ADMIT

Tutorial – Functions and Syntax

S. Streif, A. Savchenko, P. Rumschinski, S. Borchers, R. Findeisen

Otto-von-Guericke University Magdeburg, Germany
Institute for Automation Engineering
Chair for Systems Theory and Automatic Control

Quick guide to ADMIT

Aims of this tutorial: Explain the work flow, main functions and constraint syntax

For further and more detailed information and extended functionality, please see examples and help texts using e.g.:

```
>> help ADMITconstraint
>> doc ADMITestimate
```

…

Please contact us if you need further help or guidance:

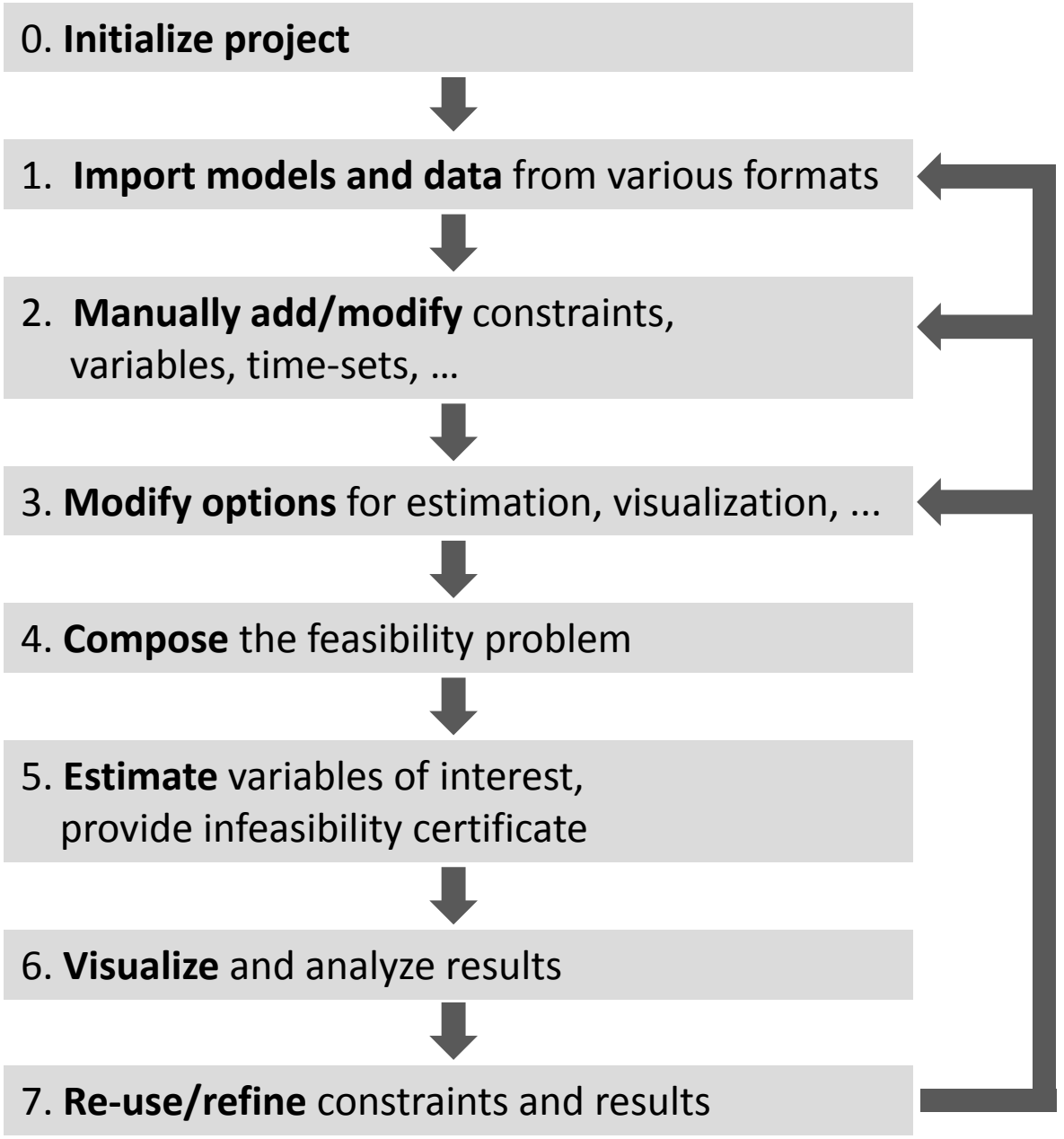
http://wwwifat.et.uni-magdeburg.de/syst/ADMIT

Note: make sure that you have installed the toolbox properly by running

```
>> installADMIT
```

in the toolbox main folder. If problems occur during installation then make sure to install required external tools or libraries (see also INSTALL.txt)

ADMIT: Typical workflow



ADMIT: Main functions, data types and objects

0. – 1. Initialize import models and data

Initialize:

`ADMITproject`

• creates empty or loads existing project

Examples:

```
>> opt = ADMITproject()
>> opt = ADMITproject('MichaelisMenten.opt')
```

Import existing models:

`ADMITimportModel`

• accepted formats:

- `SBML` or `xml`
- SBToolbox2 models (file extensions : `'txt'` or `'txtbc'`)

Examples:

To import a dynamical model stored in a xml file:

```
>> opt = ADMITimportModel(opt,'t_sim','MichaelisMenten.xml')
```

This will add the constraints imposed by the dynamics of the model on the time-set (see below) `t_sim`.

1. Import and preprocess data

Import data:

`ADMITimportData`

- imports data from files, accepted formats: `'cvs', 'txt' and others`
- multiple experiments supported

Preprocess data:

`ADMITprocessData`

- adds absolute or relative uncertainty
- adds an uncertainty description to the data, e.g. 'monotonicity', 'decreasing',…

Add data:

`ADMITaddData`

- adds data to the project

Example:

```
>> data = ADMITimportData(…
            './examples/CarnitineShuttle/carnitinedata.dat')
```

Add relative uncertainty of 5% on `x1` at the time-points 0:250, then plot data

```
>> data = ADMITprocessData(data,'x1',[0:250],0.05)
>> ADMITplotData(data,'x1')
```

Add data to previously generated ADMITproject:

```
>> opt = ADMITaddData(opt,data)
```

2. Manually define time-sets

Define time-sets:

`ADMITtime(timeDef)`

- defines a time-sets
- can contain numerical values or names of other time-sets
- multiple time-sets can be defined

Syntax:

`t_name` := `{ ... }`

`t_name`: name of time-set, always starting with '`t_`'

`{ ... }`: set containing numbers, vectors, or other time-sets

Examples:

```
>> ADMITtime('t_0 := 0')    % can leave out {} if it contains a single number
>> ADMITtime('t_dyn := { 0.1:0.1:10 }')
>> ADMITtime('t_sim := { t_0, t_dyn }')
```

Add time-set to ADMITproject using '+':

```
>> opt = ADMITproject('MichaelisMenten.opt')
>> opt = opt + ADMITtime('t_dyn := { 0.1:0.1:10 }')
```

2. Manually define variables

Define variables:

`ADMITvariable(variableDefinition)`

- creates a variable
- set data-type and time-variance properties
- mark a variable to be estimated (i.e. of interest)

Syntax:

`varName` := `{ ... }`

name of variable — comma-separated list of properties

Properties:

• variable type (default `real`):	`real | binary | integer`
• time-variance (default `timeInvariant`):	`timeVariant | timeInvariant`
• to be estimated (default not-set):	`ofInterest`
• to be estimated at a certain time:	`ofInterest(t_0)`

Examples:

```
>> ADMITvariable('p1 := {real, ofInterest}')
>> ADMITvariable('x1 := {real, timeVariant}')
>> ADMITvariable('x2 := {real, timeVariant, ofInterest(t_0)}')
```

Add variable to ADMITproject using '+':

```
>> opt = opt + ADMITvariable('p2 := {real}')
```

2. Manually define constraints

Define constraint:

`ADMITconstraint(constraintDefinition)`

- creates a constraint

Syntax: `conName (t=t_name) := conExpression`

- `conName`: constraint name
- `(t=t_name)`: assigned time-set
- `conExpression`: expression that defines the constraint

Constraint name:

- give the name of a variable to the constraint that defines it; all other constraints should not have names.
- named constraints can be used to remove (using `'-'` operator) variables from the project

Constraint time-set:

- assign elements from time-set `t_name` to all time-variant variables in formula
- can be left out if constraint contains only time-invariant variables
- `'t='` can be omitted

Constraint expression:

- format `[lb, ub]` defines bounds on a variable, i.e. `lb ≤ variable ≤ ub`
- define equality (==), inequality (<=, =>), or logical formula (<==>, &, |)
- formulae can contain time-variant and time-invariant variables
- formulae can contain polynomial or rational equations of the variables

2. Manually define constraints (continued)

Bounds for variables:

```
>> ADMITconstraint('p1 := [1,2]')
```
(i.e. 1≤p1≤2)

```
>> ADMITconstraint('x1(t_0) := [0.1,0.2]')
>> ADMITconstraint('x1(*) := [0,1]')
```

Note: use `*` to address all time-points, i.e. put global bounds on variable

Unnamed equality/inequality constraints:

```
>> ADMITconstraint('p1 >= p2 + p3')
>> ADMITconstraint('p2 == 1')
>> ADMITconstraint('(t=t_0) := x1(t) >= 0.1*x2(t)')
>> ADMITconstraint('(t=*)   := x2(t) <= 0.5')
```

Named equality constraints:

```
>> opt = opt + ADMITconstraint('p3 := p3 == p2 + 1')
```

Only equality constraint can be used as a named constraint

To remove a variable (defined by a named constraint) from the system use:

```
>> opt = opt - ADMITvariable('p3')
```

It will be solved for `p3` (i.e. `p3 == p2 + 1`) and each occurrence of `p3` in the ADMITproject will be substituted by `p2+1`

2. Manually define constraints (continued)

Using time-sets in constraints:

Expressions like `x1(0)` can be used to address certain time-points of a variable.
Expressions like `x1(t)` can be used to mark a variable as time-variant. Then

```
>> ADMITconstraint(…
   'x1(t=t_dyn) := x1(t) == x1(t-1) + p1*x1(t-1)*x2(t-1)')
```

will assign values from `t_dyn` to `x1(t)` and `x2(t)`. For instance, if `t_dyn` is defined as

```
>> opt = opt + ADMITtime('t_dyn := {0,1,2}')
```

then the above equation will be expanded in ADMITcompose to:

```
x1(t=1) := x1(1) == x1(0) + p1*x1(0)*x2(0)
x1(t=2) := x1(2) == x1(1) + p1*x1(1)*x2(1)
```

Logical constraints:

```
>> ADMITconstraint('b1 := b1 <==> a >= 0.15')
```

defines a binary variable `b1` to be 1 if and only if `a` is greater or equal to 0.15

```
>> ADMITconstraint('b2(t_dyn) := b2(t) <==> x1(t) >= 0.5')
```

defines a time-dependent binary variable `b2(t)` to be 1 if and only if `x1(t)≥0.5`

```
>> ADMITconstraint('b3 := b3 <==> &{b1,b2(t)}')
```

defines a binary variable `b3` to be 1 if and only if `b1==1` AND `b2(t)==1` for all t;
Note, similarly `b1` OR `b2(t)` can be expressed by: `'… |{b1,b2(t)}'`

3. Modify options

Modify options:

`ADMITgetOptions()` or `ADMITgetOptions(options,name)`

- returns default options or returns value of option 'name'

`ADMITsetOptions(options,name,value,name,value,...)`

- modify/set options

Syntax:

- options (as returned e.g. by ADMITgetOptions) contain different categories for the different tasks/purposes:

COMPOSE	: composition of feasibility problem
ESTIMATE	: estimation
SIMULATE	: Monte-Carlo simulation
YALMIP	: YALMIP options
PARALLEL	: parallelize estimation + simulation
PLOT	: plotting and visualization of results
DISPLAY	: display of messages

Option names:

`PREFIX.POSTFIX` (category, option) or `PREFIX.POSTFIX1.POSTFIX2` (category, sub-category, option)

Examples:

```
>> ops = ADMITsetOptions('ESTIMATE.outerBounding.use',1)
>> ops = ADMITsetOptions(ops,'YALMIP.solver','cplex')
```

4 – 6. Compose feasibility problem, perform estimation, visualize results

`ADMITcompose`

• composes the feasibility problem from the ADMITproject

`ADMITestimate`

• performs the estimation

`ADMITplotResults` or `ADMITplotBisectioning`

• plots all results, or plot bisectioning results (2D or 3D plot)

Example:

```
>> opt = ADMITproject('MichaelisMenten.opt')
```

... after adding further constraints and setting options, compose+estimation can be run:

```
>> optInfo = ADMITcompose(opt)
>> optResults = ADMITestimate(opt)
```

to plot the results:

```
>> ADMITplotResults(optResults)
```

7. Re-use and refine results

Example:

```
>> opt = ADMITproject('MichaelisMenten.opt')
```

... after adding further constraints and setting options, compose+estimation can be run:

```
>> optInfo = ADMITcompose(opt)
>> optResults = ADMITestimate(opt)
>> ADMITplotResults(optResults)
```

Now we want to estimate the initial conditions and leave the parameters unchanged; this is done by overwriting the 'ofInterest' properties of the variables:

```
>> opt = opt + ADMITvariable('s := {real,ofInterest(0)}')
>> opt = opt + ADMITvariable('s := {real,ofInterest(0)}')
>> opt = opt + ADMITvariable('p1 := {real}')
>> opt = opt + ADMITvariable('p2 := {real}')
>> opt = opt + ADMITvariable('p3 := {real}')
```

We also add another constraint:

```
>> opt = opt + ADMITconstraint('p3 >= p2')
```

Then we re-use the previous estimation results and start estimation again:

```
>> data = ADMITimportData(data,optResult,'Reuse')
>> opt = ADMITaddData(opt,data)
>> optInfo = ADMITcompose(opt)
>> optResults = ADMITestimate(opt)
```

References (selected)

- P. Rumschinski, S. Borchers, S. Bosio, R. Weismantel, and R. Findeisen. *Set-based dynamical parameter estimation and model invalidation for biochemical reaction networks*. BMC Systems Biology, 4:69, 2010.
- S. Borchers, S. Bosio, R. Findeisen, U. Haus, P. Rumschinski, and R. Weismantel. *Graph problems arising from parameter identification of discrete dynamical systems*. Mathematical Methods of Operations Research, 73(3), 381-400. 2011.
- J. Hasenauer, P. Rumschinski, S. Waldherr, S. Borchers, F. Allgöwer, and R. Findeisen. *Guaranteed steady state bounds for uncertain biochemical processes using infeasibility certificates*. J. Proc. Contr., 20(9):1076-1083, 2010.
- S. Borchers, P. Rumschinski, S. Bosio, R. Weismantel, and R. Findeisen. *A set-based framework for coherent model invalidation and parameter estimation of discrete time nonlinear systems*. In 48th IEEE Conf. on Decision and Control, pages 6786 - 6792, Shanghai, China, 2009.
- P. Rumschinski , S. Streif . R. Findeisen. *Combining qualitative information and semi-quantitative data for guaranteed invalidation of biochemical network models*. Int. J. Robust Nonlin. Control, 2012. In Press.
- S. Streif, S. Waldherr, F. Allgöwer, and R. Findeisen. *Systems Analysis of Biological Networks,* chapter *Steady state sensitivity analysis of biochemical reaction networks: a brief review and new methods*, pages 129 -148. Methods in Bioengineering. Artech House MIT Press, August 2009.
- A. Savchenko, P. Rumschinski , R. Findeisen. *Fault diagnosis for polynomial hybrid systems*. In 18th IFAC World Congress, Milan, Italy, 2011