Forecasting on the Medium Term for advice using **FLasher**

13 September, 2017

This tutorial describes how Medium-Term Forecasts (MTF) can be performed using FLR. It uses the `FLasher` package for running projections, an updated version of `FLash`.

MTFs use the same engine as Short-Term Forecasts (STFs). However, there are some key differences between them.

- MTFs typically project over 5 to 10 years instead of the usual 3 years for a STF. Because of this increase in projection length it is necessary to include a stock-recruitment relationship to simulate the dynamics of the biological stock (an STF uses a constant recruitment assumption).
- MTFs may also have a more complicated projection control object because they can try to simulate management objectives (e.g. decreases in F over time).
- Finally, MTFs may also include consideration of uncertainty by including stochasticity in the projections.

Special attention must be paid to the conditioning and future assumptions of the stock.

Required packages

To follow this tutorial you should have installed the following packages:

- FLR: FLCore, FLasher, FLFishery

You can do so as follows,

```
install.packages(c("FLCore"), repos="http://flr-project.org/R")
install.packages(c("FLasher"), repos="http://flr-project.org/R")
install.packages(c("FLFishery"), repos="http://flr-project.org/R")
```

```
# Load all necessary packages, trim pkg messages
library(FLCore)
library(FLasher)
```

Introduction to Medium Term Forecasts

Running an MTF is similar to running an STF in that we need several components:

1. An `FLStock` object set up for the future (assumptions);
2. A stock-recruiment relationship (SRR);
3. A projection control object;

However, there are some significant differences between an MTF and an STF:

i. An MTF is usually run for 5 to 10 years (an STF for 3 years);
ii. An MTF can use different target types (e.g. setting catch targets, not just F targets);
iii. A dynamic SRR should be used (the STF assumption of mean recruitment is not a good one for a projection of more than 3 years);
iv. We can include uncertainty in the recruitment and target values.

In this tutorial we will build a 10 year projection, introduce a range of target types (including minimum and maximum target values, as well as relative target values), use a dynamic SRR and introduce uncertainty.

As ususal, we base the projections on plaice in the North Sea.

```r
data(ple4)
```

Conditioning the projection

The first step is to condition the projection by making assumptions about the stock in the future, and to fit the SRR.

Making the future stock

We use the `stf()` function to set up our stock into the future. 'stf()' makes a lot of assumptions to set up a future stock. We may want to change some of these assumptions, but for the moment we will use the defaults.

```r
ple4_mtf <- stf(ple4, nyears = 10)
# Now the stock goes up to 2018
summary(ple4_mtf)
```

```
An object of class "FLStock"

Name: Plaice in IV
Description: Imported from a VPA file. ( N:\Projecten\ICES WG\Demersale werkgroep  [...]
Quant: age
Dims:  age  year    unit    season  area    iter
    10  62  1   1   1   1

Range:  min max pgroup  minyear maxyear minfbar maxfbar
    1   10  10  1957    2018    2   6

catch         : [ 1 62 1 1 1 1 ], units =  t
catch.n       : [ 10 62 1 1 1 1 ], units =  10^3
catch.wt      : [ 10 62 1 1 1 1 ], units =  kg
discards      : [ 1 62 1 1 1 1 ], units =  t
discards.n    : [ 10 62 1 1 1 1 ], units =  10^3
discards.wt   : [ 10 62 1 1 1 1 ], units =  kg
landings      : [ 1 62 1 1 1 1 ], units =  t
landings.n    : [ 10 62 1 1 1 1 ], units =  10^3
landings.wt   : [ 10 62 1 1 1 1 ], units =  kg
stock         : [ 1 62 1 1 1 1 ], units =  t
stock.n       : [ 10 62 1 1 1 1 ], units =  10^3
stock.wt      : [ 10 62 1 1 1 1 ], units =  kg
m             : [ 10 62 1 1 1 1 ], units =  m
mat           : [ 10 62 1 1 1 1 ], units =
harvest       : [ 10 62 1 1 1 1 ], units =  f
harvest.spwn  : [ 10 62 1 1 1 1 ], units =
m.spwn        : [ 10 62 1 1 1 1 ], units =
```

The stock-recruitment relationship

In these examples we use a Beverton-Holt model (see the tutorial on fitting SRRs for more detail). The resulting SRR fit can be seen in Figure 1.

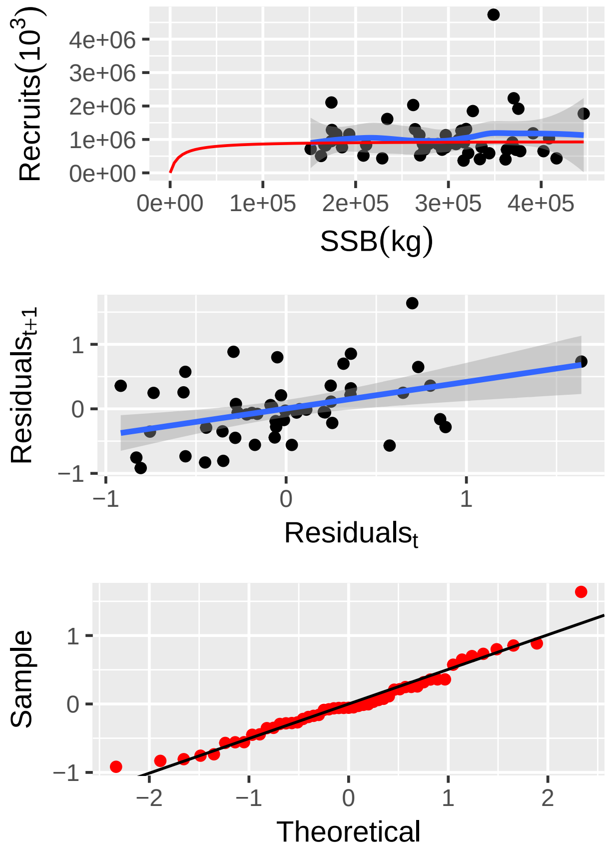


Figure 1: Fitted Beverton-Holt stock-recruitment relationship for the *ple4* stock object

```
ple4_sr <- fmle(as.FLSR(ple4, model="bevholt"), control=list(trace=0))
```

```
plot(ple4_sr)
```

Example 1: Fbar targets

We saw in the STF tutorial how to set an Fbar target (LINK). Here is some quick revision.

We will set the future F at F status quo and assume that F status quo is the mean of the last 4 years

```
f_status_quo <- mean(fbar(ple4)[,as.character(2005:2008)])
f_status_quo
```

```
[1] 0.4978
```

Make the control `data.frame` including all the years of the projection (note that `FLash` used *quantity* and *val* as column names and `FLasher` uses *quant* and *value*)

```
ctrl_target <- data.frame(year = 2009:2018,
                          quant = "f",
                          value = f_status_quo)
```

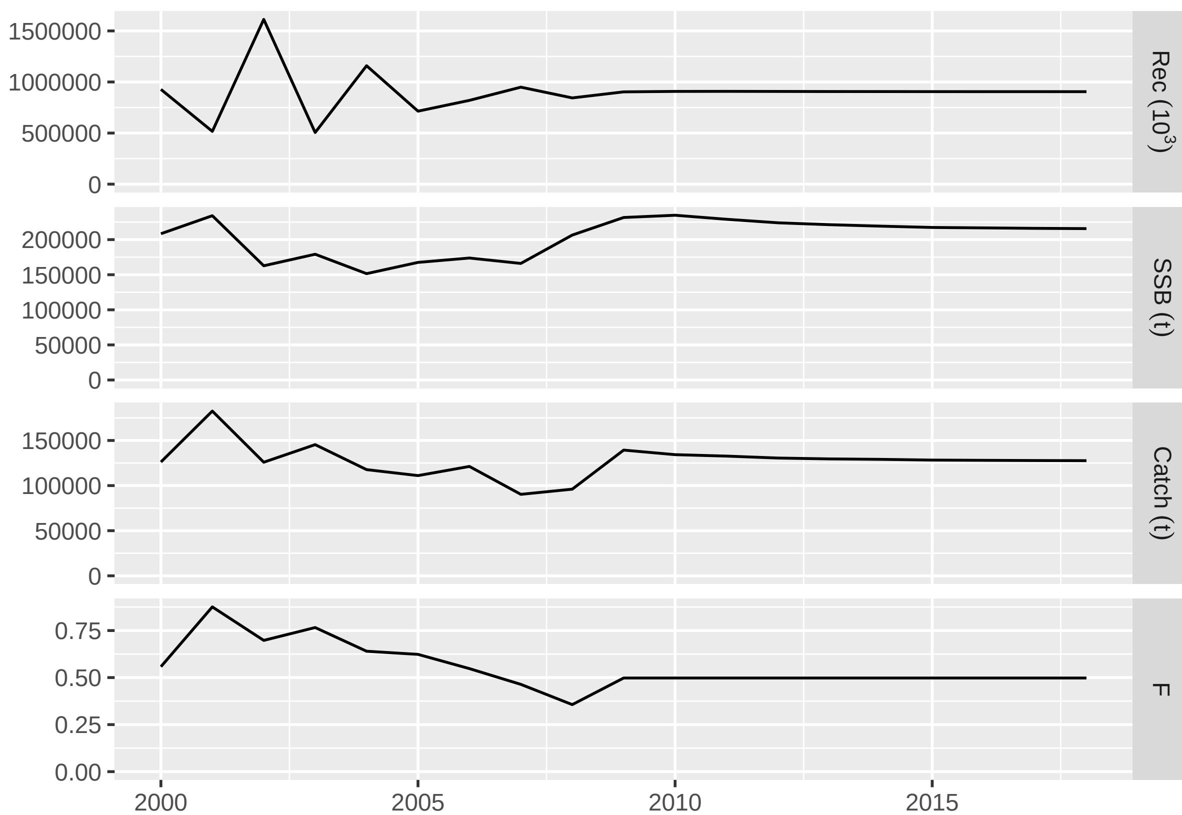
Make the `fwdControl` object from the control `data.frame`

```
ctrl_f <- fwdControl(ctrl_target)
ctrl_f
```

```
An object of class "fwdControl"
 (step) year quant min value max
      1 2009     f  NA 0.498  NA
      2 2010     f  NA 0.498  NA
      3 2011     f  NA 0.498  NA
      4 2012     f  NA 0.498  NA
      5 2013     f  NA 0.498  NA
      6 2014     f  NA 0.498  NA
      7 2015     f  NA 0.498  NA
      8 2016     f  NA 0.498  NA
      9 2017     f  NA 0.498  NA
     10 2018     f  NA 0.498  NA
```

We have columns of *year*, *quant* (target type), *min*, *value* and *max* (and others not necessarily shown). Here we are only using *year*, *quant* and *value*. We can now run fwd() with our three ingredients. Note that the *control* argument used to be called *ctrl* in `FLash`. Also, with `FLasher` the *control* and *sr* arguments must be named.

```
ple4_f_sq <- fwd(ple4_mtf, control = ctrl_f, sr = ple4_sr)
# What just happened? We plot the stock from the year 2000.
plot(window(ple4_f_sq, start=2000))
```



The future Fs are as we set in the control object

```
fbar(ple4_f_sq)[,ac(2005:2018)]
```

```
An object of class "FLQuant"
```

```
, , unit = unique, season = all, area = unique

     year
age   2005    2006    2007    2008    2009
  all 0.62343 0.54764 0.46392 0.35631 0.49783

      [ ...  4 years]

     year
age   2014    2015    2016    2017    2018
  all 0.49783 0.49783 0.49783 0.49783 0.49783
```

What about recruitment? Remember we are now using a Beverton-Holt model.

```
rec(ple4_f_sq)[,ac(2005:2018)]
```

```
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

   year
age 2005   2006   2007   2008   2009
  1 714344 820006 949341 844041 903372

      [ ...  4 years]

   year
age 2014   2015   2016   2017   2018
  1 906070 905709 905388 905275 905160
```

The recruitment is not constant but it is not changing very much. That's because the fitted model looks flat (Figure 1).

Example 2: A decreasing catch target

In this example we introduce two new things:

1. A new target type (catch)
2. A changing target value

Setting a catch target allows exploring the consequences of different TAC strategies. In this example, the TAC (the total catch of the stock) is reduced 10% each year for 10 years.

We create a vector of future catches based on the catch in 2008:

```
future_catch <- c(catch(ple4)[,"2008"]) * 0.9^(1:10)
future_catch
```

```
 [1] 86436 77793 70013 63012 56711 51040 45936 41342 37208 33487
```

We create the `fwdControl` object, setting the target quantity to *catch* and passing in the vector of future catches

```
ctrl_catch <- fwdControl(
    data.frame(
        year=2009:2018,
        quant = "catch",
        value=future_catch))
```

```r
# The control object has the desired catch target values
ctrl_catch
```

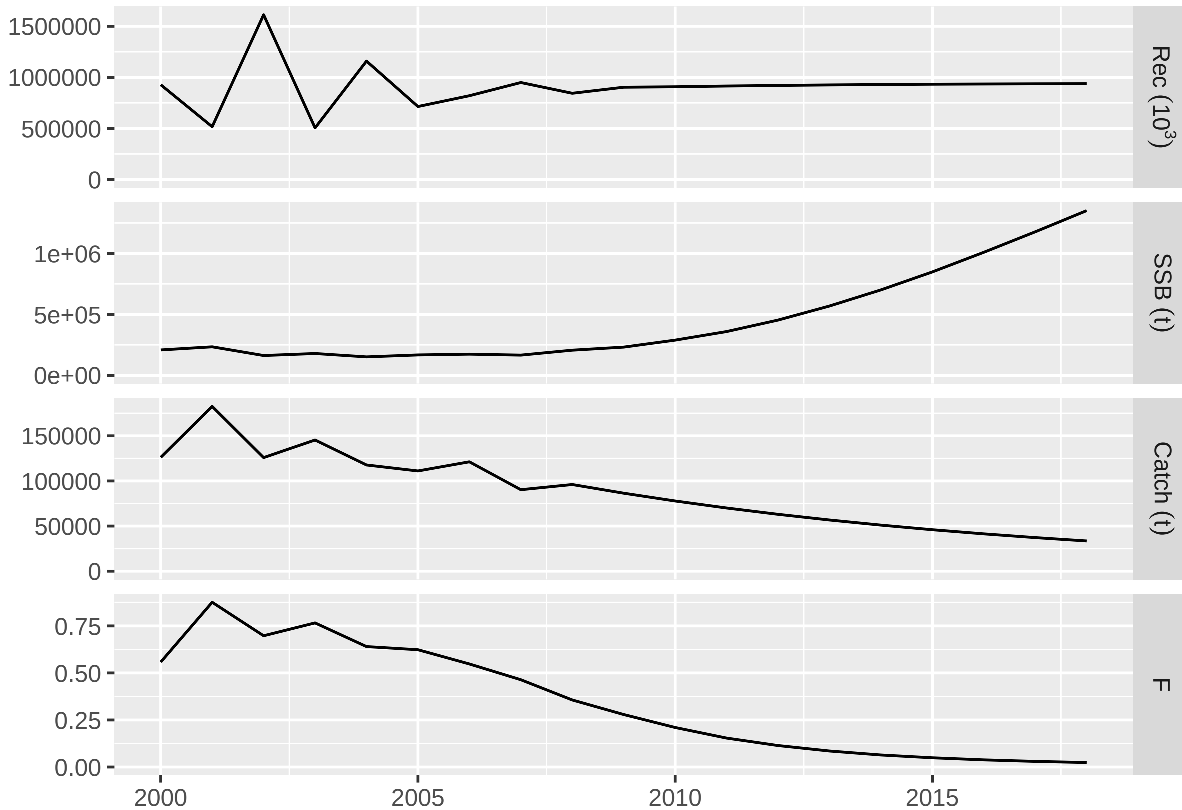
```
An object of class "fwdControl"
 (step) year quant min     value max
      1 2009 catch  NA 86436.404  NA
      2 2010 catch  NA 77792.764  NA
      3 2011 catch  NA 70013.487  NA
      4 2012 catch  NA 63012.139  NA
      5 2013 catch  NA 56710.925  NA
      6 2014 catch  NA 51039.832  NA
      7 2015 catch  NA 45935.849  NA
      8 2016 catch  NA 41342.264  NA
      9 2017 catch  NA 37208.038  NA
     10 2018 catch  NA 33487.234  NA
```

We call `fwd()` with the stock, the control object and the SRR, and look at the results

```r
ple4_catch <- fwd(ple4_mtf, control = ctrl_catch, sr = ple4_sr)
catch(ple4_catch)[,ac(2008:2018)]
```

```
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2008  2009  2010  2011  2012
  all 96040 86436 77793 70013 63012

      [ ...  1 years]

     year
age   2014  2015  2016  2017  2018
  all 51040 45936 41342 37208 33487
```

```r
plot(window(ple4_catch, start=2000))
```



The decreasing catch targets have been hit. Note that F has to be similarly reduced to hit the catch targets, resulting in a surge in SSB.

Example 3: Setting biological targets

In the previous examples we have set target types based on the activity of the fleet (F and catch). We can also set biological target types. This is useful when there are biological reference points, e.g. Bpa.

Setting a biological target must be done with care because it may not be possible to hit the target. For example, even when F is set to 0, the stock may not be productive enough to increase its abundance sufficiently to hit the target.

There are currently three types of biological target available in `FLasher`: *SRP*, *SSB* and *biomass*. Of these, there are several flavours of *SSB* and *biomass* that differ in terms of timing.

The *SRP* target is the Stock Recruitment Potential *at the time of spawning*, i.e. if a stock spawns in the middle of the year, after the abundance has been reduced by fishing and natural mortality, this is the SRP at that point in time. At the moment, SRP is calculated as the mass of mature fish. If setting an *SRP* target, you must be aware of the timing of spawning and the timing of the fishing period.

Internally, `FLasher` attempts to hit the desired target in a time step by finding the appropriate value of F in that timestep. If the stock spawns before fishing starts, then changing the fishing activity in that timestep has no effect on the SRP at the time of spawning. It is not possible to hit the target by manipulating F in that timestep and `FLasher` gives up.

SSB is the Spawning Stock Biomass calculated as the total biomass of mature fish. The *biomass* is simply the total biomass of the stock. For the *SSB* and *biomass* targets, there are three different flavours based on timing:

- *ssb_end* and *biomass_end* - at the end of the time step after all mortality (natural and fishing) has ceased;
- *ssb_spawn* and *biomass_spawn* - at the time of spawning (mimics the `ssb()` method for `FLStock` objects);
- *ssb_flash* and *biomass_flash* - an attempt to mimic the behaviour of the original `FLash` package.

This last bullet needs some explanation. If fishing starts before spawning (i.e. the *harvest.spwn* slot of an `FLStock` is greater than 0) then the SSB or biomass at the time of spawning in that timestep is returned. If fishing starts after spawning, or there is no spawning in that time step (which may happen with a seasonal model), then the SSB or biomass at the time of spawning in the next timestep is returned.

However, this second case can be more complicated for several reasons. If there is no spawning in the next time step then we have a problem and `FLasher` gives up (F in the current timestep does not affect the SSB or biomass at the time of spawning in the current or next timestep). Additionally, if there is no next time step (i.e. we have reached the end of the projection) then `FLasher` gives up.

There is also a potential problem that if the fishing in the next timestep starts before spawning, the SSB or biomass at the time of spawning in the next timestep will be affected by the effort in the current timestep AND the next timestep. `FLasher` cannot handle this and weird results will occur (although it is an unusal situation).

For these reasons, it is better to only use the `FLash`-like target for annual models and when fishing and spawning happen at the same time in each year through the projection.

Demonstrating the biological targets

Here we give simple demonstrations of the different types of biological targets using SSB. The results of using a biomass target will have the same behaviour. Only a 1 year projection is run.

The timing of spawning and fishing are controlled by the *m.spwn* and *harvest.spwn* slots. Our test `FLStock` object has *m.spwn* and *harvest.spwn* values of 0. This means that spawning and fishing happens at the start of the year and that spawning is assumed to happen before fishing.

Targets at the end of the timestep

Here we set a target SSB for the end of the timestep

```r
final_ssb <- 100000
ctrl_ssb <- fwdControl(data.frame(year=2009, quant = "ssb_end", value=final_ssb))
ple4_ssb <- fwd(ple4_mtf, control=ctrl_ssb, sr = ple4_sr)
# Calculate the final SSB to check the target has been hit
survivors <- stock.n(ple4_ssb) * exp(-harvest(ple4_ssb) - m(ple4_ssb))
quantSums((survivors * stock.wt(ple4_ssb) * mat(ple4_ssb))[,ac(2009)])
```

```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2009
  all 1e+05

units:  t
```

Targets at the time of spawning

If fishing occurs after spawning, the level of fishing will not affect the SSB or biomass at the time of spawning. This is currently the case because *m.spwn* and *harvest.spwn* have values of 0. The result is that the projection will fail with a warning (intentionally). We see this here.

```
spawn_ssb <- 100000
ctrl_ssb <- fwdControl(data.frame(year=2009, quant = "ssb_spawn", value=spawn_ssb))
ple4_ssb <- fwd(ple4_mtf, control=ctrl_ssb, sr = ple4_sr)
```

```
Warning in operatingModelRun(fishery, biolscpp, control,
effort_mult_initial = 1, : In operatingModel eval_om, ssb_spawn target.
Either spawning happens before fishing (so fishing effort has no impact on
SRP), or no spawning in timestep. Cannot solve.
```

```
# Using the `ssb()` method to get the SSB at the time of spawning,
# we can see that the projection failed
ssb(ple4_ssb)[,ac(2009)]
```

```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2009
  all 231522

units:  t
```

In our example, spawning happens at the start of the year. We can change this with the *m.spwn* slot. Natural mortality is assumed to happen continuously through the year. Therefore, if we set the *m.spwn* slot to 0.5, then half the natural mortality happens before spawning, i.e. spawning happens half way through the year. Similarly, the current value of *harvest.spwn* is 0, meaning that spawning happens before any fishing happens. If we set this value to 0.5 then half of the fishing mortality has occurred before spawning. With these changes, the example now runs.

```
m.spwn(ple4_mtf)[,ac(2009)] <- 0.5
harvest.spwn(ple4_mtf)[,ac(2009)] <- 0.5
spawn_ssb <- 100000
ctrl_ssb <- fwdControl(data.frame(year=2009, quant = "ssb_spawn", value=spawn_ssb))
ple4_ssb <- fwd(ple4_mtf, control=ctrl_ssb, sr = ple4_sr)
# We hit the target
ssb(ple4_ssb)[,ac(2009)]
```

```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2009
  all 1e+05

units:  t
```

At the moment `FLasher` calculates the SRP as SSB. This means that the *SRP* target type behaves in the same way as the *ssb_spawn* target.

```
srp <- 100000
ctrl_ssb <- fwdControl(data.frame(year=2009, quant = "srp", value=srp))
ple4_ssb <- fwd(ple4_mtf, control=ctrl_ssb, sr = ple4_sr)
# We hit the target
ssb(ple4_ssb)[,ac(2009)]
```

```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2009
  all 1e+05

units:  t
```

FLash-like targets

As mentioned above, the `FLash`-like targets can have different behaviour depending on the timing of spawning and fishing. If fishing starts before spawning, the SSB or biomass at the time of spawning *in the current timestep* will be hit (if possible). This is demonstrated here.

```
# Force spawning to happen half way through the year
# and fishing to start at the beginning of the year
m.spwn(ple4_mtf)[,ac(2009)] <- 0.5
harvest.spwn(ple4_mtf)[,ac(2009)] <- 0.5
flash_ssb <- 150000
ctrl_ssb <- fwdControl(data.frame(year=2009, quant = "ssb_flash", value=flash_ssb))
ple4_ssb <- fwd(ple4_mtf, control=ctrl_ssb, sr = ple4_sr)
# Hit the target? Yes
ssb(ple4_ssb)[,ac(2009)]
```

```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2009
  all 150000

units:  t
```

However, if fishing starts after spawning, the SSB or biomass at the time of spawning *in the next timestep* will be hit (if possible). This is because fishing in the current timestep will have no impact on the SSB at the time of spawning in the current timestep.

```
# Force spawning to happen at the start of the year before fishing
m.spwn(ple4_mtf)[,ac(2009)] <- 0.0
harvest.spwn(ple4_mtf)[,ac(2009)] <- 0.0
flash_ssb <- 150000
ctrl_ssb <- fwdControl(data.frame(year=2009, quant = "ssb_flash", value=flash_ssb))
ple4_ssb <- fwd(ple4_mtf, control=ctrl_ssb, sr = ple4_sr)
# We did hit the SSB target, but not until 2010.
ssb(ple4_ssb)[,ac(2009:2010)]
```

```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2009   2010
  all 231522 150000

units:  t
```

A longer SSB projection

Here we run a longer projection with a constant `FLash`-like SSB target. Spawning happens before fishing so the target will not be hit until the following year.

```r
# Force spawning to happen at the start of the year before fishing
m.spwn(ple4_mtf)[,ac(2009)] <- 0.0
harvest.spwn(ple4_mtf)[,ac(2009)] <- 0.0
future_ssb <- 200000
ctrl_ssb <- fwdControl(data.frame(year=2009:2018, quant = "ssb_flash", value=future_ssb))
ple4_ssb <- fwd(ple4_mtf, control = ctrl_ssb, sr = ple4_sr)
```

We get a warning about running out of room. This is because future stock object, *ple4_mtf*, goes up to 2018. When we set the SSB target for 2018, it tries to hit the final year target in 2019. The targets that were set for 2009 to 2017 have been hit in 2010 to 2018. However, we cannot hit the target that was set for 2018. This means that the returned value of F in 2018 needs to be discounted.

```r
ssb(ple4_ssb)[,ac(2009:2018)]
```

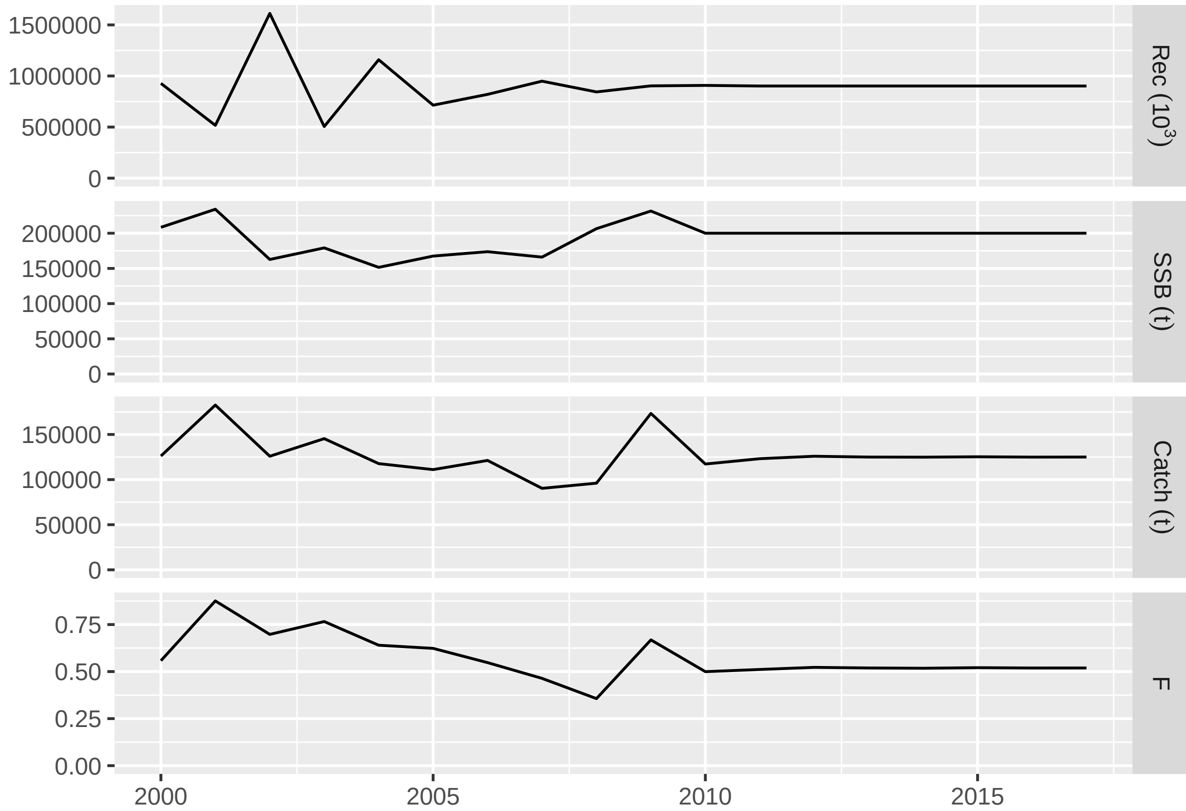
```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2009   2010   2011   2012   2013   2014   2015   2016   2017
  all 231522 200000 200000 200000 200000 200000 200000 200000 200000
     year
age   2018
  all 200000

units:  t
```

```r
fbar(ple4_ssb)[,ac(2009:2018)]
```

```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2009    2010    2011    2012    2013    2014    2015    2016
  all 0.66844 0.49965 0.51120 0.52245 0.51927 0.51772 0.52075 0.51900
     year
age   2017    2018
  all 0.51921 0.45596
```

```
units:  f
```

```r
plot(window(ple4_ssb, start=2000, end=2017))
```



Example 4: Relative catch target

The examples above have dealt with *absolute* target values. We now introduce the idea of *relative* values. This allows us to set the target value *relative* to the value in another time step.

We do this by using the *relYear* column in the control object (the year that the target is relative to). The *value* column now holds the relative value, not the absolute value.

Here we set catches in the projection years to be 90% of the catches in the previous year, i.e. we want the catche in 2009 to be 0.9 * value in 2008 etc.

```r
ctrl_rel_catch <- fwdControl(
    data.frame(year = 2009:2018,
           quant = "catch",
           value = 0.9,
           relYear = 2008:2017))
# The relative year appears in the control object summary
ctrl_rel_catch
```

```
An object of class "fwdControl"
 (step) year quant relYear min value max
      1 2009 catch    2008  NA 0.900  NA
      2 2010 catch    2009  NA 0.900  NA
      3 2011 catch    2010  NA 0.900  NA
      4 2012 catch    2011  NA 0.900  NA
```

```
    5 2013 catch    2012  NA 0.900  NA
    6 2014 catch    2013  NA 0.900  NA
    7 2015 catch    2014  NA 0.900  NA
    8 2016 catch    2015  NA 0.900  NA
    9 2017 catch    2016  NA 0.900  NA
   10 2018 catch    2017  NA 0.900  NA
```

We run the projection as normal

```
ple4_rel_catch <- fwd(ple4_mtf, control = ctrl_rel_catch, sr = ple4_sr)
catch(ple4_rel_catch)
```

```
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   1957   1958   1959   1960   1961  
  all  78423  88240 109238 117138 118331

      [ ...  52 years]

     year
age   2014  2015  2016  2017  2018 
  all 51040 45936 41342 37208 33487
```

```
catch(ple4_rel_catch)[,ac(2008:2018)] / catch(ple4_rel_catch)[,ac(2007:2017)]
```

```
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2008   2009   2010   2011   2012  
  all 1.0638 0.9000 0.9000 0.9000 0.9000

      [ ...  1 years]

     year
age   2014 2015 2016 2017 2018
  all 0.9  0.9  0.9  0.9  0.9 
```

```
plot(window(ple4_rel_catch, start = 2001, end = 2018))
```

This is equivalent to the catch example above (LINK) but without using absolute values.

Example 5: Minimum and Maximum targets

In this Example we introduce two new things:

1. Multiple target types;
2. Targets with *bounds*.

Here we set an F target so that the future F = F0.1. However, we also don't want the catch to fall below a minimum level. We do this by setting a *minimum* value for the catch.

First we set a value for F0.1 (you could use the `FLBRP` package to do this (LINK))

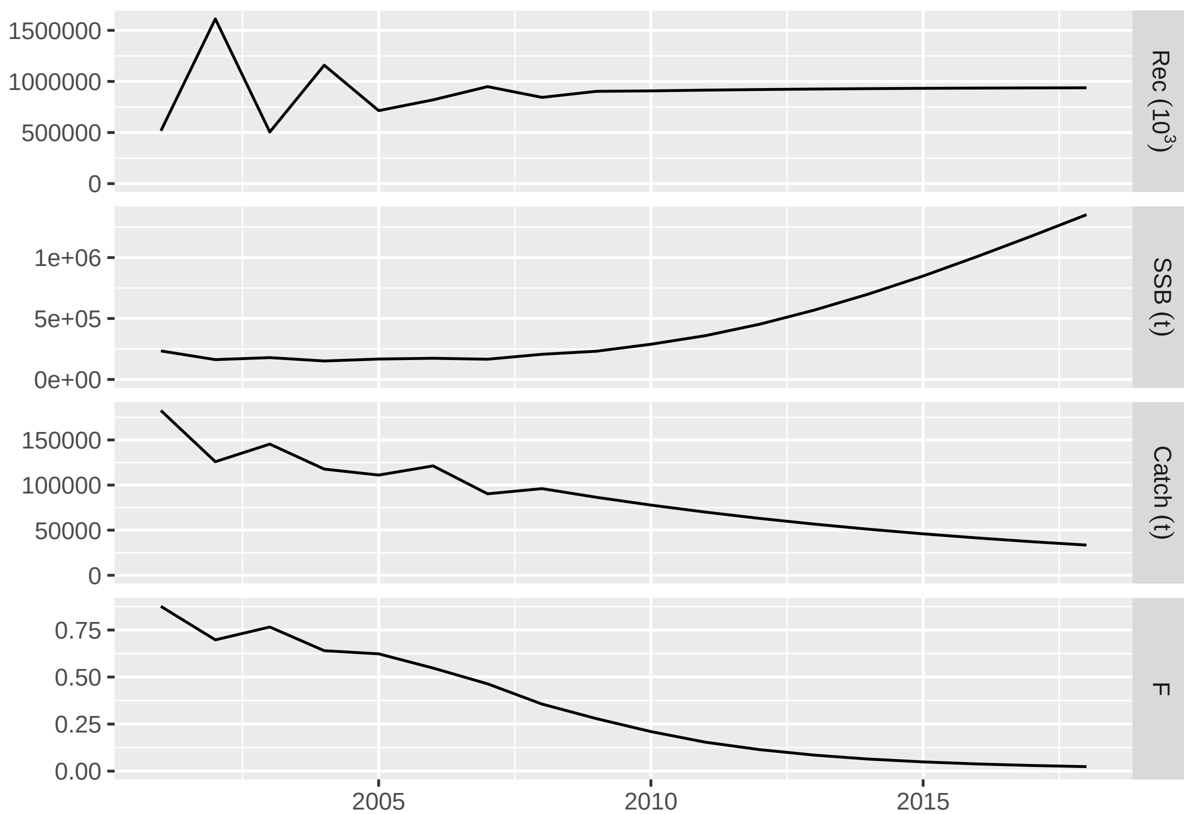


Figure 2: Relative catch example

```
f01 <- 0.1
```

We'll set our minimum catch to be the mean catch of the last 3 years.

```
min_catch <- mean(catch(ple4_mtf)[,as.character(2006:2008)])
min_catch
```

```
[1] 102510
```

To create the control object, we make a `data.frame` with both target types. Note that we include a *min* column.

```
df <- data.frame(
    year = rep(2009:2018, each=2),
    quant = c("f","catch"),
    value = c(f01, NA),
    min = c(NA, min_catch))
```

It is also important that when running the projection, the bounding targets (the *min* and the *max*) are processed after the non-bounding targets. This should be sorted out by the `fwdControl` constructor.

Make the control object

```
ctrl_min_catch <- fwdControl(df)
ctrl_min_catch
```

```
An object of class "fwdControl"
 (step) year quant        min value max
      1 2009     f         NA 0.100  NA
      2 2009 catch 102509.578    NA  NA
      3 2010     f         NA 0.100  NA
      4 2010 catch 102509.578    NA  NA
      5 2011     f         NA 0.100  NA
      6 2011 catch 102509.578    NA  NA
      7 2012     f         NA 0.100  NA
      8 2012 catch 102509.578    NA  NA
      9 2013     f         NA 0.100  NA
     10 2013 catch 102509.578    NA  NA
     11 2014     f         NA 0.100  NA
     12 2014 catch 102509.578    NA  NA
     13 2015     f         NA 0.100  NA
     14 2015 catch 102509.578    NA  NA
     15 2016     f         NA 0.100  NA
     16 2016 catch 102509.578    NA  NA
     17 2017     f         NA 0.100  NA
     18 2017 catch 102509.578    NA  NA
     19 2018     f         NA 0.100  NA
     20 2018 catch 102509.578    NA  NA
```

What did we create? We can see that the *min* column has now got some data (the *max* column is still empty) and the targets appear in the correct order. Now project forward

```
ple4_min_catch <- fwd(ple4_mtf, control = ctrl_min_catch, sr = ple4_sr)
fbar(ple4_min_catch)[,ac(2008:2018)]
```

```
An object of class "FLQuant"
, , unit = unique, season = all, area = unique
```

```
     year
age   2008    2009    2010    2011    2012
  all 0.35631 0.34056 0.30514 0.26839 0.23888

      [ ...  1 years]

     year
age   2014    2015    2016    2017    2018
  all 0.18829 0.16898 0.15188 0.13752 0.12523
```

```
catch(ple4_min_catch)[,ac(2008:2018)]
```

```
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2008   2009   2010   2011   2012
  all  96040 102510 102510 102510 102510

      [ ...  1 years]

     year
age   2014   2015   2016   2017   2018
  all 102510 102510 102510 102510 102510
```

What happens? The catch constraint is hit in every year of the projection. The projected F decreases but never hits the target F because the minimum catch constraint prevents it from dropping further.

```
plot(window(ple4_min_catch, start = 2001, end = 2018))
```

It is possible to also set a maximum constraint, for example, to prevent F from being too large.

Example 6 - Relative targets and bounds

In this example we use a combination of *relative* targets and *bounds*.

This kind of approach can be used to model a recovery plan. For example, we want to decrease F to F0.1 by 2015 (absolute target value) but catches cannot change by more than 15% each year (relative bound). This requires careful setting up of the control object.

We make a vector of the desired F targets using the F0.1 we calculated above. We set up an F sequence that decreases from the current Fbar in 2008 to F01 in 2015, then F01 until 2018.

```
current_fbar <- c(fbar(ple4)[,"2008"])
f_target <- c(seq(from = current_fbar, to = f01, length = 8)[-1], rep(f01, 3))
f_target
```

```
 [1] 0.3197 0.2831 0.2465 0.2098 0.1732 0.1366 0.1000 0.1000 0.1000 0.1000
```

We set maximum annual change in catch to be 10% (in either direction).

```
rel_catch_bound <- 0.10
```

We make the control `data.frame` with the F target and the catch target. Note the use of the *relYear*, *min* and *max* columns in the dataframe.

```
df <- data.frame(
    year = rep(2009:2018, 2),
```

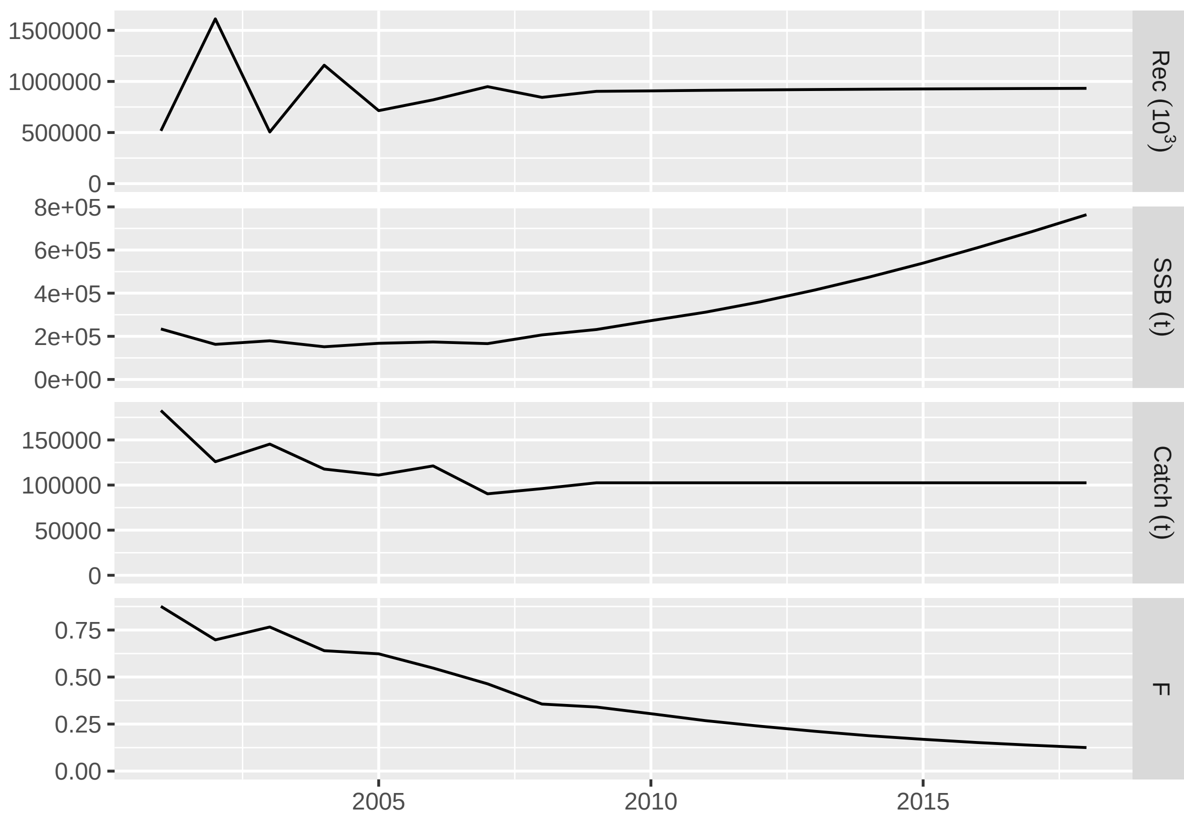


Figure 3: Example with a minimum catch bound and constant F target

```r
    relYear =c(rep(NA,10), 2008:2017),
    quant = c(rep("f",10), rep("catch",10)),
    value = c(f_target, rep(NA,10)),
    max = c(rep(NA,10), rep(1+rel_catch_bound, 10)),
    min = c(rep(NA,10), rep(1-rel_catch_bound, 10)))
```

Make the control object. The *min* and *max* columns now both have data

```r
ctrl_rel_min_max_catch <- fwdControl(df)
ctrl_rel_min_max_catch
```

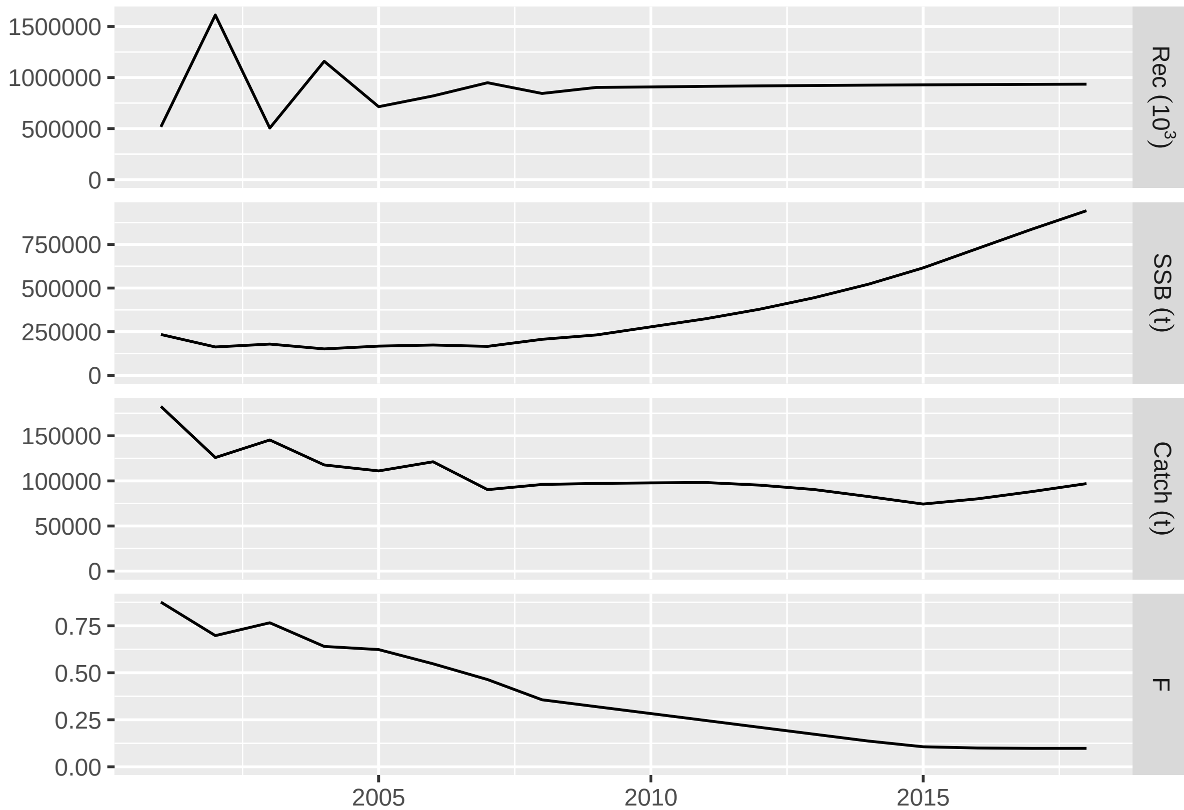
```
An object of class "fwdControl"
 (step) year quant relYear   min value   max
      1 2009     f      NA    NA 0.320    NA
      2 2009 catch    2008 0.900    NA 1.100
      3 2010     f      NA    NA 0.283    NA
      4 2010 catch    2009 0.900    NA 1.100
      5 2011     f      NA    NA 0.246    NA
      6 2011 catch    2010 0.900    NA 1.100
      7 2012     f      NA    NA 0.210    NA
      8 2012 catch    2011 0.900    NA 1.100
      9 2013     f      NA    NA 0.173    NA
     10 2013 catch    2012 0.900    NA 1.100
     11 2014     f      NA    NA 0.137    NA
     12 2014 catch    2013 0.900    NA 1.100
     13 2015     f      NA    NA 0.100    NA
     14 2015 catch    2014 0.900    NA 1.100
     15 2016     f      NA    NA 0.100    NA
     16 2016 catch    2015 0.900    NA 1.100
     17 2017     f      NA    NA 0.100    NA
     18 2017 catch    2016 0.900    NA 1.100
     19 2018     f      NA    NA 0.100    NA
     20 2018 catch    2017 0.900    NA 1.100
```

Run the projection:

```r
recovery<-fwd(ple4_mtf, control=ctrl_rel_min_max_catch, sr=ple4_sr)
```

What happened? The F decreased and then remains constant, while the catch has changed by only a limited amount each year.

```r
plot(window(recovery, start = 2001, end = 2018))
```



The minimum and maximum bounds on the catch are operational in several of the years. They prevent the catch from increasing as well as decreasing too strongly, (allegedly) providing stability to the fishery.

```
catch(recovery)[,ac(2009:2018)] / catch(recovery)[,ac(2008:2017)]
```

```
An object of class "FLQuant"
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

     year
age   2009    2010    2011    2012    2013    2014    2015    2016
  all 1.01188 1.00637 1.00410 0.97043 0.94868 0.91367 0.90000 1.07802
     year
age   2017    2018
  all 1.10000 1.10000

units:
```

Projections with stochasticity

So far we have looked at combinations of:

- Absolute target values;
- Relative target values;
- Bounds on targets, and
- Mixed target types.

But all of the projections have been deterministic, i.e. they all had only one iteration. Now, we are going start looking at projecting with multiple iterations. This is important because it can help us understand the impact of uncertainty (e.g. in the stock-recruitment relationship).

fwd() is happy to work over iterations. It treats each iteration separately. "All" you need to do is set the arguments correctly.

There are two main ways of introducing iterations into fwd():

1. By passing in residuals to the stock-recruitment function (as another argument to *fwd()*);
2. Through the control object (by setting target values as multiple values)

You can actually use both of these methods at the same time. As you can probably imagine, this can quickly become very complicated so we'll just do some simple examples to start with.

Preparation for projecting with iterations

To perform a stochastic projection you need a stock object with multiple iterations. If you are using the output of a stock assessment method, such as *a4a*, then you may have one already. Here we use the *propagate()* method to expand the ple4 stock object to have 200 iterations. We'll use the ten year projection as before (remember that we probably should change the assumptions that come with the *stf()* method).

```
niters <- 200
ple4_mtf <- stf(ple4, nyears = 10)
ple4_mtf <- propagate(ple4_mtf, niters)
```

You can see that the 6th dimension, iterations, now has length 200:

```
summary(ple4_mtf)
```

```
An object of class "FLStock"

Name: Plaice in IV
Description: Imported from a VPA file. ( N:\Projecten\ICES WG\Demersale werkgroep  [...]
Quant: age
Dims:  age  year    unit    season  area    iter
    10  62  1   1   1   200

Range:  min max pgroup  minyear maxyear minfbar maxfbar
    1   10  10  1957    2018    2   6

catch         : [ 1 62 1 1 1 200 ], units =  t
catch.n       : [ 10 62 1 1 1 200 ], units =  10^3
catch.wt      : [ 10 62 1 1 1 200 ], units =  kg
discards      : [ 1 62 1 1 1 200 ], units =  t
discards.n    : [ 10 62 1 1 1 200 ], units =  10^3
discards.wt   : [ 10 62 1 1 1 200 ], units =  kg
landings      : [ 1 62 1 1 1 200 ], units =  t
landings.n    : [ 10 62 1 1 1 200 ], units =  10^3
landings.wt   : [ 10 62 1 1 1 200 ], units =  kg
stock         : [ 1 62 1 1 1 200 ], units =  t
stock.n       : [ 10 62 1 1 1 200 ], units =  10^3
stock.wt      : [ 10 62 1 1 1 200 ], units =  kg
m             : [ 10 62 1 1 1 200 ], units =  m
mat           : [ 10 62 1 1 1 200 ], units =
harvest       : [ 10 62 1 1 1 200 ], units =  f
harvest.spwn  : [ 10 62 1 1 1 200 ], units =
```

```
m.spwn        : [ 10 62 1 1 1 200 ], units =
```

Example 7: Stochastic recruitment

There is an argument to *fwd()* that we haven't used yet: *residuals*

This is used for specifying the recruitment residuals (*residuals*) which are multiplicative. In this example we'll use the residuals so that the predicted recruitment values in the projection = deterministic recruitment predicted by the SRR model * residuals. The residuals are passed in as an **FLQuant** with years and iterations. Here we make an empty **FLQuant** that will be filled with residuals.

```
rec_residuals <- FLQuant(NA, dimnames = list(year=2009:2018, iter=1:niters))
```

We're going to use residuals from the stock-recruitment relationship we fitted at the beginning. We can access these using:

```
residuals(ple4_sr)
```

```
An object of class "FLQuant"
, , unit = unique, season = all, area = unique

   year
age 1958      1959      1960      1961      1962
  1 -0.268830 -0.058033 -0.190040 -0.063352 -0.443513

      [ ...  41 years]

   year
age 2004      2005      2006      2007      2008
  1  0.255971 -0.218722 -0.086510  0.057988 -0.057139
```

These residuals are on a log scale i.e. log_residuals = log(observed_recruitment) - log(predicted_recruitment). To use these log residuals multiplicatively we need to transform them with *exp()*:

We want to fill up our *multi_rec_residuals* **FLQuant** by randomly sampling from these log residuals. We can do this with the *sample()* function. We want to sample with replacement (i.e. if a residual is chosen, it gets put back in the pool and can be chosen again).

First we get generate the samples of the years (indices of the residuals we will pick).

```
sample_years <- sample(dimnames(residuals(ple4_sr))$year, niters * 10, replace = TRUE)
```

We fill up the **FLQuant** we made earlier with the residuals using the sampled years:

```
rec_residuals[] <- exp(residuals(ple4_sr)[,sample_years])
```

What have we got?

```
rec_residuals
```

```
An object of class "FLQuant"
An object of class "FLQuant"
iters:  200

, , unit = unique, season = all, area = unique

     year
quant 2009            2010            2011            2012
  all 0.92550(0.426) 0.94446(0.358) 0.96235(0.475) 0.94446(0.454)
```

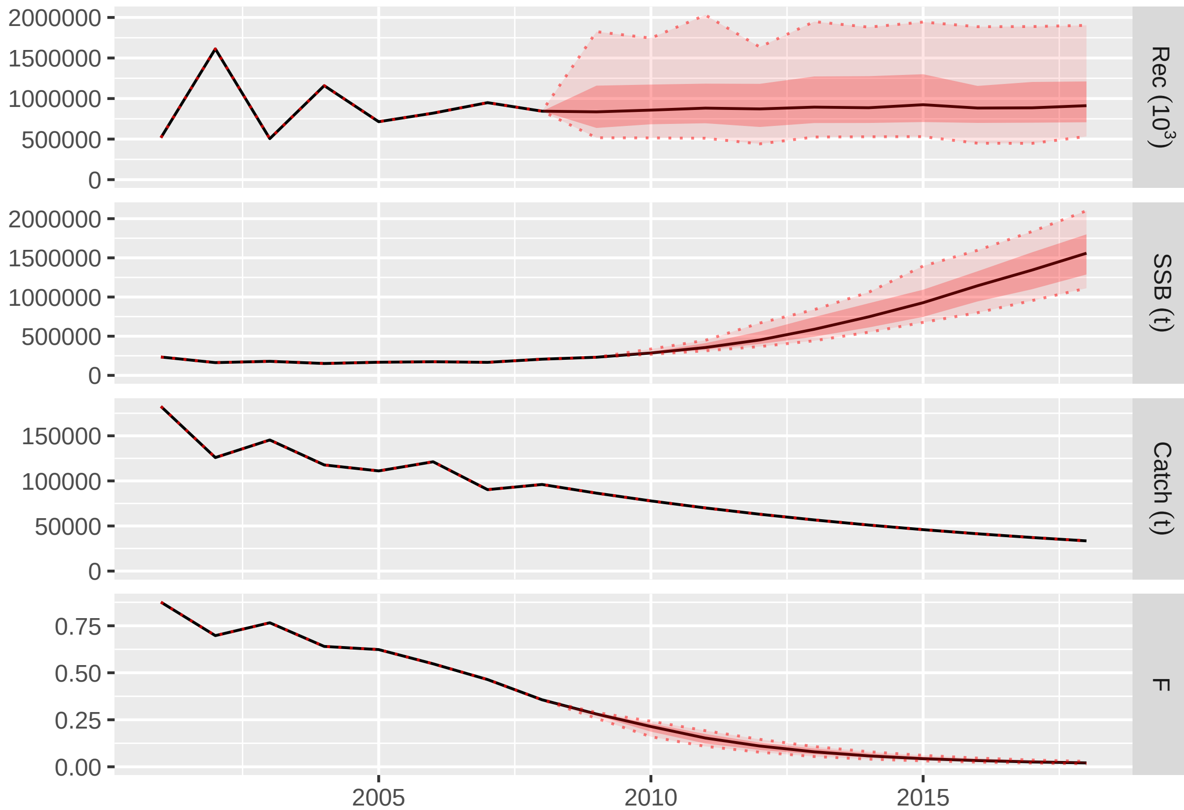


Figure 4: Example projection with stochasticity in the recruitment residuals

```
      year
quant 2013           2014           2015           2016
  all 0.96235(0.407) 0.95227(0.428) 0.99172(0.428) 0.94362(0.355)
      year
quant 2017           2018
  all 0.94446(0.358) 0.97243(0.398)

units:  NA
```

It's an **FLQuant** of SRR residuals but what do those brackets mean? The information in the brackets is the Median Absolute Deviation, a way of summarising the iterations. We have 200 iterations but don't want to see all of them - just a summary.

We now have the recruitment residuals. We'll use the *ctrl_catch* control object we made earlier with decreasing catch. We call *fwd()* as usual, only now we have a *residuals* argument. This takes a little time (we have 200 iterations).

```
ple4_stoch_rec <- fwd(ple4_mtf, control = ctrl_catch, sr = ple4_sr, residuals = rec_residuals)
```

What just happened? We can see that now we have uncertainty in the recruitment estimates, driven by the residuals. This uncertainty feeds into the SSB and, to a lesser extent, the projected F and catch.

```
plot(window(ple4_stoch_rec, start = 2001, end = 2018))
```

We can see that the projected stock metrics also have uncertainty in them.

```
rec(ple4_stoch_rec)[,ac(2008:2018)]
```

```
An object of class "FLQuant"
iters:  200

, , unit = unique, season = all, area = unique

   year
age 2008           2009           2010           2011
  1 844041(     0) 836074(385110) 857335(325390) 881970(432963)
   year
age 2012
  1 872054(418206)

      [ ...  1 years]

   year
age 2014           2015           2016           2017
  1 844041(     0) 836074(385110) 857335(325390) 881970(432963)
   year
age 2018
  1 872054(418206)
```

```r
fbar(ple4_stoch_rec)[,ac(2008:2018)]
```

```
An object of class "FLQuant"
iters:  200

, , unit = unique, season = all, area = unique

     year
age   2008            2009            2010            2011
  all 0.35631(0.0000) 0.28059(0.0100) 0.21405(0.0300) 0.15290(0.0373)
     year
age   2012
  all 0.11049(0.0297)

      [ ...  1 years]

     year
age   2014            2015            2016            2017
  all 0.35631(0.0000) 0.28059(0.0100) 0.21405(0.0300) 0.15290(0.0373)
     year
age   2018
  all 0.11049(0.0297)
```

```r
ssb(ple4_stoch_rec)[,ac(2008:2018)]
```

```
An object of class "FLQuant"
iters:  200

, , unit = unique, season = all, area = unique

     year
age   2008          2009          2010          2011
  all 206480(    0) 231522(    0) 285544(19908) 355350(45295)
     year
```

```
age   2012
  all 451938(98260)

      [ ...  1 years]

     year
age   2014           2015           2016           2017
  all 206480(    0) 231522(    0) 285544(19908) 355350(45295)
     year
age   2018
  all 451938(98260)
```

Example 8: stochastic target values

In this example we introduce uncertainty by including uncertainty in our target values. This example has catch as the target, except now catch will be stochastic.

We will use the ctrl_catch object from above (we make a copy):

```r
ctrl_catch
```

```
An object of class "fwdControl"
 (step) year quant min     value max
      1 2009 catch  NA 86436.404  NA
      2 2010 catch  NA 77792.764  NA
      3 2011 catch  NA 70013.487  NA
      4 2012 catch  NA 63012.139  NA
      5 2013 catch  NA 56710.925  NA
      6 2014 catch  NA 51039.832  NA
      7 2015 catch  NA 45935.849  NA
      8 2016 catch  NA 41342.264  NA
      9 2017 catch  NA 37208.038  NA
     10 2018 catch  NA 33487.234  NA
```

```r
ctrl_catch_iters <- ctrl_catch
```

Let's take a look at what else is in the control object:

```r
slotNames(ctrl_catch_iters)
```

```
[1] "target" "iters"  "FCB"
```

The iterations of the target value are set in the *iters* slot.

```r
ctrl_catch_iters@iters
```

```
, , iter = 1

    val
row  min value max
  1   NA 86436  NA
  2   NA 77793  NA
  3   NA 70013  NA
  4   NA 63012  NA
  5   NA 56711  NA
  6   NA 51040  NA
  7   NA 45936  NA
```

```
  8   NA 41342  NA
  9   NA 37208  NA
  10  NA 33487  NA
```

What is this slot?

```
class(ctrl_catch_iters@iters)
```

```
[1] "array"
```

```
dim(ctrl_catch_iters@iters)
```

```
[1] 10  3  1
```

It's a 3D array with structure: target no x value x iteration. It's in here that we set the stochastic projection values. Each row of the *iters* slot corresponds to a row in the control **data.frame** we passed in.

Here we set 10 targets (one for each year in the projection), so the first dimension of *iters* has length 10. The second dimension always has length 3 (for *min*, *value* and *max* columns). The third dimension is where the iterations are stored. This is currently length 1. We have 200 iterations and therefore we need to expand *iters* along the iter dimension so it can store the 200 iterations.

One way of doing this is to make a new array with the right dimensions. Note that we need to put in dimnames.

```
new_iters <- array(NA, dim=c(10,3,niters), dimnames = list(1:10, c("min","value","max"),iter=1:niters))
dim(new_iters)
```

```
[1]  10   3 200
```

Now we can fill it up with new data (our stochastic catch targets).

We need to generate random catch target data. This could come from a number of sources (e.g. MSY estimated with uncertainty). In this example we make it very simple, by using lognormal distribution with a fixed standard deviation of 0.3. We multiply the deterministic catch target values by samples from this distribution.

```
future_catch_iters <- ctrl_catch_iters@iters[,"value",] * rlnorm(10 * niters, meanlog = 0, sdlog=0.3)
```

We fill up *iters* with these values. We just fill up the *value* column (you can also set the *min* and *max* columns to set stochastic bounds).

```
new_iters[,"value",] <- future_catch_iters
```

We put our new *iters* into the control object:

```
ctrl_catch_iters@iters <- new_iters
```

We can see that now we have stochasticity in the target values.

```
ctrl_catch_iters
```

```
An object of class "fwdControl"
 (step) year quant min                value max
      1 2009 catch  NA 84109.290(23820.449)  NA
      2 2010 catch  NA 80533.342(23428.470)  NA
      3 2011 catch  NA 70844.549(20788.435)  NA
      4 2012 catch  NA 61861.350(18906.882)  NA
      5 2013 catch  NA 56576.740(17997.790)  NA
      6 2014 catch  NA 50966.141(13603.201)  NA
      7 2015 catch  NA 48688.132(14718.497)  NA
      8 2016 catch  NA 41822.384(12024.879)  NA
```

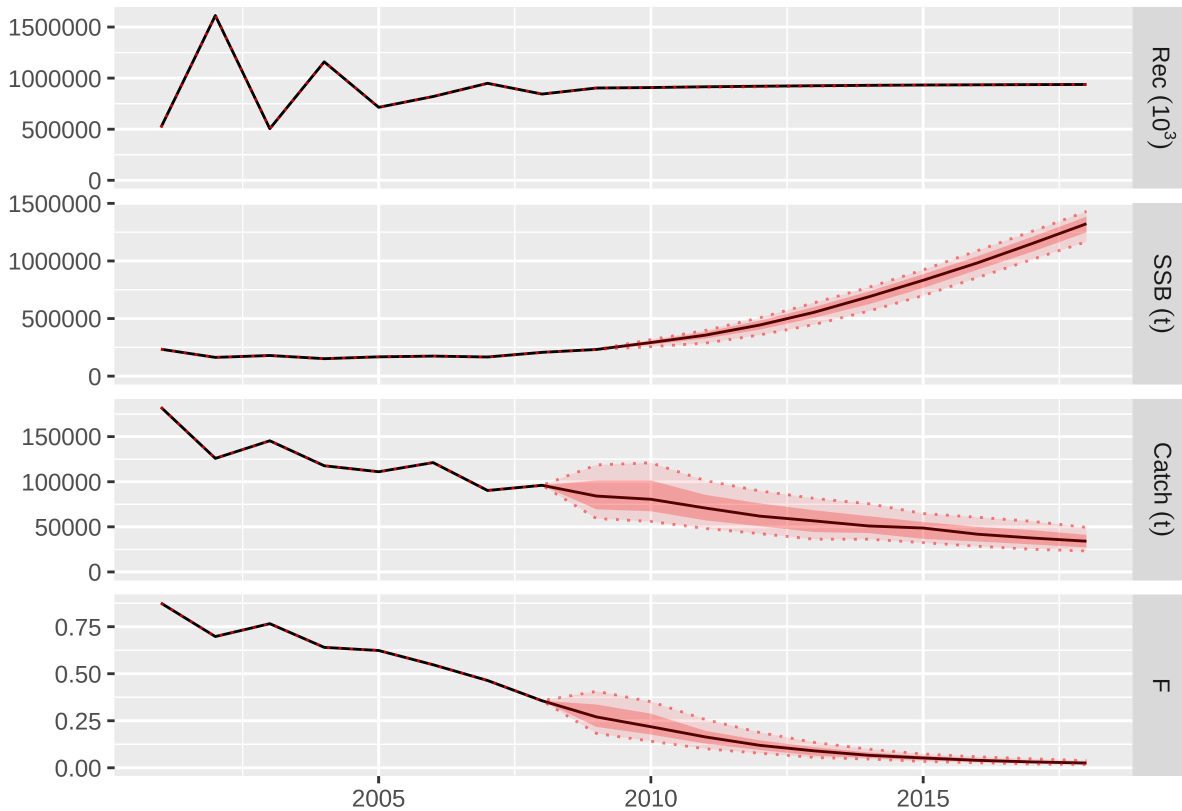
```
 9 2017 catch  NA 37644.407(11618.056)  NA
10 2018 catch  NA 34090.572(10278.505)  NA
iters:  200
```

We project as normal using the deterministic SRR.

```
ple4_catch_iters <- fwd(ple4_mtf, control=ctrl_catch_iters, sr = ple4_sr)
```

What happened?

```
plot(window(ple4_catch_iters, start = 2001, end = 2018))
```



The projected catches reflect the uncertainty in the target.

```
catch(ple4_catch_iters)[,ac(2008:2018)]
```

```
An object of class "FLQuant"
iters:  200

, , unit = unique, season = all, area = unique

     year
age   2008          2009          2010          2011          2012
  all 96040(    0) 84109(23820) 80533(23428) 70845(20788) 61861(18907)

      [ ...  1 years]

     year
age   2014          2015          2016          2017          2018
  all 96040(    0) 84109(23820) 80533(23428) 70845(20788) 61861(18907)
```

Example 9: A projection with stochastic catch and recruiment

What is going on with recruitment in the results of the previous example?

```
rec(ple4_catch_iters)[,ac(2008:2018)]
```

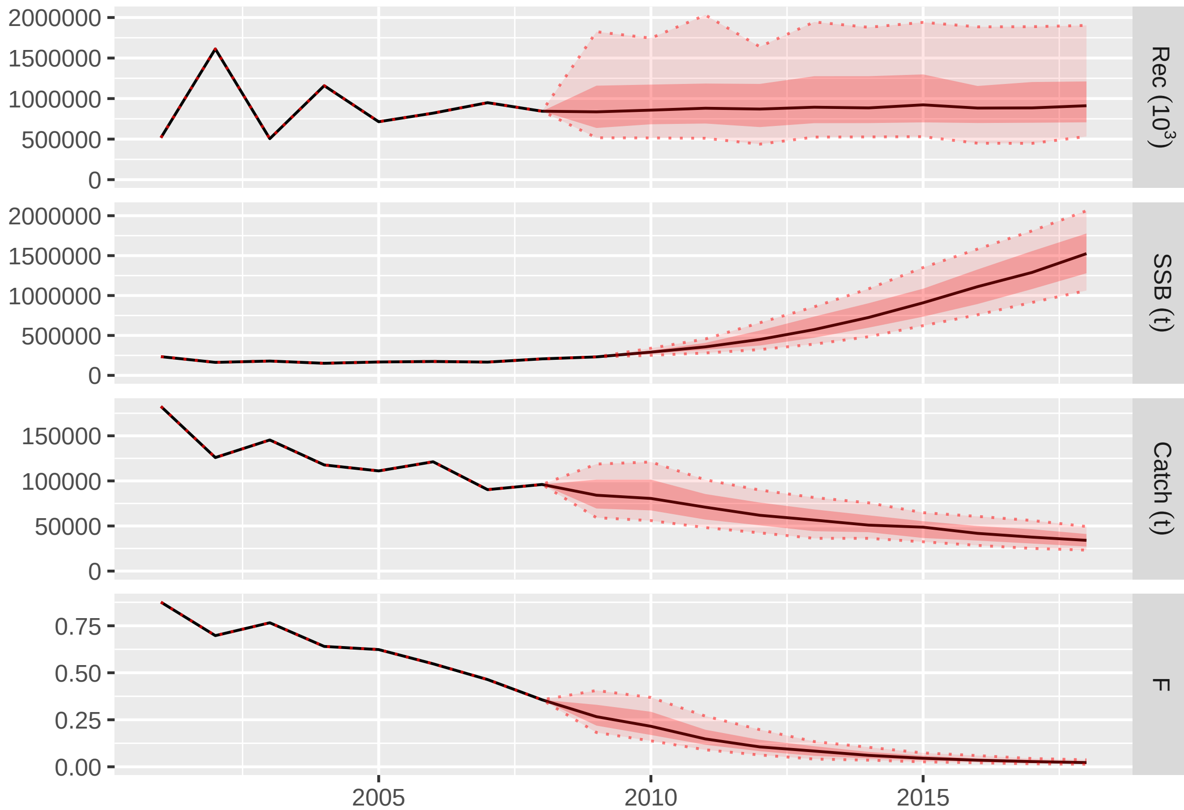
```
An object of class "FLQuant"
iters:  200

, , unit = unique, season = all, area = unique

   year
age 2008          2009          2010          2011          2012
  1 844041(   0)  903372(   0)  907749(   0)  915265(2469)  920593(2761)

      [ ...  1 years]

   year
age 2014          2015          2016          2017          2018
  1 844041(   0)  903372(   0)  907749(   0)  915265(2469)  920593(2761)
```

Remember that here recruitment is not being driven by random residuals, it is only be driven by SSB. The recruitment in year Y is a result of the SSB in year Y-1. The SSB in year Y-1 is a result of the catch in year Y-2. So if catch is stochastic in 2009, we don't see the impact of the stochasticity on the recruitment until 2011. Even then the impact is small. This seems unlikely so we can also put in recruitment residuals (we already made them for Example 7).

```
ple4_catch_iters <- fwd(ple4_mtf, control=ctrl_catch_iters, sr = ple4_sr, residuals = rec_residuals)
```

What happened?

```
plot(window(ple4_catch_iters, start = 2001, end = 2018))
```



We have a projection with stochastic target catches and recruitment.

```
catch(ple4_catch_iters)[,ac(2008:2018)]
```

```
An object of class "FLQuant"
iters:  200

, , unit = unique, season = all, area = unique

     year
age   2008          2009          2010          2011          2012         
  all 96040(    0) 84109(23820) 80533(23428) 70845(20788) 61861(18907)

      [ ...  1 years]

     year
age   2014          2015          2016          2017          2018         
  all 96040(    0) 84109(23820) 80533(23428) 70845(20788) 61861(18907)
```

```
rec(ple4_catch_iters)[,ac(2008:2018)]
```

```
An object of class "FLQuant"
iters:  200

, , unit = unique, season = all, area = unique

   year
age 2008           2009           2010           2011          
  1 844041(     0) 836074(385110) 857335(325390) 880963(433424)
```

```
   year
age 2012
  1 870721(416835)

      [ ...  1 years]

   year
age 2014           2015           2016           2017
  1 844041(     0) 836074(385110) 857335(325390) 880963(433424)
   year
age 2018
  1 870721(416835)
```

TO DO

Alternative syntax for controlling the projection

SOMETHING ON CALLING FWD() AND SPECIFYING TARGETS AS ARGUMENTS

Notes on conditioning projections

SOMETHING ON FWD WINDOW

References

More information

- You can submit bug reports, questions or suggestions on this tutorial at https://github.com/flr/doc/issues.
- Or send a pull request to https://github.com/flr/doc/
- For more information on the FLR Project for Quantitative Fisheries Science in R, visit the FLR webpage, http://flr-project.org.

Software Versions

- R version 3.4.1 (2017-06-30)
- FLCore: 2.6.5
- FLasher: 0.0.2.9008
- **Compiled**: Wed Sep 13 16:37:44 2017

License

Author information

Finlay Scott. European Commission, DG Joint Research Centre, Directorate D - Sustainable Resources, Unit D.02 Water and Marine Resources, Via E. Fermi 2749, 21027 Ispra VA, Italy. https://ec.europa.eu/jrc/