Stock simulation and testing of methods

Chato, Iago, Coilin

During the Santa Barbara meeting in January 2013 there was a revision of the factorial design generating the simulated stocks and the outputted catches. The old design was returning 400+ stocks which were considered to many for synthesizing the results. Therefore the factorial design has been trimmed down by reducing the levels of some factors and by moving some factors to a sensitivity testing. A second level of testing will be implemented via stochastic simulations where increasing levels of observation and process errors are added to the stock simulation as detailed later.

The agreed factorial design is now generating 72 stocks and related catches via the following factor interactions between ID, ED, LH and TS:

Factorial design

Factor	**level 1**	**level 2**	**level 3**	**level 4**
Initial depletion (**ID**)	0	30	60	
Effort dynamics (**ED**)	Constant flat	power increase (avoiding crash)	dome-shape (manageme nt kicks in)	Jim's effort dyn. model (choosing a realistic 'x', setting 'a' at 40% - but see sensitivity analysis)
Time-series length (**TS**)	20	60		
Life-history (**LH**)	sardine	haddock	tuna	

Additional sensitivity analyses (on a 'base' combination of factors above, i.e. assuming no interactions with factors above)

Factor	**level 1**	**level 2**	**level 3**	**level 4**
Underreporting (**UR**)	0	50%		
Selectivity (**SEL**)	Dome	flat	change in time	
Bmey/k - 'a' in Jim's effort dyn. model (intended to capture "bluefin" or "krill" like scenarios) (**ED**)	20	40	80	

Stochastic design: on a 'base' combination of factors above, i.e. assuming no interactions with factors above, we include recruitment errors plus

Factor	**level 1**	**level 2**
Catch measurement error (CV%)	0%	15%
Autocorrelation to recruitment AR(1) parameter	0	0.6

At the implementation level JRC is working on the new set of simulated stocks and will make them available soon.

Simulation Runs and Sensitivity

In Santa Barbara it was agreed to first run the full factorial dataset through the 4 assessment methods, this will give estimates of B/B_{msy}, diagnostics, etc. . We then propose to run the sensitivity analysis in the following way: select one scenario, out of the full factorial, that has performed well (high convergence rate), re-simulate the stock by adding new parameters such as UR, SEL and ED and assess sensitivity of these interactions with original scenario.

For the stochastic design, we propose to follow as similar approach with the re-simulation of stocks by addition of Autocorrelation to recruitment* and measurement error* to catches. * to be defined how much is realistic.

SIMULATION TESTING

Running the full testing of the 4 methods on the simulated data will be duty of Chato, Iago and Coilin. The necessary steps for a methods developer for running the simulations are the following:

1. Each method will run on a time series of catch data with information available being generic indication of a pseudo species (herring, haddock, tuna). Although there has been quite some discussion on the life history (LH) parameters of each simulated stock, ideally we would want to implement a double blind between data generation and methods development, so the final LH values will not be disclosed before testing.
2. Methods need to be streamlined so that there is no read/write to .csv or other type of files outside the R workspace. This would be very problematic when running the methods on different cores on a server.
3. Each method needs to be able to run sequentially on a list of stock, to deal with non-convergence and moving to the next stock and to give a report on convergence or no convergence and a number of statistics described below. The methods should not crash or in case of crash to save the runs before the crash. In any case to allow recovery after a crash in the simulation runs the methods will be run in a wrapper "try" which allows the user's code to handle error-recovery.

4. Each method needs to output an R list composed by two elements:
    a. First a Data Frame with the following variables:
        i. "stock_id" this needs to match exactly the code of the simulated stock, for example "SP_ID10_ED0.2_SELFD_TS60_UR0"
        ii. "b_bmsy" the estimated ratio of B/B_{msy} by year
        iii. "b_bmsyUpper" 85% CI
        iv. "b_bmsyLower" 85% CI
        v. "b_bmsy_iq25" 25% interquartile
        vi. "b_bmsy_iq75" 75% interquartile
        vii. "year"
        viii. "seed" starting seed for random number generator (if used, else NULL)
        ix. "convergence", logical 3 levels (see below)
            1. Not converged
            2. Weak convergence
            3. Strong convergence
        x. "n_iterations" a record of the number of iterations used.

xi. “effective_sample_size” calculated as indicated by Carolina and Jim see details below.
xii. “run_time” using system.time() for each model run
xiii. “method_id” (you choose!)

b. List or data frame with all model diagnostics/chains, priors (for r and K) and r/k pairs from catch-MSY etc in the form specified by the method. Data from this list might not be used but it’s good to keep for reference.

Notes:

1. CatchMSY needs to output b/bmsy
2. Convergence: it is defined as $C=N_{\text{effective samples}}$ OR $N_{\text{retained samples}}$
   1. If C<30 => Not converged
   2. If 30<C<200 => Weak convergence
   3. If C>200 => Strong convergence

Effective_sample_size, how to make the number of iterations or Bernoulli draws comparable.
Carolina and Jim proposed the following (discussion on this included for reference below):
For Catch_MSY use the number of retained samples as the effective sample size.
For JAGS, use the estimated effective sample size output from JAGS/CODA

Jim and Carolina’s discussion on this:
-effective sample size = 1/(sum of (importance weights^2))

-The Bernoulli draw is easier, and will just be the sum of the number of accepted draws (each is independently sampled from the priors). The number of draws in this case are from the second run of the Bernoulli once the bounds have already been defined:

effective sample size = sum(accepted draws)

-The MCMC effective sample size is calculated automatically in package CODA or R2WinBUGS function "monitor()".

3. Convergence Criterion, how long is each method left to run maximally? So if a method hasn’t converged, practically we need to impose a limitation. Need to discuss this with developers.