Handbook for the Stochastic Production model in Continuous Time (SPiCT)

Martin W. Pedersen, Alexandros Kokkalis, Tobias K. Mildenberger, Casper W. Berg

20 February, 2021

Basic functionality

Getting started

This vignette explains basic and more advanced functions of the `spict` package. The package is installed from gihtub using the `devtools` package:

```
devtools::install_github("DTUAqua/spict/spict")
```

installs the stable version of `spict`. When loading the package the user is greeted and the the installed version is shown:

```
library(spict)
```

The printed version follows the format ver@SHA, where *ver* is the spict version number and *SHA* is a unique github commit on github. The content of this vignette pertains to the version printed above that can be found here.

A specific version of `spict` can be installed using the following:

```
devtools::install_github("DTUAqua/spict/spict", ref = "v1.3.4")
```

Loading built-in example data

The package contains the catch and index data analysed in Polacheck, Hilborn, and Punt (1993) that can be loaded using

```
data(pol)
```

Data on three stocks are contained in this dataset: South Atlantic albacore, northern Namibian hake, and New Zealand rock lobster. Here focus will be on the South Atlantic albacore data. This dataset contains the following

```
pol$albacore
   $obsC
    [1] 15.9 25.7 28.5 23.7 25.0 33.3 28.2 19.7 17.5 19.3 21.6 23.1 22.5 22.5 23.6
   [16] 29.1 14.4 13.2 28.4 34.6 37.5 25.9 25.3

   $timeC
    [1] 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981
   [16] 1982 1983 1984 1985 1986 1987 1988 1989

   $obsI
    [1] 61.89 78.98 55.59 44.61 56.89 38.27 33.84 36.13 41.95 36.63 36.33 38.82
```

```
 [13] 34.32 37.64 34.01 32.16 26.88 36.61 30.07 30.75 23.36 22.36 21.91

$timeI
 [1] 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981
[16] 1982 1983 1984 1985 1986 1987 1988 1989
```

Note that data are structured as a list containing the entries `obsC` (catch observations), `timeC` (time of catch observations), `obsI` (index observations), and `timeI` (time of index observations). If times are not specified it is assumed that the first observation is observed at time 1 and then sequentially onward with a time step of one year. It is therefore recommended to always specify observation times.

Each catch observation relates to a time interval. This is specified using `dtc`. If `dtc` is left unspecified (as is the case here) each catch observation is assumed to cover the time interval until the next catch observation. For this example with annual catches `dtc` therefore is

```
inp <- check.inp(pol$albacore)
inp$dtc
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

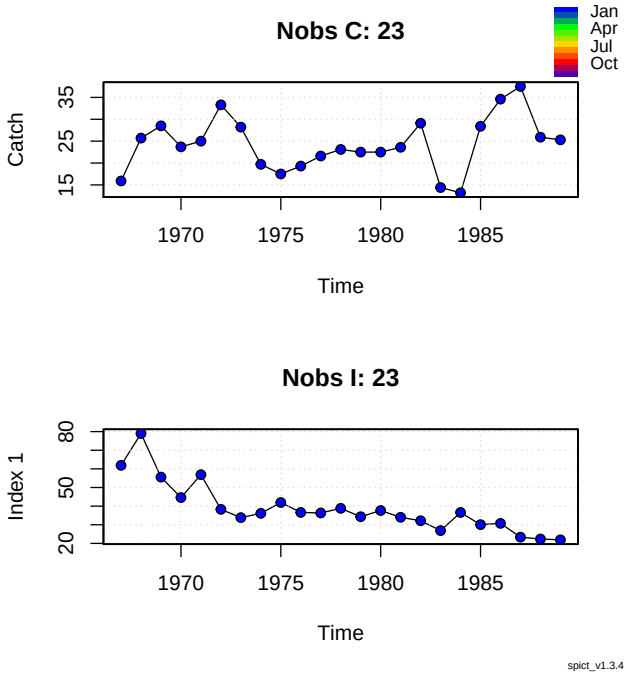
It is important to specify `dtc` if the default assumption is not fulfilled.

Note that the index observations for all data sets analysed in Polacheck, Hilborn, and Punt (1993) represent commercial CPUE data and the respective times (`inp$timeI`) should thus correspond to the middle of the year, e.g. 1988.50, rather than the beginning of the year, e.g. 1988.00. The 'incorrect' times of the commercial CPUE observations were kept only for reasons of consistency regarding previous studies using models which were not implemented in 'continuous time'. We emphasise the importance of correct times of the index observations, which should most accurately correspond to the timing of the survey or to the middle of the year in case of commercial CPUE.

Plotting data

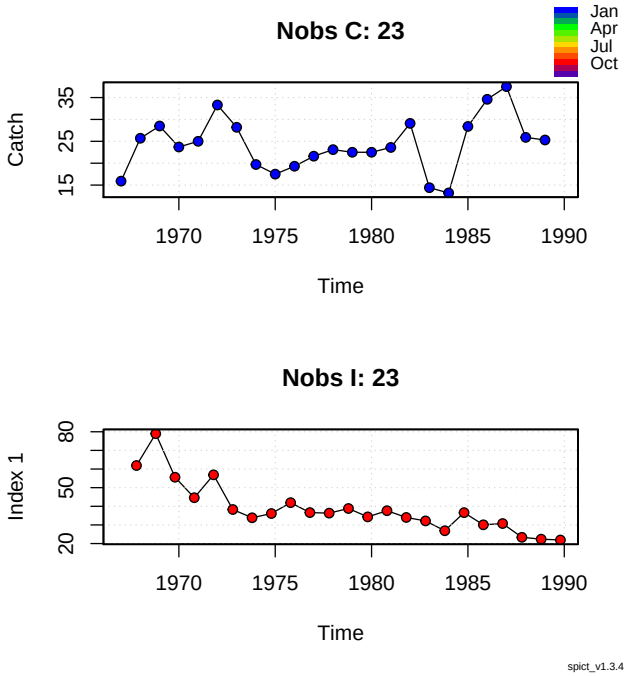
The data can be plotted using the command

```
plotspict.data(pol$albacore)
```



Note that the number of catch and index observations are given in the respective plot headers. Furthermore, the color of individual points shows when the observation was made and the corresponding colors are shown in the color legend in the top right corner. For illustrative purposes let's try shifting the biomass index data a bit

```
inpshift <- pol$albacore
inpshift$timeC <- inpshift$timeC
inpshift$timeI <- inpshift$timeI + 0.8
plotspict.data(inpshift)
```

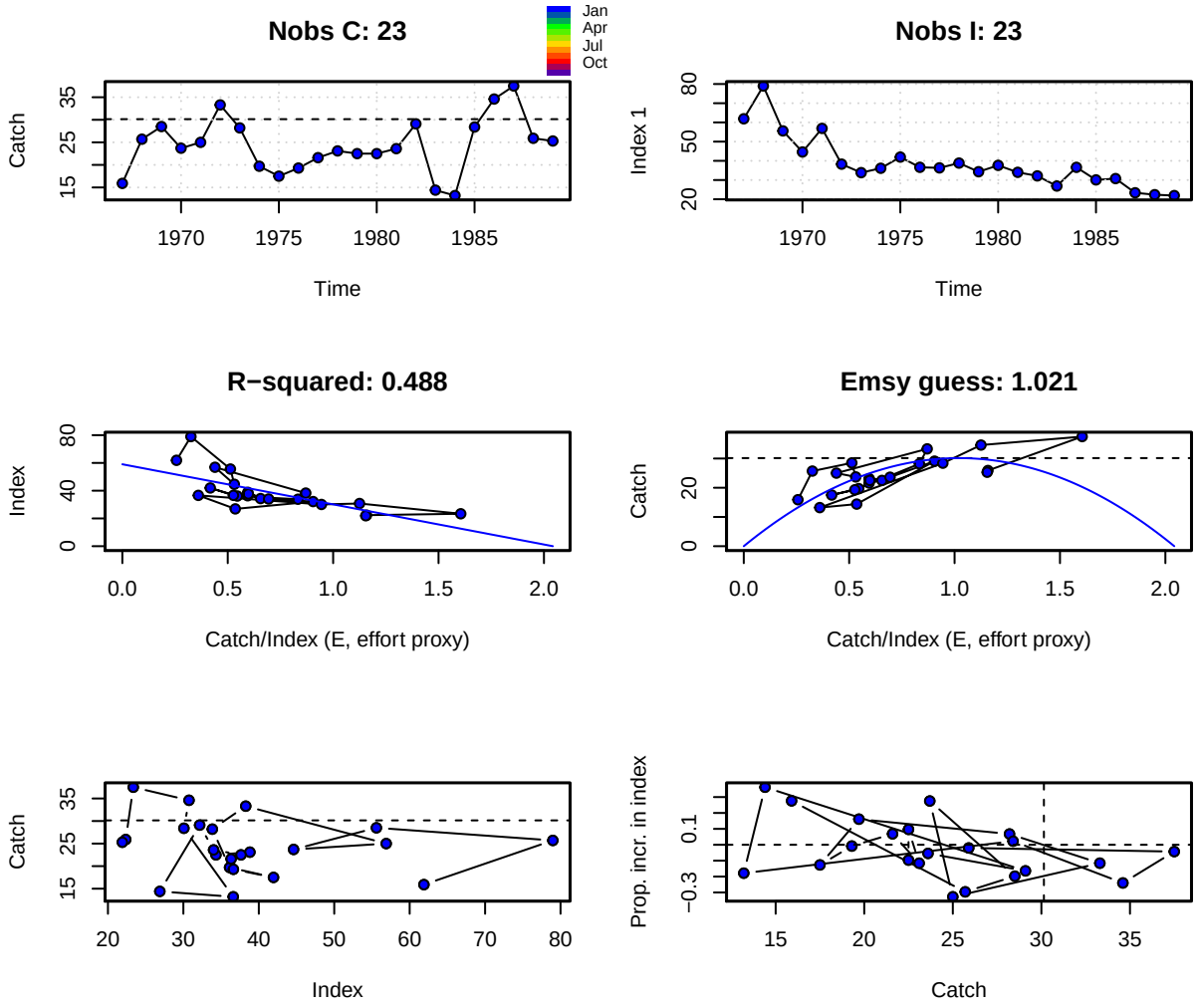


Now the colours show that the index takes place in autumn.

Advanced data plotting

There is also a more advanced function for plotting data, which at the same time does some basic model fitting (linear regression) and shows the results

```
plotspict.ci(pol$albacore)
```



The two top plots come from `plotspict.data`, with the dashed horizontal line representing a guess of MSY. This guess comes from a linear regression between the index and the catch divided by the index (middle row, left). This regression is expected to have a negative slope. A similar plot can be made showing catch versus catch/index (middle row, right) to approximately find the optimal effort (or effort proxy). The proportional increase in the index as a function of catch (bottom row, right) should show primarily positive increases in index at low catches and vice versa. Positive increases in index at large catches could indicate model violations. In the current plot these are not seen.

Fitting the model

The model is fitted to data by running

```
res <- fit.spict(pol$albacore)
```

Here the call to `fit.spict` is wrapped in the `system.time` command to check the time spent on the calculations. This is obviously not required, but done here to show that fitting the model only takes a few seconds. The result of the model fit is stored in `res`, which can either be plotted using `plot` or summarised using `summary`.

The results are returned as a list that contains output as well as input. The content of this list is

```
names(res)
    [1] "value"          "sd"             "cov"            "par.fixed"
    [5] "cov.fixed"      "pdHess"         "gradient.fixed" "par.random"
```

```
 [9] "diag.cov.random" "env"             "inp"             "obj"
[13] "opt"             "pl"              "Cp"              "report"
[17] "computing.time"
```

Many of these variables are generated by `TMB::sdreport()`. In addition to these `spict` includes the list of input values (`inp`), the object used for fitting (`obj`), the result from the optimiser (`opt`), the time spent on fitting the model (`computing.time`), and more less useful variables.

Interpreting summary of results

The results are summarised using

```
summary(res)
```

```
 [1] "Convergence: 0  MSG: relative convergence (4)"
 [2] "Objective function at optimum: 2.0654937"
 [3] "Euler time step (years):  1/16 or 0.0625"
 [4] "Nobs C: 23,  Nobs I1: 23"
 [5] ""
 [6] "Priors"
 [7] "     logn  ~  dnorm[log(2), 2^2]"
 [8] " logalpha  ~  dnorm[log(1), 2^2]"
 [9] "  logbeta  ~  dnorm[log(1), 2^2]"
[10] ""
[11] "Model parameter estimates w 95% CI "
[12] "           estimate       cilow       ciupp    log.est  "
[13] " alpha    8.5380744   1.2232675  59.5934380  2.1445355  "
[14] " beta     0.1212598   0.0180690   0.8137643 -2.1098201  "
[15] " r        0.2556012   0.1010590   0.6464730 -1.3641371  "
[16] " rc       0.7435384   0.1445713   3.8240608 -0.2963349  "
[17] " rold     0.8179929   0.0019101 350.3063985 -0.2009016  "
[18] " m       22.5827814  17.0681833  29.8791035  3.1171877  "
[19] " K      201.4753831 138.1193695 293.8931023  5.3056672  "
[20] " q        0.3512552   0.1942693   0.6350990 -1.0462423  "
[21] " n        0.6875264   0.0636693   7.4241810 -0.3746550  "
[22] " sdb      0.0128136   0.0018407   0.0892017 -4.3572450  "
[23] " sdf      0.3673757   0.2673605   0.5048051 -1.0013701  "
[24] " sdi      0.1094038   0.0808973   0.1479555 -2.2127095  "
[25] " sdc      0.0445479   0.0073370   0.2704793 -3.1111903  "
[26] " "
[27] "Deterministic reference points (Drp)"
[28] "         estimate      cilow     ciupp    log.est  "
[29] " Bmsyd 60.7440893 15.4030075 239.55350  4.1066698  "
[30] " Fmsyd  0.3717692  0.0722856   1.91203 -0.9894821  "
[31] " MSYd  22.5827814 17.0681833  29.87910  3.1171877  "
[32] "Stochastic reference points (Srp)"
[33] "         estimate      cilow      ciupp    log.est  rel.diff.Drp  "
[34] " Bmsys 60.7364389 15.4031663 239.490696  4.1065438 -0.0001259603  "
[35] " Fmsys  0.3717814  0.0722787   1.912338 -0.9894493  0.0000328184  "
[36] " MSYs  22.5806757 17.0626482  29.883223  3.1170945 -0.0000932523  "
[37] ""
[38] "States w 95% CI (inp$msytype: s)"
[39] "                  estimate      cilow       ciupp    log.est  "
[40] " B_1989.94      56.6970766 30.1355966 106.6698142  4.0377227  "
[41] " F_1989.94       0.4464502  0.2145050   0.9291989 -0.8064275  "
```

```
[42] " B_1989.94/Bmsy  0.9334936  0.2961492   2.9424701 -0.0688212  "
[43] " F_1989.94/Fmsy  1.2008407  0.2864262   5.0345195  0.1830219  "
[44] ""
[45] "Predictions w 95% CI (inp$msytype: s)"
[46] "                prediction      cilow      ciupp    log.est  "
[47] " B_1991.00      54.3059071 27.8527120 105.883102  3.9946330  "
[48] " F_1991.00       0.4464504  0.1573033   1.267094 -0.8064269  "
[49] " B_1991.00/Bmsy  0.8941240  0.2498558   3.199677 -0.1119108  "
[50] " F_1991.00/Fmsy  1.2008413  0.2390582   6.032087  0.1830224  "
[51] " Catch_1990.00  24.7359945 15.3328367  39.905820  3.2082595  "
[52] " E(B_inf)       49.9856474         NA         NA  3.9117359  "
```

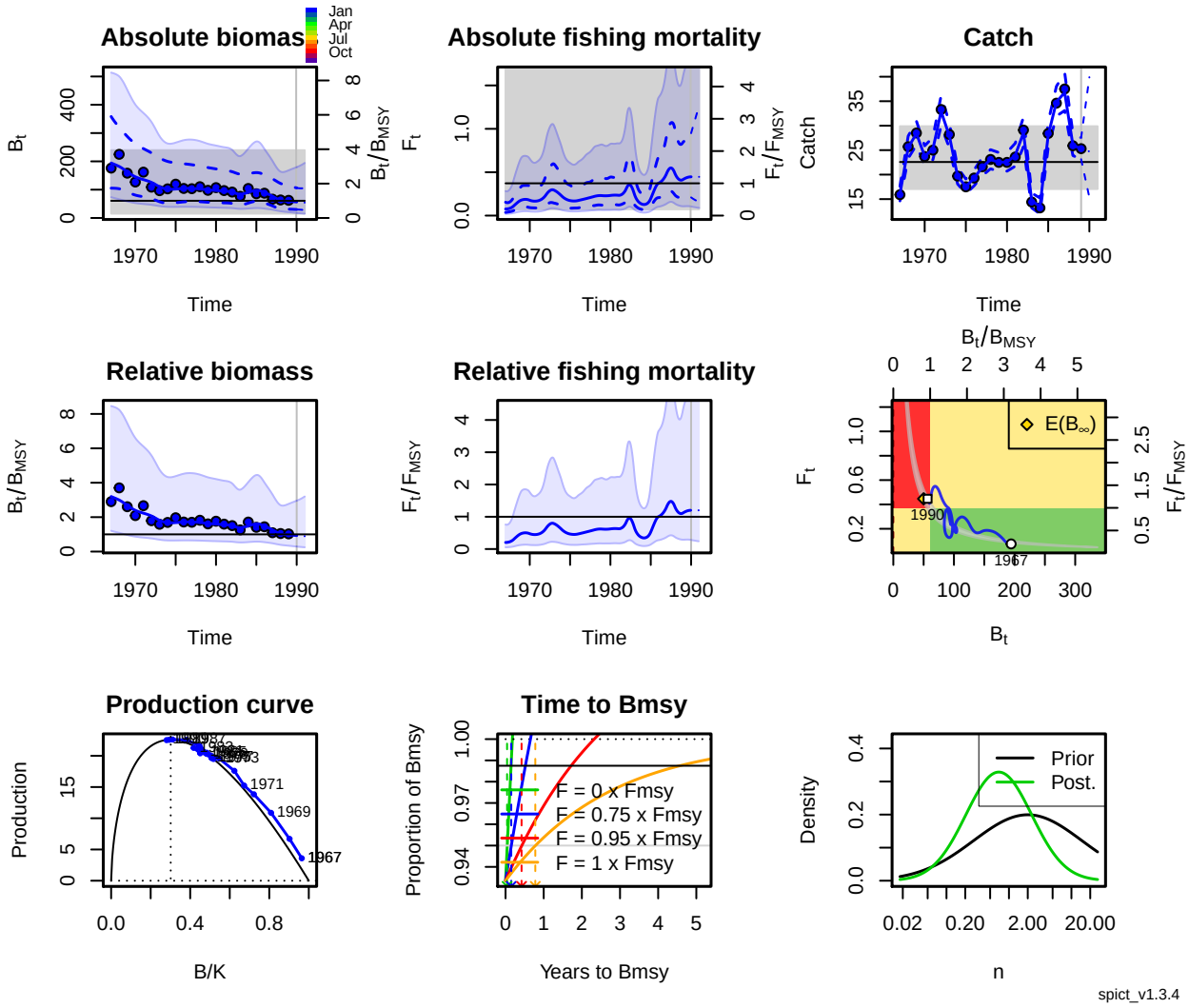
Here is a line-by-line description of the summary:

- Line 1: Convergence of the model fit, which has code 0 if the fit was succesful. If this is not the case convergence was not obtained and reported results should not be used. In case of non-convergence results will still be reported to aid diagnosis of the problem.
- Line 2: Objective function value at the optimum. The objective function is the likelihood function if priors are not used and the posterior density function if priors are used.
- Line 3: The Euler time step used in the calculation.
- Line 4: Number of observations for the time series used.
- Line 6-9: Summary of the priors used in the fit. The priors shown here are the default priors that are applied when priors are unspecified. These are relatively uninformative and are applied because most data-limited situations do not allow simultaneous estimation of all noise parameters and `logn`. The default priors can be disabled (see the section on priors).
- Line 11-25: Summary of the parameter estimates and their 95% CIs. These can be extracted as a data frame with `sumspict.parest(res)`.
- Line 27-31: Estimates of deterministic reference points with 95% CIs. These are the reference points one would derive if stochasticity were ignored. Can be extracted with `sumspict.drefpoints(res)`.
- Line 32-36: Estimates of stochastic reference points with 95% CIs. These are the reference points of the stochastic model. The column ´rel.diff.Drp´ shows the relative difference when compared to the deterministic reference points. The information can be extracted with `sumspict.srefpoints(res)`.
- Line 38-43: State estimates in the final year where data were available. The states of the model are biomass (`B`) and fishing mortality (`F`) with the year of the estimates appended. The year is shown as a decimal number as estimates within year are possible. Both absolute (`B` and `F`) and relative estimates (`B/Bmsy` and `F/Fmsy`) are shown. The relative estimates are calculated using the type of reference points given by `msytype` (line 38), where `s` is stochastic and `d` is deterministic. Here `msytype` is ´s´. This information can be extracted using `sumspict.states(res)`.
- Line 45-52: Predictions of absolute and relative biomass and fishing mortality at the time indicated by `inp$timepredi`, here 1990 (line 47-50). In addition, predicted catch at the time indicated by `inp$timepredc` (line 51). Finally, the equilibrium biomass, indicated by E(B_inf), if current conditions remain constant. There predictions or forecasts are calculated under the fishing scenario given by `inp$ffac`. See the section on forecasting for more information. The prediction summary can be extracted using `sumspict.predictions(res)`.

Interpreting plots of results

`spict` comes with several plotting abilities. The basic plotting of the results is done using the generic function `plot` that produces a multipanel plot with the most important outputs.

```
plot(res)
```



Some general comments can be made regarding the style and colours of these plots:

- Estimates (biomass, fishing mortality, catch, production) are shown using blue lines.
- 95% CIs of absolute quantities are shown using dashed blue lines.
- 95% CIs of relative biomass and fishing mortality are shown using shaded blue regions.
- Estimates of reference points (B_{MSY}, F_{MSY}, MSY) are shown using black lines.
- 95% CIs of reference points are shown using grey shaded regions.
- The end of the data range is shown using a vertical grey line.
- Predictions beyond the data range are shown using dotted blue lines.
- Data are shown using points colored by season. Different index series use different point characters (not shown here).

The individual plots can be plotted separately using the `plotspict.*` family of plotting functions; all functions are summarised in Table 1 and their common arguments that control their look in Table 2:

Table 1: Available plotting functions.

Function	Plot
Data	
`plotspict.ci`	Basic data plotting (see section)
`plotspict.data`	Advanced data plotting (see section)

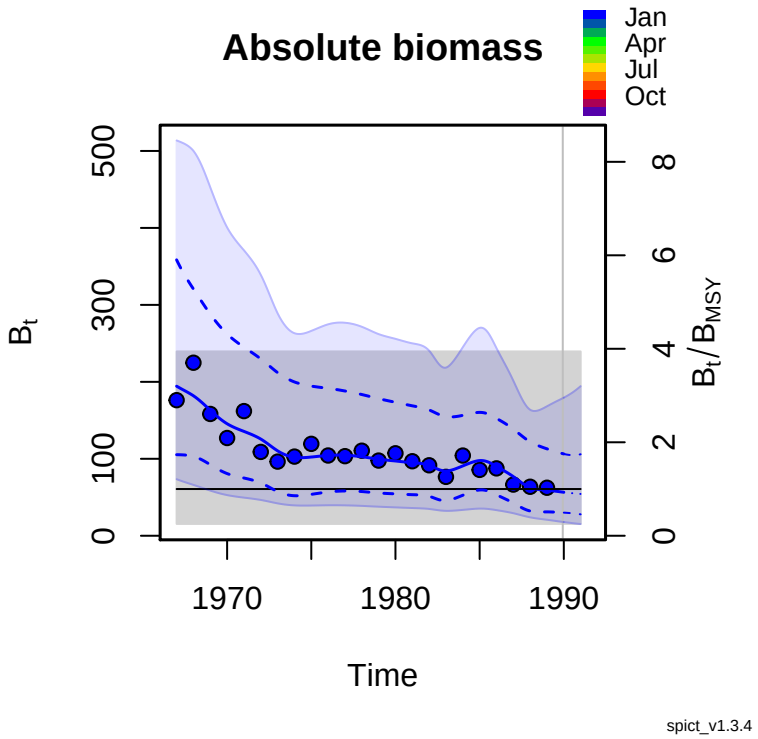
Function	Plot
Estimates	
`plotspict.bbmsy`	Relative biomass B/B_{MSY} estimates with uncertainty
`plotspict.biomass`	Absolute (and relative) biomass estimates with uncertainty
`plotspict.btrend`	Expected biomass trend
`plotspict.catch`	Catch data and estimates
`plotspict.f`	Absolute (and relative) fishing mortality F
`plotspict.fb`	Kobe plot of relative fishing mortality over biomass estimates
`plotspict.ffmsy`	Relative fishing mortality F/F_{MSY}
`plotspict.priors`	Prior-posterior distribution of all parameters that are estimated using priors
`plotspict.production`	Production over B/K
`plotspict.season`	Seasonal pattern of fishing mortality F
Diagnostics & extras	
`plotspict.diagnostic`	OSA residual analysis to evaluate the fit
`plotspict.osar`	One-step-ahead residual plots, one for data time-series
`plotspict.likprof`	Profile likelihood of one or two parameters
`plotspict.retro`	Retrospective analysis
`plotspict.retro.fixed`	Retrospective analysis of fixed effects
`plotspict.infl`	Influence statistics of observations
`plotspict.inflsum`	Summary of influence of observations
`plotspict.tc`	Time to B_{MSY} under different scenarios about F

Table 2: Common arguments in the `plotspict.*` family of funtions

Argument	Value	Result
`logax`	logical	If `TRUE`, the y-axis is in log scale
`main`	string	The title of the plot
`ylim`	numeric vector	The limits of the y-axis
`plot.obs`	logical	If `TRUE` (default) the observations are shown
`qlegend`	logical	If `TRUE` (default) the color legend is shown
`xlab`, `ylab`	string	The x and y axes labels
`stamp`	string	Adds a "stamp" at the bottom right corner of the plotting area Default is the version and SHA hash of `spict`. An empty string removes the stamp.

We will now look at them one at a time. The top left is the plot of absolute biomass

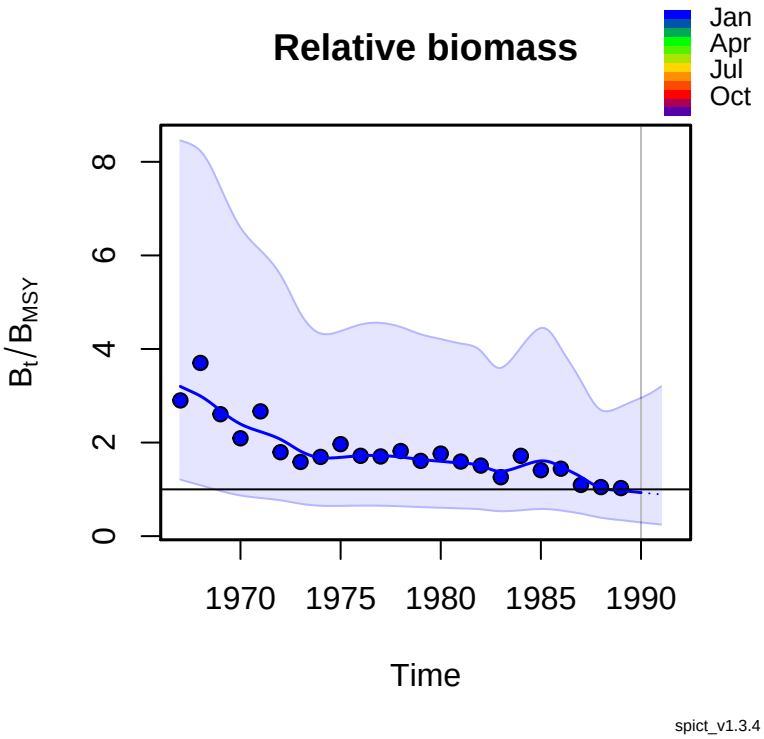
```
plotspict.biomass(res)
```



Note that this plot has a y-axis on the right side related to the relative biomass (B_t/B_{MSY}). The shaded 95% CI region relates to this axis, while the dashed blue lines relate to the left y-axis indicating absolute levels. The dashed lines and the shaded region are shown on the same plot to make it easier to assess whether the relative or absolute levels are most accurately estimated. Here, the absolute are more accurate than the relative. Later, we will see examples of the opposite. The horizontal black line is the estimate of B_{MSY} with 95% CI shown as a grey region.

The plot of the relative biomass is produced using

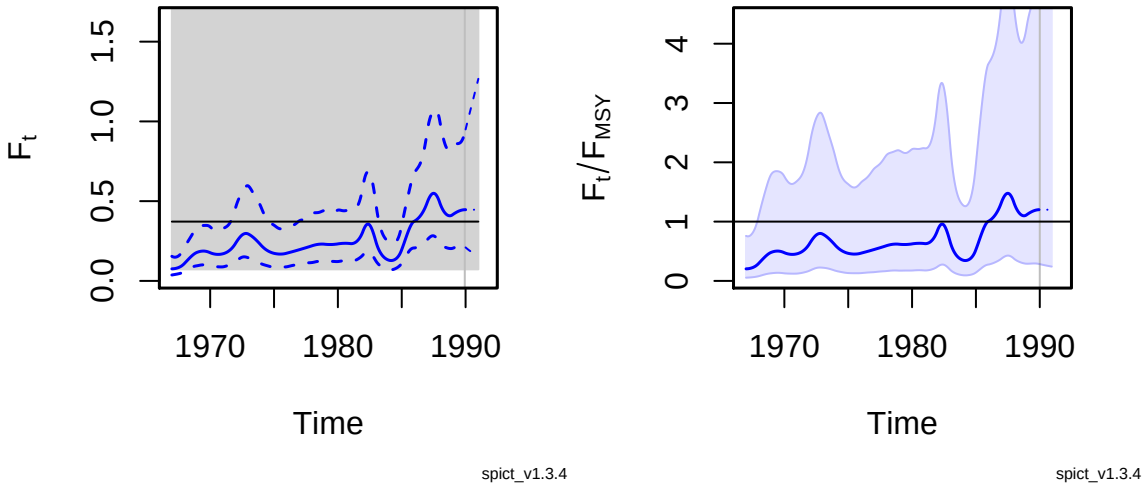
```
plotspict.bbmsy(res)
```



This plot contains much of the same information as given by `plotspict.biomass`, but without the information about absolute biomass and without the 95% CI around the B_{MSY} reference point.

The plots of fishing mortality follow the same principles

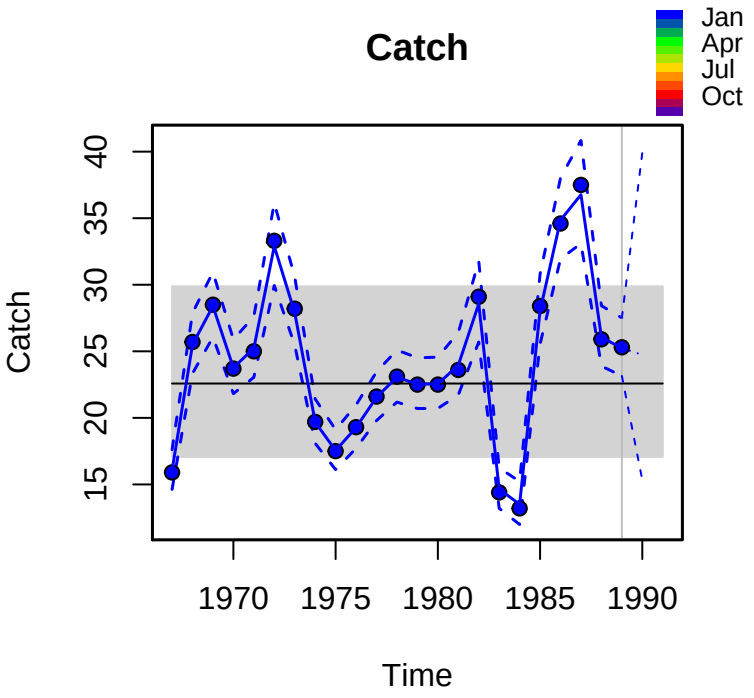
```
plotspict.f(res, main='', qlegend=FALSE, rel.axes=FALSE, rel.ci=FALSE)
plotspict.ffmsy(res, main='', qlegend=FALSE)
```



The estimate of F_{MSY} is shown with a horizontal black line with 95% CI shown as a grey region (left plot). The 95% CI of F_{MSY} is very wide in this case. As shown here it is quite straightforward to remove the information about relative levels from the plot of absolute fishing mortality. Furthermore, the argument `main=''` removes the heading and `qlegend=FALSE` removes the colour legend for data points.

The plot of the catch is produced using

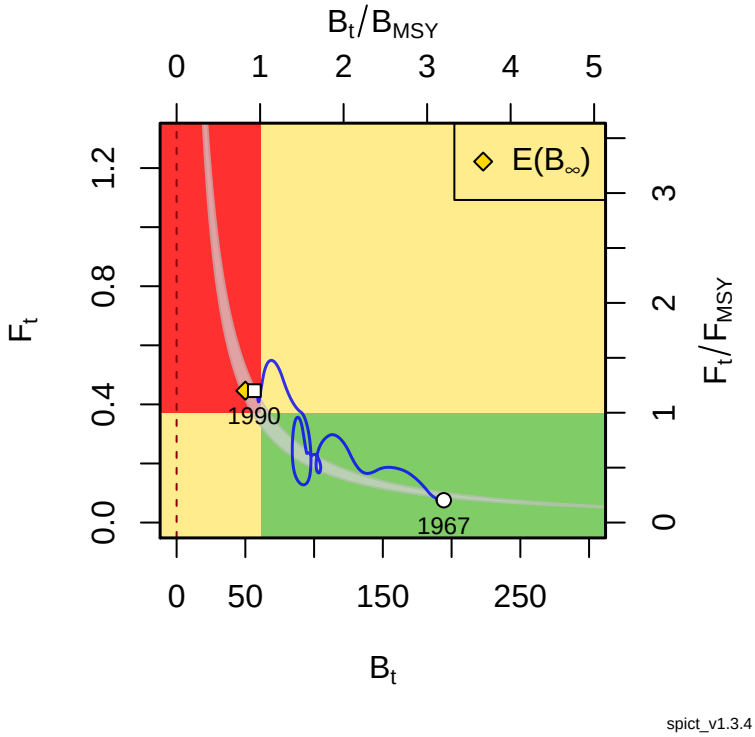
```
plotspict.catch(res)
```



This plot shows estimated catches (blue line) versus observed catches (points) with the estimate of MSY plotted as a horizontal black line with its 95% CI given by the grey region.

A phase plot (or kobe plot) of fishing mortality versus biomass is plotted using

```
plotspict.fb(res, ylim=c(0, 1.3), xlim=c(0, 300))
```



The plot shows the development of biomass and fishing mortality since the initial year (here 1967) indicated

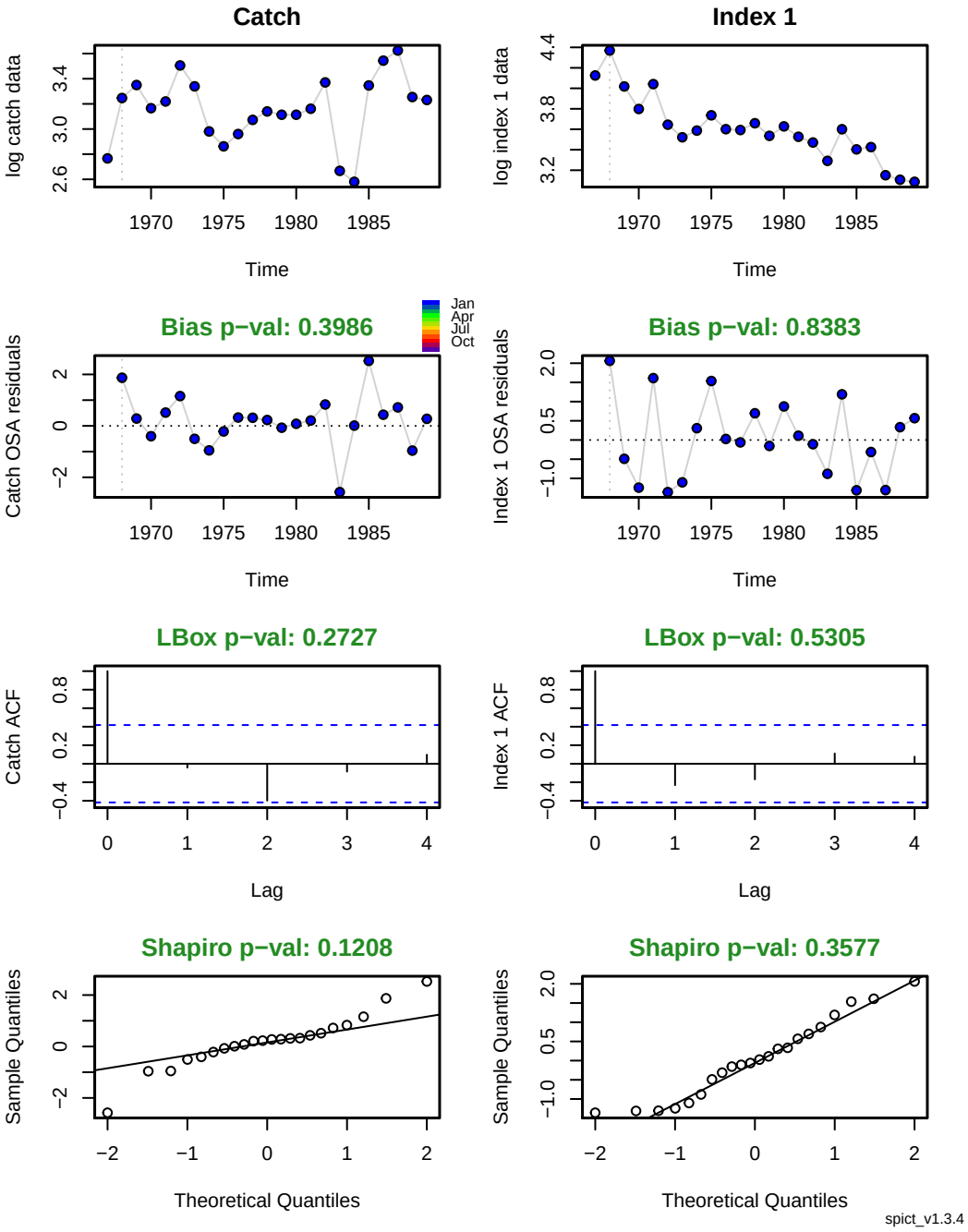
with a circle until the terminal year (here 1990) indicated with a square. The yellow diamond indicates the mean biomass over a long period if the current (1990) fishing pressure remains. This point can be interpreted as the fished equilibrium and is denoted $E(B_\infty)$ in the legend as a statistical way of expressing the expectation of the biomass as $t \to \infty$. As the current fishing mortality is close to F_{MSY} the expected long term biomass is close to B_{MSY}.

A vertical dashed red line at $B_t = 0$ indicates the biomass level below which the stock has crashed. The grey shaded banana-shaped area indicates the 95% confidence region of the pair F_{MSY}, B_{MSY}. This region is important to visualise jointly as the two reference points are highly (negatively) correlated.

Residuals and diagnostics

Before proceeding with the results for an actual assessment it is very important that the model residuals are checked and possible model deficiencies identified. Residuals can be calculated using `calc.osa.resid()`. OSA stands for one-step-ahead, which are the proper residuals for state-space models. More information about OSA residuals is contained in Pedersen and Berg (2017). To calculate and plot residuals and diagnostics do

```
res <- calc.osa.resid(res)
plotspict.diagnostic(res)
```



The first column of the plot contains information related to catch data and the second column contains information related to the index data. The rows contain

1. Log of the input data series.
2. OSA residuals with the p-value of a test for bias (i.e. that the mean of the residuals is different from zero) in the plot header. If the header is green the test was not significant, otherwise the header would be red.
3. Empirical autocorrelation of the residuals. Two tests for significant autocorrelation is performed. A Ljung-Box simultaneous test of multiple lags (here 4) with p-value shown in the header, and tests for individual lags shown by dashed horizontal lines in the plot. Here no violation is identified.
4. Tests for normality of the residuals both as a QQ-plot and with a Shapiro test with p-value shown in the plot header.

This data did not have any significant violations of the assumptions, which increases confidence in the results. For a discussion of possible violations and remedies the reader is referred to Pedersen and Berg (2017).

Extracting parameter estimates

To extract an estimated quantity, here `logBmsy` use

```
get.par('logBmsy', res)
                ll      est       ul        sd       cv
   logBmsy 2.734573 4.106544 5.478515 0.6999851 0.170456
```

This returns a vector with `ll` being the lower 95% limit of the CI, `est` being the estimated value, `ul` being the upper 95% limit of the CI, `sd` being the standard deviation of the estimate, and `cv` being the coefficient of variation of the estimate. The estimated quantity can also be returned on the natural scale (as opposed to log scale) by running

```
get.par('logBmsy', res, exp=TRUE)
                ll      est       ul        sd        cv
   logBmsy 15.40317 60.73644 239.4907 0.6999851 0.7951617
```

This essentially takes the exponential of `ll`, `est` and `ul` of the values in log, while `sd` is unchanged as it is the standard deviation of the quantity on the scale that it is estimated (here log). When transforming using `exp=TRUE` the $CV = \sqrt{e^{\sigma^2} - 1}$. Most parameters are log-transformed under estimation and should therefore be extracted using `exp=TRUE`.

For a standard fit (not using robust observation error, seasonality etc.), the quantities that can be extracted using this method are

```
list.quantities(res)
    [1] "Bm"                "Bmsy"              "Bmsy2"
    [4] "BmsyB0"            "Bmsyd"             "Bmsys"
    [7] "Cp"                "Emsy"              "Emsy2"
   [10] "Fmsy"              "Fmsyd"             "Fmsys"
   [13] "gamma"             "isdb2"             "isdc2"
   [16] "isde2"             "isdf2"             "isdi2"
   [19] "K"                 "logalpha"          "logB"
   [22] "logBBmsy"          "logbeta"           "logbkfrac"
   [25] "logBl"             "logBlBmsy"         "logBlK"
   [28] "logBm"             "logBmBmsy"         "logBmK"
   [31] "logBmsy"           "logBmsyd"          "logBmsyPluslogFmsy"
   [34] "logBmsys"          "logBp"             "logBpBmsy"
   [37] "logBpK"            "logCp"             "logCpred"
   [40] "logEmsy"           "logEmsy2"          "logEp"
   [43] "logF"              "logFFmsy"          "logFFmsynotS"
   [46] "logFl"             "logFlFmsy"         "logFlFmsynotS"
   [49] "logFm"             "logFmFmsy"         "logFmFmsynotS"
   [52] "logFmsy"           "logFmsyd"          "logFmsys"
   [55] "logFnotS"          "logFp"             "logFpFmsy"
   [58] "logFpFmsynotS"     "logFs"             "logIp"
   [61] "logIpred"          "logK"              "logm"
   [64] "logMSY"            "logMSYd"           "logMSYs"
   [67] "logn"              "logq"              "logq2"
   [70] "logr"              "logrc"             "logrold"
   [73] "logsdb"            "logsdc"            "logsdf"
   [76] "logsdi"            "m"                 "MSY"
   [79] "MSYd"              "MSYs"              "p"
```

```
 [82] "q"                  "r"                  "rc"
 [85] "rold"               "sdb"                "sdc"
 [88] "sde"                "sdf"                "sdi"
 [91] "seasonsplinefine"
```

These should be relatively self-explanatory when knowing that reference points ending with `s` are stochastic and those ending with `d` are deterministic, quantities ending with `p` are predictions and quantities ending with `l` are estimates in the final year. If a quantity is available both on natural and log scale, it is preferred to transform the quantity from log as most quantities are estimated on the log scale.

Extracting correlation between parameter estimates

The covariance between the model parameters (fixed effects) can be extracted from the results list

```
res$cov.fixed
                  logm         logK         logq         logn       logsdb
logm    0.0204040270 -0.005706715  0.026834551 -0.156740713  0.032240093
logK   -0.0057067153  0.037105157 -0.038074892 -0.011343165 -0.021784058
logq    0.0268345508 -0.038074892  0.091311219 -0.227633753  0.040958405
logn   -0.1567407130 -0.011343165 -0.227633753  1.473743206 -0.190011567
logsdb  0.0322400932 -0.021784058  0.040958405 -0.190011567  0.980089527
logsdf  0.0014253361 -0.002407435  0.003270277 -0.006648031 -0.002144231
logsdi -0.0007603786 -0.002521057  0.002848540  0.007158237  0.010535581
logsdc  0.0015535702  0.001300312 -0.008392550  0.001997844  0.137920130
               logsdf        logsdi       logsdc
logm    0.0014253361 -0.0007603786  0.001553570
logK   -0.0024074353 -0.0025210567  0.001300312
logq    0.0032702768  0.0028485398 -0.008392550
logn   -0.0066480310  0.0071582370  0.001997844
logsdb -0.0021442311  0.0105355812  0.137920130
logsdf  0.0262881996 -0.0002784388 -0.035159489
logsdi -0.0002784388  0.0237200094  0.005885861
logsdc -0.0351594891  0.0058858615  0.846804258
```

It is however easier to interpret the correlation rather than covariance. The correlation matrix can be calculated using

```
cov2cor(res$cov.fixed)
                 logm         logK        logq         logn      logsdb
logm    1.00000000 -0.207401134  0.62169062 -0.903884554  0.22798458
logK   -0.20740113  1.000000000 -0.65412309 -0.048507198 -0.11423226
logq    0.62169062 -0.654123088  1.00000000 -0.620531281  0.13691407
logn   -0.90388455 -0.048507198 -0.62053128  1.000000000 -0.15810160
logsdb  0.22798458 -0.114232259  0.13691407 -0.158101597  1.00000000
logsdf  0.06154304 -0.077082728  0.06674858 -0.033775494 -0.01335852
logsdi -0.03456324 -0.084978303  0.06120724  0.038285801  0.06909843
logsdc  0.01181902  0.007335657 -0.03018147  0.001788377  0.15139212
              logsdf       logsdi       logsdc
logm    0.06154304 -0.03456324  0.011819019
logK   -0.07708273 -0.08497830  0.007335657
logq    0.06674858  0.06120724 -0.030181469
logn   -0.03377549  0.03828580  0.001788377
logsdb -0.01335852  0.06909843  0.151392119
logsdf  1.00000000 -0.01115044 -0.235651722
logsdi -0.01115044  1.00000000  0.041529920
```

```
   logsdc -0.23565172  0.04152992  1.000000000
```

For this data most parameters are well separated, i.e. relatively low correlation, perhaps with the exception of `logm` and `logn`, which have a correlation of -0.9. Note that `logr` is absent from the covariance matrix. This is because the model is parameterised in terms of `logm`, `logK`, and `logn` from which `logr` can be derived. The estimate of `logr` is reported using TMB's `sdreport()` function and can be extracted using `get.par()`.

The covariance between random effects (biomass and fishing mortality) is not reported automatically, but can be obtained by setting `inp$getJointPrecision` to `TRUE` (this entails longer computation time and memory requirement).

The covariance between sdported values (i.e. the values reported in `res$value`) are given in `res$cov`. As this matrix is typically large, the function `get.cov()` can be used to extract the covariance between two scalar quantities

```
cov2cor(get.cov(res, 'logBmsy', 'logFmsy'))
              [,1]       [,2]
   [1,]  1.0000000 -0.9982507
   [2,] -0.9982507  1.0000000
```

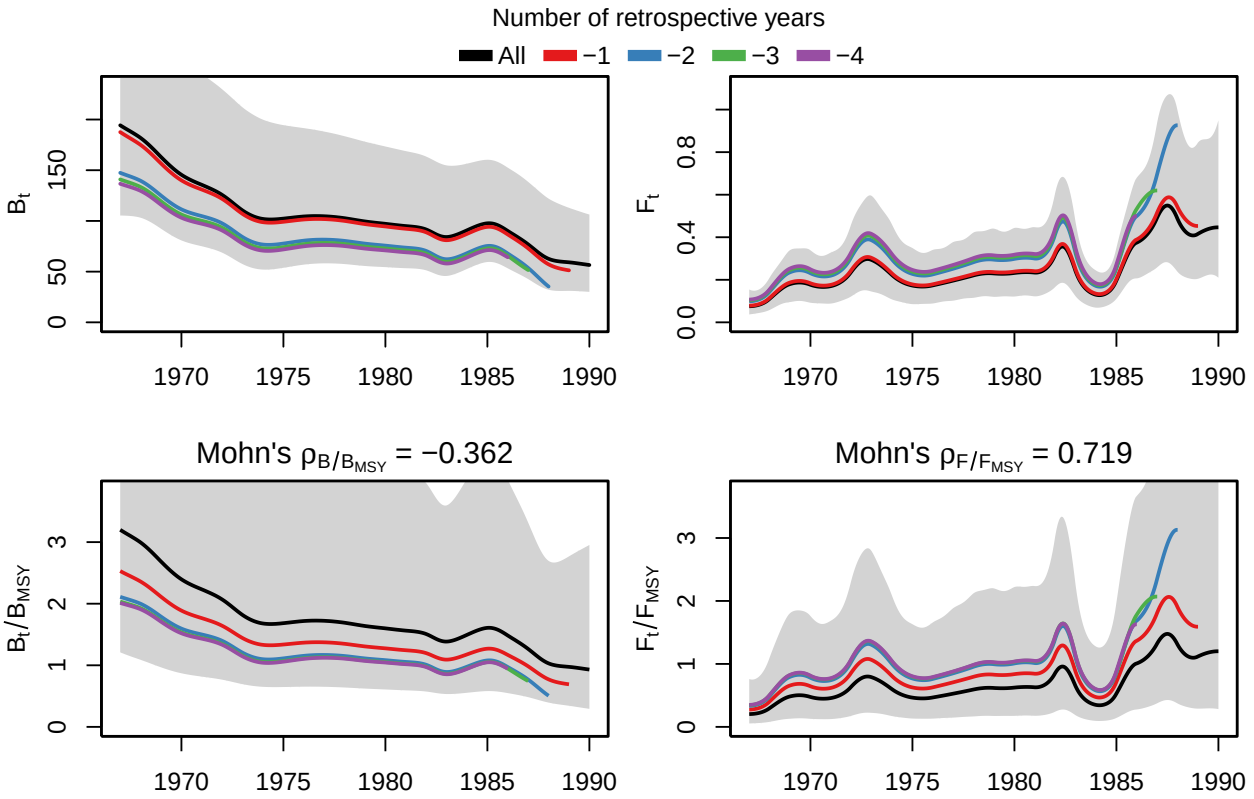
This reveals that for this data set the estimates of log Fmsy and log Bmsy are highly correlated. This is often the case and the reason why the model is reparameterised.

Advanced functionality

Retrospective plots

Retrospecitive plots are sometimes used to evaluate the robustness of the model fit to the introduction of new data, i.e. to check whether the fit changes substantially when new data becomes available. Such calculations and plotting thereof can be performed using `retro()` as shown here

```
## res <- fit.spict(pol$albacore)
res <- retro(res, nretroyear = 4)
plotspict.retro(res)
        FFmsy      BBmsy
    0.7187925 -0.3619995
```

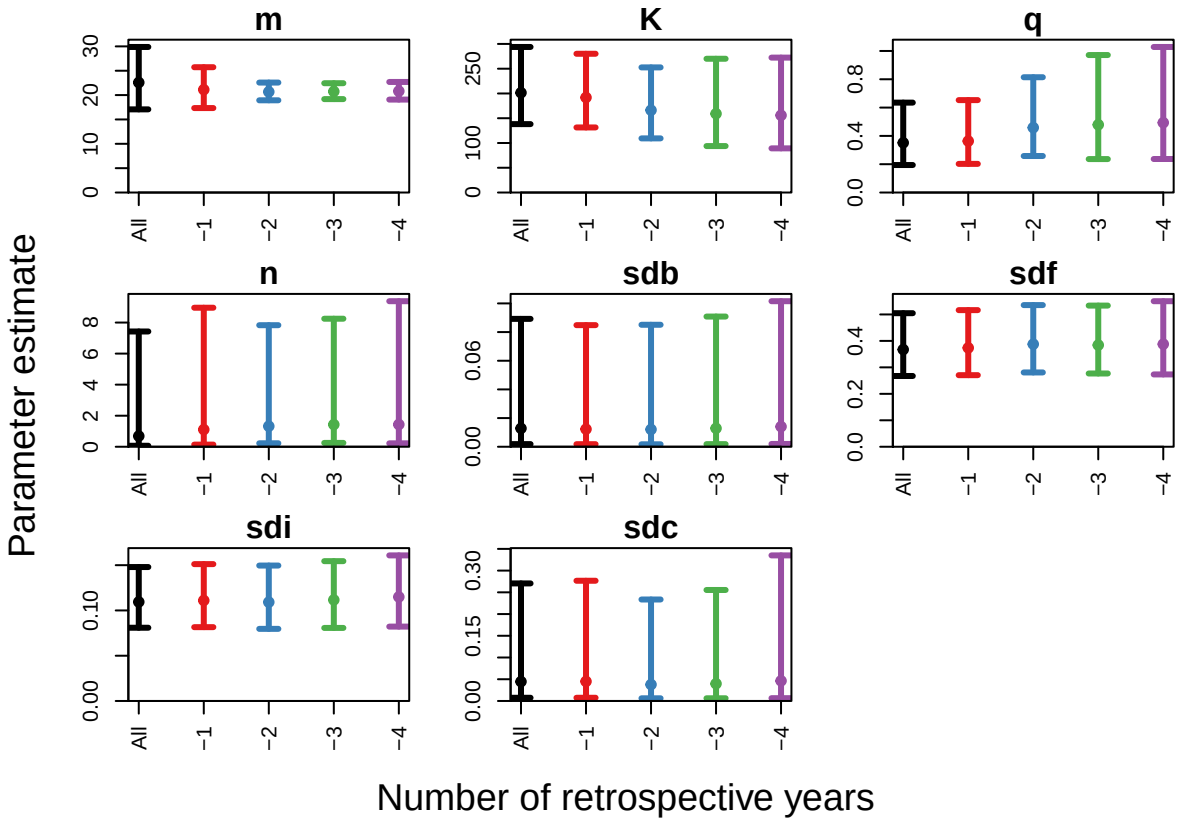


By default `retro` creates 5 scenarios with catch and index time series which are shortened by the 1 to 5 last observations. The number of scenarios and thus observations which are removed can be changed with the argument `nretroyear` in the function `retro`. The graphs show the different peels with different colors. For the relative biomass and fishing mortality, the Mohn's rho are shown inside the corresponding plots. Mohn's rho for other quantities can be calculated using the function `mohns_rho`.

```
## res <- fit.spict(pol$albacore)
## res <- retro(res)
mohns_rho(res, what = c("FFmsy", "BBmsy"))
        FFmsy      BBmsy
    0.7187925 -0.3619995
```

The function `plotspict.retro.fixed` makes a plot of the paramter estimates, with corresponding 95% confidence intervals for each of the retrospective runs.

```
## res <- fit.spict(pol$albacore)
## res <- retro(res)
plotspict.retro.fixed(res)
```

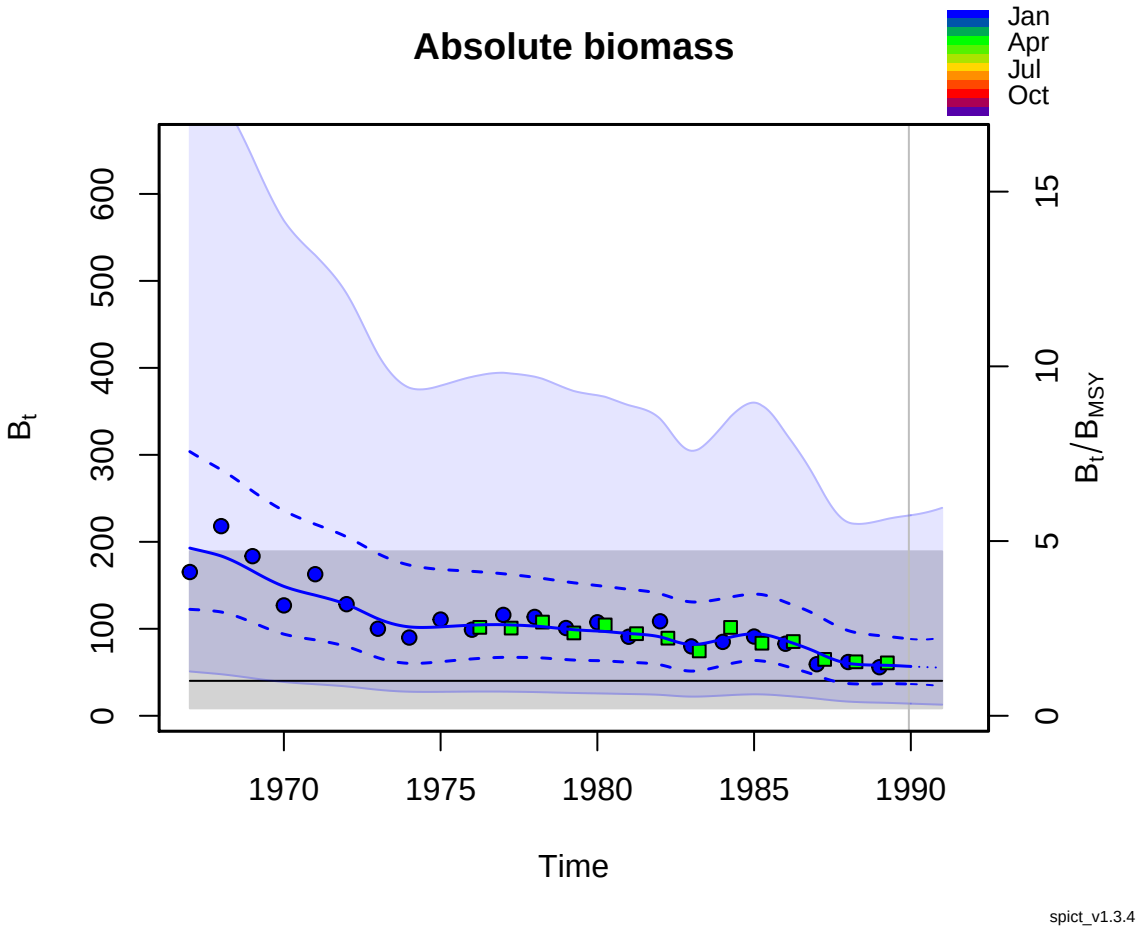


Estimation using two or more biomass indices

The estimation can be done using more than one biomass index, for example when scientific surveys are performed more than once every year or when there are both commercial and survey CPUE time-series available. The following example emulates a situation where a long but noisy first quarter index series and a shorter and less noisy second quarter index series are available with different catchabilities

```
set.seed(123)
inp <- list(timeC=pol$albacore$timeC, obsC=pol$albacore$obsC)
inp$timeI <- list(pol$albacore$timeI, pol$albacore$timeI[10:23]+0.25)
inp$obsI <- list()
inp$obsI[[1]] <- pol$albacore$obsI * exp(rnorm(23, sd=0.1)) # Index 1
inp$obsI[[2]] <- 10*pol$albacore$obsI[10:23] # Index 2
res <- fit.spict(inp)
sumspict.parest(res)
             estimate        cilow       ciupp    log.est
alpha1     5.54953281   0.94904912  32.4507063  1.7137137
alpha2     3.45424303   0.58223845  20.4929699  1.2396033
beta       0.12057032   0.01912937   0.7599416 -2.1155222
r          0.18601585   0.08915984   0.3880884 -1.6819234
rc         1.20960483   0.19387709   7.5467597  0.1902937
rold       0.26864004   0.05296981   1.3624264 -1.3143829
m         24.30175989  17.98198037  32.8426303  3.1905488
K        220.56430343 148.87723219 326.7699919  5.3961893
q1         0.35403668   0.22557137   0.5556644 -1.0383548
q2         3.60405904   2.32437447   5.5882741  1.2820607
n          0.30756467   0.03135787   3.0166595 -1.1790699
sdb        0.02006319   0.00372273   0.1081282 -3.9088686
```

```
   sdf      0.36879350   0.26948764   0.5046934 -0.9975184
   sdi1     0.11134131   0.08158779   0.1519454 -2.1951549
   sdi2     0.06930312   0.04587566   0.1046944 -2.6692653
   sdc      0.04446555   0.00764489   0.2586282 -3.1130406
plotspict.biomass(res)
```



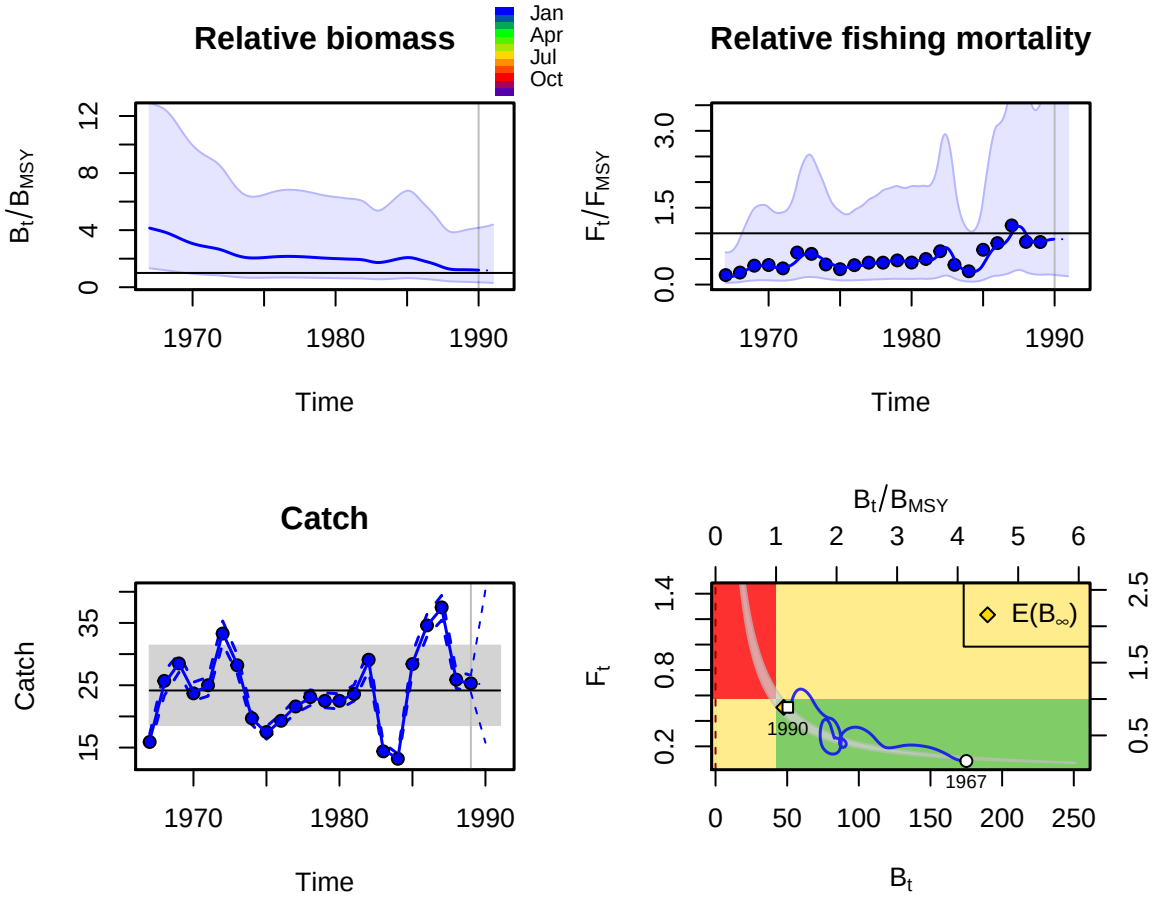
The model estimates seperate observation noises and finds that the first index (`sdi1`) is more noisy than the second (`sdi2`). It is furthermore estimated that the catchabilities are different by a factor 10 (`q1` versus `q2`). The biomass plot shows both indices, with circles indicating the first index and squares indicating the second index (the two series can also be distinguished by their colours). It is possible to force the model to assume that two or more index time-series have the same level of observation noise (CV). For example, to assume that `sdi1` equals `sdi2` one must add `inp$mapsdi <- c(1,1)` before calling `fit.spict(inp)`. The length of `mapsdi` should equal the number of indices. In case of 3 index series one could for example use `inp$mapsdi <- c(1, 1, 2)` to have series 1 and 2 share sdi and have a separate sdi for series 3.

Using effort data instead of commercial CPUE

It is possible to use effort data directly in the model instead of calculating commercial CPUE and inputting this as an index. It is beyond the scope of this vignette to discuss all problems associated with indices based on commercial CPUEs, however it is intuitively clear that using the same information twice (catch as catch and catch in catch/effort) induces a correlation, which the model does not account for. These problems are easily avoided by putting catch and effort seperately

```
inpeff <- list(timeC=pol$albacore$timeC, obsC=pol$albacore$obsC,
               timeE=pol$albacore$timeC, obsE=pol$albacore$obsC/pol$albacore$obsI)
```

```
repeff <- fit.spict(inpeff)
sumspict.parest(repeff)
              estimate        cilow        ciupp    log.est
   beta     0.07385346   0.01464722   0.37238022 -2.6056724
   r        0.23822940   0.10656857   0.53255145 -1.4345212
   rc       1.14399161   0.21452287   6.10059352  0.1345236
   rold     0.40826821   0.03851363   4.32789486 -0.8958309
   m       24.16376097  18.62118103  31.35608549  3.1848540
   K      189.53177049 132.49527183 271.12131269  5.2445567
   qf       0.41117182   0.25562631   0.66136489 -0.8887441
   n        0.41648801   0.04180196   4.14962009 -0.8758976
   sdb      0.01430671   0.00236752   0.08645421 -4.2470267
   sdf      0.37625012   0.27938785   0.50669401 -0.9775012
   sde      0.09820098   0.06985342   0.13805239 -2.3207391
   sdc      0.02778737   0.00568261   0.13587725 -3.5831736
par(mfrow=c(2, 2))
plotspict.bbmsy(repeff)
plotspict.ffmsy(repeff, qlegend=FALSE)
plotspict.catch(repeff, qlegend=FALSE)
plotspict.fb(repeff)
```



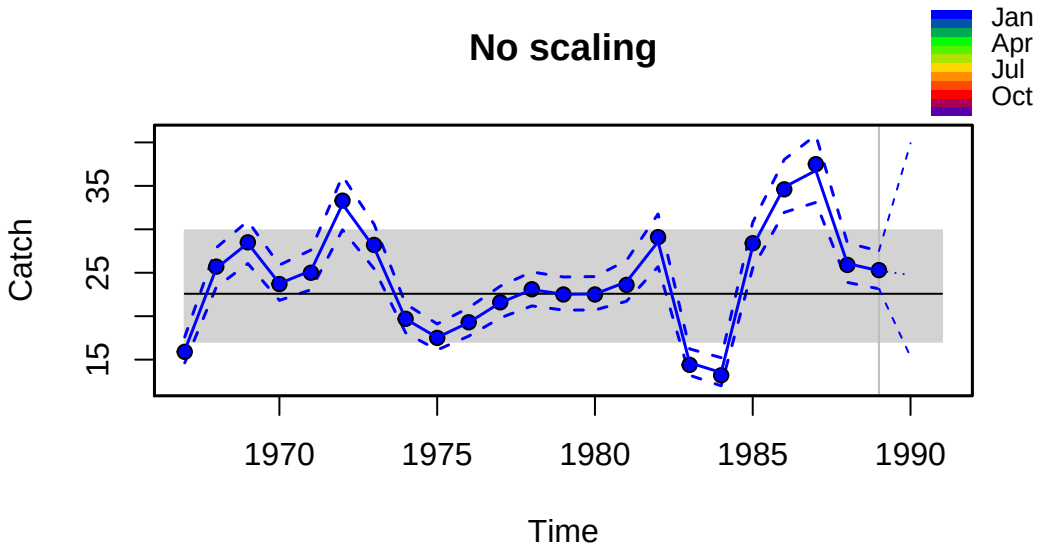
Here the model runs without an index of biomass and instead uses effort as an index of fishing mortality Note that index observations are missing from the biomass plot, but effort observations are present in the plot of fishing mortality. Note also that `q` is missing from the summary of parameter estimates and instead `qf` is present, which is the commercial catchability.

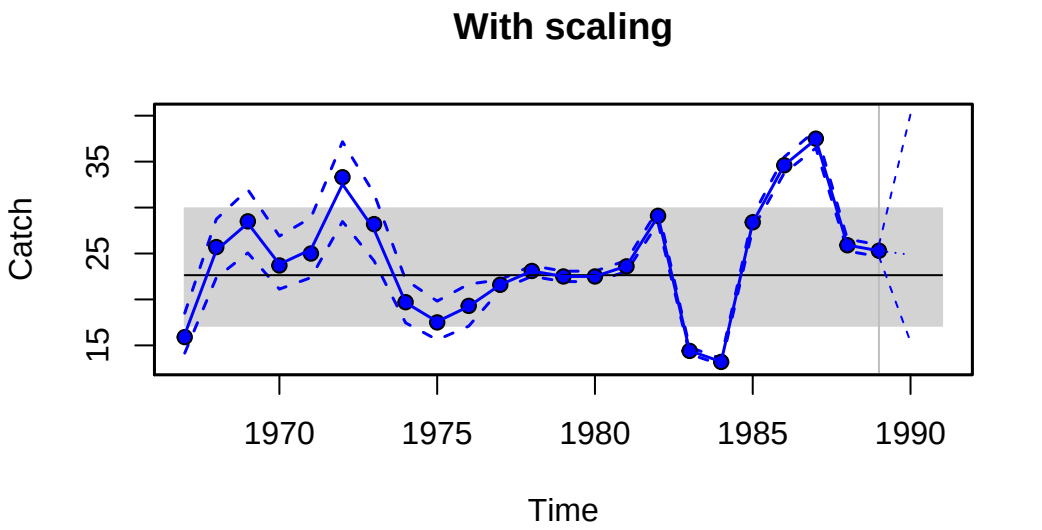
Overall for this data set the results in terms of stock status etc. do not change much, and this will probably often be the case, however using effort data directly instead of commercial CPUE is cleaner and avoids inputting the same data twice.

Scaling the uncertainty of individual data points

It is not always appropriate to assume that the observation noise of a data series is constant in time. Knowledge that certain data points are more uncertain than others can be implemented using `stdevfacC`, `stdevfacI`, and `stdevfacE`, which are vectors containing factors that are multiplied onto the standard deviation of the data points of the corresponding observation vectors. An example where the first 10 years of the biomass index are considered uncertain relative to the remaining time series and therefore are scaled by a factor 5.

```
inp <- pol$albacore
res1 <- fit.spict(inp)
inp$stdevfacC <- rep(1, length(inp$obsC))
inp$stdevfacC[1:10] <- 5
res2 <- fit.spict(inp)
par(mfrow=c(2, 1))
plotspict.catch(res1, main='No scaling')
plotspict.catch(res2, main='With scaling', qlegend=FALSE)
```





From the plot it is noted that the scaling factor widens the 95% CIs of the initial ten years of catch data, while narrowing the 95% CIs of the remaining years.

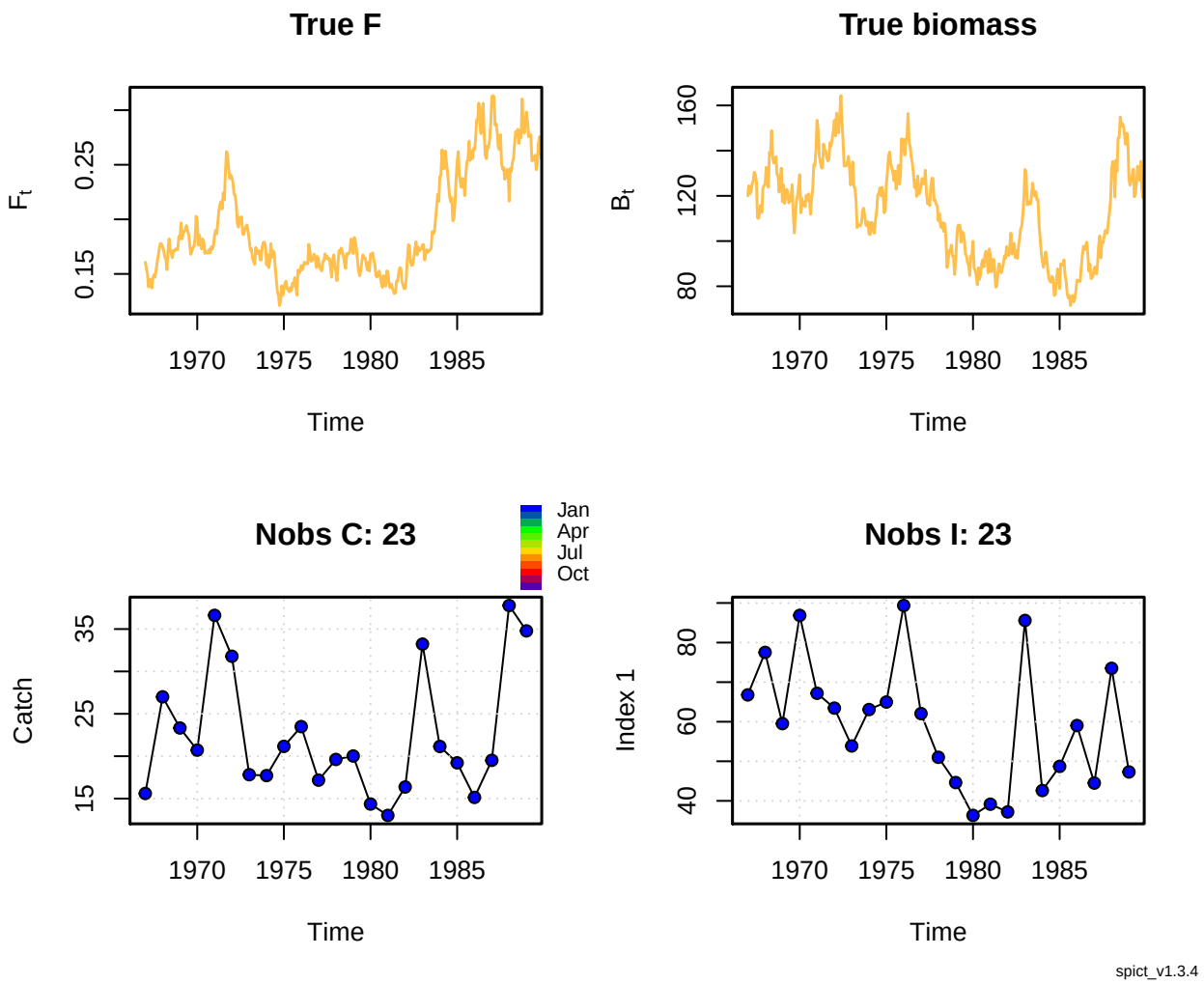
Simulating data

The package has built-in functionality for simulating data, which is useful for testing.

Annual data

Data are simulated using an input list, e.g. `inp`, containing parameter values specified in `inp$ini`. To simulate data using default parameters run

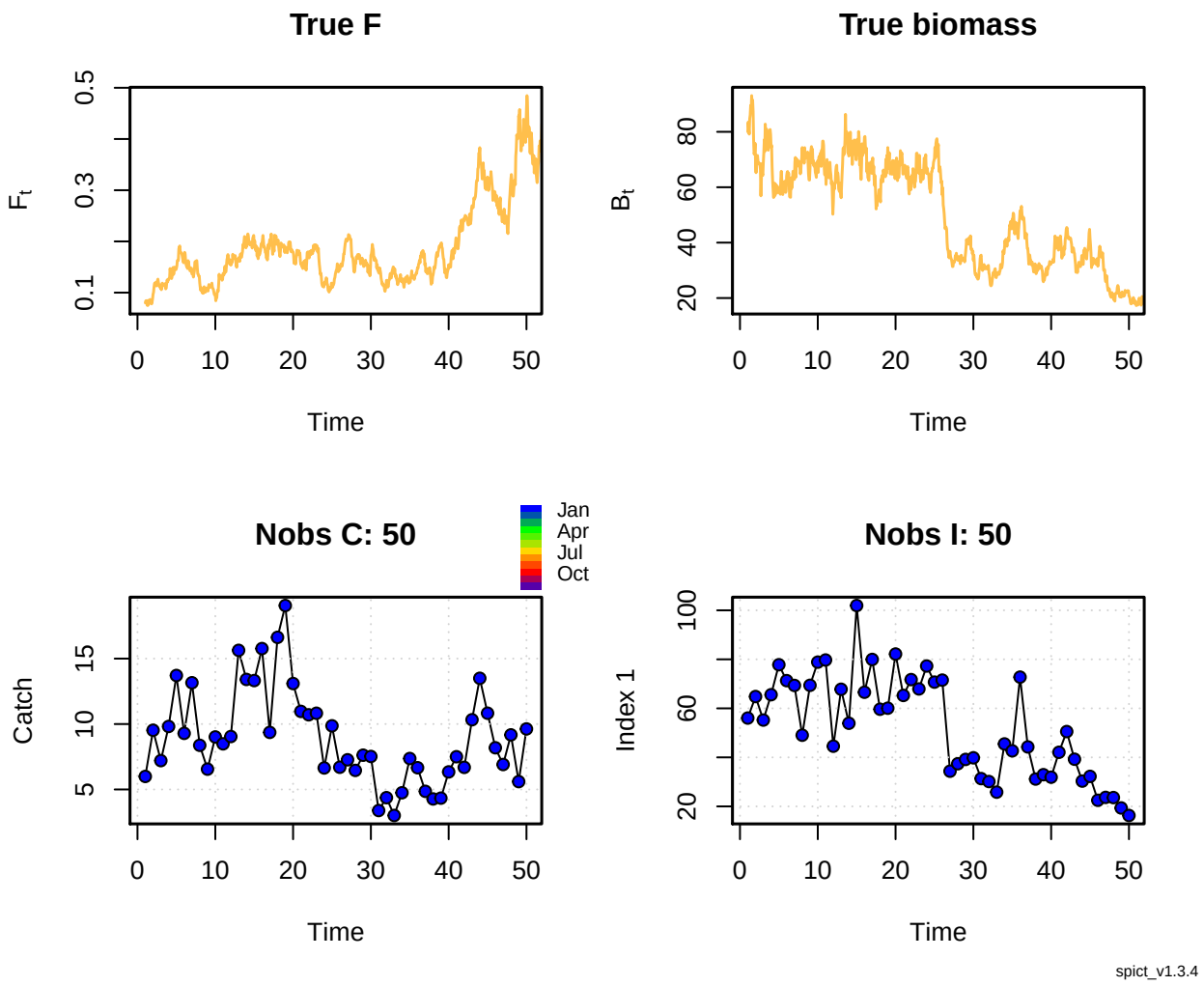
```
inp <- check.inp(pol$albacore)
sim <- sim.spict(inp)
plotspict.data(sim)
```



This will generate catch and index data of same length as the input catch and index time series (here 23 of each) at the time points of the input data. Note when plotting simulated data, the true biomass and fishing mortality are also included in the plot.

Another simple example is

```
inp <- list(ini=list(logK=log(100), logm=log(10), logq=log(1)))
sim <- sim.spict(inp, nobs=50)
plotspict.data(sim)
```

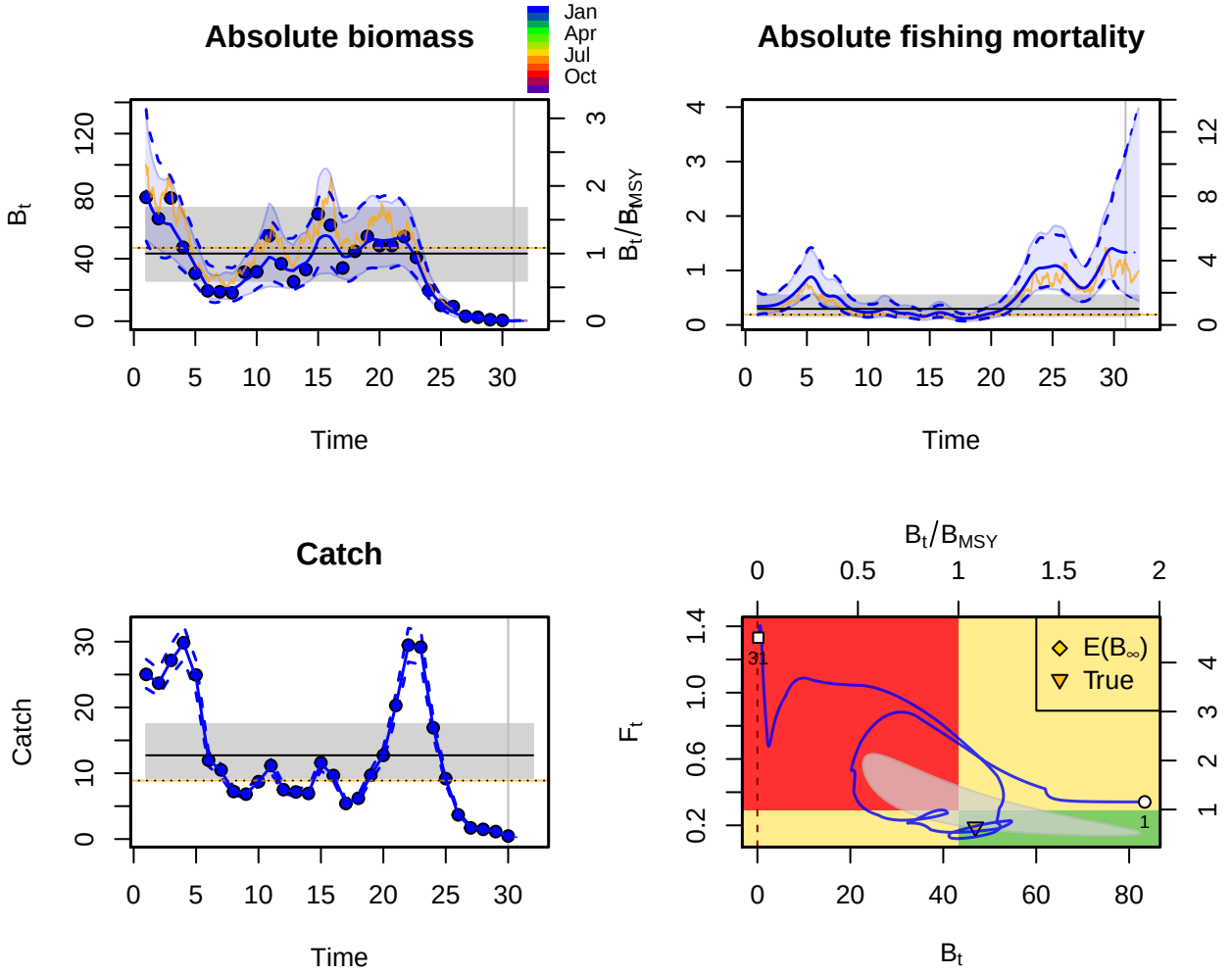


Here the required parameters are specified (the rest use default values), and the number of observations is specified as an argument to `sim.spict()`.

A more customised example including model fitting is

```
set.seed(31415926)
inp <- list(ini=list(logK=log(100), logm=log(10), logq=log(1),
                     logbkfrac=log(1), logF0=log(0.3), logsdc=log(0.1),
                     logsdf=log(0.3)))
sim <- sim.spict(inp, nobs=30)
res <- fit.spict(sim)
sumspict.parest(res)
             estimate  true       cilow       ciupp true.in.ci     log.est
   alpha   1.04607703  -9.0  0.31604360   3.4624247         -9  0.04504701
   beta    0.13191760  -9.0  0.02269879   0.7666599         -9 -2.02557780
   r       1.07672256  -9.0  0.35061490   3.3065665         -9  0.07392176
   rc      0.63474119  -9.0  0.34468263   1.1688909         -9 -0.45453794
   rold    0.45001540  -9.0  0.22252979   0.9100528         -9 -0.79847347
   m      14.48076845  10.0 10.46872821  20.0303848          0  2.67282146
   K      76.02606693 100.0 46.11318911 125.3429434          1  4.33107627
   q       1.19617688   1.0  0.85013703   1.6830688          1  0.17913054
   n       3.39263491   2.0  1.36225528   8.4492032          1  1.22160688
   sdb     0.19525937   0.2  0.08857956   0.4304178          1 -1.63342651
```

```
   sdf      0.34363005   0.3  0.25198733   0.4686014          1 -1.06818964
   sdi      0.20425634   0.2  0.12166919   0.3429024          1 -1.58837950
   sdc      0.04533085   0.1  0.00814470   0.2522975          1 -3.09376744
par(mfrow=c(2, 2))
plotspict.biomass(res)
plotspict.f(res, qlegend=FALSE)
plotspict.catch(res, qlegend=FALSE)
plotspict.fb(res)
```



Here the ratio between biomass in the initial year relative to `K` is set using `logbkfrac`, the initial fishing mortality is set using `logF0`, process noise of `F` is set using `logsdf`, and finally observation noise on catches is specified using `logsdc`.

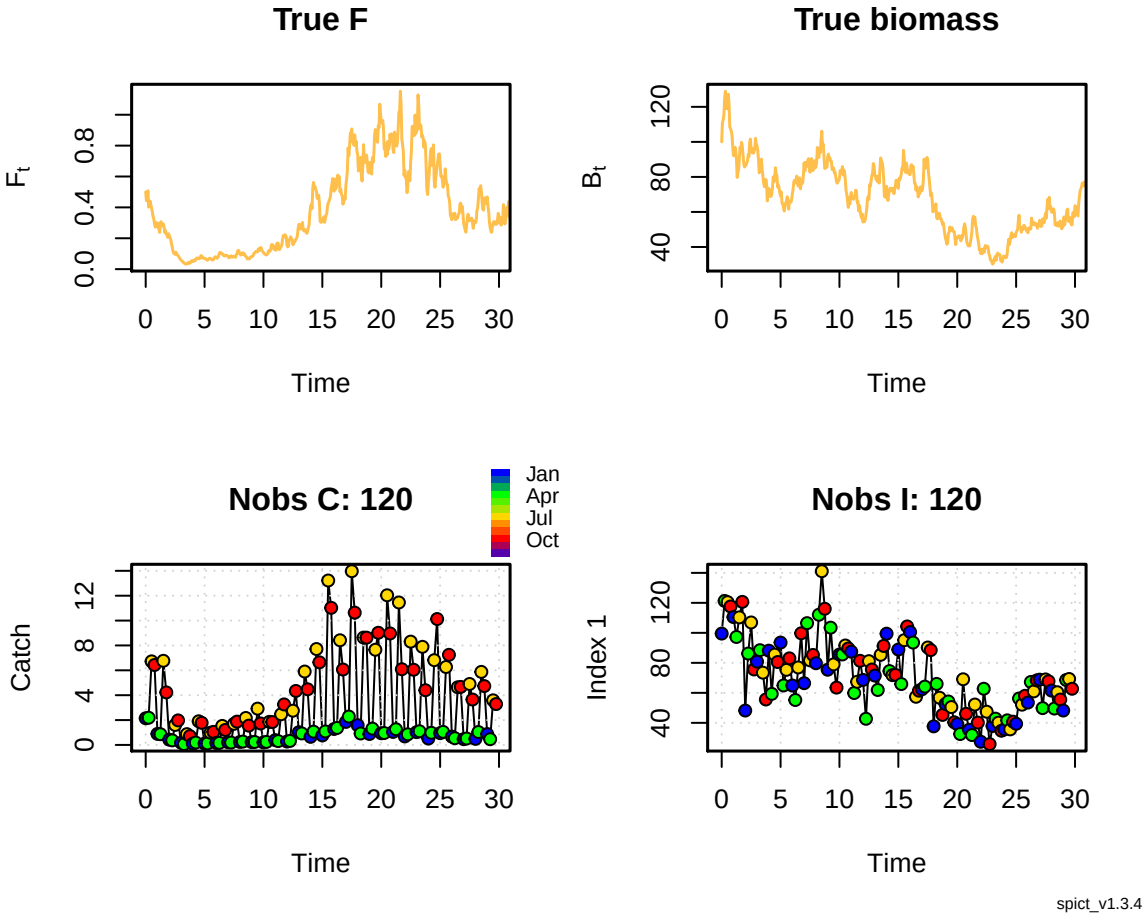
When printing the summary of the parameter estimates the true values are included as well as a check whether the true value was inside the 95% CIs. Similarly, the true biomass, fishing mortality, and reference points are included in the results plot using a yellow/orange colour.

Seasonal data

It is possible to simulate seasonal data (most often quarterly). Additional variables must be specified in the input list that define the type of seasonality to be used. Spline based seasonality is shown first (`inp$seasontype = 1`). This is the default and therefore does not need to be explicitly specified. It is required that number of seasons is specified using `nseasons` (4 indicates quarterly), the order of the spline

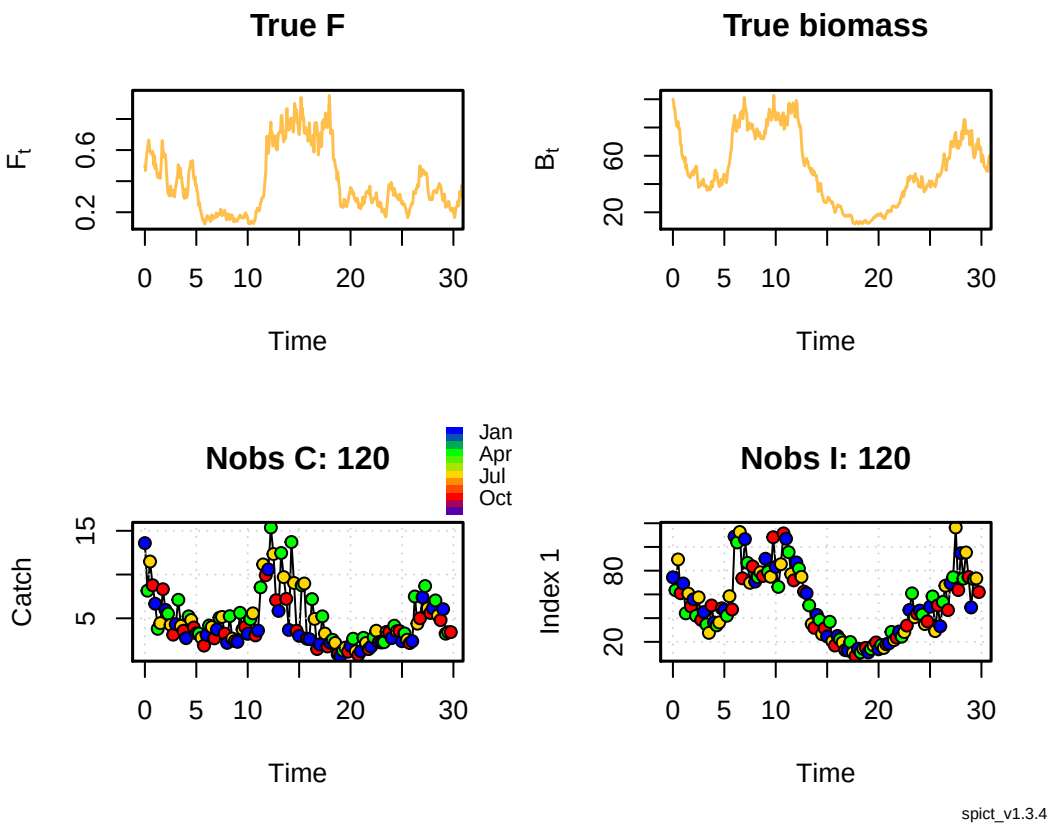
must be specified using `splineorder` (3 for quarterly data), time vectors for catch and index containing subannual time points must be specified, and finally the spline parameters (`logphi`) must be set. With four seasons `logphi` must be a vector of length 3, where each value in the vector gives the log fishing intensity relative to level in season four, which is `log(1)`. An example of simulating seasonal data using a spline is

```
set.seed(1234)
inp <- list(nseasons=4, splineorder=3)
inp$timeC <- seq(0, 30-1/inp$nseasons, by=1/inp$nseasons)
inp$timeI <- seq(0, 30-1/inp$nseasons, by=1/inp$nseasons)
inp$ini <- list(logK=log(100), logm=log(20), logq=log(1),
                logbkfrac=log(1), logsdf=log(0.4), logF0=log(0.5),
                logphi=log(c(0.05, 0.1, 1.8)))
seasonsim <- sim.spict(inp)
plotspict.data(seasonsim)
```



The data plot shows clear seasonality in the catches. To simulate seasonal data using the coupled SDE approach `seasontype` must be set to 2 and `nseasons` to 4.

```
set.seed(432)
inp <- list(nseasons=4, seasontype=2)
inp$timeC <- seq(0, 30-1/inp$nseasons, by=1/inp$nseasons)
inp$timeI <- seq(0, 30-1/inp$nseasons, by=1/inp$nseasons)
inp$ini <- list(logK=log(100), logm=log(20), logq=log(1),
                logbkfrac=log(1), logsdf=log(0.4), logF0=log(0.5))
seasonsim2 <- sim.spict(inp)
plotspict.data(seasonsim2)
```



Estimation using quarterly data

Catch information available in sub-annual aggregations, e.g. quarterly catch, can be used to estimate the seasonal pattern of the fishing mortality. The user can choose between two types of seasonality by setting `seasontype` to 1, 2 or 3:

1. using cyclic B-splines.
2. using coupled stochastic differential equations (SDEs).
3. combination of a cyclic spline (fixed seasonal pattern) and mean-zero autoregressive process

Technical description of the first two season types is found in Pedersen and Berg (2017). The third is described by the following equation:

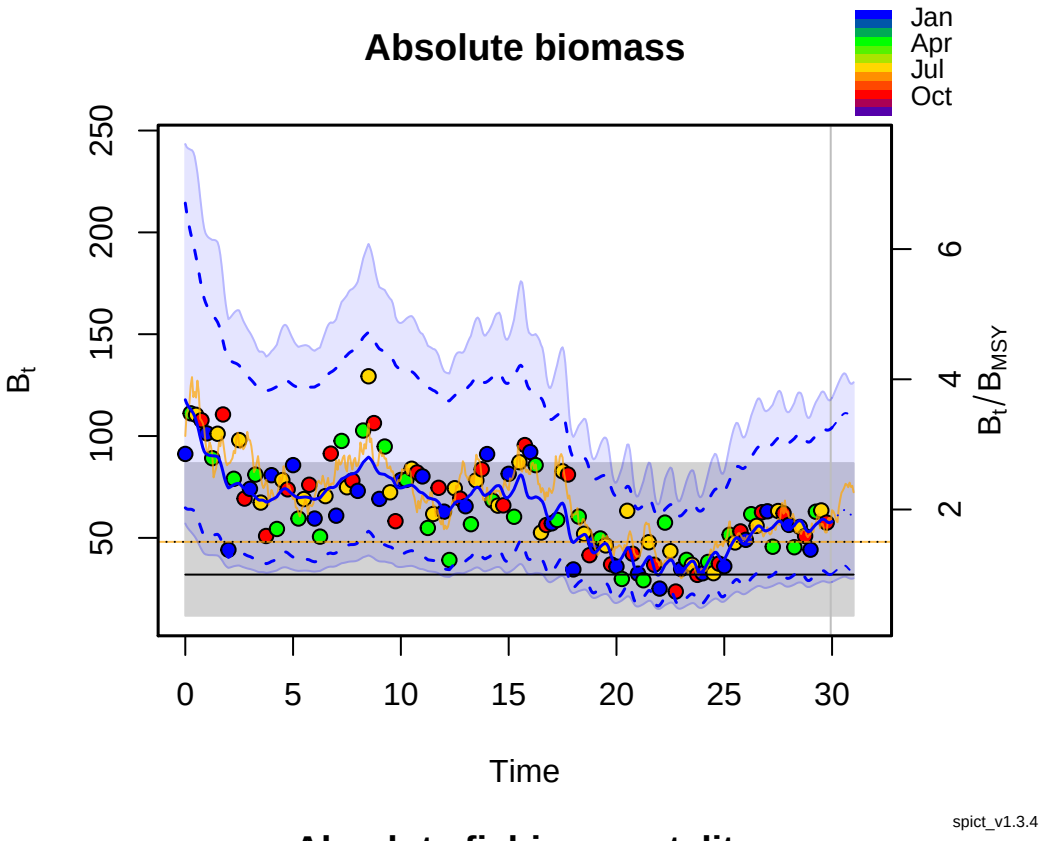
$$F_t = S_t G_t \exp(A_{q(t)}) \tag{1}$$

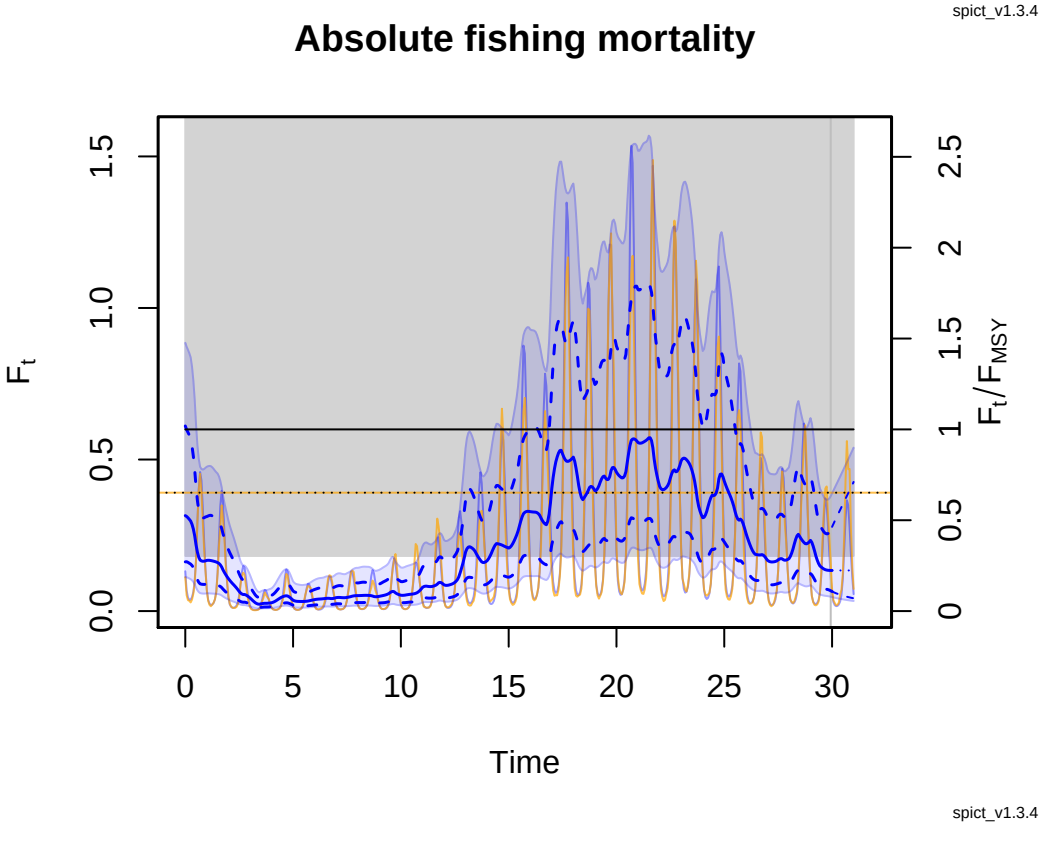
$$d\log G_t = \sigma_F dV_t \tag{2}$$

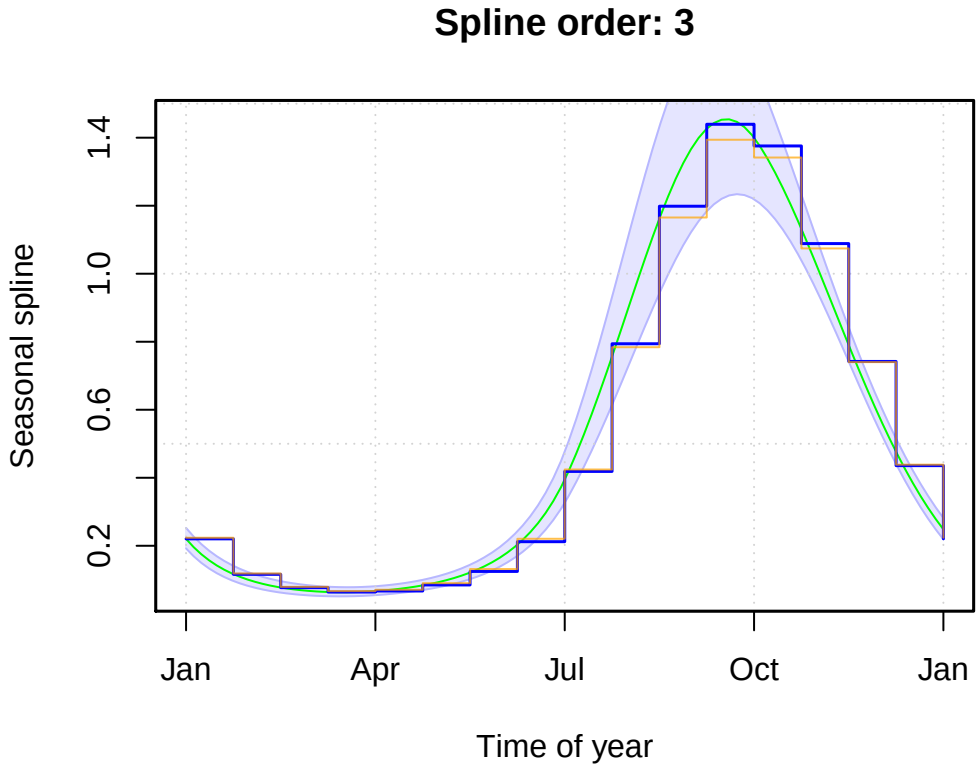
- S_t is a cyclic spline and G_t is diffusion process.
- $A_{q(t)} = \varphi_A A_{q(t-1)} + \varepsilon_{A,q(t)}$ is a discrete time mean zero autoregressive process , and q maps t to a quarter.

Here, an example of a spline-based model fitted to quarterly data simulated in section is shown

```
seasonres <- fit.spict(seasonsim)
plotspict.biomass(seasonres)
plotspict.f(seasonres, qlegend=FALSE)
plotspict.season(seasonres)
```


Absolute biomass
Jan
Apr
Jul
Oct
B_t
B_t/B_{MSY}
50
100
150
200
250
2
4
6
0
5
10
15
20
25
30
Time
spict_v1.3.4

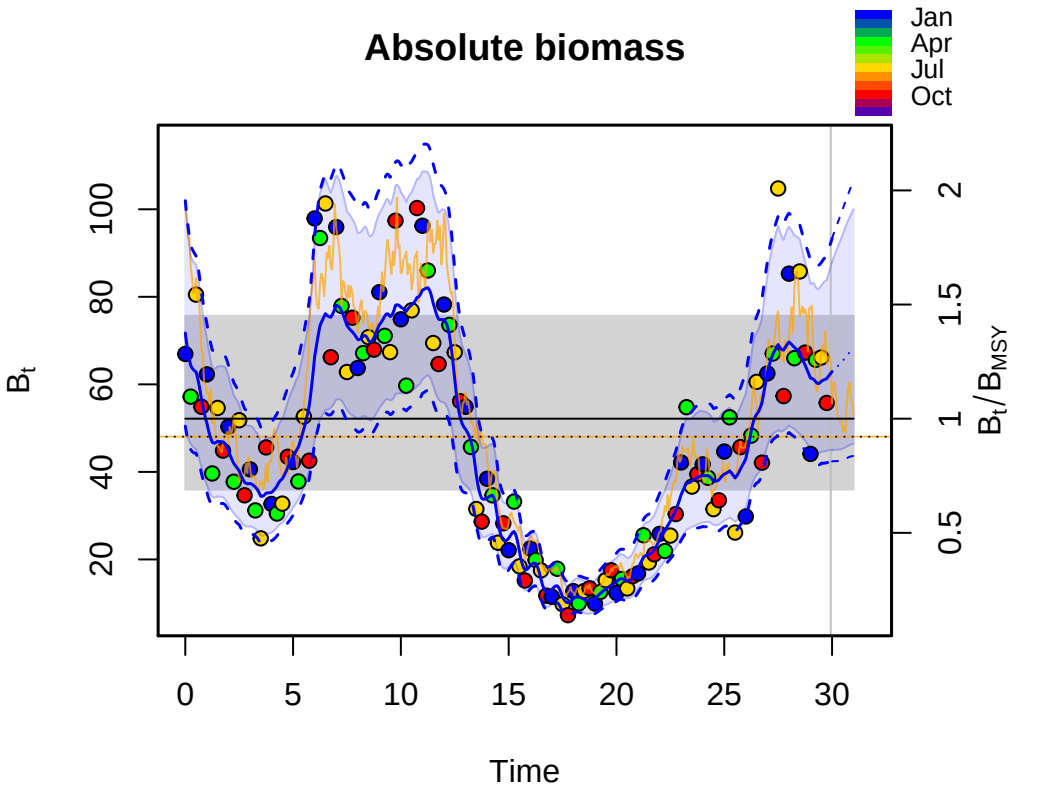

Absolute fishing mortality
F_t
F_t/F_{MSY}
0.0
0.5
1.0
1.5
0
0.5
1
1.5
2
2.5
0
5
10
15
20
25
30
Time
spict_v1.3.4

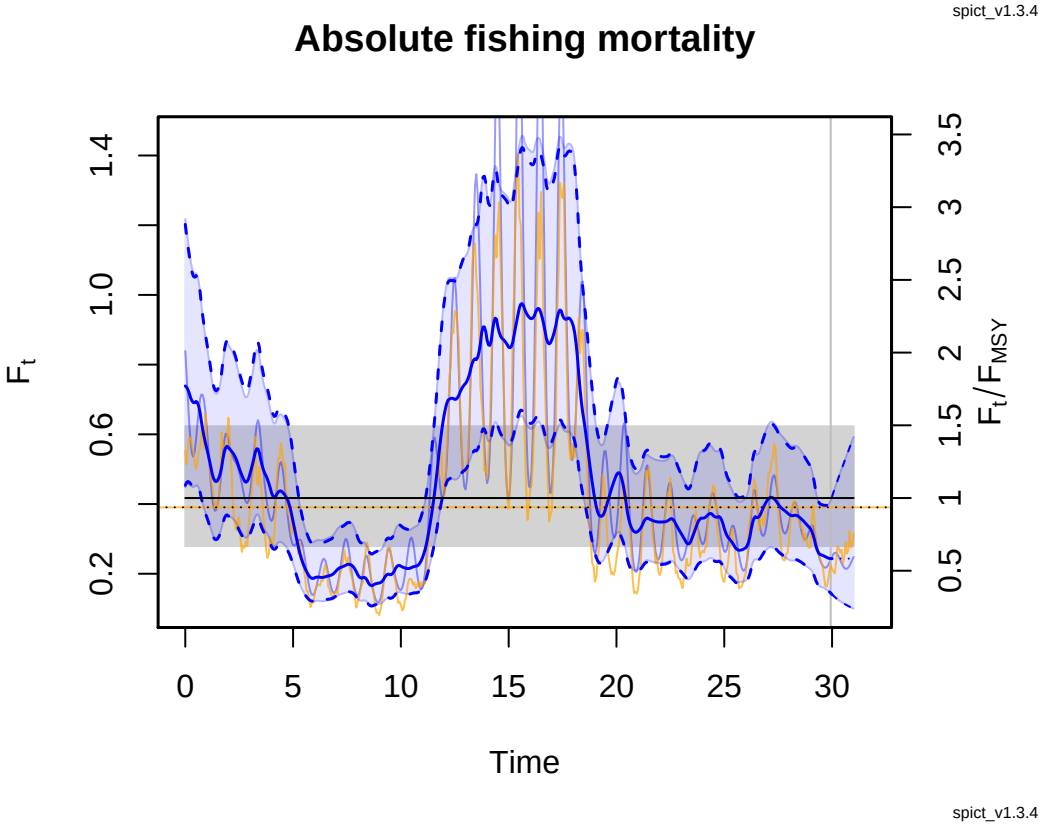


The model is able to estimate the seasonal variation in fishing mortality as seen both in the plot of `F` and in the plot of the estimated spline, where blue is the estimated spline, orange is the true spline, and green is the spline if time were truly continuous (it is discretised with the Euler steps shown by the blue line).

To fit the coupled SDE model run

```
seasonres2 <- fit.spict(seasonsim2)
sumspict.parest(seasonres2)
             estimate  true       cilow       ciupp true.in.ci    log.est
   alpha    1.1638022  -9.0  0.67490220   2.0068620         -9  0.1516924
   beta     0.4961819  -9.0  0.30347194   0.8112662         -9 -0.7008126
   r        0.8526429  -9.0  0.26693387   2.7235209         -9 -0.1594144
   rc       0.8468346  -9.0  0.56021208   1.2801023         -9 -0.1662499
   rold     0.8411048  -9.0  0.42323803   1.6715352         -9 -0.1730390
   m       22.6501792  20.0 18.08388206  28.3694960          1  3.1201678
   K      106.7057712 100.0 61.53726693 185.0280680          1  4.6700753
   q        1.1111248   1.0  0.81096947   1.5223735          1  0.1053729
   n        2.0137179   2.0  0.84868643   4.7780423          1  0.6999827
   sdb      0.1619837   0.2  0.10231723   0.2564447          1 -1.8202597
   sdu      0.1259182   0.1  0.07102428   0.2232391          1 -2.0721227
   sdf      0.3565349   0.4  0.25805882   0.4925897          1 -1.0313232
   sdi      0.1885170   0.2  0.15969165   0.2225454          1 -1.6685673
   sdc      0.1769062   0.2  0.13548557   0.2309899          1 -1.7321358
   lambda   0.1032635   0.1  0.02267847   0.4701972          1 -2.2704712
plotspict.biomass(seasonres2)
plotspict.f(seasonres2, qlegend=FALSE)
```

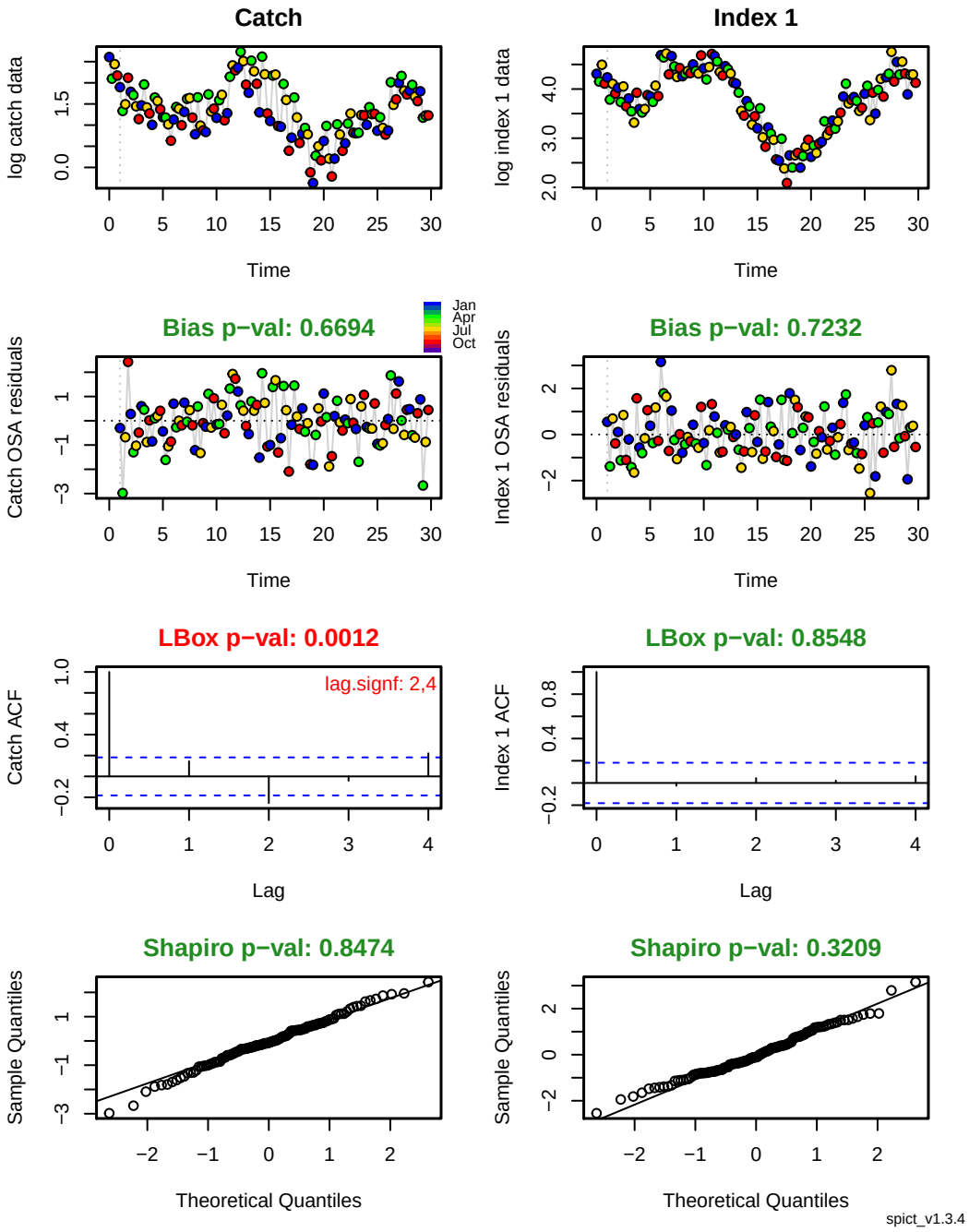




Two parameters related to the coupled SDEs are estimated (`sdu` and `lambda`) as evident from the summary of estimated parameters. In the plot of fishing mortality it is noted that the amplitude of the seasonal pattern

varies over time. This is a property of the coupled SDE model, which is not possible to obtain with the spline based seasonal model. The spline based model has a fixed amplitude and phases, which will lead to biased estimates and autocorrelation in residuals if in reality the seasonal pattern shifts a bit. This is illustrated by fitting a spline based model to data generated with a coupled SDE model

```
inp2 <- list(obsC=seasonsim2$obsC, obsI=seasonsim2$obsI,
             timeC=seasonsim2$timeC, timeI=seasonsim2$timeI,
             seasontype=1, true=seasonsim2$true)
rep2 <- fit.spict(inp2)
rep2 <- calc.osa.resid(rep2)
plotspict.diagnostic(rep2)
```



From the diagnostics it is clear that autocorrelation is present in the catch residuals.

Setting initial parameter values

Initial parameter values used as starting guess of the optimiser can be set using `inp$ini`. For example, to specify the initial value of `logK` set

```
inp <- pol$albacore
inp$ini$logK <- log(100)
```

This procedure generalises to all other model parameters. If initial values are not specified they are set to default values. To see the default initial value of a parameter, here `logK`, run

```
inp <- check.inp(pol$albacore)
inp$ini$logK
   [1] 5.010635
```

This can also be done posterior to fitting the model by printing `res$inp$ini$logK`.

Checking robustness to initial parameter values

It is prudent to check that the same parameter estimates are obtained if using different initial values. If the optimum of the objective function is poorly defined, i.e. possibly containing multiple optima, it is possible that different parameter estimates will be returned depending on the initial values. To check whether this is the case run

```
set.seed(123)
check.ini(pol$albacore, ntrials=4)
   Checking sensitivity of fit to initial parameter values...
    Trial 1 ... model fitted!
    Trial 2 ... model fitted!
    Trial 3 ... model fitted!
    Trial 4 ... model fitted!
   $propchng
            logm  logK  logq  logn logsdb logsdf logsdi logsdc
   Trial 1 -1.41  0.26 -0.12 -2.75  -1.26   1.30  -0.08  -1.12
   Trial 2  0.34 -0.04  0.62  0.34  -0.51  -0.21   1.14  -1.14
   Trial 3 -1.69 -0.42 -0.23 -3.26  -1.11  -0.55  -0.40  -1.41
   Trial 4  1.03  0.19  0.06 -0.68   0.60   1.01  -1.32  -1.15

   $inimat
           Distance  logn logK logm  logq logsdb logsdf logsdi logsdc
   Basevec     0.00  0.69 5.01 3.41 -0.64  -1.61  -1.61  -1.61  -1.61
   Trial 1     4.22 -0.29 6.34 2.99  1.12   0.42  -3.70  -1.48   0.20
   Trial 2     3.48  0.93 4.81 5.51 -0.86  -0.79  -1.27  -3.44   0.23
   Trial 3     4.52 -0.48 2.90 2.61  1.45   0.18  -0.72  -0.96   0.67
   Trial 4     3.64  1.41 5.97 3.61 -0.21  -2.58  -3.23   0.52   0.24

   $resmat
           Distance     m      K    q    n  sdb  sdf  sdi  sdc
   Basevec        0 22.58 201.48 0.35 0.69 0.01 0.37 0.11 0.04
   Trial 1        0 22.58 201.48 0.35 0.69 0.01 0.37 0.11 0.04
   Trial 2        0 22.58 201.48 0.35 0.69 0.01 0.37 0.11 0.04
   Trial 3        0 22.58 201.48 0.35 0.69 0.01 0.37 0.11 0.04
   Trial 4        0 22.58 201.48 0.35 0.69 0.01 0.37 0.11 0.04
   $obsC
    [1] 15.9 25.7 28.5 23.7 25.0 33.3 28.2 19.7 17.5 19.3 21.6 23.1 22.5 22.5 23.6
   [16] 29.1 14.4 13.2 28.4 34.6 37.5 25.9 25.3
```

```
$timeC
 [1] 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981
[16] 1982 1983 1984 1985 1986 1987 1988 1989

$obsI
 [1] 61.89 78.98 55.59 44.61 56.89 38.27 33.84 36.13 41.95 36.63 36.33 38.82
[13] 34.32 37.64 34.01 32.16 26.88 36.61 30.07 30.75 23.36 22.36 21.91

$timeI
 [1] 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981
[16] 1982 1983 1984 1985 1986 1987 1988 1989

$check.ini
$check.ini$propchng
          logm  logK  logq  logn logsdb logsdf logsdi logsdc
Trial 1 -1.41  0.26 -0.12 -2.75  -1.26   1.30  -0.08  -1.12
Trial 2  0.34 -0.04  0.62  0.34  -0.51  -0.21   1.14  -1.14
Trial 3 -1.69 -0.42 -0.23 -3.26  -1.11  -0.55  -0.40  -1.41
Trial 4  1.03  0.19  0.06 -0.68   0.60   1.01  -1.32  -1.15

$check.ini$inimat
        Distance  logn logK logm  logq logsdb logsdf logsdi logsdc
Basevec     0.00  0.69 5.01 3.41 -0.64  -1.61  -1.61  -1.61  -1.61
Trial 1     4.22 -0.29 6.34 2.99  1.12   0.42  -3.70  -1.48   0.20
Trial 2     3.48  0.93 4.81 5.51 -0.86  -0.79  -1.27  -3.44   0.23
Trial 3     4.52 -0.48 2.90 2.61  1.45   0.18  -0.72  -0.96   0.67
Trial 4     3.64  1.41 5.97 3.61 -0.21  -2.58  -3.23   0.52   0.24

$check.ini$resmat
        Distance     m      K    q    n  sdb  sdf  sdi  sdc
Basevec        0 22.58 201.48 0.35 0.69 0.01 0.37 0.11 0.04
Trial 1        0 22.58 201.48 0.35 0.69 0.01 0.37 0.11 0.04
Trial 2        0 22.58 201.48 0.35 0.69 0.01 0.37 0.11 0.04
Trial 3        0 22.58 201.48 0.35 0.69 0.01 0.37 0.11 0.04
Trial 4        0 22.58 201.48 0.35 0.69 0.01 0.37 0.11 0.04
```

The argument `ntrials` set the number of different initial values to test for. To keep it simple only few trials are generated here, however for real data cases more should be used, say 30. The `propchng` contains the proportional change of the new randomly generated initial value relative to the base initial value, `inimat` contains the new randomly generated initial values, and `resmat` contains the resulting parameter estimates and a distance from the estimated parameter vector to the base parameter vector. The distance should preferably be close to zero. If that is not the case further investigation is required, i.e. inspection of objective function values, differences in results and residual diagnostics etc. should be performed. The example shown here looks fine in that all converged runs return the same parameter estimates. One trial did not converge, however non-converging trials are to some extent expected as the initial parameters are generated independently from a wide uniform distribution and may thus by chance be very inappropriately chosen.

Phases and how to fix parameters

The package has the ability to estimate parameters in phases. Users familiar with AD model builder will know that this means that some parameters are held constant in phase 1, some are then released and estimated in phase 2, more are released in phase 3 etc. until all parameters are estimated. Per default all parameters are estimated in phase 1. As an example the standard deviation on the biomass process, `logsdb`, is estimated in phase 2:

```
inp <- pol$albacore
inp$phases$logsdb <- 2
res <- fit.spict(inp)
   Estimating - phase 1
   Estimating - phase 2
```

Phases can also be used to fix parameters to their initial value by setting the phase to `-1`. For example

```
inp <- pol$albacore
inp$phases$logsdb <- -1
inp$ini$logsdb <- log(0.1)
res <- fit.spict(inp)
summary(res)
   Convergence: 0  MSG: relative convergence (4)
   Objective function at optimum: 5.8647408
   Euler time step (years):  1/16 or 0.0625
   Nobs C: 23,  Nobs I1: 23

   Priors
        logn  ~  dnorm[log(2), 2^2]
    logalpha  ~  dnorm[log(1), 2^2]
     logbeta  ~  dnorm[log(1), 2^2]

   Fixed parameters
        fixed.value
    sdb         0.1

   Model parameter estimates w 95% CI
              estimate      cilow       ciupp    log.est
    alpha    1.0503613  0.6791518   1.6244657  0.0491342
    beta     0.1713918  0.0256773   1.1440119 -1.7638031
    r        0.3471988  0.1465244   0.8227093 -1.0578579
    rc       1.7791120  0.2676089  11.8278543  0.5761143
    rold     0.5694637  0.0689479   4.7033880 -0.5630603
    m       27.4846393 18.9163558  39.9339814  3.3136273
    K      144.5708271 82.7912269 252.4509520  4.9737695
    q        0.4966500  0.2541097   0.9706881 -0.6998696
    n        0.3903057  0.0392947   3.8768222 -0.9408250
    sdf      0.3738950  0.2594014   0.5389234 -0.9837802
    sdi      0.1050361  0.0679152   0.1624466 -2.2534509
    sdc      0.0640825  0.0113802   0.3608517 -2.7475832

   Deterministic reference points (Drp)
           estimate      cilow      ciupp    log.est
    Bmsyd 30.897032  6.2838317 151.917910  3.4306601
    Fmsyd  0.889556  0.1338045   5.913927 -0.1170328
    MSYd  27.484639 18.9163558  39.933981  3.3136273
   Stochastic reference points (Srp)
            estimate      cilow      ciupp    log.est  rel.diff.Drp
    Bmsys 30.8170236  6.3481137 149.601754  3.4280673 -0.0025962352
    Fmsys  0.8901021  0.1349377   5.871465 -0.1164191  0.0006135517
    MSYs  27.4303399 18.8037379  40.014573  3.3116497 -0.0019795390

   States w 95% CI (inp$msytype: s)
```

```
                    estimate      cilow     ciupp    log.est
  B_1989.94       44.7251083 21.9035625 91.324656  3.8005351
  F_1989.94        0.5737431  0.2575318  1.278216 -0.5555735
  B_1989.94/Bmsy   1.4513117  0.3383802  6.224673  0.3724678
  F_1989.94/Fmsy   0.6445813  0.1046979  3.968419 -0.4391543

 Predictions w 95% CI (inp$msytype: s)
                  prediction      cilow     ciupp    log.est
  B_1991.00       45.2794307 20.9338294 97.938452  3.8128529
  F_1991.00        0.5737434  0.1907854  1.725402 -0.5555730
  B_1991.00/Bmsy   1.4692993  0.3184821  6.778530  0.3847856
  F_1991.00/Fmsy   0.6445816  0.0900490  4.613992 -0.4391539
  Catch_1990.00   25.8407665 15.9509294 41.862464  3.2519533
  E(B_inf)        45.9908947         NA        NA  3.8284434
```

Priors

SPiCT is a generalisation of previous surplus production models in the sense that stochastic noise is included in both observation and state processes of both fishing and biomass. Estimating all model parameters is only possible if data contain sufficient information, which may not be the case for short time series or time series with limited contrast. The basic data requirements of the model are limited to only catch and biomass index time series. More information may be available, which can be used to improve the model fit. This is particularly advantageous if the model is not able to converge with only catch and index time series. Additional information can then be included in the fit via prior distributions for model parameters.

Default priors and how to disable them

Quantities that are traditionally difficult to estimate are `logn`, and the noise ratios `logalpha` and `logbeta` where `logalpha = logsdi - logsdb` and `logbeta = logsdc - logsdf`, respectively. Therefore, to generally stabilise estimation default semi-informative priors are imposed on these quantities that inhibit them from taking extreme and unrealistic values. If informative data are available these priors should have limited effect on results, if informative data are not available estimates will reduce to the priors.

If informative data are available and the default priors therefore are unwanted they can be disabled using

```
inp <- pol$albacore
inp$priors$logn <- c(1, 1, 0)
inp$priors$logalpha <- c(1, 1, 0)
inp$priors$logbeta <- c(1, 1, 0)
fit.spict(inp)
   Convergence: 0  MSG: relative convergence (4)
   Objective function at optimum: 5.0598269
   Euler time step (years):  1/16 or 0.0625
   Nobs C: 23,  Nobs I1: 23

   No priors are used

   Model parameter estimates w 95% CI
             estimate       cilow        ciupp    log.est
    alpha   39.0515853   0.0402350 3.790301e+04  3.6648835
    beta     0.0245072   0.0000265 2.269261e+01 -3.7087885
    r        0.1955746   0.0313372 1.220576e+00 -1.6318133
    rc       1.2604846   0.0097765 1.625147e+02  0.2314962
    rold     0.2835716   0.0024028 3.346585e+01 -1.2602908
    m       24.3112585  12.0982017 4.885332e+01  3.1909396
```

```
    K      210.4517437 123.9786100 3.572385e+02  5.3492564
    q        0.3842439   0.2070704 7.130106e-01 -0.9564777
    n        0.3103166   0.0004170 2.309135e+02 -1.1701624
    sdb      0.0028039   0.0000029 2.728603e+00 -5.8767274
    sdf      0.3809117   0.2827808 5.130960e-01 -0.9651878
    sdi      0.1094986   0.0812449 1.475776e-01 -2.2118440
    sdc      0.0093351   0.0000103 8.475515e+00 -4.6739763

   Deterministic reference points (Drp)
            estimate      cilow      ciupp   log.est
    Bmsyd 38.5744642  0.5969708 2492.56641  3.652591
    Fmsyd  0.6302423  0.0048882   81.25737 -0.461651
    MSYd  24.3112585 12.0982017   48.85332  3.190940
   Stochastic reference points (Srp)
            estimate      cilow      ciupp    log.est  rel.diff.Drp
    Bmsys 38.5743443  0.5969804 2492.51087  3.6525874 -3.108371e-06
    Fmsys  0.6302434  0.0048883   81.25688 -0.4616493  1.767619e-06
    MSYs  24.3112241 12.0980500   48.85379  3.1909381 -1.414967e-06

   States w 95% CI (inp$msytype: s)
                     estimate      cilow      ciupp    log.est
    B_1989.94      52.6709906 29.3005771 94.6818639  3.9640648
    F_1989.94       0.4858021  0.2429955  0.9712266 -0.7219539
    B_1989.94/Bmsy  1.3654410  0.0268836 69.3520164  0.3114774
    F_1989.94/Fmsy  0.7708167  0.0077207 76.9561969 -0.2603047

   Predictions w 95% CI (inp$msytype: s)
                   prediction      cilow     ciupp    log.est
    B_1991.00      51.2819174 28.0678775 93.695544  3.9373382
    F_1991.00       0.4858023  0.1724933  1.368192 -0.7219535
    B_1991.00/Bmsy  1.3294307  0.0231015 76.505211  0.2847508
    F_1991.00/Fmsy  0.7708171  0.0072430 82.032615 -0.2603042
    Catch_1990.00  25.2158724 15.5407662 40.914342  3.2274737
    E(B_inf)       49.5039157         NA        NA  3.9020518
```

The model is able to converge without priors, however the estimates of `alpha`, `beta` and `n` are very uncertain indicating that limited information is available about these parameters.

Setting a prior

The model parameters to which priors can be applied can be listed using

```
list.possible.priors()
    [1] "logn"      "logalpha"  "logbeta"   "logr"      "logK"      "logm"
    [7] "logq"      "logqf"     "logbkfrac" "logB"      "logF"      "logBBmsy"
   [13] "logFFmsy"  "logsdb"    "logsdf"    "logsdi"    "logsde"    "logsdc"
   [19] "logsdm"    "logpsi"    "mu"        "BmsyBO"    "logngamma"
```

A prior is set using

```
inp <- pol$albacore
inp$priors$logK <- c(log(300), 2, 1)
fit.spict(inp)
   Convergence: 0  MSG: relative convergence (4)
   Objective function at optimum: 3.6972089
```

```
Euler time step (years):  1/16 or 0.0625
Nobs C: 23,  Nobs I1: 23

Priors
     logK  ~  dnorm[log(300), 2^2]
     logn  ~  dnorm[log(2), 2^2]
 logalpha  ~  dnorm[log(1), 2^2]
  logbeta  ~  dnorm[log(1), 2^2]

Model parameter estimates w 95% CI
          estimate       cilow       ciupp    log.est
alpha    8.5541387   1.2276036  59.6066094  2.1464152
beta     0.1213066   0.0180837   0.8137337 -2.1094339
r        0.2543932   0.0999823   0.6472739 -1.3688740
rc       0.7408800   0.1423249   3.8566923 -0.2999166
rold     0.8120644   0.0018383 358.7242279 -0.2081756
m       22.5701062  17.0283143  29.9154506  3.1166263
K      202.2161203 138.6995674 294.8196601  5.3093370
q        0.3499363   0.1930335   0.6343738 -1.0500041
n        0.6867327   0.0623506   7.5637067 -0.3758102
sdb      0.0127865   0.0018399   0.0888614 -4.3593675
sdf      0.3672884   0.2672936   0.5046913 -1.0016080
sdi      0.1093773   0.0808912   0.1478947 -2.2129522
sdc      0.0445545   0.0073418   0.2703826 -3.1110419

Deterministic reference points (Drp)
       estimate      cilow      ciupp    log.est
Bmsyd 60.92783 15.2913099 242.765396  4.1096901
Fmsyd  0.37044  0.0711624   1.928346 -0.9930638
MSYd  22.57011 17.0283143  29.915451  3.1166263
Stochastic reference points (Srp)
        estimate      cilow      ciupp   log.est  rel.diff.Drp
Bmsys 60.9201702 15.2915066 242.701210  4.109564 -1.257930e-04
Fmsys  0.3704521  0.0711556   1.928658 -0.993031  3.268226e-05
MSYs  22.5680073 17.0227442  29.919674  3.116533 -9.300394e-05

States w 95% CI (inp$msytype: s)
                 estimate      cilow       ciupp    log.est
B_1989.94      56.9259673 30.1894531 107.3409889  4.0417516
F_1989.94       0.4446516  0.2131851   0.9274336 -0.8104642
B_1989.94/Bmsy  0.9344355  0.2946906   2.9630044 -0.0678127
F_1989.94/Fmsy  1.2002944  0.2841452   5.0703177  0.1825668

Predictions w 95% CI (inp$msytype: s)
                prediction      cilow      ciupp    log.est
B_1991.00      54.5233252 27.9286098 106.442569  3.9986286
F_1991.00       0.4446518  0.1564541   1.263727 -0.8104637
B_1991.00/Bmsy  0.8949963  0.2487008   3.220811 -0.1109357
F_1991.00/Fmsy  1.2002950  0.2373731   6.069382  0.1825674
Catch_1990.00  24.7355104 15.3286452  39.915170  3.2082399
E(B_inf)       50.1633799         NA         NA  3.9152853
```

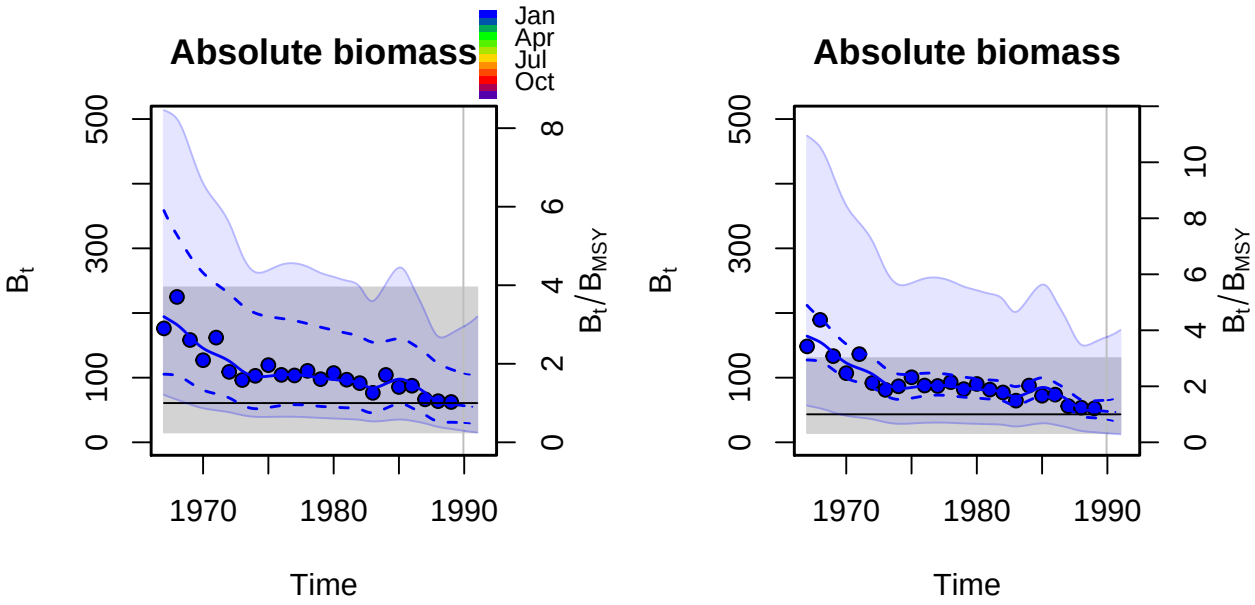
This imposes a Gaussian prior on `logK` with mean log(300) and standard deviation 2. The third entry indicates that the prior is used (1 means use, 0 means do not use). From the summary it is evident that the

default priors were also imposed.

Priors on random effects

Priors can be applied to random effects of the model, i.e. `logB`, `logF`, `logBBmsy`, (which is $\log(B/Bmsy)$) `logFFmsy` (which is $\log(F/Fmsy)$). An additional argument is required to specify these priors

```
inp <- pol$albacore
inp$priors$logB <- c(log(80), 0.1, 1, 1980)
par(mfrow=c(1, 2), mar=c(5, 4.1, 3, 4))
plotspict.biomass(fit.spict(pol$albacore), ylim=c(0, 500))
plotspict.biomass(fit.spict(inp), qlegend=FALSE, ylim=c(0, 500))
```



This imposes a Gaussian prior on `logB` with mean log(80), standard deviation 0.1 (very informative), the third entry in the vector indicates that the prior is used, the fourth entry indicates the year to which the prior should be applied, here 1980.

It is clear from the plots that the prior influences the results significantly. Furthermore, it is not only the biomass in the year 1980 that is affected, but the information propagates forward and backward because all estimates are correlated. In reality such an informative prior is rarely available, however it may be possible to derive information about the absolute biomass from acoustic survey and swept area estimates. It is, however, critical that the standard deviation used reflects the quality of the information.

Setting priors on the standard deviation of multiple indices

It is possible to set a prior for on some or all noise terms of multiple biomass indices

```
set.seed(123)
inp <- list(timeC=pol$albacore$timeC, obsC=pol$albacore$obsC)
inp$timeI <- list(pol$albacore$timeI, pol$albacore$timeI[10:23]+0.25)
inp$obsI <- list()
inp$obsI[[1]] <- pol$albacore$obsI * exp(rnorm(23, sd=0.1)) # Index 1
inp$obsI[[2]] <- 10*pol$albacore$obsI[10:23] # Index 2
inp$priors$logsdi <- list(c(1, 1, 0),          # No prior for index 1
                          c(log(0.1), 0.2, 1)) # Set a prior on index 2
res <- fit.spict(inp)
```

```
sumspict.priors(res)

   Priors
        logn  ~  dnorm[log(2), 2^2]
    logalpha  ~  dnorm[log(1), 2^2]
     logbeta  ~  dnorm[log(1), 2^2]
     logsdi2  ~  dnorm[log(0.1), 0.2^2]
```

Fixing parameters using priors

Model parameters can be fixed using `phases` as described previously. This technique can, however, only be used to fix model parameters and therefore not derived quantities such as `logalpha`, `logr` (which is derived from `logK`, `logm` and `logn`). Fixing a parameter can be regarded as imposing an highly informative prior to the parameter

```
inp <- pol$albacore
inp$priors$logn <- c(log(2), 1e-3)
inp$priors$logalpha <- c(log(1), 1e-3)
inp$priors$logbeta <- c(log(1), 1e-3)
fit.spict(inp)
   Convergence: 0  MSG: relative convergence (4)
   Objective function at optimum: -13.3777234
   Euler time step (years):  1/16 or 0.0625
   Nobs C: 23,  Nobs I1: 23

   Priors
        logn  ~  dnorm[log(2), 0.001^2] (fixed)
    logalpha  ~  dnorm[log(1), 0.001^2] (fixed)
     logbeta  ~  dnorm[log(1), 0.001^2] (fixed)

   Model parameter estimates w 95% CI
              estimate      cilow       ciupp    log.est
    alpha    1.0000039  0.9980458   1.0019658  0.0000039
    beta     0.9999976  0.9980395   1.0019595 -0.0000024
    r        0.5046213  0.1875581   1.3576733 -0.6839470
    rc       0.5046222  0.1875581   1.3576783 -0.6839452
    rold     0.5046231  0.1875574   1.3576886 -0.6839434
    m       22.0086910 17.0613921  28.3905603  3.0914374
    K      174.4568979 74.7554140 407.1305020  5.1616777
    q        0.3549422  0.1278983   0.9850324 -1.0358004
    n        1.9999964  1.9960802   2.0039202  0.6931454
    sdb      0.0966006  0.0704056   0.1325417 -2.3371705
    sdf      0.2102260  0.1562925   0.2827708 -1.5595724
    sdi      0.0966010  0.0704060   0.1325419 -2.3371666
    sdc      0.2102255  0.1562923   0.2827698 -1.5595748

   Deterministic reference points (Drp)
            estimate      cilow       ciupp   log.est
    Bmsyd 87.2283875 37.3776360 203.5653509  4.468530
    Fmsyd  0.2523111  0.0937791   0.6788391 -1.377092
    MSYd  22.0086910 17.0613921  28.3905603  3.091437
   Stochastic reference points (Srp)
            estimate      cilow      ciupp   log.est rel.diff.Drp
    Bmsys 86.1721720 37.1707123 199.771346  4.456347  -0.01225704
```

```
   Fmsys  0.2500268  0.0921861   0.678122 -1.386187  -0.00913625
   MSYs  21.5429415 16.5473168  28.046742  3.070048  -0.02161959

  States w 95% CI (inp$msytype: s)
                    estimate      cilow      ciupp    log.est
   B_1989.94      54.8231823 17.4448184 172.290777  4.0041131
   F_1989.94       0.4313749  0.1433932   1.297720 -0.8407778
   B_1989.94/Bmsy  0.6362052  0.3800215   1.065090 -0.4522342
   F_1989.94/Fmsy  1.7253144  0.9443603   3.152091  0.5454093

  Predictions w 95% CI (inp$msytype: s)
                  prediction      cilow      ciupp    log.est
   B_1991.00      50.1747640 14.1658422 177.716715  3.9155122
   F_1991.00       0.4313751  0.1324939   1.404476 -0.8407773
   B_1991.00/Bmsy  0.5822618  0.2912669   1.163980 -0.5408351
   F_1991.00/Fmsy  1.7253153  0.8254120   3.606336  0.5454098
   Catch_1990.00  22.6023879 14.8441339  34.415476  3.1180556
   E(B_inf)       20.7048449         NA         NA  3.0303677
```

The summary indicates that the priors are so informative that the quantities are essentially fixed. It is also noted that the estimates of these quantities are very close to the mean of their respective priors.

Pitfalls when fixing parameters and specifying priors

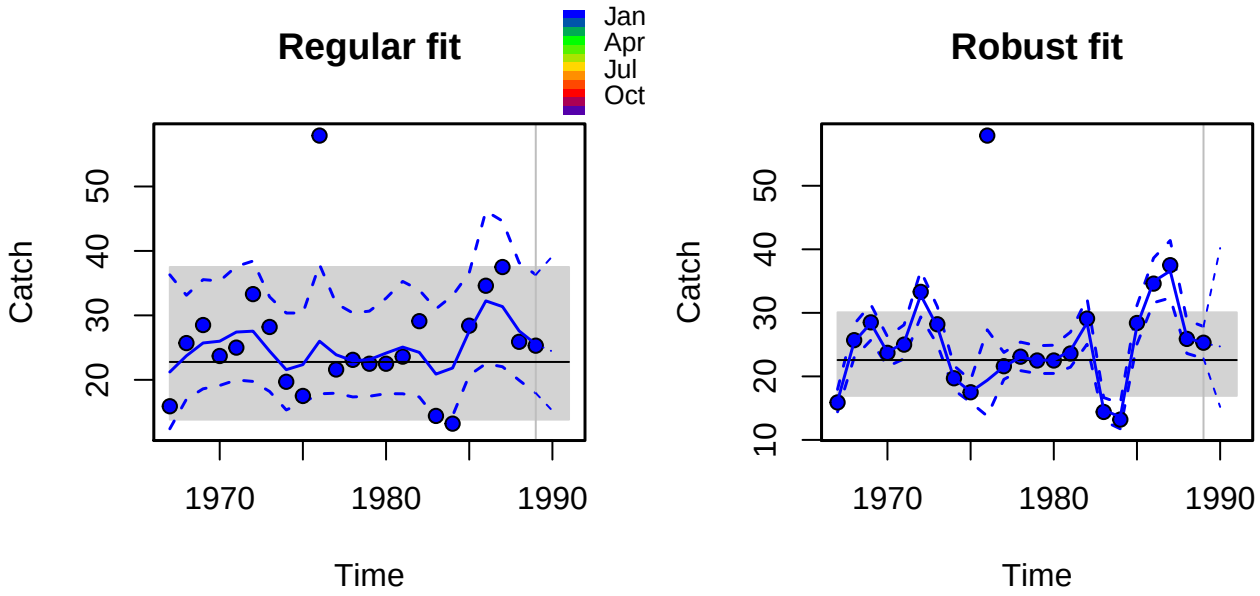
Particular caution is required when fixing a parameter that is highly correlated with other parameters because this will to some extent restrict the estimates of the correlated parameters. This could also be a problem when specifying priors depending on the amount of a priori information available.

Robust estimation (reducing influence of extreme observations)

The presence of extreme observations may inflate estimates of observation noise and increase the general uncertainty of the fit. To reduce this effect it is possible to apply a robust estimation scheme, which is less sensitive to extreme observations. An example with an extreme observation in the catch series is

```
inp <- pol$albacore
inp$obsC[10] <- 3*inp$obsC[10]
res1 <- fit.spict(inp)
inp$robflagc <- 1
res2 <- fit.spict(inp)
sumspict.parest(res2)
             estimate        cilow        ciupp    log.est
   alpha    8.01383384   1.14064157  56.30299170  2.0811693
   beta     0.13903418   0.02002425   0.96535465 -1.9730355
   r        0.25685421   0.10125071   0.65159136 -1.3592466
   rc       0.73334983   0.13978955   3.84722582 -0.3101324
   rold     0.85759781   0.00140610 523.06000718 -0.1536200
   m       22.57344297  16.94359765  30.07391569  3.1167741
   K      202.06196124 137.06067473 297.89023191  5.3085744
   q        0.34766877   0.18846112   0.64137140 -1.0565050
   n        0.70049573   0.06408928   7.65641704 -0.3559670
   sdb      0.01371149   0.00196485   0.09568409 -4.2895209
   sdf      0.37086361   0.26568561   0.51767882 -0.9919209
   sdi      0.10988163   0.08106901   0.14893450 -2.2083516
   sdc      0.05156272   0.00843494   0.31520251 -2.9649564
   pp       0.95304961   0.72886830   0.99351826  3.0105755
```

```
   robfac  20.83563432   2.67330869 236.13437305  2.9874800
par(mfrow=c(1, 2))
plotspict.catch(res1, main='Regular fit')
plotspict.catch(res2, qlegend=FALSE, main='Robust fit')
```



It is evident from the plot that the presence of the extreme catch observation generally inflates the uncertainty of the estimated catches, while the robust fit is less sensitive. Robust estimation can be applied to index and effort data using `robflagi` and `robflage` respectively.

Robust estimation is implemented using a mixture of light-tailed and a heavy-tailed Gaussian distribution as described in Pedersen and Berg (2017). This entails two additional parameters (`pp` and `robfac`) that require estimation. This may not always be possible given the increased model complexity. In such cases these parameters should be fixed by setting their phases to `-1`.

Forecasting and management scenarios

All quantities estimated by SPiCT can be forecasted into the future. By default, the forecast period starts one time step after the last observation, which can be defined by the last index observation or by the end of the last catch or effort interval. Forecasting catch, fishing mortality, and biomass is a crucial step in fisheries management that allows determining future yields and risk of overfishing under current or alternative management scenarios. The following provides details to the forecasting and management functionality of SPiCT and includes examples based on the South Atlantic albacore data set of Polacheck, Hilborn, and Punt (1993), which contains catch and commercial CPUE data from 1967 to 1990.

Forecasting

Producing short-term forecasts entails minimal additional computing time and is part of the model fitting of SPiCT. The default settings can be altered by specifying the start and end of the forecast interval with the argument `maninterval` (= 'management interval'). For example, if a one year forecast of the catch during 2021 is of interest, then `maninterval` is set to represent the start and end of the forecast interval: `c(2021,2022)`. In addition to the forecast interval, a fishing scenario can be specified. This is done by specifying a factor (`ffac`) that is a fishing mortality multiplier at the start of the forecast period by. By default, the fishing mortality is unaltered and projected forward maintaining potential seasonal patterns. The time point of the forecasted biomass and fishing mortality can be controlled by setting `maneval` (= 'management evaluation'). The code to obtain the forecasted annual catch in the interval starting 1990 under

a management scenario where the fishing pressure is reduced by 25% starting in 1990, and the time point of forecasted biomass and fishing mortality in 1991 is:

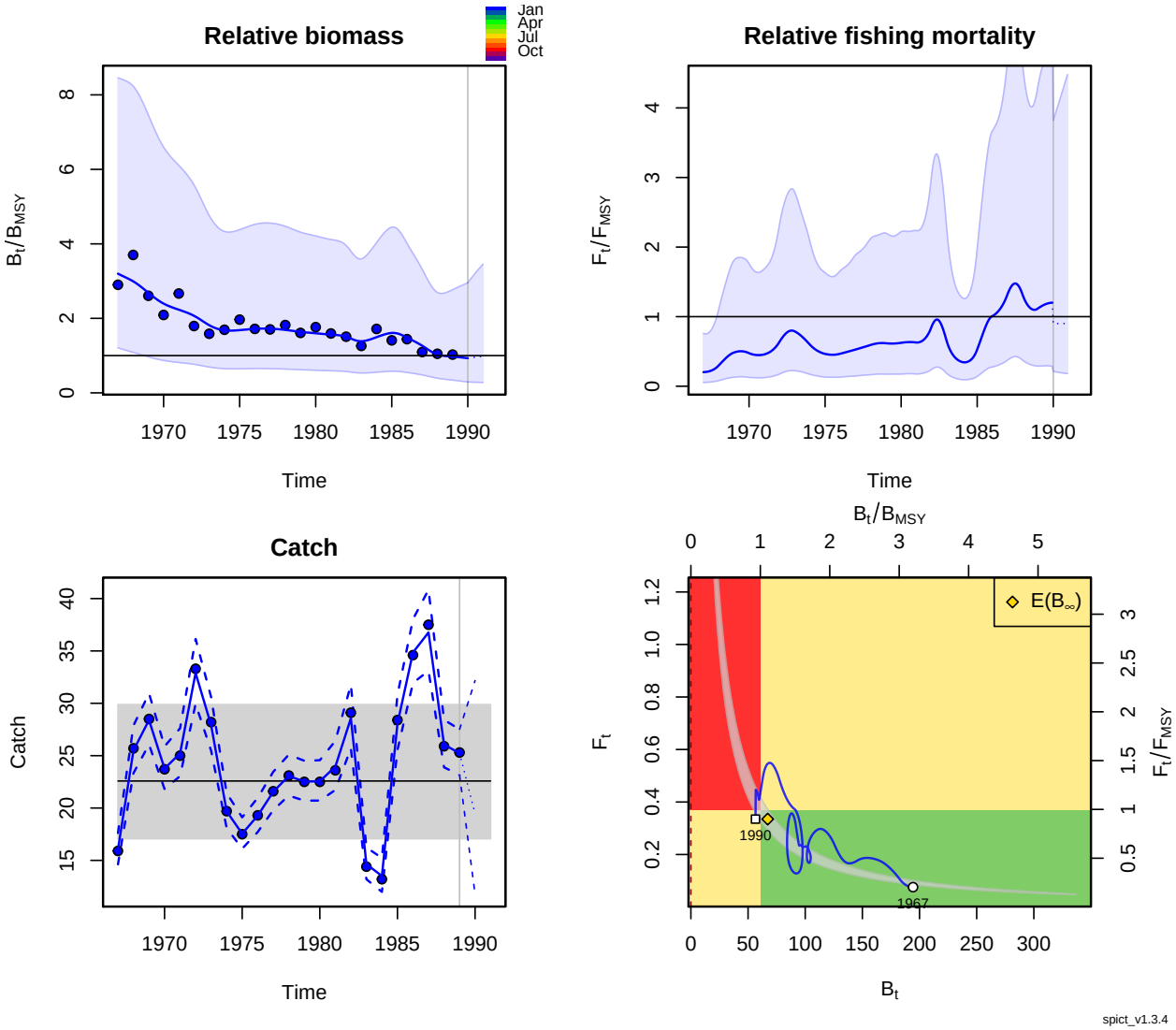
```
inp <- pol$albacore
inp$maninterval <- c(1990, 1991)
inp$maneval <- 1991
inp$ffac <- 0.75
rep <- fit.spict(inp)
```

To specifically show forecast results use

```
sumspict.predictions(rep)
                  prediction      cilow       ciupp     log.est
   B_1991.00      59.6456548 32.5511836 109.2926199  4.08842130
   F_1991.00       0.3348376  0.1179773   0.9503202 -1.09410958
   B_1991.00/Bmsy  0.9820378  0.2791343   3.4549617 -0.01812545
   F_1991.00/Fmsy  0.9006337  0.1792951   4.5240555 -0.10465670
   Catch_1990.00  19.4447465 11.7526351  32.1713523  2.96757693
   E(B_inf)       67.1855141         NA          NA  4.20745766
```

These predictions are also shown when using `summary(rep)` and as with any spict object the results can be plotted using `plot(rep)`. Here, however, we use a selection of 4 graphs to visualise the change in forecasted fishing mortality and associated change in forecasted catch more clearly:

```
plot2(rep)
```



The dotted lines in the graph represent the predicted quantities for the forecast interval. Note the clear drop and subsequent constant fishing mortality representing the 25% reduction in fishing pressure as set with `ffac`. The decrease in fishing pressure results in constant (slightly increasing) biomass as opposed to the expected decrease if fishing effort had remained constant. The larger confidence intervals indicate the larger uncertainty associated with model predictions without data. Setting `ffac` (or `fcon` for changing the fishing mortality by an absolute value instead of a factor) in the input list affects any SPiCT fit with this input list. This can be reversed back to the default (continuing the F process) by setting `inp$ffac <- NULL`. In the following section, we introduce a range of different functions which make the exploration of different F factors and other management strategies more straight-forward and do not affect the fitted SPiCT object or its input list.

Management

In addition to manually changing the forecast period and fishing factor as described above, SPiCT includes a wide range of functions related to fisheries management. The `manage` function applies 8 different management scenarios and allows to explore the effect of different management strategies on the recommended Total Allowable Catch (TAC), or predicted fishing mortality or biomass. The scenarios include advice rules such as constant catch, no fishing or the current advice rule recommended by the export group for data limited fisheries management in ICES (ICES 2019). The results of this function are appended to the `spictcls` object,

thus the application is straight-forward:

```
inp <- pol$albacore
rep <- fit.spict(inp)
rep <- manage(rep)
   Selected scenario(s):  currentCatch, currentF, Fmsy, noF, reduceF25, increaseF25, msyHockeyStick, ic
```

where `rep` is the result of `fit.spict()` from the code above. The `spictcls` object fitted in each scenario is stored in a list called `man` within `rep`. The argument `scenarios` controls which scenarios are included; it is a vector of scenario numbers, e.g. `scenarios = c(1,5,2,8)`, or scenario names, e.g. `scenarios = c("noF", "ices", "incrF25")`. The list of the predefined management scenarios in the `manage` function with index and name are:

1. **currentCatch**: Keep the catch of the current year (i.e. the last observed catch).
2. **currentF**: Keep the F of the current year.
3. **Fmsy**: Fish at Fmsy i.e. F=Fmsy.
4. **noF**: No fishing, reduce to 1% of current F.
5. **reduceF25**: Reduce F by 25%.
6. **increaseF25**: Increase F by 25%,
7. **msyHockeyStick**: Use ICES MSY hockey-stick advice rule [@msycat34].
8. **ices**: Use ICES MSY 35th hockey-stick advice rule [@wklifeix].

More information about these advice rules and other functionality of the `manage` function can be found in the function documentation (`?manage`).

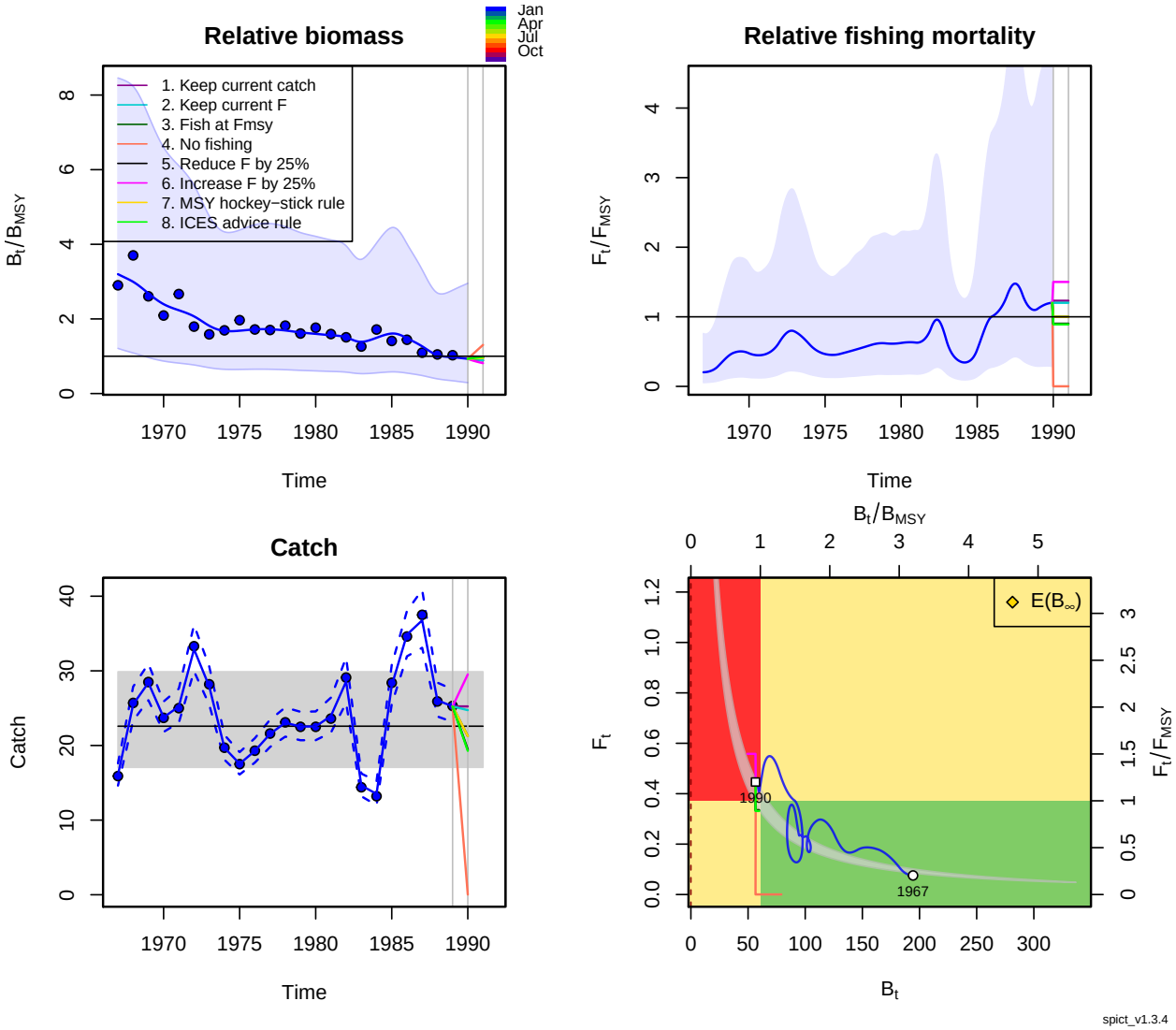
The results of the management scenarios can be summarised by:

```
sumspict.manage(rep)
   SPiCT timeline:

         Observations              Management
       1967.00 - 1990.00        1990.00 - 1991.00
    |-----------------------| -----------------------|

   Management evaluation: 1991.00

   Predicted catch for management period and states at management evaluation time:

                              C B/Bmsy F/Fmsy
   1. Keep current catch    25.2   0.89   1.23
   2. Keep current F        24.7   0.89   1.20
   3. Fish at Fmsy          21.3   0.95   1.00
   4. No fishing             0.0   1.30   0.00
   5. Reduce F by 25%       19.4   0.98   0.90
   6. Increase F by 25%     29.5   0.81   1.50
   7. MSY hockey-stick rule 21.3   0.95   1.00
   8. ICES advice rule      19.3   0.98   0.89
```

The summary function prints a timeline as well as the predicted catch during the management interval and absolute and relative biomass and fishing mortality at the management evaluation time for each scenario. Additionally, the quantities '`perc.dB`' and '`perc.dF`' show the percentage change in biomass and fishing mortality from the start to the end of the management period for each management scenario, respectively. The implications of the different management scenarios can also be shown graphically by calling the `plot`

function or any plotting function of the `plotspict.` family (e.g. `plotspict.ffmsy`) on a spict object with a `man` element:

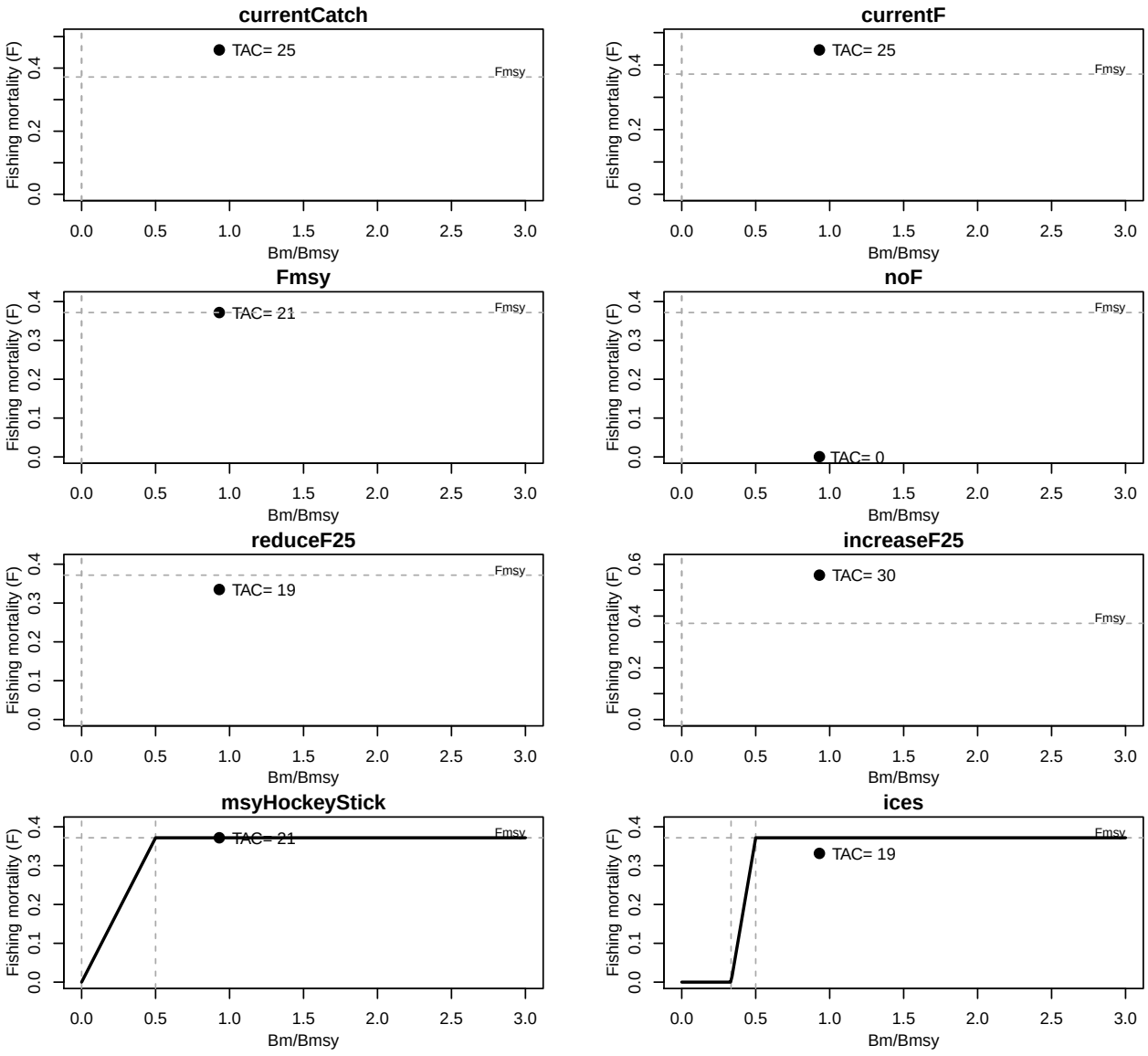
```
plot2(rep)
```



If any plotting function of the `plotspict.` family is called on a SPiCT object which includes management scenarios (`rep$man != NULL`), the prediction and confidence intervals of the base scenario (`rep`) are omitted from the plot and instead the projections of the different management scenarios are displayed in different colours. The grey vertical line corresponding to the time of the last observation in the standard plots is replaced by two lines which corresponds to the start and end of the management interval. In some cases, there might only be one vertical line displayed in the catch plot or the trajectories of the scenarios might be missing in the catch plot. Refer to the function documentation if that is the case (`help(plotspict.catch)`). In order to go back to the previous plots, one can remove the management scenarios (see below) or assign the results of the `manage` function to a different object, e.g. `man <- manage(rep)` and have both plotting functionalities with `plot(rep)` and `plot(man)`.

The function `plotspict.hcr` (spict version >= 1.3.4) provides a visual summary of the management scenarios.

```
plotspict.hcr(rep)
```



Assessment and intermediate period

Besides defining the assessment period in the input data as shown above, it can also be changed in the `manage` function directly using the argument `maninterval`. The management period or interval can correspond to any period from a fraction of a year to spanning over several years (see below for more examples of variable assessment periods). To visualise the management period the function `man.timeline` can be applied to an input list:

```
man.timeline(inp)
   SPiCT timeline:

        Observations              Management
      1967.00 - 1990.00        1990.00 - 1991.00
    |----------------------| ----------------------|

   Management evaluation: 1991.00
```

or to a fitted spict object:

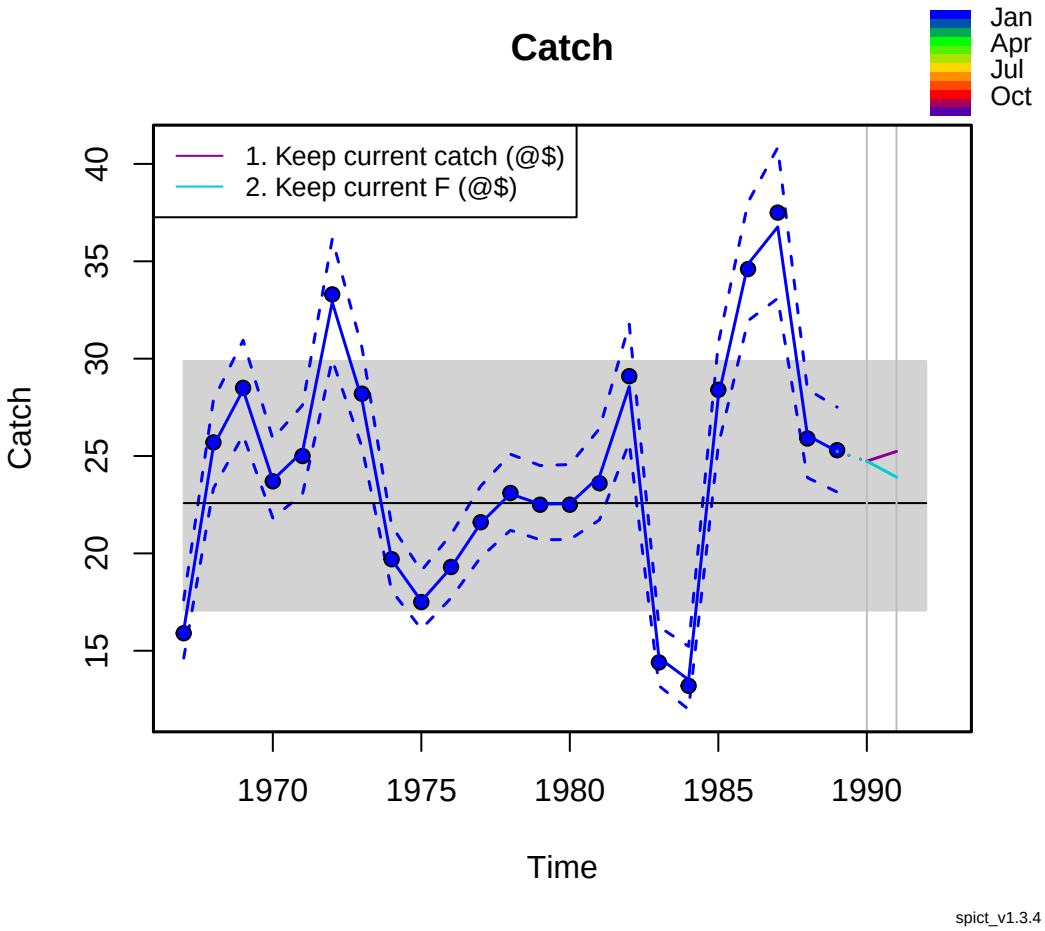
```
man.timeline(rep)
   SPiCT timeline:

         Observations              Management
       1967.00 - 1990.00        1990.00 - 1991.00
    |----------------------| ----------------------|

   Management evaluation: 1991.00
```

As the output shows, the management period period starts right after the time of the last observation, in this example in 1990. In reality, however, there is often a lag between the data and management period. In fact, the assessment is often performed within the gap year and the management scenarios are supposed to start at the beginning of the following year. Thus, it makes sense to distinguish the intermediate period from the management period. For the albacore data set we could assume a management period starting in 1991, which would change the time line as follows:

```
inp$maninterval <- c(1991,1992)
man.timeline(inp)
   SPiCT timeline:

         Observations              Intermediate              Management
       1967.00 - 1990.00        1990.00 - 1991.00        1991.00 - 1992.00
    |----------------------| ----------------------| ----------------------|

   Management evaluation: 1992.00
```

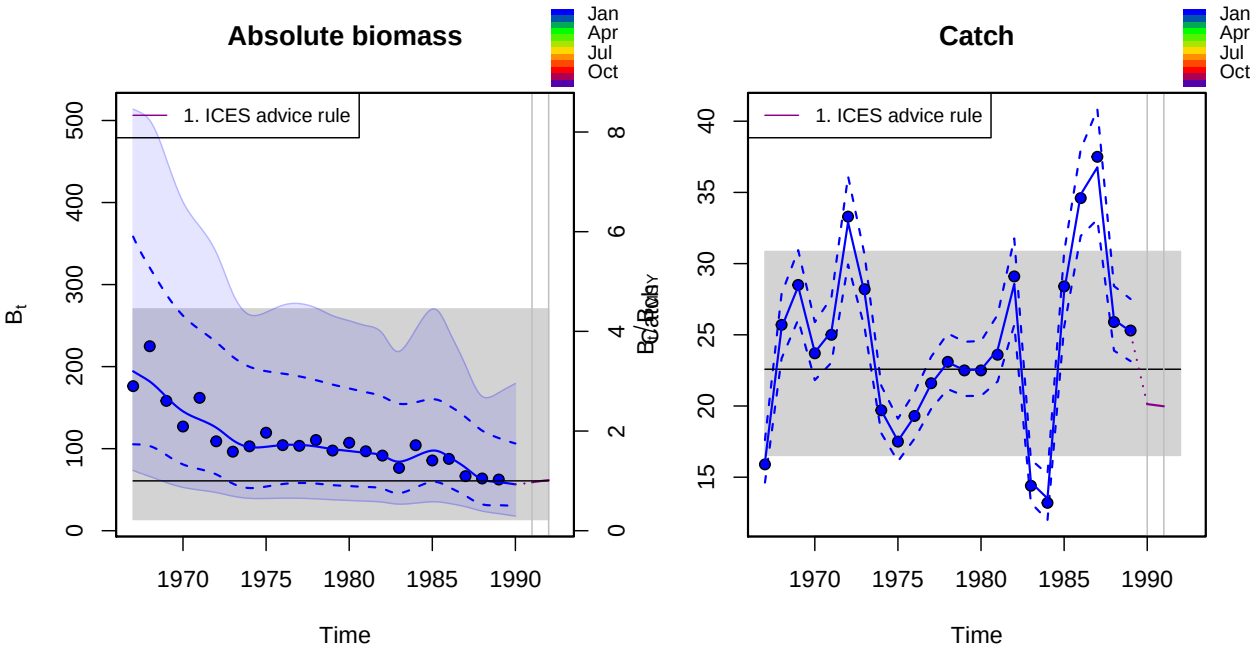
As can be seen, the timeline includes an additional section called 'Intermediate' referring to the intermediate period. Management scenarios with an intermediate period from 1990 to 1991 and an assessment period can thus be specified for the albacore data set with:

```
repIntPer <- manage(rep, scenarios = c(1,2),
                    maninterval = c(1991,1992), maneval = 1992)
   Selected scenario(s):  currentCatch, currentF
plotspict.catch(repIntPer)
```



It is important to note that defining a management interval a few time steps after the time of the last catch observation requires the model to make an assumption about the fishing mortality during the intermediate period. By default, SPiCT continues the fishing mortality process in the intermediate period, i.e. the fishing mortality corresponds to the estimated F at the time step of the last observation and the estimated seasonal F process. However, in some cases it might be meaningful to assume a certain catch during this period rather than continuing the F process. This might be in particular relevant when information about the catch in the intermediate period is available, for example in form of a TAC which was recommended and implemented the year before. Therefore, a specific catch can be specified for the intermediate period using the argument `intermediatePeriodCatch`.

```
repIntPerC <- manage(rep, scenarios=c(8), maninterval = c(1991,1992), intermediatePeriodCatch = 20)
   Selected scenario(s):  ices
par(mfrow=c(1,2))
plotspict.biomass(repIntPerC)
plotspict.catch(repIntPerC)
```



As the graph shows, the catch in the intermediate period has been set to 20t (year 1990). Dotted lines of the management scenarios reflect the intermediate period, while solid lines reflect the management period. Be aware that the vertical bars indicating the start and end of the management period might seem off by a year for the catch plot, however, they correspond to the timing of the catch observations which are drawn at the beginning of the catch interval they refer to.

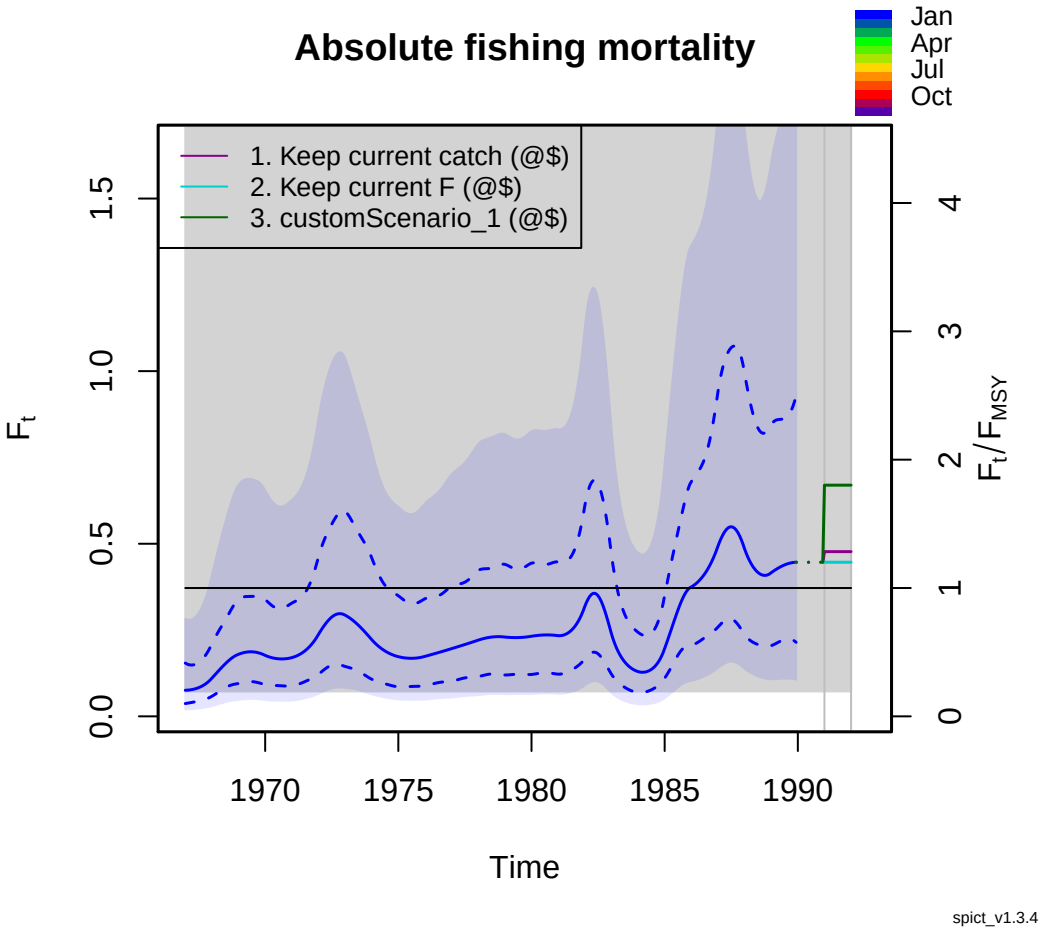
Note that the specified catch corresponds to the complete intermediate period, which might be a year, but could also be quarter of a year or 3 years depending on the time of the last observation and the start of the management period (see below for examples with variable intermediate periods). Note further, that the catch used for the scenario 'currentCatch' is the last observed catch even though a certain catch is provided in the intermediate period. The argument `intermediatePeriodCatchList` allows to specify any number of catch 'observations' corresponding to any interval during the intermediate period. It is a list with the elements 'obsC', 'timeC', 'dtc' and the optional element 'stdevfacC' (which is equal to 1 if not provided). Find more information about this and related arguments in the documentation of the `manage` function.

Custom management scenarios

The default management scenarios included in the `manage` function are a selection of advice rules that are relevant to managers and stakeholders, in particular, based on our experience with stocks managed by ICES. However, there is no limitation in terms of custom advice rules and management scenarios. The function `add.man.scenario` allows to define a wide range of management scenarios and creates a new spict object, which can easily be added to exiting scenarios in `rep$man` (contrary to `manage` which overwrites all scenarios in `rep$man`). For example, to add an additional management scenario to `repIntPer` which increases fishing mortality by 50%:

```
repIntPer <- add.man.scenario(repIntPer, ffac = 1.5)
sumspict.manage(repIntPer)
   SPiCT timeline:

        Observations            Intermediate            Management
     1967.00 - 1990.00       1990.00 - 1991.00       1991.00 - 1992.00
    |---------------------| ----------------------| ----------------------|

   Management evaluation: 1992.00
```

```
   Predicted catch for management period and states at management evaluation time:

                                C B/Bmsy F/Fmsy
   1. Keep current catch (@$) 25.2   0.85   1.28
   2. Keep current F (@$)     23.9   0.87   1.20
   3. customScenario_1 (@$)   32.7   0.72   1.80
   (@) This scenario assumes another management period. Thus, the estimates might not be comparable to
   ($) This scenario assumes another management evaluation time. Thus, the estimates might not be compa
plotspict.f(repIntPer)
```



Note that the default name of a scenario 'customScenario_1' was used for this management scenario, where '1' represents the number of scenarios in `rep$man` with this default label. By using the argument `scenarioTitle` you can specify any label for your scenario. For example, to define a scenario that reduces the last observed catch by 64% and is labelled 'reduced_catch':

```
repIntPer <- add.man.scenario(repIntPer, cfac = 0.64,
                              scenarioTitle = "reduced_catch")
names(repIntPer$man)
   [1] "currentCatch"     "currentF"         "customScenario_1" "reduced_catch"
```

Be aware that management scenarios in the list `rep$man` can be overwritten if a scenario with the specified `scenarioTitle` is already present in the list. A few important arguments of the `add.man.scenario` function to customise management scenarios are:

- **ffac**: Factor to multiply current fishing mortality by.

- **cfac**: Factor to multiply current catch by.
- **breakpointBBmsy**: Breakpoint in terms of B/B_{MSY} for the hockey-stick HCR. By default no breakpoint is assumed.
- **safeguardB**: List defining an optional precautionary buffer by means of a biomass reference level relative to B/B_{MSY} (`limitB`) and the risk aversion probability (`prob`):
  - **limitB**: Reference level for the evaluation of the predicted biomass defined as fraction of B/B_{MSY}. By default the PA buffer is not used. Theoretically, any value smaller than 1 is meaningful, but an ICES recommended value would be 30% `safeguardB$limitB = 0.3` @[wklifeiv].
  - **prob**: Risk aversion probability of the predicted biomass relative to specified reference level (`limitB`) for all rules with PA buffer. The default value is 0.95 as recommended by (ICES 2019).
- **fractiles**: List defining the fractiles of the 3 distributions of `catch`, `bbmsy`, and `ffmsy`:
  - **catch**: Fractile of the predicted catch distribution.
  - **bbmsy**: Fractile of the B/B_{MSY} distribution.
  - **ffmsy**: Fractile of the F/F_{MSY} distribution.

Note that the fractile for the F/F_{MSY} distribution is actually 1 minus the fractile specified. Otherwise, a larger fractile would be more precautionary than a smaller one, as the current fishing mortality is divided by the value of this distribution ($F_{y+1} = \frac{F_y}{\frac{F_y}{F_{MSY}}}$). This allows a consistent setting of fractiles among the different quantities.

The advice rule recommended by ICES (2019) can thus be defined by the combination of several of these arguments:

```
repIntPer <- add.man.scenario(repIntPer, scenarioTitle = "ices",
                              breakpointB = 0.5,
                              fractiles = list(catch=0.35, bbmsy=0.35, ffmsy=0.35))
sumspict.manage(repIntPer)
   SPiCT timeline:

         Observations              Intermediate              Management
       1967.00 - 1990.00         1990.00 - 1991.00         1991.00 - 1992.00
     |----------------------| ----------------------| ----------------------|

   Management evaluation: 1992.00

   Predicted catch for management period and states at management evaluation time:

                                C B/Bmsy F/Fmsy
   1. Keep current catch (@$) 25.2   0.85   1.28
   2. Keep current F (@$)     23.9   0.87   1.20
   3. customScenario_1 (@$)   32.7   0.72   1.80
   4. reduced_catch (@$)      16.2   1.00   0.76
   5. ICES advice rule (@$)   13.6   1.04   0.62
   (@) This scenario assumes another management period. Thus, the estimates might not be comparable to t
   ($) This scenario assumes another management evaluation time. Thus, the estimates might not be compa
```

A detailed description of all arguments of this function can be found in the function documentation (run `?add.man.scenario`).

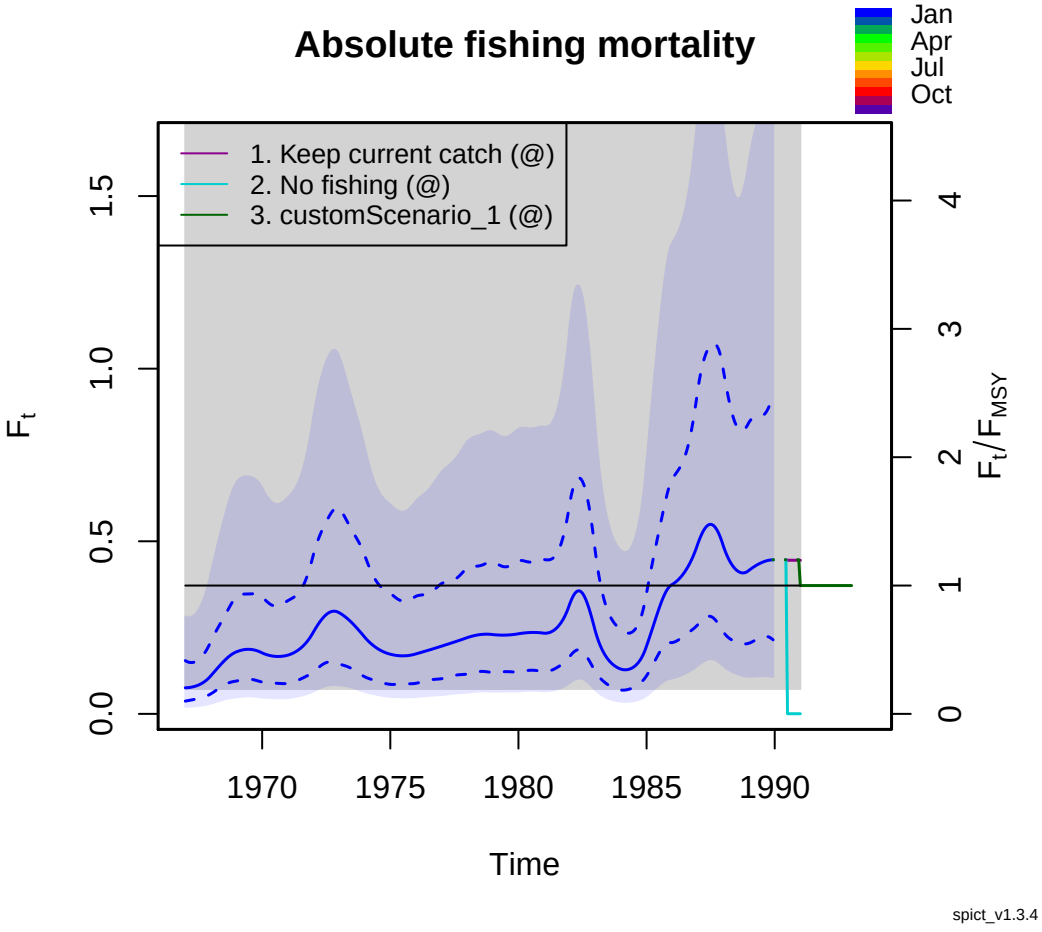
Variable management periods

As mentioned above, the management period (and intermediate period) is not limited to the length of one year nor to the start of a year, but can span half a year, several years or start within any point in time within the year (cf. in-year advice ICES (2019)). For example, applying two default scenarios for a half year period starting in the middle of 1990:

```
repVarPer <- manage(rep, scenarios = c(1,4), maninterval = c(1990.5,1991))
   Selected scenario(s):  currentCatch, noF
```

Scenarios with different management intervals can also be combined, although the functions will print a note about mismatching periods or assumptions in the intermediate period. For example, to add an advice rule which reduces the F linearly to zero if the biomass is below 50% of B/B_{MSY} over a management interval of 2 years to the previous scenarios:

```
repVarPer <- add.man.scenario(repVarPer, maninterval = c(1991, 1993),
                              breakpointB = 0.5)
   Note that the specified management interval: 1991.00 - 1993.00 differs from existing scenarios in re
plotspict.f(repVarPer)
```



Additionally, the summary output will not show percentage differences for F or B (`NaN`) and the timeline will only show the period with observations as the intermediate and management period differ between the scenarios.

```
sumspict.manage(repVarPer)
   SPiCT timeline:

        Observations            Intermediate            Management
     1967.00 - 1990.00       1990.00 - 1990.50       1990.50 - 1991.00
    |----------------------| ----------------------| ----------------------|

   Management evaluation: 1991.00
```

```
   Predicted catch for management period and states at management evaluation time:

                              C B/Bmsy F/Fmsy
   1. Keep current catch (@) 12.2   0.89    1.2
   2. No fishing (@)          0.0   1.10    0.0
   3. customScenario_1 (@)   41.7   0.89    1.0
   (@) This scenario assumes another management period. Thus, the estimates might not be comparable to
```

Advanced management settings and functionality

Besides selecting scenarios in `manage` using the argument `scenarios`, the function `man.select` can be used to change the order of scenarios or omit certain or all scenarios included in `rep$man`:

```
names(rep$man)
   [1] "currentCatch"   "currentF"       "Fmsy"           "noF"
   [5] "reduceF25"      "increaseF25"    "msyHockeyStick" "ices"
rep <- man.select(rep, scenarios = c(1,3))
   Selected scenario(s):  currentCatch, Fmsy
```

One of the important quantities in fisheries management is the predicted catch during the management interval under given fishing mortality, also called Total Allowable Catch (TAC). The function `man.tac` allows to extract a list with the TAC of all scenarios in `rep$man`:

```
man.tac(rep)
   $currentCatch
   [1] 25.2343

   $Fmsy
   [1] 21.25495
```

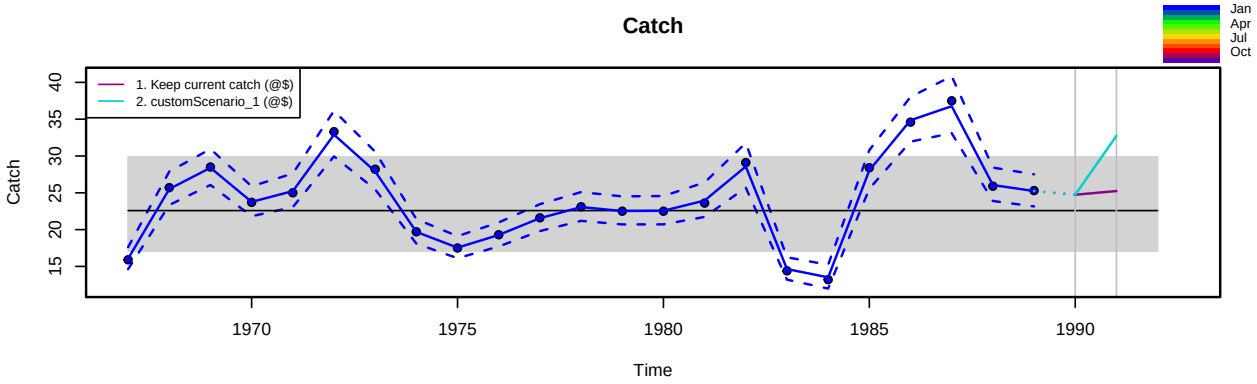
Note that the unit of the TAC corresponds to the catch unit specified in `inp$catchunit` or the unit of the catch observations if not specified. If the user is only interested in the TAC of the specified management scenario (e.f. for the implementation in a management strategy framework (MSE)), the function `get.TAC` can be used:

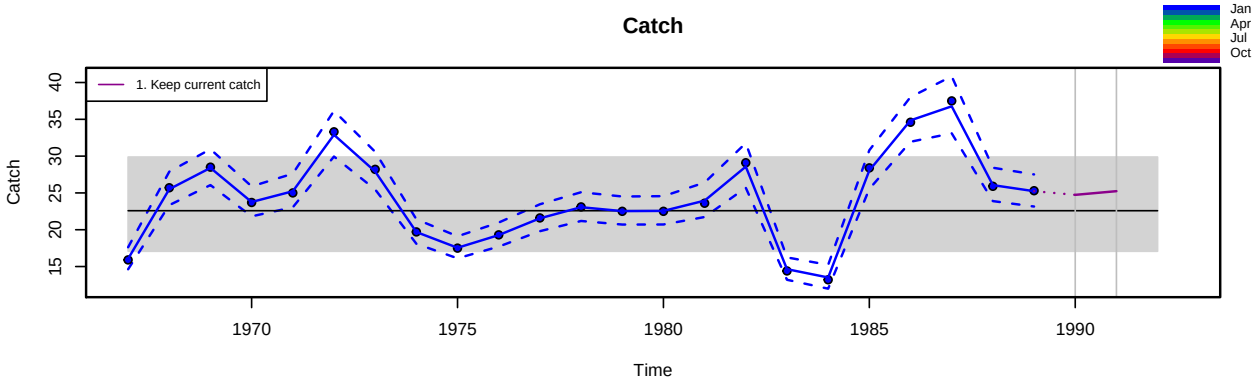
```
get.TAC(rep, maninterval = c(1990.25, 1991),
        safeguardB = list(limitB = 0.3), verbose = TRUE)
   Note that the specified management interval: 1990.25 - 1991.00 differs from existing scenarios in re
   [1] 15.73663
```

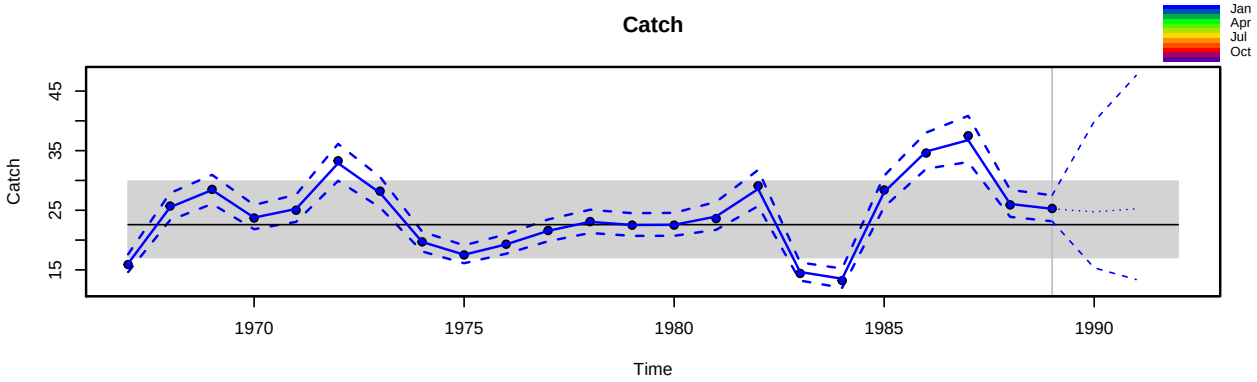
After comparing the implications of different management scenarios, it might be of interest to extract a specific scenario for further analyses, plotting, or reporting. Specific scenarios can be extracted from the SPiCT object by referencing their names or positioning in the list `rep$man`, e.g. `rep$man[[1]]`. However, care has to be taken, as some scenarios (e.g. 'constantCatch' or scenarios with a specified catch in the intermediate period) might include additional catch observations. Thus, the best way to select certain scenarios is by means of the function `man.select`. This function does not only allow to change the order of the management scenarios in `rep$man` or make a selection of scenarios, but also allows to extract a certain scenario as a standard SPiCT object of the class `spictcls`:

```
## selection by index
length(repIntPer$man)
   [1] 5
repSelect1 <- man.select(repIntPer, scenarios = c(3,1))
   Selected scenario(s):  currentCatch, customScenario_1

## selection by scenario name
```

```
names(repIntPer$man)
   [1] "currentCatch"     "currentF"         "customScenario_1" "reduced_catch"
   [5] "ices"
repSelect2 <- man.select(repIntPer, scenarios = c("currentCatch"))
   Selected scenario(s):  currentCatch

## selected object as standard spictcls object
repSelect3 <- man.select(repIntPer, scenarios = c("currentCatch"), spictcls = TRUE)

## Plot objects with selected scenarios
opar <- par(mfrow=c(3,1))
plotspict.catch(repSelect1)
plotspict.catch(repSelect2)
plotspict.catch(repSelect3)
par(opar)
```







Note the different catch trajectories of the second and third plot. Although, the same scenario was selected, the first approach still shows the management-type plot, while the second approach (with argument `spictcls = TRUE`) shows the default 'spictcls' catch plot with predicted uncertainties.

All management related results can easily be removed from the SPiCT object by overwriting the `man` element: `rep$man = NULL` or

```
rep <- man.select(rep, scenarios = NULL)
   All scenarios removed!
```

Other model settings and options

`catchunit` - Define unit of catch observations

This will print the unit of the catches on relevant plots.

Example: `inp$catchunit <- "'000 t"`.

`dteuler` and `eulertype` - Temporal discretisation and time step

To solve the continuous-time system an Euler discretisation scheme is used. This requires a time step to be specified (`dteuler`). The smaller the time step the more accurate the approximation to the continuous-time solution, however with the cost of increased memory requirements and computing time. The default value of `dteuler` is 1/16, which seems sufficiently fine for most cases, and perhaps too fine for some cases. When fitting quarterly data and species with fast growth it is important to have a small `dteuler`. The influence of `dteuler` on the results can be checked by using different values and comparing resulting model estimates. If `dteuler <- 1` the model essentially becomes a discrete-time model with one Euler step per year.

There are two possible temporal discretisation schemes which can be set to either `eulertype = 'hard'` (default) or `eulertype = 'soft'`. If `eulertype = 'hard'` then time is discretised into intervals of length `dteuler`. Observations are then assigned to these intervals. For annual and quarterly data `dteuler = 1/16` is appropriate, however if fitting to monthly data `dteuler` should be changed to e.g. `1/24`. If `eulertype = 'soft'` (careful, this feature has not been thoroughly tested), then time is discretised into intervals of length `dteuler` and additional time points corresponding to the times of observation are added to the discretisation. This feature is particularly useful if observations (most likely index series) are observed at odd times during the year. The model then estimates values of biomass and fishing mortality at the exact time of the observation instead of assigning the observation to an interval.

`msytype` - Stochastic and deterministic reference points

As default the stochastic reference points are reported and used for calculation of relative levels of biomass and fishing mortality. It is, however, possible to use the deterministic reference points by setting `inp$msytype <- 'd'`.

`do.sd.report` - Perform SD report calculations

The sdreport step calculates the uncertainty of all quantities that are reported in addition to the model parameters. For long time series and with small `dteuler` this step may have high memory requirements and a substantial computing time. Thus, if one is only interested in the point estimates of the model parameters it is advisable to set `do.sd.report <- 0` to increase speed.

`reportall` - Report all derived quantities

If uncertainties of some quantities (such as reference points) are required, but uncertainty on state variables (biomass and fishing mortality) are not needed, then `reportall <- 0` can be used to increase speed.

`optim.method` - Report all derived quantities

Parameter estimation is per default performed using R's `nlminb()` optimiser. Alternatively it is possible to use `optim` by setting `inp$optim.method <- 'optim'`.

References

ICES. 2019. "Ninth Workshop on the Development of Quantitative Assessment Methodologies based on LIFE-history traits, exploitation characteristics, and other relevant parameters for data-limited stocks (WKLIFE IX)." *ICES Scientific Reports* 1 (77): 131 pp. https://doi.org/10.17895/ices.pub.5550.

Pedersen, Martin W., and Casper W. Berg. 2017. "A stochastic surplus production model in continuous time." *Fish and Fisheries* 18 (2): 226–43. https://doi.org/10.1111/faf.12174.

Polacheck, Tom, Ray Hilborn, and Andre E Punt. 1993. "Fitting Surplus Production Models: Comparing Methods and Measuring Uncertainty." *Canadian Journal of Fisheries and Aquatic Sciences* 50 (12): 2597–2607.