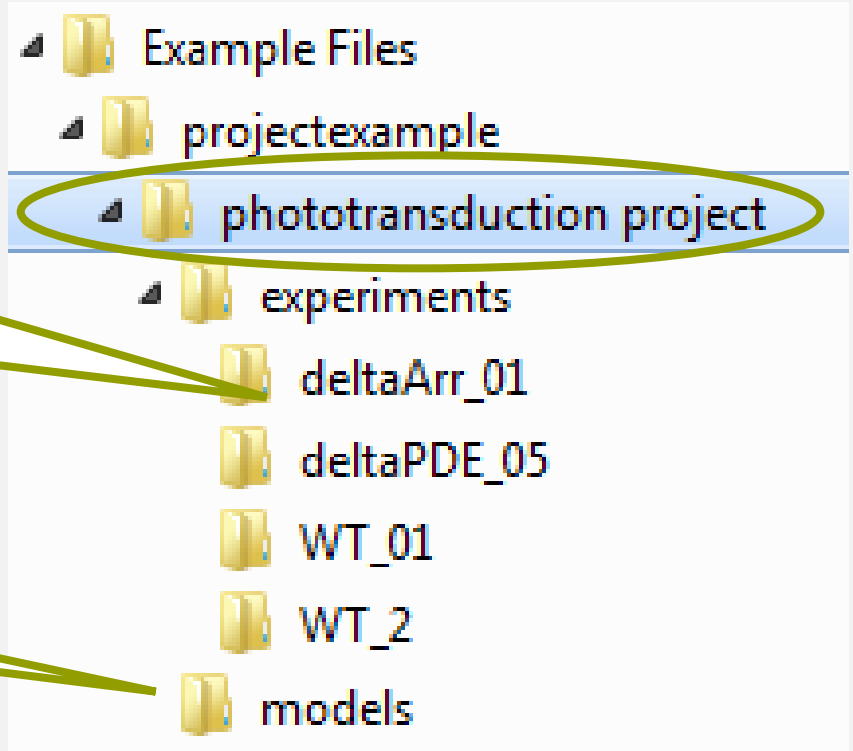
```
>> edit deltaArr_0point1_stimulation.exp  % or right-click => “open as text”
```

# Representation of SBPDprojects

- Folder structure, containing
  - Models
  - Experiments
  - Measurements

Experiment folders containing **experiment description file** and corresponding **measurement data**

Folder containing **model(s)** TEXT (ODE and/or BC) and/or SBML




**Use telling names for the experiment folders. It will help you later!**

# Projects – Import and Information

- Change into the „**example files/projectexample** “ folder
- Import of the project

```
>> sbp = SBPDproject('phototransduction project')
     SBPDproject
     ===========
     Name: phototransduction project
     Number Models:                          1
     Number Experiments:      4
     Number Measurements:     4
     Number Estimations:      0
```

- Project information

```
>> SBPDinfo(sbp)
===PROJECT INFO=================================================
Project name: phototransduction project
---NOTES---------------------------------------------------------
Project notes: This is an example project for the SBTOOLBOX2/SBPD tutorial

In this notes.txt file you can document your project. Additionally,
you can add all kinds of additional documents in the different folders.
---MODELS--------------------------------------------------------
Model 1: Project_example_model
---EXPERIMENTS---------------------------------------------------
Experiment 1: WT_0point1_stimulation
     Measurement 1: WT_0point1_stimulation
Experiment 2: WT_2_stimulation
     Measurement 1: WT_2_stimulation
Experiment 3: deltaARR_0point1_stimulation
     Measurement 1: deltaArr_0point1_stimulation
Experiment 4: deltaPDE_0point5_stimulation
     Measurement 1: deltaPDE_0point5_stimulation
---ESTIMATIONS---------------------------------------------------
0 estimations present
================================================================
```

# Adding New Experiment Results

1. Create new folder in „experiments“ folder
2. Copy measurement data in (Excel or CSV file)
3. Create an experiment description

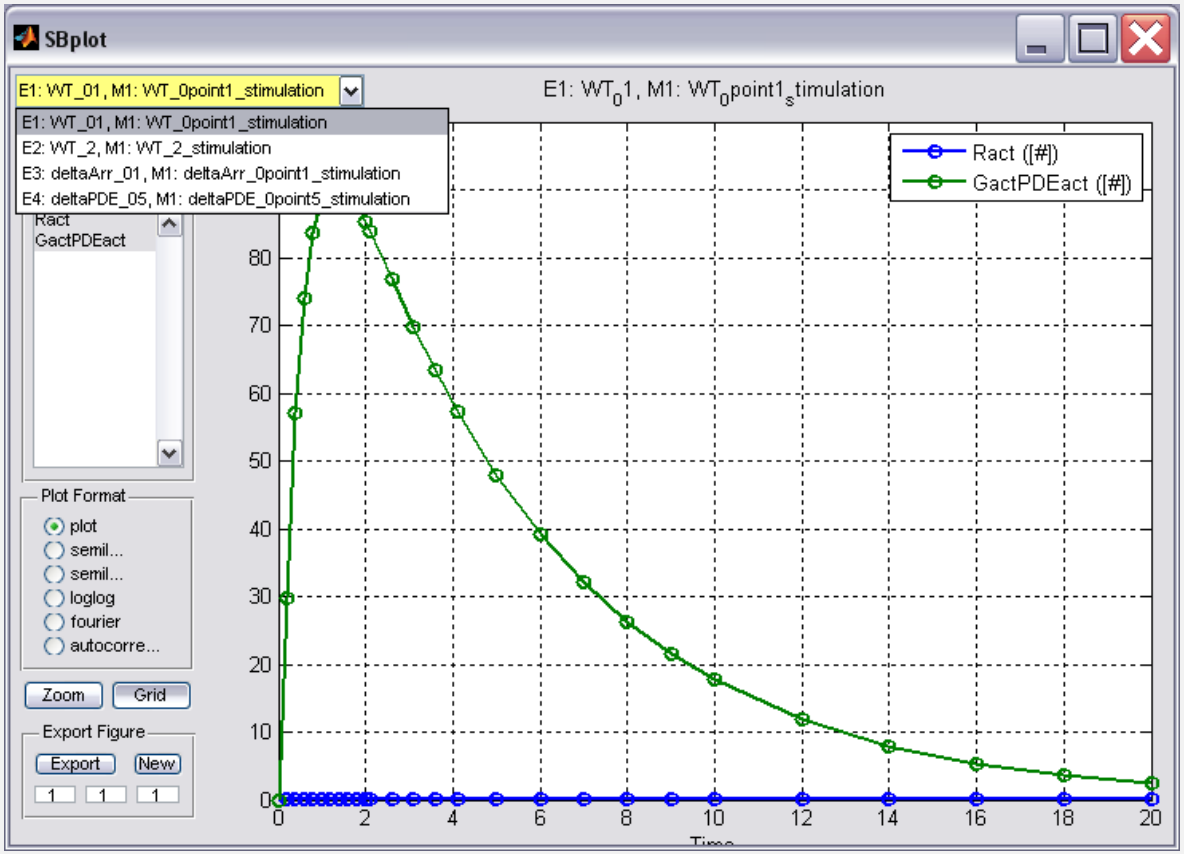
**DONE!**

- **Names in measurement files and experiment descriptions have to be the same as for the corresponding elements in the model**

# Projects – Visualize Measurement Data

- Measurement data can be plotted by

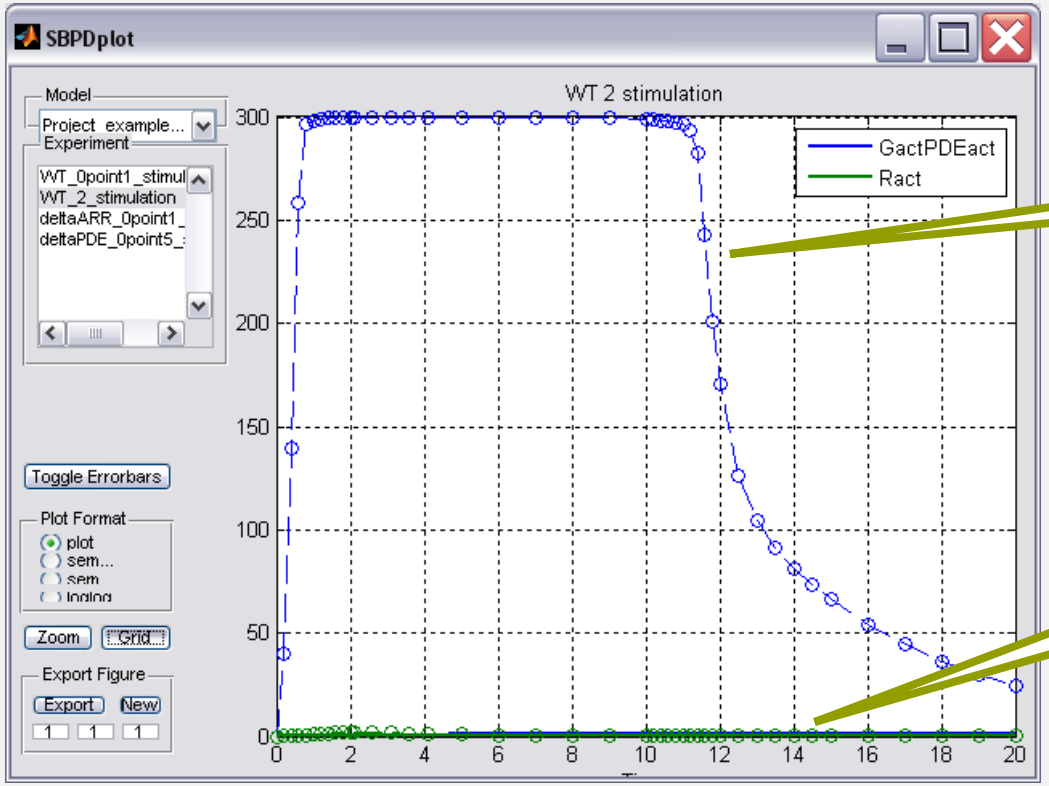
```
>> SBPDplotmeasurements(sbp)
```



# Projects – Simple Simulation

- Simulation of the experiments in the project
- Comparison with the measurement data

```
>> SBPDcomparemeasurements(sbp)
```




Measurements (-o-)

Simulations (–)

# Projects – Export

- Saving a project as single binary file

```
>> SBPDsaveproject(sbp,'projectfilename')
```

  - Saves the project in the file: **projectfilename.sbp**
  - Can be loaded again by

```
>> sbp = SBPDproject('projectfilename.sbp')
```

- Exporting the project to a folder structure

```
>> SBPDexportproject(sbp, 'projectfoldername')
```

  - During export measurement data are always saved as CSV files!

- Parameter estimation / manual tuning / parameter fit analysis / identifiability analysis

# SBPDgui++

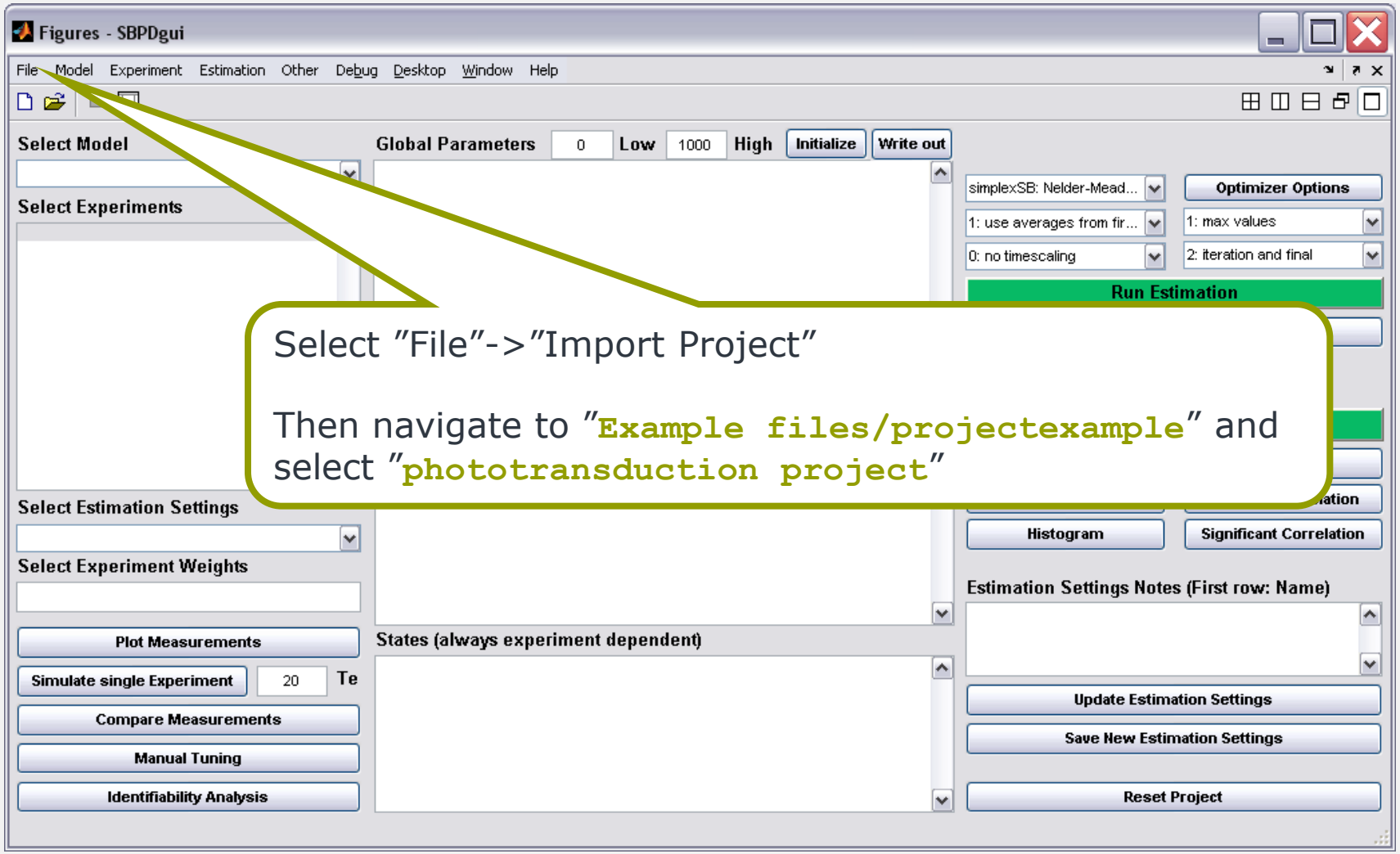
# Project is Defined – What Now?

- Several possibilities to perform parameter estimation, etc.
  - All from command line (messy)
  - Using a „`runEstimation`“ script (easy ... we will see it later)
  - Using a graphical user interface (easy)

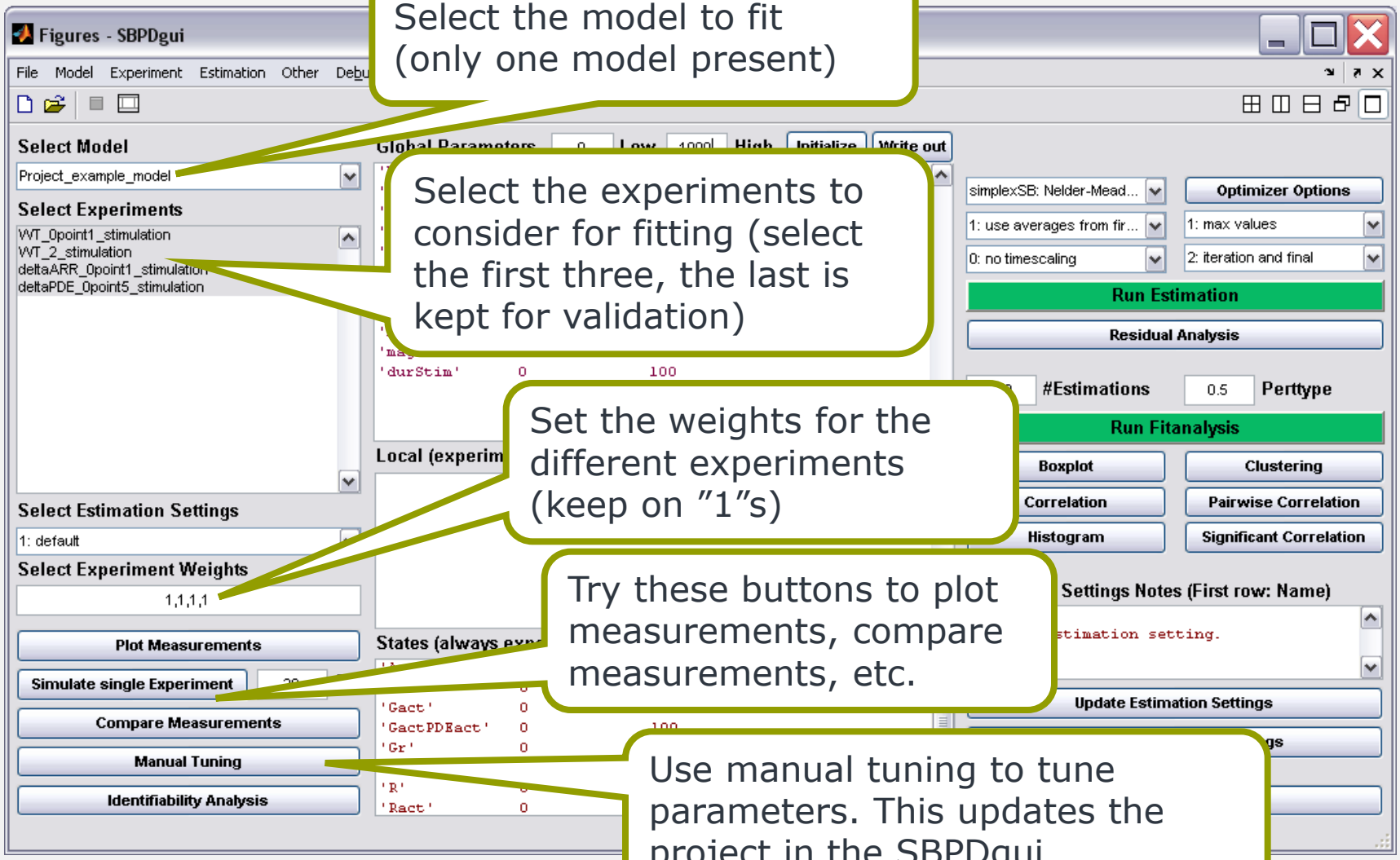
- Start the GUI

```
>> SBPDgui
```

# SBPDgui


Figures - SBPDgui
File Model Experiment Estimation Other Debug Desktop Window Help
Select Model
Global Parameters
0
Low
1000
High
Initialize
Write out
Select Experiments
simplexSB: Nelder-Mead...
Optimizer Options
1: use averages from fir...
1: max values
0: no timescaling
2: iteration and final
Run Estimation
Select Estimation Settings
Histogram
Significant Correlation
Select Experiment Weights
Estimation Settings Notes (First row: Name)
Plot Measurements
States (always experiment dependent)
Simulate single Experiment
20
Te
Compare Measurements
Update Estimation Settings
Manual Tuning
Save New Estimation Settings
Identifiability Analysis
Reset Project
Select ”File”->”Import Project”
Then navigate to ”Example files/projectexample” and select ”phototransduction project”

# SBPDgui




Select the model to fit (only one model present)

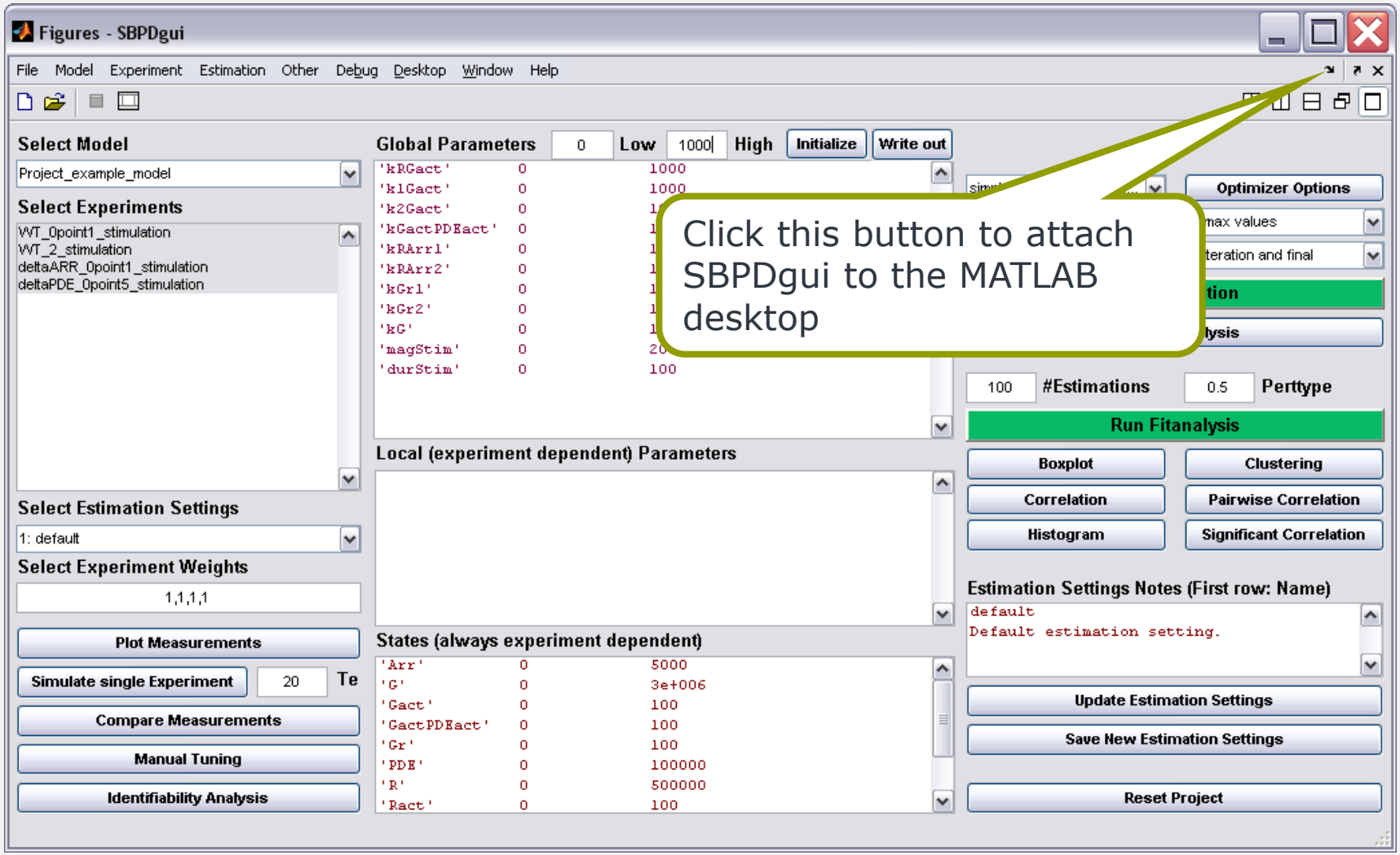
Select the experiments to consider for fitting (select the first three, the last is kept for validation)

Set the weights for the different experiments (keep on ”1”s)

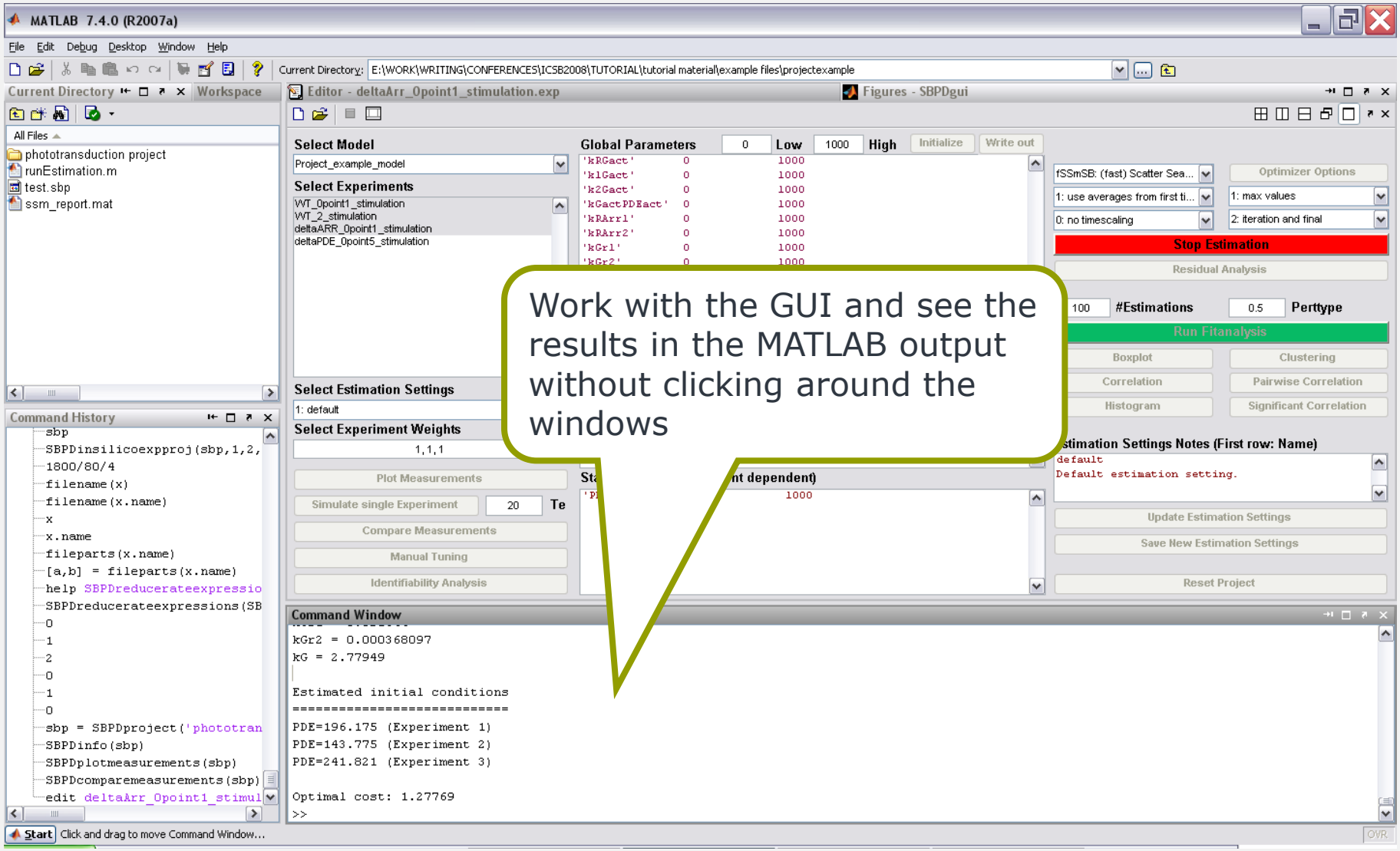
Try these buttons to plot measurements, compare measurements, etc.

Use manual tuning to tune parameters. This updates the project in the SBPDgui

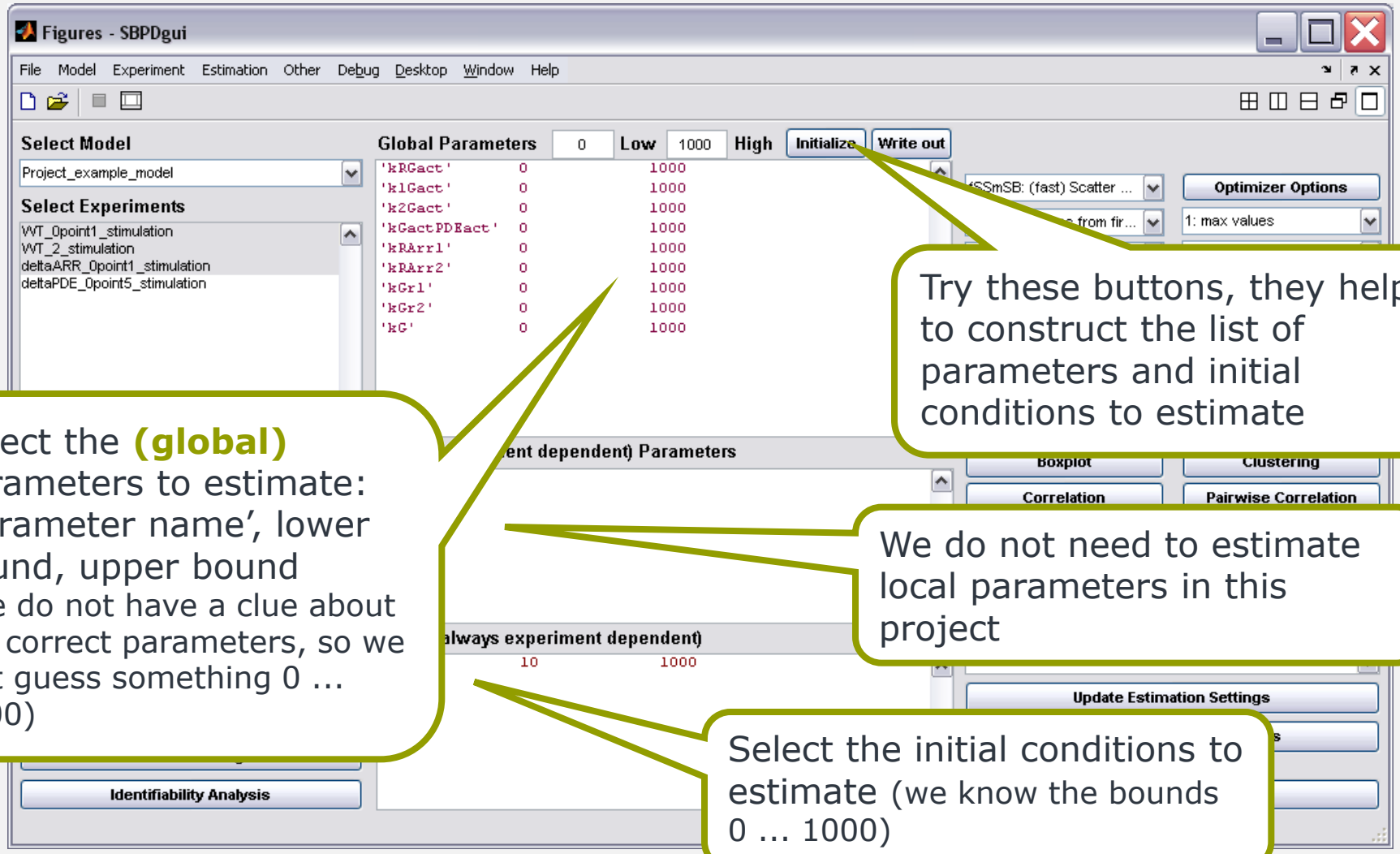
# SBPDgui


Figures - SBPDgui
File Model Experiment Estimation Other Debug Desktop Window Help
Select Model
Project_example_model
Select Experiments
WT_0point1_stimulation
WT_2_stimulation
deltaARR_0point1_stimulation
deltaPDE_0point5_stimulation
Select Estimation Settings
1: default
Select Experiment Weights
1,1,1,1
Plot Measurements
Simulate single Experiment
20
Compare Measurements
Manual Tuning
Identifiability Analysis
Global Parameters
0
Low
1000
High
Initialize
Write out
'kRGact' 0 1000
'k1Gact' 0 1000
'k2Gact' 0
'kGactPDEact' 0
'kRArr1' 0
'kRArr2' 0
'kGr1' 0
'kGr2' 0
'kG' 0
'magStim' 0
'durStim' 0 100
Local (experiment dependent) Parameters
States (always experiment dependent)
'Arr' 0 5000
'G' 0 3e+006
'Gact' 0 100
'GactPDEact' 0 100
'Gr' 0 100
'PDE' 0 100000
'R' 0 500000
'Ract' 0 100
Optimizer Options
max values
teration and final
tion
lysis
100
#Estimations
0.5
Perttype
Run Fitanalysis
Boxplot
Clustering
Correlation
Pairwise Correlation
Histogram
Significant Correlation
Estimation Settings Notes (First row: Name)
default
Default estimation setting.
Update Estimation Settings
Save New Estimation Settings
Reset Project
Click this button to attach SBPDgui to the MATLAB desktop

# SBPDgui


MATLAB 7.4.0 (R2007a)
File Edit Debug Desktop Window Help
Current Directory: E:\WORK\WRITING\CONFERENCES\ICSB2008\TUTORIAL\tutorial material\example files\projectexample
Current Directory
Workspace
All Files
phototransduction project
runEstimation.m
test.sbp
ssm_report.mat
Command History
sbp
SBPDinsilicoexpproj(sbp,1,2,
1800/80/4
filename(x)
filename(x.name)
x
x.name
fileparts(x.name)
[a,b] = fileparts(x.name)
help SBPDreducerateexpressio
SBPDreducerateexpressions(SB
0
1
2
0
1
0
sbp = SBPDproject('phototran
SBPDinfo(sbp)
SBPDplotmeasurements(sbp)
SBPDcomparemeasurements(sbp)
edit deltaArr_Opoint1_stimul
Editor - deltaArr_Opoint1_stimulation.exp
Figures - SBPDgui
Select Model
Project_example_model
Select Experiments
WT_0point1_stimulation
WT_2_stimulation
deltaARR_0point1_stimulation
deltaPDE_0point5_stimulation
Select Estimation Settings
1: default
Select Experiment Weights
1,1,1
Plot Measurements
Simulate single Experiment
20
Compare Measurements
Manual Tuning
Identifiability Analysis
Global Parameters
0
Low
1000
High
Initialize
Write out
'kRGact' 0 1000
'klGact' 0 1000
'k2Gact' 0 1000
'kGactPDEact' 0 1000
'kRArr1' 0 1000
'kRArr2' 0 1000
'kGr1' 0 1000
'kGr2' 0 1000
fSSmSB: (fast) Scatter Sea...
Optimizer Options
1: use averages from first ti...
1: max values
0: no timescaling
2: iteration and final
Stop Estimation
Residual Analysis
100
#Estimations
0.5
Perttype
Run Fitanalysis
Boxplot
Clustering
Correlation
Pairwise Correlation
Histogram
Significant Correlation
Estimation Settings Notes (First row: Name)
default
Default estimation setting.
Update Estimation Settings
Save New Estimation Settings
Reset Project
Command Window
kGr2 = 0.000368097
kG = 2.77949
Estimated initial conditions
PDE=196.175 (Experiment 1)
PDE=143.775 (Experiment 2)
PDE=241.821 (Experiment 3)
Optimal cost: 1.27769
>>
Start
Click and drag to move Command Window...
Work with the GUI and see the results in the MATLAB output without clicking around the windows

# SBPDgui



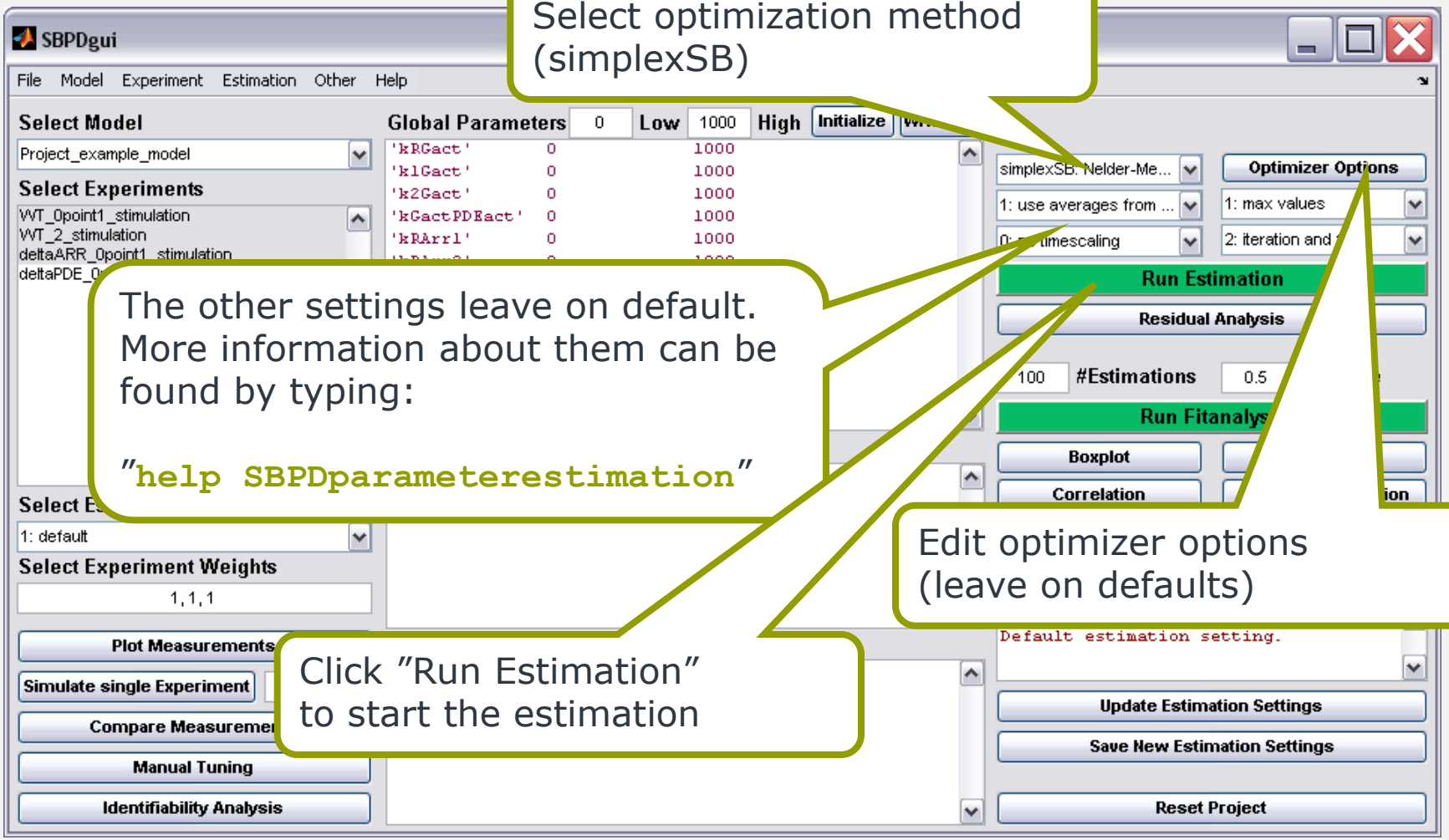

Try these buttons, they help to construct the list of parameters and initial conditions to estimate

Select the **(global)** parameters to estimate: 'parameter name', lower bound, upper bound
(We do not have a clue about the correct parameters, so we just guess something 0 ... 1000)

We do not need to estimate local parameters in this project

Select the initial conditions to estimate (we know the bounds 0 ... 1000)

# SBPDgui





henning.schmidt@novartis.com

# Parameter estimation

# ≠

# Running an optimization and wait

# Parameter Estimation

- Different optimization methods are useful for different problems
- Many options for optimization methods => translation of **parameter estimation** into **optimizer options estimation**

- Simple guideline
  - Start with a local optimimzation method and see how good it gets
  - Restart optimization with a global method (here use fSSmSB if you have installed the SSm GO toolbox, otherwise use, e.g., pswarmSB)
  - Iterate between optimization and checking the results using „Compare Measurements“ or „Manual Tuning“
  - „Simulate Single Experiment“ helps in understanding what happens to the unmeasured parts in the model

# Parameter Estimation 1st Preliminary Result

```
Estimated parameters
====================
kRGact = 0.76107
k1Gact = 999.989
k2Gact = 0.537476
kGactPDEact = 485.473
kRArr1 = 0.109028
kRArr2 = 12.9209
kGr1 = 0.0549069
kGr2 = 1.47609
kG = 0.466008

Estimated initial conditions
============================
PDE=115.492 (Experiment 1)
PDE=296.076 (Experiment 2)
PDE=921.481 (Experiment 3)

Optimal cost: 0.036001
```

VERY close to the upper bound => lets increase the upper bound to 10000

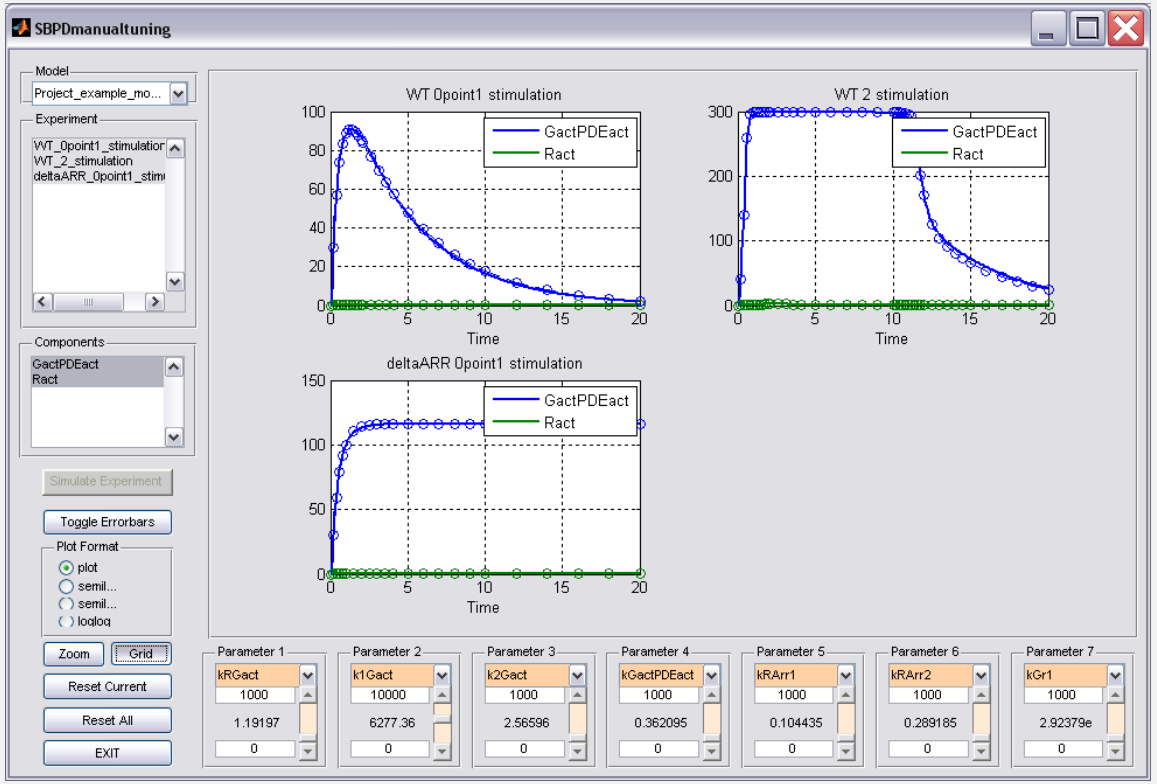
=> restart optimization

(preferably using first a local method: simplexSB, alternating with fSSmSB)

# Parameter Estimation 2nd Preliminary Result

```
Estimated parameters
====================
kRGact = 1.19197
k1Gact = 6277.36
k2Gact = 2.56596
kGactPDEact = 0.362095
kRArr1 = 0.104435
kRArr2 = 0.289185
kGr1 = 2.92379e-018
kG = 1.40081

Estimated initial conditions
============================
PDE=122.636 (Experiment 1)
PDE=300.322 (Experiment 2)
PDE=141.934 (Experiment 3)
```

**Optimal cost: 0.000615584**

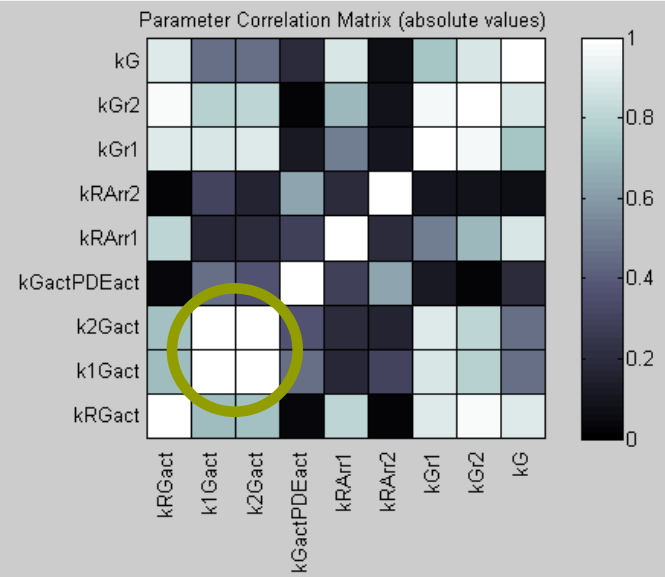



Use „Manual Tuning“ or „Compare Measurements“
=> Fit seems acceptably nice

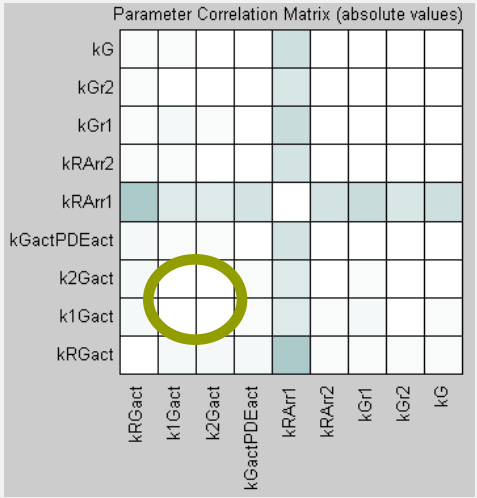
# A Posteriori Identifiability Analysis

- Select one experiment at a time and click the „Identifiability Analysis“ button

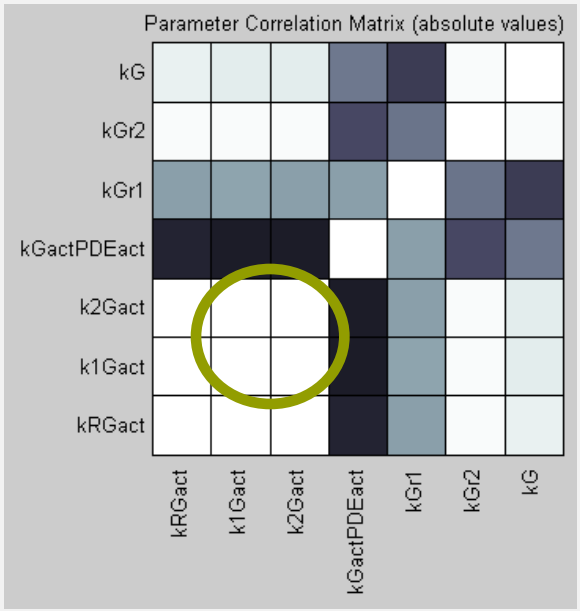
Experiment 1




Experiment 2




Experiment 3

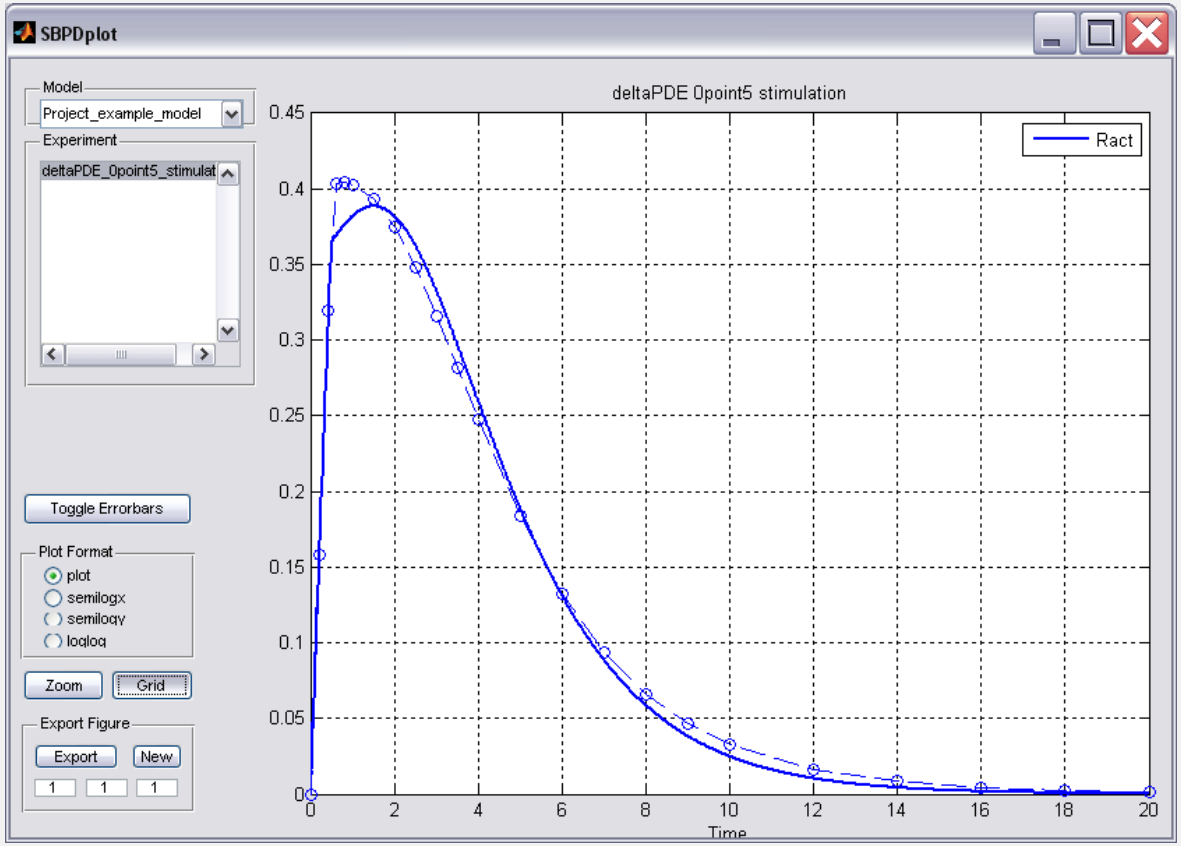




henning.schmidt@novartis.com

# Validation of Models‘ Predictiveness

- Simulate the last experiment, which was not used for estimation



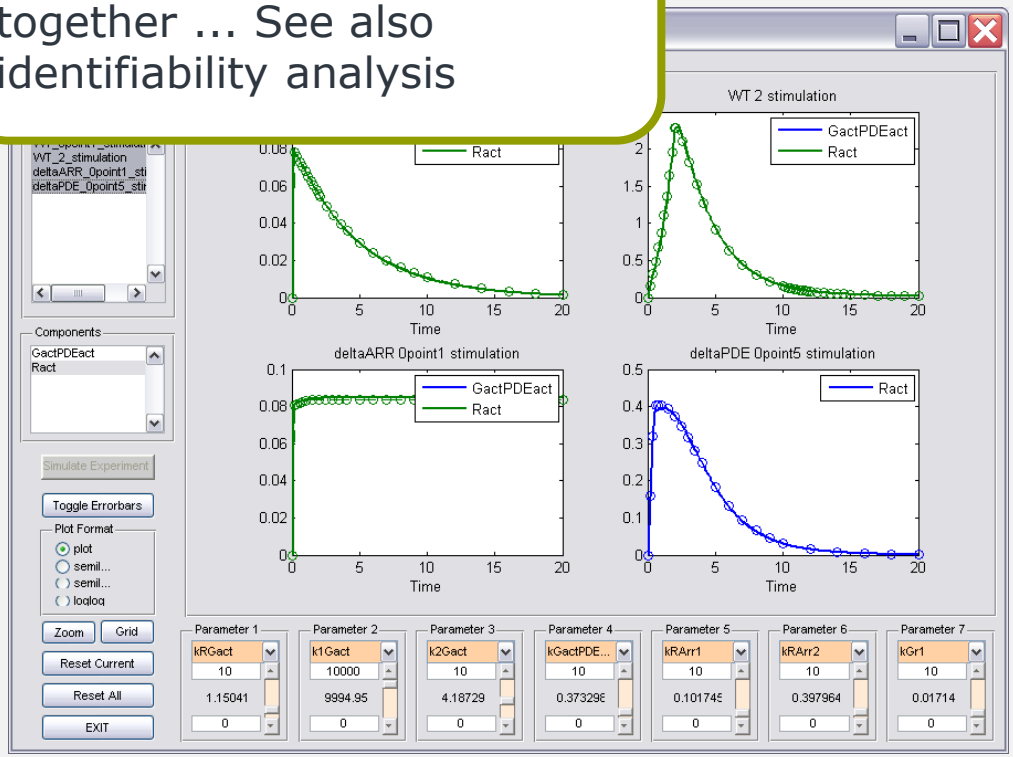

- Not perfect, but we have seen worse in other projects ☺

# Don't Waste Measurement Data

- One we have seen that the model is able to predict our validation data we should use these data to improve the model
- Reduce the upper bounds according to the current optimized values
- Again use simplexSB an

These two seem to be correlated, since increasing together ... See also identifiability analysis

```
Estimated parameters
====================
kRGact = 1.15041
k1Gact = 9994.95
k2Gact = 4.18729
kGactPDEact = 0.373298
kRArr1 = 0.101745
kRArr2 = 0.397964
kGr1 = 0.0171438
kG = 2.36096

Estimated initial conditions
============================
PDE=128.758 (Experiment 1)
PDE=302.294 (Experiment 2)
PDE=152.618 (Experiment 3)
PDE=39.4611 (Experiment 4)

Optimal cost: 0.000423773
```



# Save Optimized Project

- To export the optimized project do the following
  - Select „**File**“->“**Export Project (Folder)**“
  - Navigate to the projectexample folder
  - Click OK
  - Type new name: „**phototransduction_project_optimized**”

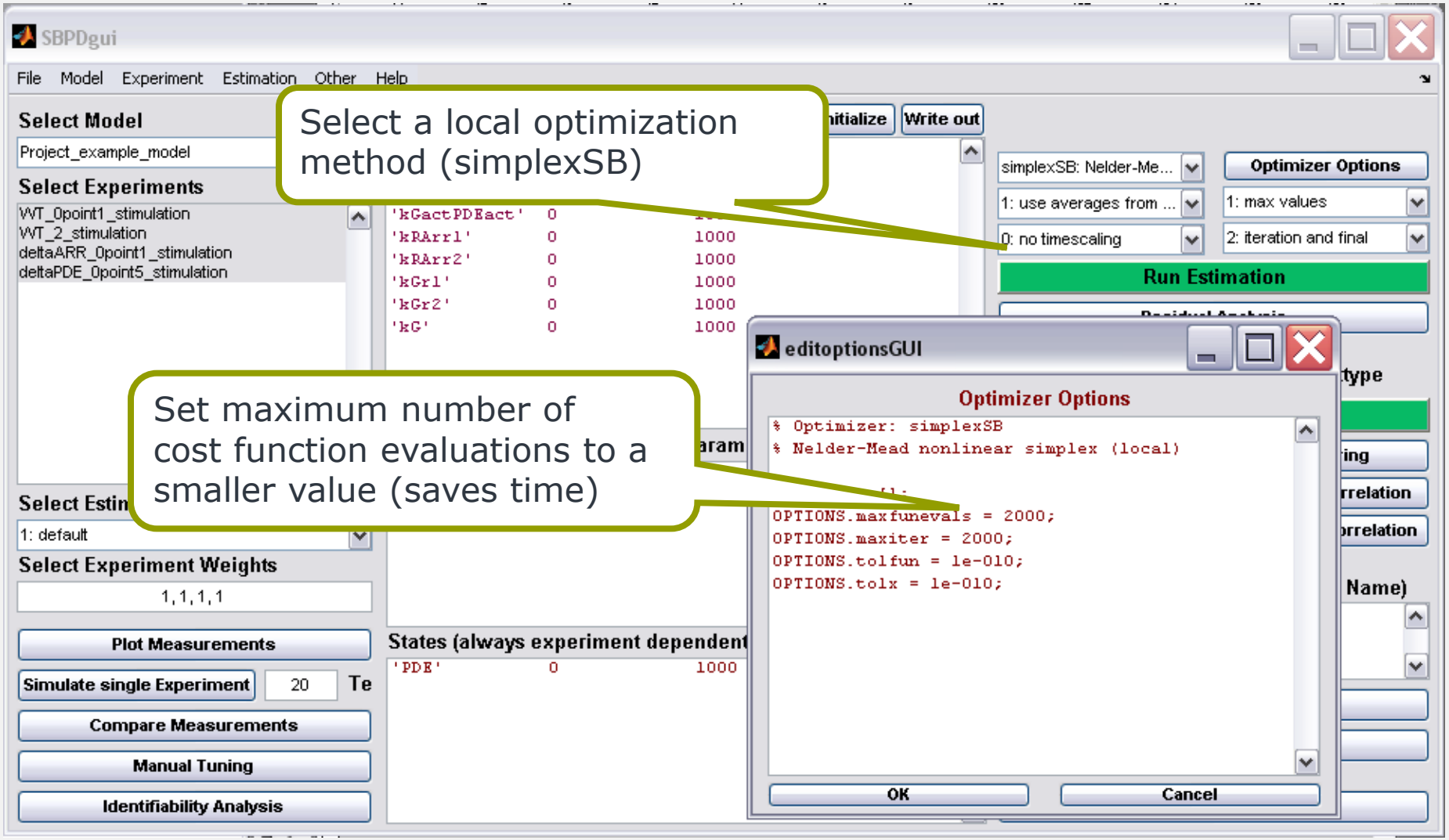
- Optimized global variables are stored in the model
- Optimized initial conditions and local variables are stored in the experiment decriptions
- Browse the newly created project folder and check the experiment descriptions and the model

# Parameter Fit Analysis

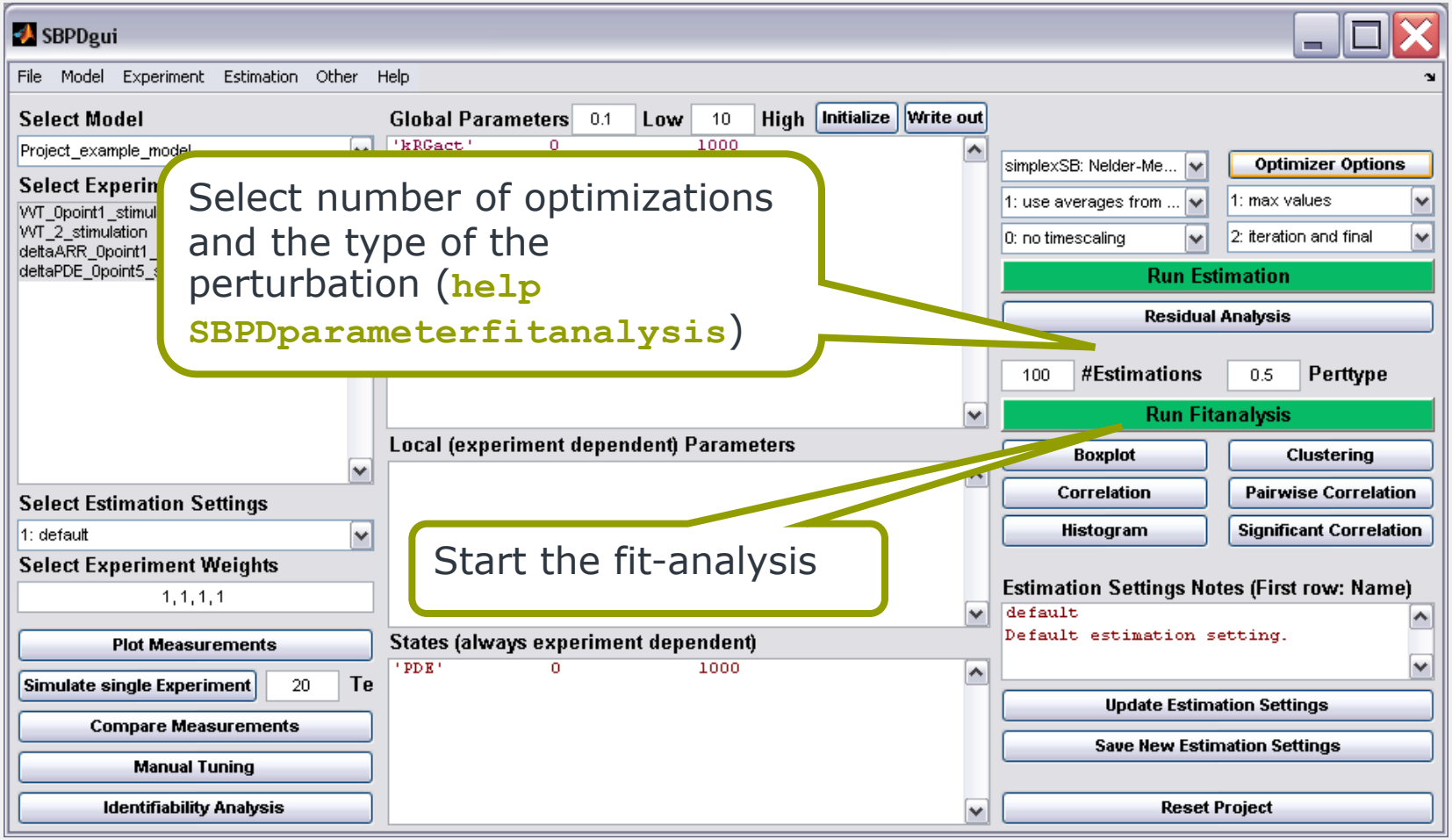
- Idea:
  1. randomly perturb starting conditions for estimation around current optimum
  2. perform optimization and collect optimal values
  3. repeat N times
  4. analyze results

- Gives information about:
  - parameter correlations
  - local optima

# Parameter Fit Analysis

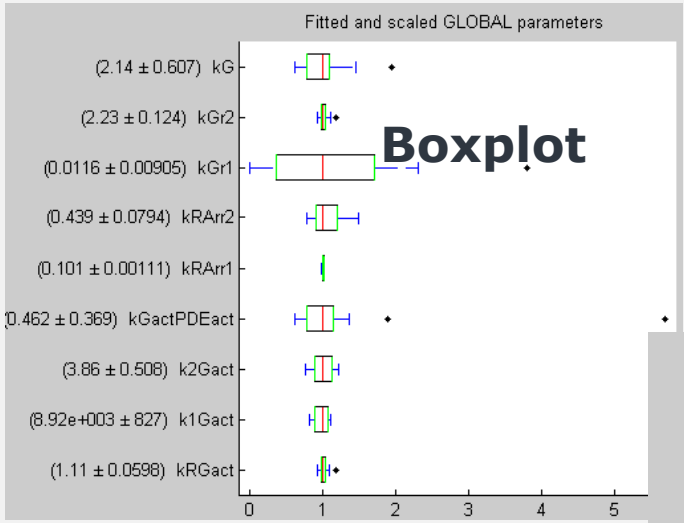


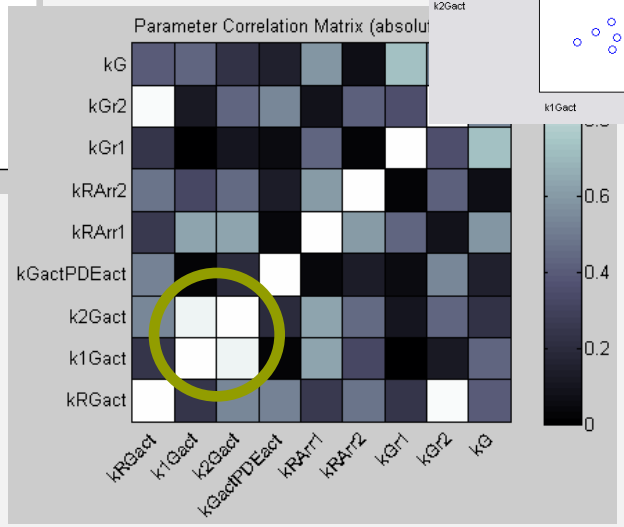
# Parameter Fit Analysis

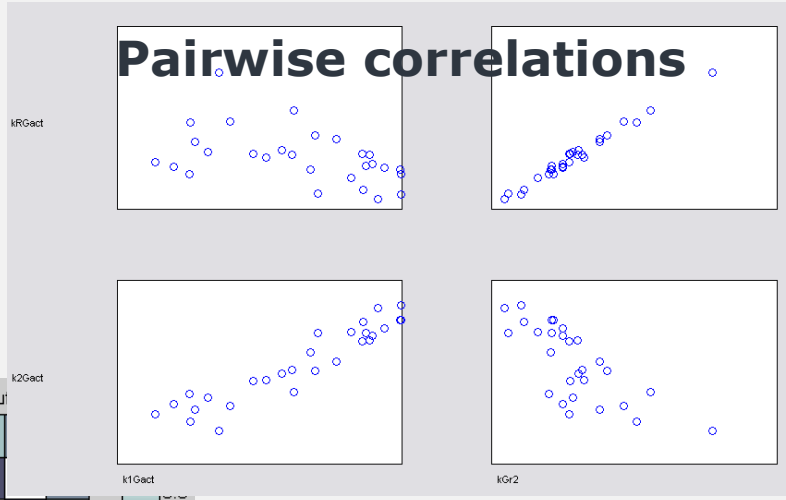


# Parameter Fit Analysis

- Correlation analysis: Even here k1Gact and k2Gact are highly correlated



# Non GUI Estimation etc.

- Parameter estimation does not require SBPDgui
- All functions are available on the command line
- Using MATLABs Cell Mode => „comfortable" estimation

- Create a **RunEstimation** script
  - In SBPDgui choose „**Other**"->"**Create RunEstimation Script**"
  - Navigate to the **projectexample** folder
  - Select **RunEstimation** as name and save

- Close SBPDgui

```
>> edit RunEstimation

% Delete line 11
% Change line 12 to sbp = SBPDproject('phototransduction_project_optimized');
```

- Model Reduction

## SBPDreducerateexpressions

Schmidt, H. et al. (2008) Complexity Reduction of Biochemical Rate Expressions, *Bioinformatics*, doi: 10.1093/bioinformatics/btn035

# Model Reduction

- Kinetic rate expressions can be very complex
- Not always they do need to be so complex in order to describe the behavior of interest

$$r = \frac{V_{max}\frac{[Glc_x]}{K_{Glc}}}{1+\frac{[Glc_x]}{K_{Glc}}+\frac{P\cdot\frac{[Glc_x]}{K_{Glc}}+1}{P\cdot\frac{[Glc]}{K_{Glc}}+1}\left(1+\frac{[Glc]}{K_{Glc}}+\frac{[G6P]}{K_{iG6P}}+\frac{[Glc]}{K_{Glc}}\frac{[G6P]}{K_{iiG6P}}\right)} - \frac{V_{max}\frac{[Glc]}{K_{Glc}}}{1+\frac{[Glc]}{K_{Glc}}+\frac{[G6P]}{K_{iG6P}}+\frac{[Glc]}{K_{Glc}}\frac{[G6P]}{K_{iiG6P}}+\frac{P\cdot\frac{[Glc]}{K_{Glc}}+1}{P\cdot\frac{[Glc_x]}{K_{Glc}}+1}\left(1+\frac{[Glc_x]}{K_{Glc}}\right)}.$$

$$r_{red} = \frac{K_4[Glc]+K_3[Glc_x]}{1+K_1[Glc][Glc_x]+K_2[G6P][Glc_x]}.$$

# Model Reduction

- The function **`SBPDreducerateexpressions`** allows to stepwise reduce complex kinetic expressions

- Here we demonstrate its use based on the previous modeling example, where we unnecessarily used a Michaelis Menten term

- A more complex example can be found in the "**`SBPD\_EXAMPLES\ModredExample`**" folder

$$r = \frac{V_{max}\frac{[Glc_x]}{K_{Glc}}}{1+\frac{[Glc_x]}{K_{Glc}}+\frac{P\cdot\frac{[Glc_x]}{K_{Glc}}+1}{P\cdot\frac{[Glc]}{K_{Glc}}+1}\left(1+\frac{[Glc]}{K_{Glc}}+\frac{[G6P]}{K_{iG6P}}+\frac{[Glc]}{K_{Glc}}\frac{[G6P]}{K_{iiG6P}}\right)} - \frac{V_{max}\frac{[Glc]}{K_{Glc}}}{1+\frac{[Glc]}{K_{Glc}}+\frac{[G6P]}{K_{iG6P}}+\frac{[Glc]}{K_{Glc}}\frac{[G6P]}{K_{iiG6P}}+\frac{P\cdot\frac{[Glc]}{K_{Glc}}+1}{P\cdot\frac{[Glc_x]}{K_{Glc}}+1}\left(1+\frac{[Glc_x]}{K_{Glc}}\right)}.$$


henning.schmidt@novartis.com

# Model Reduction

- Load the optimized project

```
>> sbp = SBPDproject('phototransduction_project_optimized')
```

- Run the model reduction

```
>> SBPDreducerateexpressions(sbp)
```

- **RESULT: Mass action kinetics is sufficient**

- More information

```
>> help SBPDreducerateexpressions    % and of course the example in the SBPD examples
```

- Simulation of projects

# Simulation of Projects

- Change into the „**Example Files/projectexample**“ folder

```
>> sbp = SBPDproject('phototransduction_project_optimized')    % import the project
```


- Compare measurements

```
>> SBPDcomparemeasurements(sbp)    % simulate all experiments and model in the project
                                   % and compare results to measurements

>> help SBPDcomparemeasurements
```

- Manual tuning

```
>> SBPDmanualtuning(sbp)    % simulate selected experiments and model in the project
                            % and compare results to measurements + allow tuning

>> help SBPDmanualtuning
```

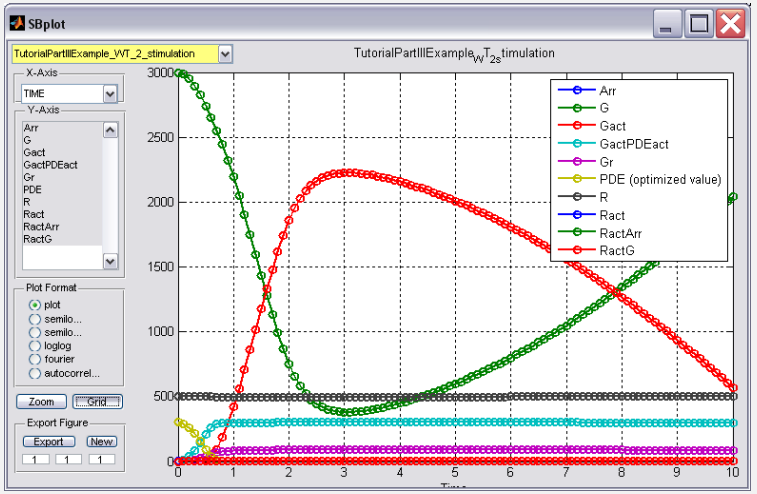
- These commands take a while to execute, since compilation is done

# Simulation of Projects

- Simulating *in silico* experiments

```
>> model = SBPDgetmodel(sbp,1)                % get first model from project sbp

>> experiment = SBPDgetexperiment(sbp,2)     % get second experiment from project sbp

>> SBPDinsilicoexp(model,experiment,[0:0.1:10])
```

- Per default the result is plotted




- Optionally, the result can be saved
  - CSV file
  - Excel file

- Or as **SBmeasurement** be returned to the workspace

```
>> help SBPDinsilicoexp
```

# Simulation of Projects

- Instead of extracting models and experiments in silico experiments can directly be performed on projects:

```
>> modelindex = 1                % first model from project sbp

>> experimentindex = 2           % second experiment from project sbp

>> SBPDinsilicoexpproj(sbp,modelindex,experimentindex,[0:2:400])    % perform experiment
```

- Per default the result is **saved in a CSV measurement file**

- Optionally, the result can be
  - displayed using SBplot
  - saved as an Excel measurement file
  - returned to the workspace as an **`SBmeasurement`**

```
>> help SBPDinsilicoexp
```

- Commenting of projects

# Commenting a Project

- A project folder can contain a `notes.txt` file
  - General information about the project
  - Focus of the project: why modeling, what to do with the model, etc.

- Every folder in a project can contain additional files. For example:
  - Word(etc.) documents in the model or experiment folder(s), describing things more in detail
  - Figures, spreadsheets, raw data files, ...

- In this way the complete information about a modeling project can be contained in the folder structure of an SBPDproject

- LIMITATION: Experiment folders should only contain Excel or CSV files that contain measurement data. However, these folders can contain additional folders in which all kinds of information can be stored.

# Documenting the Validity of a Model

- A model alone is NOT very useful
  - SBML model
  - *.txt or *.txtbc model

- A model becomes useful if the following is known about it
  - For what purpose has the model been build?
  - Which experiments have been performed to generate data for fitting and validation?
  - Under what conditions have the experiments been performed?
  - Etc.

# Documenting the Validity of a Model

- Where to store that information in the framework of the SBTOOLBOX2?
  - In the projects
    - The notes.txt can contain the purpose of the model and general assumptions
    - The model files can contain additional information
    - The experiment descriptions define the performed experiments and the conditions under which these experiments have been performed
- Don‘t exchange models! Exchange projects!

# Tutorial Outline

- **General introduction to the SBPOP Package**
- **Model definition and simulation**
- **Systems Biology relevant topics**
- **Pharmacometrics / Systems Pharmacology relevant topics**
  - **Model and dosing description**
    - Needed changes in SBmodels to allow for simulation of dosing scenarios
    - Simulation of dosing scenarios

# Tutorial Outline

- **General introduction to the SBPOP Package**
- **Model definition and simulation**
- **Systems Biology relevant topics**
- **Pharmacometrics / Systems Pharmacology relevant topics**
  - **Datasets**
    - Import, export, conversion
    - Analysis (plotting capabilities)

# Tutorial Outline

- **General introduction to the SBPOP Package**
- **Model definition and simulation**
- **Systems Biology relevant topics**
- **Pharmacometrics / Systems Pharmacology relevant topics**
  - **Interface to Monolix**
    - Creation of MLXTRAN Models
    - Creation and execution of MLXTRAN Projects
    - Import of Monolix results


henning.schmidt@novartis.com

# Tutorial Outline

- General introduction to the SBPOP Package
- Model definition and simulation
- Systems Biology relevant topics
- Pharmacometrics / Systems Pharmacology relevant topics
  - PopPK/PD workflow

# Tutorial Outline

- General introduction to the SBPOP Package
- Model definition and simulation
- Systems Biology relevant topics
- Pharmacometrics / Systems Pharmacology relevant topics
  - Clinical trial simulations

# Tutorial Outline

- General introduction to the SBPOP Package
- Model definition and simulation
- Systems Biology relevant topics
- Pharmacometrics / Systems Pharmacology relevant topics
  - More complex modeling
    - PBPK
    - Antibody modeling

# Tutorial Goal: „You should now be able to“

- **Systems Biology relevant topics**
  - Set up your own parameter estimation project
    - Model description
    - Measurement data
    - Experiment descriptions
  - Perform parameter estimation, identifiability analysis, etc.
  - Analyze the resulting model

- **Pharmacometrics / Systems Pharmacology relevant topics**
  - Define arbitrary ODE based traditional and/or mechanistic, PK, PKPD, PBPK models
  - Perform
    - NLME estimation of parameters
    - Clinical trial simulations

# THE END

**Thank you for your participation and interest!**


henning.schmidt@novartis.com