Semi-Automatic Documentation for AMIGO

The automatic documentation would cover the 3 main structures

* inputs
* privstruct
* results

Also helps to document the necessary inputs for the present and future main tasks, such as *AMIGO\_Prep, AMIGO\_PE, AMIGO\_SObs, AMIGO\_SData*, etc.

For each structure we create a corresponding **documentation structure** (for example: *inputs, doc\_inputs*). Ideally, the documentation structures has the same field as the original, but the field value is a string containing the description of that field, for example:

inputs.PEsol.PEcost\_type = 'llk'; %default value

doc\_inputs.PEsol.PEcost\_type = 'cost function type #LVL2 #GREEN';

The documentation structures are stored in a MATLAB .m file (like the AMIGO input files). *AMIGO\_default\_options\_doc.m* contains the documentation structure for the inputs.

The documentation is not only a short comment on the corresponding field, but we want to know the **default options**. AMIGO\_defualt\_options.m already creates a structure with the needed information for the inputs. We just need to print it (see later).

Some fields can take only a couple of values, for example *llk\_type, lsq\_type, nlpsolver* etc. for which we should document the **available options**. Most of the information is already in the comments of AMIGO\_private\_defaults, AMIGO\_users\_guide and the first draft of the cheat sheet. We either add to the field description or create a structure:

option A:

doc\_inputs.PEsol.PEcost\_type = 'cost function type #LVL2 #GREEN #DEF llk|lsq';

option B:

opt\_inputs.PEsol.PEcost\_type = 'llk|lsq';

Once we have these 3-4 structures, we **have all the information to generate** the main part/skeleton of the documentation. We can also **detect missing documentation**, easily by automated comparison of the used structures and the documentation structures.

By using **TAGs,** e.g. #MISSING, #PE, #LVL1, in the documentation fields we can help to process the documentation later, for example: print the fields in the order of importance, or highlight some fields by colours.

This way it is **easier to maintain** the documentation, since if we add a new field to one of the three structure, then we just have to go to the .m file and add the corresponding field to the documentation structures too.

# Example: document the inputs structure

The defaults for the inputs can be obtained by:

>> inputs = AMIGO\_default\_options;

The corresponding documentation is called by

>> doc = AMIGO\_default\_options\_doc;

We process the inputs and its documentation. The following code also takes care about the missing documentation by inserting a #MISSING tag:

>> doc\_inputs = AMIGO\_commentStruct(inputs,doc);

We can see the results by

>> inputs = inputs\_doc;

>> AMIGO\_displayStruct(inputs,fid)

inputs.

OEDsol.

OEDcost\_type: 'OED criteria'

cost\_file: '#MISSING'

n\_obs\_od: '#MISSING'

PEsol.

CramerRao: 'cramer-rao computation'

GRankmethod: 'Latin hypercube sampling'

...

The command:

*AMIGO\_generate\_developers\_cheat\_sheet();* writes the output in a HTML file, in a table format, that can be further formatted.

# Document the input for new AMIGO tasks

When we develop a new task usually we are creating an AMIGO input file, that illustrates the use of the new functionality.

I take the example from Examples\Arabidopsis\_circadian.

>> eval('circadian\_pe')

>> doc = AMIGO\_default\_options\_doc;

>> doc\_inputs = AMIGO\_commentStruct(inputs,doc);

>> pe\_inputs=doc\_inputs;

At this point, pe\_inputs contains the documentation for those fields that exist in the circadian\_pe. It is up to us, if we print it on screen, save to file for further modification etc.

>> AMIGO\_displayStruct(pe\_inputs)

pe\_inputs.

PEsol.

PEcost\_type: 'cost function type'

global\_theta\_guess: 'global estimated par. Init guess'

global\_theta\_max: 'upper bounds for global est. Parameters'

global\_theta\_min: 'lower bounds for global est. Parameters'

id\_global\_theta: 'estimated parameter name'

llk\_type: []

exps.

data\_type: 'pseudo exp. Data gen.'

error\_data: 'std. Dev. of the error'

exp\_data: 'exp. Data'

exp\_y0: 'initial condition'

n\_exp: 'number of experiments'

n\_obs: 'number of observables'

n\_pulses: []

n\_s: 'number of sampling time'

noise\_type: []

obs: 'observation equations'

obs\_names: 'observables names'

t\_con: 'time of input changing'

t\_f: 'final time for IVP'

u: 'stimuli values'

u\_interp: []

u\_max: 'max bounds on inputs'

u\_min: 'min bounds on inputs'

model.

eqns: 'char array containing the model eqns'

input\_model\_type: 'model description type'

n\_par: 'number of parameters'

n\_st: 'number of states'

n\_stimulus: 'number of stimuli'

par: 'model parameter values'

par\_names: 'model parameter names'

st\_names: 'custom state names'

stimulus\_names: 'stimuli names'

pathd.

results\_folder: 'Problem'

short\_name: 'problem'

# Codes that operate on structures (print, merge)

Our structures have a few levels, that is hard to see and there is no MATLAB tools to **show/print to file the whole content of these structures**. Some new utilities helps to merge and display these structures, recursively.

*AMIGO\_displayStruct(inputs):* prints the structure to the command window. You can do it directly to a text file.

>> AMIGO\_displayStruct(inputs)

inputs.

OEDsol.

OEDcost\_type: ' OED criteria '

cost\_file: '#MISSING'

n\_obs\_od: '#MISSING'

*AMIGO\_displayStruct\_fullsyntax(inputs):* prints the structure to the command window.

>> AMIGO\_displayStruct\_fullsyntax(inputs)

inputs.OEDsol.OEDcost\_type= ' OED criteria '

inputs.OEDsol.cost\_file= '#MISSING'

inputs.OEDsol.n\_obs\_od= '#MISSING'

inputs.PEsol.CramerRao= ' cramer-rao computation '

*AMIGO\_commentStruct(inputs,inputs\_doc)*: replaces the fields of inputs by the corresponding fields of inputs\_doc. If the corresponding field is missing, #MISSING is assigned. See previous.