Journal of
open research software

Nedelec, F 2017 preconfig: A Versatile Configuration File Generator for Varying Parameters.
Journal of Open Research Software, 5: 9, DOI: https://doi.org/10.5334/jors.156

SOFTWARE METAPAPER

preconfig: A Versatile Configuration File Generator for Varying Parameters

Francois Nedelec
Cell Biology and Biophysics Unit, European Molecular Biology Laboratory, 69117 Heidelberg, DE
nedelec@embl.de

Mathematical models in biology have many parameters, and sampling parameter values is necessary, for instance to identify robustness or sensitivities in the model, or simply calculate a phase diagram. We provide a universal program: Preconfig, that can be used within different computing pipelines to sample parameters. From a template file, Preconfig will generate multiple configuration files, covering all the combinations of parameter values specified as inclusions in the template. Fixed ranges or random sampling can be combined, and non-linear or interdependent variations are easily specified. Preconfig is open source and available on GitHub at https://github.com/nedelec/preconfig.

Keywords: Systems Biology; Mathematical Modelling; Parameter Sampling
Funding Statement: This work has been supported by the European Molecular Biology Laboratory.

(1) Overview

Introduction

Varying parameters is a common task in bioinformatics, particularly for someone performing computer modelling, but also for other kinds of computer based processing. Many applications are built in such a way that the parameters can be specified in a configuration file. Command line tools often expect their parameters to be specified directly as command-line arguments. It is however always possible to write a shell script to run the program, and this script where all the arguments are specified is then *de facto* a configuration file. This approach has the advantage that all the parameters of the calculation are safeguarded. We will thus focus on the case where parameter values are contained in a file. Several approaches are commonly used to vary the values. The first one is to manually edit the file, and save many copies as desired. This is of course simple but error prone and does not scale well. For example, generating all the combinations corresponding to $N = 0, 1, 2, 3, 4$; $D = 1, 10, 100, 1000$ and $v = -1, 0, 1$ would require editing 60 files.

A more powerful approach is to write a small program or a script to automate the generation of multiple configurations files. Following this approach, one would typically use loops to vary parameter values as desired, calling repeatedly some function to generate the output files. For a good programmer, this is a relatively easy task, especially using a language with powerful text manipulating functions, such as Python or Ruby. However, in addition to requiring programming skills, the disadvantage of this method is that such a script will necessarily need to intermingle two types of instructions, to vary the different parameters and to generate output text files. As a consequence, such scripts are hard to read, and it may not be immediately visible what parameters were changed and in what way, especially for someone other than the programmer. In our experience, this approach is hard to maintain and unsuited for distribution. Moreover, the script has to be edited every time the parameter variations have to be changed. In addition, the script is typically tailored to the particular tool that will read the configuration file, such that a new script would be needed for every new tool, or new version of the tool. In practice, this approach leads to a lot of duplicated efforts across labs.

We describe here a generic approach to the task of producing multiple files where parameter values are changed. The method is based on creating a template file made of static text containing inclusions, indicating how parameters should be varied. The variations are defined in snippets of Python code surrounded by double brackets. For example `[[ [0, 1] ]]` specifies two values: `0` and `1`. The actual generation of the variations, text manipulation and file generation is then delegated to one Python program: Preconfig (see **Fig. 1**). While this approach does not limit the ways in which parameters can be varied, it reduces the required programming skill to the bare minimum. For example, the following template file can be used to vary three parameters:

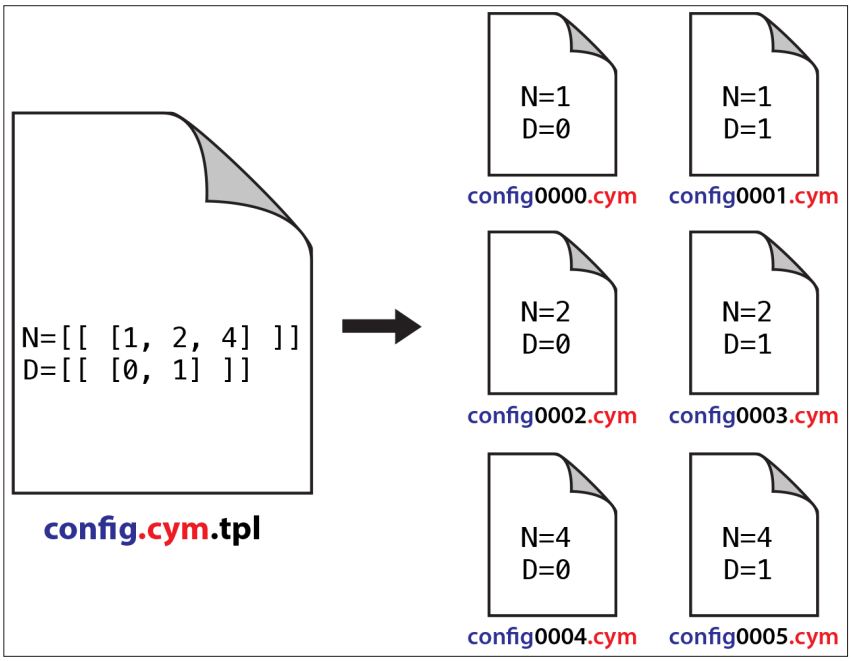

Figure 1: Running `preconfig config.cym.tpl` will generate files corresponding to all possible combinations, of parameter variations specified in the template file. The name of the files to be generated is derived from the name of the template file, by splitting the name and extension (blue and red parts).

```
N = [[ [0, 1, 2, 3, 4] ]]
D = [[ [1, 10, 100, 1000] ]]
v = [[ [-1, 0, 1] ]]
```

Here each code snipped specifies a list of values, and running Preconfig will then generate 60 files covering all combinations. For instance, the first file will contain `N = 0`, `D = 1`, `v = -1` and the second one will have `N = 0`, `D = 1`, `v = 0`, etc.

Implementation and architecture

The program Preconfig reads the template file from top to bottom, identifying snippets of code surrounded by double brackets. It then executes this code using the Python interpreter, and will recurse whenever multiple values are specified. Values are eventually converted to their string representation, and substituted in place of the code snippet. In this way, Preconfig generates all the possible combinations following the order in which these combinations were specified in the file (see **Fig. 1**). Importantly, any accompanying text in the template file is copied verbatim to the output file, such that syntax present in the configuration file is maintained during the process. This allow the same approach to be used with virtually any type of computational pipeline. The template file must be changed to reflect the desired output, but Preconfig can be used unmodified.

Comparison with other tools

Compared to other tools, the main advantage of Preconfig is to automatically generate the appropriate number of files such as to cover all the combinations. It also offers access to the extensive Python mathematical library. The standard UNIX macro processor m4 is for instance limited to integer arithmetic, and a for loop can be used to generate multiple values, but all these values will be placed in the same output file. With Embedded Ruby (eRuby), it is possible to evaluate and substitute Ruby code snippets present in a text file. Ruby has a random number generator and supports floating-point arithmetic, but like m4 and other templating systems, it is made to generate a single output file.

Non-uniform Sampling

Within the code snippets, variables can be defined and used later, in arbitrary arithmetic formula, greatly expanding the versatility of the approach. One can for example scan 2 parameters using 10 values each, one according to a linear scale, and the other with a geometric scale. For this, the easiest is to define intermediates variables `x` and `y`:

```
[[ x = range(10) ]]
[[ y = range(10) ]]
reaction_rate = [[ 2 + 0.5 * x ]]
diffusion_rate = [[ 1 / 2**y ]]
```

The snippets where `x` and `y` are defined are replaced by empty strings.

Random Sampling

The Python Random module can be used to implement random sampling, and in this case the number of files can be specified as an argument. For example `preconfig 100 config.cym.tpl` will generate 100 files, in which the values specified using the random module will be different. It is straightforward to scan multiple values independently:

```
D = [[ random.uniform(0, 1) ]]
R = [[ random.choice([1, 10, 100]) ]]
N = [[ random.randint(0, 1000) ]]
```

Here `D` is a floating-point value between 0 and 1, while `R` and `N` are integers. We use random sampling extensively, because it incurs less arbitrary choices than a regularly spaced set of values. Moreover, with this approach additional parameter sets can easily be added at any time to explore the system in more depth.

Correlated Random Sampling

By first defining a variable using the Random module, one can easily randomize two parameters while keeping their ratio constant. Random values have many digits of precisions, and one may wish to discard some of them. The value below is rounded off to 3 digits of precision:

```
[[ x = round(random.uniform(0,1), 3) ]]
binding_rate = [[ 100 * x ]]
unbinding_rate = [[ 5 * x ]]
```

To generate less-than-perfectly correlated variables one could use:

```
[[ x = random.uniform(0, 1) ]]
binding_rate = [[ 100 * x + random.uniform(0, 10) ]]
unbinding_rate = [[ 5 * x + random.uniform(0, 10) ]]
```

Boolean Variables

Boolean variables can be used to introduce qualitative differences:

```
[[ enabled = random.choice([0, 1]) ]]
feeback = [[ random.uniform(0, 1) if (enabled)
else 0 ]]
```

The values of parameters are not restricted to be numerical, and strings can also be used as values, as in `color=[[ random.choice(['red', 'green', 'blue']) ]]`. Parameters variations can also be specified on the command line in addition to the one found in the template file. If desired, Preconfig provides intuitive output in its most verbose mode, and can also generate a CSV log file of its activity containing the substitutions operated for each file. It will derive its naming scheme for the output files from the name of the template file automatically (**Fig. 1**). Detailed instructions are provided by '`preconfig --help`'.

Preconfig is not by itself a black box optimization tool such as OPAL, ParOpt or Spearmint, because it lacks an optimization algorithm, and does not include any a feedback mechanism on the basis of which it could optimize parameters. We have however ourselves used Preconfig as a component of a genetic algorithm with a selection-mutation optimization scheme in the past [1]. For this work, a Python program was in charge of managing generations of 'individuals' that were simply sets of parameter values. Cytosim [2] was in charge of calculating fitness values and the task of writing the configuration files for Cytosim was delegated to Preconfig. For this purpose, Preconfig can be loaded as a module into Python, and then called with a Python dictionary containing key-value pairs, which in this case represented the 'genome' of each individual:

```
import preconfig
file_name = preconfig.parse('config.cym.tpl', genome)
```

In conclusion, Preconfig makes it is easy to generate multiple configurations files in which parameters are varied. It is straightforward to implement exhaustive enumerations or random sampling within a certain range of values. These approaches can easily be combined, using code that remains brief and intuitive. With another script, one can invoke one's favorite simulation tools with all these files, either sequentially or in parallel, depending on the system used to conduct the calculations. Several tests and example template files are distributed along with the Python code. Importantly, the template approach is truly generic, and it has the advantage that all the information defining the variations is contained in a single file, which streamlines maintenance, archiving and distribution.

Quality control

We have used Preconfig extensively during modeling courses and daily research over the past 5 years. We provide three type of template files to test Preconfig:

- Cytosim [2] configuration files: configX.cym.tpl
- Smoldyn [3] input file: smoldyn.txt.tpl
- BioModel [4] XML file: BioModel.xml.tpl

To test them, adjust the current directory to match the file locations, and enter the following commands, one by one:

```
preconfig configA.cym.tpl            (4)
preconfig configB.cym.tpl 16         (16)
preconfig configC.cym.tpl            (4)
preconfig smoldyn.txt.tpl            (3)
preconfig BioModel.xml.tpl           (3)
```

The number of files generated by each command is indicated in the right column.

(2) Availability

Programming language and Dependencies

Preconfig is written in Python, and without other dependencies, should run on any system with a Python interpreter.

Software location

Preconfig is free and open source, and distributed under the GNU GPL license version 3.0. It is available on https://github.com/nedelec/preconfig.

Language

All documentation is provided in English.

(3) Reuse potential

The software can be used to vary parameter values for virtually any type of configuration file. The template approach does not limit the type of parameter variations that can be achieved, and is compatible with usual syntax. If double brackets are reserved, an internal variable of Preconfig can be changed to use other marks to identify embedded code. It thus covers a very broad range of applications. It is tempting to think about how Preconfig could be extended for direct parameter optimization. Support

is provided by contacting the author, and suggestions of improvement are welcome.

Additional files

The additional files for this article can be found as follows:

- **BioModel.xml.** BioModel template file. DOI: https://doi.org/10.5334/jors.156.s1
- **configA.cym.** GNU GPL License version 3 configA.cym.tpl. DOI: https://doi.org/10.5334/jors.156.s2
- **configB.cym.** Cytosim template file 1 configB.cym.tpl. DOI: https://doi.org/10.5334/jors.156.s3
- **configC.cym.** Cytosim template file 2 configC.cym.tpl. DOI: https://doi.org/10.5334/jors.156.s4
- **LICENSE.** Documentation in HTMF format LICENSE. DOI: https://doi.org/10.5334/jors.156.s5
- **preconfig.** Table of Content preconfig. DOI: https://doi.org/10.5334/jors.156.s6
- **README.** Documentation in Markdown format README.html. DOI: https://doi.org/10.5334/jors.156.s7
- **README.** Python executable program 'Preconfig' README.md. DOI: https://doi.org/10.5334/jors.156.s8
- **smoldyn.txt.** Smoldyn template file BioModel.xml.tpl. DOI: https://doi.org/10.5334/jors.156.s9

Acknowledgements

We wish to thank the members of the Nedelec group, and all users of Cytosim for their feedback which has contributed greatly to this development.

Competing Interests

The author has no competing interests to declare.

References

1. **Rupp, B** and **Nedelec, F** 2012 Patterns of molecular motors that guide and sort filaments. *Lab on a chip*, 12: 4903–10. DOI: https://doi.org/10.1039/c2lc40250e
2. **Nedelec, F** and **Foethke, D** 2007 Collective Langevin dynamics of flexible cytoskeletal fibers. *New Journal of Physics*, 9: 499510. DOI: https://doi.org/10.1088/1367-2630/9/11/427
3. **Andrews, S, Addy, N, Brent, R** and **Arkin, A** 2010 Detailed simulations of cell biology with Smoldyn 2.1. *PLoS Comput. Biol.*, 6: e1000705. DOI: https://doi.org/10.1371/journal.pcbi.1000705
4. **Le Novère, N** et al. 2006 BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Research*, 34: D689–D691.

How to cite this article: Nedelec, F 2017 preconfig: A Versatile Configuration File Generator for Varying Parameters. *Journal of Open Research Software,* 5: 9, DOI: https://doi.org/10.5334/jors.156

Submitted: 23 November 2016 **Accepted:** 15 February 2017 **Published:** 05 April 2017

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS