IBMlib

A generic tool for individual-based modelling of aquatic organisms

Global version of code: Revision: Unknown
SVN revision of this manual: Revision
Last change of this manual Date

September 26, 2018

Contents

Chapter 1

Overall concepts

IBMlib is a tool box for individual-based modelling intended for research purposes by emphasizing flexibility and detailed simulation control ability. The core vision of IBMlib is to provide a light weight environment that easily allows to combine existing/new biological modules with existing/new physical data sets. IBMlib provides and supports simple core functionality and provide templates for new developments. IBMlib abstracts biological organisms as particles which may display arbitrary behavior and have complex properties describing life processes. The IBMlib implementation concept is shown in Figure 1.1 It shows a core part IBMlib_core that contains generic core functionality for particle tracking. IBMlib_core accesses the physical environment via the physical interface; this interface can be linked to any data source that is able to describe the physical environment. The biological properties is specified in particle state module that provides the particle state interface to IBMlib; this amounts to specifying e.g. how biological particles grow and behave, if relevant. The particle state module can also describe passive particles and other idealized types. Finally the task interface contrals what IBMlib is actually going to do, e.g. a forward simulation or a data dump or any customized run type. When an actual physical, biological and task module has been selected, we refer to it as an IBMlib configuration. The setup is able to run offline (i.e. reading physical and biological fields from a database) and online (i.e. running contemporarely and coupled with physical circulation and ecosystem models) using various coupler schemes.

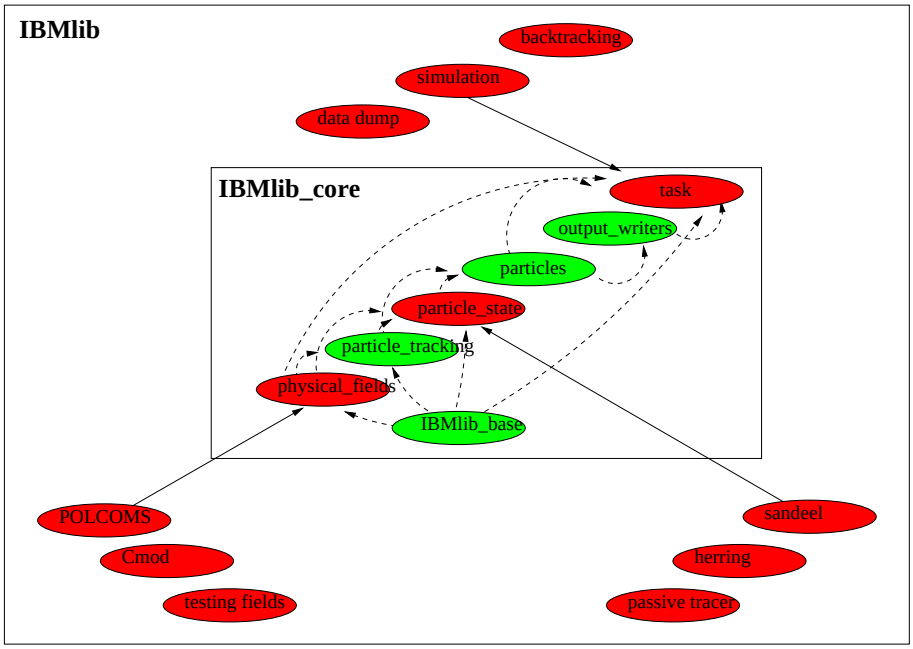

Figure 1.1: Concept diagram of the IBMlib framework.

Chapter 2

License issues

IBMlib is free open software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation. A copy of GNU Lesser General Public License pertaining to IBMlib is provided in the file lgpl.txt, referring to the GNU General Public License provided in gpl.txt. IBMlib is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Chapter 3

Installation

3.1 System resources and requirements

IBMlib is currently set up to a Linux environment, however there should be no fundamental problems to port IBMlib to other OS environments. The minimal system resources required are:

- Fortran 90 compiler. All performance code is written in Fortran 90.
- Gmake. All objects are build from source code by makefiles. All makefiles are written for and tested with gmake; other make implementations may be used, but this may require minor adaptation to makefiles
- Python. Fortran modules are scanned for use associations with a preprocessor utility in Python. At some point we may ship dependency files along to avoid this requirement.

Depending on which sub modules for physical environment, biological properties and task are selected additional system resources may be required (e.g. NetCDF and HDF) - this should be available from the automatic documentation of the sub modules. Further the usage of postprocessing tools for visualization and data analysis may require further system resources, e.g. R and MatLab. The IBMlib test suite applies bash scripts.

3.2 Obtaining IBMlib sources

IBMlib are usualyy distributed in either two ways:

1. A tarball. Place the tarball (referring to this with filename `ibmlib.tgz`) in a desired directory (referring to this with directory name `IBMLIB_DIR`). The type on the command line:

   $$tar \;\; xvfz \;\; ibmlib.tgz$$

   to inflate the tarball. Now IBMlib is ready for configuration, see Sec. 3.3 below.

2. Via SVN source control. You need a login to obtain sources via SVN. The IBMlib repository is currently hosted by DTU. To get the latest version type on the command line in `IBMLIB_DIR`:

   $$svn \;\; co \;\; svn://svn.gbar.dtu.dk/asch/IBM\_Phoenix/trunk$$

   Now IBMlib is ready for configuration, see Sec. 3.3 below. To obtain a specific revision use the `-r` option to svn.

3.3 Building an IBMlib configuration

1. Configuring IBMlib. This basically amounts to placing/editing two files in the IBMlib base directory `IBMLIB_DIR` (where the Makefile is):

   - **config.mk**. This is the *logical* configuration of IBMlib, which is platform independent. In `config.mk` you select sub modules for physical environment, biological properties and task. These are read by the Makefile. This is done by stating the file directory, where these modules is. More specifically, you assign the the variables `PHYSICAL_FIELDS_DIR, PARTICLE_STATE_DIR, TASK_DIR` in which file directory the module is. The Makefile reads this from `config.mk` and takes care of the rest By default, the offline version builds into the executable `ibmrun`; if you want it to be called something else, set the make variable `EXECUTABLE` as desired. In

directory `setups/configurations/generic` you may find examples on files that can be copyed to `config.mk` and modified as desired.

- **compiler_defaults.mk**. Here you specify the F90 compiler you wish to use to compile IBMlib, as well as the compilation flag that should be applied. In directory `setups/compilers` you may find examples on files that can be copyed to `compiler_defaults.mk` and modified as desired. For each compiler, there are different compilation flag sets corresponding to what you may want to do, e.g. debugging (slower,safer) or production runs (faster, optimized, with no unnecessary runtime checks)

Additionally, you may augment **common_rules.mk** if you add customized components requiring special make rules to build. To ease the configuration step, there are configuration scripts in `setups/combo_scripts` that does the two steps for you. Just type

setups/combo_scripts/script_name

in `IBMLIB_DIR` to configure IBMlib corresponding to *script_name*. This is identical to the usual `./configure` in Linux source packages. In this way you can quickly setup your favorite configuration without typing arguments to `./configure`.

2. Building IBMlib. When IBMlib is configured as above, you simply type

make

on the command line in `IBMLIB_DIR`. Then the selected IBMlib configuration is build into the executable name given as `EXECUTABLE` in `config.mk`. This executable may be copied to somewhere in the standard executable search PATH, if it should be installed as an application.

3. Testing IBMlib. IBMlib comes along with a automatized test suite that basically test the integrity of the IBMlib version you are trying to compile. To invoke it, you simply type

make test

on the command line in in `IBMLIB_DIR`. If you are developing modules to IBMlib, you will find additional auxillary tests useful in the development phase, as well as auxillary analysis components. `IBMLIB_DIR/test_suite`.

3.4 Running an IBMlib configuration

When the executable is build, you simply type

$$ibmrun\ [\langle arguments \rangle]$$

on the command line. Often a file with simulation parameters is required as argument after ibmrun; the task sub modules should document which arguments and options they require and support.

3.5 Simulation files

When IBMlib is used in a standard task configuration, like a basic simulation, it expects you to provide an input file on the command line like

$$ibmrun\ \langle inputfile \rangle\ [\ >\ \langle outputfile \rangle]$$

All parts of the compiled IBMlib configuration will try to read most input data from this file in a standard task configuration. File reading is distributed, meaning that different parts of IBMlib reads this file independently of the other parts. The simulation input file is based on a tagged input ASCII format containing lines as

$$tag = value\ [value*] \tag{3.1}$$

tag is a character identifier terminated at first occurence of the separator =. value of the tag is what follows the separator to the end of that line and anything can appear here. If the value is a vector, each item in the vector is separated by spaces (but stay on same line). IBMlib does not check that all parameters are read. The same tag may appear many several times - e.g. a particle emission box, in other cases the tag should be unique, e.g. the particle time step. In this case the first

occurence is picked, the rest is ignored. The following rules also apply for the input file markup:

- Comments: everything after (and including) " !" is skipped
- Empty lines and spaces are just ignored
- Malformed lines are just ignored (e.g. if you forgot "=")
- Everything after "=" is considered part of the value(s) for tag. Values need not to be of the same elementary data type, you may e.g. mix letters and numbers. The documentation should specify what is expected for tag.
- Required tag (or value) is missing. This will generate a runtime error, IBMlib will stop, unless a default applies to the tag.
- The order of the tags does not matter.

The input format is convenient for scripting, where upper-level scripts generates input files. The input is read once and cached when the simulation is started. The following are mandatory input parameters:

3.5.1 Task = basic_simulation tags

- *start_time*. The beginning of the simulation, specified as (year month day second_of_day)
- *end_time*. The end time of the simulation, specified as (year month day second_of_day)
- *particle_time_step*. Nominel time step for the integration algortihm in seconds.
- *emitbox*. Gives a window in space and time, where sandeel larvae/eggs are released. The may be specified as many emitbox entries as desired, in this way they may act in parallel and quite complex release patterns can be set up. The first 4 integers are (year month day second_of_day) where the release begins (of that

box); the next 4 integers are (year month day second_of_day) where the release stops (of that box). The next six numbers specify a spatial box (in latitude and longitude) where sandeel larvae/eggs are released. Dry (land-locked) sectors of the spatial box are omitted when releasing larvae/eggs. The first three numbers are the spatial lower SW corner of the release box given as (longitude,latitude,vertical position), the next three numbers are the spatial upper NE corner of the release box given as (longitude,latitude,vertical position). The vertical position can be spcified as either:

- absolute depth: specify the depth below the water surface as a negative number.
- relative depth$z$:$0 < z < 1$. $z = 0$ corresponds to the sea surface, $z = 1$ corresponds to the sea bed.

Biological particles are released uniformly in time and space with in space-time window specified, so that the total number of particles released adds up to the integer given as number 15. All other parameters following number 15 in emit box are passed to the biological module.

3.5.2 Particle_tracking module

- *advec_intg_method*. The integration algortihm for time forward integration of advected particles (options are euler (Euler forward) or rk2/rk4 (Runge-Kutta 2 or 4)).

3.5.3 Example on minimal input file

```
!-----------------------------------------
!       Main simulation control file
!-----------------------------------------

start_time  = 2005 03 01 0    !  year  month  day  second_of_day
end_time    = 2005 03 18 0    !  year  month  day  second_of_day
```

```
advec_intg_method  = euler    !  advection scheme: euler/rk2/rk4
particle_time_step = 1800     !  in seconds for time integration of motion

! --------------- biology spatial control ---------------
! r(1:4) start:  year month day sec_of_day
! r(5:8) end:    year month day sec_of_day
! r(9:11)        lon_min lat_min z_min  (z=0 -> surface)
! r(12:14)       lon_max lat_max z_max  (z=1 -> bottom)
! r(15)          max_number_of_tracers
! r(16:)         other input item to particle state

emitbox = 2005 03 02 0     2005 03 02 3600  2 54 1   3 55 1  100 e
emitbox = 2005 03 02 3600  2005 03 02 7200  4 54 1   5 56 1  100 l 9.66
! -----------------------------------------------------
```

There may be other mandatory entries, depending on which oceanography provider and biology provider you are applying in your configuration - consult these to find out which input fields are mandatory.