# Yggdrasil Tutorial

# 1: Introduction

Yggdrasil is a framework designed as part of the [Crops in Silico](http://cropsinsilico.org/) project (CiS). The goal of yggdrasil is to facilitate the analysis of complex systems across multiple scales and areas of interest by providing a way for models in different programming languages and resolutions to communicate. While current use of yggdrasil is confined to biological models from the gene to field level, the same principle allows models of supported languages in any field to be integrated. To do this effectively, yggdrasil is designed to be usable by domain scientists and modellers without in-depth programming knowledge; however, the technology is still in development and may present obstacles to new users. This tutorial will walk through the steps of setting up and running a model integration in yggdrasil, while also covering technical issues that may be encountered by domain users.

As an introductory resource, this will not cover more specific technical questions related to yggdrasil’s underlying functioning; for these and any other further questions about yggdrasil, please refer to the framework’s [documentation](https://cropsinsilico.github.io/yggdrasil/index.html). Specific pages from the documentation will be referred to throughout this tutorial for more information on each component of a model integration. The components needed in order to create a model integration using yggdrasil are:

1. Two or more scientific models that share input and output data (eg, an output of model A is the same variable as an input for model B)
2. An installation of the yggdrasil framework
3. YAML files describing the location, characteristics, and input/output communication channels for each model
4. A YAML file describing the links between the communication channels for each model and input/output file
5. Code wrapping the computational portion of each model, allowing yggdrasil to call the model and pass data.

This tutorial assumes the use of pre-existing models to be integrated; the implementation and use of each further component is described in its own section. Of note, while many of the components created for a model can be easily adapted and reused, each model integration should be planned and executed separately in order to insure that the relations between models are handled as expected. Relevant sections will be accompanied by code examples from an existing integration between a Gene Regulatory Network (GRN) model and a protein translation model, written respectively in R and python.

# 

# 2: Yggdrasil setup

This section will cover the installation and setup of yggdrasil. Conda is the preferred method of installation and so will be the one focused on here; however, other methods, as well as more details regarding installation, are outlined [here](https://cropsinsilico.github.io/yggdrasil/install.html).

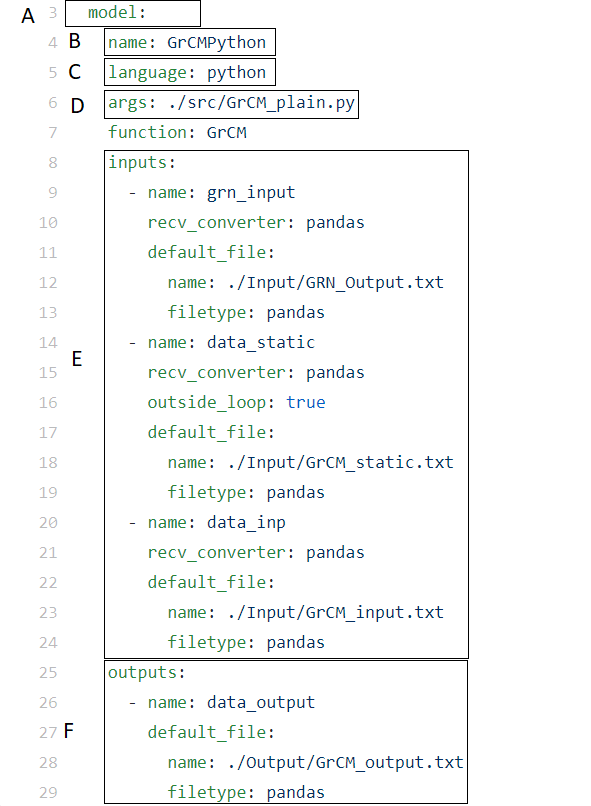
1. The preferred method for installing yggdrasil is by using Anaconda via conda-forge. If Anaconda is not already set up itself, it should be installed first. Either Anaconda or the more stripped-down Miniconda work to install yggdrasil, and are available [here](https://www.anaconda.com/products/individual#Downloads). For more information on the use and installation of Anaconda, many [other](https://www.youtube.com/watch?v=YJC6ldI3hWk) [resources](https://docs.anaconda.com/anaconda/user-guide/getting-started/) are available.
2. If working on Windows, Microsoft Visual Studio is also required to run yggdrasil. The free version, Visual Studio Community, can be installed [here](https://visualstudio.microsoft.com/vs/community/). While installing Visual Studio, the following components should be enabled:
   1. Workload: Desktop & Mobile: “Desktop development with C++”
   2. Individual Component: Compilers, Build Tools, and Runtimes: “MSVC v140 - VS 2015 C++ build tools (v14.00)”
3. After installing the preferred conda distribution, yggdrasil can be installed from a command line terminal on Linux or MacOS, or from PowerShell on Windows, by calling:

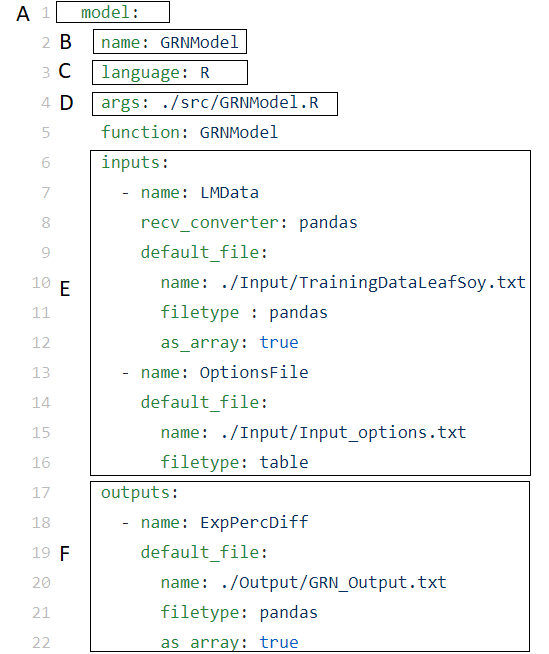
### $ conda install -c conda-forge yggdrasil

1. Depending on the model language and operating system on which yggdrasil will be run, additional steps may be required. The conda installation automatically covers most of these by installing them as dependencies. However, it is still recommended to check the [following list](https://cropsinsilico.github.io/yggdrasil/install.html#additional-steps-on-windows) of situations in which further support is necessary.

# 3: Model YAML

Yggdrasil uses YAML files to organize the communication channels between integrated models and input/output files. [YAML](https://yaml.org/) is a human-readable markup language that primarily uses indentation for formatting; because of this, it was chosen to express the data structure of models integrated by yggdrasil with minimal complex syntax. Because of their simplicity, YAML files are writable in most text editors. In addition, both [Emacs](https://www.gnu.org/software/emacs/) and [Notepad++](https://notepad-plus-plus.org/) include built-in support that will color code text, auto-indent lines, and highlight syntax errors when set to write in YAML.

The first configuration file to be written in YAML for a model integration is the descriptor YAML for each included model. These model YAMLs describe the model’s language, location, input and output channels, and any other information necessary for the model to be called properly by yggdrasil. Information in YAML is handled using key-value pairs, in which a field and the data held in that field are mapped together using the format ‘key: value'. A complete list of keys recognized by yggdrasil is available [here](https://cropsinsilico.github.io/yggdrasil/yaml.html#model-options); however, this tutorial will cover a small number of critical keys necessary for all model YAMLs. Each key referenced is also labelled in Figures 1 and 2 below, the example YAMLs for a GRN model and protein translation model, respectively.



Figures 1 and 2: GRNModel.yml, model descriptor YAML file for the GRN model, and GrCM.yml, model descriptor YAML file for the protein translation mode

1. model: Each model key denotes that all other keys within its indentation level refer to a singular model.
2. name: The name by which yggdrasil will refer to the model. Each model name in an integration must be unique.
3. language: The programming language in which the model is written. Currently, [these languages](https://cropsinsilico.github.io/yggdrasil/yaml.html#available-languages) are supported.
4. args: The location of the model relative to yggdrasil. If there are any other arguments necessary to call the model from the command line, they should also be included here, separated by spaces.
5. inputs: The inputs key includes separate values for each model input. These inputs are denoted using the ‘name’ key (note the special ‘- ’ formatting used to denote that the names are a sequence of matching keys). As shown in the example YAMLs, each input channel can be assigned many other attributes. While these are useful to provide more detailed information to yggdrasil on how to handle data passed to the model, and may be necessary depending on the data used, the only mandatory key for each input is a name. Importantly, the names assigned to inputs and outputs by the model YAMLs are used to set up communication channels by yggdrasil. Therefore, each name must be unique within a model integration.
6. outputs: Similar to inputs, the outputs key includes separate values for each model output.

# 4: Connection YAML

In addition to the YAML describing each model, a YAML file is needed that covers the connections between each file and model in an integration. These connections describe the links between each of the communication channels created in the model YAMLs. The connection YAML, much like the model YAMLs, can include a variety of keys and [additional information](https://cropsinsilico.github.io/yggdrasil/yaml.html#yaml-conn-options) to inform yggdrasil’s handling of data and driver choice. However, the following keys are necessary for each connection YAML.



Figure 3: GRNModel\_ProtTranslationModel\_CO2.yml, connection YAML file for the GRN and protein translation model integration

1. include: The include key lists the relative locations of a single model YAML or a sequence of model YAMLs that correspond with the communication channels listed in this communication YAML.
2. connections: The bulk of the connection YAML is a sequence of input/output keys, each describing a single communication channel, that are listed under the connections key. A connection YAML must have at least one input connection and one output connection in order to be recognized. The input key of a connection describes the input file or model output channel that provides that data, while the output key describes the output file or model input channel to which the data will be sent. In the example connection YAML shown in Figure 3, the connections have been further broken up by whether they describe (C) inter-model connections, (D) input file to model input connections, or (E) model output to output file connections. Note that when model input and output channels are used, they are described with both the model name and channel name.

Based on the YAMLs included and connections established in the connection YAML shown in Figure 3, the resulting model integration would pass information as shown in the diagram in Figure 4. Note that each communication channel in GrCMPython (the name of the protein translation model in yggdrasil) is connected to a corresponding input or output channel or file.

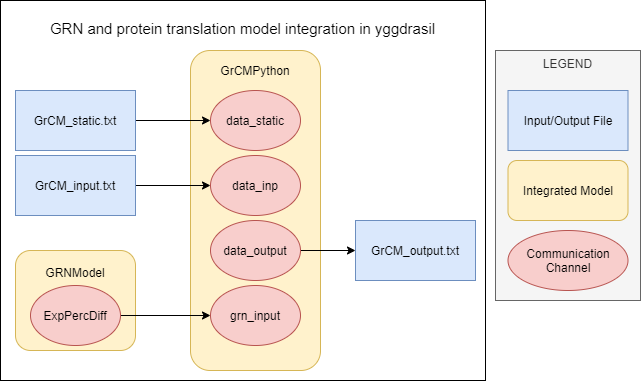


Figure 4: Diagram of model integration created in example YAML files

# 

# 5: Model wrapper

The final component necessary for models to be integrated through yggdrasil is code that ‘wraps’ the model, allowing the interface to pass and receive information directly from the model’s computational processes. This is achieved by one of two methods. The first method is yggdrasil’s built-in autowrapper function, which which allows yggdrasil to call models based on the additional ‘function’ key in the model YAML. The second method is to write code that manually links the communication channels created by yggdrasil with the appropriate parameters in the model and creates a loop over which yggdrasil will run the model. This section will cover each method in more detail.

The preferred method is to make use of the [autowrapper](https://cropsinsilico.github.io/yggdrasil/autowrap.html) function developed for yggdrasil. This can be done for any model that can be expressed as a single main function (that can in turn make calls to other functions) in any of the languages supported by yggdrasil. The autowrapper will check the function’s inputs and outputs and assign corresponding communication channels based on the inputs and outputs described in the model YAML. In turn, the number of input and output channels listed in the YAML must match the number of inputs and outputs expected by the model function.

To use the autowrapper, the ‘function’ key must be added to the model YAML, with the name of the model function to be called as its value. This can be seen in the model YAMLs in Figures 1 (line 5) and 2 (line 7), which name the functions to be called with the inputs listed in each YAML file. The main function of the GRN model described in Figure 1 is GRNModel, and is located in the file GRNModel.R listed under args.

The second method is manually write the model wrapper, adding code to the model that links the yggdrasil communication channels with the model’s inputs and outputs, and runs the model over the loop desired for the integration. This method more complex and requires the capacity to write code in the model’s programming language, but also allows yggdrasil to interface with models in custom ways, or with models that cannot make use of the autowrapper function. Figure 5 shows an example model wrapper written in python for yggdrasil’s documentation.

The model used in this wrapper is simply a placeholder to demonstrate the syntax of yggdrasil functions in python (for reference in other programming languages, see more example model wrappers and YAML files [here](https://cropsinsilico.github.io/yggdrasil/examples/examples_toc.html)). Each part of the model wrapper described below is also labelled in Figure 5.

1. The classes YggInput and YggOutput, as well as their related functions, are imported from the yggdrasil interface.
2. YggInput and YggOutput are used to initialize the communication channels described in the model and connection YAMLs. In this example the model has a single input and output with placeholder names. In a working model, the ‘in\_channel’ variable would have a more descriptive name for use in the model, while the ‘inputA’ channel name would be the corresponding communication channel named in the model YAML. The same would apply for the output variable and channel, as well as any further communication channels necessary for the model integration.
3. The model is run as determined by its place in the model integration. This example simply runs until it no longer receives data from the lone input channel; others might loop over timesteps, geographic coordinates, or other relevant parameters. This portion of the model wrapper is highly dependent on both the model used and its relation to the other models and files with which it will interface.

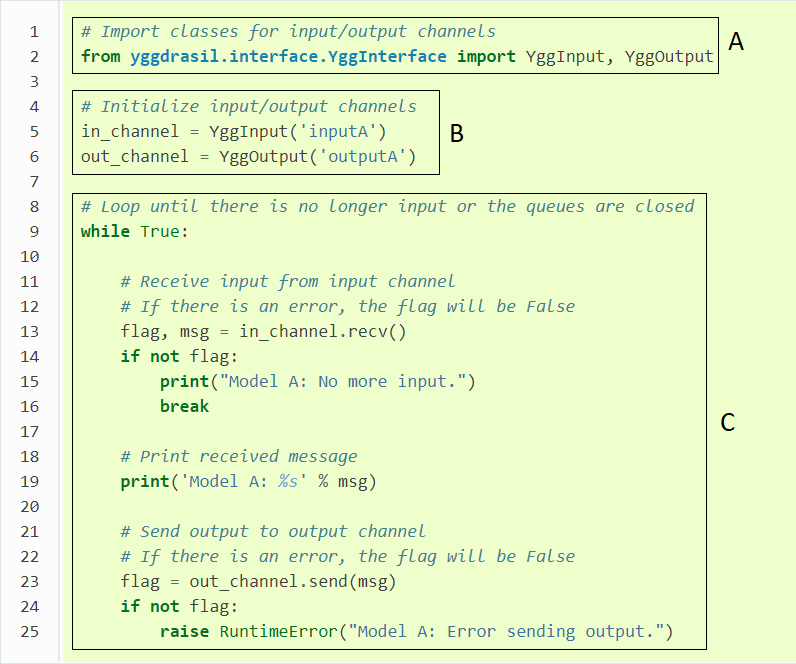


Figure 5: Example model wrapper in python

# 6: Running an integration

To run a complete model integration, use a command line interface to navigate to where yggdrasil is installed, and then call yggrun followed by the path to all YAML files associated with the integration, separated by spaces. If those YAML files are in the same location as the installation, simply list each file. For example, the GRN and protein translation model integration would be run by calling:

### $ yggrun GRNModel\_ProtTranslationModel\_CO2.yml GRNModel.yml GrCM.yml

This will run yggdrasil, which will in turn create the communication channels and call the models listed in the YAMLs. Yggdrasil will provide status messages to the terminal, notifying the user as information is passed between components as well as when the integration either finishes running or encounters an error.

Notes:

1. For resolution of further issues, please refer to the [debugging](https://cropsinsilico.github.io/yggdrasil/debugging.html) page of yggdrasil’s documentation or open an issue on [yggdrasil’s GitHub repository](https://github.com/cropsinsilico/yggdrasil). Because yggdrasil is still in development, support for further features and languages is regularly rolled out.
2. Thank you to Kavya Kannan and Meagan Lang for creating the resources used in the development of this tutorial.