Modifications to SPM

The items below are suggested modifications to the SPM code to allow further flexibility and uses. It is split between items which are now in the SPM manual and those that still need to be defined.

# Bug fixes

*All the following cases are based on tests on the 1x1 example that is in the directory* examples/1x1/

## Profiling

spm –p command creates an odd error message at run time that looks to be linked to boost. Hopefully its not just my compiler…

Check failed in file c:\progra~1\mingw64\bin\../lib/gcc/x86\_64-w64-mingw32/4.6.1

/../../../../include/boost/numeric/ublas/triangular.hpp at line 2485:

e1 () (n, n) != value\_type ()

## GammaDiff tolerance error

spm –e with GammDiff minimiser incorrectly reports the gradient tolerance (@minimiser.tolerance) as 1.13078e-317 irrespective of its true value, and also reports the current gradient as 1.13078e-317 irrespective of its true value.

These are reported correctly for the DESolver minimiser

## MCMCs

MCMC code needs implementing. But currently calling spm –m errors out and crashes. Might be related to the thread code?

With spm –m we get,

Convergence was successful

terminate called after throwing an instance of 'std::string'

This application has requested the Runtime to terminate it in an unusual way.

Please contact the application's support team for more information.

We might want to return a ‘not yet implemented’ message in the meantime.

## Case sensitivity

Currently SPM lowercases all config file commands, arguments, and labels. This behaviour should be changed so that commands and subcommands are case insensitive, arguments and labels are case sensitive. Case should be preserved in the output and in reports

## Projections

Forward projections are not implemented in SPM. We need to intercept the call spm –f and return a “not yet implemented” message.

## Threads

Cannot recall if threads have been implemented as per spm –t n. Can we check, fix as required. Let me know and also what text to add to the manual, as we give the command line argument but provide no help to the user as to what this means or why use it (and I cannot recall what its purpose was!).

## Validates not always working

Wrongly spelled or invalid subcommands are not picked up in all commands. The places where I’ve found an error is the estimation, minimiser, and profile commands (and after a bit of looking, I think these might be the only ones)

@estimation

@minimiser

@mcmc

# Improving syntax parsing

## Syntax expansion: numeric ranges

Some arguments require vectors of years to be supplied. For these, it may be helpful to expand our allowed syntax to allow year ranges to be input. This may be useful for other commands as well (and I have yet to look where this will happen!)

We want SPM config loader to expand any value followed by a hyphen and then another value into a sequential vector

e.g., 1994-2000 or 1994 – 2000 gets expanded to 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004

As an example, the following syntax

@process MyRecruitment

type BH

standardise\_years 1994-2000 2002-2004

YCS\_years 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009

Is exactly equivalent to

@process MyRecruitment

type BH

standardise\_years 1994 1995 1996 1997 1998 1999 2000 2002 2003 2004

YCS\_years 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009

# AddED to the SPM manual and needing implementation

## Validation of likelihoods

Just as layers have a validation, some likelihoods require a validation process:

* Lognormal likelihood can only take positive layer values
* Multinomial have one error value per year only
* Lognormal can have one error value per class per year

Note that layers were not allowed to have a zero value. That has been changed to be allowed, but would provide issues if used for lognormal likelihoods.

*This has been implemented: Needs some additional checks and a bit of code modification. See validation functions in the observation classes for TODOs.*

## BIOMASS

The relevant formulas are described in the SPM manual, sections 4.10 and 4.11. This requires a size at age function, and a size-weight function. Both these functions will be combined in a single age\_size\_weight function, and SPM internal operator functions will be get\_length and get\_biomass.

Each block has only one set of parameters, and is hence a discrete object. When applied in a model, the user will define, for each category, what age-size-weight relationship to use.

Biomass and biomass density layers need to be implemented using these functions (section 4.4). Biomass derived quantiles also need to be applied; these are a single value that is the sum of a biomass layer.

@age\_size\_weight TOA

type von\_bertalanffy

age\_size\_distribution normal\_by\_length

k 0.093

t0 -0.256

Linf 169.07

Cv 0.10

a 1.387e-008

b 2.965

In the report section, additions need to be made compared with the manual. The output will be for every age from age\_min to age\_max have age, mean weight, mean size, and user-defined quantiles of sizes. Also optionally, the user can define specific lengths at which the corresponding weight will be returned.

@report[label].type=age\_size\_weight

age\_size\_weight defines the label of the age\_size\_weight relationship print

sizes defines the values of the sizes on which to print the weights (optional)

quantiles defines the quantiles on which to report size at age (optional)

*As a consequence, biomass reports will need to be implemented. Not sure the format at this stage though, but similar to partition reports and layer\_derived\_views*

## Recruitment

Details of the Beverton Holt (BH\_recruitment) recruitment formula are in the SPM manual, section 4.7.1. The recruitment process needs to define also steepness and YCS for the BH type recruitment.

Constant recruitment is already implemented, and hence we need to define two new ones: BH\_recruitment and local\_BH\_recruitment.

The C++ classes for both exists, but they need work

Two options are defined:

* BH\_recruitment: recruitment in each cell is on the basis of the total recruitment over a large number of cells (for example, the SSB is a derived quantity over some specific area) and applied to some cells according to a layer (e.g. a recruitment layer), or
* local\_BH\_recruitment: recruitment in each cell is on the basis of the spawning stock biomass of that individual cell only (for example, a biomass layer); remember to note in the manual that this recruitment layer can be further submitted to dispersion through a movement function.

Formulas for calculations are given in the manual, section 4.7.1 (eqns 4.8, 4.9)

For BH\_recruitment

@process Recruitment

type BH\_recruitment

r0 5000000

steepness 0.75

categories immature # Define the categories into which recruitment occurs

SSB SSB\_TOA # a derived quantity that gives the SSB for the recruitment calculation

B0 B0\_TOA # a derived quantity that gives B0 for the recruitment calculation

Layer MyRecruitmentlayer # Name of the layer used to determine where recruitment occurs

Ages 1 # Define the ages within each category that receive recruitment (see constant recruitment)

proportions 1.0

sigma\_r 0.6 [not actually used yet]

rho 0.6 [not actually used yet]

standardise\_years 1994-2004

YCS\_years 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009

YCS 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

~~first\_free 1994~~

~~last\_free 2004~~

SSB\_yearclass\_offset 1

So a new type of recruitment needs to be defined, as BH\_recruitment. R0, steepness can be estimated or fixed, and all the “free” YCS are estimated. If standardise\_years is defined, then the YCS used in the formula is YCS/mean(sum(YCS[in range standardise\_years])), otherwise they are used as defined.

The offset should default to (min\_age) in the model, but can be defined as some integer ≥0. Also for initialisation steps, where there is no information on SSB (year 1 to year min\_age), use R0.

We will need an autocorrelation parameter, rho, but ignore its use at this stage.

For local\_BH\_recruitment

@process Recruitment

type local\_BH\_recruitment

r0 5000000 # multiples the cell value in r0\_layer by this to get actual recruitment R0 for each cell

steepness 0.75

categories immature # Define the categories into which recruitment occurs

SSB\_layer MySSBLayer # a numeric layer that gives the SSB in each cell in each year for the recruitment calculation

B0\_layer B0\_layer\_TOA # a derived quantity that gives the SSB for the recruitment calculation

r0\_layer MyRecruitmentlayer # Name of the layer used to determine the relative cell by cell r0

proportions 1.0

sigma\_r 0.6 [not required yet]

rho 0.6 [not actually used yet]

standardise\_years 1994-2004

YCS\_years 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009

YCS 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

~~first\_free 1994~~

~~last\_free 2004~~

SSB\_yearclass\_offset 1

So a new type of recruitment needs to be defined, as local\_BH\_recruitment. R0, steepness can be estimated or fixed, and all the “free” YCS are estimated. If standardise\_years is defined, then the YCS used in the formula is YCS/mean(sum(YCS[in range standardise\_years])), otherwise they are used as defined.

Currently Ages is the recruitment age and should be taken off and replaced with @model.min\_age which is obtained from the model directly. The offset should default to @model.min\_age in the model, but can be defined as some integer ≥ @model.min\_age. Also for initialisation steps, where there is no information on SSB (year 1 to year min\_age), use R0.

Note if r0\_layer has a zero or negative value, then recruitment in that cell is zero.

## Preference functions for categorical layers

Currently only numeric layers can be used as covariate layers. The proposed extension would allow categorical layers as covariate layers. The associated preference function would be all values, where all the parameters are estimable. There must be a check whereby the number of categories is the same as the number of values provided, and the categories names match the categories in the layer. There is no provision to specify alpha for this preference function as it would equate to over-parameterisation (it just multiplies the values by an arbitrary factor).

There is provision to force the values to be monotonic, increasing or decreasing. One way of coding the monotonic change is to estimate the first parameter and the difference between each other parameter in order, with those forced to be positive (or negative). In case of monotonic relationship, the categories have to be provided in increasing order.

Type = categorical or monotonic\_categorical (decreasing would be reverting the categories)

f(x) = Vx

@preference\_function ImmatureArea

type categorical

category\_labels a b c

category\_values 10 2 50

If categorical, the category\_values are the actual values for these categories. If monotonic\_categorical, the category\_values are the actual value for the first one, and the difference between the value and the value of the previous category for the other two.

The category values need to be positive whether as categorical or monotonic categorical. This needs to be tested in the code.

NOW MOSTLY CODED: REQUIRED CODE CHECKING, UNIT TESTS.

## meta layers (needs a bit more thought)

These are defined in the SPM manual (section 4.4 item 8). Meta layers are layers indexed by year and applied by year. The meta-layer class is already defined (see section 8.6.8), which is in effect a three dimensional layer. Need to add the type of layer (numeric or categorical).

From the description it seems like the meta-layer is defined as a list of the names of the individual layers, see example below. It might be useful to have another option to define the meta-layer as defining layer, list the data as a layer, with data lines for each year.

Also required is how to apply these to calculations. Everywhere a layer is used, if it is a meta-layer then the year-specific layer should be retrieved prior to being applied.

Interpolation of layers is not allowed: all years defined must be contiguous otherwise throw an error. Extrapolation method is requested in the arguments, whereby the user defines the initialisation and prediction layers.

**Note**: Fishing layers (for example) can be defined as meta-layers or as individual layers. We need to make sure the code can cope with either, whereby the layer is applied, or if a meta layer, the appropriate year is applied. This will impact the @process function.

Below an example as it is currently described

@layer Fishing

initialisation No\_Fishing

prediction Future\_Fishing

years 1998 1999

layers Fishing\_1998 Fishing\_1999

Could also be defined as follows

@layer Fishing

initialisation No\_Fishing

prediction Future\_Fishing

Years 1998 2005

data 0 0 234 0 111 0

data 0 0 0 500 10 0

… #as many rows as rows in the model

## Derived layers and derived meta layers

There is a need for more generic derived layers, with specific mathematics formula to obtain these layers, and parameters which can be estimated if needs be. Current derived layers include abundance layers or biomass layers (not yet implemented – but will be).

Two types of derived layers need to be defined: as derived layers, or as derived meta-layers (see section 4.4 of the manual where this will be defined).

* Derived layers are calculated at a specific user-defined time step and years. If no years are defined, the data overwrites the previous calculation. For example B0 is derived once only, at the last initialisation phase (defined under year); but a mortality dependent layer is calculated every year (including in the initialisation phases) and only the latest calculation is kept in memory.
* Derived meta-layers are calculated every year of the model, and the results for each year saved in a single meta-layer. Derived meta layers will be calculated for the initialisation phases as well, and the information stored in the meta-layer.

Two new layer types need to be created, a used-defined calculated layer termed a derived\_layer or derived\_meta\_layer. It will include the layers to use, calculation, etc. Parameters for the formula can come from other layers, other categories, and / or specifically defined parameters. These parameters can be then estimated in the estimation section as any other parameter. Layer calculations can also be nested in each other.

Note that we currently define that derived layers are calculated or determined at the time a calculation is required. We propose to determine the time step when the layer calculation occurs, and then preserve this value until updated.

For example B0 is defined from the SSB biomass layer saved as a derived layer. SSB is firstly defined. Note that the timestep is now hard coded rather than the model calculating the biomass when needed (is it a good idea or not?).

@layer SSB

type biomass

timestep one

categories mature mature\_tag

selectivities 1 1

age\_size\_weight TOA\_asw

The derived layer B0 is simply SSB at initialisation phase, here it is multiplied by a layer which defines your stock limit for example (called SSB\_area). If the timestep is hard-coded in both, then there needs to be a check that they do agree.

@layer B0\_TOA

type derived\_layer

years initialisation-phase3

timestep one

layers SSB BBS\_area

formula SSB \* SSB\_area

Density dependent mortality base on diet electivity can also be defined using a derived layer. In the example below the predation of toothfish on grenadier as opposed to eelcod depends on the preference for the two species tempered by their availability in each cell. The layer defined below can then be used as a layer for mortality of grenadier due to predation by toothfish.

@layer MortalityDep

type derived\_layer

timestep one

layers B\_WGR B\_ELC

parameters ElecTW ElecTE # user defined labels that become estimable parameters

values 0.75 0.25

formula ElectTW\*B\_WGR/(B\_WGR+B\_ELC) # # code resolves string, and hence applies calculation.

The actual electivities of toothfish for grenadier can be estimated as follows.

@estimate layer[MortalityDep].ElecTW

lower\_bound 0.01

upper\_bound 0.99

prior Uniform

Layers, parameters and values are optional, but need to be defined if used in the calculation. If parameters is used, values is also needed, and of the same size as parameters. Layers can be any layer known to the model. Years are optional in which case the layer gets calculated every year including initialisation.

The following math functions would need to be implemented, and include parentheses

+ - \* / exp log(e) sqrt ^2 pow(a,b) cos sin Z

All would be functions that operate on every cell in a layer independently. Z is the zeroFun implemented elsewhere in our code.

See separate document for some math parsers.

## Mortality as an annual rate

Mortality as a constant rate is currently implemented (section 4.7.3), either as a constant, or as a function of a layer. Mortality as an annual rate should be implemented using the same function of a layer, whereby meta-layers can be used, or calculated layers (expected to change every year). If annual, mortality then needs to be calculated every year.

Mortality as a biomass event also needs to be implemented once the biomass part of the model has been implemented. Once again it is all described in SPM.

## Updated report arguments to cover multiple years

For

Report\_type = partition

Report\_type = layer

Report\_type = layer\_derived\_view

modify argument **year** to **years** so a single report can be used to return a range of years or all years, rather than each report just doing one year at a time

## Other migrations

The two other migrations not previously implemented are now described in the manual and need to be implemented.

# To be further developed and defined in the SPM manual

*This section outlines thoughts about new and changed functionality. Its just a holding place for these ideas and should not be implemented.*

## Problems with meta-layers

We need to think carefully when we should allow meta-layers and when we shouldn’t. For example meta-layers under biomass events might be misleading if only applied for specific years when it should be all years.

Maybe have meta-layers used instead of layers as default, but with specific safeguards or cases where you can’t.

## Entering layer data

At the moment we require a “data” subcommand for every line of SPM to input layer data. Is it worth adopting the form used in randomstation and cala to have a ‘begin\_data’ and ‘end’data’ style command? Then the subcommand is not required on each new line anymore…?. Note one difference between the programs is that we are inputting a matrix of values by row… and not a table of data with column headings.

## Layers

At the moment if the data supplied in a config file has too many data entries, is not the correct size (number of rows and columns), SPM fails, the DOS window closes and a Windows error pops up. When layers get loaded, there is a need to check that they are the correct shape (i.e. same size as the base layer) and give an appropriate error message and exit from SPM.

No fixed, and tested. It works.

## Tags as indicators of movement only

The aim of this new implementation is to use tags as informative of movement only, based on individual tag recaptures only. A new observation data class is created which relates to individual event matching. Data is input with a line per recapture event, with the relevant information to the model (in the order specified).

@observation Tag\_match

type event\_match

categories immature\_tag mature\_tag spawning\_tag

# year\_release year\_recapture cell\_release cell\_recapture age\_recapture selectivity\_recapture selectivity\_recapture selectivity\_recapture

data 2004 2009 r3-c4 r3-c4 12 FishingSel FishingSel FishingSel

…

Each tag recapture will be associated with temporary layers, created in the year of release. The layers will be calculated as such

* At year\_release,
  + Estimate the proportion of that fish in each category
  + Create one layer per category with in cell\_release the value of the proportion of that fish in that category
  + Age of that fish is age\_recapture + year\_recapture - year\_release.
* Between year\_release +1 and year\_recapture:
  + For each category, apply the relevant processes in the model (e.g. maturation, movement, mortality)
  + Update the layers to the new values of distribution, moving fish between layers (categories) and cells
* At year of recapture, once the layer has been updated
  + Apply fishing selectivity to each layer
  + For each layer, calculate the probability of being in the recapture\_cell as the value in that cell divided by the sum of the value in all the cells. Add each value to the total likelihood
  + Delete the layers