Gadget User Guide

Updated by Andrés Gómez and David Rodríguez Penas

April 24, 2018

Contents

Chapter 1

Introduction to Gadget

Gadget is the **G**lobally applicable **A**rea **D**isaggregated **G**eneral **E**cosystem **T**oolbox. Gadget is a flexible and powerful software tool that has been developed to model marine ecosystems, including both the impact of the interactions between species and the impact of fisheries harvesting the species. Gadget simulates these processes in a biologically realistic manner, and uses a framework to test the development of the modelled ecosystem in a statistically rigorous manner. Gadget has successfully been used to investigate the population dynamics of stock complexes in Icelandic waters, the Barents Sea, the North Sea and the Irish and Celtic Seas.

1.1 What is Gadget?

Gadget can run complicated statistical models which take many features of the ecosystem into account. Gadget works by running an internal forward projection model based on many parameters describing the ecosystem, and then comparing the output from this model to observed measurements to get a likelihood score. The model ecosystem parameters can then be adjusted, and the model re-run, until an optimum is found, which corresponds to the model with the lowest likelihood score. This iterative, computationally intensive process is handled within Gadget, using a robust minimisation algorithm. The Gadget software framework consists of three parts:

- a parametric model to **simulate** the ecosystem
- statistical functions to **compare** the model output to data
- search algorithms to **optimise** the model parameters

Gadget allows the user to include a number of features of the ecosystem into the model: One or more species, each of which may be split into multiple components; multiple areas with migration between areas; predation between and within species; growth; maturation; reproduction and recruitment; multiple commercial and survey fleets taking catches from the populations.

Like any large piece of code, Gadget is based on previous ideas by several authors. Conceptually the modelling framework extends earlier multi species programming work such as MSVPA and MULTSPEC into a more generic statistical framework. Alternatively, Gadget is a conceptual extension of the Stock Synthesis statistical assessment single-species framework into a multi species setting.

The initial BORMICON code was developed as part of a multi species programme implemented at the Marine Research Institute in Reykjavik, Iceland, starting in 1992 with Ólafur K Pálsson as project coordinator and Gunnar Stefánsson coordinating the modelling work. Subsequently the code became the basis for Fleksibest at the Institute of Marine Research in Bergen, Norway.

Further development work in 1999-2003 was partly funded by EU grant QLK5-CT1999-01609 (”Development of Structurally Detailed Statistically Testable Models of Marine Populations”). Development work for 2004-2006 was partly funded by EU grant 502482 (”Critical Interactions Between Species and their Implications for a Precautionary Fisheries Management in a Variable Environment - a Modelling Approach”). Developement work for 2010-2012 was partly funded by EU project ”Defineit” under the MariFish ERA-NET (grant ERAC-CT-2006-025989)

The Gadget code is derived from the original BORMICON code of Halldór Narfi Stefánsson, Höskuldur Björnsson and Hersir Sigurgeirsson. Subsequent contributions include program additions by Morten Nygaard Åsnes and Kristin Frøysa to include the Fleksibest model. The ability to run in parallel across a network of computers using PVM was implemented primarily by Auðbjörg Jakobsdóttir with input from several other people, including Jón Guðmundsson who worked on a variant based on Condor. Several mathematical additions and changes to allow for parallel minimisation on a network were implemented by Þórdís Linda Þórarinsdóttir and Kristjana Ýr Jónsdóttir.

More recent changes include the work to use information from tagging experiments, which was implemented primarily by Sigurður Hannesson. The work on multivariate distributions was implemented by Bjarki Þór Elvarsson. Recent additions including major code cleanups, modifications to the input formats and debugging operations have been led by James Begley, who is the current maintainer of the code and coordinator of all programming efforts.

1.2 Getting Gadget

Gadget is distributed via GitHub and can be obtained at:

`http://github.com/hafro/gadget`

The most recent version of Gadget will be available as a zip file from this from this site (select "Clone or download"). As a guide, the current version of Gadget is 2.2.00, so the file to be downloaded is gadget2.2.00.tar.gz.

Gadget is a program that runs on a Unix computing platform, and is regularly tested on machines running versions of Linux, Mac OSX and Windows. It should also run on other versions of Unix, but this may require modifications to the makefile in order to do so. Gadget is distributed as a set of source code files, which need to be compiled into an executable program before it can be run. To allow for various different versions, the makefile has a list of supported computing platforms, so the makefile should be checked (and if necessary changed) to ensure that the required options are available before the source code is compiled. The default options in the makefile will compile Gadget for a Linux machine, using the GNU C++ compiler. A summary of the commands required to compile Gadget, using the default options for a Linux machine from the downloaded file are:

```
unzip gadget-master.zip
cd gadget-master
make
```

The most recent copy of this document will also be placed on the github site, along with copies of any example datasets that are available. These datasets will be checked so that they run without any problems, and will also contain some comments to help the user understand the format of the data files. It is recommended that this user guide is read in conjunction with an example dataset.

A web site has been set up that contains details of the recent changes to Gadget, documentation updates and other news. This is also a good place to look for more information about Gadget. The address of the Gadget web site is `http://www.github.com/hafro/gadget`.

Throughout this document any text in <angled brackets> should be adjusted by the user to make it relevant to the current situation, so for example the text above should be altered to the following to compile Gadget version 2.2.00 on a Linux machine:

```
gunzip gadget2.2.00.tar.gz
tar -xvf gadget2.2.00.tar
cd source
make
```

1.3 Running Gadget

Gadget is a command line program that runs on a Linux computing platform. To start Gadget, simply type the following at a standard Linux prompt:

```
gadget <options>
```

where <options> are a combination of the starting switches described in the next section. Note that this assumes that the Gadget executable has been placed in a directory that is included in the $PATH, which might not be the case for some Linux distributions. If this is the case, Gadget should be started by including the path to the Gadget executable, for example by typing `./gadget` if the Gadget executable is in the current directory.

Gadget is not an interactive program, so once it has started there is no need for any further input from the user. However, if Gadget was started incorrectly (for example, using the wrong input files) then Gadget can be stopped by pressing <CTRL><C>, which will interrupt the calculations, displaying a menu from which it is possible to either store the current calculations to a text file, or to exit (by pressing <Q>).

When Gadget starts, it will look for a file called "main", which contains a list of all the other data files required. Gadget will search for this file in the following 2 locations:

1. the directory specified by the Linux environment variable called "GADGET_WORKING_DIR", optionally taking data from the directory specified by the Linux environment variable called "GADGET_DATA_DIR"
2. the current directory (ie. from where Gadget has been started) if these environment variables have not been set (note that this is the default location)

When Gadget is reading in the input data files, these files will be checked to ensure that they are in the correct format, and if there is an error in the format Gadget will print an error message and stop. Note that when Gadget is checking the format of a data file containing a number of columns (eg. the "area" file) it will only check the first data line in the file (so, for the case of the "area" file it will check that there are 4 entries on the first row and then assume that all the other rows also have 4 entries). Hopefully, there should be enough information in these error messages to lead the user to the input file that is causing the error.

If all else fails, and the error messages do not lead the user to the source of the error, then there is an email address set up for any difficulties or questions that may arise, or reporting any bugs that get found. In addition to sorting out any problems that the user may be getting, we can also try to help with interpreting the output. This email address is `gadgethelp@hafro.is`

Sending an email to this address should get a response within a couple of working days. This email address can also be used to give us any feedback on the usefulness (or otherwise) of Gadget, so let us know what you think!

1.4 Starting Switches

The following is a list of the command line switches that Gadget will recognise, together with a brief description of what the switch means to the running of the Gadget model.

```
gadget -s
```

Starting Gadget with the -s switch will start a simulation run, where a single run of the model will take place. This option is useful since it will give the model output (see Print Files, chapter 9), and it is used when running large Gadget models, using the paramin parallel processing optimiser to find the optimum.

```
gadget -l
```

Starting Gadget with the -l switch will start a optimising run, where the overall likelihood score will be reduced to an optimum, depending on the optimising information given (see the -opt switch).

```
gadget -n
```

Starting Gadget with the -n switch will start a network run, used in conjunction with the paramin optimiser to find an optimal solution for large models.

```
gadget -v
gadget --version
```

Starting Gadget with the -v (or --version) switch will display the version of Gadget that is being run.

```
gadget -h
gadget --help
```

Starting Gadget with the -h (or --help) switch will display a help screen, giving information about the various starting switches that can be used.

```
gadget -i <filename>
```

Starting Gadget with the -i switch will give Gadget an inputfile file from which the initial values and bounds of any variables can be read (see Parameter Files, chapter 10, for more information on the format of this file).

```
gadget -opt <filename>
```

Starting Gadget with the -opt switch will give Gadget an optimisation input file, from which the information about the optimisation routines will be read. This will specify the type of optimisation to perform, and also parameters for that optimisation routine (see Optimisation Files, chapter 11, for more information on the format of this file).

```
gadget -main <filename>
```

Starting Gadget with the -main switch will specify a filename for the main Gadget model input file, which contains links to all the other data files that are to be used by Gadget (see Main File, section 3.1, for more information on the format of this file). If Gadget is started without the -main switch, Gadget will use the default filename ”main” as the name for the main file.

```
gadget -p <filename>
```

Starting Gadget with the -p switch specifies the file that Gadget will use to print the final values and bounds of the variables, in the same format as for the inputfile. This file can then be used as the starting point for a subsequent Gadget run, if required. This file will always be generated, and if Gadget is started without the -p switch the default filename is ”params.out” which will be created in the current directory (see Output Files, chapter 12, for more information on the format of this file).

```
gadget -o <filename>
```

Starting Gadget with the -o switch specifies the file that Gadget will use to print the score from the likelihood calculations. This file will give details on the parameters that have been used, the likelihood components that have been used, and the values for these parameters and likelihood components. This can be a large file if Gadget is performing an optimising run (see Output Files, chapter 12, for more information on the format of this file).

```
gadget -print <number>
```

Starting Gadget with the -print switch will specify the frequency with which information is written to the likelihood output file (specified with the -o switch). The default value for this is 1, meaning that the likelihood information is written for every iteration.

```
gadget -precision <number>
```

Starting Gadget with the -precision switch will specify the number of digits to be used when printing the output from the likelihood calculations to files specified with the -o switch.

```
gadget -parallel [spe|rep]
```

Starting Gadget with the -parallel switch will execute in parallel those algorithms that support multi-threading (OPENMP) execution (currently, Simulated Annealing, Hooke & Jeeves, PMAPSO and PMADE algorithms). By defualt, the speculative version is used (option *spe*). Reproducible execution (option *rep*) is only supported by SA and Hooke&Jeeves algorithms. Both modes of execution will use all the available cores in your computer except if it has been limited using the environment variable OMP_NUM_THREADS=<number of cores to use>. For example, on Linux, executing the next command before running Gadget:

```
export OPM_NUM_THREADS=4
```

will run these algorithms using 4 threads. If you set this variable, use less cores than the available number on your computer. If you set a higher number than the number of cores in your computer, the performance of these algorithms can degrade considerablely. It is better if you adjust it to a even number (in some algorithms, it is mandatory).

```
gadget -log <filename>
```

Starting Gadget with the -log switch will specify a file to which Gadget will write logging messages to keep a record of internal Gadget actions. This can be a large file if Gadget is performing an optimising run (see Log Output, section 12.3, for more information on the format of this file).

```
gadget -loglevel <number>
```

Starting Gadget with the -loglevel switch will specify the level of detail that is to be used by Gadget when logging messages (both to the screen and to a log file if one has been specified with the -log switch). If Gadget is started without the -loglevel switch then the default is for warning messages and error messages to be shown during a simulation run (when Gadget is started with the -s switch) and only error messages are shown during an optimising run (when Gadget is started with the -l switch).

```
gadget -seed <number>
```

Starting Gadget with the -seed switch will specify the value that is used to initialise the random number generator used within Gadget (see Repeatability, section 11.7 for more information on the use of the random number generator within Gadget runs).

```
gadget -m <filename>
```

Starting Gadget with the -m switch will specify a file from which Gadget can read the other command line options. This file should contain a simple list of the switches and their values, as they would be entered from the command line.

```
gadget -printinitial <filename>
```

Starting Gadget with the -printinitial switch will specify a file to which Gadget will write all internal information for the model at the start of the run (ie. the stock populations, likelihood calculations and other information from before the first timestep). This file will be large for moderately complicated models, and it is of most use for debugging purposes.

```
gadget -printfinal <filename>
```

Starting Gadget with the -printfinal switch will specify a file to which Gadget will write all internal information for the model at the end of the run (ie. the stock populations, likelihood calculations and other information from after the last timestep). This file will be large for moderately complicated models, and it is of most use for debugging purposes.

```
gadget -maxratio <ratio>
```

Starting Gadget with the -maxratio switch will specify the maximum ratio of prey that is allowed to be "consumed" on any one timestep. This consumption includes both the consumption by other stocks and the catch by any fleets. The default value is 0.95, which ensures that no more than 95% of the available stock biomass is consumed on a single timestep.

Most of these switches can be combined to specify more information about the Gadget run that will be performed. For instance:

```
gadget -l -i inputfile.txt -o likelihood.txt -opt optinfo.txt -print 10
```

will start Gadget for a likelihood run, taking the initial values and information about the parameters from the file inputfile.txt, with the optimisation being done in accordance with the information in optinfo.txt, and printing likelihood information (every 10 iterations) to the file likelihood.txt.

Note that it is not possible to do both a simulation run and a likelihood run at the same time, and starting Gadget with both the -l and -s switches will result in a warning, after which Gadget will perform a simulation run, with the -l switch being ignored.

Chapter 2

Input Files

All the input files for Gadget are plain ASCII text files, so they can be viewed in any plain-text text editor. Any whitespace or blank lines in the data files are ignored, so the layout of the files can be adjusted into a easily viewable form to check the content of the files. The case of any text in an input file is ignored by Gadget, so for example, ”Stock” and ”stock” would be interpreted by Gadget as being the same.

Unless stated otherwise, all the input and output files use the following measurement units:

- length - all measurements are in centimetres
- weight - all measurements are in kilogrammes
- age - all measurements are in years

Gadget is a program that runs on a Linux computing platform, so the input data files must use a Linux style end-of-line character (<linefeed>) and not a Windows style end-of-line character (<carriage return><linefeed>). All the lines of the input files containing data should end with an end-of-line character.

2.1 Comments in Input Files

Any of the input files used by Gadget can contain comments that are not used by Gadget. The start of the comment is denoted by a semi-colon ”;”. Once Gadget has read as far as the semi-colon, the rest of the line will be ignored.

2.2 What Does The # Mean?

When Gadget is performing an optimising run, a number of the parameters can be adjusted to try to find a better fit between the modelled output and the data. The parameters that are to be adjusted are termed ”switches” and are marked in the input files by the ’#’ character. There are two valid formats for switches in the data file - either directly, or as part of a function.

2.2.1 Switches

The format for the switches is given by:

```
<numerical part>#<name>
```

where the <numerical part> consists of the (optional) initial value for the switch, and the <name> is a alpha-numerical text string used to identify the switch in the parameter file. Note that there is no whitespace either before or after the '#' character. Also note that the name of the switch cannot contain a hyphen, since this will be interpreted by Gadget as the mathematical subtraction symbol.

Any initial value specified in the data files will be overwritten by the initial value given in the parameter file (see chapter 10). If the initial value is not given, then a value of 1 with be assumed by Gadget. A switch can appear in more than one place in the data files, but needs to be defined with the same initial value each time.

Examples of valid switches in the data files can include:

1. defining a simple switch (called "age2") without an initial value

   ```
   #age2
   ```

2. defining a switch with an initial value

   ```
   10#age2
   ```

2.2.2 Functions

The format for the functions is given by:

```
(<function> <parameters>)
```

where the <function> defines the name of the function, and the <parameters> is a vectors of the parameters to be used - either numbers or the names of switches denoted by the '#' character. Note that the function is contained within (round brackets). Valid function names include the 4 mathematical operators ('*', '+', '-' and '/'), trigonometric functions ('sin' and 'cos', with the argument in radians), logarithmic functions ('log', 'log10' and 'exp'), square root ('sqrt') and a function to generate a random number between 0 and 1 ('rand').

Examples of valid functions for the switches in the data files can include:

1. defining a function based on a switch

   ```
   (* 10 #age2)
   ```

   note that in this case, Gadget will use 10 times the value of parameter "age2".

2. defining a function based on more than one switch

   ```
   (* #area1 #age2)
   ```

   note that in this case, Gadget will use the value of parameter "area1" multiplied by the value of parameter "age2".

3. nesting functions

   ```
   (log (+ #area1 #age2))
   ```

   note that in this case, Gadget will use the value of the natural logarithm of parameter "area1" plus parameter "age2".

Chapter 3

Model Files

Gadget requires a number of data files to define a Gadget model. The number of data files required depends on the complexity of the Gadget model, and there is no limit on the number, or name, of these data files. The main input file gives links to all the other data files required, and must be specified with the "-main <filename>" command line option, or be called "main".

3.1 Main File

The main Gadget input file is usually called "main", unless it is specified with the "-main <filename>" command line option. This file only contains links to other files which will make up the Gadget model. The format for this file is:

```
<typeoffile>        <filename>
```

where <typeoffile> is a keyword to tell Gadget what sort of information the file will contain, and the name of the file is given by <filename>, relative to the directory in which the main file resides. Where zero or more files of a certain type could be used, the main file is divided into sections that are separated by a keyword in [square brackets]. The format for the main file is shown below:

```
timefile            <name of the time file>
areafile            <name of the area file>
printfiles          <names of the print files>
[stock]
stockfiles          <names of the stock files>
[tagging]
tagfiles            <names of the tag files>
[otherfood]
otherfoodfiles      <names of the otherfood files>
[fleet]
fleetfiles          <names of the fleet files>
[likelihood]
likelihoodfiles     <names of the likelihood files>
```

The printfile element of the main file is optional, and can be commented out if no model output is required. It should be noted that the keyword "printfiles" must be present, so to comment out the printfile section, a semi colon should be placed before the name of the printfile, as shown in the line below:

```
printfiles ; <filename> commented out so no printing will take place
```

3.2 Time File

This specifies the start and end times for the model run, and the number of timesteps per year. Note that the model can run into the future, and that datasets covering only part of the overall run can be used. Gadget splits each year up into a number of time steps, but these time steps need not all be the same length.

The format for this file is a list of the first year and timestep, and the last year and timestep, and how each year is to be divided into timesteps. This is done by specifying first the number of timesteps in a year, and then the length of each timestep (in months), and Gadget will check that the number of timesteps in a year sums up to 12. This is shown below:

```
firstyear       <first year>
firststep       <first step>
lastyear        <last year>
laststep        <last step>
notimesteps     <how the year is split up>
```

Examples for how the year can be split up include:

1. 4 equal timesteps, splitting the year into quarters

   ```
   notimesteps     4 3 3 3 3
   ```

2. 12 equal timesteps, splitting the year into months

   ```
   notimesteps     12 1 1 1 1 1 1 1 1 1 1 1 1
   ```

3. 6 unequal timesteps, splitting the year into the following periods, <January - February>, <March>, <April - June>, <July - September>, <October> and <November - December>

   ```
   notimesteps     6 2 1 3 3 1 2
   ```

3.3 Area File

This file specifies which areas the model will be run on, and gives a time dependent temperature for each area. Note that although the temperature data must be provided it need not actually be used, depending on the growth and feeding options chosen for the stock file.

The format for this file is a list of the areas that are to be used (by specifying a numeric identifier for each area), followed by the size of each area (in square kilometres) and then a listing of the temperature for each timestep and area combination. An example of this format is given below:

```
areas   <vector of area identifiers>
size    <vector of sizes>
temperature
<year>  <step>  <area>  <temperature>
```

3.4 Other Input Data Files

There are four other types of input that are important since they are used in other data files to denote a grouping of data. These are Aggregation files, which are files used to gather data into convenient groups, TimeVariable files, which are files used to denote variables that can vary over time, StockVariable files, which are files used to denote vairables that can vary with population, and ActionAtTime, which is used to define the timesteps that an action takes place within a data file.

3.4.1 Aggregation Files

Aggregation files are important since they are used to group the data in convenient groups. They consist of a text label (used to identify the group in the data) followed by a list of the data that the label represents. This data will then be read in from an associated data file. There are aggregation files to group areas, ages, lengths or preys together. Aggregation files can contain comments and blank lines, to make the format easier to view in a text editor.

Area Aggregation

Area aggregation files contain one or more identifying labels and then a list of one or more areas that the label refers to. The format for this is:

```
<label>      <areas>
```

An example of this is:

```
north        1  2  6
south        3  4  5  7
```

This example shows that for the associated data file, the label "north" will be interpreted as applying to areas 1, 2 and 6 and the label "south" will be interpreted as applying to areas 3, 4, 5 and 7.

Age Aggregation

Age aggregation files contain one or more identifying labels and then a list of one or more ages that the label refers to. The format for this is:

```
<label>      <ages>
```

An example of this is:

```
young        1  2  3  4
old          5  6  7
```

This example shows that for the associated data file, the label "young" will be interpreted as applying to ages 1 - 4 and the label "old" will be interpreted as applying to ages 5 to 7.

Length Aggregation

Length aggregation files contain one or more identifying labels and then the minimum and maximum length that the label refers to. The format for this is:

```
<label>      <minimum>  <maximum>
```

When more than one length group label is defined, then the labels should be ordered so that the smallest length group is first in the file. The data is checked, so that the maximum length associated with label <i> is the same as the minimum length for label <i+1>. An example of this is:

```
small        5   25
medium       25  55
large        55  80
```

This example shows that for the associated data file, the label "small" will be interpreted as applying to lengths 5-25cm, the label "medium" will be interpreted as applying to the lengths 25-55cm and the label "large" will be interpreted as applying to lengths 55-80cm.

Prey Aggregation

Prey aggregation files contain one or more identifying labels, and then the names of the preys, the minimum and maximum lengths for the preys and the digestion coefficients for the consumption of the preys. The format for this is:

```
<label>
<prey names>
lengths                  <minimum>  <maximum>
digestioncoefficients    d0  d1  d2
```

The digestion coefficients define a multiplier used when calculating the consumption of the prey. This multiplier is length dependant, and is calculated according to the digestion equation given below:

$$D = d_0 + d_1 l^{d_2} \tag{3.1}$$

An example of a prey aggregation file is:

```
; for the first prey
smallcapelin
immature.capelin
lengths                  5  10
digestioncoefficients    1  0  0
;
; for the second prey
mediumcapelin
immature.capelin mature.capelin
lengths                  10 15
digestioncoefficients    1  0  0
;
; for the third prey
largecapelin
mature.capelin
lengths                  15 20
digestioncoefficients    1  0  0
```

This example shows that for the associated data file, the label ”smallcapelin” will be interpreted as applying to immature capelin (a stock called immature.capelin) of lengths 5-10cm, the label ”mediumcapelin” will be interpreted as applying to immature and mature capelin (stocks immature.capelin and mature.capelin) of lengths 10-15cm, and the label ”largecapelin” will be interpreted as applying to mature capelin (a stock called mature.capelin) of lengths 15-20cm.

3.4.2 TimeVariable Files

TimeVariable files are used in place of a single variable in the input files, to define variables in the Gadget model that can vary over time. They are used by specifying the name of the TimeVariable file in place of the value or parameter in the input file. TimeVariable files can contain comments and blank lines, to make the format easier to view in a text editor.

TimeVariable files consist of a one word description of the data, followed by an optional multipler value and then the keyword ”timedata”, and then a list of the timesteps when the value of TimeVariable changes, along with the new value for the TimeVariable. The values can either be numerical values or a parameter to be optimised. The first timestep in the TimeVariable file must correspond to the first timestep in the model. The basic format for this file is shown below:

```
<description>          ; one word description of the data
multipler              <optional multipler> ; default value 1
timedata
<year>  <step>  <value>
```

An example of a TimeVariable file is:

```
annualgrowth
timedata
; year  step  value
1995    1     #grow1995
1996    1     #grow1996
1997    1     #grow1997
1998    1     #grow1998
1999    1     #grow1999
2000    1     #grow2000
```

This example shows that the value of parameter ”grow1995” is to be used for growth in 1995, ”grow1996” is to be used in 1996 and so on through to ”grow2000” which is to be used from 2000 through to the end of the simulation. Note that this example assumes that the first timestep of the simulation is the first timestep in 1995.

3.4.3 StockVariable Files

StockVariable files are used in place of a single variable in the input files, in a similar manner to the TimeVariable files, to define variables in the Gadget model that can vary due to stock populations changes. They are used by specifying the name of the StockVariable file in place of the value or parameter in the input file. StockVariable files can contain comments and blank lines, to make the format easier to view in a text editor.

StockVariable files consist of a one word description of the data, followed by an optional multipler value and then the keyword ”stockdata”, and then a list of the stocks to use when caluclating the value of the StockVariable:

```
<description>          ; one word description of the data
multipler              <optional multipler> ; default value 1
stockdata
biomass                <0 or 1> ; default value 1
<vector of stock names>
```

The optional flag <biomass> is used to specify whether the biomass or the number of the stock population is used when calcualting the value of the StockVariable. If this is set to 0, then the StockVariable value will be based on the available number of the stock(s). If this line is not specified, then a biomass value of 1 is assumed and the StockVariable value will be based on the available population numbers for the stock(s).

An example of a StockVariable file is:

```
capelinconsumption
multipler     (* #param1 #param2)
stockdata
immature.capelin mature.capelin
```

This example shows that the value of the StockVariable is to be calculated as the value of ”param1” times the value of ”param2” times the total biomass of the stocks ”immature.capelin” plus ”mature.capelin”.

3.4.4 ActionAtTime

ActionAtTime is a simple list of timesteps within a data file that define when a specified action (for example printing) will take place. The format for this is:

```
<year>  <step>
```

where year and step are either a valid timestep or the keyword ”all”. These can be grouped together to specify a more complex time period.

Examples of valid ActionAtTime entries in the data files include:

1. the action taking place on the first timestep of 2002

   ```
   2002    1
   ```

2. the action taking place on all timesteps of 2000

   ```
   2000    all
   ```

3. the action taking place on the second timestep of each year

   ```
   all     2
   ```

4. the action taking place on all timesteps of each year

   ```
   all     all
   ```

5. the action taking place on the first and second timesteps of each year

   ```
   all     1
   all     2
   ```

Chapter 4

Stock Files

The stock files contain all the information that Gadget requires for each stock in the model. To define the stock files in the Gadget model, the following lines are required in the "main" Gadget file:

```
[stock]
stockfiles              <names of the stock files>
```

The "main" file needs to list the files that define the stocks to be used in the model. Each stock requires a separate stock file. The information for the stocks are very detailed, and so these stock files are quite large and can be complicated to look at. The basic format for this file is:

```
stockname               <name of the stock>
livesonareas            <areas that the stock lives on>
minage                  <minimum age for the stock>
maxage                  <maximum age for the stock>
minlength               <minimum length for the stock>
maxlength               <maximum length for the stock>
dl                      <step size for the length groups>
refweightfile           <see Reference Weight>
growthandeatlengths     <see Growth and Eat Lengths>
doesgrow                <see Growth>
naturalmortality        <see Natural Mortality>
iseaten                 <see Stock Prey>
doeseat                 <see Stock Predator>
initialconditions       <see Initial Conditions>
doesmigrate             <see Migration>
doesmature              <see Maturation>
doesmove                <see Movement>
doesrenew               <see Renewal>
doesspawn               <see Spawning>
doesstray               <see Straying>
```

The first seven lines of this file give the basic details of the stock, and are fairly self explanatory. It should be noted that the oldest age group and the longest length group are interpreted in Gadget as plus groups for the stock. The remaining lines give more detail about the stock, and are covered in the sub sections below.

4.1 Reference Weight

The reference weight section lists the weight of the stock for various length groups. This is the reference weight that can be used in the initial conditions to set up the stock, and can also be used

by the growth functions when calculating the increase in length of the stock due to the growth. In the stock file, the format for the reference weight is given by:

```
refweightfile          <name of the reference weight file>
```

The format of the reference weight file is simply a two column list of the length and the corresponding weight for the stock. This file is ordered so that the smallest length group is given first, up to the longest length group which is given last. The format for this file is shown below:

```
<length>  <weight>
```

When this data is read into Gadget it is aggregated so that the weight is calculated for each length group defined in the stock file.

4.2 Growth and Eat Lengths

The calculations for the growth and consumption parts of the Gadget model can be done on a coarser scale than that defined in the stock file. The growth and eat lengths section of the stock file gives the name of a length aggregation file that defines this length grouping. In the stock file, the format for the growth and eat lengths is:

```
growthandeatlengths    <name of the length aggregation file>
```

4.3 Growth

The growth section of the stock file determines if, and how, the stock will grow in the Gadget model. The format for the first part of the growth section is given below:

```
doesgrow               <0 or 1> ; 0 for no growth, 1 for growth
```

If there is no growth, then the following sections don't apply, and the next section of the input file is the natural mortality, given in section 4.5 below. If the stock does grow, there are various different functions that determine the mean growth of the stock, so there are a number of different formats that the growth data can take. Once the mean growth has been calculated, the growth then needs to be statistically distributed to give the overall growth of the stock. Thus, after the growth function data has been read in, there is then data for the growth implementation. The full format for the growth of the stock is therefore given by:

```
doesgrow               1
growthfunction         <growth function>
<format for the growth function data>
<format for the growth implementation>
```

The <growth function> defines what growth function is to be used to calculate the growth of the stock. Currently, there are 7 growth functions defined, so valid function names are:

multspec - use the MULTSPEC growth function
weightvb - use the WeightVB growth function
weightjones - use the WeightJones growth function
weightvbexpanded - use the WeightVBExpanded growth function
lengthvb - use the LengthVB growth function
lengthpower - use the LengthPower growth function
lengthvbsimple - use the LengthVBSimple growth function

4.3.1 MULTSPEC Growth Function

This growth function is a simplified "MULTSPEC" growth equation, with the increase in length for each length group of the stock given by equation 4.1, and the corresponding increase in weight of the stock given by equation 4.2 below:

$$\Delta L_i = \Delta t p_0 {L_i}^{p_1} \psi(p_2 T + p_3) \tag{4.1}$$

$$\Delta W_i = \Delta t p_4 {W_i}^{p_5} (\psi - p_6)(p_7 T + p_8) \tag{4.2}$$

where:
$< \Delta t >$ is the length of the timestep
$<$ T $>$ is the temperature
$< \psi >$ is the feeding level (see section 4.8)

There are 4 parameters in the length equation, and 5 in the weight equation, giving a total of 9 parameters to be declared to define this growth function. This is given in the main stock file by declaring a single vector with 9 components, consisting of the 4 length parameters followed by the 5 weight parameters. This is shown below:

```
growthparameters        <growth parameters vector>
```

4.3.2 WeightVB Growth Function

This growth function is a form of the Von Bertalanffy growth equation, extended to introduce the concept of starvation to the growth function. The increase in the weight for each length group the stock is given by equation 4.3, and the corresponding increase in the length of the stock is given by equation 4.6 below:

$$\Delta W_i = \Delta t q_0 e^{q_1 T} \left(\left(\frac{W_i}{q_2} \right)^{q_4} - \left(\frac{W_i}{q_3} \right)^{q_5} \right) \tag{4.3}$$

$$r = \frac{W - (p_0 + p_8 (p_1 + p_2 p_8)) W_{ref}}{W} \tag{4.4}$$

$$f(x) = \begin{cases} 0 & \text{if } p_3 + p_4 x \leq 0 \\ p_5 & \text{if } p_3 + p_4 x \geq p_5 \\ p_3 + p_4 x & \text{otherwise} \end{cases} \tag{4.5}$$

$$\Delta L_i = \frac{\Delta W_i}{p_6 p_7 l^{p_7 - 1}} f(r) \tag{4.6}$$

where:
$< \Delta t >$ is the length of the timestep
$<$ T $>$ is the temperature
$< W_{ref} >$ is the reference weight

Comparing the weight to the reference weight (by using equations 4.4 and 4.5) introduces the concept of starvation to the Gadget model. When the weight of the population is less than a function of the reference weight, there is no length increase (ensuring that the growth only has an effect on the weight).

There are 6 parameters for the equation for increase of the weight, and a further 9 parameters for the increase in length. These are declared in 2 vectors, as shown below:

```
wgrowthparameters       <weight parameters vector>
lgrowthparameters       <length parameters vector>
```

4.3.3 WeightJones Growth Function

This growth function is a form of the Jones growth equation, extended to introduce the concept of starvation to the growth function. The increase in the weight for each length group the stock is given by equation 4.7, and the corresponding increase in the length of the stock is given by equation 4.10 below:

$$\Delta W_i = \Delta t \left(\frac{C}{q_0 W_i^{q_1}} - q_2 W_i^{q_3} e^{(q_4 T + q_5)} \right) \tag{4.7}$$

$$r = \frac{W - (p_0 + \psi (p_1 + p_2 \psi)) W_{ref}}{W} \tag{4.8}$$

$$f(x) = \begin{cases} 0 & \text{if } p_3 + p_4 x \leq 0 \\ p_5 & \text{if } p_3 + p_4 x \geq p_5 \\ p_3 + p_4 x & \text{otherwise} \end{cases} \tag{4.9}$$

$$\Delta L_i = \frac{\Delta W_i}{p_6 p_7 l^{p_7 - 1}} f(r) \tag{4.10}$$

where:
$< \Delta t >$ is the length of the timestep
< C > is the consumption (see section 4.8)
< T > is the temperature
$< \psi >$ is the feeding level (see section 4.8)
$< W_{ref} >$ is the reference weight

There are 6 parameters for the equation for increase of the weight, and a further 8 parameters for the increase in length. These are declared in 2 vectors, as shown below:

```
wgrowthparameters      <weight parameters vector>
lgrowthparameters      <length parameters vector>
```

4.3.4 WeightVBExpanded Growth Function

This growth function is an expanded form of the Von Bertalanffy growth equation, with additional information to allow for differing growth depending on the year, timestep and area. The increase in the weight of the stock is given by equation 4.11 below, and the corresponding increase in the length of the stock is identical to that for WeightVB growth function (see section 4.3.2) given by equation 4.6:

$$\Delta W_i = \Delta t Y_y S_s A_a q_0 e^{q_1 T} \left(\left(\frac{W_i}{q_2} \right)^{q_4} - \left(\frac{W_i}{q_3} \right)^{q_5} \right) \tag{4.11}$$

where:
$< \Delta t >$ is the length of the timestep
< T > is the temperature
$< Y_y >$ is a multiplier for year y
$< S_s >$ is a multiplier for step s
$< A_a >$ is a multiplier for area a

There are 6 parameters for the equation for increase of the weight, and a further 9 parameters for the increase in length and these are declared in 2 vectors. Additionally there are vectors for the year, step and area multipliers, as is shown below:

```
wgrowthparameters      <weight parameters vector>
lgrowthparameters      <length parameters vector>
yeareffect             <year effect vector>
stepeffect             <step effect vector>
areaeffect             <area effect vector>
```

Note that the <year effect> vector requires one entry for each year, the <step effect> vector requires one entry for each step and the <area effect> vector requires one entry for each area.

4.3.5 LengthVB Growth Function

This growth function is a simplified form of the Von Bertalanffy growth equation. The increase in the length for each length group the stock is given by equation 4.12, with the corresponding increase in the weight of the stock read in from a data file:

$$\Delta L_i = (L_\infty - L_i)\left(1 - e^{-\kappa \Delta t}\right) \tag{4.12}$$

where:
$< \Delta t >$ is the length of the timestep

There are a total of 2 parameters for the equation for increase in length. These are declared in a single vector, as shown below:

```
growthparameters       <linf> <kappa>
weightgrowthfile       <weight data file>
```

It is important to note that if the value of L_∞ is less than the mean length of a length group, then the length increase calculated by equation 4.12 would be negative for that length group, which will result in Gadget printing a warning message. To avoid this, the user should ensure that the value of L_∞ is always greater than the mean length of the largest length group.

The weight data file consists of a list of the year, timestep, area and length group label, followed by the mean increase in weight for that timestep/area/length combination. The length group label used must match those specified in the growthandeatlengths length aggregation file. The format of this file is shown below:

```
<year>  <step>  <area>  <length group>  <number>
```

4.3.6 LengthPower Growth Function

This growth function uses a simple power-based growth equation to calculate the increase in length. The increase in the length for each length group the stock is given by equation 4.13, with the corresponding increase in the weight of the stock read in from a data file:

$$\Delta L_i = L_i^{q_0} e^{q_1 \Delta t} \tag{4.13}$$

where:
$< \Delta t >$ is the length of the timestep

There are a total of 2 parameters for the equation for increase in length. These are declared in a single vector, as shown below:

```
growthparameters       <growth parameters vector>
weightgrowthfile       <weight data file>
```

The weight data file consists of a list of the year, timestep, area and length group label, followed by the mean increase in weight for that timestep/area/length combination. The length group label used must match those specified in the growthandeatlengths length aggregation file. The format of this file is shown below:

```
<year>  <step>  <area>  <length group>  <number>
```

4.3.7 LengthVBSimple Growth Function

This growth function is a simplified form of the Von Bertalanffy growth equation. The increase in the length for each length group the stock is given by equation 4.14, and the corresponding increase in the weight of the stock is given by equation 4.15 below:

$$\Delta L_i = (L_\infty - L_i)\left(1 - e^{-\kappa \Delta t}\right) \tag{4.14}$$

$$\Delta W_i = \alpha \left((L_i + \Delta L_i)^\beta - {L_i}^\beta \right) \tag{4.15}$$

where:
$< \Delta t >$ is the length of the timestep

There are a total of 4 parameters for the equation for increase in length and weight. These are declared in a single vector, as shown below:

```
growthparameters       <linf> <kappa> <alpha> <beta>
```

It is important to note that if the value of L_∞ is less than the mean length of a length group, then the length increase calculated by equation 4.14 would be negative for that length group, which will result in Gadget printing a warning message. To avoid this, the user should ensure that the value of L_∞ is always greater than the mean length of the largest length group.

4.4 Growth Implementation

The growth functions described above calculate the mean growth for the stock within the model. This must then be translated into a statistical distribution of actual growths around that mean. Currently there is only one statistical distribution implemented with Gadget, the beta-binomial, which is described below.

Note that regardless of the results of the implementation function there is a minimum width to the possible distribution implemented in Gadget - where growth is allocated between two adjacent length categories. This is a result of the discretisation within Gadget. To avoid this the user should select a length category size small enough for some fish to grow by at least 3 or 4 categories in one time step.

4.4.1 Beta-Binomial

This method uses a statistical distribution to govern the implementation of fish growth. The statistical distribution chosen is the beta-binomial, an extension of the binomial distribution with the flexibility to produce non-symmetrical distributions, which is defined for integers, x = 0, . . . , n as:

$$\binom{n}{x} p^x (1-p)^{n-x} = \frac{\Gamma(n+1)}{\Gamma(x+1)\Gamma(n-x+1)} p^x (1-p)^{n-x} \tag{4.16}$$

For more flexibility, this can be re-arranged by calculating the parameter <p> from a second beta-binomial distribution, leading to equation 4.17 shown below, which is defined for $0 \leq p \leq 1$:

$$f(p) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1} \tag{4.17}$$

$$\alpha = \frac{\beta \Delta L}{n - \Delta L} \tag{4.18}$$

The distribution is governed by three parameters; the mean length growth computed by the growth function scaled by dividing by the width of the length groups ($< \Delta L >$), a fixed limit on the number of length groups ($< n >$), and a value for beta. This distribution was chosen because it provides a high degree of flexibility resulting from changing a single estimated parameter, beta. To define the distribution data using a beta-binomial distribution the following data is required in the main stock file:

```
beta                    <beta>
maxlengthgroupgrowth    <max length group growth>
```

The parameter <beta> is the parameter that Gadget can estimate in the optimisation routines, in order to tune the distribution to best fit the data. High values of beta produce a narrow distribution, whilst lower values produce a more dispersed distribution with a larger right-hand tail. Note that very low values of beta (which lead to $\alpha < 1$ in equation 4.18) can lead to unexpected results and should be avoided.

The <max length group growth> value is the maximum number of length categories a fish is permitted to grow in a single timestep within the model. This should be set to be several length groups larger than any fish can be reasonably expected to grow in a time step, in order to provide the beta-binomial distribution with sufficient flexibility to produce a distribution around the mean.

4.5 Natural Mortality

The natural mortality is a measure of how much of the stock will be removed from the model due to additional natural causes that are not modelled as part of other processes (for example predation). In Gadget, this is simply modelled as the proportion of the stock that is removed due to these additional causes (the residual mortality) from each age group, on each timestep, as shown in equation 4.19 below:

$$N_a = e^{-m_a \Delta t} \tag{4.19}$$

where:
$< \Delta t >$ is the length of the timestep

The natural mortality parameters $< m_a >$ are specified in the stock file as a vector, with one parameter per age group, as shown below:

```
naturalmortality        <natural mortality vector>
```

4.6 Stock Prey

The stock prey section of the stock file determines if, and how, the stock will be treated as a prey in the Gadget model. The format for the first part of the prey section of the main stock file is given below:

```
iseaten                 <0 or 1> ; 0 for not eaten, 1 for eaten
```

Note that the fleets are treated as a predator by Gadget, so for the stock to be caught by the fleets, the stock needs to be declared as a prey. If the stock is not eaten, then the following section doesn't apply, and the next section of the input file is the stock predator, given in section 4.7 below. If the stock is eaten, then for Gadget to treat it as a prey the length groups for that prey, and the energy content of the prey, must be defined. The length groups need not be to the same scale as for the stock as a whole. This is done by listing a length aggregation file that is to be used, as well as the energy content (in kilojoules per kilogram), as shown for the full example below:

```
iseaten                 1
preylengths             <length aggregation file>
energycontent           <energy value>
```

4.7 Stock Predator

The stock predator section of the stock file determines if, and how, the stock will be treated as a predator in the Gadget model. The format for the first part of the predator section is given below:

```
doeseat                 <0 or 1> ; 0 for not a predator, 1 for predator
```

If the stock is not a predator, then the following sections don't apply, and the next section of the input file is the initial conditions, given in section 4.10 below. If the stock is a predator, then it is necessary to specify information about the predation. This is done by defining a suitability function, and then some prey preference, consumption and feeding parameters. The full format for this is shown below:

```
doeseat                 1
suitability             <see Suitability>
preference              <prey preference values>
maxconsumption          <maximum consumption vector>
halffeedingvalue        <half feeding value>
```

The <suitability> defines the prey that the predator will consume, and is discussed in the Suitability section below (see section 4.9). The <prey preference> values, <maximum consumption> vector and <half feeding> value parameters define how the stock consumes any prey that is eaten, and are described in the following section.

4.8 Consumption

The consumption determines how much of a given prey is consumed by the predator. This will result in the population of the prey being reduced, and can also affect the growth of the predator, depending on the growth function selected. The consumption of a prey is dependant on the length of both the predator and the prey, and the amount of the prey available, as a proportion of the total amount of food available. The consumption of prey p is given by equation 4.20 below:

$$C_p(l,L) = \frac{N_L M_L \psi_L F_p(l,L)}{\sum_{preys} F_p(l,L)} \tag{4.20}$$

$$F_p(l,L) = \Big(S_p(l,L) E_p N_l W_l\Big)^{d_p} \tag{4.21}$$

$$M_L = m_0 \Delta t e^{(m_1 T - m_2 T^3)} L^{m_3} \tag{4.22}$$

$$\psi_L = \frac{\sum_{preys} F_p(l,L)}{H\Delta t + \sum_{preys} F_p(l,L)} \tag{4.23}$$

where:
$< \mathrm{L} >$ is the length of the predator
$< \mathrm{l} >$ is the length of the prey
$< \mathrm{H} >$ is the half feeding value

$<$ d $>$ is the preference of the predator for the prey
$<$ E $>$ is the energy content of the prey (see section 4.6)
$<$ N $>$ is the number of prey in the length cell
$<$ W $>$ is the mean weight of the prey in the length cell
$<$ S $>$ is the suitability function (see section 4.9)
$< \Delta t >$ is the length of the timestep
$<$ T $>$ is the temperature of the area that the feeding takes place on

In equation 4.20, the parameter $<$ F $>$ gives the amount of a given prey that is consumed by the predator, given by multiplying the biomass of the prey by the suitability (see section 4.9), as shown in equation 4.21. The summation over preys is over all preys that the predator consumes, including non-modelled prey, given as "otherfood" in section 6. The value of $<$ C $>$ can have an affect on the growth of the predator when the growth function has been set to "WeightJones".

In equation 4.21, the parameter $<$ d $>$ gives the preference of the predator for the prey, controlling the form of the functional response that is used to model the predation. Setting this parameter to 1 will ensure that a Type II functional response is used (note that this is always used when modelling the predation by the fleets, see section 7). Any other value will mean that a Type III functional response is used. The prey preference parameter is specified by listing the names of the prey and the associated preference value, as shown in the example below:

```
...
preference
<name of prey 1>       <preference for prey 1>
<name of prey 2>       <preference for prey 2>
maxconsumption         <maximum consumption vector>
...
```

In equation 4.22, the parameter $<$ M $>$ gives the maximum possible consumption for the predator on the current timestep. This is a function of temperature and the length of the predator, using the 4 parameters specified by the <maximum consumption> vector in the input file. Note that the maximum consumption should be specified in kilojoules per month.

In equation 4.23, the parameter $< \psi >$ gives the "feeding level" which denotes the fraction of the available food that the predator is consuming. This is governed by the total amount of prey available and the <half feeding> value which is specified in the input file. The <half feeding> value is the biomass of prey required to allow the predator to consume prey at half the maximum consumption level. The value of $< \psi >$ can have an affect on the growth of the predator (ie. when there isn't sufficient food available) when the growth function has been set to "MULTSPEC" or "WeightJones".

4.9 Suitability

The suitability determines how the predators act on the preys. This selectivity relationship between the predator and the prey is based on the lengths of the predator and prey, and are defined by declaring a suitability function and the parameters for that function. The suitability values can be thought of as the proportion of the prey length group that the predator length group can consume, and as such the suitability values should be between 0 and 1.

To define a suitability relationship based on a suitability function, the stock file (for the predator) needs to contain the following data:

```
suitability
<preyname>    function <functionname> <parameters> ; for each prey
```

The <functionname> defines which suitability function is to be used to calculate the suitability for the predator acting on the prey. Currently there are 7 suitability functions defined, and the valid suitability function names are:

constant - use the Constant suitabilty function
straightline - use the StraightLine suitabilty function
exponential - use the Exponential suitabilty function
newexponentiall50 - use the ExponentialL50 suitabilty function
richards - use the Richards suitabilty function
andersen - use the Andersen suitabilty function
andersenfleet - use the Andersen suitabilty function, modified for fleets
gamma - use the Gamma suitabilty function

For the following suitability functions, the convention used is to represent the length group of the prey by l, and the length group of the predator by L.

4.9.1 Constant Suitability Function

This is a constant suitability function, where there is no dependence on either the length of the predator or the length of the prey as given by the following equation:

$$S(l, L) = \alpha \tag{4.24}$$

Hence, to specify a constant suitability function, the file format required is:

```
<preyname>    function constant <alpha>
```

4.9.2 StraightLine Suitability Function

This is a suitability function that has no dependence on the length of the predator, and a linear dependence on the length of the prey as given by the following equation:

$$S(l, L) = \alpha l + \beta \tag{4.25}$$

Hence, to specify a straight line suitability function, the file format required is:

```
<preyname>    function straightline <alpha> <beta>
```

4.9.3 Exponential Suitability Function

This is a suitability function that has a logarithmic dependence on both the length of the predator and the length of the prey as given by the following equation:

$$S(l, L) = \frac{\delta}{1 + e^{(-\alpha - \beta l - \gamma L)}} \tag{4.26}$$

Hence, to specify this suitability function, the file format required is:

```
<preyname>    function exponential <alpha> <beta> <gamma> <delta>
```

4.9.4 ExponentialL50 Suitability Function

This is a suitability function that has no dependence on the length of the predator, and a logarithmic dependence on the length of the prey as given by the following equation:

$$S(l, L) = \frac{1}{1 + e^{-\alpha(l - l_{50})}} \tag{4.27}$$

Note that the prey length dependence is actually dependant on the difference between the length of the prey and l_{50}, which is the length of the prey with a 50% probability of predation. Hence, to specify this suitability function, the file format required is:

```
<preyname>    function newexponentiall50 <alpha> <l50>
```

4.9.5 Richards Suitability Function

This is a suitability function that is an extension to the Exponential suitiability function, and has a logarithmic dependence on both the length of the predator and the length of the prey as given by the following equation:

$$S(l, L) = \left(\frac{p_3}{1 + e^{(-p_0 - p_1 l - p_2 L)}} \right)^{\frac{1}{p_4}} \tag{4.28}$$

Hence, to specify this suitability function, the file format required is:

```
<preyname>    function richards <vector of 5 parameters>
```

4.9.6 Andersen Suitability Function

This is a more general suitability function that is dependant on the ratio of the predator length to the prey length as given by the following equation:

$$S(l, L) = \begin{cases} p_0 + p_2 e^{-\frac{(\ln \frac{L}{l} - p_1)^2}{p_4}} & \text{if } \ln \frac{L}{l} \leq p_1 \\ p_0 + p_2 e^{-\frac{(\ln \frac{L}{l} - p_1)^2}{p_3}} & \text{if } \ln \frac{L}{l} > p_1 \end{cases} \tag{4.29}$$

Note that the log of the ratio of the predator/prey lengths is bounded, to ensure that the suitability function is always well defined. To specify this suitability function, the file format required is:

```
<preyname>    function andersen <vector of 5 parameters>
```

4.9.7 Andersen Fleet Suitability Function

This is a modified version of the Andersen suitability function that is dependant on the ratio of a parameter to the prey length as given by the following equation:

$$S(l, L) = \begin{cases} p_0 + p_2 e^{-\frac{(\ln \frac{p_5}{l} - p_1)^2}{p_4}} & \text{if } \ln \frac{p_5}{l} \leq p_1 \\ p_0 + p_2 e^{-\frac{(\ln \frac{p_5}{l} - p_1)^2}{p_3}} & \text{if } \ln \frac{p_5}{l} > p_1 \end{cases} \tag{4.30}$$

Note that the log of the ratio of the fleet parameter/prey length is bounded, to ensure that the suitability function is always well defined. Compared to the standard Andersen suitability function, the additional parameter is used to replace the predator length and is often used as a proxy for the mesh size used by the fleets. To specify this suitability function, the file format required is:

```
<preyname>    function andersenfleet <vector of 6 parameters>
```

4.9.8 Gamma Suitability Function

This is a suitability function that is only dependant on the prey length as given by the following equation:

$$S(l, L) = \left(\frac{l}{(\alpha - 1)\beta\gamma} \right)^{(\alpha-1)} e^{(\alpha-1-\frac{l}{\beta\gamma})} \tag{4.31}$$

Hence, to specify this suitability function, the file format required is:

```
<preyname>    function gamma <alpha> <beta> <gamma>
```

This is a suitability function that is more suitable for use when considering the predation by a fleet (see section 7.6), where the parameter $< \gamma >$ would represent the size of the mesh used by the fleet (specified in centimetres).

4.10 Initial Conditions

The initial conditions section of stock file specifies the stock population at the start of the simulation (ie. at the beginning of the first timestep specified in the "time" file). This includes setting up the population size, the length distribution and the mean weight for each length group. This is done by specifying the minimum and maximum age and length for the stock on this timestep, and either specifying parameters to allow Gadget to create a stock distribution based on a Normal distribution, or the numbers that make up the stock distribution required.

The format for the initial conditions section of the stock file is given below:

```
initialconditions
minage                  <minimum age for the initial stock>
maxage                  <maximum age for the initial stock>
minlength               <minimum length for the initial stock>
maxlength               <maximum length for the initial stock>
dl                      <step size for the initial length groups>
sdev                    <standard deviation multiplier>
<initial stock distribution data>
```

The optional <sdev> value is used to scale the standard deviation of the length of the initial stock. The standard deviation used in calculating the length distribution will be multiplied by this value. If it is not specified, then it is assumed to have a value of 1 (ie, no scaling will take place). The optional <dl> value is used to set the length group divisions for the initial stock population. If this is not specified, then it is assumed that the initial population will have the same value for <dl> as for the stock, as specified in the main stock file.

There are three formats for the initial stock distribution data, as given below:

Normal Condition - use a Normal function to generate the length distribution, with a relative condition factor to assign a mean weight to the initial population
Normal Parametric - use a Normal function to generate the length distribution, with a parametric weight-length relationship to calculate a mean weight for the initial population
Numerical - specify the numbers, and the corresponding mean weights, for the initial population

4.10.1 Normal Condition Distribution

To specify an initial stock with the lengths given by a Normal distribution, and using a relative condition factor to generate the weights, the main stock file needs to specify the name of a datafile containing the information about the Normal distribution, as shown below:

```
normalcondfile          <name of the initial stock data file>
```

The initial stock file contains all the information that Gadget requires to construct an initial population of the stock. Gadget will construct a population of 10,000 fish for each age group, with the length groups for these age groups having a Normal distribution about a specified mean length with a specified standard deviation. The mean weight for this initial population is calculated by multiplying the reference weight (specified in the main stock file) by a relative conditioning factor (which is typically set to 1).

To get from a population with 10,000 fish in each age group (for each area) to the initial population used in the model, each age group is multiplied by an age weighting factor and an area weighting factor.

Hence, the format for the initial stock file is given below:

```
<age>  <area>  <age factor>  <area factor>  <mean>  <stddev>  <relcond>
```

4.10.2 Normal Parametric Distribution

To specify an initial stock with the lengths given by a Normal distribution, and using a weight-length relationship to generate the weights, the main stock file needs to specify the name of a datafile containing the information about the Normal distribution, as shown below:

```
normalparamfile         <name of the initial stock data file>
```

The initial stock file contains all the information that Gadget requires to construct an initial population of the stock. Gadget will construct a population of 10,000 fish for each age group, with the length groups for these age groups having a Normal distribution about a specified mean length with a specified standard deviation. The mean weight for this initial population is then calculated from the standard weight-length relationship, as given in equation 4.32 below:

$$W = \alpha L^{\beta} \tag{4.32}$$

To get from a population with 10,000 fish in each age group (for each area) to the initial population used in the model, each age group is multiplied by an age weighting factor and an area weighting factor.

Hence, the format for the initial stock file is given below:

```
<age>  <area>  <age factor>  <area factor>  <mean>  <stddev>  <alpha>  <beta>
```

4.10.3 Numerical Distribution

An alternative approach is to define the initial stock population by specifying an age-length table, along with the mean weight, for the initial stock. This approach requires the main stock file to specify a data file to give the age-length table, as shown below:

```
numberfile              <name of the initial stock data file>
```

The initial stock file contains the age-length table for the initial population. This file specifies the population, and mean weight, of the stock for each area, age group and length group combination, as shown below:

```
<area>  <age>  <length>  <number>  <weight>
```

Note that the <length> value refers to the minimum length of the age-length cell that the initial population will be put into.

4.11 Migration

The migration section of the stock file determines if, and how, the stock will migrate in the Gadget model. The migration of a stock within a Gadget model is calculated based on migration matrices. The format for the first part of the migration section is given below:

```
doesmigrate            <0 or 1> ; 0 for no migration, 1 for migration
```

If the stock does not migrate, then the following section doesn't apply, and the next section of the input file is the maturation, given in section 4.12 below. If the stock does migrate, then further information about the migration is required. There are currently 2 formats for the migration data, as given in the list below:

Migration Matrices - specify the migration matrices to be used
Migration Ratios - specify the ratio of the stock that moves between areas, and let Gadget calculate the required matrices based on these ratios

4.11.1 Migration Matrices

To specify the migration of the stock by defining the migration matrices directly, the main stock file needs to specify the name of a datafile containing information about when the migration will take place, and the name of a datafile containing information about the migration matrices, as shown below:

```
doesmigrate            1
yearstepfile           <name of the migration timestep data file>
definematrices         <name of the migration matrix data file>
```

The migration timestep file contains a simple list of the timesteps that the migration will take place on, along with the name of the migration matrix that is to be used on each timestep, as shown below:

```
<year>  <step>  <migration matrix>
```

The migration matrix file contains the migration matrices to be used to move the stock between the various areas within the Gadget model. Each matrix is listed, starting with the keyword [migrationmatrix], followed by the name of the migration matrix, and then the matrix to be used, as shown below:

```
[migrationmatrix]
name   <migration matrix>
<migration matrix data>
```

The <migration matrix> data should be a square n x n matrix, for the n areas that the stock is defined on, with the proportion moving **from** each area given in the columns, and the proportion moving **to** each area given in the rows. Since migration shouldn't change the total number of fish in the model, each column in the matrix should sum to one.

An example of a migration matrix data for a stock that is defined on two areas is given below:

```
[migrationmatrix]
name    example
0.6     0.3
0.4     0.7
```

For the migration of the stock in the example shown above, 60% of the stock that are on area 1 at the start of the migration will move to area 1 (ie. they will stay there) and the remaining 40% will move to area 2. For the stock that start the timestep on area 2, 30% will move to area 1 and the remaining 70% will stay on area 2.

4.11.2 Migration Ratios

An alternative approach is to specify the migration of the stock by defining the ratio of the stock that will migrate between given areas, and let Gadget calculate the resulting migration matrices to use within the model. To use this format for the migration data, the main stock file needs to specify the name of a datafile containing information about when the migration will take place, and the name of a datafile containing information about the migration ratios, as shown below:

```
doesmigrate           1
yearstepfile          <name of the migration timestep data file>
defineratios          <name of the migration ratio data file>
```

The migration timestep file contains a list of the timesteps that the migration will take place on, along with the name of the migration matrix that is to be used on each timestep, in the same format as for the Migration Matrices, as shown above.

The migration ratio file contains the ratios to be used by Gadget to construct the migration matrices to move the stock between the various areas within the Gadget model. Each matrix is listed, starting with the keyword [migrationmatrix], followed by the name of the migration matrix, and then a list of the ratios to be used to construct the matrix, as shown below:

```
[migrationmatrix]
name    <migration matrix>
<from>  <to>    <ratio>
```

The migration ratio data is a list of the ratio of the stock that moves from the <from> area to the <to> area. Gadget will attempt to construct the migration matrix based on these ratios. Where the data is incomplete, Gadget will assume that there is no migration between areas (ie. fish will stay on the area that they are on at the start of the timestep). If Gadget cannot construct a (unique) migration matrix, then the model will stop the current simulation and display an error message.

An example of a migration ratio data for a stock that is defined on two areas is given below:

```
[migrationmatrix]
name    example
1       2       0.4
2       1       0.3
```

For the migration of the stock in the example shown above, 40% of the stock that are on area 1 at the start of the migration will move to area 2, and the remaining 60% will stay on area 1. For the stock that start the timestep on area 2, 30% will move to area 1 and the remaining 70% will stay on area 2.

4.12 Maturation

The maturation section of the stock file determines if, and how, the stock will mature in the Gadget model. The format for the first part of the maturation section is given below:

```
doesmature              <0 or 1> ; 0 for no maturation, 1 for maturation
```

If the stock does not mature, then the following sections don't apply, and the next section of the input file is the movement, given in section 4.13 below. If the stock does mature, then there are various different functions that describe how the stock can mature. The type of maturity function is denoted by a name, as shown below, and then the data required for that maturity function is given in a datafile. Thus the format for the maturity data, in the main stock file, is given below:

```
doesmature              1
maturityfunction        <maturity function>
maturityfile            <name of the maturity data file>
```

The <maturity function> defines the function that is to be used to calculate how the stock will mature. Currently there are 4 maturity functions defined, and the valid maturity function names are:

newconstant - use the Constant maturation function
newconstantweight - use the ConstantWeight maturation function
fixedlength - use the FixedLength maturation function
continuous - use the Continuous maturation function

The format for the data in the maturity data file is dependent on the maturity function that is to be used. All the maturity functions require the name of the mature stocks that the immature stock will mature into, and the ratio of the maturing part of the immature stock that is to mature into each mature stock. This allows for part of an immature stock to mature into more than one mature stock, so for example, an immature stock could mature into either a male mature stock or a female mature stock.

The maturity functions calculate the proportion of the fish in each age-length group that will become mature, and then move these fish from the age-length group for the current stock into the corresponding age-length group for the mature stock. Note that there is a check to ensure that the corresponding age-length group exists for the mature stock, and if it doesn't then the fish don't become mature and will stay in the immature stock.

4.12.1 Constant Maturity Function

This maturity function calculates the proportion of an age-length group of an immature stock that becomes mature according to the maturity equation 4.33 given below:

$$P(l,a) = \frac{1}{1 + e^{-\alpha(l-l_{50})-\beta(a-a_{50})}} \tag{4.33}$$

The maturity term is a function of l_{50} and a_{50}, which are the length and age where 50% of the stock are mature. For this maturation function, it is assumed that maturation is an annual event, taking place on the same timesteps in each year. The file format for this maturity function is the similar to that for the Continuous maturity function, and is shown below:

```
maturestocksandratios       <stockname i>  <ratio i> ; for each stock i
coefficients                <alpha>  <l50>  <beta>  <a50>
maturitysteps               <vector of timesteps>
```

The <maturitysteps> vector is a vector of timesteps that the maturation process will take place on.

4.12.2 ConstantWeight Maturity Function

This maturity function is an extension to the Constant maturity function, with the maturity proportion being calculated according to the maturity equation 4.34 given below:

$$P(l,a) = \frac{1}{1 + e^{-\alpha(l-l_{50})-\beta(a-a_{50})-\gamma(k-k_{50})}} \tag{4.34}$$

The maturity function has been extended to include the relative condition of the stock (calculated by dividing the current weight by the reference weight) and k_{50} which is the relative condition where 50% of the stock is mature. The file format for this maturity function is the similar to that for the Constant maturity function, and is shown below:

```
maturestocksandratios       <stockname i>  <ratio i> ; for each stock i
coefficients                <alpha>  <l50>  <beta>  <a50>  <gamma>  <k50>
maturitysteps               <vector of timesteps>
```

4.12.3 FixedLength Maturity Function

This maturity function takes a different approach, and bases the proportion of the immature stock that matures on the length of the immature stock, as the length varies through the year. This is approach assumes that the maturation process is the same for each year. The proportion of the immature stock that matures is given by the equation 4.35 below:

$$P(l,a) = \begin{cases} 1 & \text{if there is an } i \text{ such that } s_i \text{ is the current step and } l > l_i \\ 0 & \text{otherwise} \end{cases} \tag{4.35}$$

For each timestep in the year, the stock is assumed to mature when the length of the fish reaches a certain value. This length can change for each timestep. This information is given in a file with the format specified below:

```
maturestocksandratios       <stockname i>  <ratio i> ; for each stock i
maturitysteps               <vector of timesteps>
maturitylengths             <vector of lengths>
```

Note that the <maturitysteps> and the <maturitylengths> vectors need to be the same size.

4.12.4 Continuous Maturity Function

This maturity function calculates the proportion of an age-length group of an immature stock that becomes mature according to the maturity equations 4.36 and 4.37 given below:

$$P(l,a) = \frac{1}{1-M}\frac{dM}{dt} \tag{4.36}$$

$$M(l_t,a_t) = \frac{1}{1 + e^{-\alpha(l_t-l_{50})-\beta(a_t-a_{50})}} \tag{4.37}$$

The maturity term is a function of l_{50} and a_{50}, which are the length and age where 50% of the stock are mature. This is a continuous process, with the maturity proportion being calculated on every timestep. The file format for this maturity function is given below:

```
maturestocksandratios       <stockname i>  <ratio i> ; for each stock i
coefficients                <alpha>  <l50>  <beta>  <a50>
```

4.13 Movement ("Transition")

The movement section of the stock file determines if, and how, the stock will move (into a different stock) in the Gadget model. This allows for a Gadget model to be set up with different stock files for stock that is the same species, but with differing properties (for instance age or maturity status) and for the entries to move between these stocks when required. For the current version of Gadget, the only movement between stocks that is valid is for the stock in the oldest age group of one stock to move into a different stock. The format for the first part of the movement section is given below:

```
doesmove                <0 or 1> ; 0 for no movement, 1 for movement
```

If the stock does not move, then the following section doesn't apply, and the next section of the input file is the renewal, given in section 4.14 below. If the stock does move, then the information required to define the movement is the timestep for the movement to occur (since the movement is assumed to be an annual event) and the names of the stocks to move the oldest age group into, along with the ratio of the oldest age group that will move into that particular stock. Thus the full format for the movement of the stock is given below:

```
doesmove                    1
transitionstocksandratios   <stockname i>  <ratio i> ; for each stock i
transitionstep              <timestep for the stock to move>
```

The movement function simples move the fish from the oldest age-length groups for the current stock into the corresponding age-length group for the stock that the fish will move into (in a similar manner to that used for the maturation, given in section 4.12). Note that there is a check to ensure that the corresponding age-length group exists, and if it doesn't then the fish don't move and will stay in the oldest age group for current stock, which is modelled as a plus group.

4.14 Renewal ("Recruitment")

The renewal section of the stock file determines if, and how, the stock will be renewed in the Gadget model. The format for the first part of the renewal section is given below:

```
doesrenew               <0 or 1> ; 0 for no renewal, 1 for renewal
```

If the stock does not renew, then the following sections don't apply, and the next section of the input file is the spawning, given in section 4.15 below. If the stock does renew, then further information is required about the renewal data. This is given in a separate file, so the format for the renewal data, in the main stock file, is given below:

```
doesrenew               1
minlength               <minimum length for the recruits>
maxlength               <maximum length for the recruits>
dl                      <step size for the recruits>
<renewal distribution data>
```

The <dl> value is optional, and can be used to specify a different step size for the recruits than for the stock that the recruits will be added to. If this value is not specified here then the recruits will be given the same step length as for the stock they will be added to.

In a similar manner to the format for the initial population (see section 4.10) there are three formats for the renewal distribution data, as given below:

Normal Condition - use a Normal function to generate the length distribution, with a relative condition factor to assign a mean weight to the new recruits
Normal Parametric - use a Normal function to generate the length distribution, with a parametric weight-length relationship to calculate a mean weight for the new recruits
Numerical - specify the numbers, and the corresponding mean weights, for the new recruits

4.14.1 Normal Condition Distribution

To specify renewal data with the lengths given by a Normal distribution, and using a relative condition factor to generate the weights, the main stock file needs to specify the name of a datafile containing the information about the Normal distribution, as shown below:

```
normalcondfile          <name of the renewal data file>
```

The renewal file contains all the information that Gadget requires to construct the renewal data for the stock. For each timestep and area, this file lists the the age of the recruits (which would typically match the minimum age of the stock that the recruits are to be added to), the number of recruits (in units of 10,000 fish), parameters used to define the Normal distribution for length groups of the recruits, and the relative condition factor, used along with the reference weight to assign a mean weight for the recruits.

Hence, the format for the renewal file is given below:

```
<year>  <step>  <area>  <age>  <number>  <mean>  <stddev>  <relcond>
```

4.14.2 Normal Parametric Distribution

To specify renewal data with the lengths given by a Normal distribution, and using a weight-length relationship to generate the weights, the main stock file needs to specify the name of a datafile containing the information about the Normal distribution, as shown below:

```
normalparamfile         <name of the renewal data file>
```

The renewal file contains all the information that Gadget requires to construct the renewal data for the stock. For each timestep and area, this file lists the the age of the recruits (which would typically match the minimum age of the stock that the recruits are to be added to), the number of recruits (in units of 10,000 fish), parameters used to define the Normal distribution for length groups of the recruits, and parameters to define the weight-length relationship for these recruits.

Hence, the format for the renewal file is given below:

```
<year>  <step>  <area>  <age>  <number>  <mean>  <stddev>  <alpha>  <beta>
```

4.14.3 Numerical Distribution

An alternative approach is to define the renewal data by specifying an age-length table, along with the mean weight, for each timestep that will have new fish added to the stock. This approach requires the main stock file to specify a data file to give the age-length table, as shown below:

```
numberfile              <name of the renewal data file>
```

The renewal file contains data for an age-length table for each timestep that will have new fish added to the stock. This file specifies the number, and mean weight, of the new fish for each timestep, area, age group and length group combination, as shown below:

```
<year>  <step>  <area>  <age>  <length>  <number>  <weight>
```

Note that the <length> value refers to the minimum length of the age-length cell that the new fish will be put into.

4.15 Spawning

The spawning section of the stock file determines if, and how, the stock will spawn in the Gadget model. This covers the mortality and weight loss from the stock due to the spawning process, and optionally the creation of a new spawned stock. The format for the first part of the spawning section is given below:

```
doesspawn              <0 or 1> ; 0 for no spawning, 1 for spawning
```

If the stock does not spawn, then the following sections don't apply, and the next section of the input file is the straying information, given in section 4.17 below. If the stock does spawn, then further information is required about the spawning data. This is given in a separate file, so the format for the spawning data, in the main stock file, is given below:

```
doesspawn              1
spawnfile              <name of the spawning data file>
```

The spawning data file defines what happens to the stock as it spawns. The spawning is length-dependent, and the affect that spawning has on each length group of the mature stock is given by the spawning equations 4.38 and 4.39 below:

$$N = N\big(1 + P(e^{-m} - 1)\big) \tag{4.38}$$

$$W = W\frac{(1 + P((2 - w)e^{-m} - 1))}{(1 + P(2e^{-m} - 1))} \tag{4.39}$$

where:
$< \mathrm{N} >$ is the population of the age-length group
$< \mathrm{W} >$ is the mean weight of the population of the age-length group
$< \mathrm{P} >$ is the proportion of the length group that will spawn
$< m >$ is the spawning mortality for that length group
$< w >$ is the spawning weight loss for that length group

In equation 4.38, the population of the age-length cell of the mature stock is reduced due to the spawning mortality of the fish that spawn. In equation 4.39, the mean weight of the population in the age-length cell is adjusted to take the reduction in weight of the fish that spawn, and the change in population, into account.

Spawning is considered to be an annual event, that takes place on the same timestep and the same area in each year that the spawning occurs. To model the spawning process as it affects the parent (without the creation of a child stock), it is necessary to specify the timestep and area that the spawning will take place on, and the length selection functions to determine proportion of each length group that will spawn, and the spawning mortality and weight loss of those that spawn. This is done in the spawning data file, as shown below:

```
spawnsteps             <vector of timesteps>
spawnareas             <vector of areas>
firstspawnyear         <first year that the spawning occurs>
lastspawnyear          <last year that the spawning occurs>
onlyparent
proportionfunction     <see Length Selection>
mortalityfunction      <see Length Selection>
weightlossfunction     <see Length Selection>
```

The optional <firstspawnyear> and <lastspawnyear> values define the first and last years on which the spawning process will take place. This means that it is possible to define recruits (see section 4.14 above) for some years in the model and use a spawning process in other years in the model. If these are not specified in the input file, then it assumed that the spawning process will take place on all years in the simulation, and these will default to the first year and last year in the simulation.

Alternatively, it is possible to consider the spawning process as part of a through life-cycle model, and so the spawning process can create recruits to be added to one or more stocks (in a similar manner to the renewal data, given in section 4.14 above). The total number of recruits given by the spawning process is added to the youngest age group of the spawned stock at the start of the following timestep. The lengths of the spawned stock are distributed with a Normal distribution about a specified mean length <mean> with a standard deviation <stddev>. The mean weight of the fish in these age length cells is given from equation 4.32, with α and β specified in the spawning data file. Hence, if the spawning process is to calculate a number of recruits to the model, the format for the spawning data file is shown below:

```
spawnsteps              <vector of timesteps>
spawnareas              <vector of areas>
firstspawnyear          <first year that the spawning occurs>
lastspawnyear           <last year that the spawning occurs>
spawnstocksandratios    <stockname i> <ratio i> ; for each stock i
proportionfunction      <see Length Selection>
mortalityfunction       <see Length Selection>
weightlossfunction      <see Length Selection>
recruitment             <functionname>  <parameters>
stockparameters         <mean>  <stddev>  <alpha>  <beta>
```

The recruitment <functionname> defines which recruitment function to use to calculate the number of recruits to be added to the spawned stock. Currently there are 4 recruitment functions defined, and the valid recruitment function names are:

fecundity - use the Fecundity recruitment function
simplessb - use the SimpleSSB recruitment function
ricker - use the Ricker recruitment function
bevertonholt - use the BevertonHolt recruitment function

4.15.1 Fecundity Recruitment Function

This recruitment function calculates the number of recruits to be added to the spawned stock as a function of the length, age, number and weight of the spawning stock, given by the following equation:

$$R = p_0 \sum_{ages} \sum_{lengths} l^{p_1} a^{p_2} N_{al}^{p_3} W_{al}^{p_4} \tag{4.40}$$

Hence, to specify this recruitment function, the file format required is:

```
recruitment             fecundity  <vector of 5 parameters>
```

4.15.2 SimpleSSB Recruitment Function

This recruitment function calculates the number of recruits to be added to the spawned stock as a simple proportion of the spawning stock biomass, given by the following equations:

$$S = \sum_{ages} \sum_{lengths} N_{al} W_{al} \tag{4.41}$$

$$R = \mu S \tag{4.42}$$

Hence, to specify this recruitment function, the file format required is:

```
recruitment            simplessb  <mu>
```

4.15.3 Ricker Recruitment Function

This recruitment function calculates the number of recruits to be added to the spawned stock, as a function of the spawning stock biomass (see equation 4.41), based on the Ricker recruitment relationship given by the following equation:

$$R = \mu S e^{-\lambda S} \tag{4.43}$$

Hence, to specify this recruitment function, the file format required is:

```
recruitment            ricker  <mu>  <lambda>
```

4.15.4 BevertonHolt Recruitment Function

This recruitment function calculates the number of recruits to be added to the spawned stock, as a function of the spawning stock biomass (see equation 4.41), based on the Beverton Holt recruitment relationship given by the following equation:

$$R = \frac{\mu S}{\lambda + S} \tag{4.44}$$

Hence, to specify this recruitment function, the file format required is:

```
recruitment            bevertonholt  <mu>  <lambda>
```

4.16 Length Selection

The length selection function determines the proportion of the length group that will be selected, in a similar way to the suitability functions (see section 4.9 above). To define a length selection function, it is necessary to specify the function, the name of the function ans the parameters for the function, as shown below:

```
<function>    <functionname> <parameters>
```

The <function> defines how the length selection function will be used (for an example, see section 4.15 above). The <functionname> defines which selection function is to be used to calculate the selection of the stock. Currently there are three selection functions defined, and the valid length selection function names are:

constant - use the Constant selection function
straightline - use the StraightLine selection function
exponential - use the Exponential selection function

4.16.1 Constant Selection Function

This is a selection function, where there is no dependence on the length of the stock is given by the following equation:

$$S(l) = \alpha \tag{4.45}$$

Hence, to specify a constant selection function, the file format required is:

```
<function>    constant  <alpha>
```

4.16.2 StraightLine Selection Function

This is a selection function that has a linear dependence on the length of the stock is given by the following equation:

$$S(l) = \alpha l + \beta \tag{4.46}$$

Hence, to specify a straight line selection function, the file format required is:

```
<function>    straightline  <alpha>  <beta>
```

4.16.3 Exponential Selection Function

This is a selection function that has a logarithmic dependence on the length of the stock is given by the following equation:

$$S(l) = \frac{1}{1 + e^{\alpha(l - l_{50})}} \tag{4.47}$$

Note that the stock length dependence is actually dependant on the difference between the length of the stock and l_{50}, which is the length of the stock with a 50% probability of selection. Hence, to specify this selection function, the file format required is:

```
<function>    exponential  <alpha>  <l50>
```

4.17 Straying

The straying section of the stock file determines if, and how, the stock will stray from one substock to another substock in the Gadget model. The format for the first part of the straying section of the stock file is given below:

```
doesstray             <0 or 1> ; 0 for no straying, 1 for straying
```

If the stock does not stray, then the following section doesn't apply, and the stock file is complete. If the stock does stray, then further information is required about the straying data. This is given in a separate file, so the format for the straying data, in the main stock file, is given below:

```
doesstray             1
strayfile             <name of the straying data file>
```

The straying data file defines what happens to the stock as it strays from one substock to another. The straying is length-dependent, so that a proportion of each length group (over all age groups) will move to the corresponding length group in a different substock. This process can be thought of as an extension to the transition process (see section 4.13).

Straying is considered to be an annual event, that takes place on the same timestep and the same area in each year. To model the straying process in Gadget, it is necessary to specify the timestep and the area that the straying will take place on, the the names of the stocks to move the straying fish into, along with the ratio of the fish that will move into that particular stock (in a similar manner to that used for the transition process) and the length selection function to determine the proportion of the length group that will stray. This is done in the straying data file, as shown below:

```
straysteps             <vector of timesteps>
strayareas             <vector of areas>
straystocksandratios   <stockname i> <ratio i> ; for each stock i
proportionfunction     <see Length Selection>
```

Chapter 5

Tag Files

The tag files contain the information about the tagging experiments that are to be included in the Gadget model. Gadget will keep track of the number, and proportion, of fish in an age-length cell that have been tagged for a tagging experiment. This information can then be compared to the recaptures from that tagging experiment when calculating a likelihood score (see Recaptures, section 8.9).

To define tagged populations in the Gadget model, the "main" file must contain a list of the data files that contain the description of the tagging experiments, and the format for this is shown below:

```
[tagging]
tagfiles              <names of the tag files>
```

The main tag file lists the tagging experiments, along with basic information about the experiments and the name of the datafile that contains information about the number of fish tagged for the tagging experiment. The format for this file shown below, with each new tagging experiment starting with the keyword [component]:

```
[component]
tagid                 <name of the tagging experiment>
stock                 <name of the tagged stock>
tagarea               <area that the tagging took place>
endyear               <year of last recapture>
tagloss               <proportion of tagged fish that are lost>
numbers               <see Tagging Numbers>
```

Each tagging experiment is defined by specifying the name of the tagging experiment, the stock that is tagged for the tagging experiment, and the area that the tagging took place on. Note that it is currently possible to only tag one stock, on one area, for each tagging experiment.

The optional <endyear> value is used to define the end of a tagging experiment, and should be set to the year of the last expected recapture from the tagging experiment. This can be used to reduce the calculation time. If this is not specified, then it is assumed that the tagging experiment will run until the end of the simulation.

The <tagloss> value is used to remove tagged fish from the model. For each time step, a number of tags can be lost from the tagging experiment. The number of tagged fish that remain in the tagging experiment is given by the equation 5.1 below:

$$N = N(e^{-t}) \tag{5.1}$$

where:
$< \mathrm{N} >$ is the number of tagged fish in an age-length group
$< t >$ is the proportion of tagged fish that are lost, as specified in the input file

5.1 Tagging Numbers

The numbers section of the tag file gives the number of fish tagged for the tagging experiment. This is given in a column format in a separate file, so the main tag file simply gives the name of this tag data file, as shown in the example below:

```
numbers                 <name of data file>
```

For the tag datafile, this is a simple list of tag identifier, timestep, length and then the number of fish tagged, for that tag/timestep/length combination. This format is shown below:

```
<tagid>  <year>  <step>  <length>  <number>
```

Note that the <length> value refers to the minimum length of the length cell that the tagged fish will be put into.

Chapter 6

Otherfood Files

The otherfood files contain the information about non-dynamic prey that is available for the predators to consume. The otherfood acts as a prey that is always available, and it is used to avoid the situation where the non-availability of a prey stock prevents the predators from growing as expected.

To define otherfood in the Gadget model, the "main" file must contain a list of the data files that contain the description of the otherfood, and the format for this is shown below:

```
[otherfood]
otherfoodfiles          <names of the otherfood files>
```

The main otherfood file lists the otherfood, along with basic information about the food and the name of the datafile that contains information about the amount of food that is available for the predators to eat. The format for this file shown below, with each new otherfood starting with the keyword [component]:

```
[component]
foodname                <food name>
livesonareas            <areas>
lengths                 <min>  <max>
energycontent           <energy value>
amount                  <see Food Amounts>
```

The otherfood is defined by specifying the otherfood name and areas it is available for consumption on and the minimum and maximum length of food (for compatibility with the dynamic stock predation and printer classes).

6.1 Food Amounts

The amounts section of the otherfood file gives the biomass of otherfood that is available for the predators to eat. This data is listed in a column format in a separate file, so the main otherfood file simply gives the name of this otherfood data file, as shown in the example below:

```
amount                  <name of data file>
```

For the otherfood datafile, this is a simple list of year, timestep, area, food name and then the biomass of the food available for the predators to eat, for that timestep/area combination. This format is shown below:

```
<year>  <step>  <area>  <food name>  <amount>
```

Chapter 7

Fleet Files

The fleet files contain the information about the fleets that are reducing the population of the stocks in the Gadget model. The fleets act as a simple predator in the model, with the landings data treated as the fleets "consumption" of the stock that is caught.

To define fleets in the Gadget model, the "main" file must contain a list of the data files that contain the description of the fleets, and the format for this is shown below:

```
[fleet]
fleetfiles              <names of the fleet files>
```

There are 5 types of fleets implemented in Gadget, and the main fleet file lists the fleets and their type, along with information about the fleet and the name of the datafile that contains information about the landings. The format for this file shown below, with each new fleet starting with the keyword [component]:

```
[component]
<type>                  <fleetname>
<fleet data>
```

The fleet data for each fleet type is covered in the sub sections below. The <type> defines the type of fleet for the <fleetname> fleet, and the 5 valid fleet types that can used in Gadget are:

TotalFleet
NumberFleet
LinearFleet
EffortFleet
QuotaFleet

7.1 TotalFleet

The fleet type used that creates a predator based on the landings data (by biomass) for the fleet is called "TotalFleet". This total amount landed is then split between the various stocks, and length groups of the stocks, according to equation 7.1 below:

$$C_s(l) = \frac{ES_s(l)N_{sl}W_{sl}}{\sum_{stocks}\sum_{lengths} S_s(l)N_{sl}W_{sl}} \tag{7.1}$$

where:
$< E >$ is the biomass caught by the fleet

< N > is the number of stock in the length cell
< W > is the mean weight of the stock in the length cell
< S > is the suitability function (see section 7.6)

This fleet type is defined by specifying the fleet name and areas it operates on, along with a suitability function for each stock that the fleet will catch and a data file listing the biomass that the fleet will catch. The file format for the TotalFleet is given below:

```
[component]
totalfleet              <fleetname>
livesonareas            <areas>
multiplicative          <multi>
suitability             <see Fleet Suitability>
amount                  <see Fleet Amounts>
```

The optional <multi> value is a multiplicative constant used to scale the data if required - the default value for this multiplier is 1 (ie. no scaling).

The fleets act as a predator, so Gadget also requires a suitability function to be defined for the predation of the stocks in the model. The total amount that has been landed by the fleet is also required - this is taken from the landings data, based on timestep and area, and is specified in a separate file.

7.2 NumberFleet

The fleet type used that creates a predator based on the number of the stock landed (not the biomass) is called "NumberFleet". This total number caught is then split between the various stocks, and length groups of the stocks, according to equation 7.2 below:

$$C_s(l) = \frac{ES_s(l)N_{sl}}{\sum_{stocks}\sum_{lengths} S_s(l)N_{sl}} \tag{7.2}$$

where:
< E > is the number caught by the fleet
< N > is the number of stock in the length cell
< S > is the suitability function (see section 7.6)

This fleet type is defined by specifying the fleet name and areas it operates on, along with a suitability function for each stock that the fleet will catch and a data file listing the numbers that the fleet will catch. The file format for the NumberFleet is given below:

```
[component]
numberfleet             <fleetname>
livesonareas            <areas>
multiplicative          <multi>
suitability             <see Fleet Suitability>
amount                  <see Fleet Amounts>
```

The optional <multi> value is a multiplicative constant used to scale the data if required - the default value for this multiplier is 1 (ie. no scaling).

The fleets act as a predator, so Gadget also requires a suitability function to be defined for the predation of the stocks in the model. The total number of fish that has been caught by the fleet is also required - this is taken from the landings data, based on timestep and area, and is specified in a separate file.

7.3 LinearFleet

The fleet type used that creates a predator that removes the caught fish based on the available biomass of the stock multiplied by a scaling factor is called "LinearFleet". The biomass caught is then split between the various stocks, and length groups of the stocks, according to equation 7.3 below:

$$C_s(l) = E\Delta t S_s(l) N_{sl} W_{sl} \tag{7.3}$$

where:
< E > is the scaling factor for the stock that is to be caught, per month
$< \Delta t >$ is the length of the timestep
< N > is the number of stock in the length cell
< W > is the mean weight of the stock in the length cell
< S > is the suitability function (see section 7.6)

This fleet type is defined by specifying the fleet name and areas it operates on, along with a suitability function for each stock that the fleet will catch and a data file listing the fishing level for the fleet. The file format for the LinearFleet is given below:

```
[component]
linearfleet            <fleetname>
livesonareas           <areas>
multiplicative         <multi>
suitability            <see Fleet Suitability>
amount                 <see Fleet Amounts>
```

The optional <multi> value is a multiplicative constant used to scale the data if required - the default value for this multiplier is 1 (ie. no scaling).

The fleets act as a predator, so Gadget also requires a suitability function to be defined for the predation of the stocks in the model. The scaling factor to be used when calculating the amount that the fleet will catch is also required, and this is specified in a separate file.

The fleet of type LinearFleet acts a simple predator, and can be used for fleets acting in the future, when the landings data is not available.

7.4 EffortFleet

The fleet type used that creates a predator that removes the caught fish based on the available biomass of the stock multiplied by a scaling factor and a 'catchability' parameter for that stock is called "EffortFleet". The biomass caught is then split between the various stocks, and length groups of the stocks, according to equation 7.4 below:

$$C_s(l) = Eq_s\Delta t S_s(l) N_{sl} W_{sl} \tag{7.4}$$

where:
< E > is the scaling factor for the stock that is to be caught, per month
< q > is the catchability parameter for that stock
$< \Delta t >$ is the length of the timestep
< N > is the number of stock in the length cell
< W > is the mean weight of the stock in the length cell
< S > is the suitability function (see section 7.6)

This fleet type is defined by specifying the fleet name and areas it operates on, along with a suitability function and catchability parameter for each stock that the fleet will catch and a data

file listing the proportion of each stock that the fleet will catch. The file format for the EffortFleet is given below:

```
[component]
effortfleet             <fleetname>
livesonareas            <areas>
multiplicative          <multi>
suitability             <see Fleet Suitability>
catchability            <stock catchability parameters>
amount                  <see Fleet Amounts>
```

The optional <multi> value is a multiplicative constant used to scale the data if required - the default value for this multiplier is 1 (ie. no scaling).

The fleets act as a predator, so Gadget also requires a suitability function to be defined for the predation of the stocks in the model. The scaling factor to be used when calculating the amount that the fleet will catch is also required, and this is specified in a separate file.

The fleet of type EffortFleet is a multi-species extension to the fleet of type LinearFleet (see section 7.3 above). This means that when a fleet is used to catch more than one species (either directly or as bycatch) the different catchability for these species can be taken into account. This catchability parameter is specified by listing the names of the prey and the associated preference value, as shown in the example below:

```
...
catchability
<name of stock 1>       <catchability for stock 1>
<name of stock 2>       <catchability for stock 2>
amount
...
```

7.5 QuotaFleet

The fleet type used that creates a predator that removes the caught fish based on the available biomass of the stock multiplied by a scaling factor set according to a simple harvest control rule is called ”QuotaFleet”. The biomass caught is then split between the various stocks, and length groups of the stocks, according to equation 7.3 (see section 7.3 above). The scaling factor (the parameter ”E” in the equation above) is set according to a simple harvest control rule.

This fleet type is defined by specifying the fleet name and areas it operates on, along with a suitability function for each stock that the fleet will catch, a simple harvest control rule and a data file listing the proportion of each stock that the fleet will catch. The file format for the QuotaFleet is given below:

```
[component]
quotafleet              <fleetname>
livesonareas            <areas>
multiplicative          <multi>
suitability             <see Fleet Suitability>
quotafunction           <function name>
biomasslevel            <vector of biomass levels>
quotalevel              <vector of quota levels>
amount                  <see Fleet Amounts>
```

The optional <multi> value is a multiplicative constant used to scale the data if required - the default value for this multiplier is 1 (ie. no scaling).

The fleets act as a predator, so Gadget also requires a suitability function to be defined for the predation of the stocks in the model. A scaling factor that can be used when calculating the amount that the fleet will catch is also required, and this is specified in a separate file (although this is usually set to 1, since the scaling factor to be used is multiplied by that calculated using the simple harvest control rule below).

The simple harvest control rule that is used to calculate the scaling factor to be used to determine the fishing level is defined by the <function name> value, along with the <biomasslevel> and <quotalevel> vectors. Currently there are 6 quota functions defined, and the valid function names are:

simple - use a simple harvest control rule, based on the biomass of each stock
annual - use a annual harvest control rule, based on the biomass of each stock
simplesum - use a simple harvest control rule, based on the biomass of all the stocks
annualsum - use a annual harvest control rule, based on the biomass of all the stocks
simpleselect - use a simple harvest control rule, based on the biomass of selected stocks
annualselect - use a annual harvest control rule, based on the biomass of selected stocks

Note that if the <function name> value is set to either simpleselect or annualselect, then it is necessary to define which stocks to use to determine the fishing level. This is done be additionally specifying a <selectstocks> vector which is a list of the stocks to use, as given in the list below:

```
...
biomasslevel            <vector of biomass levels>
quotalevel              <vector of quota levels>
selectstocks            <vector of stocks to base the quote on>
amount                  <see Fleet Amounts>
```

The <biomasslevel> vector is a list of n biomass points at which the fishing level will change, and the <quotalevel> vector is the corresponding list of $n + 1$ fishing levels. Note that the first quotalevel value corresponds to the fishing level that will be used when the biomass is between 0 and the first biomasslevel value, so the quotalevel vector must have one more entry than the biomasslevel vector.

An example of valid biomasslevel and quotalevel vectors is given below:

```
biomasslevel            10000  250000
quotalevel              0.1  0.4  0.9
```

The fishing level for the stock in the example shown above would be set to 0.1 if the biomass of the stock is less than 10,000 kilogrammes, 0.4 if the biomass is between 10,000 and 250,000 kilogrammes and 0.9 if the biomass is above 250,000 kilogrammes. Note that the biomasslevel vector has 2 entries, and that the quotalevel vector has a third entry.

7.6 Fleet Suitability

The suitability determines how the fleets act on the stocks that are caught. Since Gadget treats the fleets as predators of the stocks, the format for the suitability functions for the fleets is the same as the format for the suitability functions of the stock when they are acting as a predator. The format for the suitability functions as discussed in section 4.9 above. Note that in the equations for the suitability functions, the 'length' of the predator is a meaningless concept when the predator is a fleet.

7.7 Fleet Amounts

The amounts section of the fleet file gives the landings data for the fleets. This data is listed in a column format in a separate file, so the main fleet file simply gives the name of this fleet data file, as shown in the example below:

```
amount                  <name of data file>
```

For fleets of type TotalFleet or NumberFleet, the data file is a list of year, timestep, area, fleetname and then the amount of catch landed (either biomass or number), taken from landings data, for that timestep/area/fleet combination:

```
<year>  <step>  <area>  <fleetname>  <amount>
```

For fleets of type LinearFleet, EffortFleet or QuotaFleet, the data file is a list of year, timestep, area, fleetname and then the scaling factor to be used when calculating the amount of the catch for that timestep/area/fleet combination:

```
<year>  <step>  <area>  <fleetname>  <scaling factor>
```

Chapter 8

Likelihood Files

The likelihood files are used to define the various likelihood components that are used to calculate the "goodness of fit" of the Gadget model to the available data. Each likelihood component will calculate a likelihood score for that individual component, and there is then a weighted sum of all the likelihood scores to calculate an overall likelihood score. It is this overall likelihood score that the optimiser attempts to minimise during an optimising run.

To define likelihood files in the Gadget model, the "main" file must contain a list of the data files that contain the description of the likelihood classes required, and the format for this is shown below:

```
[likelihood]
likelihoodfiles      <names of the likelihood files>
```

The likelihood files contain a list of various type of likelihood classes, separated by the keyword [component] that control the different likelihood components in the model, the name and weight for that likelihood component and various likelihood data, depending in the likelihood component type. The format of the likelihood files is follows:

```
[component]
name                 <name for the likelihood component>
weight               <weight for the likelihood component>
type                 <likelihood type>
<likelihood data>
```

The likelihood data for each likelihood type is covered in the sub sections below. The <likelihood type> defines the type of likelihood component that is to be used, and there are currently 12 valid likelihood types defined in Gadget. These are:

BoundLikelihood
Understocking
CatchDistribution
CatchStatistics
StockDistribution
SurveyIndices
SurveyDistribution
StomachContent
Recaptures
RecStatistics
MigrationPenalty
MigrationProportion
CatchInKilos

8.1 BoundLikelihood ("Penalty")

The BoundLikelihood likelihood component is used to give a penalty weight to parameters that have moved beyond the bounds, as specified in the parameter file, in the optimisation process. This file does not specify the bounds that are to be used, only the penalty that is to be applied when these bounds are exceeded. Since the Simmulated Annealing (see section 11.2) algorithm will always choose a value for the parameter that is within the bounds, this likelihood component will return a zero likelihood score during an optimisation using that algorithm. However, both the Hooke & Jeeves (see section 11.1) and the BFGS (see section 11.3) algorithms can choose a parameter outside the specified bounds, and so this likelihood component can then return a positive score.

To specify a BoundLikelihood likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                penalty
datafile            <name for the datafile>
```

The datafile defines the penalty that is to be applied to the parameter when it exceeds the bounds, as given by equation 8.1 below:

$$\ell_i = \begin{cases} lw_i(val_i - lb_i)^{p_i} & \text{if } val_i < lb_i \\ uw_i(val_i - ub_i)^{p_i} & \text{if } val_i > ub_i \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

where:
$< val_i >$ is the value of the parameter
$< lw_i >$ is the weight applied when the parameter exceeds the lower bound
$< uw_i >$ is the weight applied when the parameter exceeds the upper bound
$< lb_i >$ is the lower bound
$< ub_i >$ is the upper bound
$< p_i >$ is the power coefficient

Note that when the value of the parameter is exactly equal to the bound, this equation will give a zero likelihood score.

The datafile lists these weights and the power that is to be used for each parameter. The format for this file is shown below:

```
<switch>  <power>  <lower>  <upper>
```

where <lower> is the weighting used when the parameter hits the lower bound, and <upper> is the weighting used when the parameter hits the upper bound, for the parameter with the name <switch>.

It is possible to define a default penalty that is used for all switches that are not defined separately. To do this, simply enter a line in the data file with the switch name given as "default", and then the power, lower and upper weights that are required. For example:

```
default    2        1000     1000
```

would define a default penalty, where the lower and upper weights were 1000, and the power was 2.

8.2 Understocking

The Understocking likelihood component calculates a penalty that is applied if there are an insufficient number of a particular prey to meet the requirements of the predators. In the case of a fleet, this means that the landings data indicates that more fish have been landed than there are fish in the model, for that timetep and area combination. A well defined model will have a zero likelihood score from this component. The likelihood component that is used is the sum of squares of the overconsumption, given by the equation below:

$$\ell = \sum_{time} \sum_{areas} \Big(\sum_{preys} U_{trp} \Big)^p \tag{8.2}$$

where:
< U > is the understocking that has occurred in the model
< p > is the power coefficient (which should be 2 for sum of squares fit)

To specify an Understocking likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                understocking
powercoeff          <power>
```

The <power> value is optional, and if this is not given, the power coefficient is assumed to be 2, giving a sum of squares equation for this likelihood component.

8.3 CatchDistribution

The CatchDistribution likelihood component is used to compare distribution data sampled from the model with distribution data sampled from landings or surveys. The distribution data can either be aggregated into age groups (giving a distribution of length groups for each age), length groups (giving a distribution of age groups for each length) or into age-length groups. The likelihood score that is calculated gives some measure as to how well the data from the model fit to the data from the sample catches.

To specify a CatchDistribution likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                catchdistribution
datafile            <name for the datafile>
function            <function name>
<multivariate parameters>
aggregationlevel    <0 or 1> ; 1 to aggregate data over the whole year
overconsumption     <0 or 1> ; 1 to take overconsumption into account
epsilon             <epsilon>
areaaggfile         <area aggregation file specifying areas>
ageaggfile          <age aggregation file specifying ages>
lenaggfile          <length aggregation file specifying lengths>
fleetnames          <vector of the names of the fleets>
stocknames          <vector of the names of the stocks>
```

The optional flag <aggregationlevel> is used to specify whether the distribution data should be aggregated over the whole year (by setting aggregation level to 1) or not aggregated, and calculated for each timestep (by setting aggregation level to 0). If this line is not specified, then an aggregation level of 0 is assumed, and the distribution data is not aggregated over the whole year. Note that not all of the functions used to compare the data can aggregate the data over the whole year.

The optional flag <overconsumption> is used to specify whether any over consumption of the stock is to be taken into account when calculating the model distribution. If this is set to 1, then the model catch data will be adjusted to ensure that the fleets don't catch more stock than is available, by applying a bound to the catch of the fleets. If this line is not specified, then an overconsumption of 0 is assumed and any understocking that is present in the model is ignored, which can lead to an unrealistic result if the understocking likelihood component is not specified.

The optional <epsilon> value is used whenever the calculated probability is very unlikely, although the exact format of this depends on the function that is to be used when calculating the likelihood score. This means that the likelihood component is not dominated by one or two stray values, since these will be reset back to less unlikely values. The default value for <epsilon> is 10, which is used whenever it is not defined in the input file.

The <fleetnames> vector contains a list of all the fleets to be aggregated into a single pseudo fleet for the purposes of the data comparison. Similarly, the <stocknames> vector contains a list of all the stocks to be aggregated into a single pseudo stock.

The <function name> defines what likelihood function is to be used to compare the modelled age-length catch distribution to the input age-length catch distribution. Currently, there are 8 likelihood functions defined, and the valid function names are:

sumofsquares - use a sum of squares function
stratified - use a stratified sum of squares function
multinomial - use a multinomial function
pearson - use a Pearson function
gamma - use a gamma function
log - use a log function
mvn - use a multivariate normal function
mvlogistic - use a multivariate logistic function

The <multivariate parameters> are only required for the multivariate functions, and Gadget will generate an error if they are specified when they are not required. These parameters are described in the following sections.

Finally, the file specified by <datafile> contains a list of the age-length catch distribution that Gadget is to use to fit the likelihood function to, aggregated according to the aggregation files specified, for the numbers calculated in the model. The format of this file is given below:

```
<year>  <step>  <area>  <age>  <length>  <number>
```

where <number> is the number of samples for the timestep/area/age/length combination.

8.3.1 Sum of Squares Function

The sum of squares function calculates the likelihood component from equation 8.3 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \sum_{lengths} \left(P_{tral} - \pi_{tral} \right)^2 \tag{8.3}$$

where:
$< P >$ is the proportion of the data sample for that time/area/age/length combination
$< \pi >$ is the proportion of the model sample for that time/area/age/length combination

8.3.2 Stratified Sum of Squares Function

The stratified function is a variant of the sum of squares function that calculates an age distribution for each length class, and then calculates the likelihood component from equation 8.3 above. The difference between this function and the sum of squares function above is in the way the proportions of the samples are calculated - for this function the proportion is calculated for each length group in turn, whereas for the sum of squares function the proportion is taken over all the length groups. If there is only one length group then these two functions are identical.

8.3.3 Multinomial Function

The multinomial function calculates the likelihood component from equation 8.4 below:

$$\ell = 2 \sum_{time} \sum_{areas} \sum_{ages} \left(\log N_{tra}! - \sum_{lengths} \log N_{tral}! + \sum_{lengths} \left(N_{tral} \log \frac{\nu_{tral}}{\sum \nu_{tral}} \right) \right) \tag{8.4}$$

where:
$< N >$ is the data sample size for that time/area/age/length combination
$< \nu >$ is the model sample size for that time/area/age/length combination

8.3.4 Pearson Function

The Pearson function calculates the likelihood component from equation 8.5 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \sum_{lengths} \left(\frac{(N_{tral} - \nu_{tral})^2}{\nu_{tral} + \epsilon} \right) \tag{8.5}$$

where:
$< N >$ is the data sample size for that time/area/age/length combination
$< \nu >$ is the model sample size for that time/area/age/length combination

8.3.5 Gamma Function

The gamma function calculates the likelihood component from equation 8.6 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \sum_{lengths} \left(\frac{N_{tral}}{(\nu_{tral} + \epsilon)} + \log(\nu_{tral} + \epsilon) \right) \tag{8.6}$$

where:
$< N >$ is the data sample size for that time/area/age/length combination
$< \nu >$ is the model sample size for that time/area/age/length combination

8.3.6 Log Function

The log function calculates the likelihood component from equation 8.7 below:

$$\ell = \sum_{time} \sum_{areas} \left(\log \left(\frac{\sum_{ages} \sum_{lengths} \nu_{tral}}{\sum_{ages} \sum_{lengths} N_{tral}} \right) \right)^2 \tag{8.7}$$

where:
$<$ N $>$ is the data sample size for that time/area/age/length combination
$< \nu >$ is the model sample size for that time/area/age/length combination

8.3.7 Multivariate Normal Function

The multivariate normal function calculates the likelihood component from equation 8.8 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \Big(log|\Sigma| + (P_{tra} - \pi_{tra})^T \Sigma^{-1} (P_{tra} - \pi_{tra}) \Big) \tag{8.8}$$

where:
$< \Sigma >$ is the variance-covariance matrix for the multivariate normal distribution
$<$ P $>$ is the proportion of the data sample for that time/area/age combination
$< \pi >$ is the proportion of the model sample for that time/area/age combination

For the formulation of the variance-covariance matrix, $< \Sigma >$ is calculated from equations 8.9 and 8.10 below:

$$\Sigma = (\sigma_{ij})_{ij} \tag{8.9}$$

$$\sigma_{ij} = \begin{cases} \sum_{l=1}^{lag} c_l \sigma_{i-l,j} + \delta_j^i \sigma^2 & \text{if } i \geq j \\ \sum_{l=1}^{lag} c_l \sigma_{j,i-l} & \text{otherwise} \end{cases} \tag{8.10}$$

In equation 8.10 it is assumed that the number in each length group is autocorrelated with lag $<$ lag $>$. Note that setting the lag to be zero simplifies the multivariate normal distribution to a univariate one.

To specify this likelihood function, it is necessary to specify the parameters $< \sigma >$ and $<$ lag $>$ and a list of $<$ lag $>$ correlation parameters. This is done in the likelihood file, as shown below:

```
...
function            mvn
lag                 <lag>
sigma               <sigma>
param               <correlation parameter> ; note that a total of
param               <correlation parameter> ; <lag> correlation
...                                         ; parameters are required
aggregationlevel    <0 or 1> ; 1 to aggregate data over the whole year
...
```

8.3.8 Multivariate Logistic Function

The multivariate logistic function calculates the likelihood component from equations 8.11 and 8.12 below:

$$\ell = \frac{1}{2\sigma^2} \sum_{time} \Big((L-1) log(\sigma) + \sum_{areas} \sum_{ages} \sum_{lengths} \tau_{tral}^2 \Big) \tag{8.11}$$

$$\tau_{tral} = log(P_{tral}) - log(\pi_{tral}) - \frac{1}{L} \sum_{lengths} \Big(log(P_{tral}) - log(\pi_{tral}) \Big) \tag{8.12}$$

where:
$< L >$ is the number of length groups
$< P >$ is the proportion of the data sample for that time/area/age/length combination
$< \pi >$ is the proportion of the model sample for that time/area/age/length combination

To specify this likelihood function it is necessary to specify the parameter $< \sigma >$. This is done in the likelihood file as shown below:

```
...
function            mvlogistic
sigma               <sigma>
aggregationlevel    <0 or 1> ; 1 to aggregate data over the whole year
...
```

8.4 CatchStatistics

The CatchStatistics likelihood component is used to compare statistical data sampled from the model with statistical data sampled from landings or surveys. This is typically used to compare biological data, such as the mean length at age or mean weight at age. The likelihood score that is calculated gives some measure as to how well the data from the model fits to the data from the landings.

To specify a CatchStatistics likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                catchstatistics
datafile            <name for the datafile>
function            <function name>
overconsumption     <0 or 1> ; 1 to take overconsumption into account
areaaggfile         <area aggregation file specifying areas>
lenaggfile  <length aggregation file specifying lengthes; is optional and used only in weight at
ageaggfile          <age aggregation file specifying ages>
fleetnames          <vector of the names of the fleets>
stocknames          <vector of the names of the stocks>
```

The optional flag <overconsumption> is used to specify whether any over consumption of the stock is to be taken into account when calculating the model statistical data. If this is set to 1, then the model catch data will be adjusted to ensure that the fleets don't catch more stock than is available, by applying a bound to the catch of the fleets. If this line is not specified, then an overconsumption of 0 is assumed and any understocking that is present in the model is ignored, which can lead to an unrealistic result if the understocking likelihood component is not specified.

The <fleetnames> vector contains a list of all the fleets to be aggregated into a single pseudo fleet for the purposes of the data comparison. Similarly, the <stocknames> vector contains a list of all the stocks to be aggregated into a single pseudo stock.

The <function name> defines what likelihood function is to be used to compare the modelled statistical data to the input statistical data. Currently, there are 5 likelihood functions defined, and the format of the statistical data given in the file specified by <datafile> depends on the likelihood function used. The valid functions are:

lengthcalcstddev - use a weighted sum of squares of mean length
lengthgivenstddev - use a weighted sum of squares of mean length with given standard deviation

weightgivenstddev - use a weighted sum of squares of mean weight with given standard deviation
weightnostddev - use a unweighted sum of squares of mean weight
lengthnostddev - use a unweighted sum of squares of mean length
weightgivenstddevlen - use a weighted sum of squares of mean weight at length with given standard deviation
weightnostddevlen - use a unweighted sum of squares of mean weight at length

8.4.1 Weighted Sum of Squares of Mean Length

This likelihood function calculates the likelihood score based on a weighted sum of squares of the mean length, with the weighting given by calculating the variance of length of the modelled population, as shown in equation 8.13 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \left(\frac{(x_{tra} - \mu_{tra})^2}{\sigma_{tra}^2} N_{tra} \right) \tag{8.13}$$

where:
$< \mathrm{x} >$ is the sample mean length from the data
$< \mu >$ is the mean length calculated from the model
$< \sigma >$ is the standard deviation of the length, calculated from the model
$< \mathrm{N} >$ is the sample size

For this CatchStatistics function, the format of the statistical data required in the file specified by <datafile> is given below:

```
<year>  <step>  <area>  <age>  <number>  <mean>
```

where <number> is the number of samples for the timestep/area/age combination, and <mean> is the mean length of these samples.

8.4.2 Weighted Sum of Squares of Mean Length With Given Standard Deviation

This likelihood function calculates the likelihood score based on a weighted sum of squares of the mean length, with the weighting given the variance of length of the input population, as shown in equation 8.14 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \left(\frac{(x_{tra} - \mu_{tra})^2}{s_{tra}^2} N_{tra} \right) \tag{8.14}$$

where:
$< \mathrm{x} >$ is the sample mean length from the data
$< \mu >$ is the mean length calculated from the model
$< \mathrm{s} >$ is the standard deviation of the length from the data
$< \mathrm{N} >$ is the sample size

For this CatchStatistics function, the format of the statistical data required in the file specified by <datafile> is given below:

```
<year>  <step>  <area>  <age>  <number>  <mean>  <stddev>
```

where <number> is the number of samples for the timestep/area/age combination, <mean> is the mean length of these samples and <stddev> is the standard deviation of the length of these samples.

8.4.3 Weighted Sum of Squares of Mean Weight With Given Standard Deviation

This likelihood function calculates the likelihood score based on a weighted sum of squares of the mean weight, with the weighting given the variance of weight of the input population, as shown in equation 8.15 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \left(\frac{(x_{tra} - \mu_{tra})^2}{s_{tra}^2} N_{tra} \right) \tag{8.15}$$

where:
$< \mathrm{x} >$ is the sample mean weight from the data
$< \mu >$ is the mean weight calculated from the model
$< \mathrm{s} >$ is the standard deviation of the weight from the data
$< \mathrm{N} >$ is the sample size

For this CatchStatistics function, the format of the statistical data required in the file specified by <datafile> is given below:

```
<year>  <step>  <area>  <age>  <number>  <mean>  <stddev>
```

where <number> is the number of samples for the timestep/area/age combination, <mean> is the mean weight of these samples and <stddev> is the standard deviation of the weight of these samples.

8.4.4 Unweighted Sum of Squares of Mean Weight

This likelihood function calculates the likelihood score based on a unweighted sum of squares of the mean weight, with the variance of the weight of the population assumed to be 1, as shown in equation 8.16 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \left((x_{tra} - \mu_{tra})^2 N_{tra} \right) \tag{8.16}$$

where:
$< \mathrm{x} >$ is the sample mean weight from the data
$< \mu >$ is the mean weight calculated from the model
$< \mathrm{N} >$ is the sample size

For this CatchStatistics function, the format of the statistical data required in the file specified by <datafile> is given below:

```
<year>  <step>  <area>  <age>  <number>  <mean>
```

where <number> is the number of samples for the timestep/area/age combination, and <mean> is the mean weight of these samples.

8.4.5 Unweighted Sum of Squares of Mean Length

This likelihood function calculates the likelihood score based on a unweighted sum of squares of the mean length, with the variance of the length of the population assumed to be 1, as shown in equation 8.17 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \left((x_{tra} - \mu_{tra})^2 N_{tra} \right) \tag{8.17}$$

where:
$< x >$ is the sample mean length from the data
$< \mu >$ is the mean length calculated from the model
$< N >$ is the sample size

For this CatchStatistics function, the format of the statistical data required in the file specified by <datafile> is given below:

```
<year>  <step>  <area>  <age>  <number>  <mean>
```

where <number> is the number of samples for the timestep/area/age combination, and <mean> is the mean length of these samples.

8.5 StockDistribution

The StockDistribution likelihood component is used to compare distribution data sampled from the model with distribution data sampled from landings or surveys for different stocks within the Gadget model. This is typically used to compare Gadget stocks that are based on the same species, but have differing biological properties (eg. immature and mature fish). The distribution data can either be aggregated into age groups (giving a distribution of length groups for each age), length groups (giving a distribution of age groups for each length) or into age-length groups. The likelihood score that is calculated gives some measure as to how well the data from the model fits to the data from the landings.

To specify a StockDistribution likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                stockdistribution
datafile            <name for the datafile>
function            <function name>
aggregationlevel    <0 or 1> ; 1 to aggregate data over the whole year
overconsumption     <0 or 1> ; 1 to take overconsumption into account
epsilon             <epsilon>
areaaggfile         <area aggregation file specifying areas>
ageaggfile          <age aggregation file specifying ages>
lenaggfile          <length aggregation file specifying lengths>
fleetnames          <vector of the names of the fleets>
stocknames          <vector of the names of the stocks>
```

The optional flag <aggregationlevel> is used to specify whether the distribution data should be aggregated over the whole year (by setting aggregation level to 1) or not aggregated, and calculated for each timestep (by setting aggregation level to 0). If this line is not specified, then an aggregation level of 0 is assumed, and the distribution data is not aggregated over the whole year. Note that not all of the functions used to compare the data can aggregate the data over the whole year.

The optional flag <overconsumption> is used to specify whether any over consumption of the stock is to be taken into account when calculating the model distribution. If this is set to 1, then the model catch data will be adjusted to ensure that the fleets don't catch more stock than is available, by applying a bound to the catch of the fleets. If this line is not specified, then an overconsumption of 0 is assumed and any understocking that is present in the model is ignored, which can lead to an unrealistic result if the understocking likelihood component is not specified.

The optional <epsilon> value is used whenever the calculated probability is very unlikely, although the exact format of this depends on the function that is to be used when calculating the likelihood score. This means that the likelihood component is not dominated by one or two stray values, since these will be reset back to less unlikely values. The default value for <epsilon> is 10, which is used whenever it is not defined in the input file.

The <fleetnames> vector contains a list of all the fleets to be aggregated into a single pseudo fleet for the purposes of the data comparison. However, the <stocknames> vector contains a list of all the stocks to be compared for the data comparison. These stocks are not aggregated into a single pseudo stock.

The <function name> defines what likelihood function is to be used to compare the modelled age-length stock distribution to the input age-length stock distribution. Currently, there are two likelihood functions defined, and the valid functions are:

sumofsquares - use a sum of squares function
multinomial - use a multinomial function

Finally, the datafile is a list of the age-length catch distribution for each stock, that Gadget is to use to fit the likelihood function to, aggregated according to the aggregation files specified, for the numbers calculated in the model. The format of this file is given below:

```
<year>  <step>  <area>  <stock>  <age>  <length>  <number>
```

where <number> is the number of samples for the timestep/area/stock/age/length combination.

8.5.1 Sum of Squares Function

The sum of squares function calculates the likelihood component from equation 8.18 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \sum_{lengths} \sum_{stocks} (P_{trals} - \pi_{trals})^2 \tag{8.18}$$

where:
$< \mathrm{P} >$ is the proportion of the data sample for that time/area/age/length/stock combination
$< \pi >$ is the proportion of the model sample for that time/area/age/length/stock combination

8.5.2 Multinomial Function

The multinomial function calculates the likelihood component from equation 8.19 below:

$$\ell = 2 \sum_{time} \sum_{areas} \sum_{ages} \sum_{lengths} \left(\log N_{tral}! - \sum_{stocks} \log N_{trals}! + \sum_{stocks} \left(N_{trals} \log \frac{\nu_{tral}}{\sum \nu_{trals}} \right) \right) \tag{8.19}$$

where:
$< \mathrm{N} >$ is the data sample size for that time/area/age/length/stock combination
$< \nu >$ is the model sample size for that time/area/age/length/stock combination

8.6 SurveyIndices

The SurveyIndices likelihood component is used to compare the development of a stock in the Gadget model to indices calculated from a standardized survey for that stock. These indices can be aggregated into length groups or age groups. The likelihood component that is used is the sum of squares of a linear regression fitted to the difference between the modelled data and the specified index, given by equation 8.20 below:

$$\ell = \sum_{time} \Big(I_t - (\alpha + \beta N_t) \Big)^2 \tag{8.20}$$

where:
< I > is the observed survey index
< N > is the corresponding index calculated in the Gadget model

The exact format of this linear regression equation will vary, depending on survey index data available. It is possible to take the log of the indices and the modelled data before fitting the linear regression line. The slope and intercept of the linear regression line are controlled by the parameters alpha and beta, and it is possible to fix these to specified numbers, or let Gadget calculate these to get the best fit to the modelled data.

To specify a SurveyIndices likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                surveyindices
datafile            <name for the datafile>
sitype              <survey index type>
biomass             <0 or 1> ; 1 to base index data on biomass
<survey index data>
```

The optional flag <biomass> is used to specify whether the index data should be based on the biomass of the stock or on the population numbers for the stock. If this is set to 1, then the index data calculated in the model will be based on the available biomass of the stock. If this line is not specified, then a biomass value of 0 is assumed and the index data calculated in the model will be based on the available population numbers for the stock.

The format of the survey index data, and the contents of the datafile, depend on the type of survey index that is to be used, which is specified by the value of <survey index type>. There are currently 5 valid options, which are:

lengths - defining a length group based survey index
ages - defining an age group based survey index
fleets - defining a length group based survey index, taking the fleet selectivity into account
acoustic - defining an acoustic based survey index
effort - defining an fishing effort based survey index

8.6.1 SurveyIndices by Length

To specify a length group based SurveyIndices likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                  <name for the likelihood component>
weight                <weight for the likelihood component>
type                  surveyindices
datafile              <name for the datafile>
sitype                lengths
biomass               <0 or 1> ; 1 to base index data on biomass
areaaggfile           <area aggregation file specifying areas>
lenaggfile            <length aggregation file specifying lengths>
stocknames            <vector of the names of the stocks>
fittype               <fit type>
<fit type parameters>
```

The datafile is a list of the indices that Gadget is to use to fit the linear regression to, aggregated according to the length aggregation file specified, for the population numbers calculated in the model. The format of this file is given below:

```
<year>  <step>  <area>  <length>  <number>
```

where <number> is the survey index for that timestep/area/length combination.

The <fit type> defines the type of linear regression equation to be used to calculate the likelihood score for this likelihood component. These options specify whether or not the log of the numbers is to be used, and whether the parameters alpha and beta are to be estimated by Gadget, or fixed. If these parameters are to be fixed, then they are specified here. In total, there are 8 valid entries for <fit type>, and the associated parameters, and these are:

linearfit
loglinearfit
fixedslopelinearfit
fixedslopeloglinearfit
fixedinterceptlinearfit
fixedinterceptloglinearfit
fixedlinearfit
fixedloglinearfit

linear regression, estimating both slope and intercept

This fit type will fit a linear regression line, with the alpha and beta parameter values estimated from the data within the Gadget model. The file format for this fit type is given below:

```
fittype               linearfit
```

log linear regression, estimating both slope and intercept

This fit type will fit a log linear regression line, with the alpha and beta parameter values estimated from the data within the Gadget model. The file format for this fit type is given below:

```
fittype               loglinearfit
```

linear regression, fixing slope and estimating intercept

This fit type will fit a linear regression line, with the alpha parameter value estimated from the data within the Gadget model, and the beta parameter value specified in the input file. The file format for this fit type is given below:

```
fittype               fixedslopelinearfit
slope                 <beta>
```

log linear regression, fixing slope and estimating intercept

This fit type will fit a log linear regression line, with the alpha parameter value estimated from the data within the Gadget model, and the beta parameter value specified in the input file. The file format for this fit type is given below:

```
fittype             fixedslopeloglinearfit
slope               <beta>
```

linear regression, fixing intercept and estimating slope

This fit type will fit a linear regression line, with the beta parameter value estimated from the data within the Gadget model, and the alpha parameter value specified in the input file. The file format for this fit type is given below:

```
fittype             fixedinterceptlinearfit
intercept           <alpha>
```

log linear regression, fixing intercept and estimating slope

This fit type will fit a log linear regression line, with the beta parameter value estimated from the data within the Gadget model, and the alpha parameter value specified in the input file. The file format for this fit type is given below:

```
fittype             fixedinterceptloglinearfit
intercept           <alpha>
```

linear regression, fixing both slope and intercept

This fit type will fit a linear regression line, with the alpha and beta parameter values specified in the input file. The file format for this fit type is given below:

```
fittype             fixedlinearfit
slope               <beta>
intercept           <alpha>
```

log linear regression, fixing both slope and intercept

This fit type will fit a log linear regression line, with the alpha and beta parameter values specified in the input file. The file format for this fit type is given below:

```
fittype             fixedloglinearfit
slope               <beta>
intercept           <alpha>
```

8.6.2 SurveyIndices by Age

To specify an age group based SurveyIndices likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                surveyindices
datafile            <name for the datafile>
sitype              ages
biomass             <0 or 1> ; 1 to base index data on biomass
areaaggfile         <area aggregation file specifying areas>
```

```
ageaggfile           <age aggregation file specifying ages>
stocknames           <vector of the names of the stocks>
fittype              <fit type>
<fit type parameters>
```

The datafile is a list of the indices that Gadget is to use to fit the linear regression to, aggregated according to the age aggregation file specified, for the population numbers calculated in the model. The format of this file is given below:

```
<year>  <step>  <area>  <age>  <number>
```

where <number> is the survey index for that timestep/area/age combination.

The <fit type> defines the type of linear regression equation to be used to calculate the likelihood score for this likelihood component. The valid fit type options are the same as for the length based survey indices, given in section 8.6.1 above.

8.6.3 SurveyIndices by Fleet

To specify a length group based SurveyIndices likelihood component taking the fleet selectivity into account, the format required in the main likelihood file is as follows:

```
[component]
name                 <name for the likelihood component>
weight               <weight for the likelihood component>
type                 surveyindices
datafile             <name for the datafile>
sitype               fleets
biomass              <0 or 1> ; 1 to base index data on biomass
areaaggfile          <area aggregation file specifying areas>
lenaggfile           <length aggregation file specifying lengths>
fleetnames           <vector of the names of the fleets>
stocknames           <vector of the names of the stocks>
fittype              <fit type>
<fit type parameters>
```

The datafile is a list of the indices that Gadget is to use to fit the linear regression to, aggregated according to the length aggregation file specified, for the population numbers calculated in the model. The format of this file is given below:

```
<year>  <step>  <area>  <length>  <number>
```

where <number> is the survey index for that timestep/area/length combination.

The <fit type> defines the type of linear regression equation to be used to calculate the likelihood score for this likelihood component. The valid fit type options are the same as for the length based survey indices, given in section 8.6.1 above.

8.6.4 SurveyIndices by Acoustic

To specify an acoustic based SurveyIndices likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                 <name for the likelihood component>
weight               <weight for the likelihood component>
type                 surveyindices
```

```
datafile             <name for the datafile>
sitype               acoustic
biomass              <0 or 1> ; 1 to base index data on biomass
areaaggfile          <area aggregation file specifying areas>
surveynames          <vector of the names of the acoustic surveys>
stocknames           <vector of the names of the stocks>
fittype              <fit type>
<fit type parameters>
```

The datafile is a list of the acoustic indices that Gadget is to use to fit the linear regression to, for the population calculated in the model. The format of this file is given below:

```
<year>  <step>  <area>  <survey>  <acoustic>
```

where <acoustic> is the acoustic index for that timestep/area/survey combination.

The <fit type> defines the type of linear regression equation to be used to calculate the likelihood score for this likelihood component. The valid fit type options are the same as for the length based survey indices, given in section 8.6.1 above.

8.6.5 SurveyIndices by Effort

To specify an effort based SurveyIndices likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                 <name for the likelihood component>
weight               <weight for the likelihood component>
type                 surveyindices
datafile             <name for the datafile>
sitype               effort
biomass              <0 or 1> ; 1 to base index data on biomass
areaaggfile          <area aggregation file specifying areas>
fleetnames           <vector of the names of the fleets>
stocknames           <vector of the names of the stocks>
fittype              <fit type>
<fit type parameters>
```

The datafile is a list of the fleet effort indices that Gadget is to use to fit the linear regression to, for the fishing effort calculated in the model. The format of this file is given below:

```
<year>  <step>  <area>  <fleet>  <effort>
```

where <effort> is the effort index for that timestep/area/fleet combination.

The <fit type> defines the type of linear regression equation to be used to calculate the likelihood score for this likelihood component. The valid fit type options are the same as for the length based survey indices, given in section 8.6.1 above.

8.7 SurveyDistribution

The SurveyDistribution likelihood component is used to compare the development of a stock in the Gadget model to age-length indices calculated from a survey for that stock. The likelihood score that is calculated gives some measure as to how well the data from the model fits to the data from the calculated survey index distribution.

To specify a SurveyDistribution likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                  <name for the likelihood component>
weight                <weight for the likelihood component>
type                  surveydistribution
datafile              <name for the datafile>
areaaggfile           <area aggregation file specifying areas>
lenaggfile            <length aggregation file specifying lengths>
ageaggfile            <age aggregation file specifying ages>
stocknames            <vector of the names of the stocks>
fittype               <fit type>
parameters            <fit type parameters>
<suitability parameters>
epsilon               <epsilon>
likelihoodtype        <likelihood type>
```

The <stocknames> vector contains a list of all the stocks to be aggregated into a single pseudo stock for the purposes of the data comparison. The <suitability parameters> define the suitability of the survey fleet that was used to collect the survey index data. This is the same format as the suitability functions for the stock, as discussed in section 4.9 above. Note that only one set of suitability values is defined, which will be applied to all the stocks for this likelihood component.

The <fit type> defines what function is to be used to calculate the survey index distribution from the modelled population. Currently, there are two functions defined, and the valid function names are:

linearfit - use a linear function
powerfit - use a power function

The <fit type parameters> is a vector of 2 parameters that are used to calculate the survey index values from the modelled population. The <epsilon> value is used whenever the calculated probability is very unlikely, although the exact format of this depends on the likelihood type that is to be used when calculating the likelihood score.

The <likelihood type> defines what function is to be used to compare the modelled survey index distribution to the input survey index distribution. Currently, there are 4 functions defined, and the valid function names are:

multinomial - use a multinomial function
pearson - use a Pearson function
gamma - use a gamma function
log - use a log function

Finally, the file specified by <datafile> contains a list of the age-length survey indices that Gadget is to use to fit the likelihood function to, aggregated according to the aggregation files specified, for the numbers calculated in the model. The format of this file is given below:

```
<year>  <step>  <area>  <age>  <length>  <number>
```

where <number> is the survey index for the timestep/area/age/length combination.

8.7.1 Linear Fit

The linear fit function calculates the survey index for the modelled population from equation 8.21 below:

$$\widehat{I}_{tral} = q_0 S_l \big(N_{tral} + q_1 \big) \tag{8.21}$$

where:
< S > is the calculated suitability value for that length group
< N > is the model population for that time/area/age/length combination

8.7.2 Power Fit

The power fit function calculates the survey index for the modelled population from equation 8.21 below:

$$\widehat{I}_{tral} = q_0 S_l N_{tral}^{q_1} \tag{8.22}$$

where:
< S > is the calculated suitability value for that length group
< N > is the model population for that time/area/age/length combination

8.7.3 Multinomial Function

The multinomial function calculates the likelihood component from equation 8.23 below:

$$\ell = \sum_{time} \sum_{areas} \left(\log \left(\sum_{ages} \sum_{lengths} \widehat{I}_{tral} \right) - \frac{\sum_{ages} \sum_{lengths} \left(\widehat{I}_{tral} \log(I_{tral} + \epsilon) \right)}{\sum_{ages} \sum_{lengths} I_{tral}} \right) \tag{8.23}$$

where:
< I > is the data survey index for that time/area/age/length combination
$< \widehat{I} >$ is the model survey index for that time/area/age/length combination

8.7.4 Pearson Function

The Pearson function calculates the likelihood component from equation 8.24 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \sum_{lengths} \left(\frac{(I_{tral} - \widehat{I}_{tral})^2}{\widehat{I}_{tral} + \epsilon} \right) \tag{8.24}$$

where:
< I > is the data survey index for that time/area/age/length combination
$< \widehat{I} >$ is the model survey index for that time/area/age/length combination

8.7.5 Gamma Function

The gamma function calculates the likelihood component from equation 8.25 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{ages} \sum_{lengths} \left(\frac{I_{tral}}{(\widehat{I}_{tral} + \epsilon)} + \log(\widehat{I}_{tral} + \epsilon) \right) \tag{8.25}$$

where:
< I > is the data survey index for that time/area/age/length combination
$< \widehat{I} >$ is the model survey index for that time/area/age/length combination

8.7.6 Log Function

The log function calculates the likelihood component from equation 8.26 below:

$$\ell = \sum_{time} \sum_{areas} \left(\log \left(\frac{\sum_{ages} \sum_{lengths} \widehat{I}_{tral}}{\sum_{ages} \sum_{lengths} I_{tral}} \right) \right)^2 \tag{8.26}$$

where:
$< \mathrm{I} >$ is the data survey index for that time/area/age/length combination
$< \widehat{I} >$ is the model survey index for that time/area/age/length combination

8.8 StomachContent

The StomachContent likelihood component is used to compare consumption data sampled from the model with stomach content data obtained by analysing the stomach contents of various predators. This data can be used to give an indication of the diet composition of the stock. The likelihood score that is calculated gives some measure as to how well the consumption data from the model fits to the data from the stomach contents. Care is needed when making this comparison, since the data will give information on the stomach content at the time of capture of the predator, where as the Gadget simulation can only give information about the modelled consumption of the prey by the predator.

To specify a StomachContent likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                stomachcontent
function            <function name>
datafile            <name for the datafile>
epsilon             <epsilon>
areaaggfile         <area aggregation file specifying areas>
predatornames       <vector of the names of the predators>
predatorlengths
lenaggfile          <length aggregation file specifying predator lengths>
preyaggfile         <prey aggregation file specifying preys>
```

The optional <epsilon> value is used whenever the calculated probability is very unlikely, although the exact format of this depends on the function that is to be used when calculating the likelihood score. This means that the likelihood component is not dominated by one or two stray values, since these will be reset back to less unlikely values. The default value for <epsilon> is 10, which is used whenever it is not defined in the input file.

The <predatornames> vector contains a list of all the predators to be aggregated into a single pseudo predator for the purposes of the data comparison.

The <function name> defines what likelihood function is to be used to compare the modelled consumption data to the input stomach content data. Currently, there is only one likelihood function defined, so the valid function name is:

scsimple - use a simple ratio function

Finally, the file specified by <datafile> contains a list of the stomach content data that Gadget is to use to fit the likelihood function to, aggregated according to the aggregation files specified, for the consumption calculated in the model. The format of this file is given below:

```
<year>  <step>  <area>  <predator>  <prey>  <ratio>
```

where <ratio> is the ratio of prey <prey> in the stomachs of predator <predator> for the timestep/area combination, where <prey> is defined in the prey aggregation file, and <predator> is defined in the predator length aggregation file.

8.8.1 SCSimple Function

The scsimple function calculates the likelihood component by comparing the ratio of the consumption of different preys by a predator in the model to the ratio of the preys found in the stomach contents data specified in the input file, as shown in equation 8.27 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{predators} \sum_{preys} \left(P_{trpp} - \pi_{trpp} \right)^2 \tag{8.27}$$

where:
$< P >$ is the ratio of the stomach content data for that time/area/predator/prey combination
$< \pi >$ is the ratio of the modelled consumption for that time/area/predator/prey combination

8.9 Recaptures

The Recaptures likelihood component is used to compare recaptures data from tagging experiments within the model with recaptures data obtained from tagging experiments, aggregated according to length at recapture. The likelihood score that is calculated gives some measure as to how well the data from the model fits the recaptures data.

To specify a Recaptures likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                recaptures
datafile            <name for the datafile>
function            <function name>
areaaggfile         <area aggregation file specifying areas>
lenaggfile          <length aggregation file specifying recapture lengths>
fleetnames          <vector of the names of the fleets>
```

The <fleetnames> vector contains a list of all the fleets to be aggregated into a single pseudo fleet for the purposes of the data comparison.

The <function name> defines what likelihood function is to be used to compare the modelled recaptures data to the input recaptures data. Currently, there is only one likelihood function defined, so the only valid function name is:

poisson - use a Poisson function

Finally, the datafile is a list of the recaptures that Gadget is to use to fit the likelihood function to, aggregated according to the aggregation files specified, for the numbers calculated in the model. The format of this file is given below:

```
<tagid>  <year>  <step>  <area>  <length>  <number>
```

where <number> is the number of recaptures for the tag/timestep/area/length combination.

8.9.1 Poisson Function

The Poisson function calculates the likelihood component from equation 8.28 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{lengths} \left(N_{trl} + \log \nu_{trl}! - N_{trl} \log \nu_{trl} \right) \tag{8.28}$$

where:
$< \mathrm{N} >$ is the number of observed recaptures for that time/area/length combination
$< \nu >$ is the number of modelled recaptures for that time/area/length combination

8.10 RecStatistics

The RecStatistics likelihood component is used to compare statistical data sampled from tagged subpopulations within the model with statistical data obtained from the fish returned from tagging experiments. This is used to compare biological data, such as the mean length at age, and is similar to the CatchStatistics likelihood component (see section 8.4). The likelihood score that is calculated gives some measure as to how well the data from the model fits to the data from the recaptures.

To specify a RecStatistics likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                recstatistics
datafile            <name for the datafile>
function            <function name>
areaaggfile         <area aggregation file specifying areas>
fleetnames          <vector of the names of the fleets>
```

The <fleetnames> vector contains a list of all the fleets to be aggregated into a single pseudo fleet for the purposes of the data comparison.

The <function name> defines what likelihood function is to be used to compare the modelled statistical data to the input statistical data. Currently, there are three likelihood functions defined, and the format of the statistical data given in the file specified by <datafile> depends on the likelihood function used. The valid functions are:

lengthcalcstddev - use a weighted sum of squares of mean length
lengthgivenstddev - use a weighted sum of squares of mean length with given standard deviation
lengthnostddev - use a unweighted sum of squares of mean length

8.10.1 Weighted Sum of Squares of Mean Length

This likelihood function calculates the likelihood score based on a weighted sum of squares of the mean length, with the weighting given by calculating the variance of length of the modelled population, as shown in equation 8.29 below:

$$\ell = \sum_{tags} \sum_{time} \sum_{areas} \left(\frac{(x-\mu)^2}{\sigma^2} N \right) \tag{8.29}$$

where:
$< \mathrm{x} >$ is the sample mean length from the data

$< \mu >$ is the mean length calculated from the model
$< \sigma >$ is the standard deviation of the length, calculated from the model
$<$ N $>$ is the sample size

For this RecStatistics function, the format of the statistical data required in the file specified by <datafile> is given below:

```
<tagid>  <year>  <step>  <area>  <number>  <mean>
```

where <number> is the number of samples for the tag/timestep/area combination, and <mean> is the mean length of these samples.

8.10.2 Weighted Sum of Squares of Mean Length With Given Standard Deviation

This likelihood function calculates the likelihood score based on a weighted sum of squares of the mean length, with the weighting given the variance of length of the input population, as shown in equation 8.30 below:

$$\ell = \sum_{tags} \sum_{time} \sum_{areas} \left(\frac{(x-\mu)^2}{s^2} N \right) \tag{8.30}$$

where:
$<$ x $>$ is the sample mean length from the data
$< \mu >$ is the mean length calculated from the model
$<$ s $>$ is the standard deviation of the length from the data
$<$ N $>$ is the sample size

For this RecStatistics function, the format of the statistical data required in the file specified by <datafile> is given below:

```
<tagid>  <year>  <step>  <area>  <number>  <mean>  <stddev>
```

where <number> is the number of samples for the tag/timestep/area combination, <mean> is the mean length of these samples and <stddev> is the standard deviation of the length of these samples.

8.10.3 Unweighted Sum of Squares of Mean Length

This likelihood function calculates the likelihood score based on a unweighted sum of squares of the mean length, with the variance of the length of the population assumed to be 1, as shown in equation 8.31 below:

$$\ell = \sum_{tags} \sum_{time} \sum_{areas} \left((x-\mu)^2 N \right) \tag{8.31}$$

where:
$<$ x $>$ is the sample mean length from the data
$< \mu >$ is the mean length calculated from the model
$<$ N $>$ is the sample size

For this RecStatistics function, the format of the statistical data required in the file specified by <datafile> is given below:

```
<tagid>  <year>  <step>  <area>  <number>  <mean>
```

where <number> is the number of samples for the tag/timestep/area combination, and <mean> is the mean length of these samples.

8.11 MigrationPenalty

The MigrationPenalty likelihood component is used to give a penalty whenever there is a negative migration value from the migration matrices (which is meaningless). The MigrationPenalty component is used (rather than the BoundLikelihood component) since the values in the migration matrices are calculated from more than one parameter, and it is not necessarily the individual parameters that are wrong, rather the combination of the parameters that give the migration matrix value that is wrong. The likelihood component that is used is based on the sum of squares of the migration values, given by the equation below:

$$\ell = \left(\sum_{ij} M_{ij}^{p_0} \right)^{p_1} \tag{8.32}$$

The use of 2 power coefficients gives increased flexibility for the likelihood component. In general, a higher value of p_1 applies a higher penalty to "many small negative values", where as a higher value of p_0 applies a higher penalty to "few large negative values". For a simple sum of squares of the migration matrix values, p_0 should be set to 2, and p_1 should be set to 1.

To specify a MigrationPenalty likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                migrationpenalty
stockname           <name for the stock to check>
powercoeffs         <p0>  <p1>
```

Note that it is not possible to aggregate more than one stock into a single pseudo stock for this likelihood component.

8.12 MigrationProportion

The MigrationProportion likelihood component is used to compare population proportion data sampled from the model with population proportion data sampled from landings or surveys. The populations proportion data gives the proportion of the population is that is present on each area on a given timestep. The likelihood score that is calculated gives some measure as to how well the migration data from the model fit to the data from the sample catches.

To specify a MigrationProportion likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                migrationproportion
datafile            <name for the datafile>
function            <function name>
biomass             <0 or 1> ; 1 to base migration proportion data on biomass
areaaggfile         <area aggregation file specifying areas>
stocknames          <vector of the names of the stocks>
```

The optional flag <biomass> is used to specify whether the migration proportion data should be based on the biomass of the stock or on the population numbers for the stock. If this is set to 0, then the migration proportion data calculated in the model will be based on the available population numbers for the stock. If this line is not specified, then a biomass value of 1 is assumed and the migration proportion data calculated in the model will be based on the available population biomass of the stock.

The <stocknames> vector contains a list of all the stocks to be aggregated into a single pseudo stock for the purposes of the data comparison.

The <function name> defines what likelihood function is to be used to compare the modelled migration proportion data to the input migration proportion data. Currently, there is only one likelihood function defined, so the valid function name is:

sumofsquares - use a simple sum of squares function

Finally, the file specified by <datafile> contains a list of the migration proportion data that Gadget is to use to fit the likelihood function to, aggregated according to the aggregation files specified, for the migration proportions calculated in the model. The format of this file is given below:

```
<year>  <step>  <area>  <ratio>
```

where <ratio> is the proportion of stock that is in area <area> for that timestep.

8.12.1 Sum of Squares Function

The sum of squares function calculates the likelihood component from equation 8.33 below:

$$\ell = \sum_{time} \sum_{areas} \left(P_{tr} - \pi_{tr} \right)^2 \tag{8.33}$$

where:
$< P >$ is the proportion of the data sample for that time/area combination
$< \pi >$ is the proportion of the model sample for that time/area combination

8.13 CatchInKilos

The CatchInKilos likelihood component is used to compare the overall catch from the modelled fleets with landings data. This can be done for any fleet that has landings data available, but will give more useful information when used with fleets of type "LinearFleet", since the "TotalFleet" fleet type will catch the amount specified in the input file (see section 7 for more information on the available fleet types).

To specify a CatchInKilos likelihood component, the format required in the main likelihood file is as follows:

```
[component]
name                <name for the likelihood component>
weight              <weight for the likelihood component>
type                catchinkilos
datafile            <name for the datafile>
function            <function name>
aggregationlevel    <0 or 1> ; 1 to aggregate data over the whole year
epsilon             <epsilon>
areaaggfile         <area aggregation file specifying areas>
fleetnames          <vector of the names of the fleets>
stocknames          <vector of the names of the stocks>
```

The optional flag <aggregationlevel> is used to specify whether the catch data should be aggregated over the whole year (by setting aggregation level to 1) or not aggregated, and calculated for each timestep (by setting aggregation level to 0). If this line is not specified, then an aggregation level of 0 is assumed, and the catch data is not aggregated over the whole year.

The <fleetnames> vector contains a list of all the fleets to be aggregated into a single pseudo fleet for the purposes of the data comparison. Similarly, the <stocknames> vector contains a list of all the stocks to be aggregated into a single pseudo stock.

The optional <epsilon> value is used in the likelihood function to avoid problems that would arise from taking the logarithm of zero. Epsilon is added to both the modelled and observed landings data, to ensure that these values are always positive, and thus should be set to a small number. The default value for <epsilon> is 10, which is used whenever it is not defined in the input file.

The <function name> defines what likelihood function is to be used to compare the modelled catch to the input catch. Currently, there is only one likelihood function defined, so the only valid function name is:

sumofsquares - use a log sum of squares function

Finally, the file specified by <datafile> contains the landings data that Gadget is to use to fit the likelihood function to for the catch calculated in the model. The format of this file is given below:

```
<year>  <step>  <area>  <fleet>  <biomass>
```

where <biomass> is the catch for the timestep/area/fleet combination. The <step> column is optional if the <aggregationlevel> flag has been set to 1, since the data will be aggregated over the whole year. In this case, it is possible to specify the landings data in the following format:

```
<year>  <area>  <fleet>  <biomass>
```

8.13.1 Sum of Squares Function

The sum of squares function calculates the likelihood component from equation 8.34 below:

$$\ell = \sum_{time} \sum_{areas} \sum_{fleets} (\log(N_{trf} + \epsilon) - \log(\nu_{trf} + \epsilon))^2 \tag{8.34}$$

where:
$< \mathrm{N} >$ is the catch biomass for that time/area/fleet combination
$< \nu >$ is the modelled catch biomass for that time/area/fleet combination

Chapter 9

Print Files

The print files are used to control the output from the Gadget model (and not the output from the optimisation process). To avoid writing the model output from each iteration of a optimising process (and thus generating very large files), any printfile settings are ignored if Gadget is started with the -l switch.

To define print files in the Gadget model, the "main" file must contain a list of the data files that contain the description of the printer classes required, and the format for this is shown below:

```
printfiles          <names of the print files>
```

The print files contain a list of various type of printer classes, separated by the keyword [component], that output different information from the model, and the name of the file that the information is to be written to. All the output is written as a plain ASCII text file that can be viewed in any text editor. The format of the print file is follows:

```
[component]
type                <printer type>
<printer data>
```

The printer data for each printer type is covered in the sub sections below. The <printer type> defines the type of output that will be generated from the Gadget model, and there are currently 11 valid printer types defined in Gadget. These are:

StockStdPrinter
StockFullPrinter
StockPrinter
PredatorPrinter
PredatorOverPrinter
PreyOverPrinter
StockPreyFullPrinter
StockPreyPrinter
PredatorPreyPrinter
LikelihoodPrinter
LikelihoodSummaryPrinter

9.1 StockStdPrinter

The printer type to output the standard details of a stock is called "StockStdPrinter". This printer type is defined by specifying the stock and timesteps of interest. The file format for this component is given below:

```
[component]
type                stockstdprinter
stockname           <name of the stock>
scale               <scaling factor>
printfile           <name for the output file to be created>
precision           <precision to be used in the output file>
printatstart        <0 or 1> ; 1 to print at start of timestep
yearsandsteps       <ActionAtTime to determine when to print>
```

The optional <scale> factor is used to scale the size of the stock, which can be used to display the stock in terms of thousands of fish, for example. The default value for this parameter is 1, which will ensure that no scaling will take place.

The optional <precision> value is used to specify the number of digits to be used when printing the information to the specified output file, overriding the default settings for this printer type.

The optional flag <printatstart> is used to specify whether the information about the stock should be printed at the start or the end of the timestep. If this is set to 1, then the stock information as it is at the start of the timestep is printed, before any other calculation has taken place (see appendix A for more information on the order of the calculations). The default value for <printatstart> is 0, which means that the information at the end of the timestep is printed, and is used whenever the flag is not specified in the input file.

The output that is generated from this printer type is a file containing the following information for the stock specified on the <stockname> line:

```
year-step-area-age-number-length-weight-stddev-consumed-biomass
```

where:
<number> is the stock population for that timestep/area/age combination
<length> is the mean length for that timestep/area/age combination
<weight> is the mean weight for that timestep/area/age combination
<stddev> is the standard deviation for the length for that timestep/area/age combination
<consumed> is the stock population that has been consumed by all the predators (including fleets) for that timestep/area/age combination
<biomass> is the stock biomass that has been consumed by all the predators (including fleets) for that timestep/area/age combination

9.2 StockFullPrinter

The printer type to output some more detailed information about a stock is called "StockFullPrinter". This printer type is defined by specifying the stock and timesteps of interest. The file format for this component is given below:

```
[component]
type                stockfullprinter
stockname           <name of the stock>
printfile           <name for the output file to be created>
precision           <precision to be used in the output file>
printatstart        <0 or 1> ; 1 to print at start of timestep
yearsandsteps       <ActionAtTime to determine when to print>
```

The optional <precision> value is used to specify the number of digits to be used when printing the information to the specified output file, overriding the default settings for this printer type.

The optional flag <printatstart> is used to specify whether the information about the stock should be printed at the start or the end of the timestep. If this is set to 1, then the stock information as it is at the start of the timestep is printed, before any other calculation has taken place (see appendix A for more information on the order of the calculations). The default value for <printatstart> is 0, which means that the information at the end of the timestep is printed, and is used whenever the flag is not specified in the input file.

The output that is generated from this printer type is a file containing the following information for the stock specified on the <stockname> line:

```
year-step-area-age-length-number-weight
```

where:
<number> is the population for that timestep/area/age/length combination
<weight> is the mean weight for that timestep/area/age/length combination

9.3 StockPrinter

The printer type to output information about (one or more) stocks, with the information aggregated into a convenient grouping, is called "StockPrinter". This printer type is defined by specifying the stocks, areas, age groups, length groups and timesteps of interest. The file format for this component is given below:

```
[component]
type                stockprinter
stocknames          <vector of the names of the stocks>
areaaggfile         <area aggregation file specifying areas>
ageaggfile          <age aggregation file specifying ages>
lenaggfile          <length aggregation file specifying lengths>
printfile           <name for the output file to be created>
precision           <precision to be used in the output file>
printatstart        <0 or 1> ; 1 to print at start of timestep
yearsandsteps       <ActionAtTime to determine when to print>
```

Note that this printer type can aggregate more than one stock into a combined pseudo stock for the output file.

The optional <precision> value is used to specify the number of digits to be used when printing the information to the specified output file, overriding the default settings for this printer type.

The optional flag <printatstart> is used to specify whether the information about the stock should be printed at the start or the end of the timestep. If this is set to 1, then the stock information as it is at the start of the timestep is printed, before any other calculation has taken place (see appendix A for more information on the order of the calculations). The default value for <printatstart> is 0, which means that the information at the end of the timestep is printed, and is used whenever the flag is not specified in the input file.

The output that is generated from this printer type is a file containing the following information for all the stocks specified on the <stocknames> line:

```
year-step-area-age-length-number-weight
```

where:
<area> is the label for the area from the area aggregation file
<age> is the label for the age group from the age aggregation file
<length> is the label for the length group from the length aggregation file
<number> is the population for that timestep/area/age/length combination
<weight> is the mean weight for that timestep/area/age/length combination

9.4 PredatorPrinter

The printer type to output information about predation, with the information aggregated into a convenient grouping, is called "PredatorPrinter". This printer type is defined by specifying the predators, preys, areas, length groups and timesteps of interest. The file format for this component is given below:

```
[component]
type                predatorprinter
predatornames       <vector of the names of the predators>
preynames           <vector of the names of the preys>
areaaggfile         <area aggregation file specifying areas>
predlenaggfile      <length aggregation file specifying predator lengths>
preylenaggfile      <length aggregation file specifying prey lengths>
biomass             <0 or 1> ; 1 to print biomass consumed
printfile           <name for the output file to be created>
precision           <precision to be used in the output file>
yearsandsteps       <ActionAtTime to determine when to print>
```

Note that this printer type can aggregate more than one predator into a combined pseudo predator, and more than one prey into a pseudo prey, for the output file.

The optional flag <biomass> is used to specify whether the information about the consumption of the preys by the predators is printed by biomass consumed or by number consumed. If this is set to 1 then information about the consumption is printed by biomass consumed. If this is set to 0 then this information is printed by number consumed. The default value for <biomass> is 1, which means that the consumption by biomass will be printed if this flag is not specified in the input file.

The optional <precision> value is used to specify the number of digits to be used when printing the information to the specified output file, overriding the default settings for this printer type.

The output that is generated from this printer type is a file containing the following predation information for all the predators specified on the <predatornames> line, consuming all the preys specified on the <preynames> line:

```
year-step-area-pred-prey-amount
```

where:
<area> is the label for the area from the area aggregation file
<pred> is the label for the predator length group from the length aggregation file
<prey> is the label for the prey length group from the length aggregation file
<amount> is the biomass (or number) consumed for that timestep/area/predator length/prey length combination, depending on the value of the <biomass> flag in the input file

9.5 PredatorOverPrinter

The printer type to output information about predator over consumption (where a predator has failed to eat the required amount of prey since the prey is not available), with the information aggregated into a convenient grouping, is called "PredatorOverPrinter". This printer type is defined by specifying the predators, areas, length groups and timesteps of interest. The file format for this component is given below:

```
[component]
type                predatoroverprinter
predatornames       <vector of the names of the predators>
areaaggfile         <area aggregation file specifying areas>
lenaggfile          <length aggregation file specifying lengths>
printfile           <name for the output file to be created>
precision           <precision to be used in the output file>
yearsandsteps       <ActionAtTime to determine when to print>
```

Note that this printer type can aggregate more than one predator into a combined pseudo predator for the output file.

The optional <precision> value is used to specify the number of digits to be used when printing the information to the specified output file, overriding the default settings for this printer type.

The output that is generated from this printer type is a file containing the following over-consumption information for all the predators specified on the <predatornames> line:

```
year-step-area-length-biomass
```

where:
<area> is the label for the area from the area aggregation file
<length> is the label for the length group from the length aggregation file
<biomass> is the biomass that the predator failed to consume for that timestep/area/length combination

9.6 PreyOverPrinter

The printer type to output information about prey over consumption (where there has been insufficient prey for a predator to consume), with the information aggregated into a convenient grouping, is called "PreyOverPrinter". This printer type is the inverse of the PredatorOverPrinter printer type, in that it gives the same information, but from the point of view of the preys, not the predators. This printer type is defined by specifying the preys, areas, length groups and timesteps of interest. The file format for this component is given below:

```
[component]
type                preyoverprinter
preynames           <vector of the names of the preys>
areaaggfile         <area aggregation file specifying areas>
lenaggfile          <length aggregation file specifying lengths>
printfile           <name for the output file to be created>
precision           <precision to be used in the output file>
yearsandsteps       <ActionAtTime to determine when to print>
```

Note that this printer type can aggregate more than one prey into a combined pseudo prey for the output file.

The optional <precision> value is used to specify the number of digits to be used when printing the information to the specified output file, overriding the default settings for this printer type.

The output that is generated from this printer type is a file containing the following over-consumption information for all the preys specified on the <preynames> line:

```
year-step-area-length-biomass
```

where:
<area> is the label for the area from the area aggregation file
<length> is the label for the length group from the length aggregation file
<biomass> is the biomass of the prey that was unavailable, for that timestep/area/length combination

9.7 StockPreyFullPrinter

The printer type to output detailed information about a prey is called "StockPreyFullPrinter". This printer type is defined by specifying the prey and timesteps of interest. The file format for this component is given below:

```
[component]
type                stockpreyfullprinter
preyname            <name of the prey>
printfile           <name for the output file to be created>
precision           <precision to be used in the output file>
yearsandsteps       <ActionAtTime to determine when to print>
```

The optional <precision> value is used to specify the number of digits to be used when printing the information to the specified output file, overriding the default settings for this printer type.

The output that is generated from this printer type is a file containing the following information for the prey specified on the <preyname> line:

```
year-step-area-age-length-number-biomass
```

where:
<number> is the total population consumed for that timestep/area/age/length combination
<biomass> is the total biomass consumed for that timestep/area/age/length combination

9.8 StockPreyPrinter

The printer type to output information about (one or more) preys, with the information aggregated into a convenient grouping, is called "StockPreyPrinter". This printer type is defined by specifying the preys, areas, age groups, length groups and timesteps of interest. The file format for this component is given below:

```
[component]
type                stockpreyprinter
preynames           <vector of the names of the preys>
printfile           <name for the output file to be created>
areaaggfile         <area aggregation file specifying areas>
ageaggfile          <age aggregation file specifying ages>
lenaggfile          <length aggregation file specifying lengths>
precision           <precision to be used in the output file>
yearsandsteps       <ActionAtTime to determine when to print>
```

Note that this printer type can aggregate more than one prey into a combined pseudo prey for the output file.

The optional <precision> value is used to specify the number of digits to be used when printing the information to the specified output file, overriding the default settings for this printer type.

The output that is generated from this printer type is a file containing the following information for all the preys specified on the <preynames> line:

```
year-step-area-age-length-number-biomass
```

where:
<area> is the label for the area from the area aggregation file
<age> is the label for the age group from the age aggregation file
<length> is the label for the length group from the length aggregation file
<number> is the total population consumed for that timestep/area/age/length combination
<biomass> is the total biomass consumed for that timestep/area/age/length combination

9.9 PredatorPreyPrinter

The printer type to output detailed information about predator-prey combinations, with the information aggregated into a convenient grouping, is called "PredatorPreyPrinter". This printer type is defined by specifying the predators, preys, areas, age groups, length groups and timesteps of interest. The file format for this component is given below:

```
[component]
type                predatorpreyprinter
predatornames       <vector of the names of the predators>
preynames           <vector of the names of the preys>
areaaggfile         <area aggregation file specifying areas>
ageaggfile          <age aggregation file specifying prey ages>
lenaggfile          <length aggregation file specifying prey lengths>
printfile           <name for the output file to be created>
precision           <precision to be used in the output file>
yearsandsteps       <ActionAtTime to determine when to print>
```

Note that this printer type can aggregate more than one predator into a combined pseudo predator, and more than one prey into a pseudo prey, for the output file.

The optional <precision> value is used to specify the number of digits to be used when printing the information to the specified output file, overriding the default settings for this printer type.

The output that is generated from this printer type is a file containing the following predation information for all the predators specified on the <predatornames> line, consuming all the preys specified on the <preynames> line:

```
year-step-area-age-length-number-biomass-mortality
```

where:
<age> is the label for the prey age group from the age aggregation file
<length> is the label for the prey length group from the length aggregation file
<number> is the number consumed for that timestep/area/age/length combination
<biomass> is the biomass consumed for that timestep/area/age/length combination
<mortality> is the effective annual mortality induced in the prey by the predation for that timestep/area/age/length combination

9.10 LikelihoodPrinter

The printer type to output detailed information about a likelihood component is called "LikelihoodPrinter". This printer type is defined by specifying the likelihood component and timesteps of interest. The file format for this component is given below:

```
[component]
type                 likelihoodprinter
likelihood           <name of the likelihood component>
printfile            <name for the output file to be created>
```

The output that is generated from this printer type is a file containing the internal model information for the likelihood component specified on the <likelihood> line, in the same format as the data in the input file for that component, allowing the user to easily compare the modelled data to the input data. The exact format of this output file will vary according to the likelihood component chosen. Some examples of output are:

1. CatchDistribution likelihood component (see section 8.3)

   ```
   year-step-area-age-length-number
   ```

2. CatchStatistics likelihood component (see section 8.4)

   ```
   year-step-area-age-number-mean[-stddev]
   ```

3. SurveyIndices likelihood component (see section 8.6)

   ```
   year-step-area-index-number
   ```

9.11 LikelihoodSummaryPrinter

The printer type to output summary information about all the likelihood components that are in use for the current simulation is called "LikelihoodSummaryPrinter". The file format for this component is given below:

```
[component]
type                 likelihoodsummaryprinter
printfile            <name for the output file to be created>
```

The output that is generated from this printer type is a file containing summary information for all the likelihood components that have been defined for the current simulation:

```
year-step-area-component-weight-score
```

where:
<component> is the name of the likelihood component
<weight> is the weight of the likelihood component
<score> is the score from the likelihood component for that timestep/area combination

Chapter 10

Parameter File

The parameter file is used to specify the initial values for the switches that are to be used in the Gadget model (see What Does The # Mean?, section 2.2). This file is specified by a "-i <filename>" command line option when Gadget is started, for example, this would take the parameter information from a file called "inputfile.txt":

```
gadget -s -i inputfile.txt
```

This file contains a list of all the switches, their initial value, the lower and upper bounds and a flag to note whether the optimiser should optimise that switch or not. The first line of this file must contain the column headings, and then subsequent lines should list all switches that are used. An example of this file format is shown below:

```
switch  value    lower    upper    optimise
<name>  <value>  <lower>  <upper>  <0 or 1> ; 1 to optimise this parameter
```

Note that if Gadget is performing a simulation run, then the "optimise" column is still required, although the value of the optimise flag will have no affect on the outcome of the simulation.

In some cases an initial value for a parameter is not known, or a random starting point is wanted (for example, to ensure that the value of the starting point has little affect on the outcome of the optimisation). This is done by specifying the keyword "random" instead of an initial value, which will ensure that Gadget will set the initial value for that parameter to a random point within the bounds. This is shown in the example below, where the initial value for the parameter "age2" will be set to a random point between 1 and 10:

```
age2    random   1        10       1 ; random value between 1 and 10
```

Chapter 11

Optimisation File

The optimisation file is used to specify the type of optimisation to be used, along with any parameters that are needed for the optimisation algorithm. This file is specified by a "-opt <filename>" command line option when Gadget is started, for example, this would take the optimisation information from a file called "optinfo.txt":

```
gadget -l -opt optinfo.txt
```

There are three types of optimisation algorithms currently implemented in Gadget - these are one based on the Hooke & Jeeves algorithm, one based on the Simulated Annealing algorithm and one based on the Broyden-Fletcher-Goldfarb-Shanno ("BFGS") algorithm. These algorithms are described in more detail in the following sections. Gadget can also combine two or more of these algorithms into a single hybrid algorithm, that should result in a more efficient search for an optimum solution.

All the optimisation techniques used by Gadget attempt to minimise the likelihood function. That is, they look for the best set of parameters to run the model with, in order to get the best fit according to the likelihood functions you have specified. Thus, the optimiser is attempting to minimize a single one-dimensional measure of fit between the model output and the data, which can lead to unexpected results.

The optimisation process will tend to produce realistic values for the most significant parameters (typically growth, recruitment in large year classes and fishing selectivity) before finding realistic values for less significant parameters. It should be noted that the parameters governing recruitment and growth in the last few years with data in a simulation will be among the last to settle on realistic values. It is important to ensure that an optimum solution has been reached before accepting the model, especially if the model is to be used to predict the population into the future or to examine the strength of the last few year classes.

11.1 Hooke & Jeeves

11.1.1 Overview

Hooke & Jeeves is a simple and fast optimising method, but somewhat unreliable, and it is often described as a 'hill climbing' technique. From the initial starting point the algorithm takes a step in various 'directions', and conducts a new model run. If the new likelihood score is better than the old one then the algorithm uses the new point as it's best guess. If it is worse then the algorithm retains the old point. The search proceeds in series of these steps, each step slightly smaller than the previous one. When the algorithm finds a point which it cannot improve on with a small step in any direction then it accepts this point as being the 'solution', and exits.

It can be seen that this renders the scheme vulnerable to producing local solutions, accepting a local dip as being the solution even if a better solution exists elsewhere, beyond a local 'hill' that the algorithm cannot see past. In order to combat this tendency it is strongly recommended that you re-run the optimisation, using the final point of one run as the start of the next. This will effectively re-set the searching step size to large steps, and give Gadget a chance of escaping from local solutions. Finding the same result twice in a row does not guarantee it is the best possible solution, but finding different results certainly indicates that the larger result is not the solution you are seeking.

The Hooke & Jeeves algorithm used in Gadget is derived from that originally presented by R. Hooke and T. A. Jeeves, "Direct Search Solution of Numerical and Statistical Problems" in the April 1961 (Vol. 8, pp. 212-229) issue of the Journal of the ACM, with improvements presented by Arthur F Kaupe Jr., "Algorithm 178: Direct Search" in the June 1963 (Vol 6, pp.313-314) issue of the Communications of the ACM.

When compiled with OpenMP, Gadget can also execute a parallel version of this algorithm, selecting the opcion *-parallel*. The implementation of this option is described in the paper VÃązquez, S., et al. "Novel parallelization of simulated annealing and Hooke & Jeeves search algorithms for multicore systems with application to complex fisheries stock assessment models" on Journal of Computational Science, 17, 599âĂŞ608 (2016).

This parallel version can get profit from the new multi-core hardware to speedup the execution. It support two options: speculative and reproducible. The latter follows exactly the same sequence as the original sequential algorithm. In the case of the speculative option, the execution will be faster but the result could not follow the same pattern that the serial version, maybe producing a different result. This parallel version must be executed with an even number of threads, so, if you select this number using OMP_NUM_THREADS environtment variable, choose it correctly (and lower or equal to the number of cores of the computer).

Hooke & Jeeves is the default optimisation method used for Gadget, and will be used if no optimisation information file is specified.

11.1.2 File Format

To specify the Hooke & Jeeves algorithm, the optimisation file should start with the keyword "[hooke]", followed by (up to) 4 lines giving the parameters for the optimisation algorithm. Any parameters that are not specified in the file are given default values, which work reasonably well for simple Gadget models. The format for this file, and the default values for the optimisation parameters, are shown below:

```
[hooke]
hookeiter  1000  ; number of hooke & jeeves iterations
hookeeps   1e-04 ; minimum epsilon, hooke & jeeves halt criteria
rho        0.5   ; step length adjustment factor
lambda     0     ; initial value for the step length
```

11.1.3 Parameters

hookeiter

This is the maximum number of Gadget model runs that the Hooke & Jeeves algorithm will use to try to find the best solution. If this number is exceeded, Gadget will select the best point found so far, and accept this as the 'solution', even though it has not met the convergence criteria. A warning that Gadget has stopped without finding a solution that meets the convergence criteria will be printed.

hookeeps

This is the criteria for halting the Hooke & Jeeves algorithm at a minimum, and accepting the current point as the 'solution'. The algorithm has "converged" when the size of each step in the search has been reduced to less than this value, and no further improvements can be made. This means that each parameter in the Gadget model is less than hookeeps from their value at the (local) optimum. Larger values of hookeeps give a quicker running time, but a less accurate estimate of the minimum. Conversely, smaller values will give a longer running time, but a more accurate estimate of the minimum.

rho

This is the resizing multiplier for the step length of the search made by the algorithm. The Hooke & Jeeves algorithm works by taking "steps" from one estimate of a optimum, to another (hopefully better) estimate of the optimum. Taking big steps gets to the minimum more quickly, at the risk of 'stepping over' an excellent point. When no improvements to the minimum can be made by taking steps of the current length, the step length is reduced by multiplying it by rho.

Small values of rho correspond to big step length changes, which make the algorithm run more quickly. However, there is a chance that these big changes will accidentally overlook a promising point, leading to non-convergence. Larger values of rho correspond to smaller step length changes, which force the algorithm to carefully examine nearby points instead of optimistically forging ahead, which improves the probability of convergence. The value of rho must be between 0 and 1.

lambda

This is the initial value for the size of the steps in the search, which will be used for the the first search, before any modification to the step length. All the parameters in the Gadget model are initially scaled so that their value is 1, and the initial search will then look at the points $1 \pm \lambda$ for the next optimum. Setting lambda to zero will set the initial value for the step length equal to rho. The value of lambda must be between 0 and 1.

11.2 Simulated Annealing

11.2.1 Overview

Simulated Annealing is a global optimisation method that distinguishes between different local minimum. From the initial starting point the algorithm takes a random step in various directions, and conducts a new model run. If the new likelihood score is better than the old one then the algorithm uses the new point as it's best guess. If it is worse then the algorithm may accept this point, based on the probabilistic "Metropolis Criteria", and thus the algorithm can escape from a local minimum. The search proceeds in series of these steps, with the point that gives the overall lowest likelihood score is stored as a "best point". The algorithm exits when a stable point is found which cannot be improved on with a small step in any direction, and the Metropolis Criteria rejects all the steps away from the current best point. The best point is then accepted as being the 'solution'.

The Metropolis Criteria will accept a move in an uphill direction (ie. a point with a worse likelihood score) based on a function of both the size of the move and a parameter termed "temperature", and is given in equation 11.1 below:

$$
\begin{aligned}
M &= e^{\frac{-\Delta F}{t}} \\
P &= \begin{cases} 1 & \text{if } M > r \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{11.1}
$$

where:
$< \Delta F >$ is the change in likelihood score
< t > is the "temperature" of the algorithm
< r > is a random number between 0 and 1

Note that when the "temperature" is very high (t $\rightarrow \infty$), the Metropolis Criteria will always accept any 'uphill' move, and the Simulated Annealing algorithm will simplify to a form of a random search algorithm. Conversely, when the temperature is very low (t $\rightarrow$ 0), the Metropolis Criteria will always reject any 'uphill' move, and the Simulated Annealing algorithm will simplify to a form of a hill-climbing algorithm, similar to Hooke & Jeeves. By slowly reducing the temperature of the algorithm, the number of uphill moves that are accepted are reduced and the algorithm will find a minimum.

The terminology for the Simulated Annealing algorithm comes from a metaphor with the thermodynamic process of cooling molten metals. At very high temperatures, the atoms in molten metals move very fast, and slow considerably at lower temperatures. If the metal is cooled slowly (termed "annealing"), the atoms have time to line up to form a highly ordered, strong, crystalline structure. If the metal is cooled too quickly ("quenching") the resulting solid will have a weaker, less ordered, structure.

In comparison to the Hooke & Jeeves optimisation algorithm, where Hooke & Jeeves performs a 'local' stepwise search, Simulated Annealing searches much more widely over the surface in order to find the best point. By doing this it is less likely than Hooke & Jeeves to be fooled by a local minimum, and more likely to home in on the true optimum. However the price to paid for doing this is that it can take considerably more computer time to reach a solution.

The Simulated Annealing algorithm used in Gadget is derived from that presented by Corana et al, "Minimising Multimodal Functions of Continuous Variables with the 'Simulated Annealing' Algorithm" in the September 1987 (Vol. 13, pp. 262-280) issue of the ACM Transactions on Mathematical Software and Goffe et al, "Global Optimisation of Statistical Functions with Simulated Annealing" in the January/February 1994 (Vol. 60, pp. 65-100) issue of the Journal of Econometrics.

As in the case of Hook & Jeeves algorithm, if Gadget has been compiled with the OpenMP option, there are two additional parallel versions that can be selected with the -parallel option: speculative (-parallel spe) and reproducible (-parallel rep). By default, the speculative version is selected whan only -parallel flag is present. These versions will use the multi-core capability of your computer to speed up the execution. The algorithms are described on Vázquez et al.:"Novel parallelization of simulated annealing and Hooke & Jeeves search algorithms for multicore systems with application to complex fisheries stock assessment models" published on Journal of Computational Science, 17, 599-608 (2016).

The reproducible version follows exactly the same sequence as the original serial algorithm, and thus it finishes in the same point and with the same likelihood score, but in less time, using all the available cores of your computer, except if you limit the number using the environtment vaiable OMP_NUM_THREADS. The speculative version, is allowed to change the parameter modification sequence in order to increase the parallelism, and thus it can finish in a different

point and with a different likelihood value. In this latter case, some specification variables of the sequential algorithm were adapted to prevent the algorithm from converging to a worse likelihood value. For example, the number of *effective* evaluations is taken into account as ending criterion instead of the number of total evaluations.

11.2.2 File Format

To specify the Simulated Annealing algorithm, the optimisation file should start with the keyword "[simann]", followed by (up to) 11 lines giving the parameters for the optimisation algorithm. Any parameters that are not specified in the file are given default values, which work reasonably well for simple Gadget models. The format for this file, and the default values for the optimisation parameters, are shown below:

```
[simann]
simanniter 2000  ; number of simulated annealing iterations
simanneps  1e-04 ; minimum epsilon, simann halt criteria
t          100   ; simulated annealing initial temperature
rt         0.85  ; temperature reduction factor
nt         2     ; number of loops before temperature adjusted
ns         5     ; number of loops before step length adjusted
vm         1     ; initial value for the maximum step length
cstep      2     ; step length adjustment factor
lratio     0.3   ; lower limit for ratio when adjusting step length
uratio     0.7   ; upper limit for ratio when adjusting step length
check      4     ; number of temperature loops to check
```

11.2.3 Parameters

simanniter

This is the maximum number of Gadget model runs that the Simulated Annealing algorithm will use to try to find the best solution. If this number is exceeded, Gadget will select the best point found so far, and accept this as the 'solution', even though it has not met the convergence criteria. A warning that Gadget has stopped without finding a solution that meets the convergence criteria will be printed.

simanneps

This is the criteria for halting the Simulated Annealing algorithm at a minimum, and accepting the current point as the 'solution'. The algorithm has "converged" if the point found at the end of a temperature loop is within simanneps of the best point, and is also within simanneps of the point found at the end of the last 'check' temperature loops. This will mean that during each of the last 'check' temperature loops, the accepted point has moved less than simanneps away from the overall best point, showing that the best point found so far is stable. Lower values for simanneps will give a more stable estimate of the minimum, at the cost of a longer running time. Note that the convergence criteria is only checked at the end of each temperature loop.

t

This is the initial value for the "temperature" of the Simulated Annealing algorithm, used to control the speed of the convergence of the algorithm, using the Metropolis Criteria given in equation 11.1 above. High values for the temperature ($t \gg \Delta F$) will mean that most of the uphill moves are accepted, and the algorithm will be performing a random search, which is computationally very intensive. Low values for the temperature ($t \ll \Delta F$) will mean that most of the uphill moves are rejected and the algorithm will be performing an inefficient local search.

rt

This is the temperature reduction factor, used to lower the temperature of the algorithm as it moves closer to the minimum. Higher values here will mean that the temperature is reduced slowly, which will mean that the algorithm is will examine the search area more thoroughly, leading to better convergence at a cost of much higher computational time. Lower values of rt will mean that the temperature is reduced quickly, leading to faster convergence, possibly to a local minimum that the algorithm cannot escape from (this is analogous to ”quenching” in the thermodynamic metaphor). The value of rt must be between 0 and 1.

nt

This is the number of loops that the algorithm will perform before reducing the temperature parameter. Higher values here will mean that the algorithm will explore the current search area more thoroughly, improving the chance of finding a global minimum, at the cost of considerably higher computational time. Conversely, lower values will mean that the current search area is not explored as thoroughly, with a corresponding reduction in computation time. Note that each temperature loop consists of all of the ”ns” step loops.

ns

This is the number of loops that the algorithm will perform before adjusting the maximum step length parameter. The maximum step length is periodically adjusted so that approximately half of all moves are accepted, since too many, or too few, accepted moves is a waste of computational effort.

When the algorithm adjusts the maximum step length, the ratio of accepted moves to rejected moves since the last adjustment of the maximum step length is taken into account. If more moves are accepted than rejected, the maximum step length will fall and so the search will focus on the most promising area. However, if more moves are rejected that accepted, the maximum step length will rise so the search area will widen.

vm

This is the initial maximum step length for the Simulated Annealing algorithm. The maximum step length will get adjusted during the optimisation process. Note that each of the steps taken by the Simulated Annealing algorithm are not uniform, and will have a random length of between zero and the maximum step length.

It is important to note that unlike the Hooke & Jeeves algorithm, the parameters are not scaled during the optimisation process, and so the maximum step length should be selected so that the initial search area will cover the area of interest. For efficient use of the Simulated Annealing algorithm, it is recommended that the parameters to be estimated are scaled (by the user) so that they are all approximately the same order of magnitude.

cstep, lratio and uratio

These parameters control how the maximum step length for the Simulated Annealing algorithm is adjusted, at the end of each of the ”ns” step loops. The algorithm keeps track of the number of moves that are accepted and rejected for each direction, and then adjusts the value of the maximum step length for each direction so that the ratio of accepted to rejected moves is approximately 50:50. The maximum step length for each direction is adjusted according to equation 11.2 below:

$$vm_i = \begin{cases} vm_i * (1 + \frac{C(R_i - U)}{L}) & \text{if } R_i > U \\ vm_i / (1 + \frac{C(L - R_i)}{L}) & \text{if } R_i < L \\ vm_i & \text{otherwise} \end{cases} \tag{11.2}$$

where:
$< R_i >$ is the ratio of accepted moves for direction i
< C > is the value of the "cstep" parameter
< U > is the value of the "uratio" parameter
< L > is the value of the "lratio" parameter

check

This is the number of temperature loops that the Simulated Annealing algorithm will check to confirm that the current best point that has been found is a stable minimum, so that it can be accepted as a solution.

11.3 BFGS

11.3.1 Overview

BFGS is a quasi-Newton optimisation method that uses information about the gradient of the function at the current point to calculate the best direction to look in to find a better point. Using this information, the BFGS algorithm can iteratively calculate a better approximation to the inverse Hessian matrix, which will lead to a better approximation of the minimum value.

From an initial starting point, the gradient of the function is calculated and then the algorithm uses this information to calculate the best direction to perform a linesearch for a point that is "sufficiently better". The linesearch that is used in Gadget to look for a better point in this direction is the "Armijo" linesearch. The algorithm will then adjust the current estimate of the inverse Hessian matrix, and restart from this new point. If a better point cannot be found, then the inverse Hessian matrix is reset and the algorithm restarts from the last accepted point.

For a point at a stable minimum, the magnitude of the gradient vector will be zero, since there is no direction that the algorithm can move in to find a better local point. However, finding such a point using an iterative process can take an infinite number of steps, so the algorithm exits when the magnitude of the gradient vector is less than a small number. The current point is then accepted as being the 'solution'.

The "Armijo" linesearch calculates the stepsize that is to be used to move to a point that is "sufficiently better", along the search direction vector calculated from the gradient, to be β^n, where n is the first integer that satisfies the Armijo rule given by inequality 11.3 below:

$$f(x) - f(x + \beta^n d) \geq -\sigma \beta^n d \nabla f(x)^T \tag{11.3}$$

where:
$< \nabla f(x) >$ is the gradient of the function at the current point
< d > is the search direction vector

In comparison to the other optimising algorithms that are currently implemented in Gadget, BFGS performs a local search, and so it doesn't cover the wide search area that the Simulated Annealing algorithm can use to look for an optimum. BFGS will usually take more computer time to find an optimum than the Hooke & Jeeves algorithm, since numerically calculating the gradient of the function is computationally very intensive. However, the optimum found by the

BFGS algorithm will usually be better than that found by the Hooke & Jeeves algorithm, since a gradient search method is usually more accurate than a stepwise search method.

The BFGS algorithm used in Gadget is derived from that presented by Dimitri P Bertsekas, "Nonlinear Programming" (2^{nd} edition, pp22-61) published by Athena Scientific. The forward difference gradient algorithm used to calculate the gradient is derived from that presented by Dennis and Schnabel, "Numerical Methods for Unconstrained Optimisation and Nonlinear Equations" ("Classics" edition, published by SIAM).

11.3.2 File Format

To specify the BFGS algorithm, the optimisation file should start with the keyword "[bfgs]", followed by (up to) 7 lines giving the parameters for the optimisation algorithm. Any parameters that are not specified in the file are given default values, which work reasonably well for simple Gadget models. The format for this file, and the default values for the optimisation parameters, are shown below:

```
[bfgs]
bfgsiter   10000 ; number of bfgs iterations
bfgseps    0.01  ; minimum epsilon, bfgs halt criteria
sigma      0.01  ; armijo convergence criteria
beta       0.3   ; armijo adjustment factor
gradacc    1e-06 ; initial value for gradient accuracy
gradstep   0.5   ; gradient accuracy adjustment factor
gradeps    1e-10 ; minimum value for gradient accuracy
```

11.3.3 Parameters

bfgsiter

This is the maximum number of Gadget model runs that the BFGS algorithm will use to try to find the best solution. If this number is exceeded, Gadget will select the best point found so far, and accept this as the 'solution', even though it has not met the convergence criteria. A warning that Gadget has stopped without finding a solution that meets the convergence criteria will be printed.

bfgseps

This is the criteria for halting the BFGS algorithm at a minimum, and accepting the current point as the 'solution'. The algorithm has "converged" if the magnitude of the gradient vector is less than bfgseps. Lower values of bfgseps will give a better estimate of the minimum, at the cost of a considerably increased running time. At the true minimum, the magnitude of the gradient vector will be zero, since there would be no direction for the algorithm to move to, but this point could take an infinite number of iterations to find.

sigma

This is the criteria for stopping the Armijo linesearch at a point that is "sufficiently better", and recalculating the gradient of the function at this new point. Lower values of sigma will increase the size of the acceptance region of the Armijo linesearch (by relaxing the condition for the inequality, see equation 11.3 above), which will mean that the linesearch will stop earlier, leading to larger steps by the BFGS algorithm. Conversely, higher values of sigma will decrease the size of the acceptance region, which will mean that the BFGS algorithm will take smaller steps. The value of sigma must be between 0 and 1, and should be close to zero.

beta

This is the adjustment factor for the Armijo linesearch, used to calculate the size of the step taken by the linesearch to find the next point. Lower values of beta will mean that the size of the step that the algorithm tries to take in the direction of the gradient vector is smaller, but the step is more likely to be in the acceptance region. Conversely, higher values of beta will result in larger steps being tried, but these steps are more likely to be rejected since they may be outside the acceptance region. The value of beta must be between 0 and 1.

gradacc, gradstep and gradeps

These parameters control the accuracy that is used for the gradient calculations. The gradient of the function is calculated numerically by Gadget using a forward difference algorithm, as shown in equation 11.4 below:

$$\nabla f(x) \approx \frac{f(x+\delta x) - f(x)}{\delta x} \tag{11.4}$$

where:
$< \delta >$ is the value of the "gradacc" parameter

When the BFGS algorithm is reset (that is, if the Armijo linesearch fails to find a better point) the gradient accuracy parameter is made smaller to increase the level of accuracy that is used in the gradient calculations. This is done by multiplying the gradacc parameter by the gradstep parameter, which is a simple reduction factor (and as such must be between 0 and 1). To prevent the gradacc parameter getting too small, the BFGS algorithm will stop once the value of gradacc is less that the value of gradeps. Both gradacc and gradeps must be between 0 and 1, with gradeps smaller than gradacc, and the gradient calculations are more accurate when the gradacc parameter is very small.

11.4 Parallel Multirestart Adaptive Particle Swarm Optimization

11.4.1 Overview

The Parallel Multirestart Adaptive Particle Swarm Optimization (PMA PSO) algorithm operates on a population of solutions, called swarm. For each iteration new candidate solutions, named particles, are generated by applying movements through the search-space using both local knowledge of the particles and global information of the method.

PMA PSO is a modification of the PSO algorithm (Kennedy and Eberhart, 1995), which follows the next scheme: at the beginning, a set of N D-dimensional particles that conform the initial swarm are randomly generated within the bound constraints of the problem. Then, a certain movement is applied to each particle through the following expressions:

$$v_i \leftarrow \omega v_i + \varphi_l r_l (pbest_i - p_i) + \varphi_g r_g (gbest - p_i) \tag{11.5}$$

$$p_i \leftarrow p_i + v_i \tag{11.6}$$

where p_i is the current solution for the particle i of the swarm, v_i is the vector velocity, ω is the inertia constant, φ_g is the the social coefficient, φ_l is the cognitive coefficient, r_g and r_l are two random numbers between zero and one, $pbest_i$ is the local best solution found by the current particle, and $gbest$ is the best global solution found by any particle during the execution of the

method. Using these two formulas, a new solution is created through the sum of the current position with the new velocity vector created, which is generated taking into account the following three factors in Equation (11.5) in this order:

- Previous velocities and inertia weight: the velocity calculated for a particle "i" in past iterations is taken into account to compute the velocity in the current iteration. The inertial weight (ω) is a configurable parameter of the heuristic, with values between zero and one, to adjust the importance of this accumulation of speeds.

- Cognitive component: It is a vector calculated as the difference between the best value reached by the considered particle i and the current value of that particle. This vector takes into account the local knowledge about previous visited points. To adjust the impact of the influence of this cognitive component, it is multiplied by a random value between zero and one (this is what gives stochasticity to the method), and also by a configurable parameter called cognitive coefficient (φ_l).

- Social component: It is a vector computed as the difference between the global best solution found among all the particles from the swarm and the current particle solution. Therefore, the social component takes into account the influence of the global solution when new solutions are created. Also for this case, the vector generated is multiplied by a random number between zero and one, and by a configurable parameter known as social coefficient (φ_g).

The choice of values for the size of the population (N) and the aforementioned configuration parameters is not trivial, and it will be problem-dependent. To solve this issue, PMA PSO includes a self-tuning of the configuration parameters to intensify the search when there is a promising result and obtain a good solution in a short time or, otherwise, to diversify it with a conservative parameter configuration when the algorithm is stuck in the proximity of a local optimum. Moreover, PMA PSO also includes the following extra characteristics: (1) an additional mechanism to restart the solutions of the algorithm, without losing the best solution found, in order to explore other regions when the search is stagnated; and (2) a parallel computation of the evaluations of the cost function in each iteration of the main loop of the method, using OpenMP, to reduce the execution time of the algorithm.

11.4.2 File Format

To specify the PMA PSO algorithm, the optimisation file should start with the keyword ”[pso]”, followed by 2 lines giving the parameters for the optimisation algorithm. Any parameters that are not specified in the file are given default values, which work reasonably well for simple Gadget models. This algorithm has self-adaptation of configuration parameters, only needing parameters related with the stopping criterion. The format for this file, and the default values for the optimisation parameters, are shown below:

```
[pso]
goal        1e-6  ; optimization goal value
psoiter     100000000  ; maximum number of iterations
```

11.4.3 Parameters

goal

This is the criterion for halting the PMA PSO algorithm, accepting the current best point as the ’solution’. The algorithm has ”converged” if the value described by goal is overcomed.

psoiter

This is the maximum number of Gadget model runs that the PSO algorithm will use to try to find the best solution. If this number is exceeded, Gadget will select the best point found so far, and accept this as the 'solution', even though it has not met the convergence criteria. A warning that Gadget has stopped without finding a solution that meets the convergence criteria will be printed.

11.5 Parallel Multirestart Adaptive Differential Evolution

11.5.1 Overview

Differential Evolution (Storn and Price, 1997) is an evolutive method where, for each iteration, new solutions are created through difference operations. Starting from a population of N randomly generated D-dimensional solutions within the bounds constraints of the problem, the method creates, using a Mutation Strategy (MStr), a set of candidate solutions for each iteration, which are intended to replace the solutions stored in the population. These mutation operations represent the difference among two or more solutions randomly chosen from the population, multiplied by the so-called Mutation Factor (F). Among the many types of possible mutation strategies, this implementation uses the following ones:

- `DE/best/2/bin`:

$$new\_point_i \leftarrow xbest + F(p_a - p_b) + F(p_c - p_d) \tag{11.7}$$

- `DE/current-to-rand/1/bin`:

$$new\_point_i \leftarrow p_i + F(p_a - p_i) + F(p_b - p_c) \tag{11.8}$$

- `DE/current-to-best/1/bin`:

$$new\_point_i \leftarrow p_i + F(xbest - p_i) + F(p_a - p_b) \tag{11.9}$$

- `DE/rand-to-best/1/bin`:

$$new\_point_i \leftarrow p_a + F(xbest - p_a) + F(p_b - p_c) \tag{11.10}$$

where p_i is the current solution, and p_a, p_b, p_c and p_d are solutions randomly selected belonging to the current population, $xbest$ is the current best solution of the population, and new_point_i is the candidate solution created.

The mutation strategy is applied element by element in some of the D positions of the solution. The crossover constant (CR) is a configuration parameter to decide how many positions of the candidate solution come from the original solution, and how many are the result of the mutation process. For each existing solution in the population, a new candidate is generated, but this candidate only replaces the solution from which it has been derived if it is better. This process will be repeated until a specific stopping criteria is met. At that point, the best one of the N solutions in the population is obtained as output of the method.

The choice of values for the size of the population (N) and the aforementioned configuration parameters is not trivial, and it will be problem-dependent. To solve this issue we propose a modification to this algorithm, which has been called Parallel Multi-restart Adaptive DE (PMA DE). It includes a self-tuning of the configuration parameters to intensify the search when there is a promising result and obtain a good solution in a short time or, otherwise, to diversify it with

a conservative parameter configuration when the algorithm is stuck in the proximity of a local optimum. Moreover, PMA DE also includes the following extra characteristics: (1) an additional mechanism to restart the solutions of the algorithm, without losing the best solution found, in order to explore other regions when the search is stagnated; and (2) a parallel computation of the evaluations of the cost function in each iteration of the main loop of the method, using OpenMP, to reduce the execution time of the algorithm.

11.5.2 File Format

To specify the PMA DE algorithm, the optimisation file should start with the keyword "[DE]", followed by 2 lines giving the parameters for the optimisation algorithm. Any parameters that are not specified in the file are given default values, which work reasonably well for simple Gadget models. This algorithm has self-adaptation of configuration parameters, only needing parameters related with the stopping criterion. The format for this file, and the default values for the optimisation parameters, are shown below:

```
[DE]
goal     1e-6; optimization goal value
iter     100000000  ; maximum number of iterations
```

11.5.3 Parameters

goal

This is the criterion for halting the PMA DE algorithm, accepting the current best point as the 'solution'. The algorithm has "converged" if the value described by goal is overcomed.

iter

This is the maximum number of Gadget model runs that the DE algorithm will use to try to find the best solution. If this number is exceeded, Gadget will select the best point found so far, and accept this as the 'solution', even though it has not met the convergence criteria. A warning that Gadget has stopped without finding a solution that meets the convergence criteria will be printed.

11.6 Combining Optimisation Algorithms

11.6.1 Overview

This method attempts to combine the global search of algorithms such as Simulated Annealing, PMA-PSO or PMA-DE, with the more rapid convergence of the local searches performed by the Hooke & Jeeves and BFGS algorithms. It relies on the observation that the likelihood function for many Gadget models consists of a large 'valley' in which the best solution lies, surrounded by much more 'rugged' terrain.

A period of global search at the start of the optimisation run, performed applying Simulated Annealing, PMA-PSO or PMA-DE, serves to move the search into this valley, at which point global search algorithms are typically more inefficient than local ones. Thus at this point, the Hooke & Jeeves algorithm and/or the BFGS algorithm takes over and homes in on a solution within that valley. Hopefully the global search algorithms will have moved the current point to the correct side of any 'hills' to avoid the local searches becoming trapped into an unrealistic local minimum.

11.6.2 File Format

To specify a combination of optimisation algorithms, the optimisation file should simply list the algorithms that should be used, along with the parameters for each optimisation algorithm. Any parameters that are not specified in the file are given default values, which work reasonably well for simple Gadget models. As an example, the format for this file used to specify all optimisation algorithms, and the default values for the optimisation parameters, is shown below:

```
[simann]
simanniter 2000  ; number of simulated annealing iterations
simanneps  1e-04 ; minimum epsilon, simann halt criteria
t          100   ; simulated annealing initial temperature
rt         0.85  ; temperature reduction factor
nt         2     ; number of loops before temperature adjusted
ns         5     ; number of loops before step length adjusted
vm         1     ; initial value for the maximum step length
cstep      2     ; step length adjustment factor
lratio     0.3   ; lower limit for ratio when adjusting step length
uratio     0.7   ; upper limit for ratio when adjusting step length
check      4     ; number of temperature loops to check
[pso]
goal        1e-6  ; optimization goal value
psoiter     100000000  ; maximum number of iterations
[DE]
goal     1e-6; optimization goal value
iter     100000000  ; maximum number of iterations
[hooke]
hookeiter  1000  ; number of hooke & jeeves iterations
hookeeps   1e-04 ; minimum epsilon, hooke & jeeves halt criteria
rho        0.5   ; step length adjustment factor
lambda     0     ; initial value for the step length
[bfgs]
bfgsiter   10000 ; number of bfgs iterations
bfgseps    0.01  ; minimum epsilon, bfgs halt criteria
sigma      0.01  ; armijo convergence criteria
beta       0.3   ; armijo adjustment factor
gradacc    1e-06 ; initial value for gradient accuracy
gradstep   0.5   ; gradient accuracy adjustment factor
gradeps    1e-10 ; minimum value for gradient accuracy
```

It should be noted that the optimisation algorithms will be performed in the order that they are specified in the input file. Thus, for this example the order will be: first Simulated Annealing, second PMA-PSO, third PMA-DE, fourth Hooke & Jeeves and finally BFGS. Notice that this example is provided for completeness in order to show that Gadget can apply any combination of search algorithms in any arbitrary order. However, it is unlikely that you need to execute all the global optimisation algorithms (SA, PMA-PSO and PMA-DE), and it is better that only the most suitable for the problem is executed.

11.6.3 Parameters

The parameters for this combined optimisation algorithm are the same as for the individual algorithms, and are described in sections 11.1.3 (for the Hooke & Jeeves parameters), 11.2.3 (for the Simulated Annealing parameters), 11.4.3 (for PMA PSO parameters), 11.5.3 (for PMA Differential Evolution parameters), and 11.3.3 (for the BFGS parameters).

11.7 Repeatability

The optimisation algorithms used by Gadget contain a random number generator, used to randomise the order of the parameters (to ensure that the order of the parameters has no effect on the optimum found) and to generate the initial direction chosen by the algorithm to look for a solution. For the Simulated Annealing algorithm, this is also affects the Metropolis criteria used to accept any changes in an 'uphill' direction.

The use of a random number generator means that it is unlikely that two optimising runs will produce exactly the same 'optimum', even if they start from the same point (although it is hoped that the optima would be very similar if a stable starting point was chosen). This causes a problem with repeatability, since there is no guarantee that an optimising run can be repeated. However, it is possible to 'seed' the random number generator with an integer to avoid this problem. To specify a seed for the random number generator, an extra line must be added to the optimisation file, as shown below:

```
seed       n     ; seed the random number generator with n
```

It is also possible to specify the value that is used to seed the random number generator from the command line, by starting Gadget with the "-seed <number>" switch. It should be noted that any value for the seed that is specified in the optimisation file will override the value that is specified as a command line option.

Chapter 12

Output Files

The model output files contain information about the optimisation process (and not information about the stocks in the model - see the section on the Print Files, section 9, for information on these). The output files are specified by some of the commandline parameters used to start Gadget. There are three types of output file.

12.1 Parameter Output

The parameter output file is automatically generated by Gadget every time that Gadget is run. If Gadget is not started with the "-p <filename>" option, this file will be called "params.out". This file contains information about the values of the switches at the end of the Gadget run, in the same format as for the column output file. In the header of this file there will be information about the Gadget run, followed by the final overall likelihood score and the total number of iterations. There is then the information about the switches used in the Gadget run - the name, value, lower bound, upper bound and whether the switch is to be optimised.

This file is written in the same format as the parameter input file, and it can be used as the starting point for a subsequent Gadget run.

12.2 Likelihood Output

The likelihood output file is generated when Gadget is started with the "-o <filename>" option, and the output is written to the file specified after that option, so, for example, this would write the output to a file called "likelihood.txt":

```
gadget -l -o likelihood.txt
```

The likelihood output file is split into three sections, separated by a line containing a brief comment. The first section lists the names of the switches used in the model, together with some information about where that switch is used.

The second part of the likelihood output file contains information about the likelihood components used in the optimisation of the model. The names of the components are listed, along with an identifier to the type and the weight assigned to the component.

The final part of the likelihood output file contains the output from the optimisation process. There is a single line for each iteration of the optimisation, containing the iteration number followed by a tab character, then the value of each switch, followed by a gap (2 tab characters),

followed by the value of each of the likelihood components, followed by another gap (again, 2 tab characters) followed by the overall likelihood score for that iteration.

For a long optimisation run, this can result in this file being quite large. The default option is for this line to be written for each iteration, but this can be changed using the "-print <number>" option, which will only write out this line every <number> iterations, as shown in the example below:

```
gadget -l -o likelihood.txt -print 10
```

which will set Gadget to write this information, to the file "likelihood.txt", after every 10th iteration.

12.3 Log Output

The log output file is generated when Gadget is started with the "-log <filename>" option, and the output is written to the file specified after that option, so, for example, this would write the output to a file called "gadget.log":

```
gadget -log gadget.log
```

The log file contains a listing of the actions that Gadget is performing, as those actions are being performed - an entry can be made whenever data is read from a file, or a likelihood score is calculated, for example. The log file is of most use when initially setting up and debugging a Gadget model, since it makes it easy to trace what Gadget is doing, so, for example, the details of the last actions that were performed before displayed a warning message are recorded.

This log file that is generated can be a large file if Gadget is performing an optimising run. It is possible to control the amount of information that is written to the log file using the "-loglevel <number>" command line option. The valid values for the loglevel number, and the associated meanings, are given in the list below:

loglevel	**meaning**
0	no output, only used in conjunction with the Paramin optimiser
1	output informational messages only (cannot be set from the command line)
2	also output error messages (default value for an optimising run)
3	also output warning messages (default value for a simulation run)
4	also output debug messages (only useful when debugging Gadget code)
5	also output detailed messages (default value if a log file is specified)

Examples of valid loglevel values include:

1. specifying a log file (called "gadget.log") for an optimising run that only includes informational and error messages (and not the more detailed messages that would be written to the log file by default)

   ```
   gadget -l -log gadget.log -loglevel 2
   ```

2. specifying a log file for a simulation run that includes error and warning messages (note that it is not possible to disable the warning messages during a simulation run)

   ```
   gadget -s -log gadget.log -loglevel 3
   ```

Chapter 13

Paramin

This chapter describes how to run Gadget with paramin which runs the optimisation routines in parallel across a network of processors which will result in a much faster optimisation run. The current version of paramin is implemented with MPI, Message Passing Interface, which handles all the message passing between the master process, paramin, and the slave processes, which are Gadget simulation runs. The setup is very similar to a normal Gadget run, with all the Gadget input files are the same, it's only the optimisation execution that differs.

13.1 Installation

In order to begin one must first download and install a version of MPI. This version of paramin is made with an implementation of MPI-2 called OPEN-MPI, one can download the MPI library and a wrapper compiler called mpic++ from their website www.open-mpi.org. Now before trying to run paramin make sure you have the newest version of Gadget downloaded. Now open the Makefile with a text-editor and uncomment the section referring to a Gadget Network version and comment out the lines which refer to an install without MPI. Now you should open a console window and navigate to the folder where you saved Gadget. Now you can simply type make in the console and the program should be compiled and you get an executable, let's assume it's called gadget-PARA. Now paramin is dependent on some Gadget objects, so you now need to type the following in the console:

```
make libgadgetinput.a
```

Now you are ready to start compiling paramin. Download the newest version of paramin and open the paramin folder in the console. You need to edit the Makefile and add the path to the directory where you installed Gadget as described above. Now you can run the make command to compile paramin and get an executable, let's assume it's called paramin.

13.2 Running paramin

To be able to run paramin it's ideal to add the path of the two executables, gadget-PARA and paramin to the global variable PATH. If you're running a bash console you can edit the `.bashrc` to add it, you can also copy the executables to the directory of PATH, but you probably need root access to do that. Now you can call the functions from the console. Here is an example of a typical paramin run:

```
mpirun paramin -i <filename> -opt <filename> -network <filename>
    -o <filename> -function gadget-PARA -s -n -i <filename> > <filename>
```

Now because paramin is implemented with MPI we need to call mpirun when we run it. Now there are several files we need to call and a few switches we need. The `-i <filename>` switch tells paramin which file holds the parameters and their initial values, this file is the same as the params.in file used in Gadget (see Parameter Files, chapter 10, for more information on the format of this file). The `-i` switch with the same `<filename>` has to be on Gadget as well.
Next we need to specify the optimization parameters with the `-opt <filename>` switch, the preceding file contains the information of which type of optimization run to perform for more information see Parameter Files, chapter 10, it's the same format as the optimization file for Gadget.
The next switch tells paramin how many subprocesses paramin will spawn, this switch is not in Gadget. Note that having at least 8 subprocesses will greatly improve the time of the run, but there is no need to let the number of subprocesses exceed the number of parameters, optimally it should be best to have as many subprocesses as the number of parameters, but then one should have access to at least that many processing cores. A sample file which tells paramin to spawn 30 subprocesses looks like this:

```
;
;Sample network file
;
numproc    30
```

The `-o <filename>` switch specifies which file to output the optimized parameters. This file will also contain information about the optimization run, how much overall time was taken and how much time was taken for each optimizing algorithm. The format of this outputted file is the same as the one used as input, so the output can be used as a starting point for later runs.
The `-function gadget-PARA` -switch tells paramin the name of the executable it will spawn as it's subprocess, in this case it's the gadget-PARA executable we made earlier. The user can also make his own function to optimize with paramin. Now Gadget also needs parameters, we pass to it the `-s` and `-n` switch to make it go to network mode, then Gadget knows it's running with paramin. Finally we can add `> <filename>` to specify a file for the output of the run, this is not necessary but recommended.

13.2.1 Additional information

If you're running a potentially very long run on a remote host it's recommended to use the `screen` utility, for more information type `man screen` in the console. It allows you to detach the session so it lives after you log out of your SSH session. In short you can type `screen` at the console to open a new screen session. Now run the script or program you want to run and type `ctrl-a d`, and you detach the session. Now you can safely exit from the remote host. To retrieve your session on your next login simply type `screen -r` and you enter the screen session, then you can close it by typing `exit` in the console. Another option is to run the desired command with the `nohup` prefix, it makes the command immune to hangups.
As stated earlier paramin can be used to optimize almost any function which can be defined within the C++ language. The Rosenbrock function is often used to test nonlinear optimization algorithms because it's quite challenging for methods which do not rely on the gradient of the function. A sample rosenbrock function and instruction on how to compile and run it should be available on the MRI website under paramin. This function can be used as a template for the optimization of any function definable in the C++ language, future work will be to add a parser and an R interface, so paramin can be called from within an R session using an R defined function.
Paramin was originally written with PVM, but since PVM is no longer supported it was migrated to MPI. More information on the migration process from PVM to MPI can be found on the MRI website, there one can also find study on the improvement in time relative to the number of processes spawned for each optimization routine.

13.2.2 Running paramin on multiple hosts/cluster

As stated above when we run paramin in the console it has to be invoked with the `mpirun` command. The `mpirun` command can take some optional arguments. First of all it can take the argument `-np <numproc>` where numproc is the desired number of processes one wants to start. This argument is **not** necessary with paramin because we define the number of subprocesses in the network file and paramin spawns them dynamically. The next optional argument we will discuss is the `--hostfile <filename>`. `<filename>` is a name of a file which holds the name of the hosts in the cluster and how many processors are available on each node. The authentication in the execution process is done via ssh, so public keys have to be shared to allow for remote login and execution between hosts. The programs desired to run in parallel have to be installed on each host for this to work. For more information see the FAQ entries on the OPEN-MPI webpage, there are more things that need to be taken into consideration. Here is an example hostfile from the open-mpi FAQ:

```
# This is an example hostfile.  Comments begin with #
#
# The following node is a single processor machine:
comp1

# The following node is a dual-processor machine:
comp2.dualcore slots=2

# The following node is a quad-processor machine, and we absolutely
# want to disallow over-subscribing it:
comp3.quadcore slots=4 max-slots=4
```

Chapter 14

References

Anon. 2002. Development of Structurally Detailed Statistically Testable Models of Marine Populations (dst^2). QLK5-CT1999-01609. Progress Report for 1 January to 31 December 2001. Marine Research Institute Report No. 87, Marine Research Institute, Reykjavik, Iceland.

Anon. 2003. Development of Structurally Detailed Statistically Testable Models of Marine Populations (dst^2). QLK5-CT1999-01609. Progress Report for 1 January to 31 December 2002. Marine Research Institute Report No. 98, Marine Research Institute, Reykjavik, Iceland.

Bertsekas, D.P. 1999. Nonlinear Programming. Athena Scientific 2^{nd} edition, pp22-61.

Bogstad, B., Hiis Hauge, K., and Ulltang, Ø. 1997. MULTSPEC - A Multispecies Model for Fish and Marine Mammals in the Barents Sea. Journal of Northwest Atlantic Fisheries Science. vol 22: pp317-341.

Bogstad, B., Tjelmeland, S., Tjelta, T. and Ulltang, Ø. 1992. Description of a Multispecies Model for the Barents Sea (MULTSPEC) and a Study of its Sensitivity to Assumptions on Food Preferences and Stock Sizes of Minke Whales and Harp Seals. Institute of Marine Research Technical Report SC/44/O 9, Institute of Marine Research, Bergen, Norway.

Dennis, J.E., and Schnabel, R.B. 1996. Numerical Methods for Unconstrained Optimisation and Nonlinear Equations. SIAM "Classics" edition.

Corana, A., Marchesi, M., Martini, C., and Ridella, S. 1987, Minimising Multimodal Functions of Continuous Variables with the 'Simulated Annealing' Algorithm. ACM Transactions on Mathematical Software vol 13: pp262-280.

Frøysa, K.G., Bogstad, B., and Skagen, D.W. 2002. Fleksibest - an Age-Length Structured Fish Stock Assessment Tool with Application to Northeast Arctic Cod (Gadus morhua L.). Fisheries Research vol 55: pp87-101.

Goffe, W.L., Ferrier, G.D., and Rogers, J. 1994. Global Optimisation of Statistical Functions with Simulated Annealing Journal of Econometrics. vol. 60: pp65-100.

Hooke, R., and Jeeves, T.A. 1961. Direct Search Solution of Numerical and Statistical Problems. ACM Journal. vol 8: pp212-229.

Jones, R., and Johnston, C. 1977. Growth, Reproduction and Mortality in Gadoid Fish Species. Fisheries Mathematics, Academic Press, pp37-62.

Kaupe, A.F. 1963. Algorithm 178: Direct Search. ACM Communications. vol 6: pp313-314.

Kennedy, J., and Eberhart, R. 1995. PSO optimization. Proceedings IEEE International Conference Neural Networks. Vol 4: pp1941-1948.

Stefánsson, G., and Pálsson, Ó.K. 1997. BORMICON. A Boreal Migration and Consumption Model. Marine Research Institute Report 58, Marine Research Institute, Reykjavik, Iceland.

Storn, R., and Price, K. 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization. Vol 11: pp341-359.

Tjelmeland, S., and Bogstad, B. 1998. MULTSPEC - A Review of a Multi-Species Modelling Project for the Barents Sea. Fisheries Research vol 37: pp127-142.

Appendix A

Order of Calculations

It is important to understand the order that Gadget performs the calculations, since this could have an effect on the way the user might structure the model. For each timestep of the simulation, the order of the calculations is:

1. **printing**. *Optional* printing of the model information at the beginning of the timestep, before any calculations have taken place for this timestep. Note that the default printing takes places towards the end of the timestep.
2. **migration**. Move the stocks around between the areas in the model, using migration matrices as described in section 4.11.
3. **consumption**. Calculate the predation by, and consumption of, the stocks in the model. Since a predator can consume more than one prey, and a prey can be consumed by more than one predator, the consumption calculations take place in 4 stages:
   (a) calculate the amount that each predator wants to consume of each prey, assuming that there is enough prey to meet these demands.
   (b) check that each prey is not "over consumed", to ensure that no more than 95% of the available prey biomass is consumed on a single timestep.
   (c) adjust the consumption of the predators to avoid overconsumption of the preys.
   (d) reduce the population of the preys according to the adjusted amount that the predators consumed.

   Note that the consumption calculations include the "consumption" of the stocks by any fleets in the model.
4. **natural mortality**. Reduce the population of the stocks in the model by removing a proportion due to natural mortality.
5. **growth**. Calculate any increase in length and weight of the stocks, and move any fish that will become mature (see section 4.12) into temporary storage.
6. **spawning**. Calculate the affect that spawning will have on the adult stocks, and place any spawned stock into temporary storage.
7. **maturation**. Add the newly matured fish (calculated as part of the growth update) into the model. Note that this comes after the mature fish have spawned, which means that a fish cannot become mature and spawn in the same timestep.
8. **recruitment**. Add any new recruits to the model, including both the recruits specified directly and the recruits specified as part of the spawning process. Note that this comes after the consumption calculations, so that new recruits cannot be consumed on the same timestep as they have been added into the model.

9. **straying**. Move fish between stocks in the model that are straying, as described in section 4.17.

10. **likelihood comparison**. Calculate the likelihood score for each of the individual likelihood components. Note that most of the likelihood components use data that is based on the catch by a fleet, which has been calculated as part of the consumption process.

11. **printing**. Default printing of the model information at the end of the timestep, after the calculations have taken place for this timestep.

12. **ageing**. If this is the last timestep of the year, increase the age of the fish in the model. As part of this process, the oldest age group (the plus group) can move to another stock (see section 4.13).

Appendix B

Recent Changes

A version of this user guide was printed and published as a Marine Research Institute report, available from the MRI, Reykjavik (Hafrannsóknastofnunin Fjölrit nr. 120). That published version of the user guide described the input file formats for Gadget version 2.1.00. Since the publication of that report, a number of changes to the Gadget software have been implemented, to fix any bugs that have been found and to add new features. Some of these changes have resulted in small changes to the format of the input files, and these user visible changes are summarised below.

B.1 Gadget version 2.1.01

- removed the need to specify the lengths of the fleets (see section 7)
- added new suitability functions, the Richards suitability function and the Gamma suitability function (see section 4.9)
- added a fleet that uses catch in numbers data (see section 7.2)
- added an "energycontent" section for the preys (see section 4.6 and section 6)

B.2 Gadget version 2.1.02

- simplified the format for the migration data (see section 4.11)
- added prey preference parameters to allow for a Type III functional response when one stock is consuming another (see section 4.8)

B.3 Gadget version 2.1.03

- changed the otherfood file format to specify the biomass of otherfood available, rather than the density of the otherfood (see section 6.1)
- added a fleet that extends the LinearFleet to take the different catchability of the various stocks into account (see section 7.4)
- added the StockPreyPrinter to the list of available printer classes (see section 9.8)
- added the PredatorPreyPrinter to the list of available printer classes (see section 9.9) to replace the old PredPreyStdAgePrinter and PredPreyStdLengthPrinter printer classes, which are no longer supported

B.4 Gadget version 2.1.04

- removed the need to specify the areas, predators or timesteps when calculating the score for the Understocking likelihood component (see section 8.2)

B.5 Gadget version 2.1.05

- added a fleet that implements a simple harvest control rule (see section 7.5)
- added a survey index likelihood component that can use an index based on acoustic data (see section 8.6.4)
- added a survey index likelihood component that can use an index based on effort data (see section 8.6.5)
- removed the need to specify the timesteps for the LikelihoodPrinter printer class, since the output is generated on the timesteps that the likelihood components have data for (see section 9.10)
- changed the ”half feeding value” format to specify the biomass of the prey, rather than the density of the prey (see section 4.8)

B.6 Gadget version 2.1.06

- this version only contains bug fixes, so there are no changes to the format of the input files

B.7 Gadget version 2.2.00

- removed the hard-coded factor of 4 from various maturation and suitability functions (see equations 4.27, 4.33 and 4.34) **note** that these functions have been re-named, so using one of these functions will currently generate an error to alert the user of this change
- extended the options for a fleet that implements a harvest control rule (see section 7.5)
  - –
- added an option to aggregate the StockDistribution data (see section 8.5)
- added the MigrationProportion likelihood component (see section 8.12)
- simplified the TimeVariable component (see section 3.4.2)
- added the StockVariable component so some model parameters can be based on the biomass of a stock (see section 3.4.3)
- added a penalty to models with NaN values to ensure that the optimiser chooses a valid minimum value