User Reference Manual of ByoDyn version 4.8

Adrián López García de Lomana, Alex Gómez-Garrido, Miguel Hernández, Pau Rué Queralt and Jordi Villà-Freixa

— December 17, 2008 —

This document gives detailed description about the functionalities of ByoDyn.

Computational Biochemistry and Biophysics Laboratory
Research Unit on Biomedical Informatics
Universitat Pompeu Fabra - Institut Municipal d'Investigació Mèdica
c/ Dr. Aiguader 88, 08003, Barcelona, Spain
http://cbbl.imim.es

Contents

1 The ByoDyn Team 4

2 ByoDyn 4

2.1 ByoDyn Outputs . . . 5
2.2 ByoDyn Input Files . . . 5
2.2.1 ByoDyn Option File . . . 5
2.2.1.1 Mandatory Option File Arguments . . . 5
2.2.1.2 Optional Option File Arguments . . . 5
2.2.2 ByoDyn Model Files . . . 6
2.2.2.1 SBML files . . . 6
2.2.2.1.1 SBML Compatibility . . . 6
2.2.2.2 Tag Format Files . . . 9
2.2.2.2.1 Affectors . . . 10
2.3 ByoDyn Functionalities . . . 15
2.3.1 Exporting . . . 15
2.3.1.1 Specific Option File Arguments . . . 15
2.3.1.2 Specific Outputs . . . 15
2.3.2 Deterministic Simulation . . . 15
2.3.2.1 Specific Option File Arguments . . . 16
2.3.2.2 Specific Outputs . . . 16
2.3.2.3 Simulation Engines . . . 17
2.3.2.3.1 SciPy . . . 17
2.3.2.3.2 Octave . . . 18
2.3.2.3.3 Open Modelica . . . 18
2.3.2.3.4 XPPAUT . . . 19
2.3.2.3.5 Others . . . 19
2.3.3 Stochastic Simulation . . . 20
2.3.3.1 Specific Option File Arguments . . . 20
2.3.3.2 Specific Outputs . . . 20
2.3.4 Optimisation . . . 21
2.3.4.1 Specific Option File Arguments . . . 21
2.3.4.2 Specific Ouputs . . . 23
2.3.4.3 Minimization Methods . . . 23
2.3.4.3.1 Random Search . . . 23
2.3.4.3.2 Local Search . . . 24
2.3.4.3.3 Genetic Algorithm . . . 24
2.3.4.3.4 Hybrid Algorithm . . . 25
2.3.5 Fitness Function Calculation . . . 26
2.3.6 Trajectories Reconstruction . . . 26
2.3.7 Fitness Function Surfaces . . . 27
2.3.7.1 Specific Option File Arguments . . . 27
2.3.7.2 Specific Outputs . . . 28
2.3.8 Sensitivity Analysis . . . 28

2.3.8.1 Specific Option File Arguments . . . 28
2.3.8.2 Specific Outputs . . . 29
2.3.9 Identifiability Analysis . . . 30
2.3.9.1 Specific Option File Arguments . . . 32
2.3.9.2 Specific Outputs . . . 32
2.3.10 Optimal Experimental Design . . . 33
2.3.10.1 Specific Option File Arguments . . . 33
2.3.10.2 Specific Outputs . . . 34

3 `ByoDyn` on Parallel Environments **34**
3.1 Performance . . . 35
3.2 External Software Requirements . . . 36
3.3 Launching `ByoDyn` in Parallel . . . 36
3.4 QosCosGrid . . . 36

4 Acknowledgements **36**

1 The ByoDyn Team

Several people contributed to the development and improvement of ByoDyn. ByoDyn is a joint effort from the Computational Biochemistry and Biophysics Laboratory. It is an original idea from Jordi Villà i Freixa and Adrián López García de Lomana. Essential contributions were implemented by Àlex Gómez Garrido who improved most of the software functionalities (SBML compatibility, sensitivity analysis, identifiability analysis and other issues) and Miguel Hernández who implemented the parallel code and the byodyn-web server.

- **Jordi Villà i Freixa**: Head of the Computational Biochemistry and Biophysics Laboratory.
- **Adrián López García de Lomana**: Born in Vitoria (Spain) in 1981. Graduated in Biochemistry at University of Navarra (Spain). Currently he is a Pompeu Fabra University graduate student at the Computational Biochemistry and Biophysics Laboratory.
- **Àlex Gómez Garrido**: Born in Barcelona (Spain) in 1983. Graduated in Biology at Pompeu Fabra University (Barcelona, Spain) with the specialty of Biomedical Research the year 2006. M.Sc. Bioinformatics for Health Sciences at the Pompeu Fabra University completed the year 2008. Nowadays, he is a Ph.D. student in Biomedicine at the Computational Biochemistry and Biophysics Laboratory.
- **Miguel Hernández**: Grauated in Engineer in Informatics Systems by the Tecnológico de Monterrey (Mexico) the year 2003. He completed his M.Sc. Bioinformatics for Health Sciences at the Pompeu Fabra University the year 2008.
- **Pau Rué Queralt** He recieved a B.Sc. in Applied Mathematics at Universitat Politècnica de Catalunya in year 2005. Currently coursing a M.Sc. in Bioinformatics for Health Sciences at Universitat Pompeu Fabra and Universitat de Barcelona. Nowadays he holds a fellowship from La Caixa. His present research interests are among others, τ-leap Runge-Kutta methods with extended stability domains for the efficient approximate simulation of stochastic chemical kinetics. Within ByoDyn, he has developed the implementation of the stochastic simulation.

2 ByoDyn

ByoDyn is a command line program. To run the program you need to call the executable `byodyn`. Several basic arguments are accepted:

- `-h` or `--help`: tells how to access the complete documentation of ByoDyn.
- `-v` or `--version`: prints on the terminal the current version of the ByoDyn executable.
- `-t` or `--testing`: runs the internal tests for ByoDyn. Please run twice the command `byodyn -t` to remove the compiled python sources `.pyc`, otherwise bugs due to those files may not be detected.
- `-e` or `--example`: creates a directory called `examples` with the required files to follow the Quick Start Guide.

- `-o` or `--output` `nameOfOutputDirectory`: creates an output directory named `nameOfOutputDirectory` different from the default directory name `output`.

Alternatively we can set an option file as argument. We generally set `.rn` for the extension of the option file and we name it *runner* file although any UNIX name is valid.

2.1 `ByoDyn` Outputs

All files generated by `ByoDyn` will be created by default in a directory called `output` at the place where you execute the command. In this directory called `output` the user can find several files of interest. Additionally, inside `output` there is a directory called `scratch` where `ByoDyn` stores files that in principle are not so directed to user inspection but necessary for `ByoDyn` to run. If the user wants to name `output` differently she can use the option `-o` or `--output` and the name of the directory. Besides if the users wants to place the output in another location rather than where you execute the command, she can use the same options to specify an *absolute* path where the outputs will be created.

2.2 `ByoDyn` Input Files

2.2.1 `ByoDyn` Option File

The `ByoDyn` option file is a text file where you define the options of `ByoDyn` for the run. Commented lines start by #. The common structure of the file is the following:
`tagName argument1 argument2 ...`
Each functionality is called using the tags that are explained at each section but, furthermore, common requirements are explained in this section. Please refer to Section 2.3 for the further required arguments to use the different analysis tools of `ByoDyn`.

2.2.1.1 Mandatory Option File Arguments The following arguments are mandatory for all the funcionalities.

- `modelFile argument`

  where `argument` will be substituted by the complete path to the model input file or simply the name of the model input file if the model input file lays at the same directory from which the calculation is launched.

- `modelFormat argument`

  where `argument` should be substituted by `SBML` or `tags` depending on the model file.

2.2.1.2 Optional Option File Arguments The following arguments are optionals but can be used with all the funcionalities.

- `integrationMethod argument1 argument2`

  this line determines the software and the numerical method used to solve the system of differential equations. The two arguments are required.

- `argument1`: several tools can be called to integrate the system of differential equations. Valid options are `python` for `SciPy` routines, `octave` for `Octave`, `openModelica` for `Open Modelica`, `xpp` for `XPPAUT` and `automatic`. `matlab` option is valid but the code to use `MATLAB` is still under development on this version. `Open Modelica` is the most compatible with the SBML features. Also, `XPPAUT` option will be required to solve SBML models holding delay functions. Furthemore, if the user does not want to choose the integration method for hiself or he does not know which is the best integrator for his model, he can use the `automatic` option, in this case ByoDyn choose the best integrator depending of model features and the software installed.
- `argument2`: this argument specifies the numerical method used to integrate the system of differential equations. Available options are `default`, `adams`, `non-stiff`, `bdf` and `stiff`. The most commonly option will be `default`. The other four options are only compatible using `SciPy` or `Octave`.

- `parameter argument1 argument2 ...`

  If we are interested on running a simulation of a model varying the value of one of its parameters, alternatively to edit the model, we can specify its value using the `tagName parameter` and then, separated by tabs, the parameters to set using the following syntax:

  ```
  parameterName=parameterValue
  ```

  To avoid parsing problems, please use this syntax strictly, specially avoid blank spaces like in:

  ```
  parameterName = parameterValue
  ```

  Note that this option is only valid while working with SBML format model files.

- `checkConsistencySBML`

  This variable indicates that `ByoDyn` will run a strict full checking of SBML file syntax using the method `checkConsistency` of libSBML and it will return the errors if any. No further arguments are required.

- `figureFormat argument`

  This variable is used to select alternative formats for the output figures. By default figures will be postscript files, but figures on PNG format are also available. The variable `figureFormat` takes `ps` or `png` as possible values for `argument`.

2.2.2 `ByoDyn` Model Files

2.2.2.1 SBML files This is the preferred input format for the models subject to analysis by `ByoDyn`. For further information please check
http://sbml.org/index.psp.

2.2.2.1.1 SBML Compatibility Our intention is that `ByoDyn` would be compatible with any valid SBML[Hucka et al., 2003] file. Currently we are testing against the eleventh release of the

BioModels Database [Le Novère et al., 2006]. Previously, we have tested against the ninth release, where 149 of 150 models was simulated correctly. The results have been published in the BioUML home page where a table shows the comparison of different SBML simulators (http://www.biouml.org/biomod The detailed results are also published at http://www.sys-bio.org/, in the comparison section http://sys-bio.org/fbergman/compare/. We also include in Table 1 information about the supported SBML features, indicating which is the most suitable `ByoDyn` integration option:

	SBML feature						
Simulation Engine	Rate Rules	Assignment Rules	Algebraic Rules	Events	Events with Delays	Delay	Function Definition
SciPy	✔	✘	✘	✘	✘	✘	✔
Octave	✔	✔	✔	✘	✘	✘	✔
Open Modelica	✔	✔	✔	✔	✔	✘	✔
XPPAUT	✔	✔	✘	✘	✘	✔	✔

Table 1: SBML features compatible with ByoDyn.

2.2.2.2 Tag Format Files This is a home made format. Using this format we can specify multicellular systems although the systems will be restricted to static 2D matrices. Each cell will host the same biochemical topology and will eventually interact with 8 neighbour cells. The format of specifying the model is here outlined. First, commented lines start by # and will not be read. Similarly to the `ByoDyn` option file described in section 2.2.1 we will specify each feature line by line using the name of the variable and the value of the variable separated by tabular.

```
variableName  variableValue1  variableValue2  ...
```

Nowadays `ByoDyn` handles six variables to define a model:

- `systemName  argument`

  where `argument` will be the name of model.

- `xlength  argument`

  where `argument` will be an integer specifying the cellular length of the system.

- `ywidth  argument`

  where `argument` will be an integer specifying the cellular width of the system.

- `nodes  argument1  argument2  ...`

  where arguments are the names of nodes of the biochemical system.

- `topology  topologyType  constant1  constant2  ...`

  - `topology` tag is used to specify the biochemical relationships among nodes.
  - `topologyType`, the first argument, has the following structure:

    `affectedNode/reactionType/affectingNode1/affectingNode2/...`

    * `affectedNode` is the biochemical node affected by the interaction. In the chemical reaction it represents a product, in the other hand, in the system of differential equations, `affectedNode`, will be at the left hand side of the equations.
    * `reactionType` refers to a tag which specifies the biochemical nature of the interaction. Please check section 2.2.2.2.1 for full explanations about the available possibilities.
    * `affectingNode1`, `affectingNode2`, ... are the nodes affecting the rate of change of `affectedNode`. The number of nodes is specific of each `reactionType`, please have look to the specific one at section 2.2.2.2.1. At any case, the affecting nodes are understood as reactants in the chemical reaction and the terms representing them will be placed at the right hand side of the equations.

    Keep in mind that any of the affected or affecting nodes had to be defined at the `nodes` variable.

  - The following arguments, `constant1`, `constant2` ... refer to the reaction constants of `reactionType`. The number of arguments varies depending on the affector, please check

section 2.2.2.2.1 for more information. At any case the general structure of the constants will be the following:

`constantName/value`

* `constantName` the name of the constant. Please use any alphanumeric Al combination, specially, avoid using points (".").
* `value` a floating value of the defined constant.

- `initialCondition node argument1 argument2 ...`
  - `initialCondition` is the tag to specify the initial conditions of the system.
  - `node` defines any of the nodes of the system defined in the variable `nodes`.
  - Then with `argument1`, `argument2`, ... we specify for each cell of the system the initial concentration of node `node` with the following syntax:

    `posX,posY/value`

    * `posX,posY` defines the position of a given cell. In the case of a unicellular system, the position will be specified as `0,0`.
    * `value` is a floating number defining the value of the initial concentration of the particular node in the specified cell.

2.2.2.2.1 Affectors Affectors are a family of tags of common biochemical processes that encode a mathematical term on the differential equation affecting the rate of change of a specific node. We mainly discriminate between dimensional and non-dimensional affectors. Dimensional affectors are general mathematical terms for the determination of dynamics of specific biological phenomena. On the other hand, non-dimensional affectors are mathematical terms derived from the adimensionalisation of the model described in García de Lomana et al. [2008], related to lateral inhibition during pattern formation. For the general dimensional terms, here we describe the currently available ones:

- `complexExtraBack`: This affector defines the rate of change of a ligand and a receptor when forming a complex on the extracellular matrix. The complex rate of change will be defined by the affector `complexExtraFwd`. The ligand and the receptor sit on different cells. The syntax of the `topologyType` is the following:

  `node/complexExtraBack/receptorNode/ligandNode`

  Furthermore it requires a single constant, the binding constant. The mathematical term added to the differential equations are two, one to the ligand (l) the other to the receptor (r):

$$\frac{\mathrm{d}[r]_i}{\mathrm{d}t} = -1/64\ k_{r-l}^{binding}[r]_i \sum_{j=1}^{j=k}[l]_j \tag{1}$$

$$\frac{\mathrm{d}[l]_i}{\mathrm{d}t} = -1/64\ k_{r-l}^{binding}[l]_i \sum_{j=1}^{j=k}[r]_j \tag{2}$$

Just note that 1/64 comes from the understanding that each cell interacts with its $k = 8$ neighbours. Therefore, if we assume that membrane proteins distribute homogeneously along the cell surface, 1/8 of the total amount of a particular membrane protein at each cell will interact with 1/8 of the total amount of the interacting protein for each of the k neighbour cells.

- `complexExtraFwd`: This affector defines the rate of change of the complex when ligand and receptor sit on different cells. The ligand and receptor rate of change will be defined by the affector `complexExtraBack`. The syntax of the `topologyType` is the following:

  `complexNode/complexExtraFwd/receptorNode/ligandNode`

  It also requires additionally a single constant, the binding constant. The mathematical term added to the differential equation of the rate of change of the complex (c) is the following (r and l refers to receptor and ligand respectively):

$$\frac{\mathrm{d}[c]_i}{\mathrm{d}t} = +1/64\ k_{r-l}^{binding}[r]_i \sum_{j=1}^{j=k}[l]_j \tag{3}$$

  Again note that the 1/64 comes from the fact that each cell interacts with its $k = 8$ neighbours and the assumption that membrane proteins distribute homogeneously along the cell surface.

- `constant`: This affector defines the rate of change of a constant node (n), then the mathematical term added to the differential equation will be simply zero:

$$\frac{\mathrm{d}[n]_i}{\mathrm{d}t} = +0 \tag{4}$$

  The syntax of the `topologyType` is the following:

  `node/constant`

  No further constants are required.

- `constitutive`: This affector defines the rate of change of a node expressed constitutively. The syntax of the `topologyType` is the following:

  `node/constitutive`

  A further single `constant1` field is required. The constant (r) must take any positive value: $r > 0$. The mathematical term added to the differential equation of the node (n) will be:

$$\frac{\mathrm{d}[n]_i}{\mathrm{d}t} = +r \tag{5}$$

- `degradation`: This affector defines the rate of change of a node due to degradation. The syntax of the `topologyType` is the following:

  `node/degradation/node`

  An additional degradation constant is required. The mathematical term added to the degradating node (n) will be:

$$\frac{\mathrm{d}[n]_i}{\mathrm{d}t} = -k^{degradation}[n]_i \tag{6}$$

- `dissociationExtraBack`: This affector defines the rate of change of a complex due to dissociation of the receptor and ligand. In this case the complex is formed extracellularly, belonging the ligand and the receptor to different cells. The rate of change of the two nodes resulting from the dissociation will be defined by `dissociationExtraFwd`. An additional dissociation constant is required. The syntax used for the `topologyType` of `dissociationExtraBack` is:

  `complexNode/dissociationExtraBack/resultingDissociationReceptor/resultingDissociationLigand`

  Note that `resultingDissociationReceptor` and `resultingDissociationLigand` are not necessarily the nodes `receptorNode` and the `ligandNode` respectively from `topologyType` `complexExtraBack` or `complexExtraFwd`. After the complex dissociation two new nodes may arise. Finally, the mathematical term added to the differential equation of the complex (c) will be:

$$\frac{\mathrm{d}[c]_i}{\mathrm{d}t} = -k_c^{dissociation} \sum_{j=1}^{j=k} \frac{1}{8}[c]_i \tag{7}$$

  We assume that the membrane complex is distributed homogeneously along the cell surface and that $k = 8$ neighbours are in contact of each cell. Using the above formula we take into consideration boundary effects.

- `dissociationExtraFwd`: This affector defines the rate of change of the two nodes resulting from the dissociation of a complex formed from membrane proteins of two different cells. On the other hand the rate of change of the complex due to its dissociation will be defined by `dissociationExtraBack`. The syntax of the `topologyType` of `dissociationExtraFwd` will be:

  `resultingDissociationReceptor/dissociationExtraFwd/resultingDissociationReceptor/resultingDissociationLigand/complexNode` or

  `resultingDissociationLigand/dissociationExtraFwd/resultingDissociationReceptor/resultingDissociationLigand/complexNode`

  for the resulting nodes respectively, depending on if they come from the ligand or the receptor nodes forming the complex. Also, an additional dissociation constant is required. The mathematical terms affecting the new nodes are:

$$\frac{\mathrm{d}[r']_i}{\mathrm{d}t} = +k_c^{dissociation} \sum_{j=1}^{j=k} \frac{1}{8}[c]_i \tag{8}$$

$$\frac{\mathrm{d}[l']_i}{\mathrm{d}t} = +k_c^{dissociation} \sum_{j=1}^{j=k} \frac{1}{8}[c]_j \tag{9}$$

  where c is the complex and r' and l' are the receptor and ligand derived new nodes respectively. k is the number of neighbour cells of cell i.

- `inhibition`: This affector defines the rate of change of a node due to its inhibition by another one. The syntax of the `topologyType` will be:

  `inhibitedNode/inhibition/inhibitor`

  Further three constants should be added, a basal rate, an inhibition constant and a cooperativity coefficient. The mathematical term added to the differential equation of the inhibited

node (n) is:

$$\frac{\mathrm{d}[n]_i}{\mathrm{d}t} = + \frac{k^{basal}}{1 + ([I]_i / k_{inhibition})^s} \tag{10}$$

being k^{basal} a basal expression rate, $k_{inhibition}$ the inhibition constant, s the cooperativity coefficient and I the inhibitor.

- `transcription`: This affector defines the rate of change of a mRNA (m) due to transcription from a given transcription factor (TF). The syntax for the `topologyType` is the following:

  `transcriptNode/transcription/transcriptionFactor`

  Further three constants are required, the V_{max}, the K_M and a cooperativity coefficient (s). The mathematical term added to the differential equation of the transcript node (m) is the following:

$$\frac{\mathrm{d}[m]_i}{\mathrm{d}t} = + \frac{V_{max} [TF]_i^s}{K_M^s + [TF]_i^s} \tag{11}$$

- `translation`: This affector defines the rate of expression of a protein (p) due to its translation from a transcript (m). The syntax for the `topologyType` is:

  `proteinNode/translation/transcriptNode`

  A further constant, the translation rate, is required. The mathematical term added to the differential equation of node p will be:

$$\frac{\mathrm{d}[p]_i}{\mathrm{d}t} = + k^{translationRate} [m]_i \tag{12}$$

On the other hand other affectors are available, which were used for the models described in García de Lomana et al. [2008]. For the adimensionalisation of the terms we have assumed $t = T_0\ \tau$ and $[x]_i = [x]_0\ x_i(\tau)$:

- `NonDimConstitutiveDegradation`: This affector defines a constitutive expression and degradation for a given node (n). The syntax for the `topologyType` is:

  `node/NonDimConstitutiveDegradation/node`

  Two further constants are required, the constitutive expression constant (r_n) and the degradation constant ($K_\mathrm{deg}^\mathrm{n}$). The mathematical term added to the differential equations is the following:

$$\frac{\partial n_i}{\partial \tau} = T_0\ K_\mathrm{deg}^\mathrm{n}\ \ (r_\mathrm{node} - node_i(\tau)) \tag{13}$$

- `NonDimTranslationDegradationBinding`: This affector defines the rate of change of a binding membrane protein (p). The concentration is affected by the strength of translation from a gene product g, the amount of complex (c) formation (*binding*) and the degradation of the protein. The syntax for the `topologyType` is:

  `translatedProtein/NonDimTranslationDegradationBinding/geneTranscript/ligand`

Three additional constants are required, in order: $K^{\mathrm{p}}_{\mathrm{deg}}$ as degradation constant for the protein, $K^{\mathrm{p}}_{\mathrm{bind}}$ as binding constant with the ligand and finally a characteristic concentration, $[l]_0$ which refers to the ligand l. The mathematical term added to the differential equations would be:

$$\begin{aligned}\frac{\partial p_i}{\partial \tau} &= T_0\ K^{\mathrm{p}}_{\mathrm{deg}}\ (g_i(\tau) - p_i(\tau)) \\ &\quad -\frac{k}{n^2}\ K^{\mathrm{c}}_{\mathrm{bind}}\ [l]_0\ p_i(\tau)\ \sum_{j=1}^{k} l_j(\tau)\end{aligned} \tag{14}$$

where n defines the number of possible neighbouring cells, 8 in our case, and k refers to the actual number of neighbour cells of cell i.

- `NonDimInhibitionDegradation`: This affector defines the rate of change of a gene transcript (g) whose transcription is regulated by an inhibitor (I). The syntax for the `topologyType` is:

  `geneTranscript/NonDimInhibitionDegradation/inhibitor`

  Three additional constants are required. $K^{\mathrm{g}}_{\mathrm{deg}}$, the degradation rate of g, κ_{I} which indicates the inhibitor concentration at which the gene transcript is expressed at half of the maximal concentration and s which is a cooperative coefficient of the regulation. The mathematical term added to the differential equations is the following:

$$\frac{\partial g_i}{\partial \tau} = T_0\ K^{\mathrm{g}}_{\mathrm{deg}}\ \left(1 - \frac{I_i^s(\tau)}{\kappa_{\mathrm{I}} + I_i^s(\tau)} - g_i(\tau)\right) \tag{15}$$

- `NonDimBindingDegradation`: This affector defines the rate of change of a membrane complex (C). The biological phenomena affecting the concentration of C would be complex formation or binding from a receptor R and a ligand L and degradation. The syntax for the `topologyType` is:

  `complex/NonDimBindingDegradation/receptor/ligand`

  Three additional constants are required. $K^{\mathrm{C}}_{\mathrm{deg}}$, the degradation rate of C, $K^{\mathrm{p}}_{\mathrm{bind}}$ as binding constant with the ligand and finally a characteristic concentration, $[L]_0$ which refers to the ligand. The mathematical term added to the differential equations is the following:

$$\frac{\partial C_i}{\partial \tau} = T_0\ \left(\frac{k}{n^2}\ K^{\mathrm{C}}_{\mathrm{bind}}\ [L]_0\ R_i(\tau)\ \sum_{j=1}^{k} L_j(\tau) - K^{\mathrm{C}}_{\mathrm{deg}}\ C_i(\tau)\right) \tag{16}$$

  where again k and n refer respectively to the actual and possible number of neighbour cells.

- `NonDimTranscriptionDegradation`: This affector refers to the rate of change of a gene transcript g which is positively regulated by an activator A and finally it is also subjected to degradation. The syntax for the `topologyType` is:

  `geneTranscript/NonDimTranscriptionDegradation/Activator`

  Three additional constants are required. $K^{\mathrm{g}}_{\mathrm{deg}}$, the degradation rate of g, κ_{A} which indicates

the activator concentration at which the gene transcript is expressed at half of the maximal concentration and s which is a cooperative coefficient of the regulation. The mathematical term added to the differential equations is the following:

$$\frac{\partial g_i}{\partial \tau} = T_0 \; K^{\mathrm{g}}_{\mathrm{deg}} \; \left(\frac{A_i^s(\tau)}{\kappa_{\mathrm{A}} + A_i^s(\tau)} - g_i(\tau) \right) \tag{17}$$

- `NonDimTranslationDegradation`: This affector refers to the rate of change of a protein p which is translated from a gene transcript g and degradated. The syntax for the `topologyType` is:

  `protein/NonDimTranslationDegradation/geneTranscript`

  Only the degradation constant is required, $K^{\mathrm{p}}_{\mathrm{deg}}$. The mathematical term added to the differential equations is the following:

$$\frac{\partial p_i}{\partial \tau} = T_0 \; K^{\mathrm{p}}_{\mathrm{deg}} \; (g_i(\tau) - p_i(\tau)) \tag{18}$$

2.3 `ByoDyn` Functionalities

Different types of analysis can be performed using `ByoDyn`. Directing the flow of the program to a specific task is defined in the `ByoDyn` option file, mainly with the `runningType` argument. Possible values for `runningType` are `exporting`, `simulation`, `parameterEstimation`, `calculateFunction`, `dynamicsReconstruction`, `scoreSurface`, `sensitivityAnalysis`, `identifiabilityAnalysis` or `optimalExperimentalDesign`.

2.3.1 Exporting

Instead of changing the parameter value of a given model using other methods, you can modify the SBML model using the exporting option of `ByoDyn`.

2.3.1.1 Specific Option File Arguments Just in this special case only four variables are required in the runner file:

- `modelFile` and `modelFormat` as described on Section 2.2.1.
- `runningType exporting`

  Only one argument, `exporting`, is required.
- `parameter` as described in Section 2.2.1.2 used to specify the model new parameter values.

2.3.1.2 Specific Outputs Again, in this special case, only a new SBML file will be created on the `output` directory. The file will maintain the `.xml` extension and the name will be taken from the model name defined on the original SBML file. `scratch` directory will be empty.

2.3.2 Deterministic Simulation

Simulation of a model consists on the numerical integration of the system of differential equations.

2.3.2.1 Specific Option File Arguments

Two necessary arguments need to be added to the option file to run a simulation:

- `runningType argument1 argument2`

  Two values for the variable `runningType` need to be defined. The first one, `argument1`, will be `simulation`. The second one, `argument2` will be either `velocity` or `noVelocity` whether if you want or not to analyse not only the integration of the system of differential equations but also the rate of change of the nodes' concentration along time.

- `time&timestep argument1 argument2`

  Here again two values for the variable `time&timestep` are required. `argument1` will be a float defining the total time of the simulation and `argument2` another float defining the time step.

Furthermore, other non-required options can be set:

- `plotKeys argument`

  When models hold many nodes, trajectories graphs may be overwhelmed by nodes' names. `argument` can be set to `YES` or `NO` to plot or hide nodes' names. If the argument `plotKeys` is not specified, by default nodes' names are included in graphs.

- `optionalOutputFormat csv`

  This is an option referring to the format of the simulation output data. The values of the trajectories are stored by default in a text file (See 2.3.2.2) separated by tabular. If the user wants to save the data in a comma separated value format, this option should be used for an additional text file on the requested format.

- `withoutGraphics`

  For faster integration, the system of equations is solved but the plots are not created, saving a great amount of time if necessary. This optional `tagName` does not require further arguments.

- `separatedGraphs`

  For the unicellular models instead of plotting all the nodes in the same figure, sometimes, specially for models with large number of nodes and/or which different scales, it is much more convinient to see the trajectory of each node independently from the others. For that purpose, set the option `separatedGraphs` and instead of a figure, the user will find a directory called `separatedGraphs` within the output directory with a file for the trajectory of each node separately. Files will be named as nodes' names with `.ps` extension. If the argument `velocity` (see 2.3.3.1) was set, a new directoy called `velocity` within `separatedGraphs` will be created containing the same type of graphs but referred to the rate of change of the nodes' concentration along time.

2.3.2.2 Specific Outputs

Inside the output directory (See 2.1) we will find the following files common to any simulation:

- *modelName*`.description.txt`: A short description of the input model. We state the name of the model, the number of nodes, the number of constant species and the number of constant and non-constant parameters.

- *modelName*`.N.out`: This `.out` is a file with the concentration of each node (in columns) along time (first column). `N` refers to the processor rank that produced the data. If the calculation was done single thread, `N` will be 0. If the runner option

  ```
  optionalOutputFormat  csv
  ```

  has been set, another file called *modelName*`.N.csv` will also appear equivalent to *modelName*`.N.out` but in comma separated value format.

- *modelName*`.ps`: is equivalent from the `.out` file but represented as a graph.

- *modelName*`.tex`: it is a file with the topology of the system in latex format. Just use the program `pdflatex` to obtain the formulae in pdf format. This file contains in separated sections the kinetic laws and ODEs built in function of the kinetic laws. Furthemore, if the sbml file has got user defined functions, events or rules, it contains also sections for this equations.

- `scratch` directory: this is a directory that contains necessary files for `ByoDyn` but of secondary interest for the user. Depending on the engine use to simulate the model different files will appear.

2.3.2.3 Simulation Engines There are four simulation engines available. The user can choose his prefered integrator depending of model features, in the case of SBML look table 1, and the software installed. On other hand, if the user prefer that ByoDyn choose the integration method, he could use the argument `automatic` as section 2.2.1.2 explains. Here, we describe the specific syntax to call the different engines available.

2.3.2.3.1 `SciPy` The syntax of the option `integrationMethod` should be specified as follows:

```
integrationMethod  python  default
```

As described in 2.2.1, `SciPy` allows for other integration methods additional to `default` that are `adams`, `non-stiff`, `bdf` and `stiff`.

On the `scratch` directory we could find the following files:

- *modelName*`.N.integ.py`: This is a Python script that contain the system of equations and the integration options. It creates the solution of the trajectories in *modelName*`.N.out` at the `output` directory.

- *modelName*`.gnu`: This is a Gnuplot script that contains the commands necessary to create the graph of the trajectories, *modelName*`.ps`, in the output directory.

Additionally, only with the `SciPy` option, if the second argument of `runningType`, is set to `velocity`

```
runningType  simulation  velocity
```

the calculation of the velocity of change of the concentration is computed. Two additional files will appear at the output directory:

- *modelName*`.N.veloc.out`: This is a file of the same format of *modelName*`.N.out` but referring to the velocity of change of the concentrations for each node.
- *modelName*`.veloc.ps`: This is a graphical file showing the trajectories of the velocities of the change in concentration.

Equivalent to *modelName*`.gnu`, a file named *modelName*`.veloc.gnu` will also appear in the `scratch` directory.

2.3.2.3.2 `Octave` The syntax of the option `integrationMethod` should be specified as follows:

```
integrationMethod  octave  default
```

As described in 2.2.1, `Octave` allows for other integration methods additional to `default` that are `adams`, `non-stiff`, `bdf` and `stiff`. On the `scratch` directory we could find the following files:

- *modelName*`.N.oc`: This is a Octave script that contain the system of equations and the integration options. It creates the solution of the trajectories in *modelName*`.N.data` at the `scratch` directory. Together with *modelName*`.N.time`, these files will be joined to create the file *modelName*`.N.out` at the `output` directory.
- *modelName*`.gnu`: This is a Gnuplot script that contains the commands necessary to create the graph of the trajectories, *modelName*`.ps`, in the output directory.

2.3.2.3.3 `Open Modelica` The syntax of the option `integrationMethod` should be specified as follows:

```
integrationMethod  openModelica  default
```

Some specific files produced during the model integration using `Open Modelica` will appear at the `scratch` directory:

- *modelName*`.mo`: This file contains the topology of the model in `Open Modelica` syntax.
- *modelName*`.mos`: This file contains the `Open Modelica` commands for the simulation.
- *modelName*`.cpp`: `Open Modelica` generates `C++` code from the above commands.
- *modelName*`.makefile`: A `makefile` from `Open Modelica` is generated to compile the `C++` sources.
- *modelName*`.exe`: An executable is created as a result of the `makefile` compilation.
- *modelName*`.log`: A log file resulting from the compilation is generated.
- *modelName*`.libs`: A file defining possible additional libraries for the compilation of the `C++` code.

- *modelName*_functions.cpp: A file defining possible additional functions for the compilation of the C++ code.
- *modelName*_init.txt: A text file generated by Open Modelica defining the initial value for the simulation terms.
- *modelName*_res.plt: A text file generated by Open Modelica with the values of the trajectories along time that will be plotted.
- output.log: A text file generated by Open Modelica with runtime information about the code transformation.

2.3.2.3.4 XPPAUT The syntax of the option integrationMethod should be specified as follows:

```
integrationMethod  xpp  default
```

Some specific files produced during the model integration using XPPAUT will appear at the scratch directory:

- *modelName*.N.xpp: Commands for the simulation of the model using XPPAUT syntax.
- *modelName*.XPP.log: Text file with the run time messages of XPPAUT given the -silent running option.
- *modelName*.XPP.data: Text file produced by XPPAUT containing the results of the integration of the system.

2.3.2.3.5 Others Other engines have been implemented by ourselves based on known integration methods as Euler and Runge-Kutta. These engines are less tested than previously described and some options might not be available. We ask the users to report us any lack of compatibility to fix those issues.

A specific option available while using these engines is a quick access to the trajectories plot directly from the terminal. If the optional tagName showingPlot is added to the runner file, once the simulation is done, a graphical window will open displaying the graph of the simulation. This tagName does not require further arguments.

The specific syntax to call these integration engines is:

- Euler:

  ```
  integrationMethod  euler  default
  ```

- Runge-Kutta:

  ```
  integrationMethod  rungeKutta  default
  ```

2.3.3 Stochastic Simulation

Stochastic simulation of a model consists on the numerical exact or approximated realisation of an event as described by the associated Chemical Master Equation. Exact Stochastic Simulation is carried on in `ByoDyn` by the *Stochastic Simulation Algorithm (SSA)*. Currently we have implemented the Gillespie algorithm. In addition, `ByoDyn` can generate approximated simulations by means of several explicit τ–leaping techniques.

2.3.3.1 Specific Option File Arguments Five compulsory options must appear in the file to run a stochastic simulation: `modelFile`, `modelFormat` as for the rest of simulations. However the three other fields are interpreted differently:

- `runningType argument1 argument2`

  Two values for the variable `runningType` need to be defined. The first one, `argument1`, will be `simulation`. The second one, `argument2` will always be `noVelocity`, `velocity` is not accepted from the stochastic simulators.

- `stochasticMethod argument1 argument2 argument3`

  Three arguments for the variable `runningType` need to be defined. The first one, `argument1`, defines the stochastic method to be used, this can be `ssa` for the *SSA* or `tau-leap` for the explicit one–stage τ–leap method. At the current version of `ByoDyn` only Guillespie method for *SSA* and the regular one–stage τ–leap method have been implemented. Therefore the values for the second argument should be `gillespie` or `default` for the *SSA* or the τ–leap methods respectively. We are currently working on the implementation and development of new methods that will be provided in the next releases. The last argument, `argument3`, is an integer specifying how many simulations must be run.

- `time&timestep argument1 argument2`

  Two values for the variable `time&timestep` are required. `argument1` will be a float defining the total time of the simulation. However, the `argument2` contrary to deterministic simulations, it is not a float defining the time step. For the stochastic options, as there is no properly a time step, this argument defines the resolution at which points will be plotted. For the specific case of Gillespie simulations, if it is set to zero, the exact times of every rections occuring will be provided.

Finally, some optional settings are:

- `onlyLastState`

  This option is used to store the species amounts only for the last step (once simulation time is surpassed). It is useful when analysing the stationary distribution of the system. It can also be combined with the option `separatedGraphs` explained in Section 2.3.3.1.

2.3.3.2 Specific Outputs While in general the outuput for the stochastic simulations is the same as for the deterministic integrations, there are few differences worth noting:

- *modelName*`.N.stoch.M.out`: When several stochastic runs are required, the trajectories of each `M` simulation is stored at the file named *modelName*`.N.stoch.M.out`. `N`, as described in Section 2.3.2.2, refers to the processor number. In the case the variable `onlyLastState` is used, only the value for each species at the end of the simulation is stored.

- *modelName*`.ps`: For the stochastic simulations, trajectories are shown in two differnt ways. For the case of the Gillespie method, if the second argument of `time&timestep` is set to zero, all different trajectories are plotted in the same figure. However, if the second argument of `time&timestep` is positive, the mean with its corresponding standard deviation is plotted at each printing step. That is true for both the graphs containing all nodes' trajectories and for those produced using the variable `separatedVariables` where each node is plotted separately. Finally, if the variable `onlyLastState` is used, instead of the trajectories, the histograms for the number of particles for each species at the last step of the simulation is shown. Again, the separated histograms for each species will be build if the variable `separatedGraphs` is used.

2.3.4 Optimisation

In the case we have some knowledge about the temporal values of the concentration of the node we would like to calibrate the model in order to reproduce the experimental behaviour. For model calibration, an inverse problem, we define a fitness function, which is the distance between the experimental input and the model simulation. Therefore, using minimization algorithms, we will modify the model parameters ($\boldsymbol{\theta}$) to make this function as lower as possible. At this point, the new parameter values will reproduce the experimental target.

Specifically, the fitness function is defined as:

$$F(\boldsymbol{\theta}) = \frac{1}{L}\sum_{l=1}^{L}\frac{\left(\tilde{y}_l - y_l(\boldsymbol{\theta})\right)^2}{\sigma_l^2} \tag{19}$$

where L is the number of experimental points available, $\tilde{y}_l$ are the measured experimental values, σ_l^2 is the variance of the measurement and $y_l(\boldsymbol{\theta})$ are the simulated values for the experimental points. Note that the number of experimental points is not necessary the same for each node.

We have different minimization methods to calibrate the models. These routines are quite expensive computationally and it is always a goal to find a good compromise between the time required and the quality of the solution withdrawn.

The faster approach is a random search where no optimisation is done actually but simply random positions from the parameter space are chosen and the fitness function is evaluated. For the local search, a gradient-based algorithm is used to find a local minimum either from a random starting point or from a given position. Genetic algorithms can also be used as global search methods. Finally we have combined last two approaches into two types of hybrid algorithms. Interestingly, best results are obtained from the hybrid approach (See Section 2.3.4.3.4), specifically the *Hybrid One Phase*.

2.3.4.1 Specific Option File Arguments Several arguments on the runner file are compulsory for `ByoDyn` to run an optimisation.

- `runningType argument1 argument2`

  The previously used variable `runningType` need to have two arguments, the first one will always be `parameterEstimation` and the second one will be used to select the minimization method we want to call. Possible arguments are: `randomSearch`, `localSearch`, `geneticAlgorithm`, `hybridTwoPhases`, `hybridOnePhase` or `parallelGA`[1]. We explain them extensively in Section 2.3.4.3.

- `parametersToVary argument1 argument2 ... argumentN`

  The parameters of the model allowed to be changed to minimize the fitness function has to be defined except if the `timeArgument` for `target` is defined for zero. Each of the arguments of `parametersToVary` will be a parameter of the model subject to change. The syntax of each of the arguments is the following:

  `parameterName/lowerRangeValue/upperRangeValue/scale`

  `parameterName` should be substituted by the name of the model parameter that you want to calibrate. An special mention requires local parameters on the *KineticLaw* from SBML models. Because they are allowed to take different values at different *KineticLaw*, we understand them as different parameters. Therefore, `ByoDyn` adds to the name of the local parameter the reaction name in order to distinguish them. The best approach is to run a simulation and inspect the resulting file *modelName*`.description.txt` to figure out the appropriate names of the parameters.

  `lowerRangeValue` and `upperRangeValue` should be substituted by the numerical values of the minimum and maximum values allowed respectively for the parameter to be explored. The minimization routines will explore numerical values of the parameter within this region.

  `scale` should be substituted either by `lin` or `log` to whether explore the parameter dimension in linear or logarithmic scale respectively. We recommend to use logarithmic scale if the range is wider than one or two orders of magnitude.

- `target timeArgument argument1 argument2 ... argumentN`

  The experimental information has to be provided using the runner file argument `target`. The first argument `timeArgument` should the the time point at which you target the value of the model nodes. Separated by tabular the user has to specify the value of the nodes using additional arguments with the following syntax:

  `name/cellCoordinate/value/variance`

  `name` should be the name of the model node to target. `cellCoordinate` is the index of the cell holding the node to target. For the case of a model of one cell, it should be `0,0`. Each of the values represents the index of the two dimensions of the cellular matrix. `value` should be replaced by the experimental value of the node at that time. Finally, if the experimental measurement has a variance associated to it, it should be inserted in the `variance` field. In the case that no variance is available, set this value to 1.

- `stopper argument1 argument2`

[1] This option is a developer argument used for the use of `PGAPack` library which is under current development.

The minimization algorithm needs to be constrained by the user to stop at a certain point using the runner file argument of `stopper`. Two variants are allowed, to stop the algorithm based on the number of iterations or the value of the fitness function:

- *Iterations:* Set `argument1` to `iteration` and `argument2` to an integer defining the number of iterations required.
- *Fitness Function Value:* Set `argument1` to `score` and `argument2` to the value of the fitness function at which `ByoDyn` should stop.
- *Number of Simulations:* Set `argument1` to `numberOfSimulations` and `argument2` to the maximum number of simulations allowed during the model calibration. The number of function evaluations is measured at each step of the minimization methods, therefore, methods that involve multiple simulations for each step of the minimization routines as the genetic algorithm or the local search, it will be approximated by excess. Same holds for the hybrid methods.

Other general arguments are possible for all minimization methods although they are not compulsory, they are optional arguments:

- `parameter argument1 argument2 ...`

  It can be used the same way as described in 2.2.1.2 and it can be used to start the minimisation from an specific point on the parameter space.

2.3.4.2 Specific Ouputs In the output directory of `ByoDyn` you will find the following files common to any optimisation:

- *modelName*`.N.out`: This file holds the the concentration of each node (in columns) along time (first column) equivalently to the one described in Section 2.3.2.2. It refers to the last function evaluation from the minimization algorithm.
- `modelName` directory: This directory will contain the solutions from the model calibration. Depending on the algorithm used to calibrate the model, different files and directories will be found although the structure will be consistent. Please check Section 2.3.4.3 for more detailed information.
- `scratch` directory: This is a directory that contains necessary files for `ByoDyn` but of lower interest for the user. Depending on the engine use to simulate the model and the minimization algorithm, different files will appear.

2.3.4.3 Minimization Methods Several methods are available for model calibration.

2.3.4.3.1 Random Search This is not an actual method of minimization but random points of the parameter space are chosen and the fitness function is evaluated.

In order to direct the flow of the program to this function the user has to define the `runningType` as follows:

```
runningType  parameterEstimation  randomSearch
```

In the directory `modelName` two new structures can be found:

- *modelName*`.xml`: This file represents the best solution from the minimization in SBML format.

- `parametersFromRandom` directory: This directory contains a file for each of the parameters made free during the parameter estimation. Each of the lines of the files represents a parameter space point that has been evaluated during the algorithm flow. The structure of each of the files is the following:

  `parameterValue lowerRangeValue upperRangeValue fitnessFunctionValue`

2.3.4.3.2 Local Search It has been reported [Rodriguez-Fernandez et al., 2006] that one of the best gradient based methods to calibrate biochemical systems is `dn2fb` routine from `PORT` library. We have implemented a call to that library for that purpose.

The user can access this functionality by setting the runner file argument `runningType` as follows: `runningType parameterEstimation randomSearch`

Specific outputs intrinsic from this type of run will be located in two places:

- `modelName` directory: Apart from *modelName*`.xml` and the directory `parametersFromRandom` another directory will appear `parametersFromLocalSearch`. Again *modelName*`.xml` will be an SBML file with the model parameters which gave the best match with the experimental data set and the directory `parametersFromRandom` will hold the files corresponding to the starting points of the optimisation (check 2.3.4.3.1 for further information about the contents and structure of the directory). `parametersFromLocalSearch` holds the same structure as `parametersFromRandom` but it contains the results of the local search.

- `scratch` directory: In the case of the local search, the `scratch` directory is used to compile the Fortran routines and convert the Fortran code into python readable modules.

  - `localSearch.f`: This file is the Fortran source that calls `dn2fb` routine from `PORT` library.
  - `auxVar.dc`: This text file contains the value of specific variables necessary for the Fortran routines that change depending on the model number of variables and parameters. It is imported by `localSearch.f`.
  - `makefile`: This makefile compiles the Fortran sources and convert them into a python importable module.
  - `compilationMessages`: This file collects the messages of the compilation.
  - `localSearch.so`: Lastly this module is the result of the makefile which will be imported by `ByoDyn`.

2.3.4.3.3 Genetic Algorithm We have implemented a genetic algorithm based on Goldberg [1989]. This option arguments are sufficient to launch the genetic algorithm optimisation, however some algorithm tuning parameters can be modified using the following structure:

`gaOptions tag/value`

- Population size: `tag` should be substituted by `populationSize`. `value` is set by default to 10 although any integer higher than three can be used. In the case of odd numbers the population might be adjusted to the immediate lower even number. At any case we suggest to set a population size larger than 3.
- Translocation rate: `tag` should be substituted by `translocationRate`. `value` is set by default to 0.8 although any value from 0 to 1 can be used.
- Mutation rate: `tag` should be substituted by `mutationRate`. `value` is set by default to 0.3 although any value from 0 to 1 can be used.

The user can access this functionality by setting the runner file argument `runningType` as follows:

```
runningType  parameterEstimation  geneticAlgorithm
```

Specific outputs intrinsic to this type of run will be located in two places within the output directory:

- *modelName*`.st`: a file called `modelName.st` where the statistics of the genetic algorithm iterations are stored. The structure is the following:

  ```
  Generation  n  meanValue  bestValue
  ```

  For each of the generations, the generation number is stored in `n`, the mean value of the fitness value of the population is stored next, at `meanValue` and finally the fitness value of the best element is stored at `bestValue`.
- in the `modelName` directory: a directory called `parameterFromGA` will be created. It will contain the several files named as the parameter subject to calibration, containing the best individual. Furthermore an SBML file containing the best combination of parameters will be stored in the directory `modelName`.

2.3.4.3.4 Hybrid Algorithm We understand a hybrid algorithm as a mixture between a two algorithms of different strategies. In our case, the hybrid algorithm will consist on the combination of a global stochastic search (a genetic algorithm) and a gradient based search (a local search). There are two types of hybrid algorithms implemented in `ByoDyn`:

- *Hybrid Two Phases:* We ran first a genetic algorithm until the stopping criterion is accomplished, either score or iteration. For the case of the score criterion, all elements of the current iteration that are below the threshold are selected for a local search. Results are stored in two different directories of `modelName` from the output directory named `parametersFromGA` and `parametersFromHybridOnePhase`. Within the last one, only the best fitted element is saved.

  The syntax for `runningType` is the following:

  ```
  runningType  parameterEstimation  hybridTwoPhases
  ```

  If the `stopper` is based on the score, the score will be evaluated for the genetic algorithm. The same is true for the `numberOfSimulations`, which will be approximated.

- *Hybrid One Phase:* In this case the degree of mixture is much larger. For each of the elements of the genetic algorithm we launch a local search optimization. The resulting values of score and parameters are the ones used as actual values of the elements of that generation. For the obtention of the next generation, the same procedures as for the genetic algorithm are taken. The best fitted element of the optimisation is stored in directory called `parametersFromHybridOnePhase`.

  The syntax for `runningType` is the following:

  ```
  runningType  parameterEstimation  hybridOnePhase
  ```

We want to note some differences for this running type.

- If the `stopper` is based on the score, the score will be evaluated at each of the generations of the genetic algorithm. The same is true for the `numberOfSimulations`, which will be approximated.
- Another difference is the structure of the file *modelName*`.st` stored in the output directory. In this case, for each generation, the fitness value of all elements of the population are stored.
- Within the directory `modelName` a directory will be created named `parametersFromHybridOnePhase`. Several files named as the parameters made free for the calibration will be found. However, differing from Section 2.3.4.3.3 only the best combination of parameters is stored.

From our experience, the hybrid algorithm has been the best performing algorithm, specially the hybrid one phase. However the running time is quite long and we recommend it for complicate minimisation problems. For short and easy model calibrations, the local search algorithm might be a better choice.

2.3.5 Fitness Function Calculation

In some cases we want to evaluate how far is a given model from the experimental data. In that case, a runner similar to a simulation runner should be created, adding the corresponding experimental concentrations to evaluate as in Section 2.3.4. To be precise, the runner file should contain as arguments: `modelFile`, `modelFormat`, `integrationMethod`, `runningType`, `time&timestep` and `target`. `runningType` should be defined as follows:

```
runningType  calculateFunction
```

holding the unique additional value.

No specific output files are generated but simply the fitness function of the system (model *plus* target data) is displayed on the screen.

Just note that the argument `parameter` can always be used to specify the value of some of the model parameters without need to edit the SBML file. The parameter value used to evaluate the fitness function will be the one specified on runner file.

2.3.6 Trajectories Reconstruction

Once we have run an optimisation and we have collected a bunch of parameter values we feel comfortable with, we want to see the trajectories of the new models. For that specific purpose

you should create a runner file with the defined common fields of `modelFile`, `modelFormat`, `integrationMethod` and `time&timestep`. Furthermore the user has to set the `runningType` to `dynamicsReconstruction`:

```
runningType  dynamicsReconstruction
```

This tag will direct the flow of `ByoDyn` to take the new parameter values, modify the model and run the simulations of the new defined models. Finally the directory with the solutions has to be specified using the tag `solutionsDirectory`:

```
solutionsDirectory  /path/to/parameters/files
```

Typically the files containing the parameter solutions were obtained from an optimisation named as one of the `parametersToVary` and with the following syntax:

```
#
# generated by ByoDyn version 4.0
#
parameterValue lowerRange upperRange fitnessValue
...                ...                ...                ...
...                ...                ...                ...
```

Those files are automatically parsed when you call `ByoDyn` with a runner file as argument containing the variable `runningType` selected for `dynamicsReconstruction`.

The result is a set of pdf files stored in a specific directory called `reconstructedDynamics` inside the output directory, one for each node of the system, showing the trajectories for all the parameter solutions. Moreover, if the targets are appended (they are not required), they will be plotted too.

If you want to highlight a given trajectory, the parameter values combination that is on first position on the parameter solution files will be plotted using black thick lines. This option could be useful, for example, by setting the first value of the parameter files to be the ones of the original model. Therefore it can be compared the original model behaviour with the trajectories of the new parameters.

Finally, the numerical values of the trajectories are stored in *.sd* files on the scratch directory.

2.3.7 Fitness Function Surfaces

Another type of analysis that can be performed from a given solution of the optimisation analysis is the calculation of the fitness in the close vicinity of a parameter space point. Selecting pairs of model parameters, `ByoDyn` will construct a fitness value surface at the parameter range. This result is achieved by constructing a grid of points on the selected parameter space and evaluating the fitness (Eq. 19) at each point.

2.3.7.1 Specific Option File Arguments The generally compulsory fields of `modelFile`, `modelFormat`, `integrationMethod` and `time&timestep` are also required for this functionality. Moreover three more arguments are required:

- `runningType  argument1  argument2  ...  argumentN`

The first argument of `runningType` should always be `scoreSurface`. The second argument will be an integer representing the resolution of the surface: it indicates the number of equidistant points at which the fitness function is evaluated for each parameter component. From the second to N arguments, they determine the pair of model parameters for which the surface is calculated. The syntax for these arguments will be `parameterA/parameterB`. A surface will be calculated for each of the parameter pair combination selected.

- `parametersToVary` and `target` are also required as described in Section 2.3.4.1.

2.3.7.2 Specific Outputs Some files derived from the simulation are *modelName*.`.N.out` in `output` directory and *modelName*.`N.integ.py` in `output/scratch` directory. Moreover, the specific files of interest are:

- *parameterA/parameterB*.`.txt`: This text file contains the fitness function values of the grid.
- *parameterA/parameterB*.`.ps`: This postscript figure is the graphical representation of the previous file. In color code the value of the fitness function is represented on the model parameter space. A scale and each of the parameter ranges is also depicted.

A text file and a postscript figure will be created for each of the parameter combinations selected on the runner file.

2.3.8 Sensitivity Analysis

The sensitivity analysis consists on the calculation of the variation of the output of the model with respect to some source of variation. In our case we evaluate the variation on the trajectories of k nodes with an infinitesimal change on the value of p model parameters.

$$S(t) = \frac{\partial \mathbf{y(t)}}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial y_1(t)}{\partial \theta_1} & \frac{\partial y_1(t)}{\partial \theta_2} & \cdots & \frac{\partial y_1(t)}{\partial \theta_p} \\ \frac{\partial y_2(t)}{\partial \theta_1} & \frac{\partial y_2(t)}{\partial \theta_2} & \cdots & \frac{\partial y_2(t)}{\partial \theta_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_N(t)}{\partial \theta_1} & \frac{\partial y_N(t)}{\partial \theta_2} & \cdots & \frac{\partial y_N(t)}{\partial \theta_p} \end{pmatrix} \tag{20}$$

For each node i and each parameter j the sensitivity along time would be,

$$S_{i,j}(t) = \frac{\partial y_i(t)}{\partial \theta_j} = \lim_{h \to \infty} \frac{y_i(\theta_j + h, t) - y_i(\theta_j, t)}{h} \tag{21}$$

we normalize the values of the sensitivity,

$$\Sigma_{i,j}(t) = \frac{\theta_j}{y_i(t)} \, S_{i,j}(t) \tag{22}$$

2.3.8.1 Specific Option File Arguments Apart from the compulsory arguments of `modelFile`, `modelFormat`, `integrationMethod` and `time&timestep`, other arguments are necessary:

- `runningType sensitivityAnalysis`

  The argument `runningType` is also compulsory but its value should be set to `sensitivityAnalysis`. In this case, on the contrary of other functionalities, only a single value is required.

- `sensitivityParameters argument1 argument2 ...`

  This argument is required to select the name of the parameters for which the sensitivity will be calculated. Each of the `argument` should be substituted by the name of the model parameters of interest.

2.3.8.2 Specific Outputs Within the output directory, two files named *modelName*`.N.out` and *modelName*`.ps` are created as for the simulation. Moreover other specific files from the sensitivity analysis are created in the output directory:

- *modelName*`.sens.global.timeCourse.ps`: Once calculated $\Sigma_{i,j}(t)$, the overall sensitivity for each of the parameters is calculated,

$$\bar{\Sigma}_j(t) = \frac{1}{k}\sum_{i=1}^{k}\sqrt{\Sigma_{i,j}^2(t)} \tag{23}$$

  and plotted for each parameter j in this file.

- *modelName*`.sens.global.txt`: In order to have a single value along time, we compute the global sensitivity for each parameter, $\langle\bar{\Sigma_j}\rangle$, defined as follows,

$$\langle\bar{\Sigma_j}\rangle = \frac{1}{t_T}\sum_{t=0}^{t=t_T}\bar{\Sigma}_j(t) \tag{24}$$

  Their values are shown in this text file.

- *modelName*`.sens.relative.txt`: From $\Sigma_{i,j}(t)$ in Eq. 22, we can obtain a single value along the time, $\langle\Sigma_{i,j}\rangle$ defined as follows:

$$\langle\Sigma_{i,j}\rangle = \frac{1}{t_T}\sum_{t=0}^{t=t_T}\sqrt{\Sigma_{i,j}^2(t)} \tag{25}$$

  Each value of the matrix is saved in this text file.

- *modelName*`.sens.relative.ps`: This file is the graphical representation of *modelName*`.sens.relative.`

- `scratch`: In the scratch directory, apart from the files *modelName*`.gnu` and *modelName*`.0.integ.py` characteristic from the simulation (See Section 2.3.2), we have several files specific from the sensitivity analysis:

  - *modelName*`.sens.global.gnu`: This is a Gnuplot script that contains the commands necessary to create the graph of the trajectories, *modelName*`.sens.global.timeCourse.ps`, in the output directory.
  - *modelName*`.sens.global.out`: This is a file of the same format of *modelName*`.N.out` but referring to the overall sensitivity described in Eq. 23. These data will be used to create the figure *modelName*`.sens.global.timeCourse.ps`.
  - *modelName.parameterName*`.out`: For each of the perturbed parameters, the calculation of the output of the system has to be done. It is stored in these files, with an equivalent structure to *modelName*`.N.out`

2.3.9 Identifiability Analysis

Identifiability analysis is a major help for parameter estimation. It uses sensitivity-based variables to determine local topological difficulties on the fitness function minimisation:

- parameters whose change affects minimally the output of the system.
- parameter correlations that which make different parameter combinations yield the same system output.

Quantification of these features provides the essential bricks of information necessary for optimal experimental design (OED). OED is a technique that points us which measurements are more useful to make the parameter estimation problems more tractable. More information is available at Section 2.3.10.

The development of our code is based on Rodriguez-Fernandez et al. [2006], Yue et al. [2006]. However we describe here precisely the identifiability-related variable that `ByoDyn` calculates.

The first main pillar on which the identifiability analyses are based is the Fisher information matrix (FIM). The FIM is defined as the variance of the score and topologically it gives an idea of the sharpness of the local well of the fitness function F, defined on Eq. 19.

$$FIM \equiv var(U) = \mathbb{E}\left[\left(\frac{\partial}{\partial \boldsymbol{\theta}} lnL(\boldsymbol{\theta};\boldsymbol{y})\right)^2\right] \tag{26}$$

With some calculus we can show that

$$\mathbb{E}\left[\left(\frac{\partial}{\partial \boldsymbol{\theta}} lnL(\boldsymbol{\theta};\boldsymbol{y})\right)^2\right] = \int_{-\infty}^{\infty} \frac{\left(\frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta};\boldsymbol{y})\right)^2}{L(\boldsymbol{\theta};\boldsymbol{y})}\, dx \tag{27}$$

and

$$\mathbb{E}\left[\frac{\partial^2}{\partial \boldsymbol{\theta}^2} lnL(\boldsymbol{\theta};\boldsymbol{y})\right] = \int_{-\infty}^{\infty} \frac{\partial^2}{\partial \boldsymbol{\theta}^2} L(\boldsymbol{\theta};\boldsymbol{y})\, dx - \int_{-\infty}^{\infty} \frac{\left(\frac{\partial}{\partial \boldsymbol{\theta}} lnL(\boldsymbol{\theta};\boldsymbol{y})\right)^2}{L(\boldsymbol{\theta};\boldsymbol{y})}\, dx \tag{28}$$

Assuming the condition,

$$\int_{-\infty}^{\infty} \frac{\partial^2}{\partial \boldsymbol{\theta}^2} L(\boldsymbol{\theta};\boldsymbol{y}) dx = 0 \tag{29}$$

which is true for the multivariate normal distribution $\boldsymbol{y}$, the FIM can be written as,

$$FIM = -\mathbb{E}\left[\frac{\partial^2}{\partial \boldsymbol{\theta}^2} lnL(\boldsymbol{\theta};\boldsymbol{y})\right] \tag{30}$$

Again as the variable of measurement is a multivariate normal distribution, the likelihood function is defined as,

$$L(\boldsymbol{\theta};\boldsymbol{y}) = \frac{1}{2\pi^{\frac{NK}{2}} \sqrt{\prod_{i=1}^{N}\prod_{k=1}^{K} \sigma_{i,k}^2}} e^{-\frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K}\left(\frac{\tilde{y}_{i,k} - y_{i,k}(\boldsymbol{\theta})}{\sigma_{i,k}}\right)^2} \tag{31}$$

We prepare Eq. in order to insert it on Eq. 30, we calculate the ln of the likelihood,

$$-ln\ L(\boldsymbol{\theta};\boldsymbol{y}) = \frac{NK}{2} + \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K} ln\ \sigma_{i,k}^2 + \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K}\left(\frac{\tilde{y}_{i,k} - y_{i,k}(\boldsymbol{\theta})}{\sigma_{i,k}}\right)^2 \tag{32}$$

which can be reduced to

$$-ln\ L(\boldsymbol{\theta};\boldsymbol{y}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K}\left(\frac{\tilde{y}_{i,k} - y_{i,k}(\boldsymbol{\theta})}{\sigma_{i,k}}\right)^2 \tag{33}$$

At this point we can insert Eq. 33 into Eq. 30,

$$FIM = \frac{\partial^2}{\partial \boldsymbol{\theta}^2}\ \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K}\left(\frac{\tilde{y}_{i,k} - y_{i,k}(\boldsymbol{\theta})}{\sigma_{i,k}}\right)^2 \tag{34}$$

which can be defined as

$$FIM = \tfrac{\partial^2 J}{\partial \boldsymbol{\theta}^2}, \quad \text{if} \quad J = \tfrac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K}\left(\frac{\tilde{y}_{i,k} - y_{i,k}(\boldsymbol{\theta})}{\sigma_{i,k}}\right)^2 \tag{35}$$

At this point we calculate the gradient of J with respect to the parameters, $\nabla_{\theta_j} J$,

$$\nabla_{\theta_j} J = \frac{\partial J}{\partial \theta_j} = -\sum_{i=1}^{N}\sum_{k=1}^{K}\frac{1}{\sigma_{i,k}^2} r_{i,k}\frac{\partial y_{i,k}(\boldsymbol{\theta})}{\partial \theta_j} = -\sum_{i=1}^{N}\sum_{k=1}^{K}\frac{1}{\sigma_{i,k}^2} r_{i,k} s_{i,j,k} \tag{36}$$

in which $r_{i,k} = \tilde{y}_{i,k} - y_{i,k}(\boldsymbol{\theta})$. The second derivative will be the Hessian matrix,

$$H(j,u) = \frac{\partial^2 J}{\partial \theta_j \partial \theta_u} = \sum_{i=1}^{N}\sum_{k=1}^{K}\frac{1}{\sigma_{i,k}^2} s_{i,j,k}\ s_{i,u,k} - \sum_{i=1}^{N}\sum_{k=1}^{K} r_{i,k}\frac{1}{\sigma_{i,k}^2}\frac{\partial s_{i,j,k}}{\partial \theta_u} \tag{37}$$

If the residuals are small, the second term can be neglected and the Hessian matrix is approximated by

$$H(j,u) = \sum_{i=1}^{N}\sum_{k=1}^{K}\frac{1}{\sigma_{i,k}^2} s_{i,j,k}\ s_{i,u,k} \tag{38}$$

At this point, we are summing up sensitivities of different nodes and therefore we understand that it is more appropriate to use normalised sensitivities $\Sigma_{i,j}$ instead of $s_{i,j}$ as in Eq. 22.

$$FIM = \sum_{i=1}^{N}\sum_{k=1}^{K}\frac{1}{\sigma_{i,k}^2}\Sigma_{i,j,k}^T \Sigma_{i,u,k} \tag{39}$$

Other measures of interest is the covariance matrix which is the inverse of the FIM,

$$cov = FIM^{-1} \tag{40}$$

and the correlation matrix,

$$\rho_{j.u} = \frac{cov_{j,u}}{\sqrt{cov_{j,j}\ cov_{u,u}}} \tag{41}$$

Finally, OED techiniques, as we introduced above, use information from identifiability measurements to make the optimisation problem much more tractable. This is achieved modifying the shape of the fitness function surface in such a way that the optimisation methods outperform the previous situation. Basically correlations are avoided and Hessian components are made more homogeneous. Different criteria can be used to improve the FIM [Rodriguez-Fernandez et al., 2006]:

- modified A-optimal criteria: max tr(FIM), which represents minimising the arithmetic mean of the identification errors.
- D-optimal criteria: max det(FIM), which represents minimising the geometric mean of the identification errors. Topologically means minimising the asymptotic confidence ellipsoids volume.
- E-optimal criteria: max $\lambda_{\min}(FIM)$, which represents minimising the largest idetifiability error. We search for minimising the largest dispersion.
- modified E-optimal criteria: min $\frac{\lambda_{\max}(FIM)}{\lambda_{\min}(FIM)}$, which represents minimising the ratio of the largest to the smallest dispersion axes. Topologically means to make the ellipsoid volumes as spherical as possible.

2.3.9.1 Specific Option File Arguments The generally compulsory fields of `modelFile`, `modelFormat`, `integrationMethod` and `time&timestep` are also required for this functionality. Moreover three more arguments are required:

- `runningType identifiabilityAnalysis`

  The argument `runningType` is also required for all other functionalities but in this case its value should be set to `identifiabilityAnalysis`.

- `sensitivityParameters argument1 argument2 ...`

  This argument is required with the same structure and meaning as in Sec. 2.3.8.1.

- `target timeArgument argument1 argument2 ... argumentN`

  Used the same way as described in Sec. 2.3.4.1.

Additionally, the tag

`identifiabilityCriteria argument1 argument2 ...`

can be used as described in Section 2.3.10 to restrict the analysis of the identifiability to only the specified criteria.

Finally, as in previous section, note that the argument `parameter` can always be used to specify the value of some of the model parameters without need to edit the SBML file.

2.3.9.2 Specific Outputs As previously described, files characteristic from the simulation arise on the output directory, *modelName*`.N.out` and *modelName*`.ps`. Moreover, in the output directory of `ByoDyn` you will find the following files specific from the identifiability analysis:

- *modelName*`.FIM.txt`: This is a text file containing the numerical values of the FIM described in Eq. 39.
- *modelName*`.COV.txt`: This is a text file containing the numerical values of *cov* described in Eq. 40.
- *modelName*`.correlation.txt`: This a text file containing the numerical values of the parameter correlation matrix, $\rho_{j.u}$, defined in Eq. 41.
- *modelName*`.correlation.ps`: This is a postscript file, a figure representing the data contained in *modelName*`.correlation.txt`. Red colour is used for positive correlation, blue for negative. The level of intensity is referred to the grade of the correlation. Please note that the diagonal of the matrix ($\rho_{j.u}$, defined in Eq. 41), which will be 1, it is the maximum value possible and therefore it will be coloured as pure red.
- *modelName*`.criteria.txt`: This is a text file with the numerical values of the different criteria used for OED: modified A, D, E and modified E-optimal design criteria.
- `scratch` directory: In the scratch directory, apart from the files *modelName*`.gnu` and *modelName*`.0.integ.py` characteristic from the simulation (See Section 2.3.2), several files named *modelName*.`parameterName.out` are created with the same structure as the ones described in the sensitivity analysis (See Section 2.3.8).

2.3.10 Optimal Experimental Design

OED is used to find which is the most informative constrain for a successful model calibration. In our context that means to find which is the new temporal experimental data that improves one of the selected criteria explained on Section .

2.3.10.1 Specific Option File Arguments The generally compulsory fields of `modelFile`, `modelFormat`, `integrationMethod` and `time&timestep` are also required for this functionality. Moreover six more arguments are required:

- `runningType optimalExperimentalDesign addNewPoint`

  The argument `runningType` is also required for all other functionalities but in this case its value should be set to `optimalExperimentalDesign` and a second argument `addNewPoint` is required too.

- `sensitivityParameters argument1 argument2 ... argumentN`

  This argument is required with the same structure and meaning as in Section 2.3.8.1.

- `target timeArgument argument1 argument2 ... argumentN`

  Also this required argument, it will be used as described in Section 2.3.4.

- `identifiabilityCriteria argument1 argument2 ... argumentN`

  This variable define which criteria is used to select the most informative experimental point.

If more than one criteria is asked, several arguments can be used. The possible values for the arguments are `MA`, `D`, `E` or `ME` for modified A, D, E or modified E-optimal experimental design criteria.

- `OEDResolution argument`

  The variable `OEDResolution` is used to define the precision of the sampling points for optimal experimental design. The only accepted argument will be an integer defining the rate of precision compared with the simulation, that means that if `argument` is set to 1, the analysis of the identifiability will be calculated at each time point of the simulation, otherwise, if it is set, for example, to 10, the identifiability analysis will be calculated only once every ten simulation steps.

- `targetSpecies argument1 argument2 ... argumentN`

  The model node names for which the OED is evaluated have to be stated.

Briefly note that other common variables to all functionalities as `parameters` are also valid for OED.

2.3.10.2 Specific Outputs Within the output directory, two files named *modelName*`.N.out` and *modelName*`.ps` are created as for the simulation. Moreover other specific files from the sensitivity analysis are created in the output directory:

- *modelName*`.oed.txt`: This is a text file where the most optimal values are stored. For each required criteria the best temporal point for each selected model species is saved.

- *modelName*`.oed.`*criteria*`.ps`: For each OED criteria a postscript figure is created showing the value of the corresponding criteria for each selected node and time.

In the scratch directory, apart from the files *modelName*`.gnu` and *modelName*`.0.integ.py` characteristic from the simulation (See Section 2.3.2), we have too several files *modelName.parameterName*`.out` characteristic of the sensitivity analysis (See Section 2.3.8) that correspond to the perturbed model output, stored while sensitivity was numerically calculated.

3 `ByoDyn` on Parallel Environments

`ByoDyn` can run in a parallel environment. This option is not explicitly defined as a command within the program, it should be set according to the hardware available. Running `ByoDyn` in parallel have the advantage of reducing the amount of time, and/or increasing the amount of operations as much as processors we may have available.

Specifically, we have implemented the parallelization of the genetic algorithm, improving the time of the optimisations. When the user runs a genetic algorithm in `ByoDyn` in a parallel environment, the individuals of the population will be balanced in all the processors trying to decrease the total wall time of the process. At an ideal situation where we have n processors for n individuals, `ByoDyn` will run each individual on each processor. Whereas if the number of processors available is the half, `ByoDyn` will run two individuals on each processor. For a better performance we recommend to set the number of individuals as a multiple of the total number of processors (i.e. 16

individuals with 2, 4, 8, 16 processors). Other combinations are valid, although we may have *lazy* processors at some point.

3.1 Performance

The speed up of the genetic algorithm time will depend on the number of processors, basically. In general the speed-up is multiplied as much as processors we have, i.e. with a fixed number of individuals from 1 to 2 processors, we will observe an improvement of almost 2 times; from 1 to 4 processors, an improvement of nearly 4 times. The speed-up is not perfectly linear as some running time is dedicated for the communication between processors (overhead), as well as the extra effort which is dedicated to split up and summarize the information from the different processors.

A relative short time in the simulation of each of the individuals represents a bottleneck for the speed-up. Assuming a population of n individuals of a relative big size, distributed in a parallel machine of n CPUs. We define the processing time of each individual as T. On the other hand we also have an small time τ dedicated to communication processes. In the case $T \approx \tau$ the speed-up at high number of processors will be very poor. For example, the speed-up from 1 to 2 processors the speed-up will be 2 if we assume that $nT \gg \tau$:

$$\text{speed} - \text{up} = \frac{t_{1\text{CPU}}}{t_{2\text{CPUs}}} = \frac{nT + \tau}{\frac{n}{2}T + \tau} \tag{42}$$

Assuming $nT \gg \tau$ because the number of processors is high, we simplify as,

$$\text{speed} - \text{up} = \frac{nT}{\frac{n}{2}T} = 2 \tag{43}$$

which is a linear speed-up. On the contrary, the speed-up will be poor if we use many processors:

$$\text{speed} - \text{up} = \frac{t_{50\text{CPU}}}{t_{100\text{CPUs}}} = \frac{2T + \tau}{T + \tau} \tag{44}$$

$T \approx \tau$,

$$\text{speed} - \text{up} = \frac{3T}{2T} = \frac{3}{2} \tag{45}$$

The decrease of the speed-up, from a theoretical 2 to 3/2 does not seem so bad in principle. But, we assumed a constant τ for a communication time from $n/2$ to n processors, which is not true for a real system. In fact, the communication time τ from $n/2$ to n could perfectly be 2τ as the number of instructions is doubled. If we take that into account, the speed-up of the last example lowers to 1:

$$\text{speed} - \text{up} = \frac{t_{50\text{CPU}}}{t_{100\text{CPUs}}} = \frac{2T + \tau}{T + 2\tau} = 1 \tag{46}$$

Finally we need to take into consideration the fact that increasing the population number n, the variability of the population increases, widening the time differences between the *fast* and *slow* integration models. Therefore the *communication* time will increase dramatically and the speed-up will be concomitantly affected.

3.2 External Software Requirements

An implementation of the MPI (Message Passage Interface) API (application programming interface) is required. Although any available should do work fine, we used OpenMPI (www.open-mpi.org). The resulting binaries will be the `mpirun` program and by default those implementations have *wrappers* compilers for some languages like `mpicc`, for C, `mpiCC` for C++ and general Fortran implementations. Keep in mind that depending on the implementation of MPI you chose, you will be constrained to use it with other software. In our case, we require an implementation of the MPI for Python. By default, most of the MPI API's do not include this *wrapper*, although, once we have installed the `mpirun` executable in our computer, we can use the MPI Python *wrapper* of the Python library `ScientificPython`. The library includes a *wrapper* Python for MPI. Please refer to the Installation Guide of `ByoDyn` for further details about how to create the *wrapper*. For the specific case of the `ScientificPython` *wrapper*, we will need the `mpicc` in order to be able to compile the MPI module for Python.

3.3 Launching `ByoDyn` in Parallel

Once all requirements have been successfully installed, you should execute `ByoDyn` as follows:

```
mpirun <mpirun options> mpipython byodyn <byodyn options>
```

So for example to run a simple case of `ByoDyn` with 2 processors and the `ByoDyn` running options `optimisation.rn` as input, we will type:

```
mpirun -np 2 mpipython byodyn optimisation.rn
```

We need to have in mind that all the settings like alias and paths must be already fixed, in order to be able to run `ByoDyn` in parallel without any problem. We also recommend to run a genetic algorithm or a hybrid one phase (See Section 2.3.4.3.3 and 2.3.4.3.4) given the fact that the parallel routine is the genetic algorithm. Therefore, we recommend a command like:

```
mpirun -np 2 /usr/bin/mpipython /home/user/software/bin/byodyn
/home/user/project/optimisationg.rn
```

3.4 QosCosGrid

QosCosGrid (http://www.qoscosgrid.eu/) is a group effort to enable parallel execution of complex systems tools on quasi-opportunistic grids. Currently on testing status, `ByoDyn` has been able to run on cross-cluster environments taking profit of the QosCosGrid services [Coti et al., 2008].

4 Acknowledgements

The `ByoDyn` team wants to acknowledge the constant help of Michael Johnston on computer problems during the development of the software. We want to acknowledge Mike Hucka and Ben Bornstein for helping on the support of SBML. Also Miquel de Cáceres, David Sportouch and Elisenda Feliu for helpful discussions and help. Finally Pau Rué for reporting bugs.

References

C. Coti, T. Herault, S. Peyronnet, A. Rezmerita, and F. Cappello. Grid Services for MPI. *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08)*, pages Lyon, Fra, 2008.

A. L. García de Lomana, À. Gómez-Garrido, D. Sportouch, and J. Villà-Freixa. Optimal Experimental Design in the Modelling of Pattern Formation. *LNCS*, accepted, 2008.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.

M Hucka, A Finney, HM Sauro, H Bolouri, JC Doyle, H Kitano, and the rest of the SBML forum. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. 19(4):524–31.524–31, 2003.

N. Le Novère, B. Bornstein, A. Broicher, M. Courtot, M. Donizelli, H. Dharuri, L. Li, H. Sauro, M. Schilstra, B. Shapiro, J. L. Snoep, and M. Hucka. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular models. *Nucleic Acids Research*, 34:D689–D691, 2006.

M. Rodriguez-Fernandez, P. Mendes, and J. R. Banga. A hybrid approach for efficient and robust parameter estimation in biochemical pathways. *Biosystems*, 83:248–65, 2006.

H. Yue, M. Brown, J. Knowles, H. Wang, D.S. Broomhead, and D.B. Kell. Insights into the behaviour of systems biology models from dynamic sensitivity and identifiability analysis: a case study of an NF-κB signalling pathway. *Mol Biosyst*, 2:640–649, 2006.