A Roadmap for improvement of the Verification and Validation of the NEMO ocean model.

Draft v1.0 - November 2020

Authors (alphabetical order):
Mike Bell, Massimiliano Drudi, Jerome Chanut, Andrew Coward, Dorotea Iovino, Julien Le Sommer, Claire Lévy, Nicolas Martin, Sébastien Masson, Pierre Mathiot, Simon Müller

Summary :
We propose a roadmap for improving the procedures and tools used to guarantee the continuous improvement of the reliability and scientific quality of the NEMO ocean model. The roadmap is based on a description of the tools and processes we currently use, opportunities to improve them, plans for short term and long term (by 2024) improvements and practical actions for the coming years.

1. Introduction

Our aim is to develop a road-map for improvement of the NEMO tools and processes for verification that is well-founded, practical and builds on good ideas from others.

Verification is about confirming that a code does what it is intended to do (mathematics & logic). This includes:

- Unit testing (thorough checks that a “small” unit behaves as intended for any input)
- Functionality (that the system “works” as intended for all its envisaged configurations)
- Restartability (results after a restart are the same as for a continuous run)
- Reproducibility (e.g. results don’t depend on domain decompositions)
- Bit comparison when expected across versions (regression testing)

We distinguish two types of validation. The first is about confirming that the scientific quality is as expected. This includes, for instance, checking that model solutions are adequate in terms of :

- representing specific physical processes (eg. propagation and trajectory of an eddy structure, dispersion of internal waves, mixing in overflows)
- preserving integrated properties of oceanic flows (usually derived from first principles) in a variety of flow configurations (energy conservation, mass conservation)
- guaranteeing any other specific property of the discrete model that is considered useful (eg monotonicity/TVD property of advection schemes)

A second type of validation is defined in terms of measuring or ensuring the overall skill/quality of a system that simulates or predicts the “real-world” (e.g. by comparison with observations/measurements) . This latter activity depends on the purpose of the prediction/simulation system. It is very important but outside the scope of this discussion because it is not the responsibility of the NEMO System Team or the NEMO Working Groups. Input from teams that perform this latter validation does however need to be part of the NEMO development cycle and this type of validation should ideally be performed on a new model version prior to its release.

Key objectives for NEMO’s V&V include:

- to ensure that the NEMO code base is a reliable and efficient implementation of its scientific formulation as described in the NEMO documentation
- to ensure that the NEMO code meets specific scientific criteria in idealised test cases
- to achieve the above as efficiently as possible.

In order to meet the first objective the tests performed as part of V&V on a new release must demonstrate:

- correct implementation of scientific formulation
- restartability and repeatability (bit reproduceability)
- that the code functions as expected for the different use cases and scientific options
- that the computational performance is acceptable in some standard configurations

To meet the second objective it is expected that an extendable test environment will be made available that checks whether the code meets specific scientific criteria in a suite of idealised test cases.

The V&V processes and systems should be designed to make the most efficient use of the time of NEMO System Team members, of other NEMO developers, and of the users of the NEMO model. More specifically they should aim to

- minimise overall staff effort (both within and outside the NST)
- make clear what tests have been done and what the results are
- make clear what has changed in a new release from the previous one

- be easy to follow / use
- test the right things
- avoid duplication of effort.

The sections of the document are as follows:

1. Introduction: describes our aim, what we mean by verification and the structure of the document
2. Overview of standard methodologies and terminology; the objective is to provide the context and a broad conceptual framework for verification
3. NEMO's current tools & processes: a description of what we have now
4. Short-term practical steps to improve what we have now
5. Potential tools & processes: methods deployed in other systems; ideas for unit testing
6. A road-map: describing medium-term goals and short-term actions consistent with them.

Ideally the ideas described in sections 4 and 5 would be fully scoped before writing section 6. It has proved to be impossible however to do the scoping required in the margins of other activities. So the roadmap in section 6 currently consists of an initial set of tasks, including some scoping tasks, that are proposed to be included in the NEMO 2021 Work Plan.

In most sections we discuss how verification relates to two sets of tasks:

1. design, development and testing of a new science option or a new functionality
2. system integration and testing

The first set of tasks can be undertaken by individuals or small teams, usually working on a small subset of the code (one or two modules). Some members of these teams need not be members of the NEMO System Team (NST). The second set of tasks is the responsibility of the NST. It involves the integration of changes developed within the first set of tasks and tests of the consistency and performance of a given version of the code system.

2. Overview of standard methodologies and terminology

2.1 Design, development and testing of new capability

2.1.1 Position of verification and validation within the development process

The development process of any major code system is designed to ensure that the developments submitted to it have been properly designed, reviewed and verified. It's important that the process is both followed by developers and sufficiently rigorous to avoid introducing errors that can waste a lot of time during integration or remain undetected for a long time. Intelligent use of the process is important to maintain and improve the quality of the system. The design and testing processes are clearly inter-linked but to avoid loss of focus we do not explore that here. A separate document discusses the NEMO development process.

2.1.2 Unit testing

A good summary is at http://softwaretestingfundamentals.com/unit-testing/ . A commonly used framework for unit testing in Fortran is https://github.com/Goddard-Fortran-Ecosystem/pFUnit . This framework is used by FESOM2.

There is little visible use of unit testing within NEMO at the moment. Presumably developers print out a lot of information whilst debugging the code but no record of what testing has been done and what the results are is logged within the NEMO system. It is generally considered to be difficult to introduce unit testing retrospectively into a system but Mark Petersen has developed a simple method for supporting useful unit tests within MPAS (see section 2.2.2).

2.1.3 Use of idealised configurations (scientific validation)

It has recently become much easier to run NEMO with idealised configurations. There is clear scientific value in comparing results for suitably chosen configurations with those generated by other groups and publishing the results in the scientific literature. The COMMODORE group has proposed to do work of this sort. Where well established test cases exist it is clear that they should be used. The scientific literature is the best place for these results but some useful results may not be easy to publish. Development of new idealised tests is a scientific research activity. The NEMO development process now requires new code to be accompanied by an idealised test case.

2.1.4 Real-world configurations

The scientific performance of code within "real-world" configurations also needs to be assessed. At a minimum this includes the impacts on the scientific results (validation) and the computational cost. Some developers will not have access to "real-world" configurations and there are no clear expectations about testing and documenting the impacts of NEMO code developments on real-world configurations. Validation in real-world configurations is considered outside the scope of this document.

2.2 System integration and testing

2.2.1 Introduction

Much of system integration and testing is concerned with building a new version from the previous one and testing that the new code works as expected and does not interfere with the previous results.

In a new system one would expect the system integration to ensure that at least some of the unit tests work within the integrated system. The system tests should include regression testing of these unit tests.

2.2.2 Types of tests

As mentioned in the introduction there are various types of tests:

- Repeatability: the code gives the same results when run on different numbers of processors on the same machine (with same compiler …)
- Restartability: stopping and restarting an integration does not the change results
- Reproduceability: usually means different groups obtain the “same” results (what does it mean here?)
- Regression: the results for a new version agree with the previous ones in cases where agreement is expected
- Performance: the computer resources used are as expected
- Invariance properties: Certain “symmetries” of the system are reproduced (MOM6 has several tests of this sort)

2.2.3 Processes

New releases of software are released periodically but integration of new developments to the head of the trunk take place continuously and multiple branches from the trunk are also updated continuously. The processes by which it is verified that the new releases or the head of the trunk pass an agreed set of tests need to be efficient and clearly articulated so that all the members of the NST are able to follow them (see section 3.1).

2.2.4 Tools

Excellent tools for code development are now available. NEMO has used the svn and trac systems for many years. Git is a popular alternative. Many sites compare svn and git and discuss their pros and cons e.g. https://walty8.com/comparison-of-git-and-svn/ . Moving to a new tool is an investment of time and effort. So there has to be good reasons for making a change. Git and github are still being developed and supported whilst svn and trac are viewed as outdated by most software development companies. These days, more developers new to

NEMO are familiar with git than svn. Github also has a range of tools to support continuous integration practices.

The NEMO SETTE tool exercises a set of standard configurations and enables the results to be checked against those from a previous version. The Trusting tool is intended to run SETTE on a routine basis and provide alerts when the results are not acceptable. NEMO has many options and SETTE only tests a small number of combinations of them.It would be possible to use job scheduling tools like cylc with light-weight configurations to exercise a much wider combination of options.

3. Status of existing tools and procedures

3.1. Development procedures (PM and CL)

3.1.1. Bug fixes

- Open a ticket about a bug
- Run SETTE script before fixing the bug (in order to have a reference SETTE result) or create a branch to fix this bug.
- Fix the ticket in your trunk copy or in a separate branch
- Can be reviewed if the ticket owner feel the need for it
- Run the testing toolbox SETTE
- If successful, the fix is committed to the trunk and the ticket is closed

For all details, see: https://forge.ipsl.jussieu.fr/nemo/wiki/Developers/WorkingOnTickets

3.1.2. Yearly work plan

- Describe work, associated test case and development plan in the wiki page
- Validation of the development plan and test case during the preview stage
- Coding development and test case
- Validation (SETTE test and other validation test described in the development plan)
- Review of the code, its test case, documentation on line and in reference manual
- Integration of the development into the trunk during the merge party

For all details, see: https://forge.ipsl.jussieu.fr/nemo/wiki/Developers/DevelopingCodeChanges

3.1.3. Release

- Major release:
  Once sufficient novelty has been added to the model or a need of a stable and maintained code is needed for a long period of time (CMIP exercise for example), a major release is created. It usually does not occur straight after a merge party but during the year after sufficient testing has been done on multiple platforms, with multiple

configurations at various resolutions over various time-scales (decade to century). A beta release open to experienced user is usually done before the final release. The validation is usually done by the consortium institute that run the simulations and there is no common validation procedure for such simulation across the institutions.

- Minor release:
  During the merge party 2019, it has been decided that the released version are now on tag (ie no bug fixes will be done in a released version). All bug fixes affecting a supported version will be fixed in a 'v4 trunk' (need to find a name) and once significant improvement has been done and agreed by the system team, a tagged version will be released.

As a general rule, **every commit that is made to the trunk should be SETTE tested before being committed**.

3.2. Status of the testing tool SETTE (PM and CL)

SETTE is based on 2 shell scripts. The first one sets up, compiles and runs basic test simulations and then a second script tests if the SETTE tests are successful or not. The tests carried out are:

- Restartability
- Reproducibility of the results with different MPP decomposition
- Comparison of the results with a reference revision if relevant

These tests are done by comparing a summary of the model state (max ssh, max |U|, max/min sss) every time step. A test is called successful if the model state between the 2 simulations is fully identical for all the common time steps. Each sette test directory is associated with a specific branch.

These tests are applied to various configurations in order to assess various parts of the NEMO code (sigma coordinate, ice shelf, iceberg, sea-ice, AGRIF, bdy**,** biogeochemisty (TOP and PISCES) ...) .:

- the reference configurations (ie configurations maintained by the NEMO system team)
- the test cases (ie some of the simple configurations to study specific oceanic physical processes)

At the end, a report is generated summarizing the results of the tests for all these configurations. All the results are saved in a separate directory for each compiler and revision sette is run with.

Once the startup file is set up correctly, it is straightforward to run it for a specific branch or configuration. Some utility scripts are available to check what results are available (what configuration and what revision) and to retrieve these results.

Open Questions on the purpose of SETTE:

- for some configurations, sette runs also multiple other simulations (different advection scheme, …) but no tests are made on the results (is it the purpose of sette ?)
- Is sette supposed to at least test exactly the reference configuration ? (the ORCA2 tested is not exactly the one defined in cfgs)

4. Short-term practical steps for improving current tools & processes

4.1 Integration testing

We describe below what is not working or annoying in SETTE. Many of the proposed actions could be considered as short term, does not require a full re-write of SETTE and are independent of one another. However tackling all of these suggestions could take a large amount of time. Furthermore, before starting we need to agree on the exact purpose of SETTE (see open questions above in section 3.2) and also be sure that the long term plan does not require a re-write of SETTE.

In most cases the SETTE tool works well, however, in some specific cases, it can be annoying to run or extend to new configurations. There are 3 types of short term actions: ones affecting the submission script (script to compile and run simulations), ones affecting the script which compute the final report on the SETTE test, and more general actions :

- **Submission script**
    - Script to download (or simply describe the zenodo paths) the input file tarballs are not available in the SETTE repository. This will lead to sette failure and location of the zenodo path will have to be found.
    - **Make sette universal** (independent of the configuration). Today, adding or testing a new configuration is not straight forward and it will be harder and harder to maintain. For example, in case a user wants to setup sette for one of its configurations, ~150 lines of code need to be added and 4 files to be modified. This is because many configuration dependent names, parameters are hard coded in sette as the input_CONFIG files, the namelist parameters, the time decomposition and processor decomposition.
    - **Optimise the number of runs**. 1 useless simulation is run for each configuration tested. Currently we run one LONG and one SHORT simulation then 2 simulations with different decompositions. So it could be organised differently

with the LONG one being the reference and the SHORT and one REPRO being compared against the REF. Furthermore, sette is used to run also multiple test (sensitivity of the advection scheme …) for some specific test-cases not used in the report which are useless in the current state of the report script.
    - The previous 2 points, makes it hard to extend the number of simulations/configurations to run, to maintain it and add any new functionality.
    - **Make the full sette test faster.** The full sette test from scratch is slow to run mainly because
        - the configurations are compiled one after the other. In order to improve this, there are 2 ways: decrease the number of compilation (in fact most of them are compiling the same code because of the very limited number of cpp keys) or compile the code in parallel.
        - The number of time step for each configuration (can reach 1000 steps). To reduce the number of time step, a better way of comparing simulation is needed. Currently, we only comparing the min/max of some variables. We probably need a reliable checksum of the full 3d array of u,v,t,s and ssh to find differences much earlier. Decreasing the number of time step will also make sette in debug mode easier and more reliable.
          (first guess of an algorithm: see isfutils, it contains a debug tools used with success to debug isf and icb. It catch a change in operation order in icb much earlier in the process than classic mpp_sum/max/min)
    - **Give an option to run sette in debug mode.** Sette is not run routinely on debug mode because it is too slow to run (not to compile, compilation in debug mode is quite fast) with these compiling options. So an option to run only a few time steps in debug mode for at least MAX(nn_fsbc,2) time steps (step 1 is a bit special as it is an euler time step) could lead to an easier find of bug unseen with the default compiling options.

- **Report script**
    - Reproducibility of all the netcdf output variables after a code change is not tested.
    - Tracer and volume conservation are not tested (despite conservation level being available in NEMO as output)

- **General**
    - A full list of exactly what is tested is not available, so it is really hard what % of the code is really tested. For example the coupling interface is not tested.
    - Find a way to manage different branches within a single directory. Currently if you develop 2 branches, you need 2 sette directory.
    - To build a new configuration, NEMO relies on the tools. So the tools should be tested too.

One achievement for :

- An easier use of SETTE
- A wider use of SETTE
- A simplification of the dev process
- A more robust code

will be to be able to run './sette -t MY_DEV_CONFIG' with forcing in an external directory and an input file describing all the specifications in MY_DEV_CONFIG if you want to test a homemade configuration and that's it. A detailed description about what need to be done in order to achieve an independent SETTE submission script is detailed here: https://forge.ipsl.jussieu.fr/nemo/attachment/wiki/SystemTeam/Agenda/2020-02-27/ST_sette.pdf , ticket #2417 and pseudo code available in branch: utils/CI/sette_ticket2417

4.2 Continuous Integration testing

Trusting tool

The so-called trusting tool has been developed by Nicolas Martin in 2015 and has been running up to 2017. As an insight on the functionalities of this tool, it was producing the following web pages (with two different options below):

- New interface on https://pagesperso.locean-ipsl.upmc.fr/ntmlod/trusting_dev
- Older interface on https://pagesperso.locean-ipsl.upmc.fr/ntmlod/trusting

This tool needs a few updates to be functional for the most recent NEMO releases, especially in relation with the changes in the tree structure of NEMO svn repository.
Once this done, hopefully on a few centers/target computers to check portability, this action should also allow to easily add new configurations or test cases to the trusting process.
This update work is now underway (action taken by Claire Lévy for now):http://forge.ipsl.jussieu.fr/nemo/wiki/2020WP/VALID-12_clevy_Trusting_ContinuousIntegration

Proposition for short term continuous integration testing that is easy to set up:

Currently, until trusting has been fully updated (action expected to be completed for end 2020), proper continuous integration needs multiple human actions. It requires that every time a revision is made:

- 1. Check if new revision is available. (*svn info*)
  If yes, move to next step:
- 2. Update the trunk or a specific branches to the head. (svn up)
- 3. Set up the new reference revision to check if simulation results changed (update NEMO_REV_REF)
- 4. Run sette.sh (ie submit all the run and generate the report) (./sette.sh)
- 5. Assess the report: is SETTE pass ? is the results the same ? (grep FAILED)

If no, move to next step:

- 6. Take action if needed (send an email to the system team, the owner of the last commit, …). This point is probably the key issue and need to be tackled with care.

Assuming SETTE is set up correctly. The action 1 to 5 can each be easily replaced by a shell function and the automation can be done via a cron job. This could easily be deployed on each consortium member's cluster or HPC.

The bottleneck is what to do when an action is required (ie SETTE test failed on one of the HPC infrastructure or one of the compilers, unexpected differences in results ...).

- A 'harsh' solution would be to revert the change and put it back later when a proper fix is found (could require a lot of extra work for various institutes in communication with the developer and testing on other computer than the developer's computer).
- A 'lazy' solution would be to wait troubles to happen, see how we fix it and assess how we can make this a bit more automatic.
- Other solution ? Probably out of scope for 'short term plan'

Aside from evolution of the tool to run automatically the sequence of tests and produce a readable summary of the results, the core questions are more scientifically related. For each unit test, one needs to make a relevant choice for its "verification value":

Each development implementing a new functionality should now include its associated test case. A test case is a simple and light setup of NEMO both in terms of computing resources (a single proc application if it makes sense), and input files (none except the namelists would be the best).

This setup must allow demonstration of the developed functionality, and - most important - to automatically test the result through a logical (true or false) or a simple mathematical test (some mean value which should stand within a give range, for example). This "verification value" will allow to integrate the test case in the NEMO automatic testing suite, and to check if the functionality is correct, for each given version of NEMO and/or of a development branch in the future.

5. Potential tools and processes used in other systems

5.1 Unit testing

It would be a lot of work to set up a comprehensive set of unit tests for NEMO. However it would be useful to make a start by investigating how in practice unit testing could be incorporated into NEMO in a compact and sustainable manner. This task could focus on prototyping for one "typical" module (e.g. for the equation of state or the horizontal pressure gradient). It could start by attempting to adapt the approaches used by FESOM2 and MPAS for NEMO. The MPAS

approach is outlined in the appendix. This study should include an assessment of the costs and value of further work and priorities for its introduction.

Developers should be encouraged to supply unit tests for new developments through their test cases. The unit tests should become part of the regression testing within the NEMO continuous integration process.

5.2 MOM6 approach to testing of invariance properties

The options for incorporating the tests of invariance properties devised by Bob Hallberg that were presented at the Jan 2020 Commodore into the NEMO regression test suite should also be considered.

5.3 Potential to test a wider range of combinations of code options

As mentioned in section 2.2.4 SETTE does not exercise many of the combinations of options available within NEMO. An analysis of the pros and cons of using cylc or other schedulers to exercise a wider range of combinations could be usefully undertaken.

6 A Roadmap for development

The tasks on V&V proposed for the NEMO 2021 Work Plan are :

1) To scope out how in practice unit testing could be incorporated into NEMO in a compact, extensible and sustainable manner. This task will probably include prototyping with one “typical” module (e.g. eos or hpg). Sibylle Techene has past experience in unit testing so is best placed to take on this scoping.
2) To scope out the costs of transition to git/github: this includes assessment of the options, how it would be done, how to mitigate cons (from pros/cons), and the effort required. Nicolas Martin is best placed to lead this scoping; he is expected to return to work in January.
3) To scope out the prospects for containerisation of SETTE tests. CMCC or NOC are best placed to do this task.
4) To scope out whether cylc or other tools could be flexibly and sustainably used to test a wider range of permutations of NEMO namelist options. A prototype of this idea exists so the task is to consider how the idea could be realised in a flexible and sustainable manner, the benefits, and the ongoing costs. The Met Office is best placed to do this task.
5) To start work on short-term improvements to SETTE. Andrew Coward will propose what this task should include.
6) To re-establish the Trusting tool at one or two institutes and to propose and trial a process for resolving issues revealed by it. (CNRS with at least one other consortium member)

7) To update this roadmap at the end of 2011 to reflect conclusions from the above tasks.

Appendix Description of MPAS approach to unit testing

Within the MPAS system, it is possible to disable all but a small number of modules and run the modules with ‘artificial’ inputs for which the outputs are known:

An example with the Redi term verification is available at

https://github.com/MPAS-Dev/MPAS-Model/pull/280#issuecomment-576527278

The python script using sympy (symbolic python) to create the analytic functions and solutions is:

https://github.com/MPAS-Dev/MPAS-Model/blob/ocean/develop/testing_and_setup/compass/ocean/tendency_verification/all/Redi/polynomial_validation.py#L29

The test cases are set up using :

https://github.com/MPAS-Dev/MPAS-Model/blob/ocean/develop/testing_and_setup/compass/ocean/tendency_verification/all/Redi/add_initial_state.py#L182