

General modules

package for iFlow

Yoeri Dijkstra

Copyright © 2017. Y.M. Dijkstra

When using iFlow, please cite Dijkstra, Y. M., Brouwer, R. L., Schuttelaars, H. M., and Schramkowski, G. P (Manuscript submitted to Geoscientific Model Development). The iFlow Modelling Framework v2.4. A modular idealised process-based model for flow and transport in estuaries.

Additionally you may refer to this manual as Dijkstra, Y. M. (2017). *iFlow modelling framework. User manual & technical description*.

Note the license obligations that come with iFlow.



Contents

- 1 Modules reference 5**
- 1.1 Output saving and loading 6
- 1.2 Sensitivity and calibration 8

1. Modules reference

The package `general` provides auxiliary modules that may be used in any simulation. The functionalities of the modules include saving output, loading saved files, sensitivity analyses and calibration. This chapter provides a short overview of all modules in the package `general` and the required input and expected output. The modules have been ordered into sections for the purpose of providing structure to this chapter.

Explanation of terms and colours

Behind the input variables we will mention several data types. While some data types may be obvious, some others are explained in the table below:

<i>Space-separated numbers</i>	real numbers separated by one or more spaces. Do not use comma's or other markers to separate the numbers.
<i>Grid-conform array n-dimensional</i>	a numpy array with n (i.e. some number) or fewer (!) dimensions. More dimensions than n is not allowed. All axes should be grid conform. That means that the length of a dimension should either be 1 or equal to the size of the corresponding grid axis. If n is larger than the grid size, the length of this axis is free. Note that a single number counts as a grid-conform array.
<i>General n-dimensional</i>	either a grid-conform array or a numerical or analytical function. In both cases they may n (i.e. some number) or fewer dimensions.
<i>iFlow grid</i>	a grid variable with underlying subvariables as described in the manual (Dijkstra, 2017)

The cells with input variables have been colour-coded to indicate whether the variable is likely to be given in the input file, computed by another module or given in the configuration file. By the very nature of iFlow this is only indicative and depends on the modules used. As an example, almost any variable given in the input file may be used as a variable in a

sensitivity analysis. It then becomes an input parameter of the sensitivity analysis module in the input file. The sensitivity analysis module delivers it to the module that uses this variable.

	Likely a parameter in the input file
	Either in the input file or from another module
	Likely a parameter computed by another module
	Likely a constant in the configuration file <code>src.config</code>

1.1 Output saving and loading

1.1.1 Output

The module Output is the standard output module of iFlow. It saves the variables listed after `requirements` on the output grid to a Python Pickle file. Additionally to the listed variables, it saves the output grid under the name `'grid'` and it saves all the configuration and input file variables. For more information and examples, see the iFlow framework manual (Dijkstra, 2017).

Type	Normal
Submodules	None
Input	<p>requirements <i>Space-separated strings.</i> The keyword <code>requirements</code> should be placed in the input file. It may be placed anywhere, but it is best practice to put it at the last line. iFlow uses this keyword in general to determine what variables should be computed. The module Output uses this information to determine the variables to save. Additionally Output saves the output grid under the name <code>'grid'</code> and it saves all the configuration and input file variables.</p> <p>path <i>String.</i> Path to the output folder. May be relative to the working directory or absolute.</p> <p>filename <i>String.</i> File name without extension. If the file name already exists, iFlow will save the output under the file name appended by the smallest available integer to create a unique name. File names allow for dynamic naming, i.e. replacing part of the file name by the value of a variable at runtime. This may be a variable given in the configuration file, input file or computed by a module before Output is called. For this use <code>@{ ... }</code>, replacing <code>...</code> by a DataContainer call. For example <code>@{'Q0'}</code>, will be replaced by the value of the variable <code>Q0</code> and <code>@{'A0', 1}</code> will be replaced by the second element (i.e. with index 1) in the list/array <code>A0</code>.</p> <p>iteratesWith <i>Optional, module name.</i> Optional input indicates that the output module should be called during every iteration of the module specified in this input variable. For example <code>iteratesWith general.Sensitivity</code> indicates that the output module should be called during every iteration of the module <code>general.Sensitivity</code>.</p> <p>saveAnalytical <i>Optional, space-separated strings.</i> Optional list of variable names. The output module will try to save these variables as analytical functions. If the data type of this variable is indeed a reference to an analytical function, it will not convert the reference to numerical data, but save the reference along with any class variables of the class that this function refers to.</p>

<code>dontConvert</code>	<i>Optional, space-separated strings.</i> Optional list of variable names. These variables will not be converted to the output grid upon saving. This should be used for variables that might seem grid-conform, while they have nothing to do with a grid. For example on a grid with axes x, z, f a variable with axes a, b might seem grid-conform if a and b have the same length as x and z . iFlow will try to convert this to the output grid, while this is not appropriate. Never choose <code>dontConvert</code> for real grid-conform variables. This would mean that the variable is saved on the computational grid, which is not saved along with the output. It will therefore be hard to reload the saved data.
--------------------------	---

1.1.2 ReadSingle

Read the contents of a single file saved using the standard Output module and restore the contents to a DataContainer structure.

Type		Normal
Submodules		None
Input	<code>folder</code>	<i>String.</i> File path to folder with the file to read. The path may be relative with respect to the working directory or absolute.
	<code>file</code>	<i>String.</i> File name without extension. Alternatively <code>all</code> may be used to read all files in the folder, but only the last file that is read is kept in memory.
	<code>variables</code>	<i>Space-separated strings.</i> List of variables to read from the file or <code>all</code> to read all variables. Other modules will see this input variable as the list of variables that becomes available for them. Therefore it is not recommended to use <code>all</code> , as other modules then do not know what variables are available and it might be impossible to build a call stack.
Output	<code>@variables</code>	<i>Dictionary/DataContainer with variables.</i> Returns a data structure with the loaded variables.

1.1.3 ReadMultiple

Read the contents of a multiple files saved using the standard Output module. The results are stored in a variable `experimentdata` with subvariables equal to the file name. For example consider a folder with files 'file1.p' and 'file2.p', each containing a variable u . Assume a module receives the results of `ReadMultiple` in a DataContainer `self.input`. The can then access this data as

■ Code sample 1.1

```

1  dc1 = self.input.v('experimentdata', 'file1') # returns a DataContainer with contents of file1.p
2  u1 = dc1.v('u') # extract the variable u from this DataContainer
3  dc2 = self.input.v('experimentdata', 'file2')
4  u2 = dc2.v('u')
```

Alternatively, when the file names are unknown, many or difficult a workflow can be

■ Code sample 1.2

```

1  filenames = self.input.getKeysOf('experimentdata') # loads all subkeys,
2  #i.e. filenames, under experimentdata
3  u = [ ]
4  for file in filenames:
```

```

5     dc = self.input.v('experimentdata', file) # returns a DataContainer with contents of file
6     u.append(dc.v('u'))

```

Type		Normal
Submodules		None
Input	folder	<i>String</i> . File path to folder with the file to read. The path may be relative with respect to the working directory or absolute.
	files	<i>Space-separated strings</i> . File names without extension. Alternatively <code>all</code> may be used to read all files in the folder.
	variables	<i>Space-separated strings</i> . List of variables to read from the file or <code>all</code> to read all variables. Other than in the module <code>ReadSingle</code> , <code>ReadMultiple</code> will output a variable <code>experimentdata</code> . Other modules will thus not see the contents of the input variable <code>variables</code> . Using <code>all</code> here thus yields no problems in the creation of the call stack.
Output	<code>experimentdata</code>	See explanation above.

1.2 Sensitivity and calibration

1.2.1 Sensitivity

The module `Sensitivity` is a simple yet powerful tool to perform sensitivity studies, i.e. repeat a simulation for multiple values of one or more input variables. The variables and number of variables to include in the sensitivity study can be chosen on input. This is best illustrated using an example of the input file entry of `Sensitivity`

■ Code sample 1.3 — input file entry of `Sensitivity`.

```

1 ## Sensitivity ##
2 module     general.Sensitivity
3 variables  Q1 ws
4 Q1         100 200 300 400 500
5 ws         10**np.linspace(-4.5, -1.5, 20)
6 loopstyle  permutations                # simultaneous or permutations

```

The input `variables` provides a list of the variables to loop over. This can be one or multiple. These variables, here `Q1` and `ws`, should be included as input variables as well. Each should be followed by the values to loop over. In case of `Q1` these values are provided as a space-separated list, containing five values. In case of `ws` this is provided as Python code. This code is interpreted by the module `Sensitivity`. The Python code here allows for Numpy commands that follow after `np..` The variable `ws` loops over 20 values. Finally, `loopstyle` allows for two options `permutations` or `simultaneous`. In the first case, all combinations of all variables are taken. In this example this results in 5 times 20 equals 100 simulations. In the case `simultaneous` is used, the values of all variables are changed simultaneously. This requires all variables to have the same number values and is therefore not possible in this example.

Type		Iterative
Submodules		None
Input	variables	<i>Space-separated strings</i> . See example above.
	@variables	<i>Space-separated numbers or Python code</i> . Values of each variable listed under <code>variables</code> . See example above.
	loopstyle	<i>String</i> . Allows for <code>permutations</code> or <code>simultaneous</code> . See example above.

Output	@variables	values of all the variables listed in the input argument.
--------	------------	---

1.2.2 ManualCalibrationPlot

Post-processing and visualisation of a calibration study for the water level elevation in a 2DV simulation. This module requires multiple output files with model results for different values of one or two calibration parameters (e.g. produced by the module Sensitivity). This data should be loaded back into a variable `experimentdata` (e.g. through the module LoadMultiple). ManualCalibrationPlot will then evaluate a cost function for each of these output files by comparing the modelled water levels to measurements. The results are plotted.

Type		Normal
Submodules		None
Input	<code>calibration_parameter</code>	<i>Space-separated strings.</i> Names of one or two calibration parameters
	<code>axis</code>	<i>Space-separated strings.</i> Axis modification for plotting. Number of elements should match the number of calibration parameters. Allows for <code>log</code> for a logarithmic axis or any other value for a normal linear axis.
	<code>unit</code>	<i>Optional, Space-separated strings.</i> Units to be placed at the axes. Number of elements should match the number of calibration parameters.
	<code>label</code>	<i>Optional, Space-separated strings.</i> Labels to be placed at the axes. Number of elements should match the number of calibration parameters.
	<code>measurementset</code>	<i>String.</i> Name of the variable that contains measurement data.
	@ <code>measurementset</code>	<i>iFlow measurementset.</i> Output of a measurement data module. No example given at the moment. Contact the authors for an example module.



Bibliography

Dijkstra, Y. M. (2017). *iFlow modelling framework. User manual & technical description*.

Dijkstra, Y. M., Brouwer, R. L., Schuttelaars, H. M., and Schramkowski, G. P. (Manuscript submitted to Geoscientific Model Development). The iFlow Modelling Framework v2.4. A modular idealised process-based model for flow and transport in estuaries.