

PottersWheel 1.4

Multi-Experiment Fitting Toolbox

Thomas Maiwald
Freiburg Center for Data Analysis and Modeling
University of Freiburg, Germany
maiwald@fdm.uni-freiburg.de

July 21, 2007

Contents

1	Quickstart	11
1.1	General approach and recommendations	11
1.2	Example models	12
1.2.1	Model 1: Four variables, closed system	13
1.2.2	Model 2: Four variables, open system	14
1.2.3	Model 3: Positive and negative feedbacks	15
1.2.4	Model 4: Rule based modeling	17
1.3	Example session	19
1.4	Example script	24
2	Model creation	29
2.1	Naming conventions	29
2.2	Reaction based models	30
2.2.1	General information	30
2.2.2	Dynamic variables and start values	30
2.2.3	Reactions	31
2.2.4	Reaction kinetics	32
2.2.5	Delay reactions	32
2.2.6	Compartments	33
2.2.7	Dynamic parameters	33
2.2.8	Driving inputs	34
2.2.9	Sampling	34
2.2.10	Observables	35
2.2.11	Scaling parameters	36
2.2.12	Derived variables	36
2.2.13	Derived parameters	37
2.2.14	Algebraic equations and rules	37
2.2.15	Example: JakStat5 signaling pathway	38
2.3	Using the model wizard	39
2.4	ODE based models	39
2.4.1	Automatic reconstruction of reaction schemes	39
2.5	SBML based models	39
2.6	Power law models	42
2.7	Model visualization	42

3	Rule based modeling and combinatorial complexitiy	43
4	Manually investigating model properties	45
4.1	PottersWheel Equalizer	45
4.2	PottersWheel Input Designer	47
5	IDSO	49
5.1	Instructions	49
5.2	Dynamical parameters	49
5.3	Start values	50
5.4	Observation functions	51
6	Saving and loading simulated and experimental data	53
6.1	Format of data files	53
6.1.1	User-specific information	53
6.1.2	Driving functions	53
6.1.3	Column names	54
6.1.4	Data matrix and complete example	54
6.2	Saving of simulated data	55
6.3	Loading of external data and mapping of observables	55
6.4	Investigating external data sets efficiently	56
6.4.1	Data Viewer	56
6.4.2	Plots including driving inputs	56
7	Fitting	59
7.1	Introduction	59
7.2	Optimization strategies: line search and trust region	59
7.2.1	Line search	60
7.2.2	Trust region	61
7.3	Nonlinear least-square problems	62
7.3.1	The Gauss-Newton method	63
7.3.2	The Levenberg-Marquardt method	63
7.3.3	Large-residual problems	63
7.4	Simulated annealing	63
7.5	Genetic algorithm	63
7.6	Optimization function	64
7.7	Parameter limits	64
7.8	Confidence intervals	64
7.9	Fitting in logarithmic parameter space	64
7.10	Single and multi-experiment fitting	65
7.11	Single fit and fit sequences	65
7.12	Boosted fit	66

8	Fit analysis	67
8.1	Fit sequence based analysis	67
8.1.1	Best fit selection	68
8.1.2	Histograms	69
8.1.3	Boxplot	70
8.1.4	Correlation matrix	71
8.1.5	Significant correlations	72
8.1.6	Detailed significant correlations	73
8.1.7	Principal Component Analysis (PCA)	74
8.1.8	Biplot	75
8.1.9	Hierarchical clustering	76
8.2	Derived parameters	76
8.3	Residual analysis	77
8.3.1	Residuals over time	78
8.3.2	Residuals embedded	79
8.3.3	Autocorrelation of residuals	80
8.3.4	Histogram	81
8.3.5	QQ-Plot	82
8.3.6	Residuals against predicted values	83
8.3.7	Predicted against observed values	84
9	Reporting: PDF Latex, MS Word, HTML	85
10	Installation	87
10.1	Requirements	87
10.2	Installation	87
10.2.1	Windows	87
10.2.2	Linux, Macintosh and Unix	89
10.3	Updating	89
11	Configuration settings	91
12	Reaction kinetics	93
12.1	General considerations	93
12.1.1	Change of units	93
12.2	List of reaction kinetics	94
12.2.1	Constant flux	94
12.2.2	Mass action kinetics, irreversible	94
12.2.3	Mass action kinetics, reversible	95
12.2.4	Henri-Michaelis-Menten	95
12.2.5	Reversible Michaelis-Menten	95
12.2.6	Reversible Michaelis-Menten with Haldane adjustment	95
12.2.7	Hill cooperative kinetics	95
12.2.8	Reversible Hill kinetics	96
12.2.9	Irreversible reaction with competitive inhibition	96

13 Shortcuts and functions	97
13.1 Shortcuts	97
13.2 Alphabetical function list	98
13.2.1 PottersWheel	98
13.2.2 pc	98
13.2.3 pe	99
13.2.4 pw	99
13.2.5 pwAddC	99
13.2.6 pwAddData	99
13.2.7 pwAddDynamicalParameters	99
13.2.8 pwAddInstructions	99
13.2.9 pwAddK	100
13.2.10 pwAddModel	100
13.2.11 pwAddObservations	100
13.2.12 pwAddP	100
13.2.13 pwAddR	101
13.2.14 pwAddRule	101
13.2.15 pwAddS	101
13.2.16 pwAddStartValues	101
13.2.17 pwAddU	101
13.2.18 pwAddX	101
13.2.19 pwAddY	101
13.2.20 pwAddZ	102
13.2.21 pwAnalysisOfResiduals	102
13.2.22 pwArrange	102
13.2.23 pwAutoResetQRNG	102
13.2.24 pwCheckSystem	102
13.2.25 pwClear	102
13.2.26 pwCloseEqualizer	102
13.2.27 pwCloseInputDesigner	103
13.2.28 pwCloseMainGUI	103
13.2.29 pwCombine	103
13.2.30 pwConfiguration	103
13.2.31 pwCreateModelGUI	103
13.2.32 pwDataAndFits	103
13.2.33 pwDelete	103
13.2.34 pwDisturb	103
13.2.35 pwDraw	104
13.2.36 pwDuplicate	104
13.2.37 pwEdit	104
13.2.38 pwEqualizer	104
13.2.39 pwF1	104
13.2.40 pwF2	104
13.2.41 pwF3	104
13.2.42 pwF4	104

13.2.43 pwF5	105
13.2.44 pwFit	105
13.2.45 pwFitBoost	105
13.2.46 pwFitDynamicParameters	105
13.2.47 pwFitHistory	105
13.2.48 pwFitHistoryDeleteFits	105
13.2.49 pwFitHistoryGetFitGroups	105
13.2.50 pwFitReport	106
13.2.51 pwFitScalingParameters	106
13.2.52 pwFitSequenceAnalysis	106
13.2.53 pwFitSequencePlots	107
13.2.54 pwFitSequenceResults	107
13.2.55 pwFitSettings	107
13.2.56 pwFitStartValues	107
13.2.57 pwFixParameters	108
13.2.58 pwForcedCompilation	108
13.2.59 pwGetConfig	108
13.2.60 pwGetChisqAndN	108
13.2.61 pwGetDrivingFunction	108
13.2.62 pwGetFitParameters	108
13.2.63 pwGetFitParametersByID	108
13.2.64 pwGetIDs	109
13.2.65 pwGetListOfParameters	109
13.2.66 pwGetNumberOfCouples	109
13.2.67 pwGetParameterValues	109
13.2.68 pwGetPlottingData	109
13.2.69 pwGetQRNGIndex	110
13.2.70 pwGraph	110
13.2.71 pwHelp	110
13.2.72 pwIDSO	110
13.2.73 pwInfo	110
13.2.74 pwInfo2	110
13.2.75 pwInputDesignerGUI	110
13.2.76 pwInstall	110
13.2.77 pwLoadConfigFromMFunction	111
13.2.78 pwLoadRepository	111
13.2.79 pwModelInfo	111
13.2.80 pwODEsToReactions	111
13.2.81 pwOpenODE	111
13.2.82 pwParseAndShow	111
13.2.83 pwPlot	111
13.2.84 pwPropertiesOfDataFile	112
13.2.85 pwReload	112
13.2.86 pwReportAppendGraphsAndReactions	112
13.2.87 pwReportAppendLastStep	112

13.2.88	pwReportClear	112
13.2.89	pwReportGUI	112
13.2.90	pwReportGenerate	112
13.2.91	pwReportNew	112
13.2.92	pwReset	113
13.2.93	pwSBML2PW	113
13.2.94	pwSaveConfigToMFunction	113
13.2.95	pwSaveFigures	113
13.2.96	pwSavePlottingData	113
13.2.97	pwSaveRepository	113
13.2.98	pwSaveSelectedModelsWithFittedParValues	113
13.2.99	pwScale	114
13.2.100	pwScaleAndMergeDataSets	114
13.2.101	pwSelect	114
13.2.102	pwSelectCurrentData	114
13.2.103	pwSensitivityAnalysis1D	114
13.2.104	pwSensitivityAnalysis2D	114
13.2.105	pwSensitivityAnalysis3D	115
13.2.106	pwSetConfig	115
13.2.107	pwSetFitParameterValuesByID	115
13.2.108	pwSetFitSettings	115
13.2.109	pwSetIntegrationStartTime	115
13.2.110	pwSetLogFitting	116
13.2.111	pwSetOptimizer	116
13.2.112	pwSetParameterValuesByID	116
13.2.113	pwSetPlottingState	116
13.2.114	pwSetQRNGIndex	116
13.2.115	pwShortcut	116
13.2.116	pwShowFitting	117
13.2.117	pwShowGraph	117
13.2.118	pwShowGraphAsPNG	117
13.2.119	pwShowHelp	117
13.2.120	pwShowODE	117
13.2.121	pwShowShortcuts	117
13.2.122	pwSim	117
13.2.123	pwSplitStimuli	117
13.2.124	pwUndo	117
13.2.125	pwUpdateRepository	118

Foreword

Systems Biology Recently, new measurement techniques at a molecular and cellular level have been established, enabling mathematical modeling of biological processes, which are so far only qualitatively understood [1, 18, 19, 23, 24, 27, 28]. One important group of models are ordinary differential equations (ODE). They describe the time-dependent kinetic behavior and causality. In order to find a suitable or even *true* model, an interdisciplinary approach of theoretical modeling and experimental measurements is applied.

PottersWheel has been developed in order to support the interactive procedure of model creation, fitting to experimental data, and designing new experiments based on a given hypothesis. It is designed as a Matlab toolbox and tries to reduce technical or tedious work of the modeler like a potter's wheel enhancing the modeling of pottery. Models can be imported from SBML or entered as reaction schemes or as an ODE system. I started the development in April 2005. The current version has about 80,000 lines of Matlab and C++ code. PottersWheel has a strong focus on modeling and fitting of real experimental data, is numerically fast and enables the user to investigate the dynamic properties of a model by changing model parameters and the shape of driving input functions in real time with a set of sliders. Fitting of experimental data includes multi-experiment fitting, where different experiments, e.g. a pulsed and a continuous stimulation, are used simultaneously to estimate the kinetic parameters. This highly increases the power to distinguish competing model hypotheses. FORTRAN integrators and dynamically generated and compiled C MEX files for the differential equations strongly improve integration speed, which is required for data fitting and an interactive, real time modeling experience. Several optimization algorithms are supported so far: Line search, trust-region, genetic algorithm and simulated annealing. Rule based modeling simplifies to create and maintain models with combinatorial complexity. A rich application programming interface (API) allows to integrate PottersWheel functions into own Matlab programs. The use of Macros helps to automate and to document the modeling work. User contributions are welcome as plug-ins to extend PottersWheel. This documentation together with the software, example models and a user forum is available at www.potterswheel.de. Please use the forum for suggestions, questions and to report problems or contact me directly at maiwald@fdm.uni-freiburg.de.

Thomas Maiwald, Freiburg, March 14, 2007

Chapter 1

Quickstart

1.1 General approach and recommendations

New models are created as simple structured text files – the model definition files. One or more models can be loaded into the PottersWheel upper list box, the *repository*. For each model, several data sets can be simulated or added from external text or Excel files. A model with one data file is a *couple*. In order to work with a couple, e.g. to fit the model to the coupled data or to investigate the model properties with the equalizer, one or more couples have to be *combined* into the lower listbox. This allows also for multi-model and multi-experiment fitting. Some functions, like the simulation of data work on the currently *selected* models, i.e. all selected models in the upper listbox of PottersWheel. This is distinguished from combined models.

Simulated and fitted time series are optionally plotted for driving input, observables, dynamic and derived variables. The figures can be arranged automatically. Settings like the used optimization algorithm or visualization can be changed in the configuration dialog. The current working state can be saved to hard disk and reloaded afterwards. This allows also to exchange modeling work with colleagues. It is recommended to create the following folders: Models, WorkingFolder, Data, IDSO, Repositories, Configurations and Reports. The current Matlab work folder should be changed to the WorkingFolder, since PottersWheel creates temporary files. The current folder can be checked with the Matlab

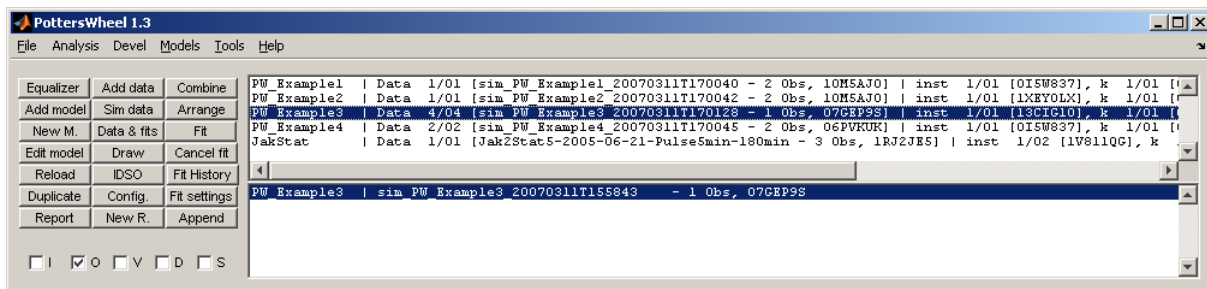


Figure 1.1: The PottersWheel main graphical user interface

`pwd` function. Shortcuts are available for most important functions, e.g.

- M Add an existing model
- s Simulate data for selected models
- a Arrange all figures
- B Combine the selected models
- f Fit data for combined couples
- g Show model graph of selected models
- G Show model graphs as png
- e Open the equalizer
- p Open the PottersWheel main window
- c Open the configuration dialog
- d Draw the model trajectories
- E Edit the selected models
- D Add an external data set
- h Show all shortcuts

Data is simulated for the current model observation functions, usually as defined in the model definition file. When external data is added, the user has to map the columns of the data file to the observation functions of the model. The mapping can be saved and reused, if the same data set is used again. External data files have a simple structure, please compare section 6.

PottersWheel supports the creation of reports with latex pdf files. The *report designer* displays all sections which are already part of the report. A section can be added after an analysis is applied. Special sections exist like a graphical model visualization or a list of model reactions. Section titles and ordering can be changed.

The next section introduces several basic models. Afterwards, an example session is described step by step. In order to illustrate how all commands can be used within custom Matlab scripts or directly from command line, the quickstart finishes with an example script comprising the steps of the example session.

1.2 Example models

The example models are available in the documentation folder and at www.potterswheel.de. Model 1 is a small, closed four-player system. Model 2 enlarges the system by an external driving function. Model 3 introduces negative and positive feedbacks and model 4 exemplifies the use of rule based modeling for systems with combinatorial complexity.

For each reaction the reactants, products, modifiers as e.g. enzymes, parameters, and the rate signature are specified in the model definition file. The rate signature is an arbitrary mathematical expression depending on $r_1, r_2, \dots, p_1, p_2, \dots, m_1, m_2, \dots, k_1, k_2, \dots$, the i -th given reactant, product, modifier or parameter of the current reaction, respectively. For further details about the structure of model definition files please compare with section 2.

1.2.1 Model 1: Four variables, closed system

Model 1 describes the mass action reaction of A to B with a kinetics $\dot{A} = -AtoB \cdot A$ represented by the rate signature $k1 * r1$ and the first parameter $AtoB$ and the first reactant A . Reactions $B \rightarrow C$, $C \rightarrow D$, and $C \rightarrow A$ follow the same mass action kinetics and hence have the same mathematical rate signature $k1 * r1$. The parameter $k1$ refers to parameter of each reaction, i.e. $BtoC$, $CtoA$, and $CtoD$.

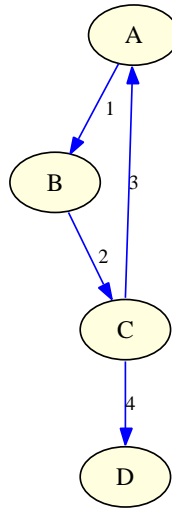


Figure 1.2: Model 1: Four variables, closed system.

```
% PottersWheel model definition file
function m = getModel()

m = pwGetEmptyModel();

%% Meta information
m.ID          = 'PW_Example1';
m.name        = 'Small model with 4 players';
m.description = '';
m.authors     = {'Thomas Maiwald'};
m.dates       = {'2006-09-28', '2007-03-08'};
m.type        = 'PW-1-3';

%% Dynamic variables
% m = pwAddX(m, ID, startValue, type, minValue, maxValue)
m = pwAddX(m, 'A', 10);

%% Reactions
% m = pwAddR(m, reactants, products, modifiers, type, ..
%           options, rateSignature, parameters)
% First reaction:
% Reactants:    A
% Products:     B
% Modifiers:    None ({})
```

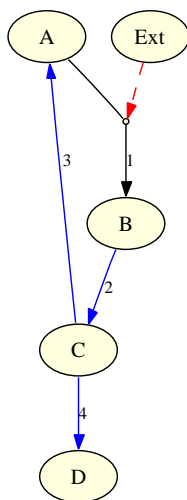


Figure 1.3: Model 2: Four variables, open system

```
% Type:          Custom kinetics ('C')
% Options:       None ([])
% Rate signature: k1*r1
% Parameters:    AtoB
m = pwAddR(m, {'A'}, {'B'}, {}, 'C', [], 'k1*r1', {'AtoB'});
m = pwAddR(m, {'B'}, {'C'}, {}, 'C', [], 'k1*r1', {'BtoC'});
m = pwAddR(m, {'C'}, {'A'}, {}, 'C', [], 'k1*r1', {'CtoA'});
m = pwAddR(m, {'C'}, {'D'}, {}, 'C', [], 'k1*r1', {'CtoD'});

%% Observables
% m = pwAddY(m, rhs, ID, scalingParameter, errorModel)
m = pwAddY(m, 'A');
m = pwAddY(m, 'B');

%% Derived variables
% m = pwAddZ(m, rhs, ID)
m = pwAddZ(m, 'A + B + C', 'SumABC');
m = pwAddZ(m, 'D');
```

1.2.2 Model 2: Four variables, open system

The second example model differs only in a few points from the first example, which are marked. Essentially, reaction $A \rightarrow B$ is now controlled via an external driving function. The shape of the driving input function is specified via

```
m = pwAddU(m, 'Ext', 'steps', [-100 0 10], [0 1 0]);
```

representing a step input starting at $t = -100$ with a value of 0, jumping at $t = 0$ to 1 and returning at $t = 10$ to 0. The first triple of numbers corresponds to the timing and the second triple to the values of all jumps.

```

% PottersWheel model definition file
function m = getModel()

m = pwGetEmptyModel();

%% Meta information
m.ID          = 'PW_Example2'; % changed from example 1
m.name        = 'Small model with 4 players and one external driving function'; % changed
m.description = '';
m.authors     = {'Thomas Maiwald'};
m.dates       = {'2006-09-28','2007-03-08'};
m.type        = 'PW-1-3';

%% Dynamic variables
% m = pwAddX(m, ID, startValue, type, minValue, maxValue)
m = pwAddX(m, 'A', 10);

%% Reactions
% m = pwAddR(m, reactants, products, modifiers, type, ...
%           options, rateSignature, parameters)
m = pwAddR(m, {'A'}, {'B'}, {'Ext'}, 'C', [], 'k1*r1*m1', {'AtoB'}); % changed
m = pwAddR(m, {'B'}, {'C'}, {},      'C', [], 'k1*r1',      {'BtoC'});
m = pwAddR(m, {'C'}, {'A'}, {},      'C', [], 'k1*r1',      {'CtoA'});
m = pwAddR(m, {'C'}, {'D'}, {},      'C', [], 'k1*r1',      {'CtoD'});

%% Observables
% m = pwAddY(m, rhs, ID, scalingParameter, errorModel)
m = pwAddY(m, 'A');
m = pwAddY(m, 'B');

%% Default driving input
% m = pwAddU(m, ID, uType, uTimes, uValues)
m = pwAddU(m, 'Ext', 'steps', [-100 0 10], [0 1 0]); % new in example 2

%% Derived variables
% m = pwAddZ(m, rhs, ID)
m = pwAddZ(m, 'A + B + C', 'SumABC');
m = pwAddZ(m, 'D');

```

1.2.3 Model 3: Positive and negative feedbacks

Model 3 describes the activation of B to pB triggered by pA which itself is enzymatically created via an external driving input Ext . A positive feedback enhances the activation: pB enables the reaction of C to pC , which also activates B . On the other hand, pB triggers D to pD which builds a complex with pA , lowering the concentration of free pA – a negative feedback. Depending on the parameter values, the positive or negative feedback has a stronger impact on the system dynamics.

```

% PottersWheel model definition file
function m = getModel()

m = pwGetEmptyModel();

```

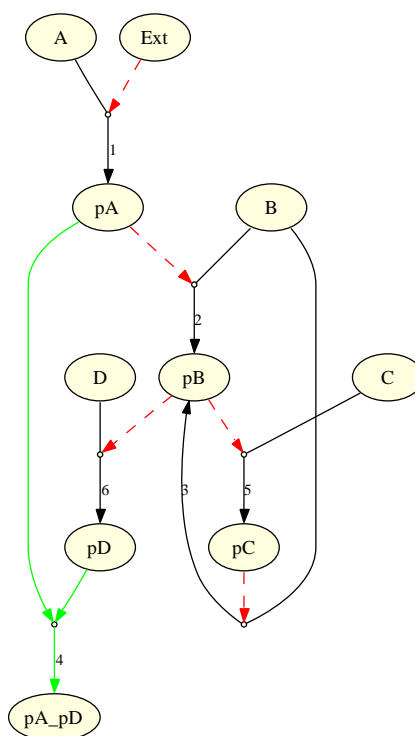


Figure 1.4: Model 3: Positive and negative feedback

```

%% Meta information
m.ID          = 'PW_Example3';
m.name        = 'Model with positive and negative feedback';
m.description = '';
m.authors     = {'Thomas Maiwald'};
m.dates       = {'2007-03-08'};
m.type        = 'PW-1-3';

%% Dynamic variables
% m = pwAddX(m, ID, startValue, type, minValue, maxValue)
m = pwAddX(m, 'A', 1);
m = pwAddX(m, 'B', 1);
m = pwAddX(m, 'C', 1);
m = pwAddX(m, 'D', 10);

%% Reactions
% m = pwAddR(m, reactants, products, modifiers, ...
%             type, options, rateSignature, parameters)
m = pwAddR(m, {'A'}, {'pA'}, {'Ext'}, 'C', [], 'k1*r1*m1-k2*p1', {});
m = pwAddR(m, {'B'}, {'pB'}, {'pA'}, 'C', [], 'k1*r1*m1-k2*p1', {});
m = pwAddR(m, {'B'}, {'pB'}, {'pC'}, 'C', [], 'k1*r1*m1-k2*p1', {});
m = pwAddR(m, {'pA', 'pD'}, {'pA_pD'}, {}, 'C', [], 'k1*r1*r2-k2*p1', {});
m = pwAddR(m, {'C'}, {'pC'}, {'pB'}, 'C', [], 'k1*r1*m1-k2*p1', {});
m = pwAddR(m, {'D'}, {'pD'}, {'pB'}, 'C', [], 'k1*r1*m1-k2*p1', {});

```

```

%% Observables
% m = pwAddY(m, rhs, ID, scalingParameter, errorModel)
m = pwAddY(m, 'pB');

%% Default driving input
% m = pwAddU(m, ID, uType, uTimes, uValues)
%m = pwAddU(m, 'Ext', 'steps', [-100 0 10], [0 1 0]); % Pulsed stimulation
m = pwAddU(m, 'Ext', 'steps', [-100 0], [0 1]); % Continuous stimulation

%% Derived variables
% m = pwAddZ(m, rhs, ID)
m = pwAddZ(m, 'pB');

```

1.2.4 Model 4: Rule based modeling

Model 4 is an example for rule based modeling, where some reactions are not given explicitly, but rather as a general expression with placeholders. Depending on all available dynamical variables of the systems, the placeholders are substituted leading to several reactions. PottersWheel supports a simplified regular expression like syntax. For more details please compare section 3.

The model is based on the phosphorylation of Mek via active Raf and the subsequent activation of Erk to ppErk, which then acts as a negative feedback on Raf. Artificially, complexes of Mek and Erk with species A, B, and C are introduced, in order to illustrate the power of rule based modeling. A more realistic example, e.g. a receptor with several independent binding sites, would increase the size of the overall model unnecessarily.

```

% PottersWheel model definition file
function m = getModel()

m = pwGetEmptyModel();

%% Meta information
m.ID          = 'PW_Example4';
m.name        = '';
m.description = 'Artificial model to exemplify the use of rule based modeling.';
m.authors     = {'Thomas Maiwald'};
m.dates       = {'2007-03-14'};
m.type        = 'PW-1-3';

%% Dynamic variables
% m = pwAddX(m, ID, startValue, type, minValue, maxValue)
m = pwAddX(m, 'Mek',      1);
m = pwAddX(m, 'Mek_A',   1);
m = pwAddX(m, 'ppMek',   0);
m = pwAddX(m, 'ppMek_A', 0);
m = pwAddX(m, 'Erk',     1);
m = pwAddX(m, 'Erk_A',   1);
m = pwAddX(m, 'Erk_B',   1);
m = pwAddX(m, 'Erk_C',   1);

```

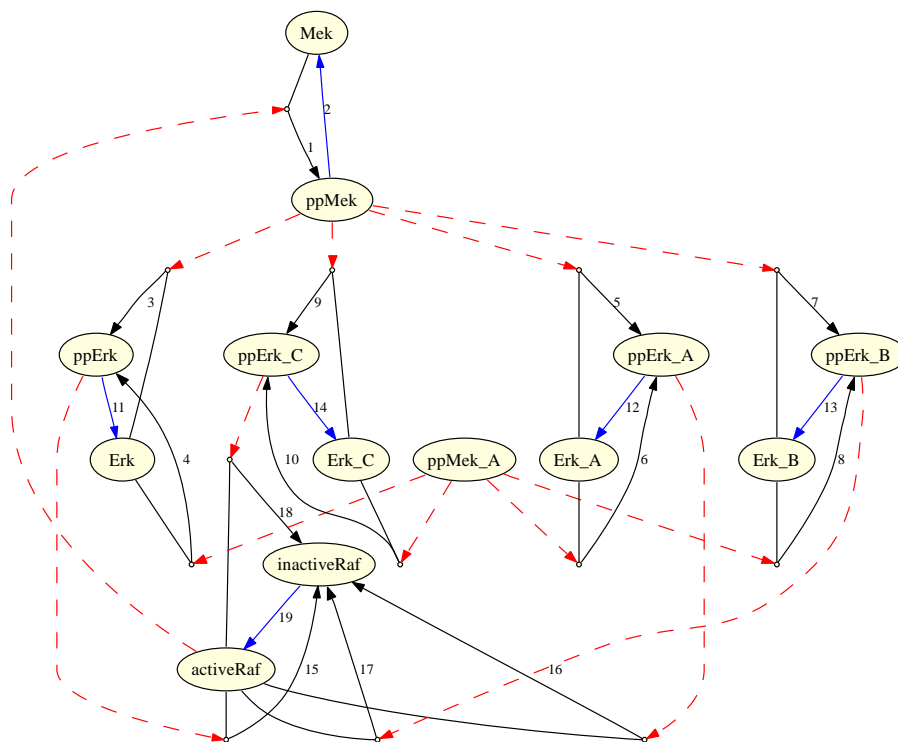


Figure 1.5: Model 4: Rule based modeling

```

m = pwAddX(m, 'pErk', 0);
m = pwAddX(m, 'pErk_A', 0);
m = pwAddX(m, 'pErk_B', 0);
m = pwAddX(m, 'pErk_C', 0);
m = pwAddX(m, 'ppErk', 0);
m = pwAddX(m, 'ppErk_A', 0);
m = pwAddX(m, 'ppErk_B', 0);
m = pwAddX(m, 'ppErk_C', 0);

%% Reactions
% m = pwAddR(m, reactants, products, modifiers, ...
%           type, options, rateSignature, parameters)
m=pwAddR(m,{'Mek'},          {'ppMek'},          {'activeRaf'},'C', [], 'k1*r1*m1');
m=pwAddR(m,{'ppMek'},       {'Mek'},          {},          'C', [], 'k1*r1');
m=pwAddR(m,{'<1:*>_Erk_<2:*>'},{'<1>_ppErk_<2>'},{'*_ppMek_*'},'C', [], 'k1*r1*m1');
m=pwAddR(m,{'<1:*>_ppErk_<2:*>'},{'<1>_Erk_<2>'},{},          'C', [], 'k1*r1');
m=pwAddR(m,{'activeRaf'},{'inactiveRaf'},          {'*_ppErk_*'},'C', [], 'k1*r1*m1');
m=pwAddR(m,{'inactiveRaf'},{'activeRaf'},          {},          'C', [], 'k1*r1');

%% Observables
m = pwAddY(m, 'Mek');
m = pwAddY(m, 'Erk');

```

1.3 Example session

Keyboard shortcuts are given in round brackets.

1. After installation of PottersWheel, start Matlab and change the working directory to some suitable location.
2. Type `PottersWheel` or just `pw`. The main PottersWheel graphical user interface appears with two empty listboxes: The upper one – the *repository* – shows all available model-data-couples, the lower listbox – the *combination list* – contains the currently combined couples for fitting.
3. Add model 'PW_Example1.m' from the examples folder (Button 'Add Model' or shortcut 'M'). The model will be parsed, translated into a set of ordinary differential equations and compiled into a C MEX file. It appears in the repository. You can add many models to the repository or duplicate existing models.
4. View the model graph (g).
5. Select the model and simulate data (s). Now, the first model is coupled to one data set. The red line belongs to the integrated model trajectory. The blue circles are simulated data points, i.e. the red line plus Gaussian noise with a certain standard deviation.
6. The lower list represents the current models which are combined for fitting to real or simulated data. We have only one model - nonetheless we 'combine' it (B) and it appears in the lower list.
7. In order to experience the dynamic model properties, open the PW equalizer (e). Here, every parameter can be changed with sliders by using the mouse or the arrow keys. Arrange the figures (a). Pressing 1, 2, 3, ... changes the currently used slider, arrow left, arrow right distinguishes between small and large change mode, and arrows up and down change the slider value. You can also specify a parameter value directly into the edit boxes.
8. Open the configuration dialog (c) and specify which optimization algorithm should be used. Recommended are the simulated annealing or the trust region approaches. The latter requires the optimization toolbox.
9. Go back to the equalizer and disturb the parameter values (;), see the effect, and fit the model to the data (f). The maximum number of iterations and the wished accuracy can be specified in the configuration dialog: 'Iterations', 'Tolerance χ^2 ', 'Tolerance fit parameters'.
10. Change to the main window (p).
11. Check out that with 'w' the command window appears and 'pw' brings you back to PottersWheel. 'pe' would start the equalizer.

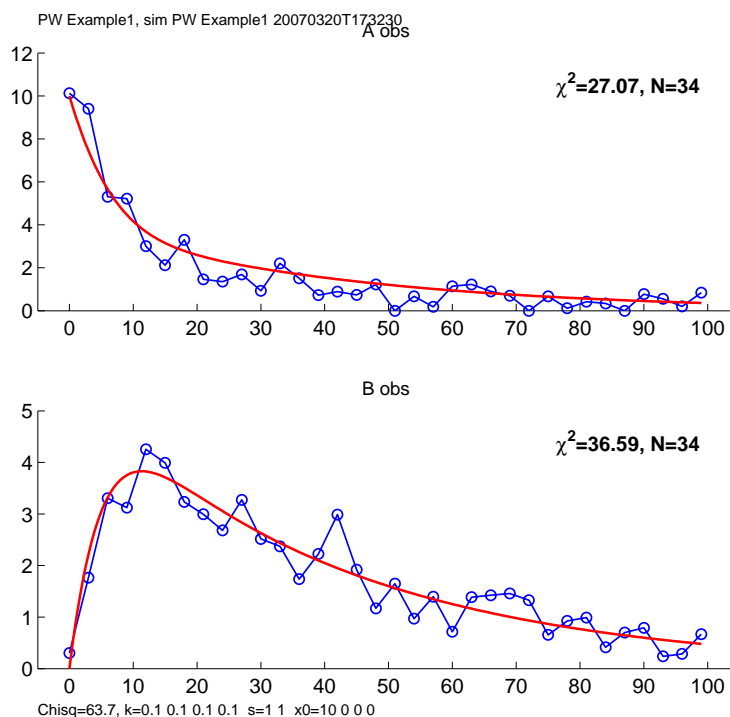
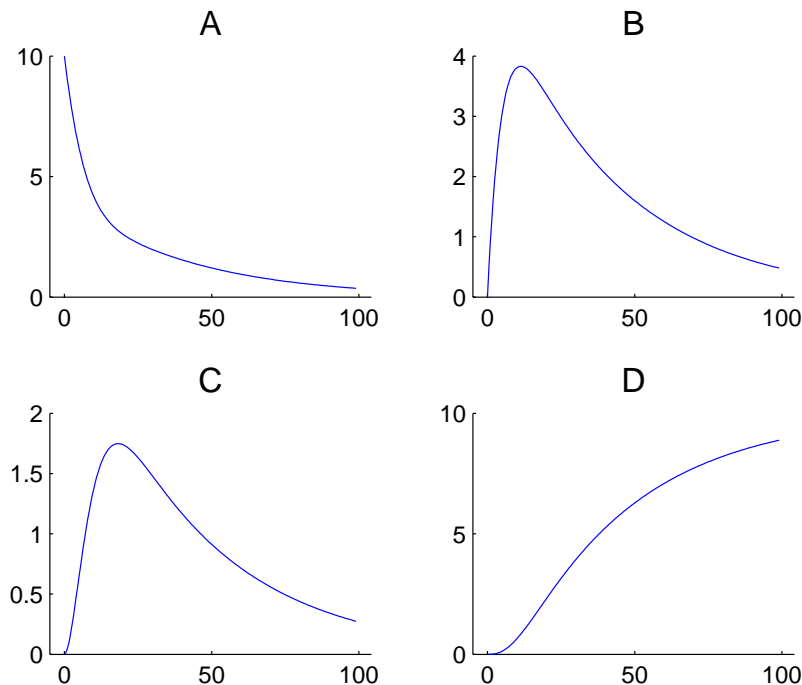


Figure 1.6: Simulated data for model 1

12. Activate the 'x' checkbox in order to see the time dependency of the internal dynamic variables (x). Draw (d) and arrange the figures again (a). The dynamic variables comprise all players that are available in your reactions, in this case A, B, C and D.
13. Go to the main window (p) and add a data set (D), e.g., PW_Example1_data.txt. If the column titles of the data file can not automatically be mapped to the model's observations, you are asked to map it in a dialog.
14. Afterwards, combine the model (B), go to the equalizer (e), arrange the figures (a), draw model and data (d) and fit (f).
15. Open the configuration (c) and deselect 'Show plots and infos during fitting'. Set 'N fit sequence' to 20. Close the dialog and start a fit sequence in the equalizer by pressing F2 or with the shortcut '(.'.
16. After the 20 fits, it would be interesting to analyse the fitted parameter values. Since some fits could be trapped in a local minimum, we want to analyse only the best 50% of the fits. Enter 50 into the edit box near the analysis combo box in the equalizer. Select 'Fit Sequence Analysis' and press 'Go'. Check the χ^2 value of the best 50% of the fits (red circles).
17. The boxplots gives you an impression of the fitted parameter values for the best fits. This requires the statistics toolbox. Otherwise, you can regard the histograms of

Figure 1.7: Dynamical variables x

the parameter value distribution. Mean and standard deviation are also displayed at the command window.

18. Correlated parameters correspond to an unidentifiability.
19. Save the model with the currently fitted parameter values (/) or by the menu entry 'Models – Save fitted model'.
20. Save the whole session into a PottersWheel repository file (S). You can reload it any time (R).
21. With 'h' you get a list of available shortcuts.
22. The second model, PW_Example2, has an external driving function. Load it and view the graph. This time use the G shortcut to view the graph as a png file with your favorite image viewer.
23. Edit the model (E) in order to view the chemical reactions. The first reaction from 'A' to 'B' is now influenced by modifier 'Ext' with an enzymatic kinetics 'E'. All other reactions have mass action kinetics 'MA'. The kinetic parameters are specified in the last field, e.g. 'AtoB'.
24. Investigate the structure and comments of the opened model definition file PW_Example2.m. The default driving input for 'Ext' is a pulse of ten minutes duration. Simulate some

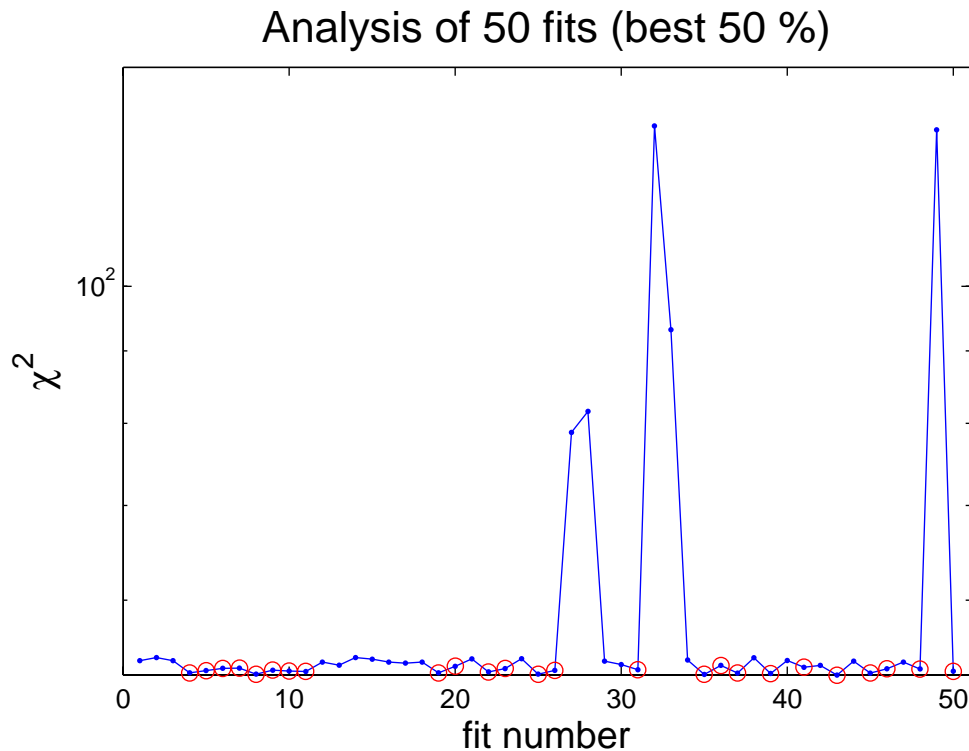


Figure 1.8: χ^2 values of fit sequence. Best 50% are selected for analysis.

data to see the effect: The Ext concentration drops at $t=10$ to zero. In consequence, the concentration of B decreases very fast.

25. In order to experience the effect of different stimuli in real time, start the input designer (i).
26. Change the input for example to a slowly increasing exponential ramp. The model trajectory will change, too.
27. In the PottersWheel main GUI, via 'Analysis – 3D Sensitivity Analysis', you can determine the influence of parameter changes to the mean concentration of the variables, observables or derived variables. For PW_Example2, parameter CtoA has only little influence on B.
28. Close all figures with shortcut Q.
29. In order to investigate more closely the little effect of CtoA on B, start the 1D sensitivity analysis. Select CtoA and press 'Go'. In this case 13 model trajectories are drawn for slightly changed values of CtoA. The upper window shows the mean concentration. The lower figure displays the actual trajectories. Apparently the B concentration is not affected by changes to CtoA. The lowest value for the changed parameter results in a thicker trajectory.

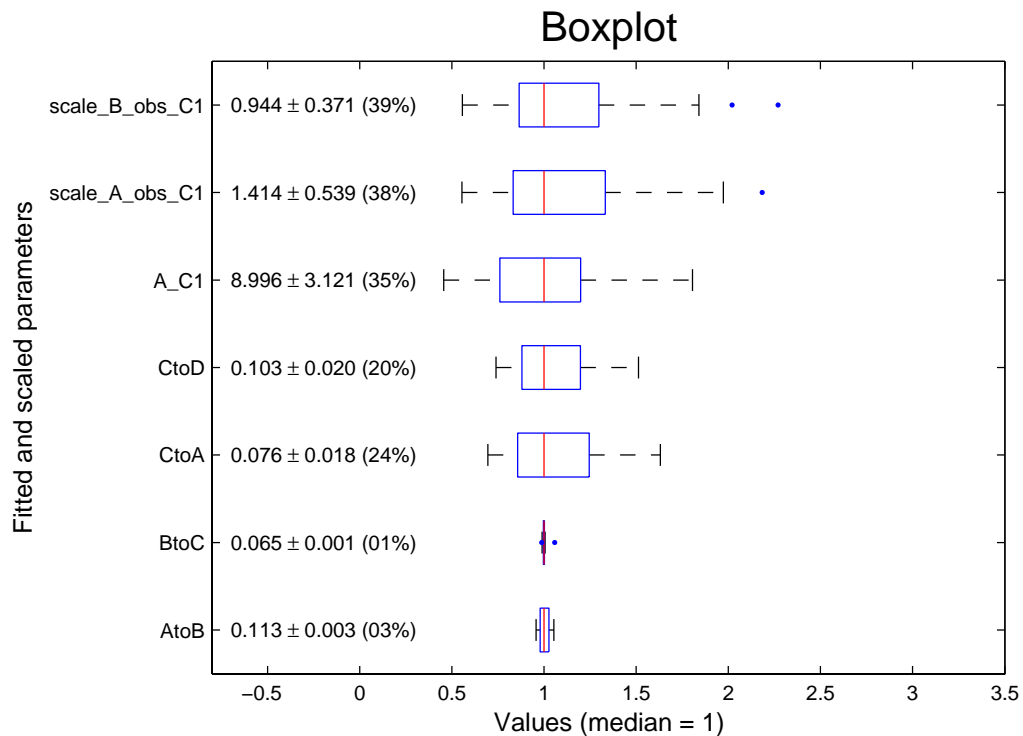


Figure 1.9: Box plots of the fitted parameters.

30. Close the figures with Q.
31. Compare the impact of changing CtoA and CtoD on player A. This can be done with the 2D sensitivity analysis. In the example, we multiply the two parameters with factors 0.1, 0.2, ... up to 0.9.
32. Increasing CtoA increases also the concentration of A. On the other hand, an increase of CtoD decreases the A concentration. Again, thick lines belong to the lowest parameter values, i.e. in this case multiplied with factor 0.1. The black line corresponds to no change – the model with the current parameter values as adjusted or fitted in the equalizer.
33. PottersWheel strength is multi-experiment fitting. It is possible to combine several experiments described by one or more models and to fit some parameters *globally* and some *locally*, i.e., some parameters get the same value for all experiments and some may have different values depending on the experiment. Often, start values x_0 or scaling parameters s are fitted locally and dynamic parameters k are fitted globally. Now it's time for you to create a new model (N). Let the wizard include comments into the created model frame in order to learn the structure of PottersWheel model definition files.

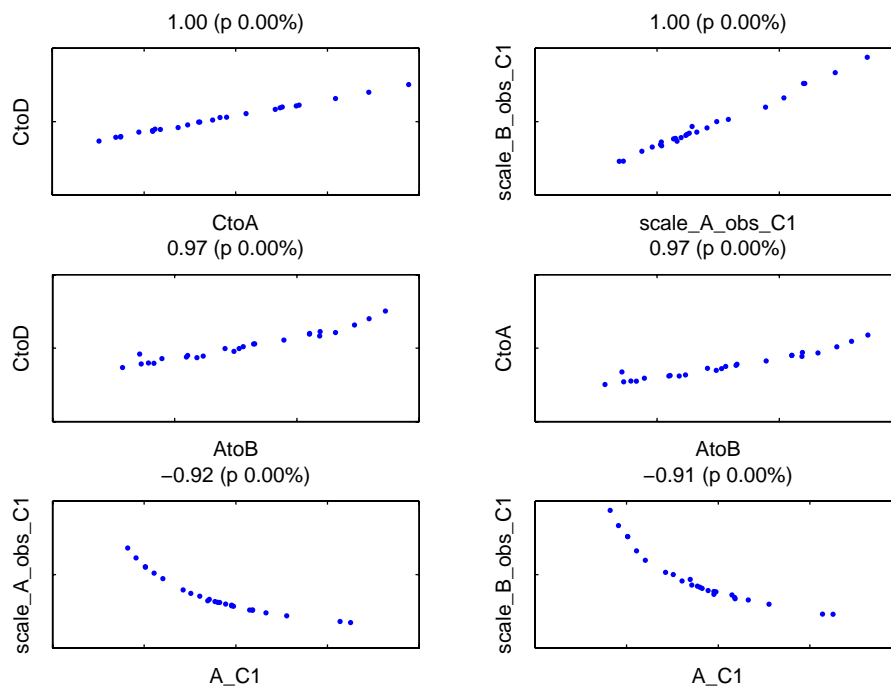


Figure 1.10: Significantly correlated parameters.

1.4 Example script

All PottersWheel commands are also available from command line or within a Matlab script. This provides a high level of customization. The following example script assumes the directory structure and files as given in the example folder. A list of available functions can be found in section 13.2. Please use `pwHelp` to get information about a specific PottersWheel function within the Matlab command line, e.g. `pwHelp pwFit`.

```
% PottersWheel Macro for the example session
%
% Thomas Maiwald, 2007-03-20

% Clear PottersWheel
pwClear

% Add models
pwAddModel({'../Models/PW_Example1.m'});
pwAddModel({'../Models/PW_Example2.m'});
pwAddModel({'../Models/PW_Example3.m'});
pwAddModel({'../Models/PW_Example4.m'});

% Select all models and show graphs
pwSelect([1 2 3 4]);
pwShowGraph;
```

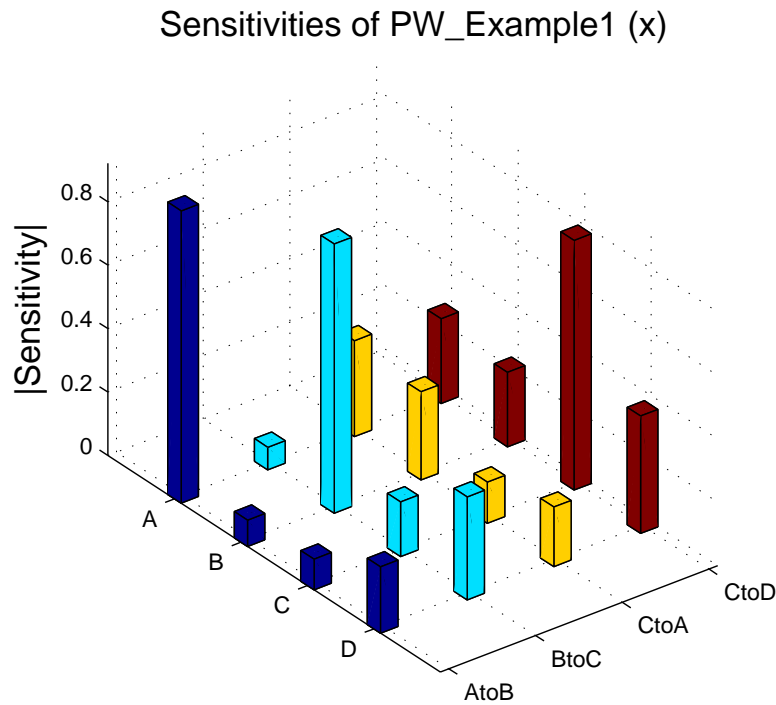


Figure 1.11: 3D sensitivity analysis.

```

% Simulate data for the first model
pwSelect(1);
pwSim;

% 'Combine' the first couple to the combination list,
% arrange the figures and draw
pwCombine(1);
pwArrange;
pwDraw;

% Show separate figures for the observables y,
% variables x and derived variables z,
% but not for the driving input u
pwSetPlottingState('x',true);
pwSetPlottingState('y',true);
pwSetPlottingState('z',true);
pwSetPlottingState('u',false);

% Open the equalizer and arrange all figures
pe
pwArrange;

% Select trust region as optimizer. All optimizers:
% 'LineSearch', 'TrustRegion', 'Annealing', 'Genetic'
% Attention: TrustRegion requires the optimization toolbox.

```

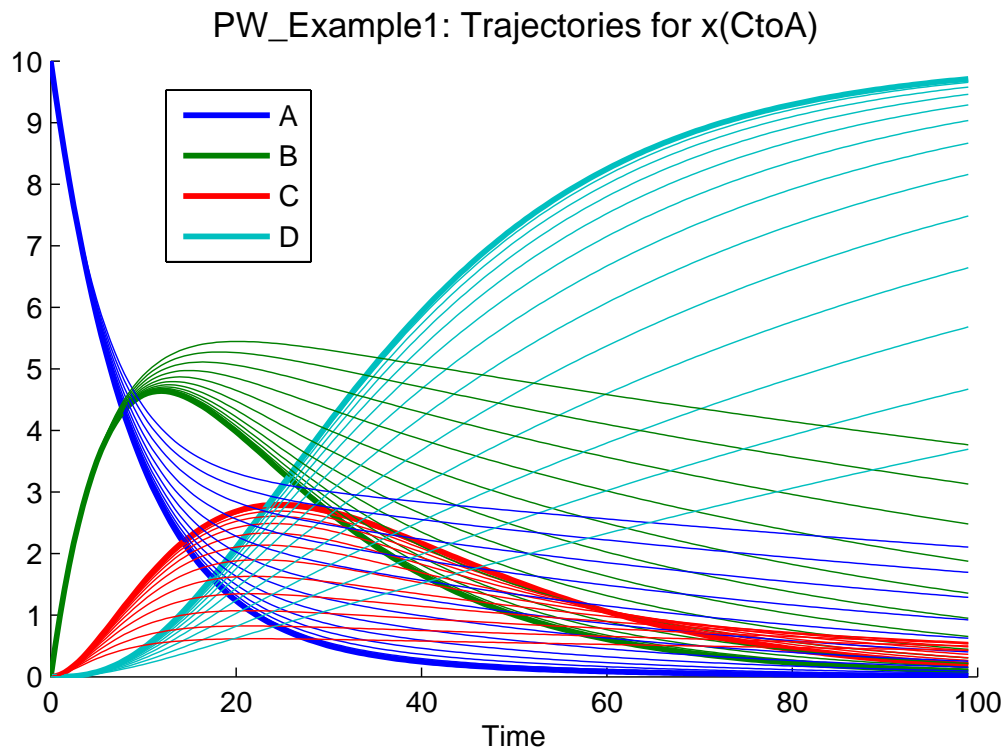


Figure 1.12: Detailed sensitivities.

```

pwSetOptimizer('TrustRegion');

% Disturb the parameters and fit
pwDisturb;
pwFit;

% Make a fit sequence with 20 fits and without plotting
pwShowFitting(false);
pwF2(20);
pwShowFitting(true);

% Analyse the best 50% of the fits
pwFitSequenceAnalysis(50);

% Go back to the main window and add
% an external data set to the first model
pw
pwSelect(1)
pwAddData('../Data/PW_Example1_data.txt');

% Combine, show only x and y, arrange and fit
pwCombine(1);
pwSetPlottingState('z', false);
pwArrange
pwFit;

```

```
% Apply a 3D sensitivity analysis to the first model
pwSelect(1);
pwSensitivityAnalysis3D;

% Add data to the second model and fit
pwSelect(2);
pwAddData(' ../Data/PW_Example2_data.txt');
pwCombine(2);
pwArrange;
pwDraw;
pwFit;
```


Chapter 2

Model creation

Models are defined in model definition files, which are simple structured text files. They contain either a set of chemical reactions or a system of differential equations. Alternatively, Systems Biology Markup Language (SBML) models can be imported. This chapter describes the structure of required and optional paragraphs of model files. The used help functions to add reactions, players, etc. to the model have arguments of the following types:

Type	Explanation	Example
ID	String of letters, numbers and '_'	'A_B'
string	String of ASCII characters	'r1 * k1'
integer	A whole number	17
float	A real number	23.12
string array	Matlab cell array of strings	{'A', 'B', 'C'}
ID array	Matlab cell array of IDs	{'A', 'B', 'C'}
float array	Matlab numeric vector	[0, 12, 14]

The following keywords are reserved and should not be used within IDs: `only_`, `any_`, `withAny_`, `withOnly_`. IDs must be used in a unique fashion throughout the whole model, i.e. the same ID for a parameter and a variable or observable is not allowed.

2.1 Naming conventions

PottersWheel supports arbitrary IDs for reactants, products, and modifiers (enzymes), as long as only letters, numbers and the underscore are used and the first character is not a number. However, in order to make use of several features like subreaction-networks and combinatorial complexity, the following conventions are recommended:

- Basic species start with a capital letter, e.g. Erk and Mek
- Modifications are lower case prefixes, e.g. pErk and ppErk
- Species of a complex are separated by an underscore, e.g. pR_pR and Gab1_Grb2

2.2 Reaction based models

2.2.1 General information

Every model definition file is a Matlab m-file starting with general information about the model, comprising a model ID, a model name and a description of the model. A list of authors, working dates and the used PottersWheel version complete the meta information, as shown in the example below:

```
function m = getModel()

m = pwGetEmptyModel();

m.ID          = 'PW_Example1'; % no blanks, no leading number
m.name        = 'Model with 4 players';
m.description = 'This is a small example model.';
m.authors     = {'Thomas Maiwald', 'Peter Smith'};
m.dates       = {'2006-09-28', '2006-12-24'};
m.type        = 'PW-1-3';
```

2.2.2 Dynamic variables and start values

The dynamic variables comprise all species that are part of the left hand side of the differential equations. Essentially their initial value, fitting type and limits have to be specified:

```
m = pwAddX(m, ID, startValue, type, minValue, maxValue, ...
           unit, compartment, name, description, typeOfStartValue);
```

Argument	Type	Description
ID	ID	
startValue	float	initial value for integration (default 0)
type	string	fit type: 'global', 'local', 'fix'
minValue	float	minimal value during fitting
maxValue	float	maximal value during fitting
unit	string	
compartment	ID	
name	string	descriptive name
description	string	
typeOfStartValue	string	'amount' or 'concentration' (default)

Not listed dynamic variables which appear in the list of reactions have a default start value of 0 and are fixed during fitting. The default compartment is the first specified compartment in the compartment section. The default fit setting for non-vanishing players is 'local'. Please compare section 7.10 for an explanation of local and global fit settings. Example:

```
m = pwAddX(m, 'Stat', 1.7, 'local', 0, 100);
m = pwAddX(m, 'pStat');
```

2.2.3 Reactions

For each reaction the reactants, products, modifiers such as enzymes, kinetic information and parameters are specified. Additionally, an optionally reaction description, ID, name and compartment information may be entered. The help function `pwAddR` adds a reaction to the model `m` and has the following arguments:

```
m = pwAddR(m, reactants, products, modifiers, type, ...
           options, rateSignature, parameters, ...
           description, ID, name, fast, compartments);
```

Argument	Type	Description
reactants	ID array	IDs of the used reactants
products	ID array	IDs of the products
modifiers	ID array	IDs of the modifiers
type	string	'MA', 'MM', 'E', 'C', 'D'
options	integer	
rateSignature	string	
parameters	ID array	
description	string	
ID	ID	
name	string	
fast	integer	1 yes, 0 no
compartments	ID array	

String arguments which may have more than one entry, e.g. the reactants, are entered as string cell arrays, e.g. `{'Reactant1', 'Reactant2', 'Reactant3'}`. This applies also to the products, modifiers, parameters and compartments. Reaction types are either predefined as for mass action (MA), Michaelis-Menten (MM), enzymatic (E), delay (D) or custom (C). A simple irreversible reaction $A \rightarrow B$ with mass action kinetics and rate A_to_B would be specified via

```
m = pwAddR(m, {'A'}, {'B'}, {}, [], [], 'k1 * r1', {'A_to_B'});
```

Here, `{}` and `[]` are used for empty arguments. The rate signature `'k1 * r1'` reflects that the concentration of A is decreased per time unit by the first parameter k_1 , which is A_to_B , times the first reactant r_1 , which is A . A reversible reaction $A \leftrightarrow B$ could be entered as two irreversible reactions $A \rightarrow B$ and $B \rightarrow A$ or via

```
m = pwAddR(m, {'A'}, {'B'}, {}, [], [], 'k1*r1 - k2*p1', {'AtoB', 'BtoA'});
```

introducing `p1` as the placeholder for the first product, which is B . The second parameter `B_to_A` is used in the rate signature via `k2`. In general, r_i , p_i , m_i , k_i , and c_i are placeholders for the i -th reactant, product, modifier, parameter, and compartment of the current reaction. This way, the reaction kinetic is always specified in a simple text book like format and does not change from reaction to reaction as long as the underlying kinetics are the same. For example a Michaelis-Menten reaction will have always the rate signature `'k1 * r1 * m1 / (k2 * r1)'`, independent of the current reaction number. Only the used reactants, products, modifiers, and parameters will be different for two different Michaelis-Menten reactions. The following set of reactions corresponds to a simplified model of the JakStat pathway, with a Michaelis-Menten like phosphorylation of the receptor R and subsequent deactivation, a dimerization of $Stat$ triggered by the active receptor pR , entering of the dimer into the nucleus with an irreversible mass action kinetic, breaking of the dimer and leaving the nucleus in one step:

```

m = pwAddR(m, {'R'}, {'pR'}, {}, [], [], ...
    'k1 * r1 / (k2 + r1)', {'R_act_Vmax', 'R_act_Km'});
m = pwAddR(m, {'pR'}, {'R'}, {}, {}, [], [], ...
    'k1 * r1', {'R_deact'});
m = pwAddR(m, {'Stat', 'Stat'}, {'Dimer'}, {'pR'}, [], [], ...
    'k1 * r1 * r2 * m1', {'dimerization'});
m = pwAddR(m, {'Dimer'}, {'Dimer_nucleus'}, {}, [], [], ...
    'k1 * r1', {'import'});
m = pwAddR(m, {'Dimer_nucleus'}, {'Stat', 'Stat'}, {}, [], [], ...
    'k1 * r1', {'export_and_break'});

```

2.2.4 Reaction kinetics

PottersWheel kinetics describe the change of the *concentration* of the first reactant or the first product, if no first reactant is available. SBML uses kinetic laws describing the change of amount per time. This is very useful if rates are not entered manually and if the system of interest contains many compartments. PottersWheel uses rate laws describing the change of concentration per time, reducing the modelers workload, since it is not necessary to keep track of compartment information - PottersWheel will do that during the automatic translation into the set of differential equations. Rate laws of PottersWheel v_{PW} correspond to kinetic laws of SBML v_{SBML} via

$$v_{PW} = \frac{v_{SBML}}{\text{Compartment size of first reactant}} \quad (2.1)$$

If no first reactant is available:

$$v_{PW} = \frac{v_{SBML}}{\text{Compartment size of first product}} \quad (2.2)$$

Section 12.2 provides a list of often used kinetics. It is useful to define important reaction kinetics as a Matlab string variable above the reactions paragraph, e.g.

```

Hill = 'k1 * r1^k2 / (k3^k2 + r1^k2)';
m = pwAddR(m, {'A'}, {'B'}, {}, 'C', [], Hill, {'v', 'h', 'Shalf'});

```

2.2.5 Delay reactions

Signal transduction pathway models often contain delays, e.g. if a set of reactions is not modeled on a detailed basis but rather on an effective level as for example binding of a transcription factor to the DNA. Usually, this would require integrators for delay differential equations (DDE) and specification of the first τ seconds of the system variables with τ being the largest occurring delay. This approach leads to exact delays. In $A \xrightarrow{\tau} B$, a step input for A would lead to a delayed step input for B . PottersWheel prefers the linear chain trick since many biochemical processes are not expected to always last exactly the same period of time but follow rather a distribution of times: The delay of a scaled

input signal $g(t)$ is approximated by N steps with equal reaction rate k :

$$\dot{x}_1 = kg(t) - kx_1 \quad (2.3)$$

$$\dot{x}_2 = kx_1 - kx_2 \quad (2.4)$$

$$\dot{x}_3 = kx_2 - kx_3 \quad (2.5)$$

$$\vdots$$

$$\dot{x}_N = kx_{N-1} \quad (2.6)$$

The mean delay is given as $\bar{\tau} = \frac{N}{k}$. Delay reactions are entered with reaction type 'D' and N as the options arguments, e.g.,

```
m = pwAddR(m, {'Stat_nuc'}, {'Stat_nuc_tau'}, {}, 'D', 5, [], {'delayRate'});
```

2.2.6 Compartments

Chemical reactions depend on the concentration of their reactants and products. If a molecule enters a different compartment, e.g. the nucleus, the concentration decrease in the cytoplasm may differ from the concentration increase in the nucleus. Hence, handling the compartment volume is a crucial aspect for modeling of chemical reaction networks. Within PottersWheel, compartment volume information is entered by help of the `pwAddC` function having the following syntax:

```
m = pwAddC(m, ID, size, outside, spatialDimensions, name, unit, constant);
```

Argument	Type	Description
ID	ID	Compartment ID
size	float	Size of the compartment
outside	ID	ID of the embracing compartment
spatialDimensions	integer	Dimension of the compartment
name	string	
unit	string	
constant	integer	constant size of compartment (1 yes, 0 no)

Currently, only the ID and size information are processed within PottersWheel. However, the other arguments are exported to SBML files. Example:

```
m = pwAddC(m, 'cytoplasm', 1);
m = pwAddC(m, 'nucleus', 1, 'cytoplasm');
```

2.2.7 Dynamic parameters

Similar like the dynamic variables, the value, fitting type and limits can be specified for the dynamic parameters:

```
m = pwAddK(m, ID, value, type, minValue, maxValue, unit, name, description);
```

Argument	Type	Description
ID	ID	
value	float	Default 0.1
type	string	fit type: 'global' (default), 'local', 'fix'
minValue	float	minimal value during fitting
maxValue	float	maximal value during fitting
unit	string	
name	string	descriptive name
description	string	

Example:

```
m = pwAddK(m, 'Stat_act', 1.2, 'global', 0, 100);
```

2.2.8 Driving inputs

Some players like the ligand concentration may be controlled externally and are represented in the model definition file as driving inputs. Linear, step, and exponential interpolations are supported.

```
m = pwAddU(m, ID, uType, uTimes, uValues, compartment, name, description);
```

Argument	Type	Description
ID	ID	
uType	string	'steps', 'linear', 'exponential'
uTimes	float vector	time values
uValues	float vector	step values
compartment	ID	
name	string	descriptive name
description	string	

Example: A step input of ligand 'L' starting at t=-100 at level 0, jumping at t=0 to 5 and decreasing at t=10 to level 2:

```
m = pwAddU(m, 'L', 'steps', [-100 0 10], [0 5 2]);
```

2.2.9 Sampling

A default sampling for simulation can be specified. A sampling from 0 to 100 in steps of 3 would be entered as:

```
m.t = 0:3:100;
```

2.2.10 Observables

Observables are functions of dynamical variables and correspond to measurable quantities in an experiment. The right hand side of the observation function is specified as an arbitrary mathematical expression of the IDs belonging to dynamic variables.

```
y = pwAddY(m, rhs, ID, scalingParameter, errorModel, noiseType, unit, name, desc);
```

Argument	Type	Description
rhs	string	Right hand side of the observation
ID	ID	
scalingParameter	ID	
errorModel	string	
noiseType	string	Currently only 'Gaussian' supported.
unit	string	
name	string	descriptive name
desc	string	Description of the observation

Example with Gaussian noise with a standard deviation of 10% relative to y plus 5% absolute (relative to $\max(y)$ over all y):

```
m = pwAddY(m, 'Stat + pStat + 2 * pStat_pStat', 'totalStat_obs', ...
            'scale_totalStat_obs', '0.1*y + 0.05*max(y)');
```

The right hand side may contain expressions like `any_Stat`. These expressions will be replaced by a sum of all species which contain `Stat`, independent of its current modification state. Species with two or more molecules of `Stat` get a multiplier, like `2 * pStat_pStat`. Hence, the above observation function can be shortly written as

```
m = pwAddY(m, 'any_Stat');
```

If only a subset of species is required, make use of the `only_pStat` or `only_Stat` syntax. The first will be replaced by the sum over all species containing `pStat`, the second by the sum over all species containing `Stat` (*ignoring* `pStat`). Example with all phospho Stats in the nucleus (`npStat`) and the cytoplasm (`cpStat`):

```
m = pwAddY(m, 'only_cpStat + only_npStat');
```

Finally, one is sometimes interested in species that are within a complex with other species. You can control this by the suffix `_withAny_` or `_withOnly_`. The expression `only_A_withAny_B` will be replaced by the sum over all species with a `B` independent of its modification and one or more unmodified `A`'s. Similarly, `any_A_withOnly_pB` matches all species containing `pB` and any modification of `A`. The multiplier is based in both cases on `A`, i.e. the species `A_A_pB` enters the sum as `2 * A_A_pB`.

The same holds for the derived variables.

2.2.11 Scaling parameters

Often, the concentration of the observable players can not be measured directly, but through a scaling factor as for example with Western Blotting. The scaling parameters behave like dynamic parameters, but are related to a set of observations and can be fitted separately. Please use the help function `pwAddS` with the syntax

```
m = pwAddS(m, ID, value, type, minValue, maxValue, unit, name, description);
```

Argument	Type	Description
ID	ID	Should start with 'scale_'
value	float	Default 1
type	string	'global', 'local' (default), or 'fix'
minValue	float	minimal value during fitting
maxValue	float	maximal value during fitting
unit	string	
name	string	descriptive name
description	string	

Example:

```
m = pwAddS(m, 'scale_Stat_obs', 1.2, 'global', 0, 100);
```

2.2.12 Derived variables

PottersWheel supports independent display and analysis of all dynamic variables and all observables. Beyond, it is sometimes useful to focus on a subset of dynamic variables or on interesting function of variables, especially while working with large models. For this, derived variables can be defined depending – similar as observables – on the dynamic variables.

```
m = pwAddZ(m, rhs, ID, unit, name, description);
```

Argument	Type	Description
rhs	string	Right hand side of the derived variable
ID	ID	
unit	string	
name	string	descriptive name
description	string	

If no ID is given, the right hand side is used as ID. Example:

```
m = pwAddZ(m, 'pStat + 2 * pStatDimer', 'totalPStat');
m = pwAddZ(m, 'Stat');
```

The same abbreviations like only `_pStat` as in the observables section (2.2.10) can be used.

2.2.13 Derived parameters

Like derived variables, also derived parameters can be defined. This is interesting for example if only the fraction of two parameters can unambiguously be determined.

```
m = pwAddP(m, rhs, ID, unit, name, description);
```

Argument	Type	Description
rhs	string	Right hand side of the derived parameter
ID	ID	
unit	string	
name	string	descriptive name
description	string	

The right hand side may contain IDs of dynamic parameters, dynamic variables, and scaling parameters. Since the set of scaling parameters may change if a new observation function is used for the model, please make only use of scaling parameters if you are working with the default observation, i.e. with the observation defined in the model definition file. Else, the behavior is not defined!

Example:

```
m = pwAddP(m, 'Stat_act / Stat_deact', 'Stat_act_over_deact');
```

2.2.14 Algebraic equations and rules

PottersWheel supports explicit algebraic equations, which are evaluated before every integration step. This allows for SBML assignment rules. The assigned quantities can be used as modifiers or parameters in the reactions.

```
m = pwAddRule(m, lhs, reactants, parameters, ruleSignature, type);
```

Argument	Type	Description
lhs	ID	ID (left hand side) of the assigned quantity.
reactants	string array	
parameters	string array	
ruleSignature	string	
type	string	Only 'assignment' supported (Default).

The list of reactants is interpreted similarly as in the reactions paragraph: r_1 in the rule signature corresponds to the first reactant and k_2 to the second parameter. Example:

```
m = pwAddRule(m, 'Stat_cyt', {'Stat', 'pStat', 'pStatDimer'}, {}, 'r1+r2+r3');
```

It is possible to use t in the rule signature, e.g. for time dependent compartment sizes like

```
m = pwAddRule(m, 'NucleusSize', {}, {'InitialNucleusSize', 'GrowthRate'}, 'k1 + k2*t');
```

The parameters 'InitialNucleusSize' and 'GrowthRate' are normal parameter and can be fitted.

2.2.15 Example: JakStat5 signaling pathway

The following example describes a simplified JakStat5 signaling pathway inspired by [27]. Not listed paragraphs get default values, for example the compartment paragraph.

```
function m = getModel()

m = pwGetEmptyModel();

% Meta information
m.ID          = 'JakStat';
m.name        = 'Simplified JakStat model';
m.description = 'Inspired by Swameye et al.';
m.authors     = {'Thomas Maiwald'};
m.dates       = {'2007-03-06'};
m.type        = 'PW-1-3';

% Dynamic variables
% x = pwAddX(m, ID, startValue, type, minValue, maxValue, unit, compartment, name);
m = pwAddX(m, 'R', 1, [], [], [], [], [], 'Receptor');
m = pwAddX(m, 'S', 10, [], [], [], [], [], 'cytoplasmatic Stat');
m = pwAddX(m, 'pS', 0, [], [], [], [], [], 'phosphorylated Stat');
m = pwAddX(m, 'nS', 0, [], [], [], [], [], 'nuclear Stat');

% Reactions
% m = pwAddR(m, reactants, products, modifiers, type, options, rateSignature, parameters);
m = pwAddR(m, {'R'},          {'pR'}, {'Epo'}, 'C', [], 'k1*r1*m1', {'R_act'});
m = pwAddR(m, {'pR'},         {'R'},  {},    'MA', [], [], {'R_deact'});
m = pwAddR(m, {'pR'},         {'iR'},  {},    'MA', [], [], {'R_inter'});
m = pwAddR(m, {'S'},          {'pS'}, {'pR'}, 'E',  [], [], {'S_act'});
m = pwAddR(m, {'pS','pS'},    {'pS_pS'}, {},    'MA', [], [], {'dimerization'});
m = pwAddR(m, {'pS_pS'},     {'npS_npS'}, {},    'MA', [], [], {'import'});
m = pwAddR(m, {'npS_npS'},    {'taunpS_taunpS'}, {}, 'D', 5, [], {'delay_rate'});
m = pwAddR(m, {'taunpS_taunpS'}, {'nS','nS'}, {}, 'MA', [], [], {'break'});
m = pwAddR(m, {'nS'},         {'S'},  {},    'MA', [], [], {'export'});

% Dynamic parameters
% m = pwAddK(m, ID, value, type, minValue, maxValue)
m = pwAddK(m, 'R_act',        0.115461, 'global', 0, 10000);
m = pwAddK(m, 'R_deact',     0.001150172, 'global', 0, 10000);
m = pwAddK(m, 'R_inter',     0.115545, 'global', 0, 10000);
m = pwAddK(m, 'S_act',       1.35761, 'global', 0, 10000);

% Default driving input
% m = pwAddU(m, ID, uType, uTimes, uValues)
m = pwAddU(m, 'Epo', 'steps', [-100 0 10], [0 1 0]);

% Default sampling time points
m.t = 0:3:100;

% Observables
% pwAddY(m, rhs, ID)
m = pwAddY(m, 'pR');
m = pwAddY(m, 'S + pS + 2 * pS_pS', 'totalCytoS');
```

```

m = pwAddY(m, 'pS + 2 * pS_pS',      'totalPhosphoS');

% Derived variables
% pwAddZ(m, rhs, ID)
m = pwAddZ(m, 'R');
m = pwAddZ(m, 'pR');
m = pwAddZ(m, 'iR');
m = pwAddZ(m, 'pS + 2 * pS_pS', 'tpCytoS');

```

2.3 Using the model wizard

The model creation wizard is a graphical user interface which provides a commented template depending on the number of reactions, observations, etc. Start the wizard by pressing 'New' or typing `pwCreateModelGUI`.

2.4 ODE based models

ODE based PottersWheel model definition files have a similar structure as reaction based files. Only the reactions part is replaced with an ODE part.

Example:

```

ODE          = {};
ODE{end+1} = 'd(A)/dt = - A_to_B * A + C_to_A * C';
ODE{end+1} = 'd(B)/dt = + A_to_B * A - B_to_C * B';
ODE{end+1} = 'd(C)/dt = + B_to_C * B - C_to_A * C - C_to_D * C';
ODE{end+1} = 'd(D)/dt = + C_to_D * C';
m.ODE       = ODE;

```

2.4.1 Automatic reconstruction of reaction schemes

PottersWheel tries to reconstruct from a system of differential equations the underlying reaction scheme. This is currently only possible if no brackets are used in the differential equations.

2.5 SBML based models

SBML, the systems biology markup language, has been designed to enable a standardized way to express, store and exchange kinetic models based on reaction networks [12, 15]. Many programs are compatible with or provide interfaces for SBML, as e.g. the Systems Biology Matlab toolbox from Schmidt et al. [25].

Most SBML files level 2-1 can be converted into PottersWheel reaction based model definition files. Events and function definitions are not yet supported. Since observation

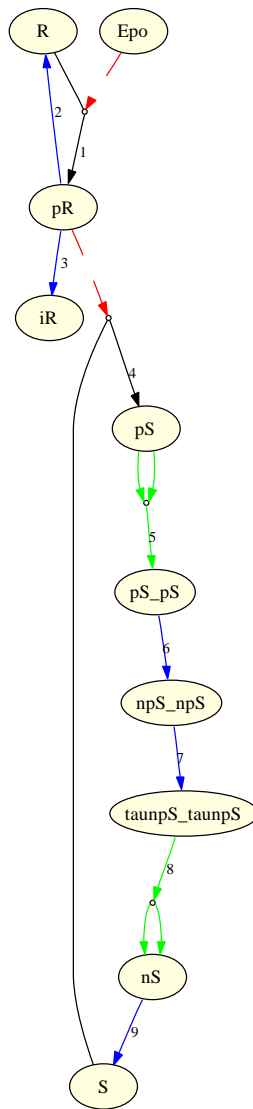


Figure 2.1: Model graph for the simplified JakStat signaling pathway.

Model properties can be changed at any time.
This dialog will set up the structure of the model definition file for your convenience.

Model ID:

Filename:

Model name:

Author(s):

Description:

Number of reactions:	<input type="text" value="3"/>	Number of observables:	<input type="text" value="1"/>
Number of dynamic variables:	<input type="text" value="3"/>	Number of derived variables:	<input type="text" value="0"/>
Number of dynamic parameters:	<input type="text" value="3"/>	Number of scaling parameters:	<input type="text" value="0"/>
Number of compartments:	<input type="text" value="0"/>	Number of derived parameters:	<input type="text" value="0"/>
Number of driving input players:	<input type="text" value="1"/>	Number of rules:	<input type="text" value="0"/>

include comments

Figure 2.2: Model creation wizard

functions and driving functions are not included in SBML, after conversion a little manual work on the created model definition file is required. The SBML Matlab toolbox from Sarah Keating et al. is required for SBML model import [17]. Use the menu button 'File - Convert SBML to PW'. Export to SBML can be accomplished without any toolbox. Since driving functions do not exist in SBML, they enter the reactions as normal species.

Please check the converted model definition file.

2.6 Power law models

Power law models are based on kinetic laws with fractional exponents. Use custom kinetics in order to apply power law models within PottersWheel, like

```
pwAddR(m, {'A'}, {'B'}, {}, 'C', [], [], 'k1 * r1^k2');
```

The exponents behave like normal parameters and can also be fitted. Make sure to specify reasonable limits for the exponent during fitting, e.g. between 0.5 and 100. Otherwise the integrator could get problems to integrate the system.

2.7 Model visualization

Reaction and SBML based models can be visualized and saved as png, ps or svg files:

blue arrows	reactions comprising only one reactant and one product
black arrows	reactions which depend on one or more modifiers, e.g. enzymes
red dashed arrows	effect of modifiers
green arrows	reactions comprising several reactants and/or products
brown arrows	players linked together by an assignment rule

The reaction number is displayed next to each reaction arrow. Optionally, the parameter values can be displayed, too. For ODE based models, the reaction network can automatically be derived if no brackets are used in the differential equations.

Chapter 3

Rule based modeling and combinatorial complexity

This chapter assumes use of the naming conventions as described in section 2.1.

Suppose a protein E which acts enzymatically on the reaction $A \rightarrow B$ independent on the current binding state of E , E_1, \dots, E_n , itself. An exact mathematical ODE formulation requires to distinguish all states of E which leads to n very similar reactions. If A has also m different states which behave the same for the considered reaction $A \rightarrow B$, already $m \cdot n$ reactions have to be formulated. This phenomenon is called *combinatorial complexity* [2, 3, 10, 4]. In order to keep the number of entered and maintained reactions small, PottersWheel supports *rule based* reactions (which should not be mistaken with assignment rules). The above example with only one species A would be formulated as



```
pwAddR(m, {'A'}, {'B'}, {'*_E_*'}, [], [], 'k1*r1*m1');
```

As in regular expressions, the $*$ represents any other species that are in a complex with E . Considering also all complexes involving A :



```
pwAddR(m, {'*_A_*'}, {'B'}, {'*_E_*'}, [], [], 'k1*r1*m1');
```

In the second case, the complex with A does not react to B in general, but rather to the original complex with A being replaced by B . Hence, the information of the reactant structure must be *restored* for the product. This is done within PottersWheel by tagged expressions:

```
pwAddR(m, {'<1:*>_A_<2:*>'}, {'<1>_B_<2>'}, {'*_E_*'}, [], [], 'k1*r1*m1');
```

Not only different complexes involving a molecule can be distinguished, but also different *modification* states of one molecule itself, like the phosphorylation state. The Erk molecule exists for example in the states **Erk**, **pErk**, **ppErk** with no, one, or two phosphorylations. In order to consider all phosphorylated Erk molecules, one can write **p|ppErk**. All Erk

modifications can be expressed as `*Erk` or `p|pp|Erk`.

The following patterns are supported:

Pattern	Description	Examples
<code>*A</code>	All modifications of A and A	A, xA, yA, zA
<code>+A</code>	Only modified A's	xA, yA, zA
<code>x y A</code>	Specific modifications and A	A, xA, yA
<code>x yA</code>	Only specific modifications	xA, yA
<code>*_A_*</code>	All complexes with A and free A	A, B_A, A_C, B_A_C
<code>A_*</code>	All complexes with <i>left</i> A and free A	A, A_C, A_C_D
<code>*_A</code>	All complexes with <i>right</i> A and free A	A, B_A, E_B_A
<code>+_A_+</code>	All complexes with A	B_A, A_C, B_A_C
<code>A_+</code>	All complexes with <i>left</i> A	A_C, A_C_D
<code>+_A</code>	All complexes with <i>right</i> A	B_A, E_B_A
<code>?_A_?</code>	Complexes with A and up to one neighbor	B_A, A_C, B_A_C
<code>A_?</code>	Complexes with <i>left</i> A and up to one neighbor	A, A_C
<code>?_A</code>	Complexes with <i>right</i> A and up to one neighbor	A, B_A

These patterns can be combined arbitrarily.

Example:

```
m = pwAddR(m, {'Mek'}, {'ppMek'}, {'activeRaf'});
m = pwAddR(m, {'ppMek'}, {'Mek'}, {});
m = pwAddR(m, {'<1:*>_Erk_<2:*>'}, {'<1>_ppErk_<2>'}, {'*_ppMek_*'});
m = pwAddR(m, {'<1:*>_ppErk_<2:*>'}, {'<1>_Erk_<2>'}, {});
m = pwAddR(m, {'activeRaf'}, {'inactiveRaf'}, {'*_ppErk_*'});
m = pwAddR(m, {'inactiveRaf'}, {'activeRaf'}, {});
```

Very often, a single letter represents a certain modification of a molecule. Taking for example a receptor *R* with three different phosphorylation sites and *p* corresponding to a phosphorylation and *o* to a free site, the following eight states exist: *oooR*, *pooR*, *opoR*, *oopR*, *ppoR*, *popR*, *oppR*, *pppR*. In order to dephosphorylate the first site independent of sites two and three, the following syntax can be used:

```
m = pwAddR(m, {'p<1:..>R'}, {'o<1>R'}, {'Phosphatase'});
```

For the second site it is a bit more complicated:

```
m = pwAddR(m, {'<1:..>p<2:..>R'}, {'<1>o<2>R'}, {'Phosphatase'});
```

Each time a single dot '.' represents one arbitrary letter as in regular expressions.

Chapter 4

Manually investigating model properties

4.1 PottersWheel Equalizer

In order to rapidly investigate the dynamic properties of a model, each parameter can be changed with a slider. The effects of parameter changes are visualized in real time. Left sliders are used for fine-tuning, i.e. change the mantisse, right sliders have a strong impact on the parameter value - they change the exponent of a 'mantisse * 10^{exponent}' representation.

Slider values can also be changed via the keyboard: Use number keys (1, 2, 3, ..., 0) to select a slider pair. The left arrow key then selects a left, fine tuning slider. The right arrow key selects a right, coarse slider. Up and down keys change the slider position. Edit boxes display the current parameter value and can be edited directly, too. If more than ten parameters belong to the active model (or combination of models), a slider pair can be linked to a specific parameter with the list-box slider.

Buttons:

PW Activate PottersWheel main window

Arrange Arranges figure positions on screen.

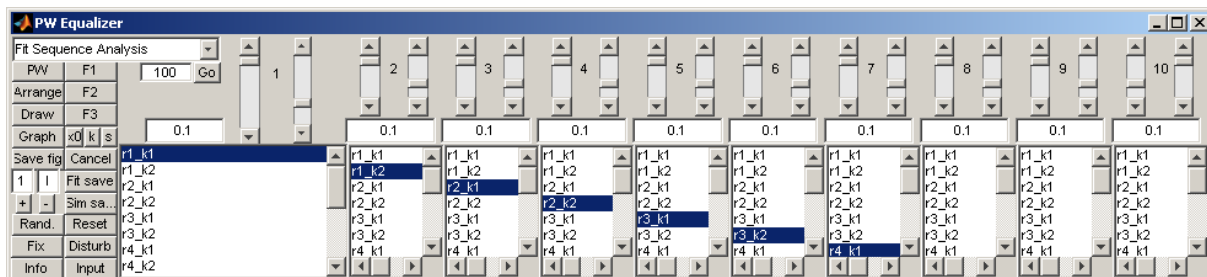


Figure 4.1: PottersWheel Equalizer

- Draw Integrates and draws the models with the current parameter values. Uses the line and marker style as specified in the configuration dialog.
- Graph All combined models are visualized within the dotted add on if installed.
- Save fig The combined models are integrated, drawn and the figures are saved in a 'Plots' folder. Figure types can be specified in the configuration dialog.
- +/- Shifting of the selected parameters from the parameter list up and down. This can also be done with the keyboard (+ and - keys).
- Rand Displays the list of parameters which are randomized, if a parameter disturbance is applied. A subgroup of parameters can be selected to be excluded from randomization.
- Fix Displays the list of parameters which are not fitted. A subgroup can be selected.
- Info The list of parameters with current value and fitting state is displayed in the command window.
- Input Starts the input designer, where the characteristics of all driving input functions can be changed.
- F1 The data is fitted once.
 - F2 The data is fitted n times, with n specified in the configuration dialog. Every fit starts from initial parameter values plus disturbance. The best fit is used after all fits.
 - F3 Same as F2, but each fit starts at the best fit of the current fit sequence plus disturbance.
 - x0 Only start values are fitted
 - k Only dynamic parameters are fitted
 - s Only scaling parameter are fitted
- Cancel The current fit or fit sequence is cancelled (Please press several times. Matlab ignores user interaction in very busy situations. Fitting is a very busy situation ;-)
If you are not successful, use Control + C. This may lead to a loss of information.)
- Fit save The current parameter values are saved as manual fit to the fit history.
- Sim save The current parameter values will be used for simulations of the model.
- Reset The parameter values are reset to their initial values as specified in the model definition file or in the IDSO dialog.
- Disturb Parameter values are disturbed, the model is reintegrated and plotted. Strength of disturbance can be specified in the configuration dialog.

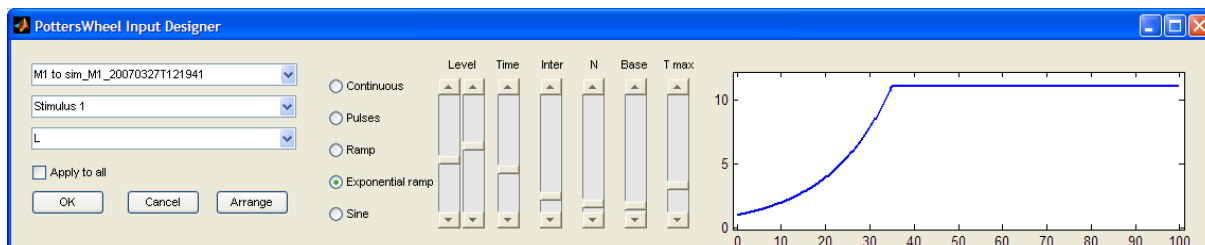


Figure 4.2: PottersWheel Input Designer

Go The selected analysis from the combobox is executed. The best fits of the last fit sequence are analyzed. The percentage can be specified in the text box.

4.2 PottersWheel Input Designer

The driving input designer overwrites temporarily selected driving functions of the currently combined models. Continuous, pulsed, linear, exponential, and sine stimulations are available. Characteristics like amplitude level, time to maximum, time between pulses, number of pulses or sine waves, base level, maximum time for sine waves can be changed. The effect on the models is shown directly. By pressing 'OK', the temporary change is made permanent.

The input designer is very helpful for investigating the expected result of different stimulation experiments. It helps to identify experimental designs for model discrimination.

Chapter 5

IDSO

The IDSO dialog – instructions, dynamical parameters, start values and observations – allows to change instructions comprising sampling and stimuli, values and limits for parameters k and start values x_0 and the observation functions. All changes can be saved to hard disk and reloaded later.

5.1 Instructions

Either via the IDSO dialog or with the function `pwAddInstructions(filename)`; an experimental setting – the instructions – can be added to a selected model. The instructions file is constructed as a Matlab function and contains for each stimulation a list of time points and driving functions for all required driving inputs. Additional driving inputs are ignored.

```
function instructions = getInstructions()

% pwGetDrivingFunction(uType, uTimes, uValues)

instructions = [];

instructions(end+1).t      = [0:10 15:5:60];
instructions(end).stimuli.TGFb = pwGetDrivingFunction('steps', [-100 0], [0 1]);

instructions(end+1).t      = [0:10 15:5:40];
instructions(end).stimuli.TGFb = pwGetDrivingFunction('steps', [-100 0 10], [0 1 0]);

instructions(end+1).t      = [0 1 2 3 4 5 6 7 8 9 10 12 14 16 18 20 40 50 60 70 90];
instructions(end).stimuli.TGFb = pwGetDrivingFunction('steps', [-100 0 20], [0 1 0.5]);
```

5.2 Dynamical parameters

Similarly as for the instructions, dynamical parameters k and their limits for fitting, can be added with the IDSO dialog or via `pwAddDynamicalParameters(filename)`. The

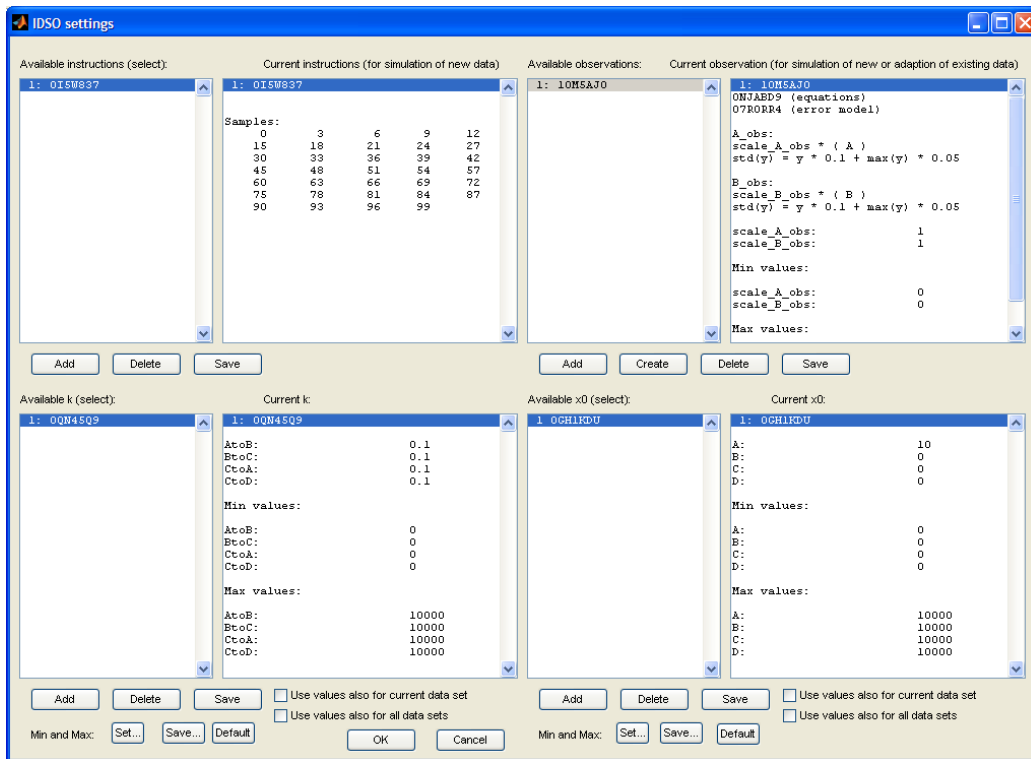


Figure 5.1: IDSO dialog

file has to be a Matlab function returning k , $kMin$ and $kMax$. Example:

```
function [k, kMin, kMax] = myValues()

k.StatProd    = 0.4;
kMin.StatProd = 0;
kMax.StatProd = 100;

k.StatDeg     = 0.9;
kMin.StatDeg  = 0;
kMax.StatDeg  = 100;
```

5.3 Start values

```
pwAddStartValues(filename);
```

Adds start values saved in filename to the currently selected models. The file has to be a Matlab function returning $x0$, $x0Min$ and $x0Max$. Example:

```
function [x0, x0Min, x0Max] = myValues()

x0.Stat       = 10;
```

```
x0Min.Stat = 0;  
x0Max.Stat = 1000;  
  
x0.pStat = 0;  
x0Min.pStat = 0;  
x0Max.pStat = 100;
```

5.4 Observation functions

```
pwAddObservations(filename);
```

The observation file is constructed as a Matlab function and contains a list of observation functions defined similarly as in a model definition file. Example:

```
function y = getObservations()  
  
y = pwGetY('Stat');  
y(end+1) = pwGetY('pStat + 2 * pStat_pStat', 'total_pStat');  
y(end+1) = pwGetY('Stat + pStat + 2 * pStat_pStat', 'total_Stat');
```


Chapter 6

Saving and loading simulated and experimental data

6.1 Format of data files

Data files are ASCII text or Excel files and contain meta information and numerical data structured as a matrix. Each line of the meta information section starts with '#'. Three subsections are distinguished:

- free user-specific information,
- driving functions, i.e. the used stimulations in the corresponding experiment or simulation,
- column names of the data matrix.

It is recommended to use only the following characters: A-Za-z_,0-9 #.'-[]() :. In ASCII files, columns have to be separated by a tabulator. The filename should contain only [a-zA-Z0-9] and the underscore _.

6.1.1 User-specific information

This section can be used for individual information describing the data set.

6.1.2 Driving functions

In order to specify under which experimental conditions the data has been collected, all external driving functions have to be specified. The **first** column must contain the 'MCode' keyword. The **second** column comprises Matlab commands and introduces either the name of the external input or the shape of a certain stimulus, e.g. for an experiment with two different stimuli:

```
# MCode drivingFunctions(1).name = 'Ext';  
# MCode drivingFunctions(1).stimuli(1)=pwGetDrivingFunction('steps',[-1 0 10],[0 5 1]);  
# MCode drivingFunctions(1).stimuli(2)=pwGetDrivingFunction('steps',[-1 0], [0 4]);
```

The help function `pwGetDrivingFunction(uType, uTimes, uValues)` has a similar syntax as `pwAddU` used in model definition files: The argument 'uType' describes the used interpolation type, either 'steps' or 'linear'. The interpolation connects points given by time points `uTimes` and values `uValues`. Stimulus 1 of the example corresponds to a stepwise driving input, starting at `t=-1` with value 0, jumping at `t=0` to value 5 and decreasing at `t=10` to value 1. The second stimulus represents a continuous stimulation starting at `t=0` with a value of 4. If two or more different ligands are used, the driving input section could be like follows:

```
# MCode drivingFunctions(1).name = 'Epo';
# MCode drivingFunctions(1).stimuli(1)=pwGetDrivingFunction('steps',[-1 0 10],[0 5 1]);
# MCode drivingFunctions(1).stimuli(2)=pwGetDrivingFunction('steps',[-1], [0]);
# MCode drivingFunctions(1).stimuli(3)=pwGetDrivingFunction('steps',[-1 0], [0 4]);
# MCode drivingFunctions(2).name = 'IL6';
# MCode drivingFunctions(2).stimuli(1)=pwGetDrivingFunction('steps',[-1 0 10],[0 5 1]);
# MCode drivingFunctions(2).stimuli(2)=pwGetDrivingFunction('steps',[-1 0], [0 4]);
# MCode drivingFunctions(2).stimuli(3)=pwGetDrivingFunction('steps',[-1], [0]);
```

It is important, that each driving function, here 'Epo' and 'IL6', has the same number of stimuli, in this case three. The arguments ('steps', [-1], [0]) correspond to a zero input.

6.1.3 Column names

The last meta information line before the data matrix contains the column names. The prefix 'parCol-' specifies parameter columns, whereas 'xCol-' belongs to the column of the independent variable, usually time. Columns starting with 'ignore-' are ignored. All other columns contain data of directly observed variables or functions of variables. It is not important if more data columns are available than required by the observation functions of a model – the additional columns are neglected. Also the order of the columns is not important – the user has to determine when loading the data file which column corresponds to which model observation. However, if the column names corresponds unambiguously to the model observations, the mapping is done automatically (compare 6.3).

Two columns are obligatory: The 'parCol-Stimulus' column and the 'xCol-...' column of the independent time variable. The stimulus column contains integers values linking one line of the data matrix to a specific stimulus number. If e.g. row 5 contains data of stimulus 2, then the 'parCol-Stimulus' column would show the number 2. The time column contains the experimental time values when the data of the row has been collected. An example with two data columns:

```
# parCol-Stimulus xCol-Time A_obs B_obs
```

6.1.4 Data matrix and complete example

The numerical data is given as a matrix with numbers. Together with the meta information, a data file could look like this:

```

# Test file with experimental data.
# Stimulation with Epo.
# Experiment: Adam Smith.
# Two stimulations: 1 Continuous, 2 Pulse (5 min)
# Two observations: A and B
#
# Driving functions:
# MCode drivingFunctions(1).name = 'Epo';
# MCode drivingFunctions(1).stimuli(1)=pwGetDrivingFunction('steps',[-1 0], [0 6]);
# MCode drivingFunctions(1).stimuli(2)=pwGetDrivingFunction('steps',[-1 0 5],[0 6 0]);
#
# parCol-Stimulus   xCol-Time   A_obs   B_obs
1                   0           11.707  0.38078
1                   3           4.17596  3.51887
1                   6           1.71684  6.62109
1                   9           1.83223  6.56341
1                   12          1.34706  6.32869
1                   15          0.894643 6.08994
1                   18          1.69888  7.73398
1                   21          2.2955   6.85444
1                   24          0.982606 7.30223
1                   27          0.775678 6.75022
1                   30          1.13776  6.55881
2                   0           1.6924   7.3692
2                   3           0.725329 7.65743
2                   6           0.493984 9.02214
2                   9           1.1537   5.29255
2                   12          1.17226  5.56614
2                   15          1.49826  7.60935
2                   18          0.81421  6.06208
2                   21          1.07511  4.98056
2                   24          0.885173 6.61405
2                   27          1.0894   7.76428
2                   30          2.06378  6.83843

```

6.2 Saving of simulated data

The current data files of all selected models can be saved within the main interface via 'Models' → 'Save data set'.

6.3 Loading of external data and mapping of observables

Observation functions as given for example in a model definition file must not necessarily correspond to column names of an external data file. If the file is loaded for the first time, the user is inquired in a graphical user interface to map all observations to one column of the file. The mapping can be saved to a `*_connected_to_*.mat` file in the current folder and will be used automatically in the future, if the same observations, data file and working folder are used.

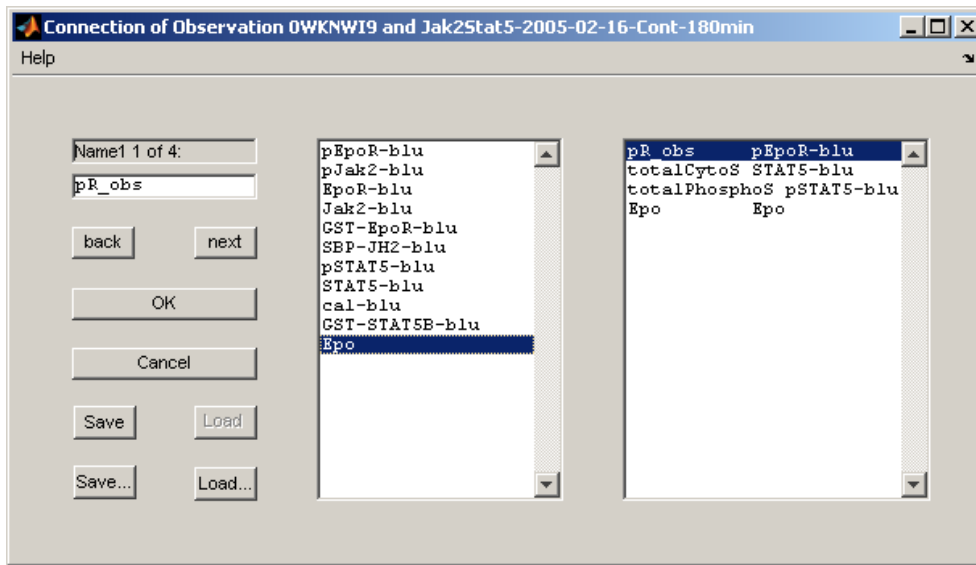


Figure 6.1: Mapping of data columns to observables

As described in 6.1.3, the automatic mapping is also used if the observation functions have the same name as columns given in the file.

6.4 Investigating external data sets efficiently

Two approaches exist to explore data saved in external Excel or ASCII files:

- Interactively via the Data Viewer tool
- Statically via `pwPropertiesOfDataFile`

6.4.1 Data Viewer

The Data Viewer starts either by typing `Data_Viewer` or with the PottersWheel menu 'Tools → DataViewer'. It shows all suitable files in the currently selected folder and all suitable workspace variables which can be interpreted as data objects. If a new object or file is selected, all available columns appear automatically and can be selected for plotting. One-click saving into important image formats helps to document interactive findings.

6.4.2 Plots including driving inputs

In addition to the Data Viewer, the static `pwPropertiesOfDataFile` function supports visualization of driving input functions and different stimuli. The resulting figure and summary can be added to the current report.

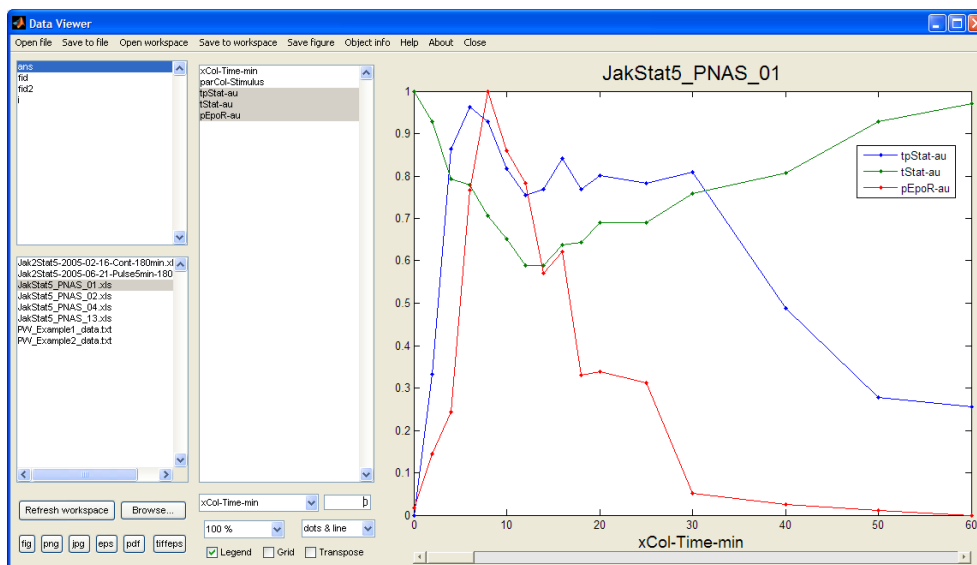


Figure 6.2: Data Viewer showing Western Blotting data of the JakStat5 signaling pathway.

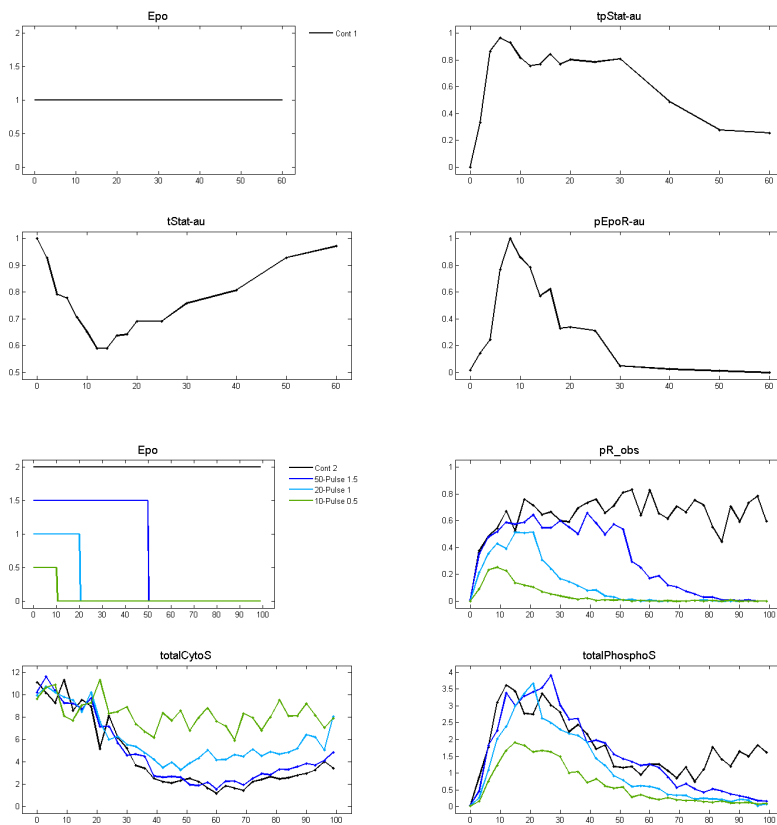


Figure 6.3: Top: The same data as shown above with the Data Viewer but now with the driving input function Epo. Bottom: Simulated data with four different stimulations.

Chapter 7

Fitting

This chapter begins with a concise repetition of important aspects of numerical optimization as given for example in [5, 6, 7, 21, 22, 26] and references therein.

7.1 Introduction

In unconstrained optimization, an objective function is minimized with no restrictions to one of its arguments, mathematically expressed via

$$\min_x f(x), \quad (7.1)$$

with a real vector $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \rightarrow \mathbb{R}$. A point x^* is a *global minimizer* if $f(x^*) \leq f(x) \forall x$. For a *local optimizer*, a neighborhood \mathcal{N} of x^* exists such that $f(x^*) \leq f(x) \forall x \in \mathcal{N}$. A *strict optimizer* fulfills $f(x^*) < f(x) \forall x \neq x^*$. Local minima can be identified for twice differentiable functions f by examining the gradient $\nabla f(x^*)$ and the Hessian $\nabla^2 f(x^*)$, based on

Taylor's Theorem. *Given a twice continuously differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $p \in \mathbb{R}^n$. Then*

$$f(x + p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x + tp) p, \quad (7.2)$$

for some $t \in (0, 1)$.

Sufficient conditions for x^* being a strict local minimizer of f are that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. When f is convex, then x^* is also a global optimizer.

7.2 Optimization strategies: line search and trust region

Most optimization algorithms require an initial value x_0 and try iteratively to reach x^* via a sequence x_k by using information of f at x_k and earlier iterates x_0, x_1, \dots, x_{k-1} . A new iterate x_{k+1} should usually decrease the value of f . Two main strategies to construct x_{k+1} are distinguished: line search and trust region.

7.2.1 Line search

In the line search strategy, the algorithm chooses a direction p_k and searches along this one-dimensional space a new iterate which decreases f . The distance of the new iterate x_{k+1} to x_k along p_k can be approximately found by minimizing

$$\min_{\alpha > 0} f(x_k + \alpha p_k). \quad (7.3)$$

Steepest-descent

The *steepest-descent* direction $p_k = -\nabla f(x_k)$ is an intuitive choice for the search direction and requires only calculation of the gradient of f and no higher derivatives. However, in some cases this approach converges very slowly. In general, the search direction has often the form

$$p_k = -B_k^{-1} \nabla f_k, \quad (7.4)$$

where B_k is a symmetric and nonsingular matrix. In the steepest descent method, B_k is the identity matrix I . In Newton's method, B_k is the exact Hessian $\nabla^2 f(x_k)$. In quasi-Newton methods, B_k is an approximation to the Hessian that is updated at every iteration. Since

$$p_k^T \nabla f_k = -\nabla f_k^T B_k^{-1} \nabla f_k < 0, \quad (7.5)$$

the angle between p_k and ∇f_k is smaller than $\pi/2$ and this way it is guaranteed that p_k is a descent direction.

Newton's method

The second order Taylor series approximation to $f(x_k + p)$ reads

$$f(x_k + p) \approx f_k + p^T \nabla f_k + \frac{1}{2} p^T \nabla^2 f_k p \equiv m_k(p) \quad (7.6)$$

and defines the quadratic model $m_k(p)$. In order to find a direction p_k which minimizes $f(x_k + p)$ we set the derivative of $m_k(p)$ to zero and obtain

$$0 = \nabla m_k(p) \quad (7.7)$$

$$= \nabla \left(f_k + p^T \nabla f_k + \frac{1}{2} p^T \nabla^2 f_k p \right) \quad (7.8)$$

$$= 0 + \nabla f_k + \nabla^2 f_k p \quad (7.9)$$

$$\Rightarrow p_k^N = -(\nabla^2 f_k)^{-1} \nabla f_k. \quad (7.10)$$

Methods that use the Newton direction have a fast rate of local convergence, typically quadratic. The main drawback is the need for the Hessian $\nabla^2 f(x)$, which is tackled by Quasi-Newton methods.

Quasi-Newton methods

Instead of using the true Hessian, an approximation B_k is used and updated after each iteration taking into account information of the last update step. The most popular recursive calculation of B_k are the *symmetric-rank-one* (SR1) formula

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \quad (7.11)$$

with

$$s_k = x_{k+1} - x_k \text{ and } y_k = \nabla f_{k+1} - \nabla f_k, \quad (7.12)$$

and the *BFGS* formula, named after its inventors Broyden, Fletcher, Goldfarb, and Shanno, defined by

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \quad (7.13)$$

In the second case, a rank-two matrix is the difference between B_k and B_{k+1} . The quasi-Newton search direction is then given by

$$p_k = -B_k^{-1} \nabla f_k. \quad (7.14)$$

Nonlinear conjugate gradient methods

The search directions of nonlinear conjugate gradient methods have the form

$$p_k = -\nabla f(x_k) + \beta_k p_{k-1}, \quad (7.15)$$

with $\beta_k \in \mathbb{R}$ so that p_k and p_{k-1} are conjugate. A set of vectors p_0, p_1, \dots, p_l is said to be conjugate with respect to the symmetric positive definite matrix A , if

$$p_i^T A p_j = 0, \quad \forall i \neq j. \quad (7.16)$$

The origin of this approach are systems of linear equations $Ax = b$ with a symmetric and positive definite matrix A . Solving this linear system is equivalent to minimizing the convex quadratic function defined by

$$\phi(x) = \frac{1}{2} x^T A x + b^T x. \quad (7.17)$$

This approach is in general much more effective than steepest descent and almost as simple to compute. The convergence rate of Newton or quasi-Newton methods is not achieved, but it is not required to store matrices.

7.2.2 Trust region

The trust region approach uses a model m_k to approximate f in the neighborhood of iterate x_k . Then, the subproblem

$$\min_p m_k(x_k + p) \quad (7.18)$$

is solved with $x_k + p$ lying in the neighborhood – the *trust region*. If f is not decreased sufficiently, a smaller trust region is chosen and 7.18 is solved again. The trust region is usually ball-, elliptical-, or box-shaped with radius Δ_k . The model m_k is often a quadratic function of the form

$$m_k(x_k + p) = f_k + p^T \nabla f_k + \frac{1}{2} p^T B_k p. \quad (7.19)$$

In case of a vanishing B_k , the minimization problem can be solved directly:

$$p_k = -\Delta_k \frac{\nabla f_k}{\|\nabla f_k\|}. \quad (7.20)$$

This corresponds to a steepest-descent step with a step length determined by the trust-region radius. With B_k being the Hessian or an approximation of it, the trust-region Newton method and trust-region quasi-Newton methods are obtained, respectively.

7.3 Nonlinear least-square problems

In least-square problems, the objective function f has the form

$$f(x) = \frac{1}{2} \sum_{j=1}^m r_j^2(x), \quad (7.21)$$

with r_j being a sufficiently smooth function from \mathbb{R}^n to \mathbb{R} , for example residuals of observation j between m data points and model values depending on n parameters x . The r_j can be summarized in a residuals vector $r: \mathbb{R}^n \rightarrow \mathbb{R}^m$ with

$$r(x) = (r_1(x), \dots, r_m(x))^T. \quad (7.22)$$

Then, the derivatives of $f(x)$ can be expressed via the Jacobian of r :

$$J(x) = \begin{bmatrix} \partial r_j \\ \partial x_i \end{bmatrix}_{j=1, \dots, m, i=1, \dots, n}. \quad (7.23)$$

It holds

$$\nabla f(x) = \sum_{j=1}^m r_j(x) \nabla r_j(x) = J(x)^T r(x) \quad (7.24)$$

$$\nabla^2 f(x) = \sum_{j=1}^m \nabla r_j(x) \nabla r_j(x)^T + \sum_{j=1}^m r_j(x) \nabla^2 r_j(x) \quad (7.25)$$

$$= J(x)^T J(x) + \sum_{j=1}^m r_j(x) \nabla^2 r_j(x). \quad (7.26)$$

The Hessian of f divides into two terms with the first term depending only on the Jacobian. The second term is often very small and can be neglected, because of near-linearity of f near the solution ($\nabla^2 r(x)$ small) or because of small residuals (r_j small).

7.3.1 The Gauss-Newton method

The Gauss-Newton method can be viewed as a modification of Newton's line search method by exploiting the structure of the gradient ∇f and Hessian $\nabla^2 f$. Instead of solving the Newton equations $\nabla^2 f(x_k)p = -\nabla f(x_k)$, the second term of $\nabla^2 f(x_k)$ is neglected, so that

$$J_k^T J_k p_k^{GN} = -J_k^T r_k. \quad (7.27)$$

The speed of convergence of Gauss-Newton near a solution x^* compared to a full Newton-method depends on how much the leading term $J^T J$ dominates the second-order term in the Hessian (7.26).

7.3.2 The Levenberg-Marquardt method

The Levenberg-Marquardt method can be derived by replacing the line search strategy of Gauss-Newton with a trust-region approach.

7.3.3 Large-residual problems

In large-residual problems, the second order of the Hessian $\nabla^2 f(x)$ is too significant to be ignored. Gauss-Newton and Levenberg-Marquardt have in the large-residual case only a linear asymptotic convergence – slower than the superlinear convergence rate attained by algorithms for general unconstrained problems, such as Newton or quasi-Newton. Since often it is unknown beforehand whether a problem will turn out to have small or large residuals at the solution, it is reasonable to consider *hybrid algorithms*, which would behave like Gauss-Newton or Levenberg-Marquardt if the residuals turn out to be small, but switch to Newton or quasi-Newton if the residuals at the solution appear to be large.

7.4 Simulated annealing

Lester Ingber developed a fast simulated annealing algorithm [16]. It is an algorithm to statistically find the best global fit of a nonlinear non-convex cost-function. The algorithm permits an annealing schedule for temperature T decreasing exponentially in annealing-time k , $T = T_0 \exp(-ck^{1/D})$.

7.5 Genetic algorithm

The genetic algorithm uses the Augmented Lagrangian Genetic Algorithm (ALGA) to solve nonlinear constraint problems. The Augmented Lagrangian Genetic Algorithm (ALGA) attempts to solve a nonlinear optimization problem with nonlinear constraints, linear constraints, and bounds. Bounds and linear constraints are handled separately from nonlinear constraints. A subproblem is formulated by combining the fitness function and nonlinear constraint function using the Lagrangian and the penalty parameters. A sequence of such optimization problems are approximately minimized using the genetic

algorithm such that the linear constraints and bounds are satisfied. For further details compare with the genetic algorithm toolbox and with [8, 9].

7.6 Optimization function

PottersWheel uses the weighted χ^2 value for parameter optimization,

$$\chi^2 = \sum_{i=1}^N \frac{(y_{Model}(i) - y_{Meas}(i))^2}{\sigma_{Meas}^2(i)}. \quad (7.28)$$

N is the number of all data points and $\sigma_{Meas}(i)$ the standard deviation of measurement i , $y_{Meas}(i)$.

It is planned to provide an interface for arbitrary, custom optimization functions. Please contact maiwald@fdm.uni-freiburg.de if you need this functionality.

7.7 Parameter limits

For each parameter a minimum and maximum value can be specified for example directly in the model definition file or within the IDSO dialog.

7.8 Confidence intervals

Two ways exist to calculate confidence intervals for fitted parameter values:

1. The model is fitted several times with different initial parameter values and the variance and covariance of the fitted parameters are calculated.
2. The covariance can be derived from the curvature of the optimization function in the local or global minimum, which is essentially given by the Hessian. A higher curvature leads to a better defined minimum and smaller confidence intervals for the parameters.

PottersWheel supports both ways to determine the parameter covariance matrix. However, the first way seems to be more robust and the second way currently requires use of the trust region optimization approach.

7.9 Fitting in logarithmic parameter space

If parameter limits extend several orders of magnitude, it may be reasonable to fit in logarithmic parameter space, which can easily be activated in the configuration dialog. This leads to a better representation of small parameter values.

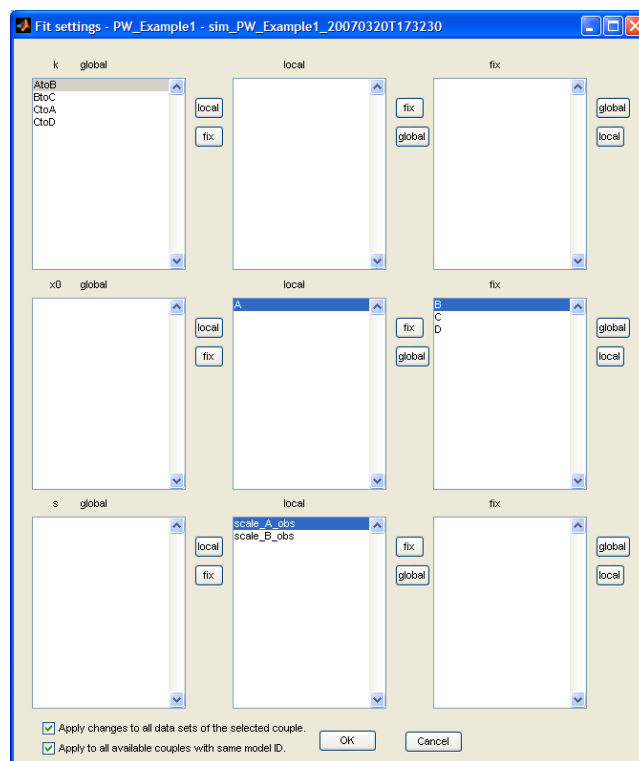


Figure 7.1: Fit settings dialog.

7.10 Single and multi-experiment fitting

The statistical power to discriminate competing model hypotheses increases strongly if different experiments are modeled simultaneously – multi experiment fitting. Global and local parameters are distinguished. Global parameters will be fitted to the same value for each experiment. They reflect structural parameters with a unique meaning. Local parameters on the other hand get different values depending on the experiment. This occurs for example for scaling parameters of observation functions for relative measurement techniques like Western Blotting or for initial values of protein concentrations.

Besides in the model definition file, the fit settings dialog allows to specify local and global parameters. Alternatively, parameters can also be fixed – they are not fitted any more.

7.11 Single fit and fit sequences

A combination of model-data-couples can be fitted once – a single fit, F1 – or several times in a fit sequence. In the second case the parameter values are disturbed after each fit. They are either chosen from a quasi-random-distribution between the minimum and maximum parameter values or drawn from an exponential distribution around p_0 :

$$p_{new} = p_0 \cdot 10^{s \cdot \varepsilon} \quad (7.29)$$

Here, s represents the *disturbance strength*. It can be changed in the configuration dialog. The random number ε is drawn from the standard normal distribution $N(0, 1)$. Depending on the fit sequence type, p_0 is either

- the fitted value of the last fit in the sequence (F2),
- the parameter value of the best fit in the sequence (F3),
- the initial value of the last fit (F4).

7.12 Boosted fit

With `pwFitBoost`, a trust region and simulated annealing strategy in normal and logarithmic parameter space are combined in order to reduce local minima. This approach takes more time but is significantly superior for complicated optimization problems compared to a single optimizer method.

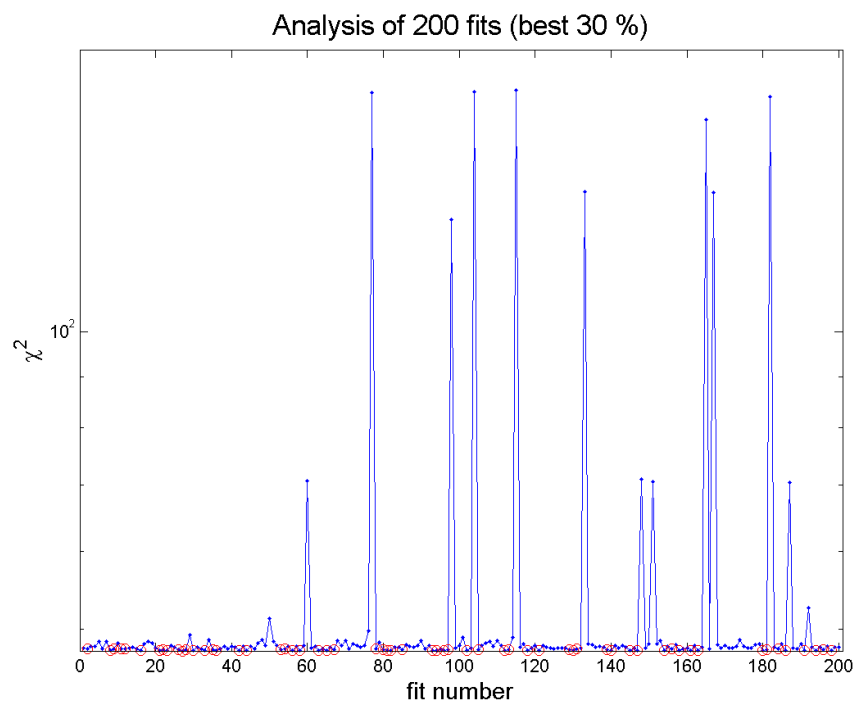
Chapter 8

Fit analysis

8.1 Fit sequence based analysis

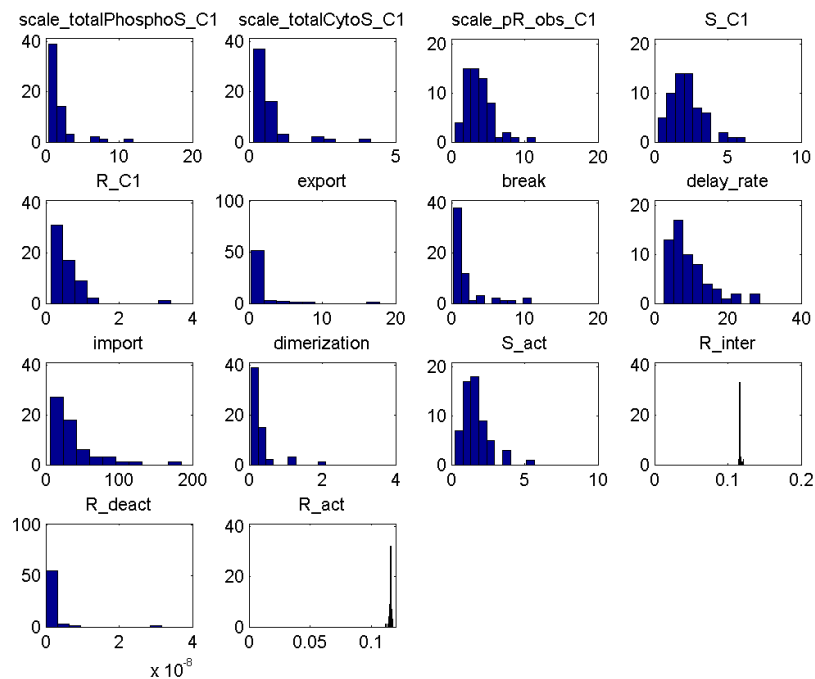
Based on a fit sequence, several analyses can be applied. We recommend to select a subset of fits with the lowest χ^2 values, in order to circumvent local minima in the parameter space. Histograms and boxplots depict the mean value and variance of the found parameter values. A correlation analysis between pairs of parameters indicates how strong the parameters depend on each other linearly. Detailed scatter plots for all significant correlations often reveals which unidentifiabilities may exist in the model structure. If only linear dependencies occur, then a principal component analysis would determine the total number of degrees of freedom of the system. However, the dependencies are often non-linear and other methods are required (Hengl et al., submitted). The two main components of the principal component analysis are shown in a biplot analysis. Hierarchical clustering of all parameter values based on their euclidian distance shows whether distinct local minima were found by the optimization routine, which should not be the case.

8.1.1 Best fit selection



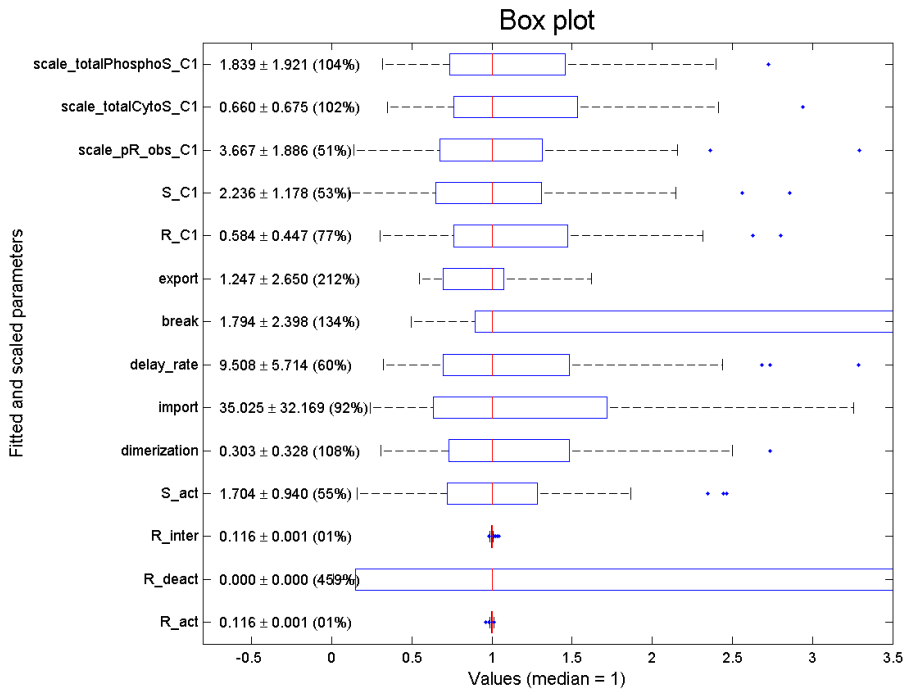
In order to neglect local minima fits only a fraction of fits with lowest χ^2 value are selected, here the best 30% of 200 fits circled in red.

8.1.2 Histograms



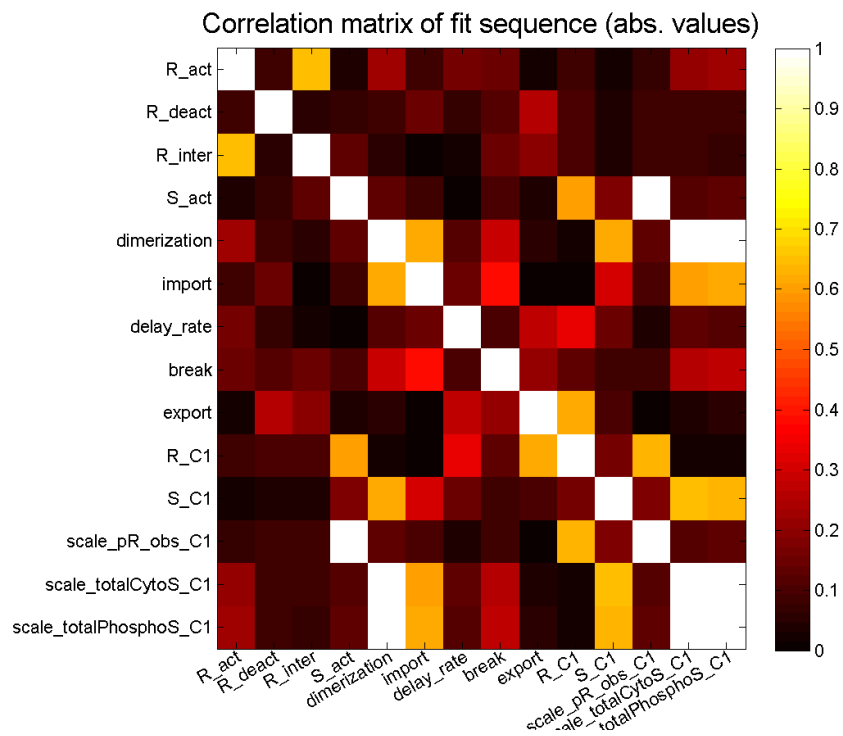
Histograms of the fitted parameter values give a first impression how unique each parameter can be determined. Parameters R_{inter} and R_{act} are always fitted to the same value, whereas the other parameters exhibit a broad distribution. This can be due to non-identifiabilities in the model structure or insufficient amount or characteristics of the measurements.

8.1.3 Boxplot



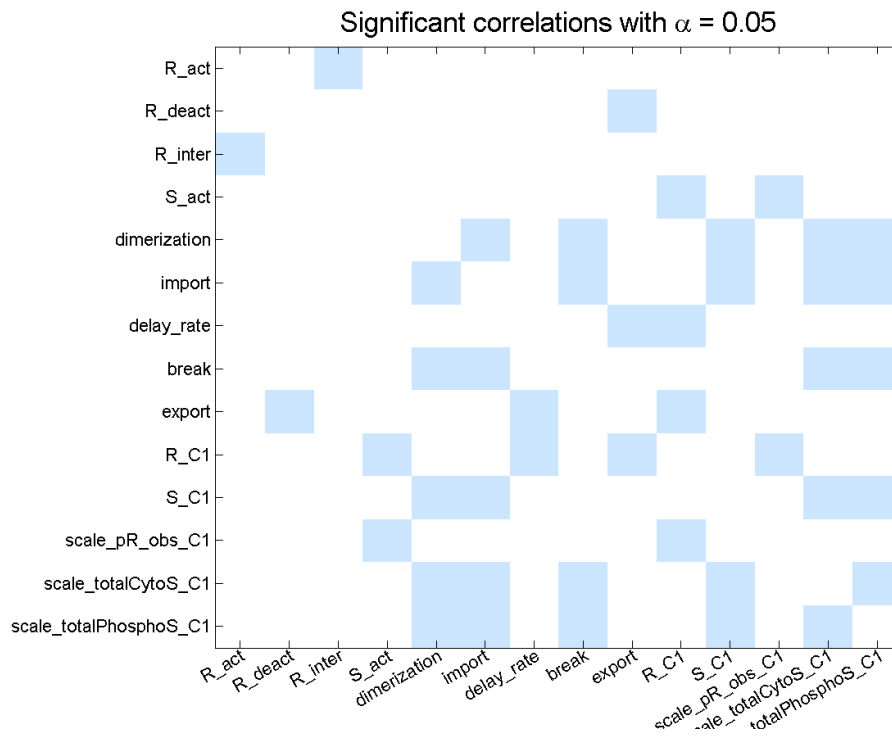
For the boxplot the distribution of each parameter is scaled to median 1. The labels show mean and standard deviation. The red line represents the median, the box comprises all values between the lower and upper quartile and the whiskers at the end of the dashed lines show the last data point inside of quartile * 1.5. Outliers beyond are drawn as single dots.

8.1.4 Correlation matrix



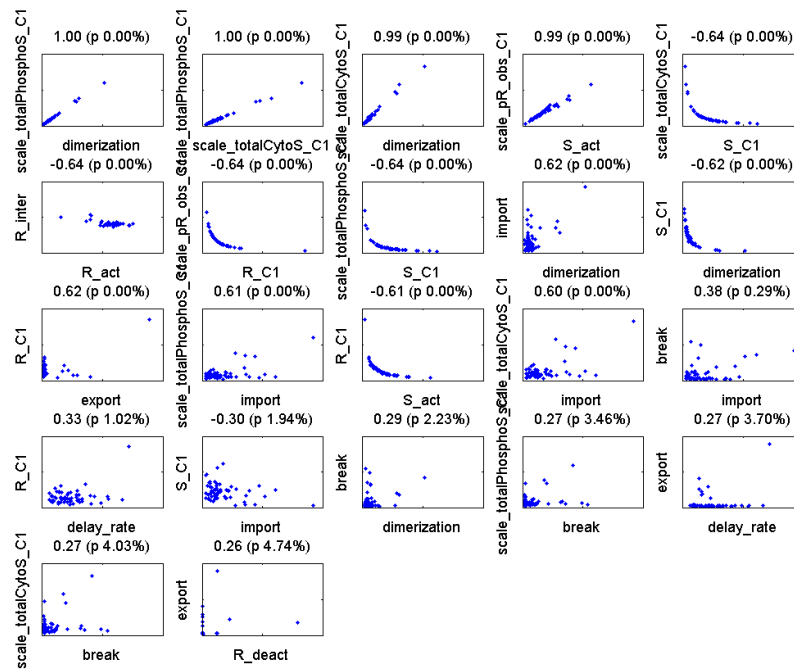
The correlation matrix shows the pairwise correlation of all parameters. The diagonal of the symmetric square matrix has the value 1, since the correlation of a parameter with itself is one. A covariance matrix would contain the variance at the diagonal.

8.1.5 Significant correlations



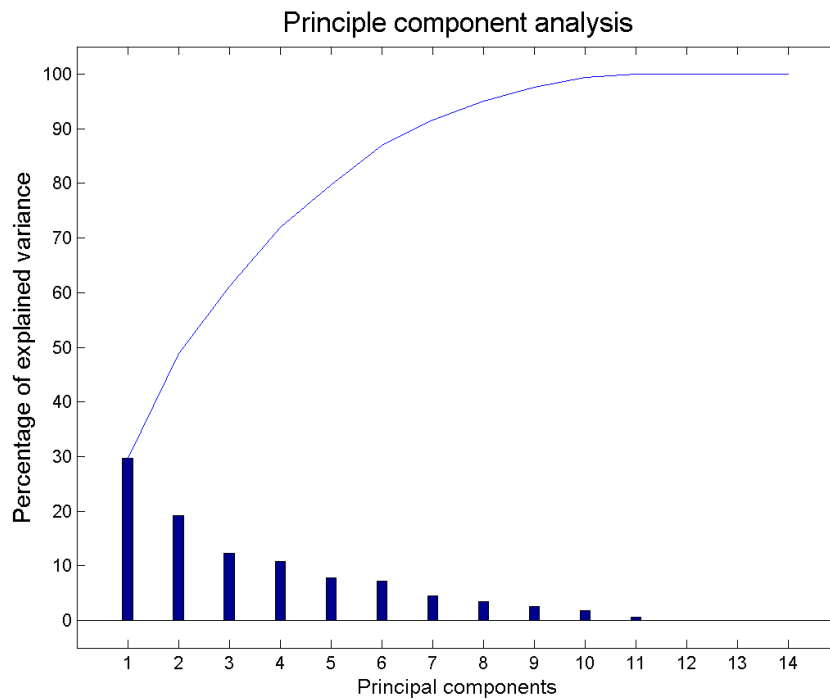
Some parameters may be significantly correlated which is depicted in the above figure.

8.1.6 Detailed significant correlations



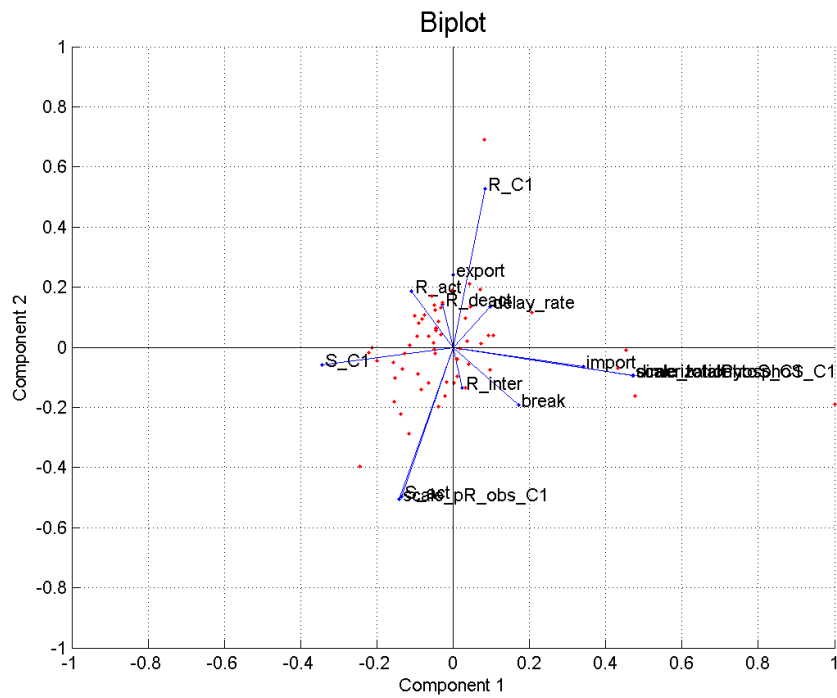
A detailed view on the significantly correlated parameters show that e.g. that only the fraction of 'scale_totalPhosphoS_C1' and 'dimerization' can be determined. Their single values are non-identifiable.

8.1.7 Principal Component Analysis (PCA)



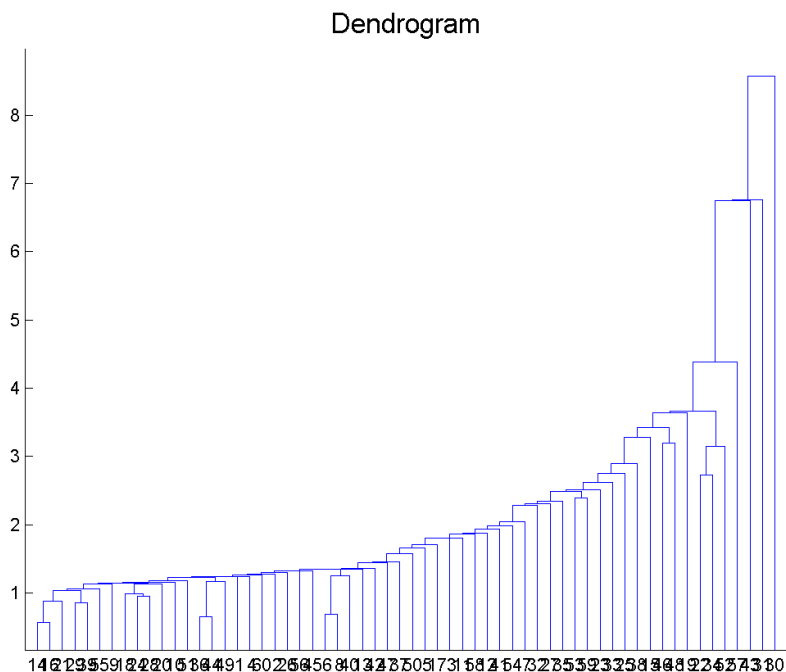
A principal component analysis tries to explain the variation of all parameters by a smaller set of linear combined parameters which have no direct interpretation. However, the number of required combinations to explain most of the variance, in this case 10 of the 14 original parameters, gives insight into the real complexity of the model.

8.1.8 Biplot



The biplot displays the first two components of the principal component analysis for each original parameter.

8.1.9 Hierarchical clustering



This hierarchical clustering is based on the Euclidian distance of the median to 1 scaled parameters. If two or more large branches occur instead of the shown large single tree, the optimization procedure found several distinct, equally good parameter settings to explain the data. An example application is given in [20].

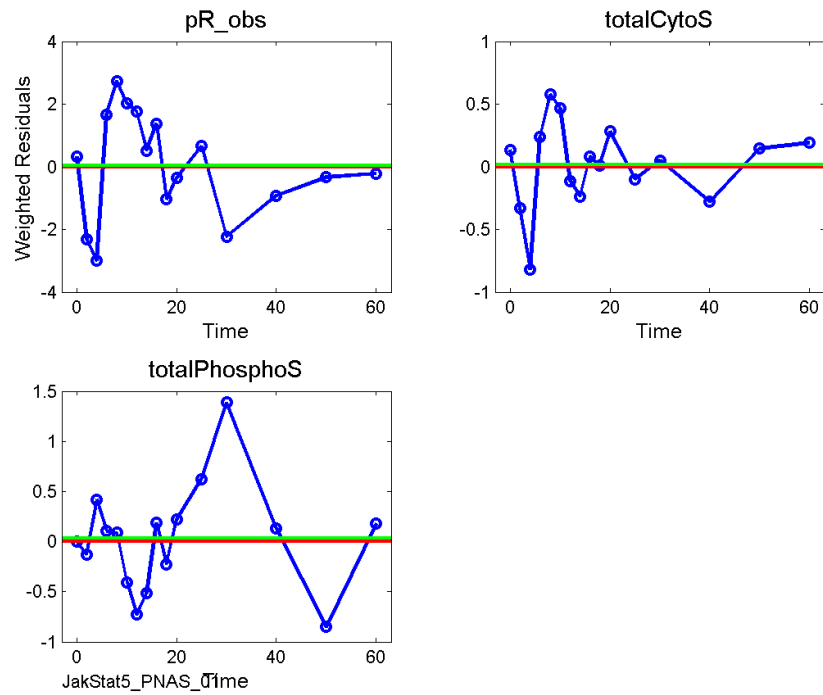
8.2 Derived parameters

The fit sequence analysis of section 8.1 showed several highly correlated parameter pairs. Some correlations may be reduced if only the fraction or product of the parameters is estimated and not their single values. This can be done within PottersWheel with the concept of derived parameters, introduced in 2.2.13 and based on the `m = pwAddP(m, rhs, ID)` paragraph in the model definition file. If, e.g., in the JakStat5 example, a derived parameter

```
m = pwAddP(m, 'scale_totalPhosphoS / dimerization', 'fraction1');
```

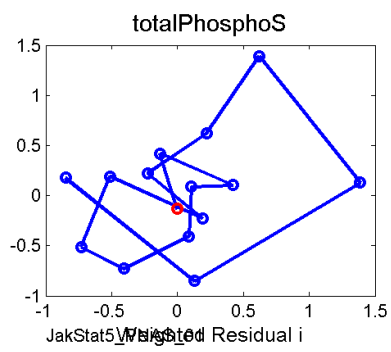
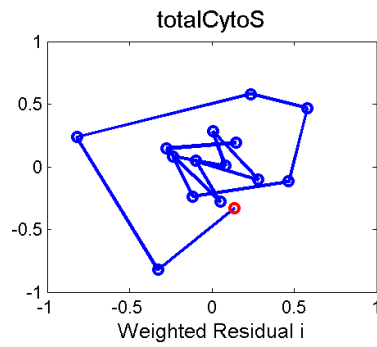
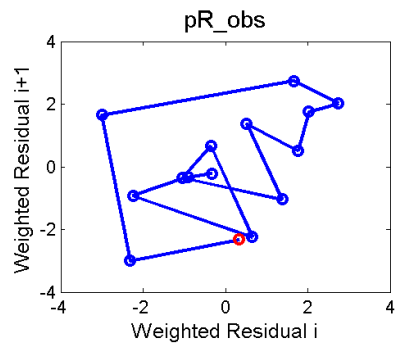
then a fit sequence yields much lower standard deviations for the estimated fraction of `scale_totalPhosphoS` and `dimerization` as compared to the initial fits.

8.3.1 Residuals over time

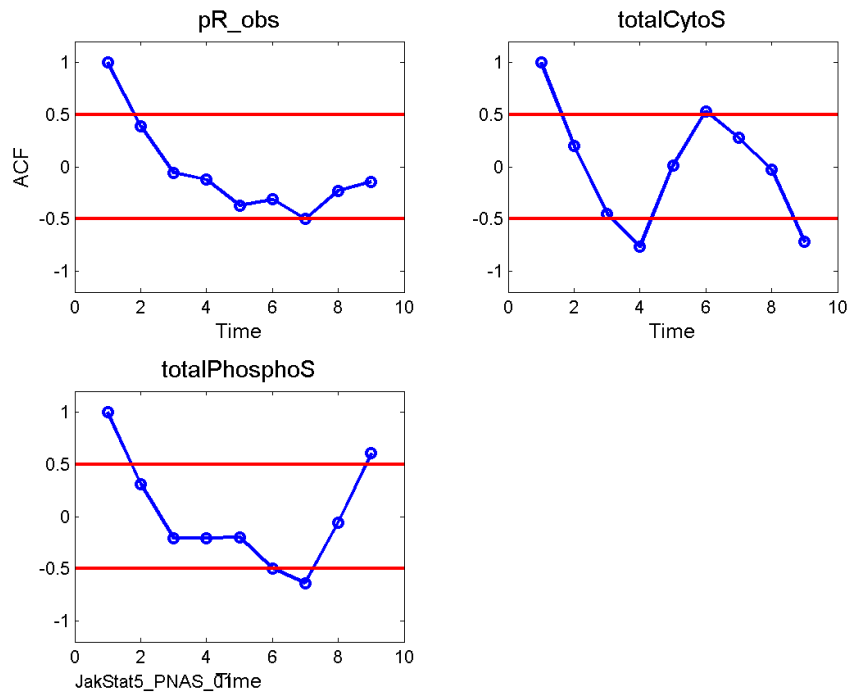


The average over all residuals (green line) should be around zero (red line). Besides, consecutive residuals should have different signs.

8.3.2 Residuals embedded

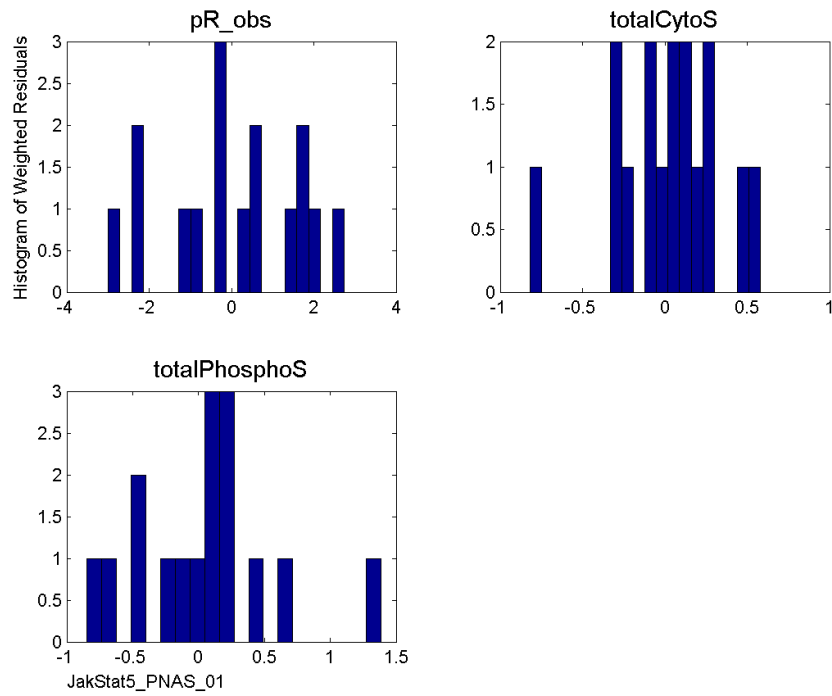


8.3.3 Autocorrelation of residuals



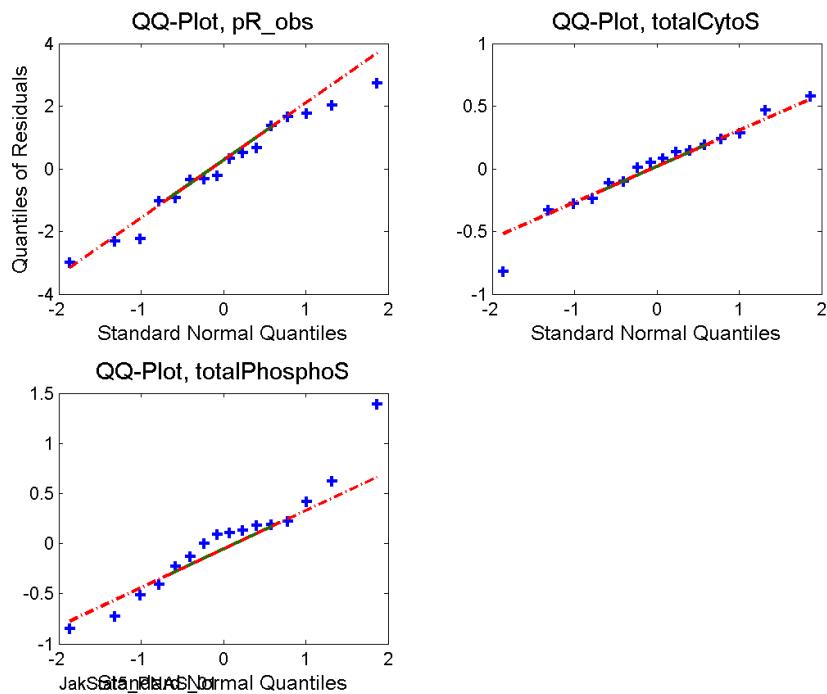
The autocorrelation function should be compatible with Gaussian white noise. The 95%-confidence interval is depicted by the red lines.

8.3.4 Histogram



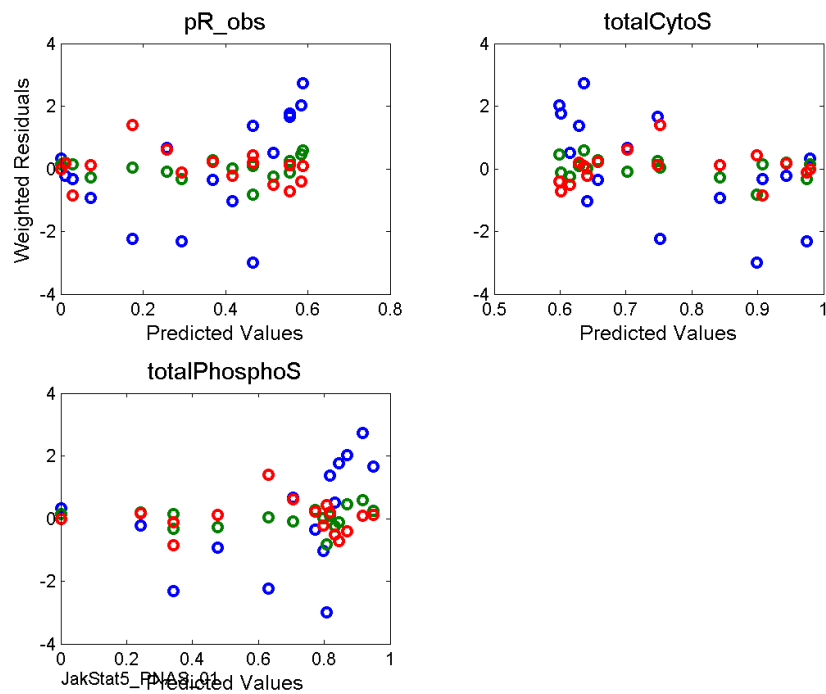
With only a few data points it is difficult to detect violations of a Gaussian distribution for the residuals within their histogram.

8.3.5 QQ-Plot



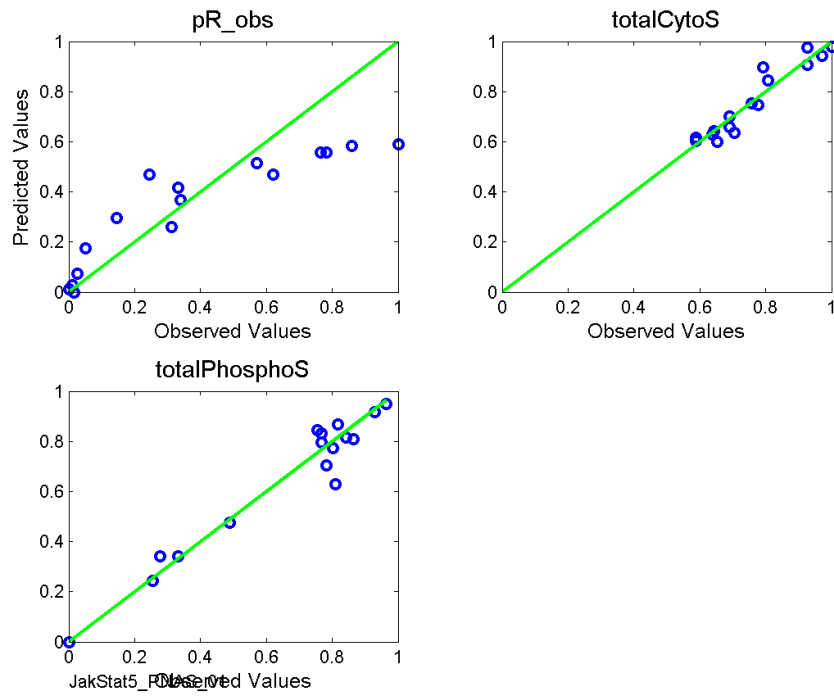
A QQ-Plot matches quantiles of a Gaussian distribution with the empirical quantiles of the residuals. A perfect Gaussian distribution would lay on the bisecting line. Here, the residuals of the fitted receptor data differ from a Gaussian distribution.

8.3.6 Residuals against predicted values



Often the residuals increase for increased data- or model-values.

8.3.7 Predicted against observed values



The upper left panel of figure 8.3.7 shows clearly that the model underestimates the maximum of the measured pEpoR concentration.

Chapter 9

Reporting: PDF Latex, MS Word, HTML

The results of each analysis can be appended as a section to a report object. The *append* button in the main user interface is enabled if a new section has been created but not yet added to a report.

The sections of a report can be deleted or reordered in the report designer. Besides, all figures corresponding to one section can be investigated as *.png files. Currently, PDF Latex, Microsoft Word and HTML reports are supported. However, PDF Latex is recommended. Please start the Adobe Acrobat Reader before creating the pdf file. Else, use Strg-C to get the control back to the Matlab command window.

The report designer allows also to add graphs and reaction lists of the *combined* models. The creation of sub-network graphs require use of the naming convention.

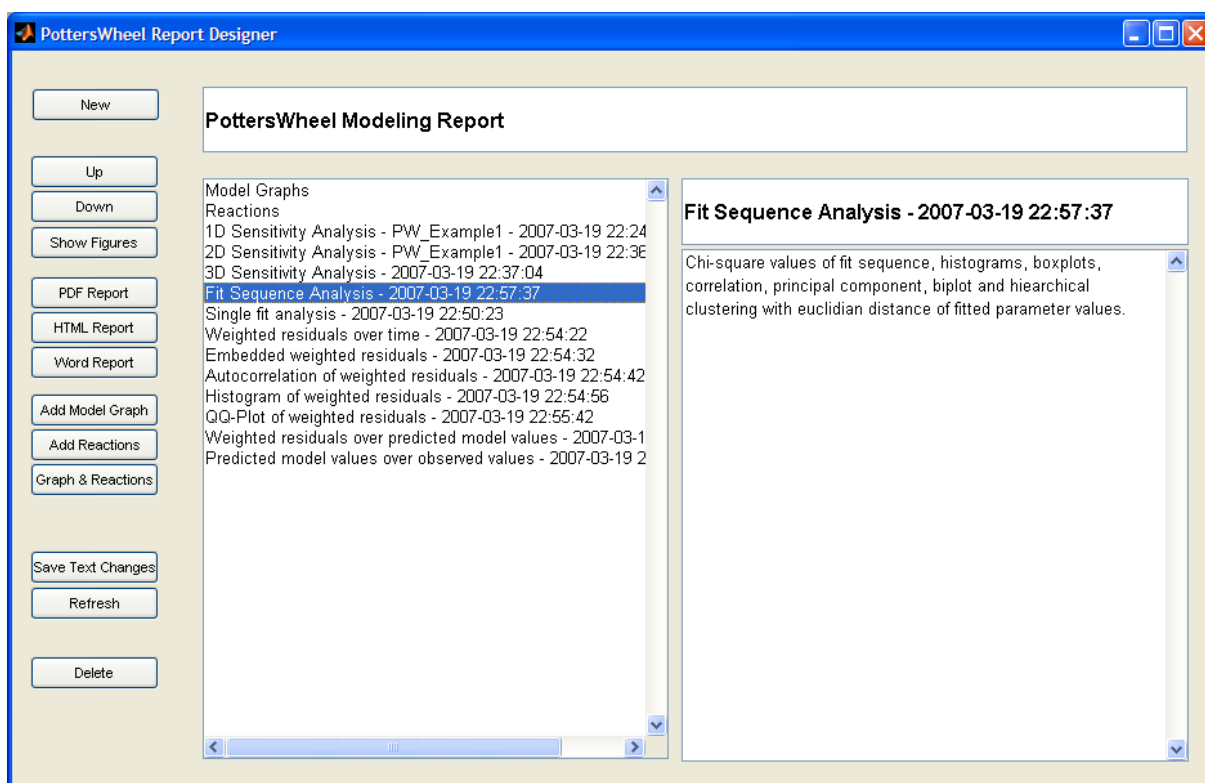


Figure 9.1: PottersWheel report designer

Chapter 10

Installation

10.1 Requirements

Minimum requirements:

- Matlab 7.1 SP 3

Optional:

- Optimization toolbox (recommended)
- R (recommended, free)
- SBML toolbox (recommended, free)
- Graphviz (recommended, free)
- MikTeX Latex (recommended, free)
- Statistics toolbox (would be fine)
- Splines toolbox (alternatively, splines can be calculated with R)
- Genetic algorithm toolbox (not important)

10.2 Installation

10.2.1 Windows

1. Install **Matlab** 7.1 or higher (www.mathworks.com). Probably, your Physics, Mathematics or Informatics department has a Matlab license. An academic license with optimization toolbox should cost about 800 - 1000 Euro for academic usage. Matlab 6.5 or 7.0 is not sufficient, since Matlab has been improved fundamentally towards the 7.1 series and PottersWheel makes strong use of the new functionality.

2. Install the **Graphviz** dot program from www.graphviz.org, which is required for graphical visualizations of models [13].
3. Install **MikTeX** 2.5 or better from www.miktex.org for latex pdf documentation. If you choose the basic MikTeX installer, select automatic download of required packages during the installation.
4. Install the statistics program **R** from www.r-project.org if you do not have the splines toolbox.
5. Unzip the **PottersWheel** toolbox from www.potterswheel.de, ideally not into the 'C:\Program Files\Matlab\toolbox' folder, but e.g. into 'C:\PottersWheelToolbox'.
6. Install the **SBML toolbox** from <http://sbml.org/software/sbmltoolbox> for importing SBML models. Follow the given installation instructions.
7. (Re-)Start Matlab and type `mex -setup`. Select 'lcc'.
8. Change the working directory of Matlab to the unzipped PottersWheelToolbox folder and type `pwInstall`. You can change the working directory e.g. with `cd 'C:\PottersWheelToolbox'`.
9. Check the installation by typing `pwCheckSystem`.
10. Type either `PottersWheel` or just `pw` to start PottersWheel, if the installation was successful. It is a good idea to change the current directory to an empty working folder, because PottersWheel will produce temporary files.

Troubleshooting

- If `pwCheckSystem` can not find one of the installed external programs Graphviz, R, or Latex, open a Windows command window via 'Start - Run - cmd' and type 'dot -V', 'R', or 'latex -version', respectively. If the program is not available, you must change the system PATH variable. Via 'Start - My Computer - right click - Properties - Advanced - Environment variables - System variables - PATH' you can access a list of program folders separated with ';'. Just append the missing program, e.g. append ';C:\Program Files\R\R-2.2.0\bin' for R, or ';C:\Program Files\ATT\Graphviz\bin' for Graphviz. Close and reopen the Windows command window and check whether Windows finds the programs now. If not R but the command `Rterm` is available, go to the R installation folder and copy `Rterm.exe` to `R.exe`. Restart Matlab after a change to the PATH variable.
- If dot has the correct version in the Windows command window but not within Matlab, type `!which dot`. If a Matlab folder like 'c:/MATLAB7/bin/win32/dot' appears, rename this dot.exe file for example to dotOld.exe in the windows file explorer.

10.2.2 Linux, Macintosh and Unix

For some Unix based distributions and Matlab versions it is not trivial to configure the mex compiler. If you are lucky, the included test file `pwMexTest.c` can be compiled without problems. Please change the current Matlab working folder to the PottersWheel toolbox folder and try within Matlab

```
mex pwMexTest.c
```

If the compilation is successful, type `'pwMexTest'`. A short message should appear. If this doesn't work, you have to adapt your mex configuration file. Type `'mex -setup'` and try to make a good choice for the configuration.

On my test system, an Ubuntu Linux PC with `g++-3.3` and Matlab 7.1 SP3, the above steps were not successful. Only a manually changed options file could resolve the problem. You find it under www.fdm.uni-freiburg.de/~maiwald/PottersWheel/Doc/mexopts.sh. Let me know, if you encounter any problems on your system!

Please follow the instructions for Windows users except for configuration of the mex compiler. MikTeX is only available for Windows, but usually latex and pdflatex are already available on a Unix based system. Mac binaries for Graphviz are available from www.ryandesign.com/graphviz/.

Do not forget to switch within Matlab to the PottersWheel toolbox folder and to type `pwInstall`.

10.3 Updating

Whenever you update the PottersWheel toolbox, just replace the old files by the new ones in the PottersWheel toolbox folder and restart Matlab. Old repositories can be used as long as the PottersWheel version number is the same (currently 1.3). Type `'pwUpdateRepository'` in the Matlab command window after loading an old repository.

Chapter 11

Configuration settings

All settings to control PottersWheel are saved in one Matlab struct which can be accessed via

```
config = pwGetConfig();  
pwSetConfig(config);
```

New versions of PottersWheel will often include extended configuration settings and hence an enlarged configuration struct. However, fields may only be removed in major releases. The configuration dialog provides a robust way to change configuration settings. In addition, advanced users may change the settings programmatically.

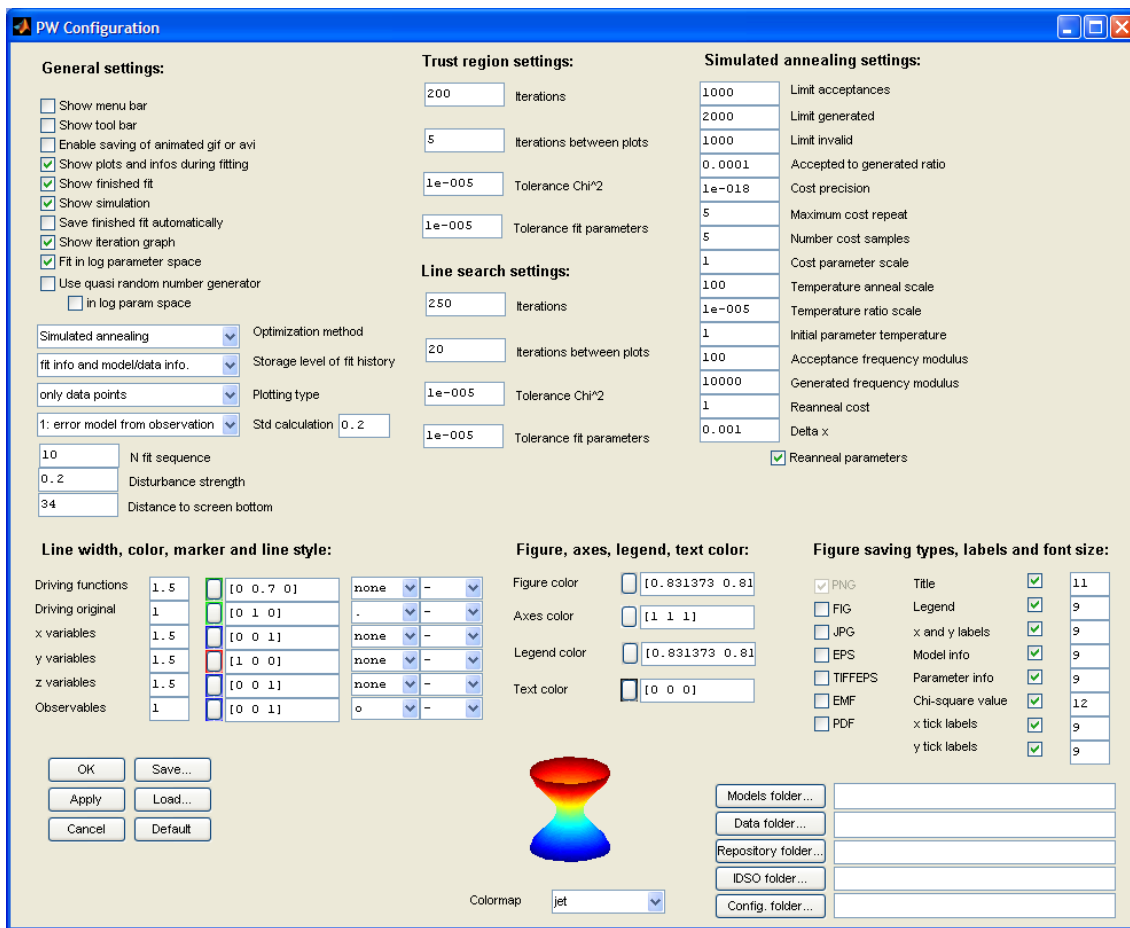


Figure 11.1: Configuration dialog

Chapter 12

Reaction kinetics

12.1 General considerations

12.1.1 Change of units

Often, parameter values are given in different units as required for the model definition. Or, the x variables represent scaled entities and the fitted parameter values have to be translated into a common scaling like nM. For linear systems like mono-molecular reactions with mass action kinetics, these unit changes have no effect on the parameter values, whereas for bi-molecular and higher reactions, the parameter have to be translated as shown in the following:

Linear systems

Consider the reversible reaction of A to B with mass action kinetics,



$$\dot{A} = -k_1 A + k_2 B. \quad (12.2)$$

$$(12.3)$$

For a transformation from 10^4 mol to 1 mol one could write $8.2 \cdot 10^4 \text{ mol} = 8.2 \cdot 10^4 \cdot \text{mol}$, i.e. the dynamic variables increase from the old value A to \tilde{A} by a factor of 10^4 :

$$\tilde{A} = A \cdot 10^4, \quad (12.4)$$

$$\tilde{B} = B \cdot 10^4. \quad (12.5)$$

$$(12.6)$$

Replacing A by $\tilde{A} 10^{-4}$ changes the differential equations to

$$\dot{\tilde{A}} 10^{-4} = -k_1 \tilde{A} 10^{-4} + k_2 \tilde{B} 10^{-4} \quad (12.7)$$

$$\Rightarrow \dot{\tilde{A}} = -k_1 \tilde{A} + k_2 \tilde{B}, \quad (12.8)$$

$$(12.9)$$

hence the (linear) system is as expected invariant under scaling transformations.

Nonlinear systems

In the case of a bi-molecular reaction with mass action kinetics,



$$\dot{A} = -kAB, \quad (12.11)$$

$$(12.12)$$

the above transformation leads to

$$\dot{\tilde{A}} 10^{-4} = -k \tilde{A} 10^{-4} \tilde{B} 10^{-4}, \quad (12.13)$$

$$\Rightarrow \dot{\tilde{A}} = -k 10^{-4} \tilde{A} \tilde{B}, \quad (12.14)$$

$$\Rightarrow \dot{\tilde{A}} = -k' \tilde{A} \tilde{B}. \quad (12.15)$$

$$(12.16)$$

Here, the scaled system comprises a parameter k' which is related to the original parameter via $k = k' 10^4$.

Regarding the units of the parameter, one can summarize the result as:

If the unit of the system variables is changed from 10^4 mol to 1 mol, then a parameter k with $[k] = 1/(mol\ s/l)$ is changed to $k' = k 10^{-4}$. Example:

$$3.9 \frac{1}{10^4 mol\ s} = 3.9 10^{-4} \frac{1}{mol\ s} \quad (12.17)$$

12.2 List of reaction kinetics

This chapter lists a subset of reaction kinetics as given in [11].

12.2.1 Constant flux

Reaction rate does not depend on any species and has a constant value:

$$v = V \quad (12.18)$$

12.2.2 Mass action kinetics, irreversible

The reaction rate is proportional to the product of reactants r_i :

$$v = k \cdot \prod_i r_i \quad (12.19)$$

12.2.3 Mass action kinetics, reversible

The reaction rate is proportional to the product of reactants r_i reduced by the product of products p_i :

$$v = k \cdot \prod_i r_i - \prod_i p_i \quad (12.20)$$

12.2.4 Henri-Michaelis-Menten

One reactant r is converted into a product with a limiting reaction rate V and the Michaelis constant K_m , where $v = V/2$.

$$v = \frac{Vr}{K_m + r} \quad (12.21)$$

12.2.5 Reversible Michaelis-Menten

The reversible Michaelis-Menten kinetic comprises limiting forward and reverse reactions rates, V_f and V_r respectively. K_{mR} is the reactant concentration, such that $v = V_f/2$ for a vanishing product concentration $p = 0$. K_{mP} on the other hand, is the product concentration, such that $v = -V_r/2$ for $r = 0$.

$$v = \frac{V_f \frac{r}{K_{mR}} - V_r \frac{p}{K_{mP}}}{1 + \frac{r}{K_{mR}} + \frac{p}{K_{mP}}} \quad (12.22)$$

12.2.6 Reversible Michaelis-Menten with Haldane adjustment

In contrast to the unadjusted reversible Michaelis-Menten kinetic the equilibrium constant K_{eq} enters the kinetic rate.

$$v = \frac{V_f \left(r - \frac{p}{K_{eq}} \right)}{r + K_{mR} \left(1 + \frac{p}{K_{mP}} \right)} \quad (12.23)$$

12.2.7 Hill cooperative kinetics

Sigmoidal enzyme kinetics where the reactant binds cooperatively to the enzyme. A Hill coefficient h larger than 1 corresponds to positive cooperativity, where the binding of one reactant facilitates binding of another reactant, in contrast to negative cooperativity with $h < 1$, where binding of further reactants is made more difficult. Like in Michaelis-Menten kinetics, $R_{0.5}$ is the reactant concentration for $v = V/2$.

$$v = \frac{Vr^h}{R_{0.5}^h + r^h} \quad (12.24)$$

12.2.8 Reversible Hill kinetics

As suggested in [14].

$$v = \frac{\left(V_f \frac{r}{R_{0.5}}\right) \left(1 - \frac{p}{pK_{eq}}\right) \left(\frac{r}{R_{0.5}} + \frac{p}{P_{0.5}}\right)^{h-1}}{1 + \left(\frac{r}{R_{0.5}} + \frac{p}{P_{0.5}}\right)^h} \quad (12.25)$$

12.2.9 Irreversible reaction with competitive inhibition

An inhibitor I competes with the reactants for the binding site with K_i being the inhibition constant. For vanishing inhibitor, K_m is again the reactant concentration such that $v = V/2$.

$$v = \frac{V \frac{r}{K_m}}{1 + \frac{r}{K_m} + \frac{I}{K_i}} \quad (12.26)$$

Chapter 13

Shortcuts and functions

13.1 Shortcuts

Several shortcuts for the most important functions are available if the PottersWheel main window or the Equalizer have the focus:

- a Arrange
- c Configuration
- d Draw
- e Equalizer
- f Fit
- g Graph
- h Show shortcuts
- i Input designer
- j Fit only start values x_0
- k Fit only dynamic parameters k
- l Fit only scaling parameters s
- m Set randomized parameters
- n Set fixed parameters
- o OK
- p PottersWheel
- q Quit current dialog (Cancel)
- r Reset parameter values
- s Simulate
- u Change plotting state for driving input u
- x Change plotting state for variables x
- y Change plotting state for observables y
- z Change plotting state for derived variables z

A	Analysis of fits
B	Combine
D	Add data
E	Edit model
F	Fit settings
G	Show graph as PNG
H	Online help
I	IDSO
L	Load repository
M	Add model
N	New model
O	Show ODE
P	Save figures
Q	Close all figures
R	Reload model
S	Save repository
V	Duplicate
W	Editor
X	Export to pdf latex and open pdf
1-0	PW: De-/Select model. Eq: Select slider for arrows-use
+−	Shift parameter list up/down
.	Show fit info in command window
Del	Delete selected model
Back	Delete selected model
Ret	Update PottersWheel model lists
(Fit sequence 2
)	Fit sequence 3
<,>	Change position of selected model
;	Disturb

Short commands for the Matlab command line:

pw	Open PottersWheel
pe	Open Equalizer
pc	Open configuration dialog

13.2 Alphabetical function list

13.2.1 PottersWheel

Opens the PottersWheel main graphical user interface.

13.2.2 pc

Opens the configuration dialog.

13.2.3 pe

Opens the PottersWheel Equalizer.

13.2.4 pw

Opens the PottersWheel main graphical user interface.

13.2.5 pwAddC

Adds a compartment to a model. For use within a model definition file. Please compare with section [2](#).

13.2.6 pwAddData

```
pwAddData(filename);
```

Adds the data set saved in *filename* to all selected models. If the columns of the data file do not correspond to the names of the observation function, a column mapping dialog opens. If no file argument is specified, a file select dialog is used.

If the data file is not in the current working directory, the current data directory is used.

13.2.7 pwAddDynamicalParameters

```
pwAddDynamicalParameters(filename);
```

Adds dynamical parameters k and their limits for fitting to the currently selected couples. The file has to be a Matlab function returning k , $kMin$ and $kMax$.

```
function [k, kMin, kMax] = myValues()
```

```
k.StatProd    = 0.4;  
kMin.StatProd = 0;  
kMax.StatProd = 100;
```

```
k.StatDeg     = 0.9;  
kMin.StatDeg  = 0;  
kMax.StatDeg  = 100;
```

13.2.8 pwAddInstructions

```
pwAddInstructions(filename);
```

Add an experimental setting – the instructions – to the currently selected models. The instructions file is constructed as a Matlab function and contains for each stimulation a list of time points and driving functions for all required driving inputs. Additional driving

inputs are ignored.

```
function instructions = getInstructions()

% pwGetDrivingFunction(uType, uTimes, uValues)

instructions = [];

instructions(end+1).t          = [0:10 15:5:60];
instructions(end).stimuli.TGFb = pwGetDrivingFunction('steps', [-100 0], [0 1]);

instructions(end+1).t          = [0:10 15:5:40];
instructions(end).stimuli.TGFb = pwGetDrivingFunction('steps', [-100 0 10], [0 1 0]);

instructions(end+1).t          = [0 1 2 3 4 5 6 7 8 9 10 12 14 16 18 20 40 50 60 70 90];
instructions(end).stimuli.TGFb = pwGetDrivingFunction('steps', [-100 0 20], [0 1 0.5]);
```

13.2.9 pwAddK

Adds a dynamical parameter to a model. For use within a model definition file. Please compare with section 2.

13.2.10 pwAddModel

`pwAddModel(filenamees)`; Adds one or more models to the repository list. If the model is not in the current working directory, the current model directory is used.

13.2.11 pwAddObservations

`pwAddObservations(filename)`;

The observation file is constructed as a Matlab function and contains a list of observation functions defined similarly as in a model definition file. Example:

```
function y = getObservations()

y          = pwGetY('Stat');
y(end+1) = pwGetY('pStat + 2 * pStat_pStat', 'total_pStat');
y(end+1) = pwGetY('Stat + pStat + 2 * pStat_pStat', 'total_Stat');
```

13.2.12 pwAddP

Adds a derived parameter to a model. For use within a model definition file. Please compare with section 2.

13.2.13 pwAddR

Adds a reaction to a model. For use within a model definition file. Please compare with section 2.

13.2.14 pwAddRule

Adds a rule to a model. For use within a model definition file. Please compare with section 2.

13.2.15 pwAddS

Adds a scaling parameter to a model. For use within a model definition file. `m = pwAddS(m, ID, value, type, minValue, maxValue, unit, name, description);` Please compare with section 2.

13.2.16 pwAddStartValues

```
pwAddStartValues(filename);
```

Adds start values saved in filename to the currently selected models. The file has to be a Matlab function returning `x0`, `x0Min` and `x0Max`. Example:

```
function [x0, x0Min, x0Max] = myValues()

x0.Stat      = 10;
x0Min.Stat   = 0;
x0Max.Stat   = 1000;

x0.pStat     = 0;
x0Min.pStat  = 0;
x0Max.pStat  = 100;
```

13.2.17 pwAddU

Adds an external driving input function to a model. For use within a model definition file. Please compare with section 2.

13.2.18 pwAddX

Adds a dynamical variable to a model. For use within a model definition file. Please compare with section 2.

13.2.19 pwAddY

Adds an observation function to a model. For use within a model definition file. Please compare with section 2.

13.2.20 pwAddZ

Adds a derived variable to a model. For use within a model definition file. Please compare with section 2.

13.2.21 pwAnalysisOfResiduals

`pwAnalysisOfResiduals(type)`

The residuals, i.e. the differences between model trajectory and data points, are analyzed. Depending on `type`, the following analyses are available:

- 1 Residuals against time
- 2 Residuals embedded
- 3 Autocorrelation of the residuals
- 4 Histogram
- 5 QQ-Plot
- 6 Residuals against predicted values
- 7 Predicted against observed values

13.2.22 pwArrange

Arranges all figures in order to draw trajectories of the currently *combined* couples.

13.2.23 pwAutoResetQRNG

`pwAutoResetQRNG(ok)` ;

`ok = true`: Before each fit sequence where QRNG is needed, the QRNG index is set to 1.

`ok = false`: The current QRNG index is unchanged.

Use `pwGetQRNGIndex` and `pwSetQRNGIndex` to get and set the QRNG index.

13.2.24 pwCheckSystem

Checks the installation of PottersWheel.

13.2.25 pwClear

Deletes all loaded models and data sets and resets the configuration settings.

13.2.26 pwCloseEqualizer

`pwCloseEqualizer` ;

Closes the equalizer GUI.

13.2.27 pwCloseInputDesigner

```
pwCloseInputDesigner;
```

Closes the input designer.

13.2.28 pwCloseMainGUI

```
pwCloseMainGUI;
```

Closes the main graphical user interface. No data is lost. The commands `pw` or `PottersWheel` reopen the GUI.

13.2.29 pwCombine

Combines all selected model-data-couples into the combination list.

13.2.30 pwConfiguration

Open the configuration dialog.

13.2.31 pwCreateModelGUI

Opens the model creation wizard.

13.2.32 pwDataAndFits

```
pwDataAndFits;
```

Shows the data and fits dialog.

13.2.33 pwDelete

Deletes the currently selected models.

13.2.34 pwDisturb

Disturbs all parameters with

$$p_{new} = p_{old} \cdot 10^{s \cdot \varepsilon}. \quad (13.1)$$

The value of s represents the *disturbance strength*. It can be changed in the configuration dialog. The random number ε is drawn from the standard normal distribution $N(0, 1)$. If you want to disturb only a subset of parameters, use `pwDisturbedParameters` to select which parameters should not be disturbed.

The disturbance strength can also be specified as an argument, `pwDisturb(s)`.

13.2.35 pwDraw

Refreshes and draws all combined couples.

13.2.36 pwDuplicate

Appends a complete copy of the selected couples to the repository list.

13.2.37 pwEdit

Opens the selected models in the Matlab editor.

13.2.38 pwEqualizer

Opens the Equalizer.

13.2.39 pwF1

Applies a single fit. Please compare `pwFit`.

13.2.40 pwF2

Fits the currently combined models N times. N can be changed in the configuration dialog. Each fit starts with the disturbed initial parameter values of the first fit. N and the disturbance strength can also be specified as optional arguments:

```
pwF2(N, disturbanceStrength).
```

13.2.41 pwF3

Fits the currently combined models N times. N can be changed in the configuration dialog. Each fit starts with the disturbed initial parameter values of the best fit in the current fit sequence. N and the disturbance strength can also be specified as optional arguments:

```
pwF3(N, disturbanceStrength).
```

13.2.42 pwF4

Fits the currently combined models N times. N can be changed in the configuration dialog. Each fit starts with the disturbed initial parameter values of the last fit. N and the disturbance strength can also be specified as optional arguments:

```
pwF4(N, disturbanceStrength).
```

13.2.43 pwF5

```
pwF5(parameterMatrix);
```

Applies n fits with $n = \text{size}(\text{parameterMatrix}, 1)$. Fit i start from $\text{parameterMatrix}(i,:)$. The number of columns of p must correspond to the currently fitted parameters. Selects the best fit of the sequence.

13.2.44 pwFit

Fits the currently combined models to their corresponding data sets. The used optimizer can be selected in the configuration dialog. Best experiences are made with the trust region approach which requires the Matlab optimization toolbox and the simulated annealing algorithm.

13.2.45 pwFitBoost

With `pwFitBoost`, a trust region and simulated annealing strategy in normal and logarithmic parameter space is combined in order to reduce local minima. This approach takes more time but is significantly superior for complicated optimization problems compared to a single optimizer method.

13.2.46 pwFitDynamicParameters

Fits only the dynamical parameters k .

13.2.47 pwFitHistory

Shows all applied fits in the current session and allows selection of fits for likelihood ratio tests and standard fit analysis.

13.2.48 pwFitHistoryDeleteFits

```
pwFitHistoryDeleteFits(percentage, fitGroup);
```

Deletes worst percentage of all fits of fit group `fitGroup`. `fitGroup = 1..number of fit groups`, `percentage = 0..100`

If no fit group is given, the procedure is applied to all fit groups.

13.2.49 pwFitHistoryGetFitGroups

```
[fitGroups, fitGroupsIDs, fitGroupsNFits, fitGroupsChisq, fitGroupsNParams,  
fitGroupsNData, fitGroupsPositionInHistory] = pwFitHistoryGetFitGroups();
```

fitGroups Cell array, where each cell contains all fits belonging to the same fit group, i.e. fits with the same fitted parameters.

fitGroupsIDs Cell array of the unique ID representing the fitted parameters.

fitGroupsNFits Number of fits per group.

fitGroupsChisq Cell array of the chisq values.

fitGroupsNPars Number of the fitted parameters per group.

fitGroupsNData Number of the fitted data points per group.

fitGroupsPositionInHistory Cell array of the position in the fit history.

If no output argument is given, the fit groups are visualized.

13.2.50 pwFitReport

Creates a latex pdf report about the last fit including all combined models, current figures and fitted parameter values.

13.2.51 pwFitScalingParameters

Fits only the scaling parameters s .

13.2.52 pwFitSequenceAnalysis

`pwFitSequenceAnalysis(percentage, minimumPValue, useDerivedParameters);`

Analysis of fits including:

1. Chi square values
2. p values
3. Histograms
4. Boxplots
5. Correlation matrix
6. Significant correlations
7. Details of significant correlations
8. Principal component analysis

9. Biplot

10. Dendrogram

percentage	Percentage of best fits used for analysis. Default 100
minimumPValue	Minimum accepted p value. Default 0
useDerivedParameters	true or false (default)

Please compare with `pwFitSequenceResults` and `pwFitSequencePlots`.

13.2.53 `pwFitSequencePlots`

```
pwFitSequencePlots(K, IDs, chisqValues, pValues, ...
                  percentage, minimumPValue, modelIDs, dataIDs);
```

Produces the same figures as `pwFitSequenceAnalysis` but based on saved `pwFitSequenceResults` rather than on the last applied fit sequence.

13.2.54 `pwFitSequenceResults`

```
[K, IDs, chisqValues, pValues, modelIDs, dataIDs] = ...
  pwFitSequenceResults(useDerivedParameters);}
```

Returns all parameters of the last fit or last fit sequence together with the parameter IDs, the χ^2 and p values, and the model and data set IDs.

useDerivedParameters	true or false (default)
K	Matrix of parameter values (fits x parameters)
IDs	IDs of the parameters as cell array
chisqValues	χ^2 values of all fits
pValues	p-values of all fits
modelIDs	IDs of all models used for the fits
dataIDs	IDs of all data sets used for the fits

13.2.55 `pwFitSettings`

Shows a GUI where the fit settings of all dynamical parameters k , start values x_0 and scaling parameters s of the highest selected model are shown and can be changed. Possible settings:

global	Fitting to the same value for all combined couples
local	Separate fit for each combined couple
fix	No change of the parameter value during fitting.

13.2.56 `pwFitStartValues`

Fits only the start values x_0 .

13.2.57 pwFixParameters

Shows a graphical user interface, where a subset of parameters can temporarily be fixed during fitting. Use the `pwFitSettings` function in general.

13.2.58 pwForcedCompilation

Compiles all selected models even if they are up to date.

13.2.59 pwGetConfig

```
config = pwGetConfig();
```

The returned configuration struct can be changed and applied with `pwSetConfig(config);`. It comprises a superset of the settings available in the configuration dialog. New versions of PottersWheel have usually an enlarged configuration struct with new fields. Only in major releases, old fields may be removed.

13.2.60 pwGetChisqAndN

```
[chisq, N] = pwGetChisqAndN(nCouple, nStimulus);
```

Returns the χ^2 value and number of data points of stimulus *nStimulus* of combined couple *nCouple*. If no stimulus is specified, the sum over all stimuli is calculated. If in addition no couple is given, the sum over all combined couples is calculated.

13.2.61 pwGetDrivingFunction

Help function used in external data files to specify under which experimental conditions the measured or simulated data was collected. Please compare section [6.1.2](#).

13.2.62 pwGetFitParameters

```
[IDs, values] = pwGetFitParameters();
```

Returns IDs and values of the currently fitted parameters as given e.g. in the equalizer.

13.2.63 pwGetFitParametersByID

```
values = pwGetFitParameterValuesByID(IDs);
```

Returns the value for all parameters given as IDs for the currently fitted parameters as given e.g. in the equalizer.

13.2.64 pwGetIDs

```
[kIDs, xIDs, sIDs] = pwGetIDs();
```

Returns the IDs of the highest selected model.

13.2.65 pwGetListOfParameters

Prints a list of parameter values to the Matlab command window with the same syntax as required in model definition files. This is useful if you only want to change the parameter section.

13.2.66 pwGetNumberOfCouples

Returns the number of currently available couples in the upper list box.

13.2.67 pwGetParameterValues

```
[k, x0, s] = pwGetParameterValues();
```

Returns the parameter values of the highest selected model of the current data set and fit.

13.2.68 pwGetPlottingData

This help function returns all data that can be shown if the 'Draw' button is pressed, i.e. of the currently combined couples. The first argument 'type' specifies which data matrix should be returned, e.g. the dynamic variables x , the derived variables z or rather the observed values y . *Fine* values are distinguished from *for fitting* values. The latter have the same sampling as the current fitted data set whereas the fine values have an equidistant and dense sampling between the smallest and largest sampling time point.

```
[matrix, IDs] = pwGetPlottingData(types, coupleNr, stimulusNr)
```

types	Cell array with one or more of the fields: 'tFine', 'xFine', 'yFine', 'zFine', 'uFine', 'tForFit', 'xModelForFit', 'yExtForFit', 'yExtStdForFit', 'residuals', or 'yModelForFit'
coupleNr	number of the <i>combined</i> couple
stimulusNr	number of the stimulus
matrix	Matrix with length(tFine) or length(tForFit) rows
IDs	Cell array with the column names of the matrix

Example:

```
[m, IDs] = pwGetPlottingData({'tFine', 'xFine'}, 2, 1);
```

Note that the *fine* variables can not be combined with the other variables, since the corresponding matrices have a different number of rows.

13.2.69 pwGetQRNGIndex

```
index = pwGetQRNGIndex();
```

Returns the currently used index of the quasi random number generator.

13.2.70 pwGraph

Shows graphs of all selected models in an extra window.

13.2.71 pwHelp

```
pwHelp pwFunction
```

Displays information about the function 'pwFunction'. Similar to the Matlab help function.

13.2.72 pwIDSO

Opens the IDSO-dialog, where

- the instruction including sampling and input functions,
- the dynamical parameters,
- the start values, and
- the observation functions

are displayed and can be changed.

13.2.73 pwInfo

Prints the current fitted parameter values to the command window in a table manner.

13.2.74 pwInfo2

Prints the current fitted parameter values to the command window as a list.

13.2.75 pwInputDesignerGUI

Opens the input designer, where the shape of the current driving input can be changed in real time.

13.2.76 pwInstall

Installs PottersWheel. The Matlab path is set and the mex compiler is tested.

13.2.77 pwLoadConfigFromMFunction

Loads a configuration file from hard disk.

13.2.78 pwLoadRepository

Loads a repository from hard disk.

13.2.79 pwModelInfo

```
pwModelInfo(filename, html);
```

With no argument: Displays a list of all models in the repository including number of reactions, parameters, and variables. If filename is given, the information is also saved as a text file. With `html = true`, a html page is saved similar to the one from the PottersWheel web site.

13.2.80 pwODEsToReactions

```
reactions = pwODEsToReactions(odes, xIDs, kIDs, uIDs);
```

Tries to re-create the set of reactions given a set of differential equations. Currently no brackets are allowed within the ODE.

13.2.81 pwOpenODE

```
pwOpenODE(mode);
```

With `mode = 'PottersWheel'` (default), the ODE C files of the currently selected models are opened in the editor. Else, the ODE C files of the currently combined models are opened.

13.2.82 pwParseAndShow

```
pwParseAndShow
```

Reloads the currently selected models without compilation and shows their graphs immediately. This is useful if many changes to a model should be compared interactively with the model graph.

13.2.83 pwPlot

```
pwPlot
```

Plots trajectories of the currently combined models.

13.2.84 pwPropertiesOfDataFile

```
pwPropertiesOfDataFile(filename);
```

Displays all experimental data saved in filename and provides a short summary of stimuli, driving functions and y columns. Supports several stimuli and can be appended to a report. Opens a save file dialog if no filename is given.

13.2.85 pwReload

Reloads all selected models. If the model structure didn't change, no fit is lost. Else, all fits of the model are deleted. Data sets are kept. If the observation function changed, the user has to map the old data sets to the new observables. Please compare section 6.3.

13.2.86 pwReportAppendGraphsAndReactions

Adds model graphs and reaction lists of all combined models to the current report:

```
pwReportAppendGraphsAndReactions(addGraphs, addReactions)
```

The arguments addGraphs and addReactions are either:

- true Section will be added
- false Section will not be added

13.2.87 pwReportAppendLastStep

Appends last analysis to the report.

13.2.88 pwReportClear

Deletes all current sections but does not create a new report or a new report folder.

13.2.89 pwReportGUI

Opens the PottersWheel Report Designer.

13.2.90 pwReportGenerate

```
pwReportGenerate(type);
```

Creates a Latex (default), MS Word or HTML report. Valid types are 'tex', 'word', or 'html'. Word and HTML contain only little text information.

13.2.91 pwReportNew

```
pwReportNew
```

Starts a new report and creates a report folder where the report figures will be saved and the report itself.

13.2.92 pwReset

Resets all parameter values as if the models were just combined.

13.2.93 pwSBML2PW

Converts an SBML model into the PottersWheel model definition format.

13.2.94 pwSaveConfigToMFunction

Opens a 'save file dialog' to save the current configuration settings to a Matlab m-file. This file can be edited and reloaded in future sessions, which is an alternative approach to manual changes in the configuration dialog.

13.2.95 pwSaveFigures

Saves the current figures following the figure saving settings in the configuration dialog. All figures are in the 'Plots' subfolder of the current working folder.

13.2.96 pwSavePlottingData

Function to save data of combined couples into a tabulator separated ascii file, which can be opened with a text editor or Excel. Please compare with `pwGetPlottingData`. If filename is empty, the user can select from a file dialog.

```
pwSavePlottingData(types, coupleNr, stimulusNr, filename)
```

13.2.97 pwSaveRepository

Saves the current working state to a repository file.

```
pwSaveRepository(filename, showDialog)
```

filename Name of used file. Default 'pwRepository'
showDialog If true, lets the user select a file. Default is false.

13.2.98 pwSaveSelectedModelsWithFittedParValues

Saves the highest selected model with the current fitted parameter values into a complete model definition file.

13.2.99 pwScale

```
pwScale;
```

Sets the mean of all observables of all selected couples to 1.

13.2.100 pwScaleAndMergeDataSets

```
pwScaleAndMergeDataSets(inputFiles, newFile);
```

Automatically detects one x column in all input files starting with 'xCol-' and combines all y columns after calculating an optimal scaling, so that the square distance between the spline approximation of a single y column for all files is minimized. Both arguments are optional.

Example:

```
pwScaleAndMergeDataSets('Exp1.xls', 'Exp2.xls', 'Exp3', 'mergedData.xls');
```

13.2.101 pwSelect

`pwSelect(indeces)` selects all models as given in the `indeces` argument in the PotersWheel repository list. For example `pwSelect([1 4])` would select model 1 and 4. `pwSelect('all')` selects all models. `pwSelect('last')` is also supported.

13.2.102 pwSelectCurrentData

Opens a graphical user interface and lets the user select which data set should be used for fitting. This defines the *couple*, consisting of one model and one data set. Note that one model may have several data sets, but only one, the *current* data set, will be used for fitting.

13.2.103 pwSensitivityAnalysis1D

One parameter is changed around an initial value. The model trajectories for all dynamic variables x , all observables y , and all derived variables z are plotted (lower row of figures) together with the calculated sensitivity, which is here the fraction of changed mean concentration to the initial mean concentration over the whole time span.

13.2.104 pwSensitivityAnalysis2D

In contrast to the 1D sensitivity analysis, a set of variables and of parameters can be selected to plot the model trajectories for different changes of the parameters.

13.2.105 pwSensitivityAnalysis3D

Applies a sensitivity analysis for all variables, observables and derived variables and for all parameters. For very large models, the user can select a subset of parameters in a graphical user interface. In contrast to the 1D and 2D methods, no detailed trajectories are plotted. Sensitivity of a state variable x_i on a parameter p_i is defined as

$$s_{ij}(t) = \frac{\partial x_i(t)/x_i}{\partial p_i/p_i} \quad (13.2)$$

for a certain time point t . Here, the sensitivity of the averaged concentration \bar{x}_i for the integration time is calculated. Numerical approximation:

$$s_{ij} = \frac{(\bar{x}_i(1.01 p_i^{org}) - \bar{x}_i(p_i^{org}))/\bar{x}_i(p_i^{org})}{0.01}. \quad (13.3)$$

13.2.106 pwSetConfig

```
pwSetConfig(config);
```

For advanced users: sets the configuration settings to the given config struct. Please compare pwGetConfig.

13.2.107 pwSetFitParameterValuesByID

```
pwSetFitParameterValuesByID(IDs, values);
```

Sets the value for all parameters given as IDs for the currently fitted parameters as given e.g. in the equalizer.

13.2.108 pwSetFitSettings

```
pwSetFitSettings(IDs, setting);
```

Sets the global, local and fix fit setting for all parameters given as IDs for all selected couples. Refers only to the currently selected data set. IDs is a cell array of IDs for k, x0 and s parameters. The second argument is either 'global', 'local', or 'fix'.

13.2.109 pwSetIntegrationStartTime

```
pwSetIntegrationStartTime(tStart);
```

Sets the integration start time of all selected couples. This is useful if the system should reach a basal level before time point 0.

13.2.110 pwSetLogFitting

```
pwSetLogFitting(trueOrFalse);
```

Activates or deactivates fitting in logarithmic parameter space.

13.2.111 pwSetOptimizer

```
pwSetOptimizer(optimizer);
```

Select the optimization routine. Possible arguments: 'LineSearch', 'TrustRegion', 'Annealing', 'Genetic'.

13.2.112 pwSetParameterValuesByID

```
pwSetParameterValuesByID(IDs, values, forSimulation);
```

Sets the value for all parameters given as IDs for all selected couples.

forSimulation = true or false (default)

If true: Uses the values for simulation.

If false: Refers to the current fit of the currently selected data set.

13.2.113 pwSetPlottingState

```
pwSetPlottingState(type, value);
```

Specifies whether separate figures should appear for the dynamic variables x , the observables y , the derived variables z , and the driving input u . Example:

```
pwSetPlottingState('x', true);
pwSetPlottingState('u', false);
```

13.2.114 pwSetQRNGIndex

```
pwSetQRNGIndex(index);
```

Sets the index of the quasi random number generator. Must be $i=1$.

13.2.115 pwShortcut

All shortcuts can also be used from the command line with `pwShortcut(letter)`. Depending on the letter argument, the same function is executed as if the letter was pressed

when the main graphical user interfaces have the focus. For example with `pwShortcut('R')`, all selected models are reloaded.

13.2.116 `pwShowFitting`

`pwShowFitting(trueOrFalse)`

- true PottersWheel shows plots and finished fits during fitting
- false No plots are shown which increases the fitting procedure

The settings could also be changed in the configuration dialog.

13.2.117 `pwShowGraph`

Shows graphs of all selected models in an extra window.

13.2.118 `pwShowGraphAsPNG`

Saves graphs of all selected models as PNG files. If only one model was selected, the PNG file is opened with the default system image viewer program.

13.2.119 `pwShowHelp`

Opens www.PottersWheel.de in the default internet browser.

13.2.120 `pwShowODE`

Prints all differential equations of the selected models into the Matlab command line.

13.2.121 `pwShowShortcuts`

Shows a list of available shortcuts.

13.2.122 `pwSim`

Simulates data for all selected models and sets the new data as the currently selected data set. Simulated data is essentially the model trajectory for the observables plus noise from the error model attached to the observables.

13.2.123 `pwSplitStimuli`

Some data sets may comprise several stimuli. This function creates copies of the highest selected couple, where the data set of each couple is one stimuli of the original data set.

13.2.124 `pwUndo`

Undoes the last step. This functionality is currently only available for a few functions.

13.2.125 pwUpdateRepository

The structure of saved repositories may change between different versions of PottersWheel. If you encounter any problems after loading an old repository, please apply this function.

Chapter 14

Acknowledgements

Thank you very much for your help, suggestions and findings!

Eva	Baloso-Canto	Instituto de Investigaciones Marinas de Vigo, Spain
Julie	Blumberg	University of Freiburg, Germany
Nikolai M.	Borisov	Thomas Jefferson University, Philadelphia
Stefan	Hengl	University of Freiburg, Germany
Clemens	Kreutz	University of Freiburg, Germany
Werner	Horbelt	Genedata, Basel, Switzerland
Stefan	Legewie	Humboldt University, Berlin
David	Liffmann	RWTH Aachen, Germany
Christian	Ludwig	Technical University, Munich
Peter	Nickel	German Cancer Research Center, Heidelberg
Tatsunori	Nishimura	University of Tsukuba, Japan
Martin	Peifer	University of Graz, Austria
Marcel	Schilling	German Cancer Research Center, Heidelberg
Thomas	Sumner	UCL, University of London
Jens	Timmer	University of Freiburg, Germany

(Alphabetical order.)

Bibliography

- [1] M. Bentele, I. Lavrik, M. Ulrich, S. Stosser, D.W. Heermann, H. Kalthoff, P.H. Kramer, and R. Eils. Mathematical modeling reveals threshold mechanism in cd95-induced apoptosis. *J.Cell Biol.*, 166(6):839–851, September 2004.
- [2] Michael L Blinov, James R Faeder, Byron Goldstein, and William S Hlavacek. Bionetgen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, Nov 2004.
- [3] Nikolay M Borisov, Nick I Markevich, Jan B Hoek, and Boris N Kholodenko. Signaling through receptors and scaffolds: independent interactions reduce combinatorial complexity. *Biophys J*, 89(2):951–966, Aug 2005.
- [4] Nikolay M Borisov, Nick I Markevich, Jan B Hoek, and Boris N Kholodenko. Trading the micro-world of combinatorial complexity for the macro-world of protein interaction domains. *Biosystems*, 83(2-3):152–166, 2006.
- [5] Richard H. Byrd, Robert B. Schnabel, and Gerald A. Shultz. Approximate solution of the trust region problem by minimization over two-dimensional subspaces. *Mathematical Programming*, 40(1 - 3):247–263, January 1988.
- [6] Thomas F. Coleman and Y. Li. An interior trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on Optimization*, 6:418–445, 1996.
- [7] Thomas F. Coleman and Arun Verma. A preconditioned conjugate gradient approach to linear equality constrained minimization. *Computational Optimization and Applications*, 20(1):61–72, October 2001.
- [8] A.R. Conn, N. I. M. Gould, and Ph. L. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28:545 – 572, 1991.
- [9] A.R. Conn, N. I. M. Gould, and Ph. L. Toint. A globally convergent augmented lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. *Mathematics of Computation*, 66:261 – 288, 1997.
- [10] Holger Conzelmann, Julio Saez-Rodriguez, Thomas Sauter, Boris N Kholodenko, and Ernst D Gilles. A domain-oriented approach to the reduction of combinatorial complexity in signal transduction networks. *BMC Bioinformatics*, 7:34, 2006.

- [11] A. Cornish-Bowden. *Fundamentals of Enzyme Kinetics (2nd edn.)*. Portland Press, London, 1995.
- [12] A. Finney and M. Hucka. Systems biology markup language: Level 2 and beyond. *Biochem Soc Trans*, 31(Pt 6):1472–1473, Dec 2003.
- [13] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A technique for drawing directed graphs. *Software Engineering*, 19(3):214–230, 1993.
- [14] J. H. Hofmeyr and A. Cornish-Bowden. The reversible hill equation: how to incorporate cooperative enzymes into metabolic models. *Comput Appl Biosci*, 13(4):377–385, Aug 1997.
- [15] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novre, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, J. Wang, and S. B. M. L. Forum. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, Mar 2003.
- [16] L. Ingber. Very fast simulated re-annealing. *Mathl. Comput. Modelling*, 12:967–973, 1989.
- [17] Sarah M Keating, Benjamin J Bornstein, Andrew Finney, and Michael Hucka. Sbml-toolbox: an sbml toolbox for matlab users. *Bioinformatics*, 22(10):1275–1277, May 2006.
- [18] B.N. Kholodenko, A. Kiyatkin, F.J. Bruggeman, E. Sontag, H.V. Westerhoff, and J.B. Hoek. Untangling the wires: a strategy to trace functional interactions in signaling and gene networks. *Proc.Natl.Acad.Sci.U.S.A*, 99(20):12841–12846, October 2002.
- [19] H. Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, March 2002.
- [20] Pontus Melke, Henrik Jansson, Evangelia Pardali, Peter ten Dijke, and Carsten Petersen. A rate equation approach to elucidate the kinetics and robustness of the tgf-b pathway. *Biophysical Journal*, 91:4368–4380, 2006.
- [21] J.J. More and D.C. Sorensen. Computing a trust region step. *SIAM Journal of Scientific and Statistical Computing*, 4:553–572, 1983.
- [22] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, 1999.

- [23] M. Schilling, T. Maiwald, S. Bohl, M. Kollmann, C. Kreutz, J. Timmer, and U. Klingmuller. Computational processing and error reduction strategies for standardized quantitative data in biological networks. *FEBS J.*, 272(24):6400–6411, December 2005.
- [24] M Schilling, T Maiwald, S. Bohl, M. Kollmann, C. Kreutz, J. Timmer, and U Klingmuller. Quantitative data generation for systems biology: the impact of randomisation, calibrators and normalisers. *IEEE Systems Biology*, 152(4):193–200, 2005.
- [25] Henning Schmidt and Mats Jirstrand. Systems biology toolbox for matlab: a computational platform for research in systems biology. *Bioinformatics*, 22(4):514–515, Feb 2006.
- [26] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.
- [27] I. Swameye, T. G. Muller, J. Timmer, O. Sandra, and U. Klingmuller. Identification of nucleocytoplasmic cycling as a remote sensor in cellular signaling by databased modeling. *Proc Natl Acad Sci U S A*, 100(3):1028–1033, Feb 2003.
- [28] H.V. Westerhoff and B.O. Palsson. The evolution of molecular biology into systems biology. *Nat.Biotechnol.*, 22(10):1249–1252, October 2004.