User's guide to the "Validator" interactive ocean model validation tool

Hagen Radtke*
Leibniz Institute for Baltic Sea Research Warnemünde (IOW)

February 14, 2019

About this document

This document describes how to install, configure and use "Validator", an interactive ocean model validation tool. It provides a web-based, interactive interface to create plots which compare modelled and measured data at oceanographic stations.

*hagen.radtke@io-warnemuende.de

Table of Contents

1 Introduction

1.1 Philosophy of Validator

Validator is an interactive ocean model validation tool in which the user can easily create plots in a web browser within seconds. These plots of different type (time series, vertical profile, scatter plot, ...) allow to compare one to four models to observations.

In order to create these plots quickly, the model data have to be preprocessed: One NetCDF file per oceanographic station has to be provided. This can be done by extracting the station data from existing 4-d model output, or, ideally, by saving data at the prescribed stations already during the model run.

In the latter case, the model can be validated while it is still running. Often a modeller tries modifications to their model to improve its performance. In this way, the success of these modifications can be checked while the model is still running, allowing to cancel unsuccessful model runs and to save valuable computing time.

On the other hand, the figures produced by this validation tool can be saved as high-resolution PNG images ready to use in scientific publications.

1.2 Details

Details on the implementation can be found in the following publication:

Radtke, H., Börgel, F., Brunnabend, S., Eggert, A., Kniebusch, M., Neumann, D., Neumann, T., Placke, M.: "Validator - a web-based interactive tool for validation of ocean models at oceanographic stations", Journal of Open Research Software, submitted.

1.3 Structure of this document

The following four sections describe installation and use of the Validator app. They are aimed at different skill levels as indicated in the following table.

Section	**describes**	**required skills**
Section 2	Installing Validator	some computer skills
Section 3	Using Validator	none (very easy)
Section 4	Supplying own data	familiarity with NetCDF format
–	Extending Validator	advanced programming in R

2 Installing Validator

2.1 Obtaining Validator

2.2 Installing on a local Linux machine

Validator requires a Linux environment. If you are working on Windows, probably the best way is to set up a virtual Linux machine, e.g. using Oracle VM VirtualBox:
`https://www.virtualbox.org/`

Inside this virtual machine, you need to install an operating system. We have tried openSUSE and can recommend it:
`https://software.opensuse.org/distributions/leap`

2.2.1 Installing R and RStudio

If you use Linux, then R and RStudio may be available as native packages for your Linux distribution, and you can install them via the package manager. Otherwise, please install R first
`https://www.r-project.org/`

When that is done, install RStudio as described here:
`https://www.rstudio.com/products/rstudio/download/`.
We recommend the Open Source license.

2.2.2 Installing required R packages

Validator requires a few R packages to be installed. Please install them using the following commands:

```
install.packages("shiny")
install.packages("ggplot2")
install.packages("RCurl")
install.packages("RNetCDF")
install.packages("plotrix")
install.packages("mgcv")
```

In case it does not work, you will probably need to install some Linux packages first, such as a netCDF package. This depends on your Linux distribution.

2.2.3 Optional: Installing MySQL server to enable data acquisition from the example database

Instead of reading measurement data from static text files, we will want to read them directly from a database. The advantage is that if the database is updated, new data will immediately become available.

Validator is able to link to a database to obtain measured data. These are then cached in ASCII files to minimize database queries and speed up the plotting.

In order to demonstrate this capability, we provide an example database which is made available by a MySQL server and the MariaDB client. The contents of this database are provided in the file
`mysql_example/measurements.sql`
The following instructions describe how to set up a MySQL database providing these data to Validator:

- Install the MySQL server first. This can be done by installing the appropriate package(s) for your Linux distribution, in openSUSE you will need the packages "mariadb" and "libmysqlclient-devel" which are included in the distribution
  `sudo zypper install mariadb libmsyqlclient-devel`
  For Ubuntu 18.04, the following packages need to be installed:
  `sudo apt-get install mysql-server libmysqlclient-dev mariadb-client`
  Installation instructions for other operating systems can be found here:
  `https://downloads.mariadb.org/mariadb/repositories/`
- Start the MySQL service. In openSUSE, this is done by the following command:
  `sudo service mysql start`
- Import the `validatortest` database into MySQL:
  ```
  sudo mysql -u root
  CREATE DATABASE measurements;
  QUIT;
  sudo mysql -u root measurements <
          /PATH_TO_DIR/mysql_example/measurements.sql
  ```
- Create a user `validator` which gets read access to your database:
  ```
  sudo mysql -u root
  CREATE USER 'validator'@'localhost' IDENTIFIED BY 'test';
  USE measurements;
  GRANT SELECT ON * TO 'validator'@'localhost';
  QUIT;
  ```
  Here, `PATH_TO_DIR` stands for the path where the contents of the `validator.zip` file were extracted to, and `test` is the password by which Validator connects to the database.

- The last step is to install the R package providing access to the MySQL database, `RMariaDB`. To do so, just run R or RStudio and type:
  `install.packages("RMariaDB")`
  More information on this package can be found here:
  `https://github.com/r-dbi/RMariaDB`

2.3 Installing on a Linux server

Installation on a Linux server is almost the same as on a local Linux machine. The difference is that instead of running the software in RStudio, we will run it by Shiny Server. This allows several users to use the program simultaneously in different sessions.

The Linux Server can be a virtual machine, we have tried openSUSE as operating system and recommend it.

2.3.1 Installing R and the required packages

Install R as described in Section 2.2. Then, install the required R packages under R directly (not in RStudio).

2.3.2 Optional: Installing MySQL to enable data acquisition from the example database

Installing MySQL works just the same way as on a local machine, see Section 2.2.3.

2.3.3 Installing Shiny Server

Afterwards, you will need to install Shiny Server. You will find installation instructions here:
`https://www.rstudio.com/products/shiny/download-server/`

After installing it, you will need to configure the shiny server. You will find detailed instructions here: `http://docs.rstudio.com/shiny-server/`. You basically have to provide a configuration file which tells the Shiny Server where to find the Shiny Apps to run under defined URLs on this server. In openSUSE, this resides under
`/etc/shiny-server/shiny-server.conf`

An example configuration file is given under
`shiny_server_example/shiny-server.conf`

2.3.4 Allowing client access to Shiny Server

Shiny Server will use the port specified in the configuration file to communicate with its clients. If you are using a virtual machine, you will need to configure port forwarding to make the ports of the virtual machine accessible from outside. In Oracle VM VirtualBox, this can be done under “Change - Network - Advanced - Port forwarding”.

Finally, make sure you open the chosen port in the firewall to allow the Shiny Server to be accessed from other machines.

3 Using Validator

3.1 Launching Validator

3.1.1 Launching Validator on a local machine

Open RStudio and execute the following commands:

```
setwd("/PATH_TO_DIR/")
library("shiny")
runApp()
```

The first command will change the working directory to the path where you downloaded Validator to. Please make sure you enter the path where `server.R` is located. The second command will load the `shiny` package and the last command will launch the Validator app.

Please click on "‘show in browser"’ then to allow full functionality.

3.1.2 Launching Validator on a Linux server

If you run Validator on a server, it will run automatically once the shiny server is launched, which will typically happen during startup. Just open the following URL in your browser:
`http://127.0.0.1:3838/validator`

This URL is correct if you run the browser on the server. Otherwise, please replace `127.0.0.1` by the ip or hostname of your server and `3838` by the port you configured in the port forwarding. If you use port 80, you can leave out `:3838` completely, in which case you may just need to enter something like
`http://shinyserver/validator`

3.2 Applying Validator

Validator runs in your web browser and appears as a web page structured into three parts. In the left column, you select which data shall be shown (which oceanographic station, which parameter, ...). In the top row, you specify how it shall be shown (as a time series, a vertical profile etc). The main frame in the bottom right shows the diagram which is produced and allows to save it.

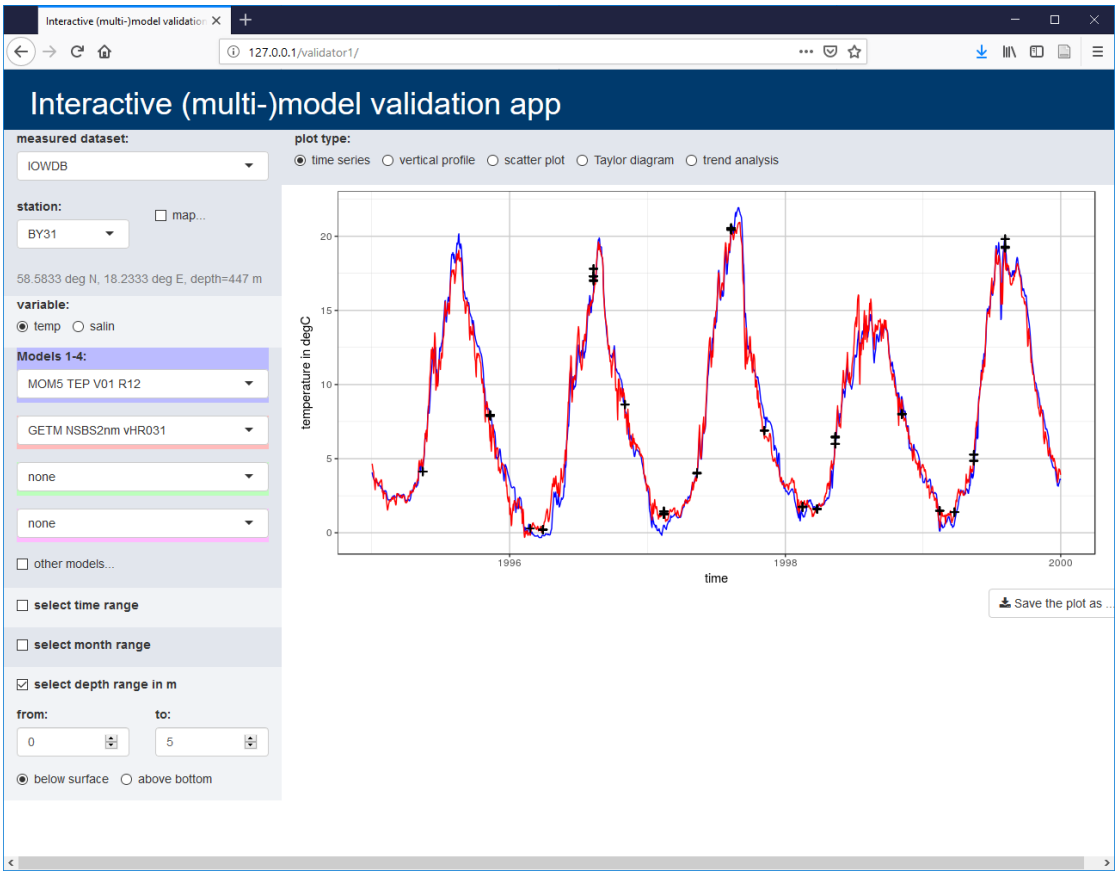

Figure 1: Screenshot of the Validator app in a browser window.

3.2.1 Selecting which data to show

The left column allows first to select an oceanographic station from a list or from a map. If you click "'map..."', a map will show up with clickable stations showing up on it. The top row then allows to switch between different maps.

The next choice to make is the parameter to compare between model and observations, e.g. salinity or temperature.

Third, up to four models can be selected from a predefined list, each of which will be shown in a different colour in the plots which are generated. The model list can be modified (e.g. extended by a new model). To do so, click "'other models..."'. Then, by clicking on "'download default model list"', you will see how a model list looks like. You can modify it and then upload it using the button "'choose own model list"', which makes the other models available for plotting.

The remaining settings are self-explanatory, they allow a subsetting of the data in time and space.

3.2.2 Selecting plot types

The top row allows a selection of plot types. Some plot types have additional settings which shall be explained here.

- **Time series:** Model data are vertically averaged (it might make sense to select a narrow depth range) and drawn as a colored horizontal curve, while individual measurements inside the depth range are shown as black crosses.
- **Vertical profile:** Model data are averaged over time and drawn as a colored vertical curve, while individual measurements from inside the specified time range are shown as black crosses. The options allow to draw not only the temporal mean, but also a range in which the model values in a given depth are scattered. This range shows up as a semi-transparent background.
- **Scatter plot:** For this plot type, we find the nearest model data point in time and space for each individual measurement. Then pairs of measured and modeled data are shown, with the measurements on the x-axis and the corresponding model results on the y-axis. A colored line shows a simple linear fit to the model data. Optionally, a confidence interval can be drawn which shows how uncertain the offset and slope of the linear fit are.
- **Taylor diagram:** Pairs of observed and modeled data are formed as in the scatter plot. The vector of observations is then compared to that of the corresponding model results in terms of correlation coefficient and standard deviation. Ideally, the model should have a perfect correlation coefficient of 1.0 and the same standard deviation as the observations. In practice it has mostly not, and the Taylor diagram allows to check how well different models correlate, and whether they over- or underestimates the variability showing up in the measurements.
- **Trend analysis:** This plot allows to discriminate between seasonal and long-term changes and see which of these are similar in the model and in the measurements. To do so, a Generalised Additive Mixed Model (GAMM) is fitted to the observations and the model results separately. This nonparametric model tries to fit the data by a sum of a smooth nonlinear long-term trend, a smooth and periodic nonlinear

seasonal cycle, and a cAR-1 correlated error term. The options allow to select which of the terms is shown by the horizontal curves in the graph.

3.2.3 Saving plots

All generated plots can be saved in `.png` format using the button "'Save the plot as..."' under the graph. Independent from the current screen resolution, they will be saved in a publication-ready resolution.

4 Supplying own data

The Validator app requires two types of data to be provided: Measured data and model data. Both have to be provided for predefined oceanographic stations.

Measurements can either be given as a collection of ASCII files, or dyamically extracted from a database. Model data need to be provided as NetCDF files. The specific format requirements are explained later in this section.

4.1 Providing measured data

More than one dataset of measurements can be provided to Validator. Each of these datasets, no matter whether the data are provided as ASCII files or from a database, should reside in a specific subdirectory. These locations are provided in the file `datasets.txt` in the Validator main directory. It contains one header line:
`name;path;format`
and one line for each dataset, where the following is provided separated by semicolons and without quotes:

- `name` – the name of the dataset that shall appear in the drop-down list where a dataset can be selected
- `path` – the path to the dataset directory, can be relative or absolute; make sure that read permissions (when reading from ASCII files) or write permissions (when reading from SQL database) exist, especially when running on a server
- `format` – the format in which the data are provided. The following options are possible:
  - `standard` – ASCII files as described in Section 4.1.4
  - `sqldb` – SQL database as described in Section 4.1.5

  Alternatively, user-specific options can be provided as long as a function to obtain the data is added to the R script `read_measured_data.R` and referenced in the files `do_the_plot.R` and `model_validation_plot.R`.

4.1.1 Defining the oceanographic stations

In the dataset directory, provide an ASCII file `stations.csv` with the following header line:
`latitude;longitude;stationname;depth;red;green;blue`
and one line for each oceanographic station, where the following is provided separated by semicolons and without quotes:

- `latitude` – latitude of the station in degrees north (negative=south), as decimal degrees using a dot as separator, e.g. `57.33`

- `longitude` – longitude of the station in degrees east (negative=west), as decimal degrees using a dot as separator, e.g. `20.0`
- `stationname` – name of the station e.g. from a monitoring program
- `depth` – depth of the station in meters, e.g. `120.5`
- `red;green;blue` – color in which this station shall appear on the map, e.g. `1.0;0.0;0.0` for red

4.1.2 Defining the set of variables

In the dataset directory, provide an ASCII file `variables.csv` with the following header line:
`varname;unit;longname`
and one line for each oceanographic station, where the following is provided separated by semicolons and without quotes:

- `varname` – a short variable name or abbreviation which is used in the file names or in the database, e.g. `temp`
- `unit` – the unit as it will be added to the plot axes, e.g. `degC`
- `longname` – the name of the variable as it will be added to the plot axes, e.g. `temperature`

4.1.3 Defining the temporal range of the dataset

In the dataset directory, provide a file `timerange.txt` which describes the temporal range your dataset covers. It shall contain two lines giving the start and end date as YYYY-MM-DD, e.g.:
`1877-01-01`
`2015-12-31`

4.1.4 Providing the data as ASCII files

Create a subdirectory `stationdata` in the dataset directory. In this subdirectory, supply text files of the following names:
`varname_stationname.csv`
where `varname` stands for the variable name as specified in `variables.csv` and `stationname` stands for the station name as specified in `stations.csv`. **Both have to be given in lowercase.**

Each of these files shall contain a header row
`depth;datetime;value`

and one line for each oceanographic station, where the following is provided separated by semicolons and without quotes:

- `depth` – depth below sea level in meters, given as decimal value with a dot as a decimal separator, e.g. `50.0`
- `datetime` – date and time of the measurement in Windows time format, that is, as a decimal number given as "'days since 1899-12-30 00:00:00 UTC"', e.g. `2.0` for January 1st, 1900; values can be negative
- `value` – the value of the measurement in the unit specified in `variables.txt`

If there are no measurements of this variable at this station, please supply a file containing the header line only.

4.1.5 Providing the data as SQL database

The data can be provided in an SQL database. Obviously the SQL queries which validator uses will depend on your database structure. This subsection describes how to connect to a database which has the same structure as the example database given in
`/PATH_TO_DIR/mysql_example/measurements.sql`
Subsection 4.1.6 will describe what to do if your database looks different.

In our example, we connect to a MySQL database with two tables:

TABLE variables

column name	data type	key
id	INT	primary
varname	CHAR(20)	
unit	CHAR(20)	
longname	CHAR(40)	

TABLE measurements

column name	data type	key
id	INT	primary
longitude	DOUBLE	
latitude	DOUBLE	
depth	DOUBLE	
datetime	DATETIME	
variable	INT	foreign (variables.id)
value	DOUBLE	

Create a file `db_connection.txt` in the dataset directory with the following header line
`username;password;host;port;databasename`
and one line following, which gives the follwing information separated by semicolons:

- `username` – the user name under which Validator connects to connect to the MySQL database, enclosed in single quotes
- `password` – the password for this user, enclosed in single quotes, `''` if no password is required
- `host` – the hostname or IP adress of the database server, enclosed in single quotes, `'localhost'` if running locally
- `port` – the port under which the connection to the MySQL server shall be established, `0` by default
- `databasename` – the name of the database, enclosed in single quotes

Finally, create an empty folder `stationdata` in the dataset directory. Make sure that Validator has write access to this folder, as it will be used to store cached data downloaded from the database.

4.1.6 Connecting to your own database

If your own database looks different, you need to do the following steps:

1. Find out how to connect to the database from R.
2. In addition to `standard` and `sqldb`, define another dataset type, e.g. `mydb`. Duplicate the parts of the code where `sqldb` is used in the following files:
   - `do_the_plot.R`
   - `model_validation_plot.R`
   - `read_measured_data.R`
3. In the duplicates, replace `sqldb` by `mydb`.
4. In `read_measured_data.R`, modify the function `read_measured_data_mydb` to fit your database.

4.2 Providing model data

4.2.1 The model list

The file `models.txt` describes the models which can be compared by Validator. it shall contain a header row
`name;path`
and one line for each model to be compared, where the following is provided separated by semicolons and without quotes:

- `name` – the name of the model which shall appear in the model list

- `path` – a path to the directory where the NetCDF files for model validation are stored, absolute or relative to the Validator base directory

Make sure Validator has read access to the files in this folder.

4.2.2 Providing NetCDF files

For each oceanographic station, provide one NetCDF file named `stationname.nc` where `stationname` is the **lowercase** station name corresponding to the file `stations.csv` in a dataset directory. It shall contain two dimensions:

dimension name	data type	attribute	value
`time`	double	units	days since 1899-12-30 00:00:00
`zaxis`	double	units	meters
		positive	down

In fact arbitrary axis names are allowed if they start with `t` or `T` / `z` or `Z`, respectively. Additional axes are allowed as long as they do not start with any of these letters. Also, the time unit or the date origin can be different and will be converted automatically. The values at the *z* axis should be zero at the surface and increase with depth.

For each variable in the data file, there should be a NetCDF variable defined with the name
`MODEL_VARNAME` where `VARNAME` is the **uppercase** variable name which matches the file `variables.csv` in a dataset directory, e.g `SALIN`. The variable shall have the *z* and *t* dimensions defined above, and possibly other dimensions of length one (e.g. longitude and latitude). It can have data type float or double.

So, there are some limitations:

- The data need to be on fixed vertical levels. If your vertical model coordinates are not fixed, you will require z level interpolation.
- Axis bounds are ignored, so cell thicknesses cannot be defined. If your z axis is not equidistant and you want precise values for vertical averages (e.g. when plotting a time series of salinity averaged over a large depth range), you should consider regridding.

4.3 Providing data for the station maps

The locations of the stations are provided in the measurement dataset. To draw them onto a map where they can be selected by clicking, two more pieces of information are needed:

- The coordinates of the map, and
- the coastline.

A set of map regions can be provided in the file `regions.txt` in the Validator main directory. It shall start with the line

```
name;latmin;latmax;lonmin;lonmax;coastline
```

and then have one line for each map view, containing the following separated by semicolons and without quotes:

- `name` – the name of the region as shown on the radio button by which it can be selected
- `latmin` – latitude of the lower map boundary, in degrees, positive in the northern hemisphere
- `latmax` – latitude of the upper map boundary, in degrees, positive in the northern hemisphere
- `lonmin` – longitude of the left map boundary, in degrees, positive in the eastern hemisphere
- `lonmax` – longitude of the right map boundary, in degrees, positive in the eastern hemisphere
- `coastline` – the path to a file containing the coordinates of coastline segments, absolute or relative to the Validator main directory

The same coastline file may be used for different map views. A coastline file is structured as follows: It shall start with the line

```
longitude;latitude
```

and then have one line for each endpoint of a coastline segment, giving longitude and latitude in degrees, separated by a semicolon. Consecutive points will be connected with lines unless they are separated by a row

```
-1000;-1000
```

which marks a break between individual polylines. These breaks are typically required to define island coasts which are separated from the mainland coastline.

5 Example data

The example data which are provided with Validator are in-situ measurements of temperature and salinity. They were extracted from the IOW database. The data request to the IOW database can be repeated using the following link:
`https://odin2.io-warnemuende.de/980-0706-377`
The data are licensed under the CC BY 4.0 license, just like the Validator itself.