Stock Assessment Software Catalogue

Contents

1 Objectives

An action under the SCRS Strategic Plan is to "Consolidate the stock assessment catalogue to ensure the best use of models that should be fully documented". The objectives of this document are to

- Reinvigorate the ICCAT Software Catalogue as required under the Strategic Plan.
- Encourage software development and innovation; while ensuring reliability, stability, auditability, accountabilty and supportability of software.
- Ensure procedures are consistent with best practice elsewhere, i.e. that of other RFMOs and bodies responsible for developing advice based on software.

Steps

1. Contact rapporteurs of assessment WG with a summary of the old requirements and issues that have arisen since the establishment of the Software Catalogue, e.g. related to the Strategic Plan, Kobe advice framework, SISAM/WCSAM, recent assessment and the evaluation of Management Procedures (MP) using Management Strategy Evaluation.
2. Ask rapporteurs to review if the old requirements are still adequate or need updating and to propose a set of revised requirements.
3. Ask rapporteurs to use these new requirements to "certify" ASPIC.
4. In parallel canvass views of software developers since if the process becomes too ***burdensome*** then no software will be developed.
5. Canvass views of other RFMOs and bodies that use stock assessment methods.
6. Presented results of the exercise to the SCRS which would approve the new protocol.

2 Strategic Plan

Action **1.3 Consolidate the stock assessment catalogue to ensure the best use of models that should be fully documented**

Strategies

1.3.1 Update the current Stock Assessment Catalogue [1] to remove outdated software and update the software versions that are currently being used.

1.3.2 Ensure that all software used in the most recent assessments are matched up with the versions in the catalogue.

1.3.3 Ensure that software is well documented and have an accompanying user's manual and code.

[1]`https://www.iccat.int/en/AssessCatalog.htm`

Table 1: Software in Catalogue

Catalogued Software			
Package	Version	Author	Description
PRODFIT	1.0	Alain Fonteneau	Equilibrium Biomass Model
ASPIC	5.05	Mike Prager	Biomass Dynamic Model fitted using maximum likelihood
ASPIC	3.82	Mike Prager	Biomass Dynamic Model fitted using maximum likelihood
BSP2	3.0	Murdoch MacAllister	Biomass Dynamic Model fitted using Bayesian simulation
VPA-2Box	3.01	Clay Porch	Virtual Population Analysis fitted using maximum likelihood
Pro-2Box	2.01	Clay Porch	Projection for VPA-2Box
FSIM	3.0	Phil Goodyear	A general purpose fish population simulator designed to simulate many forms of fisheries data routinely collected from real fisheries
SEEPA	3.0	Phil Goodyear	Simulates longline catch and effort data to test the robustness of the habitat approach to cpue standardization.

Measurable targets

Reactivate the Working Group of the Stock Assessment Catalogue and review the protocols of inclusion and updating the software used for stock assessments while maintain a historic repository of version control

3 Catalogue

4 Tables

Stock assessments underpin the scientific advice for management that is provided by the Standing Committee on Research and Statistics to the Commission. In recent years, the SCRS has implemented a number of activities whose purpose is to improve the quality of this advice. Part of this effort is the so-called "software catalogue". The aim of the catalogue is to document the procedures taken to validate some of the stock assessment programs that are commonly used by the various working groups.

Inclusion of a particular computer program in the ICCAT software catalogue does not guarantee that the software is free of bugs, nor does it imply any sort of institutional endorsement for its use. Inclusion in the catalogue is simply a way of documenting what steps, if any, the programmer has taken to ensure that the program does what it purports to do. Recent problems, even for software in the catalogue, have included different results being obtained when assessment are rerun due to lack of appropriate diagnostics, numerous versions of source code and differences between compilers and operating systems.

However, the changing requirements of the Commission and the development the demands for new scientific approaches means that the original objectives of the catalogue may need to be reviewed.

5 Proposal

Therefore it is proposed to review the protocols of inclusion and updating the software used for stock assessments while maintaining a historic repository of version control. The 1st step is to canvass rapporteurs of stock assessment working groups

5.1 Objectives

The objectives, as given in the Strategic plan are

Update the current stock assessment catalogue to remove outdated software and update the software versions that are currently being used.

Version Control Ensure that all software used in the most recent assessments are matched up with the versions in the catalogue.

Documentation Ensure that software is well documented and have an accompanying user's manual and code.

5.2 Issues

What about software that is extensively used by other RFMOs and management bodies and has been used to provide advice by ICCAT WGs but is not in the catalogue. E.g. ICES and XSA [Shepherd, 1999], WCPFC and Multifan-CL [Fournier et al., 1998], IATCC and SS [Methot, 2005].

- Do developers have the time or be interested in going through an ICCAT certification process?
- Is a peer review paper good enough?
- Could the review committee complete the application?

Should the source code be open source e.g. GPL[2]? Since the more people who can see and test a set of code, the more likely that bugs and flaws will be caught and fixed quickly [Raymond, 1999]. Benefits include

[2]http://www.gnu.org/licenses/gpl.html

- Open source software is better at adhering to standards than proprietary software, e.g. allows recompiling for running on clusters and that algorithms produce the same results across operating systems. With closed source software, you have nothing but the authors claims telling you that they're adhering to standards.
- Code can be modified to meet specific needs and encourages developers to work in parallel, so that a best solution can be chosen instead of the only solution.
- Where source code is open source users of the product will often discover and correct defects themselves.
- Developers are likely to consider reducing the complexity and improve the maintainability of software.

This requires a version control system.

5.3 Procedure

The latest version of ASPIC[3] is version 7.01, there are two versions in the ICCAT catalogue i.e. 3.82 and 5.05; while the version on the NOAA Fisheries Toolbox is 5.34.9. There is also an R version developed under the GBYP [Merino et al., 2013].

6 Stock Assessment

6.1 International Initiatives

The Strategic Initiative for Stock Assessment Methods (SISAM) is designed to ensure that scientists apply the best stock assessment methods when developing management advice. The first stage culminated in the World Conference on Stock Assessment Methods (WCSAM) and a simulation-based workshop to evaluate performance of stock assessment methods. The second stage involves continued coordination with RFMOs and national agencies to development *good practice* guidelines and further evaluation of model performance.

WCSAM included a workshop on testing assessment methods using simulations based on datasets from 14 representative fish stocks from around the world; one of which was North Atlantic albacore. Two types of simulations were used i.e. self-testing and cross-testing [see Deroba et al., 2014].

6.2 Testing

There are a variety of approaches for testing stock assessment software e.g.

Self testing: a model is first fitted to data, then psuedo data are simulated based on the fit and the same model is refitted [e.g. Lee et al., 2011]

Cross testing: again a model is fitted to data and psuedo data generated but a different model is then fitted.

[3]http://www.mhprager.com/aspic.html

Simulation: where psuedo data based on a variety of assumptions about the dynamics are simulated for fitting to a model but without fitting a model to data first.

Cross validation a model is fitted to only part of a time series of data, then the dynamics are projected forward and compared to with model fits to the entire time series [e.g. Patterson et al., 2001].

MSE: the stock assessment method is tested as part of a Management Procedure (MP) using Operating Models, which may or may not be based on fits to data [Kell et al., 2006].

7 Software

7.1 Testing

Testing is a vital part of software development as it ensures that code does what it is intended to. Unit tests are simple to write, easily invoked, and help the software development process, from early stage exploratory code, to late stage maintenance of a long-established project.

For example the Bioconductor[4] project that provides tools in the form of R packages for the analysis and comprehension of genomic data. Unit tests are a standard part of the Bioconductor build process uses the RUnit package to write unit tests. There are other R package for developing unit tests e.g. testThat[5]. All available from CRAN[6] a network of ftp and web servers that store identical, up-to-date, versions of code and documentation for R.

Benefits of testing include

Fewer bugs . Since it is explicit about how code should behave there will be fewer bugs. The reason why is a bit like the reason double entry book-keeping works: because you describe the behaviour of your code in two places, both in your code and in your tests, you able to check one against the other. By following this approach to testing, you can be sure that bugs that you've fixed in the past will never come back to haunt you.

Better Code Structure Code that's easy to test is usually better designed. This is because writing tests forces you to break up complicated parts of your code into separate functions that can work in isolation. This reduces duplication in your code. As a result, functions will be easier to test, understand and work with (it'll easier to combine them in new ways).

Easier Restarts . If you always finish a coding session by creating a failing test (e.g. for the next feature you want to implement), testing makes it easier for you to pick up where you left off: your tests will let you know what to do next.

[4]`http://www.bioconductor.org/`

[5]`http://r-pkgs.had.co.nz/tests.html`

[6]`http://cran.r-project.org/`

Robust Code If you know that all the major functionality of your package has an associated test, you can confidently make big changes without worrying about accidentally breaking something. For me, this is particularly useful when I think I have a simpler way to accomplish a task (usually the reason my solution is simpler is that I've forgotten an important use case!).

7.2 Version Control

Version controlis a system that records changes to a file or set of files over time so that you can recall specific versions later, e.g. track changes in MS Word. A Version Control System (VCS) allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover.

A simple version-control method is to copy files into another directory (time-stamped if you are wise). This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to. To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control. The next major issue that people encounter is that they need to collaborate with others. Therefore, Centralized Version Control Systems (CVCSs) were developed. These systems, such as CVS[7], Subversion[8], and Perforce[9], have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this was the standard for version control. This offers many advantages, especially over local VCSs. For example, everyone knows to a certain degree what everyone else on a project is doing. Administrators have fine-grained control over who can do what; and it's far easier to administer a CVCS than it is to deal with local databases on every client.

However, this setup also has some serious downsides. The most obvious is the single point of failure that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on. If the hard disk the central database is on becomes corrupted, and proper backups haven't been kept, you lose absolutely everything – the entire history of the project except whatever single snapshots people happen to have on their local machines. Local VCS systems suffer from this same problem – whenever you have the entire history of the project in a single place, you risk losing everything.

Therefore Distributed Version Control Systems (DVCSs) was developed. In a DVCS (such as Git[10], Mercurial[11], Bazaar[12] or Darcs[13]), clients don't just check out the latest snapshot of the files: they fully mirror the repository. Thus

[7]`http://www.tortoisecvs.org/`

[8]`https://subversion.apache.org/`

[9]`http://www.perforce.com/`

[10]`http://github.com/`

[11]`http://mercurial.selenic.com/`

[12]`http://bazaar.canonical.com/en/`

[13]`http://darcs.net/`

if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.

Furthermore, many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project.

7.3 Validation

Software validation is a complex issue. The U.S. Food and Drug Administration (FDA) takes *The Least Burdensome Approach*[14]. This does not prescribe specific practices, tools, coding methods or any other technical activity. Instead organisations determine, and then strictly adhere to their self-defined validation and verification processes.

However, development activities and outcomes must be clearly defined, documented, verified, and validated against an organisation's process. The goal of this approach is to give medical device makers enough rope to determine how to best ensure public safety. But in practice, the effect has been that organisations have enough rope to hang themselves. This is because the requirements, expressed in the relevant Federal Regulations , represent extensive planning and testing, which require validation.

[14]`http://www.fda.gov/downloads/RegulatoryInformation/Guidances/ucm126955.pdf`

References

J. Deroba, D. Butterworth, R. Methot, J. De Oliveira, C. Fernandez, A. Nielsen, S. Cadrin, M. Dickey-Collas, C. Legault, J. Ianelli, et al. Simulation testing the robustness of stock assessment models to error: some results from the ices strategic initiative on stock assessment methods. *ICES Journal of Marine Science: Journal du Conseil*, page fst237, 2014.

D. A. Fournier, J. Hampton, and J. R. Sibert. MULTIFAN-CL: a length-based, age-structured model for fisheries stock assessment, with application to south pacific albacore, thunnus alalunga. *Canadian Journal of Fisheries and Aquatic Sciences*, 55(9):2105–2116, 1998.

L. Kell, J. A. De Oliveira, A. E. Punt, M. K. McAllister, and S. Kuikka. Operational management procedures: an introduction to the use of evaluation frameworks. *Developments in Aquaculture and Fisheries Science*, 36:379–407, 2006.

H.-H. Lee, M. N. Maunder, K. R. Piner, and R. D. Methot. Estimating natural mortality within a fisheries stock assessment model: an evaluation using simulation analysis based on twelve stock assessments. *Fisheries Research*, 109 (1):89–94, 2011.

G. Merino, P. Bruyn, L. Kell, and J. Scott. A preliminary stock assessment of the albacore tuna (thunnus alalunga) stock in the northern atlantic ocean using a non-equilibrium production model. *ICCAT Collect. Vol. Sci. Pap.*, 69(56):xxx–xxx, 2013.

R. D. Methot. Technical description of the stock synthesis II assessment program version 1.17-March 2005. *Unpublished draft report provided on the CD-ROM of background materials for the STAR. NOAA Fisheries, Seattle, Washington, USA*, 2005.

K. Patterson, R. Cook, C. Darby, S. Gavaris, L. Kell, P. Lewy, B. Mesnil, A. Punt, V. Restrepo, D. W. Skagen, et al. Estimating uncertainty in fish stock assessment and forecasting. *Fish and Fisheries*, 2(2):125–157, 2001.

E. Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49, 1999.

J. Shepherd. Extended survivors analysis: An improved method for the analysis of catch-at-age data and abundance indices. *ICES Journal of Marine Science: Journal du Conseil*, 56(5):584–591, 1999.

8 Appendix

INTERNATIONAL COMMISSION FOR THE CONSERVATION
OF ATLANTIC TUNAS

COMMISSION INTERNATIONALE POUR LA CONSERVATION
DES THONIDES DE L´ATLANTIQUE

COMISION INTERNACIONAL PARA LA CONSERVACION
DEL ATUN ATLANTICO

ASSESSMENT PROGRAM DOCUMENTATION

Program: **xxx (ver. x.x.x)**
Name

Current Catalog Entry: **?**
First Cataloged by ICCAT: **?**

Catalogue Committee
External:

ICCAT Secretariat:

NOTE: As part of its efforts to carry out Quality Management, ICCAT´s Standing Committee on Research and Statistics is developing a catalog of stock assessment applications. The purpose of the catalog is not to evaluate the relative merits of various assessment methods, but rather whether the software implementing the method works as intended and is adequately documented.

ASSESSMENT PROGRAM DOCUMENTATION

1. Program name
xxx

2. Version (date) ***
Version x.x.x

3. Language

4. Programmer / contact person

5. Distribution limitations

6. Compiler needs / stand-alone

7. Purpose

8. Description

9. Required inputs

10. Program outputs

11. Diagnostics

12. Other features

13. History of method peer review

14. Steps taken by programmer for validation

15. Tests conducted by others

16. Notes by ICCAT

17. Sources cited

APPENDIX 1. Algorithm

APPENDIX 2. User´s guide