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

Martin W. Pedersen, Alexandros Kokkalis, Casper W. Berg

03 December, 2019

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)
   Loading required package: TMB
   Welcome to spict_v1.2.8
```

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 = "1.2.8")
```

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

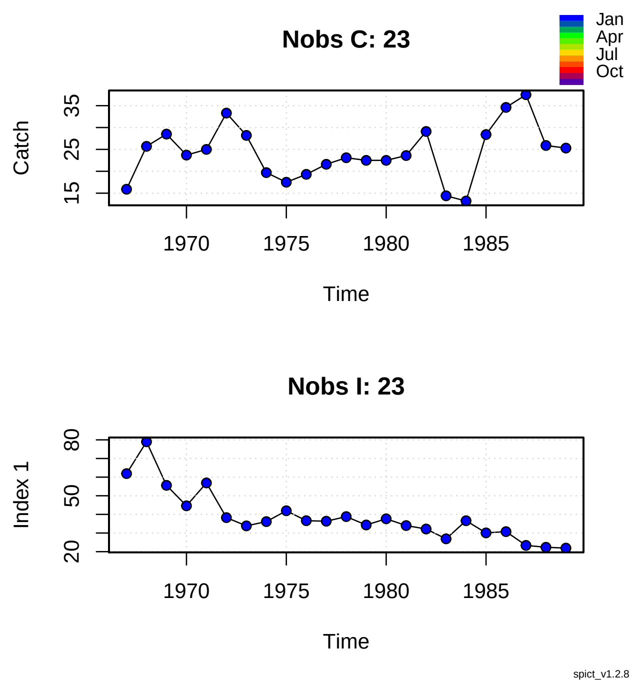
```
inp <- check.inp(pol$albacore)
inp$dtc
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

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

Plotting data

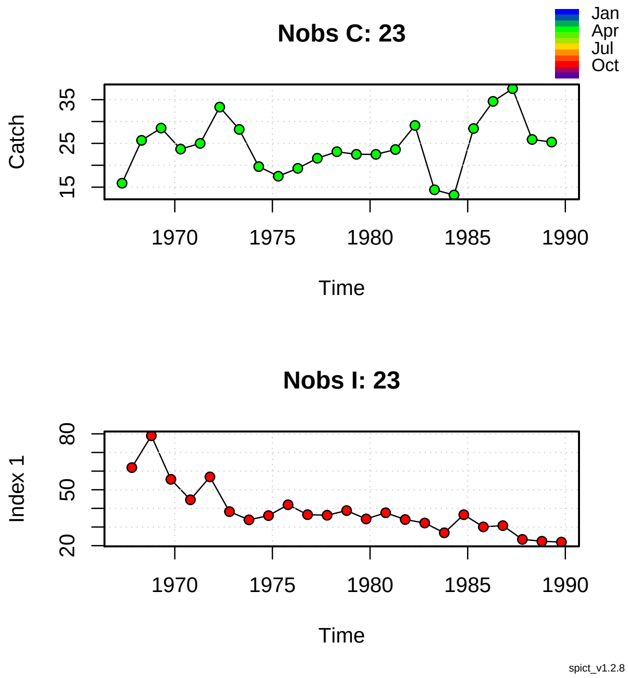
The data can be plotted using the command

```
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 data a bit

```
inpshift <- pol$albacore
inpshift$timeC <- inpshift$timeC + 0.3
inpshift$timeI <- inpshift$timeI + 0.8
plotspict.data(inpshift)
```

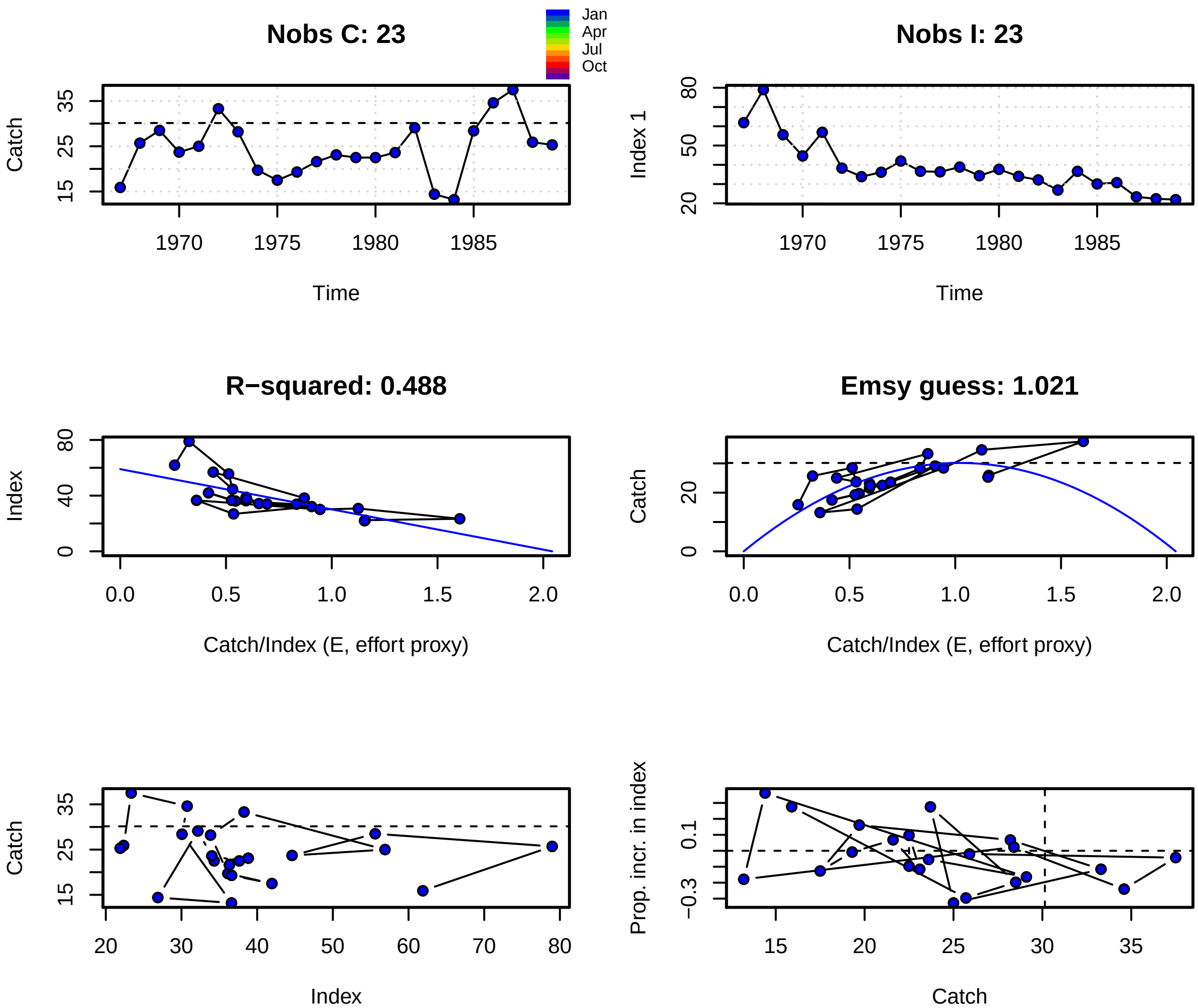


Now the colours show that catches are observed in spring and index in autumn.

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

```
capture.output(summary(res))
 [1] "Convergence: 0  MSG: relative convergence (4)"
 [2] "Objective function at optimum: 2.0654958"
 [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    8.5380751   1.2232673  59.5934547  2.1445356  "
[14] " beta     0.1212598   0.0180690   0.8137639 -2.1098202  "
[15] " r        0.2556011   0.1010590   0.6464730 -1.3641371  "
[16] " rc       0.7435385   0.1445713   3.8240616 -0.2963348  "
[17] " rold     0.8179927   0.0019101 350.3057036 -0.2009019  "
[18] " m       22.5827819  17.0681833  29.8791049  3.1171878  "
[19] " K      201.4753810 138.1193692 293.8930970  5.3056672  "
[20] " q        0.3512552   0.1942693   0.6350990 -1.0462422  "
[21] " n        0.6875264   0.0636693   7.4241809 -0.3746551  "
[22] " sdb      0.0128136   0.0018407   0.0892017 -4.3572451  "
[23] " sdf      0.3673758   0.2673605   0.5048052 -1.0013700  "
[24] " sdi      0.1094038   0.0808973   0.1479555 -2.2127095  "
[25] " sdc      0.0445479   0.0073370   0.2704792 -3.1111902  "
[26] " "
[27] "Deterministic reference points (Drp)"
[28] "         estimate      cilow      ciupp   log.est  "
[29] " Bmsyd 60.7440848 15.4030051 239.553504  4.106670  "
[30] " Fmsyd  0.3717692  0.0722856   1.912031 -0.989482  "
[31] " MSYd  22.5827819 17.0681833  29.879105  3.117188  "
[32] "Stochastic reference points (Srp)"
[33] "         estimate      cilow      ciupp    log.est  rel.diff.Drp  "
[34] " Bmsys 60.7364344 15.4031638 239.490698  4.1065438 -1.259605e-04  "
[35] " Fmsys  0.3717814  0.0722787   1.912339 -0.9894493  3.272178e-05  "
[36] " MSYs  22.5806762 17.0626482  29.883224  3.1170945 -9.325286e-05  "
[37] ""
[38] "States w 95% CI (inp$msytype: s)"
[39] "                  estimate      cilow       ciupp    log.est  "
[40] " B_1989.00      59.1916639 31.0255656 112.9279356  4.0807807  "
[41] " F_1989.00       0.4160746  0.2048129   0.8452495 -0.8768908  "
```

```
[42] " B_1989.00/Bmsy  0.9745660  0.3430172   2.7688961 -0.0257630  "
[43] " F_1989.00/Fmsy  1.1191377  0.2899257   4.3199659  0.1125585  "
[44] ""
[45] "Predictions w 95% CI (inp$msytype: s)"
[46] "                prediction      cilow       ciupp    log.est  "
[47] " B_1990.00      56.5242290 30.0511471 106.3183528  4.0346694  "
[48] " F_1990.00       0.4464502  0.2098833   0.9496597 -0.8064275  "
[49] " B_1990.00/Bmsy  0.9306478  0.2932020   2.9539540 -0.0718744  "
[50] " F_1990.00/Fmsy  1.2008405  0.2832190   5.0915295  0.1830218  "
[51] " Catch_1990.00  24.7359923 15.3328343  39.9058192  3.2082594  "
[52] " E(B_inf)       49.9856610         NA          NA  3.9117362  "
```

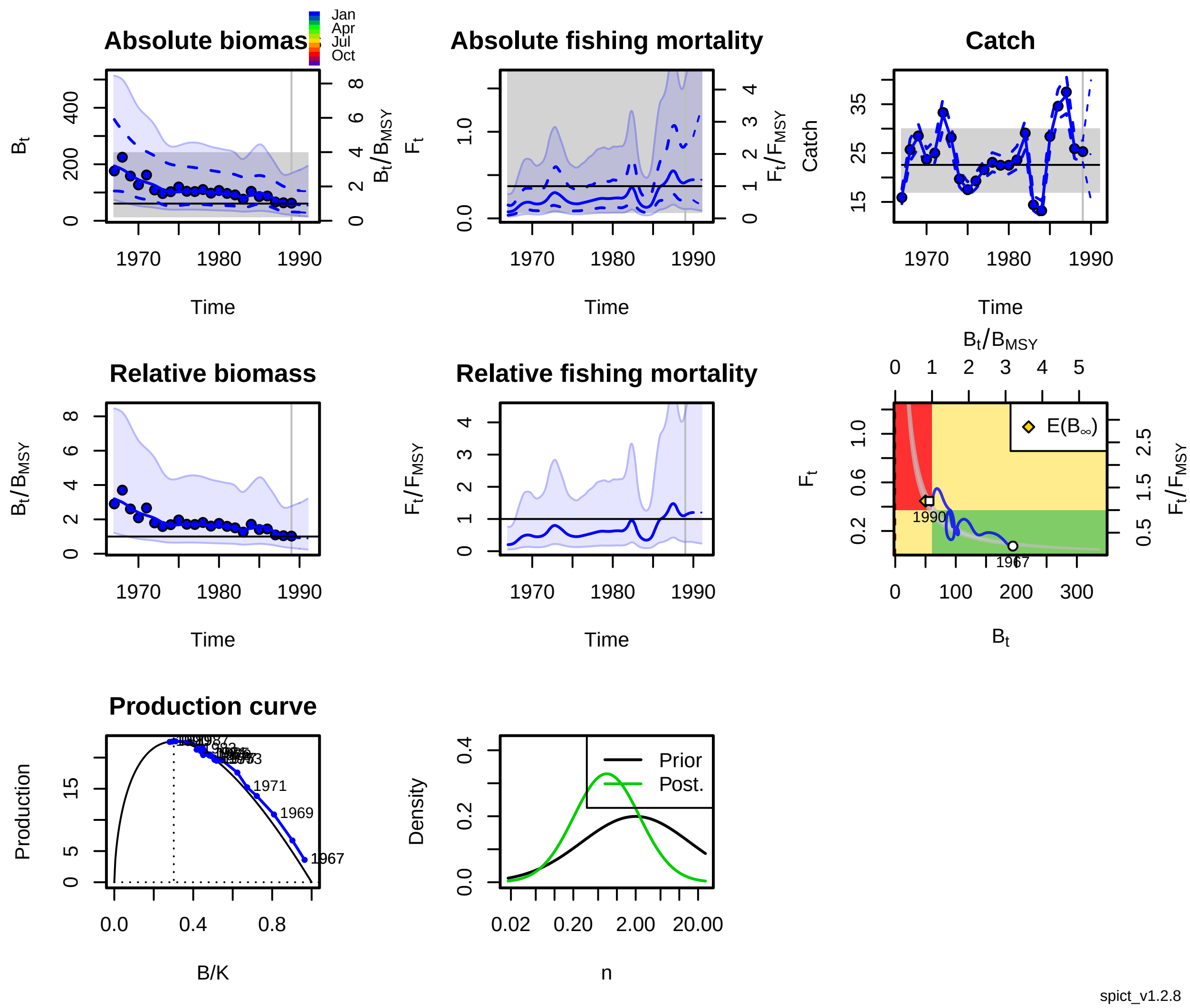
Here the `capture.output()` is only used to provide line numbers for easier reference, but the `summary()` command works without this.

- 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.

```r
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	

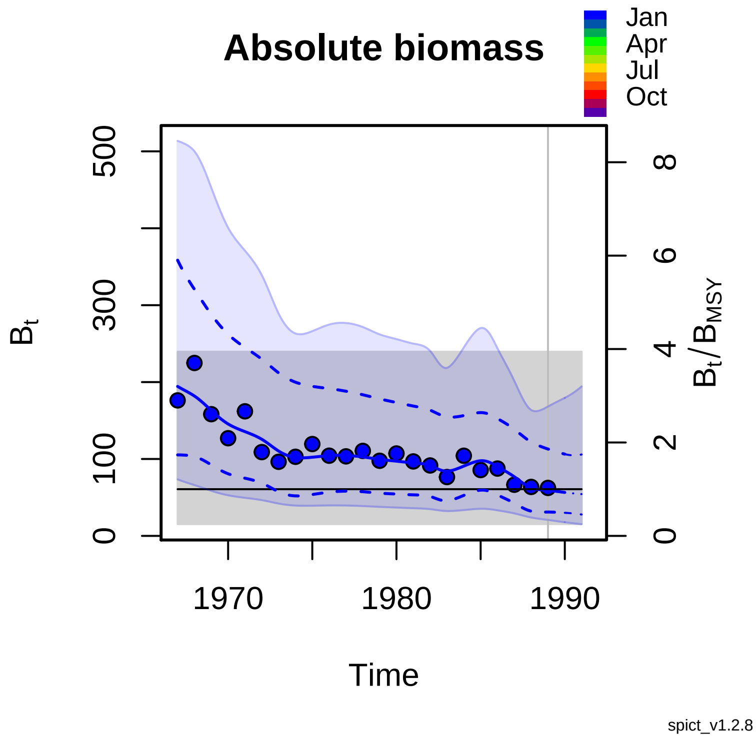
Function	Plot
`plotspict.ci`	Basic data plotting (see section)
`plotspict.data`	Advanced data plotting (see section)
Estimates	
`plotspict.bbmsy`	Relative biomass B/B_{MSY} estimates with uncertainty
`plotspict.biomass`	Absolute (and relative) biomass estimates with uncertainty
`plotspict.btrend`	Expected 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.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

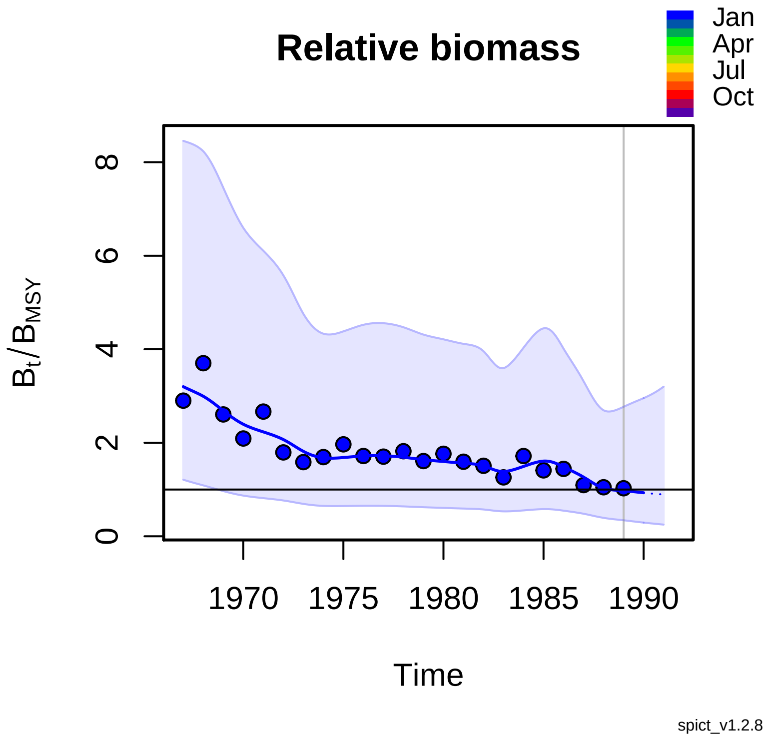
```
plotspict.biomass(res)
```



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

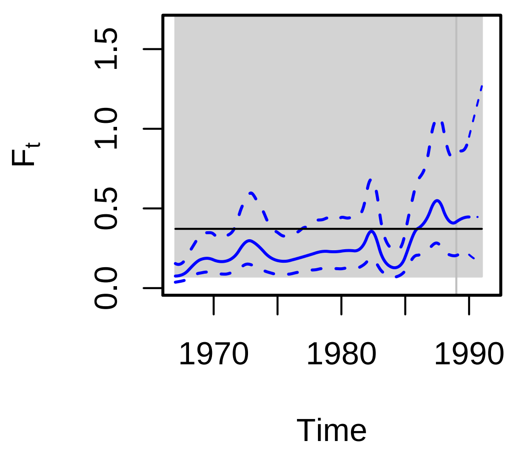
```
plotspict.bbmsy(res)
```

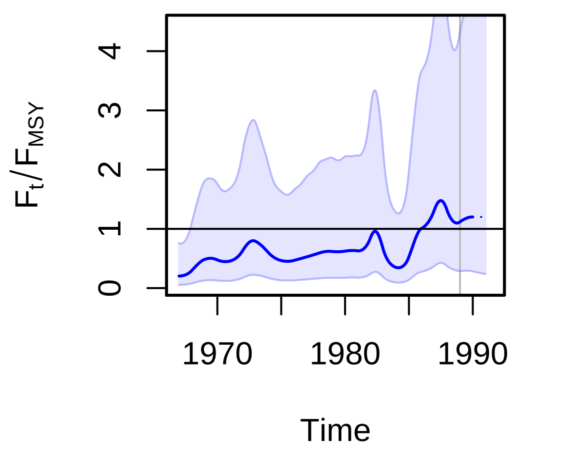


This plot contains much of the same information as given by `plotspict.biomass`, but without the information about absolute biomass and without the 95% CI around the B_{MSY} reference point.

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)
```

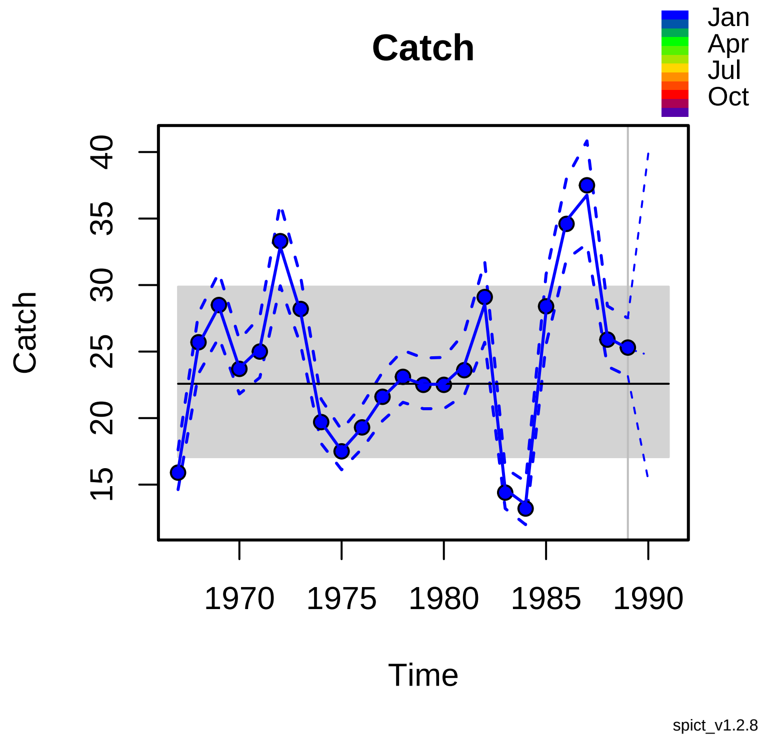




The estimate of F_{MSY} is shown with a horizontal black line with 95% CI shown as a grey region (left plot). The 95% CI of F_{MSY} is very wide in this case. As shown here it is quite straightforward to remove the information about relative levels from the plot of absolute fishing mortality. Furthermore,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

The plot of the catch is produced using

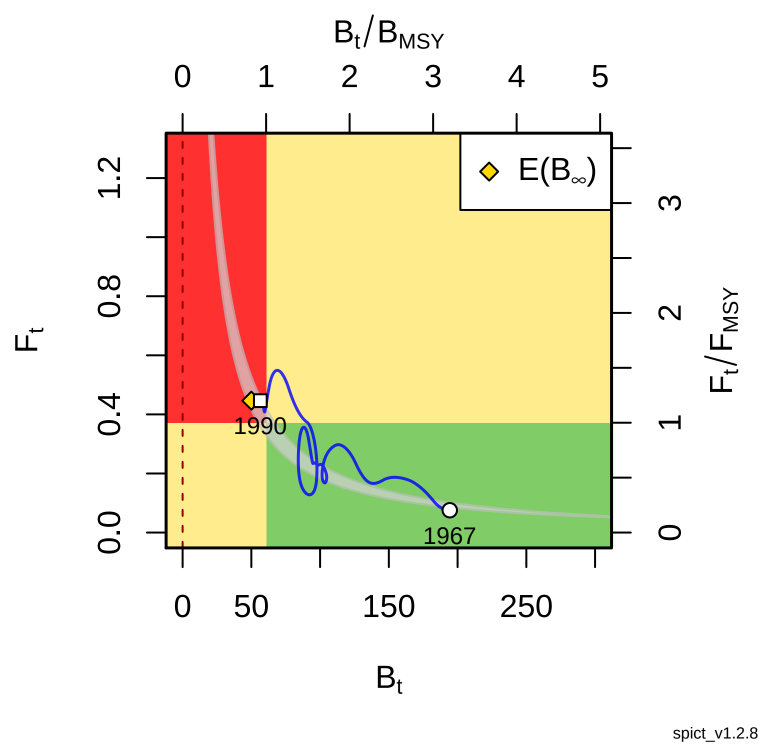
```
plotspict.catch(res)
```



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))
```



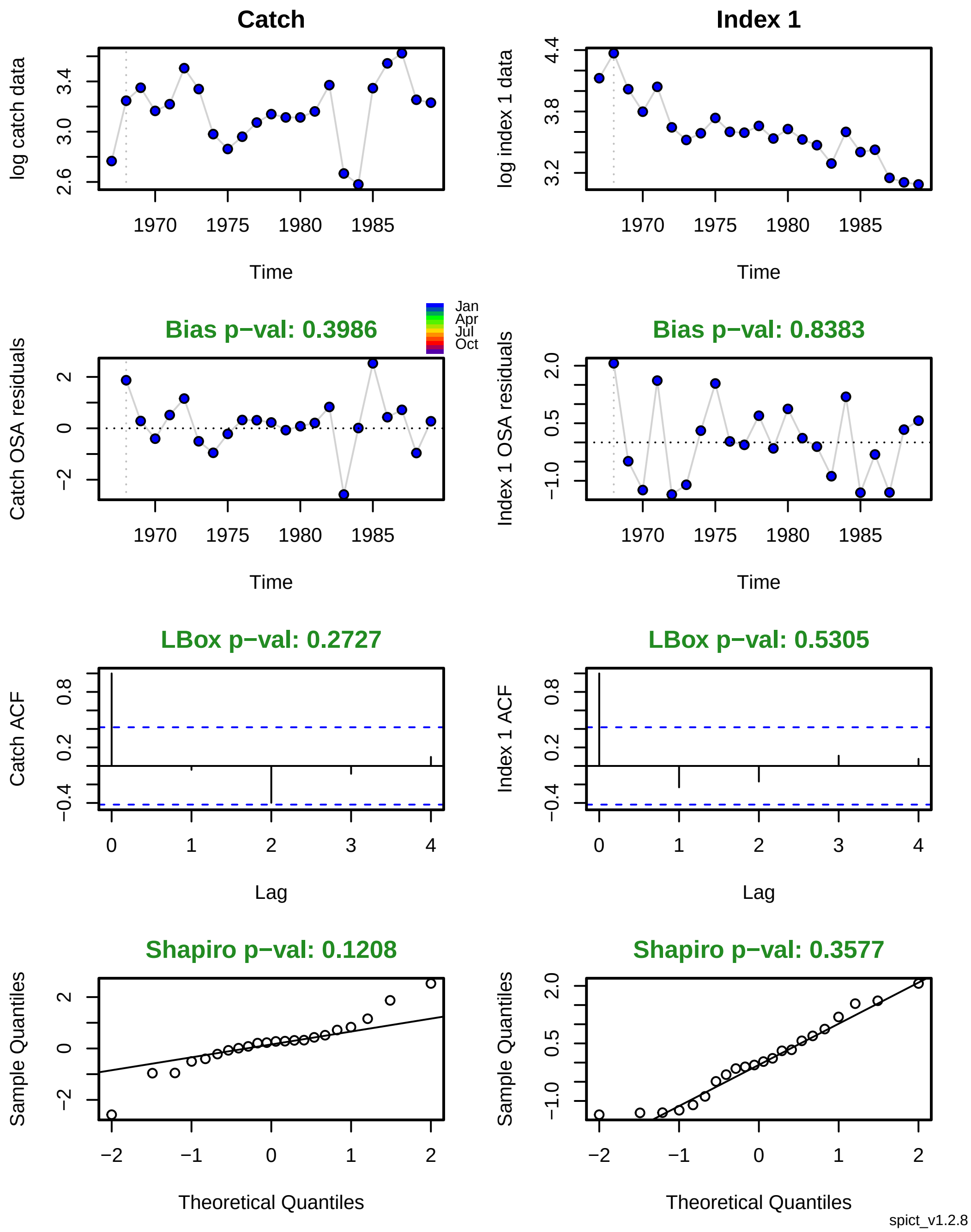
The plot shows the development of biomass and fishing mortality since the initial year (here 1967) indicated with a circle until the terminal year (here 1990) indicated with a square. The yellow diamond indicates the mean biomass over a long period if the current (1990) fishing pressure remains. This 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 15.40316 60.73643 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] "Bmsy"                "Bmsy2"               "BmsyB0"
    [4] "Bmsyd"               "Bmsys"               "Cp"
    [7] "Emsy"                "Emsy2"               "Fmsy"
   [10] "Fmsyd"               "Fmsys"               "gamma"
   [13] "isdb2"               "isdc2"               "isde2"
   [16] "isdf2"               "isdi2"               "K"
   [19] "logalpha"            "logB"                "logBBmsy"
   [22] "logbeta"             "logbkfrac"           "logBl"
   [25] "logBlBmsy"           "logBlK"              "logBmsy"
   [28] "logBmsyd"            "logBmsyPluslogFmsy"  "logBmsys"
   [31] "logBp"               "logBpBmsy"           "logBpK"
   [34] "logCp"               "logCpred"            "logEmsy"
   [37] "logEmsy2"            "logEp"               "logF"
   [40] "logFFmsy"            "logFFmsynotS"        "logFl"
   [43] "logFlFmsy"           "logFmsy"             "logFmsyd"
   [46] "logFmsys"            "logFnotS"            "logFp"
   [49] "logFpFmsy"           "logFs"               "logIp"
   [52] "logIpred"            "logK"                "logm"
   [55] "logMSY"              "logMSYd"             "logMSYs"
   [58] "logn"                "logq"                "logq2"
   [61] "logr"                "logrc"               "logrold"
   [64] "logsdb"              "logsdc"              "logsdf"
   [67] "logsdi"              "m"                   "MSY"
   [70] "MSYd"                "MSYs"                "p"
   [73] "q"                   "r"                   "rc"
   [76] "rold"                "sdb"                 "sdc"
   [79] "sde"                 "sdf"                 "sdi"
```

```
[82] "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    0.0204040310 -0.005706714  0.026834550 -0.156740736  0.032240102
logK   -0.0057067141  0.037105155 -0.038074890 -0.011343169 -0.021784062
logq    0.0268345498 -0.038074890  0.091311212 -0.227633742  0.040958407
logn   -0.1567407362 -0.011343169 -0.227633742  1.473743320 -0.190011596
logsdb  0.0322401020 -0.021784062  0.040958407 -0.190011596  0.980089724
logsdf  0.0014253353 -0.002407435  0.003270276 -0.006648026 -0.002144228
logsdi -0.0007603784 -0.002521057  0.002848540  0.007158237  0.010535579
logsdc  0.0015535778  0.001300308 -0.008392544  0.001997815  0.137920109
              logsdf        logsdi       logsdc
logm    0.0014253353 -0.0007603784  0.001553578
logK   -0.0024074352 -0.0025210569  0.001300308
logq    0.0032702760  0.0028485396 -0.008392544
logn   -0.0066480258  0.0071582372  0.001997815
logsdb -0.0021442276  0.0105355795  0.137920109
logsdf  0.0262881975 -0.0002784384 -0.035159456
logsdi -0.0002784384  0.0237200085  0.005885858
logsdc -0.0351594556  0.0058858585  0.846803907
```

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 -0.207401076  0.62169056 -0.903884565  0.22798459
logK   -0.20740108  1.000000000 -0.65412309 -0.048507215 -0.11423228
logq    0.62169056 -0.654123089  1.00000000 -0.620531252  0.13691407
logn   -0.90388456 -0.048507215 -0.62053125  1.000000000 -0.15810160
logsdb  0.22798459 -0.114232276  0.13691407 -0.158101600  1.00000000
logsdf  0.06154300 -0.077082726  0.06674857 -0.033775467 -0.01335850
logsdi -0.03456323 -0.084978313  0.06120724  0.038285801  0.06909842
logsdc  0.01181908  0.007335634 -0.03018145  0.001788351  0.15139211
            logsdf      logsdi       logsdc
logm    0.06154300 -0.03456323  0.011819078
logK   -0.07708273 -0.08497831  0.007335634
logq    0.06674857  0.06120724 -0.030181454
logn   -0.03377547  0.03828580  0.001788351
logsdb -0.01335850  0.06909842  0.151392112
logsdf  1.00000000 -0.01115042 -0.235651555
logsdi -0.01115042  1.00000000  0.041529908
logsdc -0.23565156  0.04152991  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
```

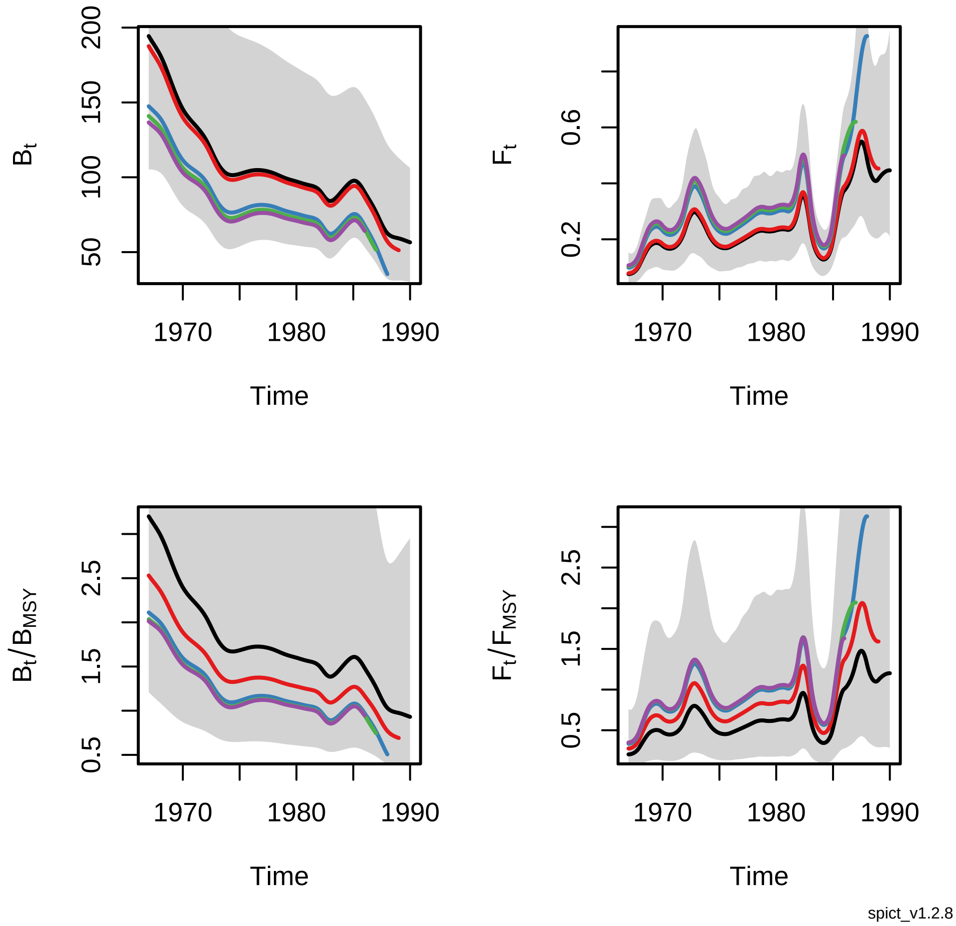
This reveals that for this data set the estimates of log Fmsy and log Bmsy are highly correlated. This is often the case and the reason why the model is reparameterised.

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 crudely performed using retro() as shown here

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



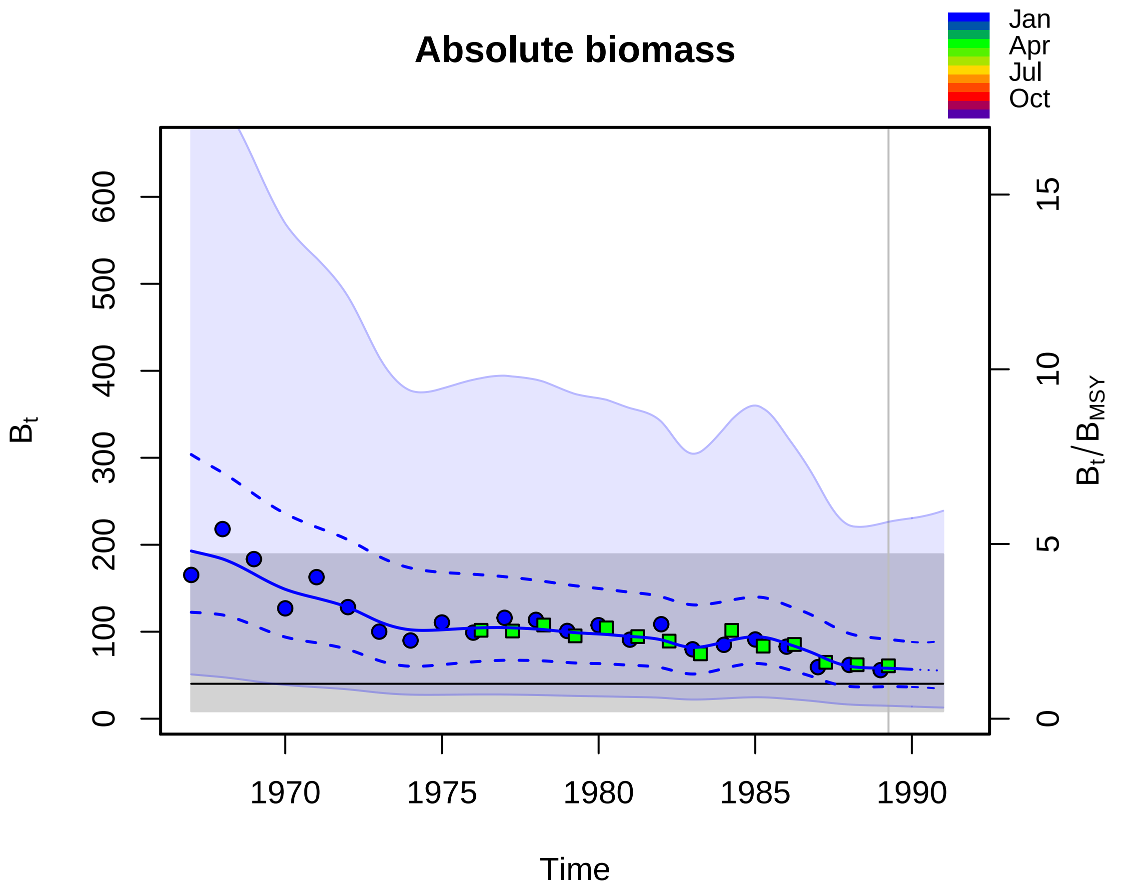
By default `retro` creates 5 scenarios with catch and index time series which are shortened by the 1 to 5 last observations. The number of scenarios and thus observations which are removed can be changed with the argument `nretroyear` in the function `retro`. The graphs show the different scenarios with different colors. For the albacore data, there is a high consistency between the scenarios except for the fishing mortalites of the second scenario (in red), which indicate a large increase in F.

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  5.54953258  0.94904896 32.4507091  1.7137137
  alpha2  3.45424294  0.58223835 20.4929722  1.2396033
  beta    0.12057029  0.01912936  0.7599414 -2.1155224
```

```
   r         0.18601586   0.08915985   0.3880884 -1.6819234
   rc        1.20960496   0.19387714   7.5467597  0.1902938
   rold      0.26864003   0.05296983   1.3624259 -1.3143830
   m        24.30176051  17.98198095  32.8426310  3.1905488
   K       220.56429948 148.87723001 326.7699849  5.3961893
   q1        0.35403669   0.22557139   0.5556644 -1.0383547
   q2        3.60405911   2.32437460   5.5882739  1.2820607
   n         0.30756464   0.03135788   3.0166586 -1.1790700
   sdb       0.02006319   0.00372273   0.1081282 -3.9088686
   sdf       0.36879354   0.26948768   0.5046935 -0.9975183
   sdi1      0.11134131   0.08158779   0.1519454 -2.1951549
   sdi2      0.06930312   0.04587566   0.1046944 -2.6692653
   sdc       0.04446554   0.00764489   0.2586282 -3.1130407
plotspict.biomass(res)
```

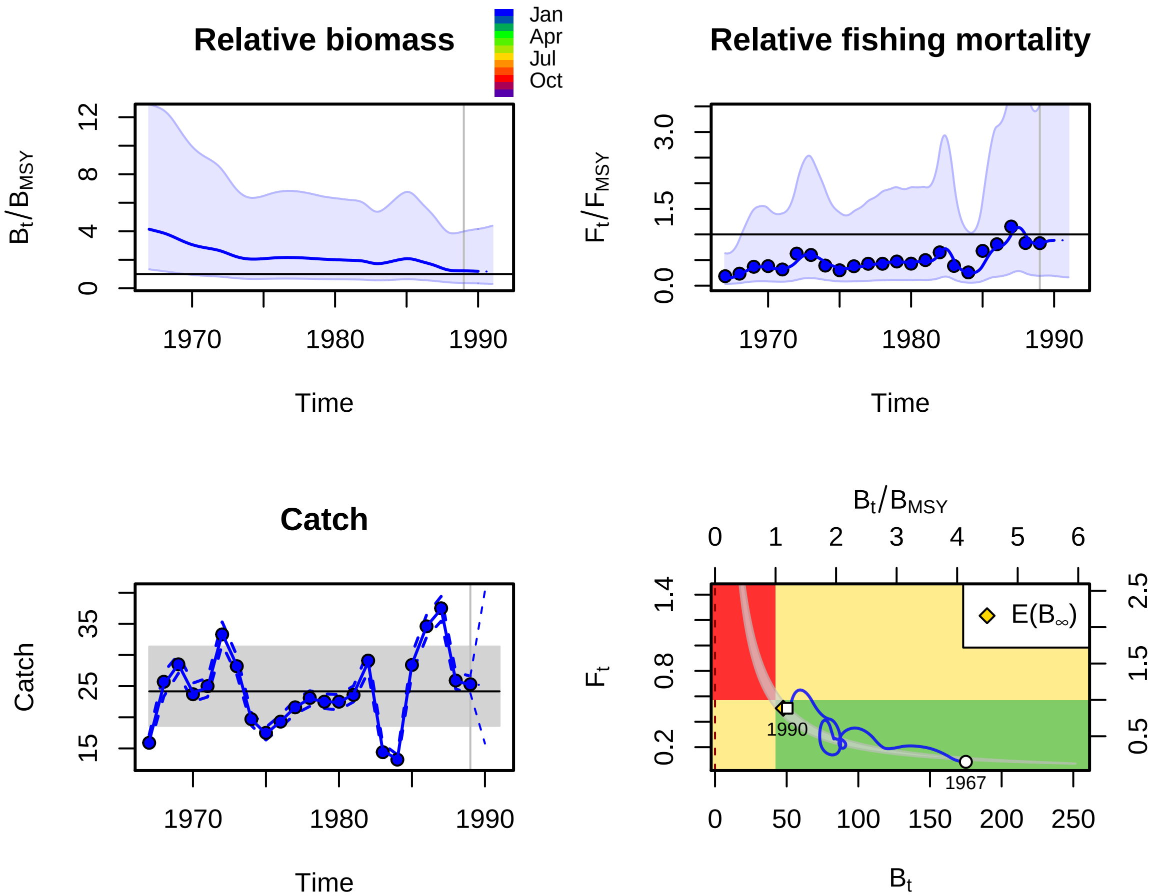


The model estimates seperate observation noises and finds that the first index (`sdi1`) is more noisy than the second (`sdi2`). It is furthermore estimated that the catchabilities are different by a factor 10 (`q1` versus `q2`). The biomass plot shows both indices, with circles indicating the first index 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     0.07385347   0.01464722   0.37238016 -2.6056723
   r        0.23822939   0.10656855   0.53255150 -1.4345212
   rc       1.14399163   0.21452271   6.10059826  0.1345236
   rold     0.40826820   0.03851360   4.32789751 -0.8958310
   m       24.16376129  18.62117901  31.35608973  3.1848540
   K      189.53177432 132.49527139 271.12132456  5.2445567
   qf       0.41117179   0.25562626   0.66136492 -0.8887442
   n        0.41648800   0.04180192   4.14962366 -0.8758976
   sdb      0.01430671   0.00236752   0.08645421 -4.2470266
   sdf      0.37625013   0.27938787   0.50669403 -0.9775011
   sde      0.09820098   0.06985342   0.13805240 -2.3207391
   sdc      0.02778738   0.00568262   0.13587724 -3.5831734
par(mfrow=c(2, 2))
plotspict.bbmsy(repeff)
plotspict.ffmsy(repeff, qlegend=FALSE)
plotspict.catch(repeff, qlegend=FALSE)
plotspict.fb(repeff)
```



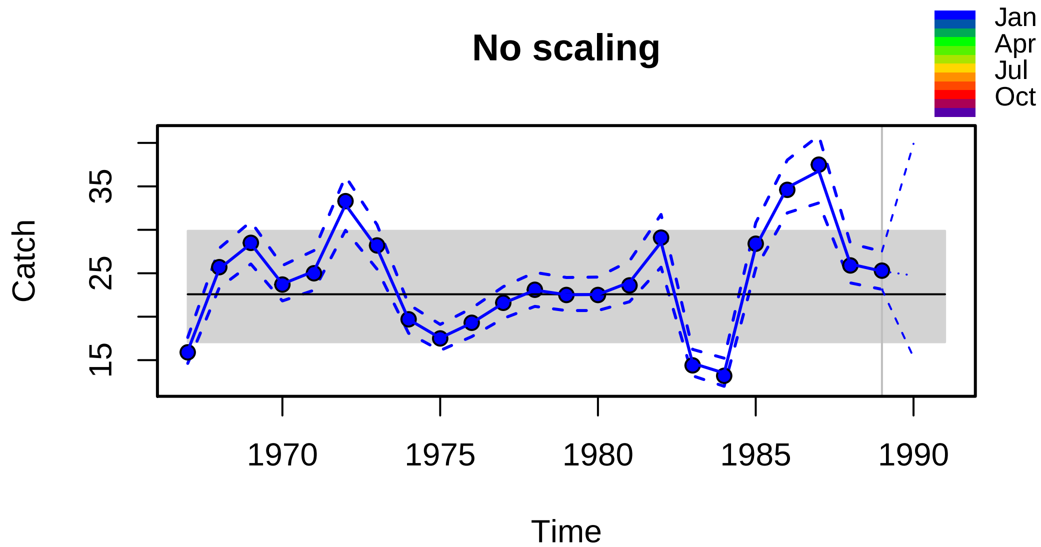
Here the model runs without an index of biomass and instead uses effort as an index of fishing mortality Note that index observations are missing from the biomass plot, but effort observations are present in the plot of fishing mortality. Note also that `q` is missing from the summary of parameter estimates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

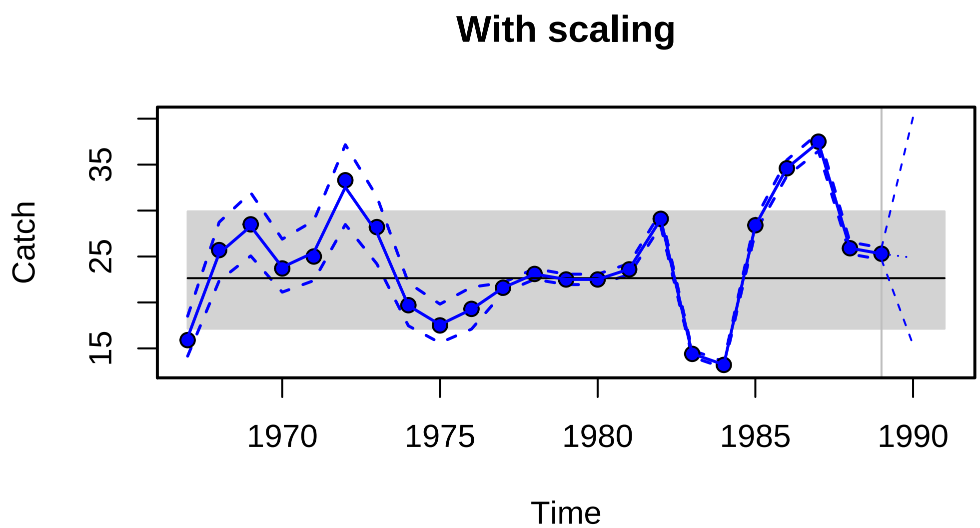
Overall for this data set the results in terms of stock status etc. do not change much, and this will probably often be the case, however using effort data directly instead of commercial CPUE is cleaner and avoids inputting the same data twice.

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.

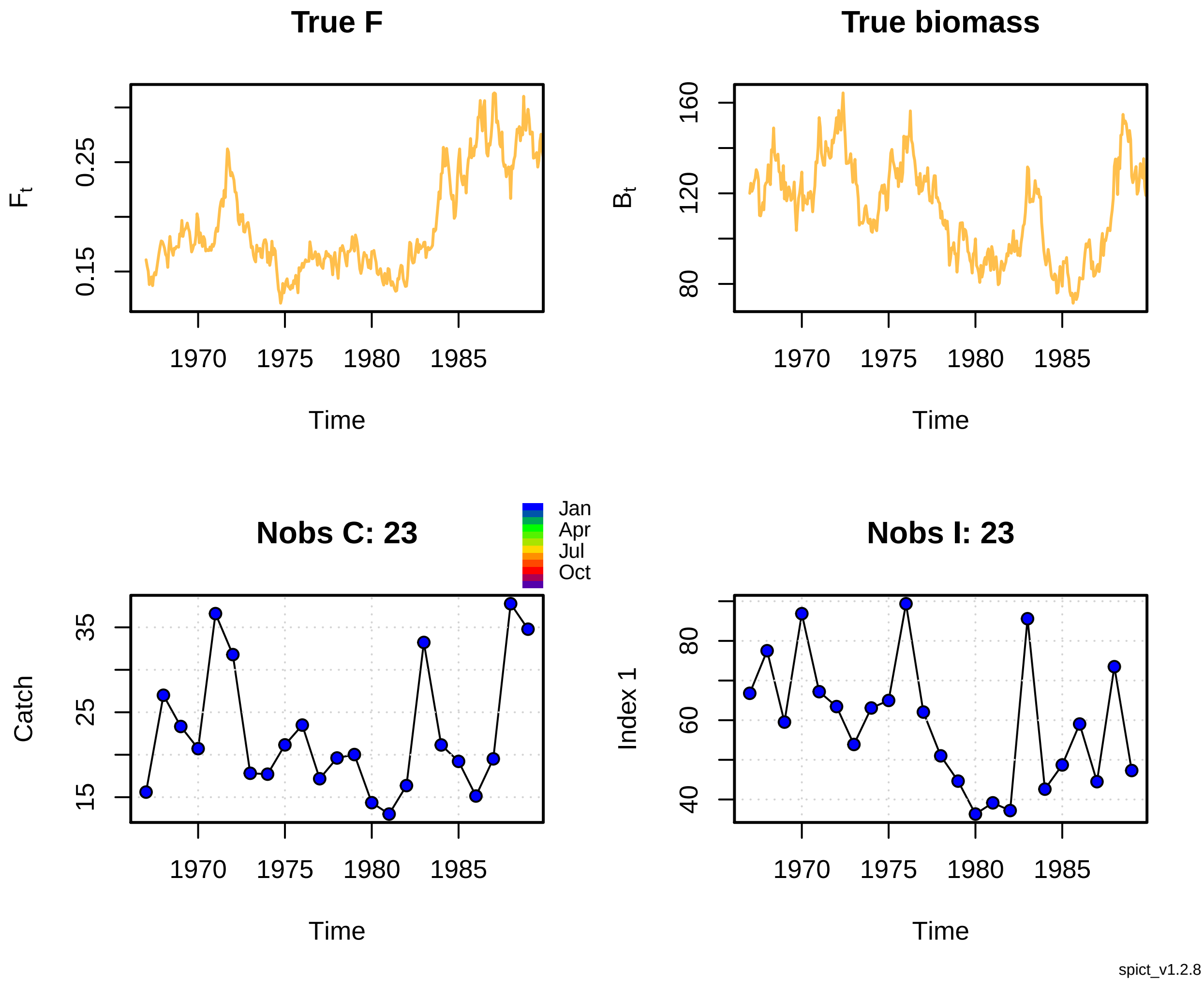
Simulating data

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

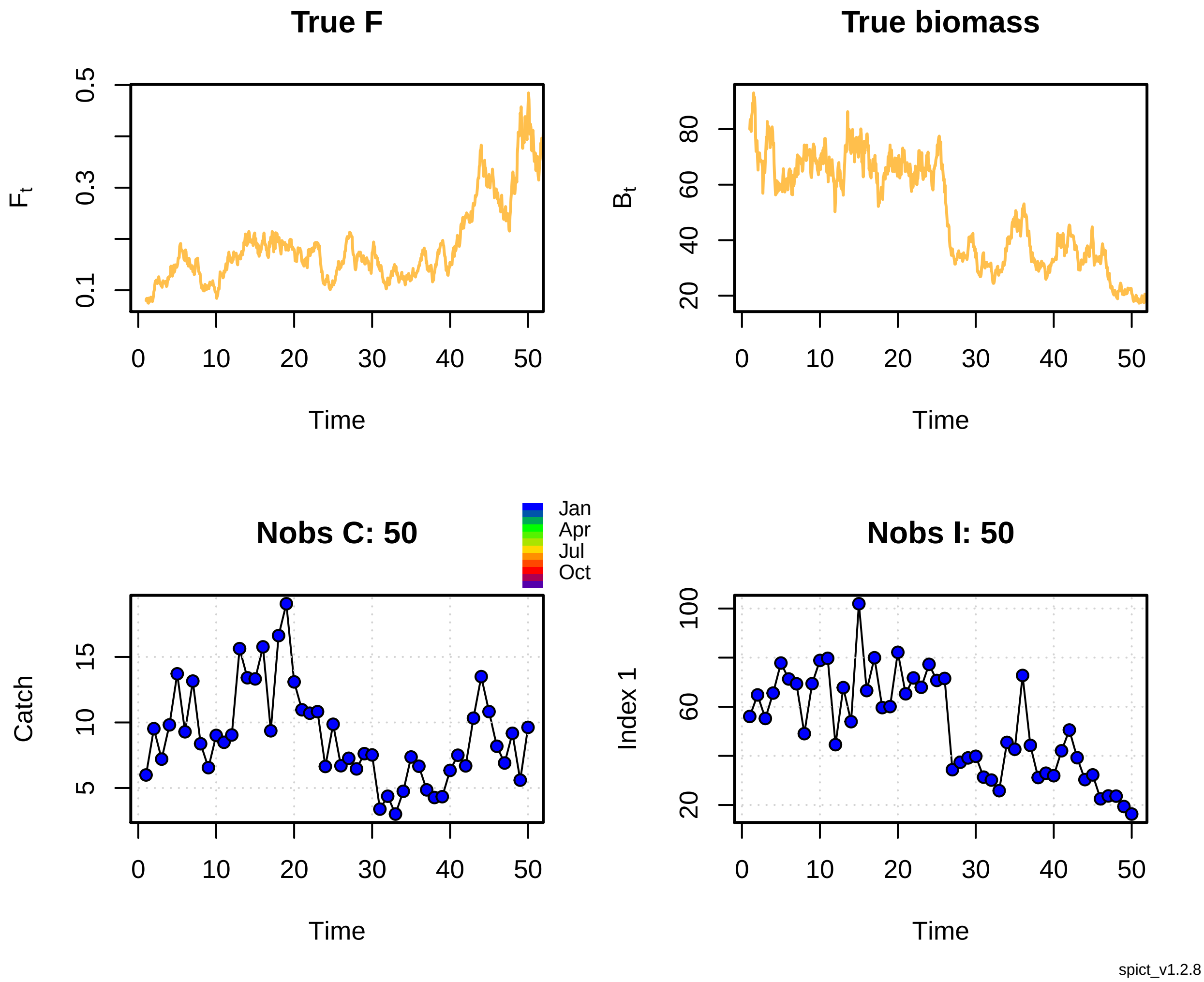
```
inp <- check.inp(pol$albacore)
sim <- sim.spict(inp)
plotspict.data(sim)
```



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)
```

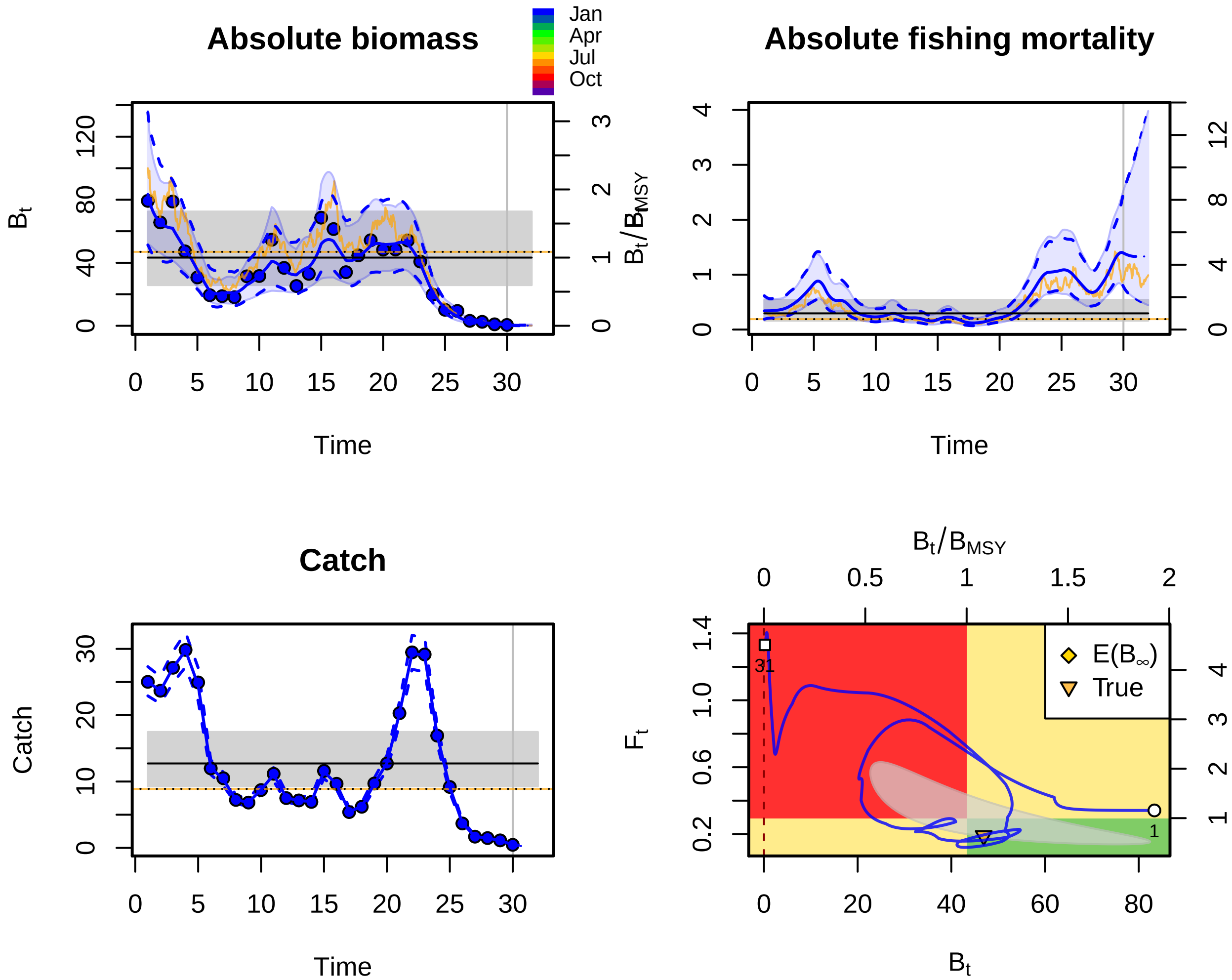


Here the required parameters are specified (the rest use default values), and the number of observations is specified as an argument to `sim.spict()`.

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    1.04607711  -9.0  0.31604357   3.4624255         -9  0.04504709
   beta     0.13191757  -9.0  0.02269878   0.7666599         -9 -2.02557801
   r        1.07672251  -9.0  0.35061486   3.3065666         -9  0.07392172
   rc       0.63474118  -9.0  0.34468261   1.1688909         -9 -0.45453795
   rold     0.45001540  -9.0  0.22252978   0.9100528         -9 -0.79847347
   m       14.48076844  10.0 10.46872828  20.0303847          0  2.67282145
   K       76.02606890 100.0 46.11318791 125.3429531          1  4.33107629
   q        1.19617687   1.0  0.85013702   1.6830688          1  0.17913053
   n        3.39263481   2.0  1.36225519   8.4492032          1  1.22160685
   sdb      0.19525936   0.2  0.08857955   0.4304178          1 -1.63342656
```

```
    sdf      0.34363008   0.3  0.25198736   0.4686014           1 -1.06818954
    sdi      0.20425635   0.2  0.12166919   0.3429024           1 -1.58837948
    sdc      0.04533085   0.1  0.00814469   0.2522975           1 -3.09376755
par(mfrow=c(2, 2))
plotspict.biomass(res)
plotspict.f(res, qlegend=FALSE)
plotspict.catch(res, qlegend=FALSE)
plotspict.fb(res)
```



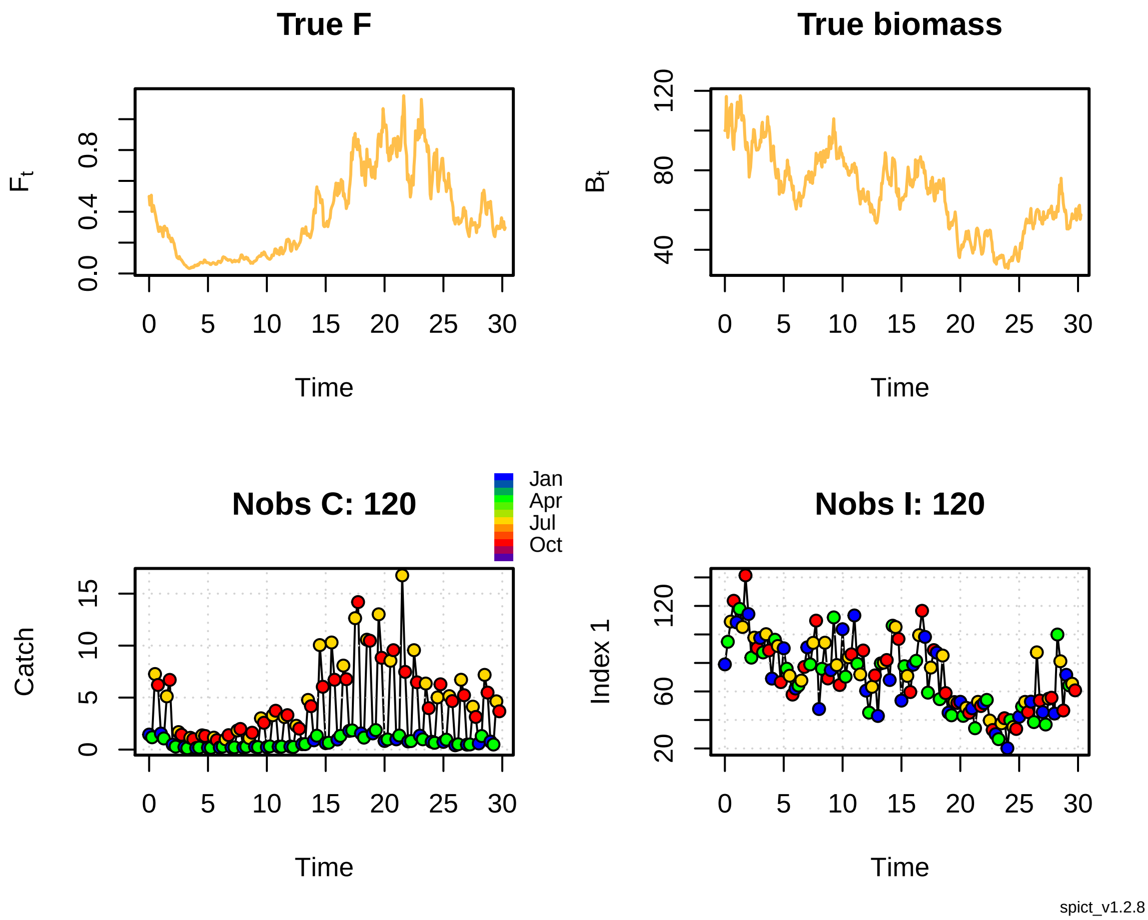
Here the ratio between biomass in the initial year relative to `K` is set using `logbkfrac`, the initial fishing mortality is set using `logF0`, process noise of `F` is set using `logsdf`, and finally observation noise on catches is specified using `logsdc`.

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

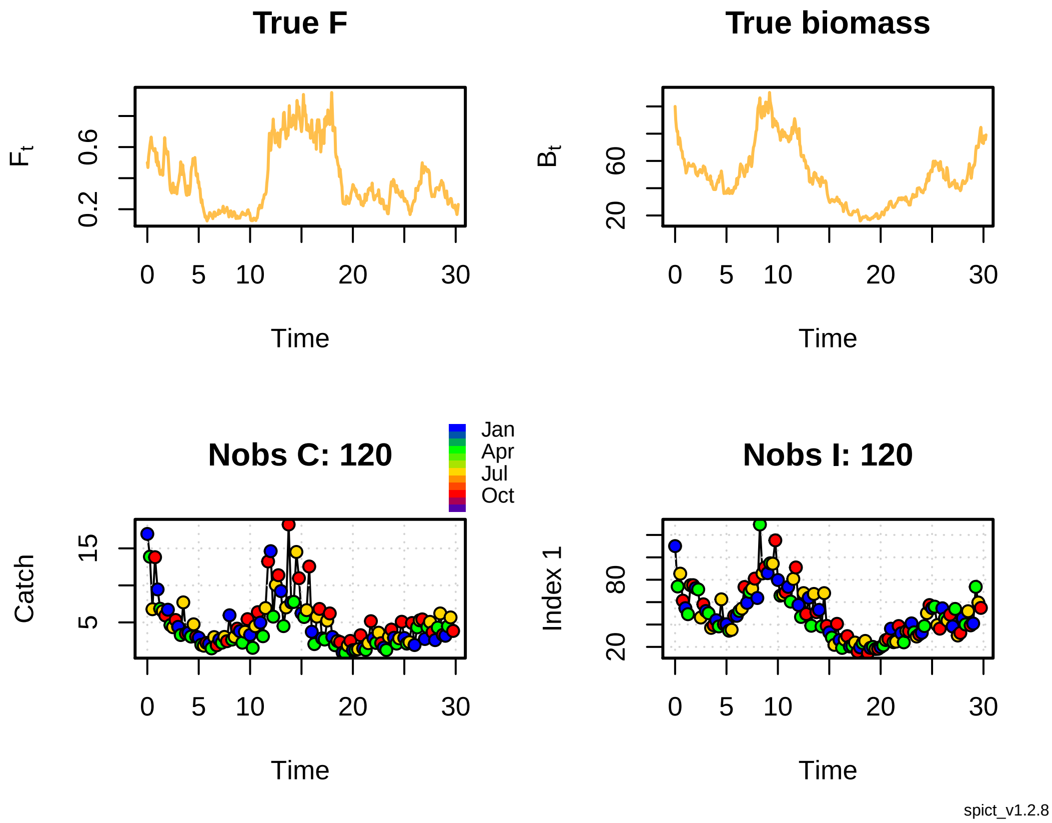
It is possible to simulate seasonal data (most often quarterly). Additional variables must be specified in the input list that define the type of seasonality to be used. Spline based seasonality is shown first (`inp$seasontype = 1`). This is the default and therefore does not need to be explicitly specified.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

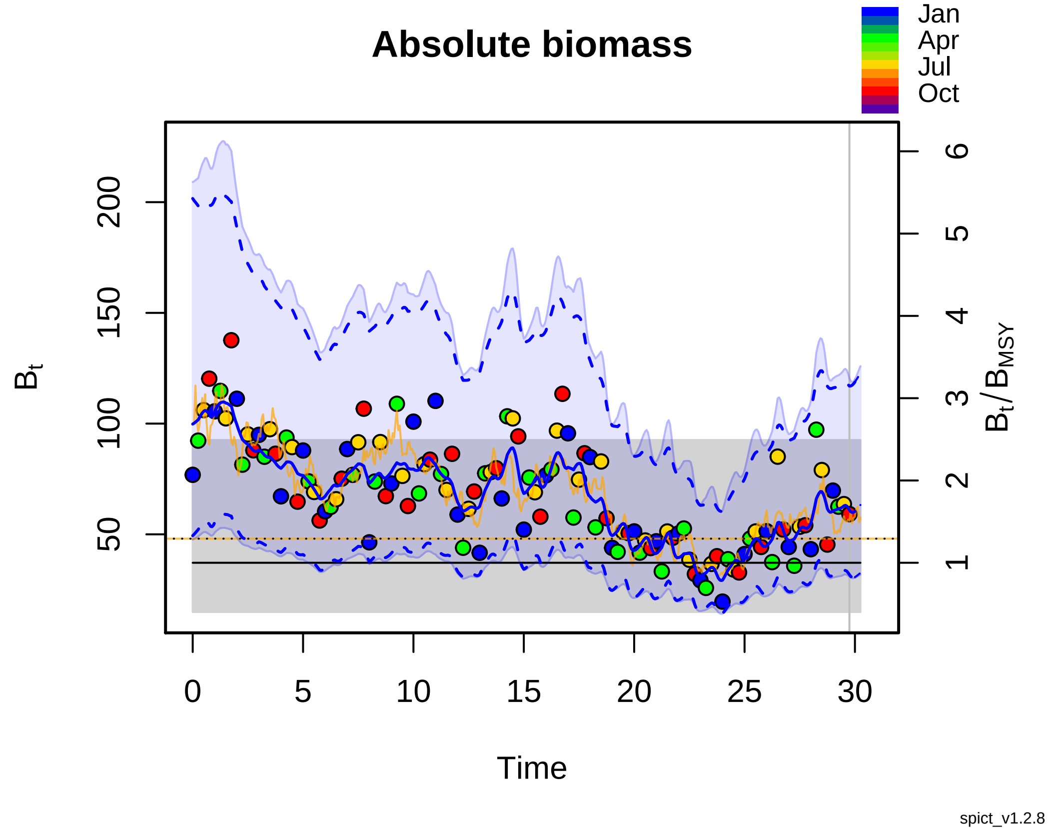
Catch information available in sub-annual aggregations, e.g. quarterly catch, can be used to estimate the seasonal pattern of the fishing mortality. The user can choose between two types of seasonality by setting `seasontype` to 1 or 2:

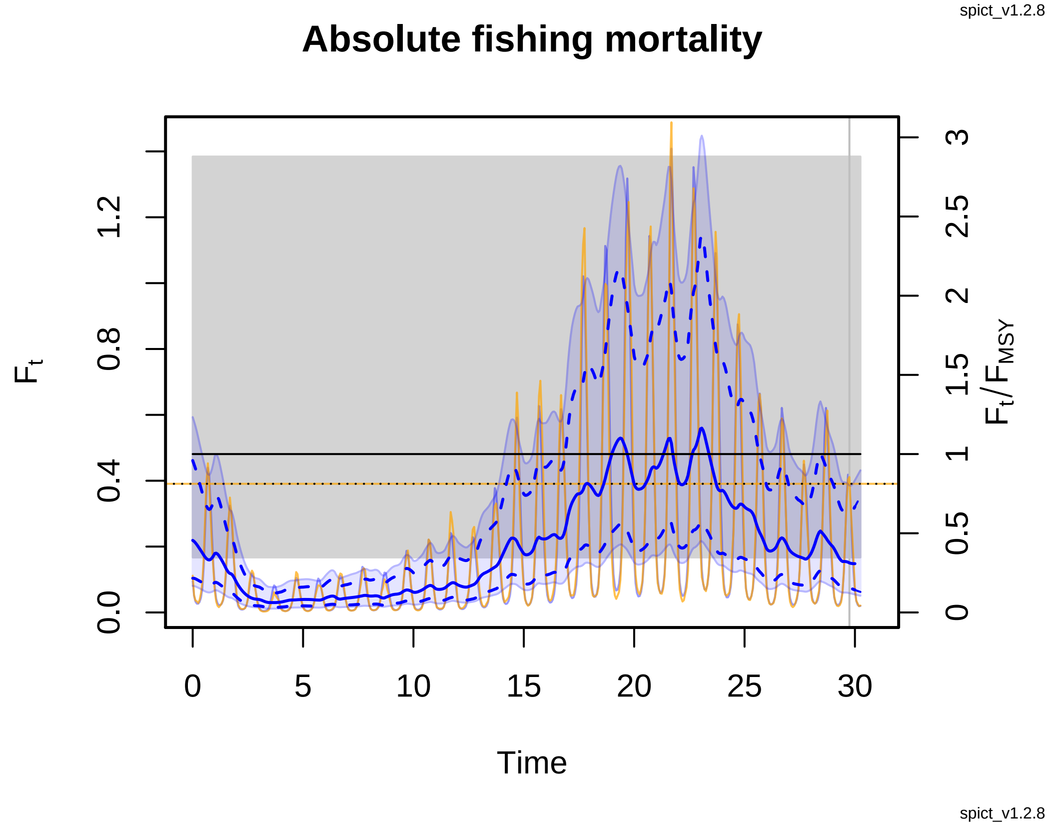
1. using cyclic B-splines.
2. using coupled stochastic differential equations (SDEs).

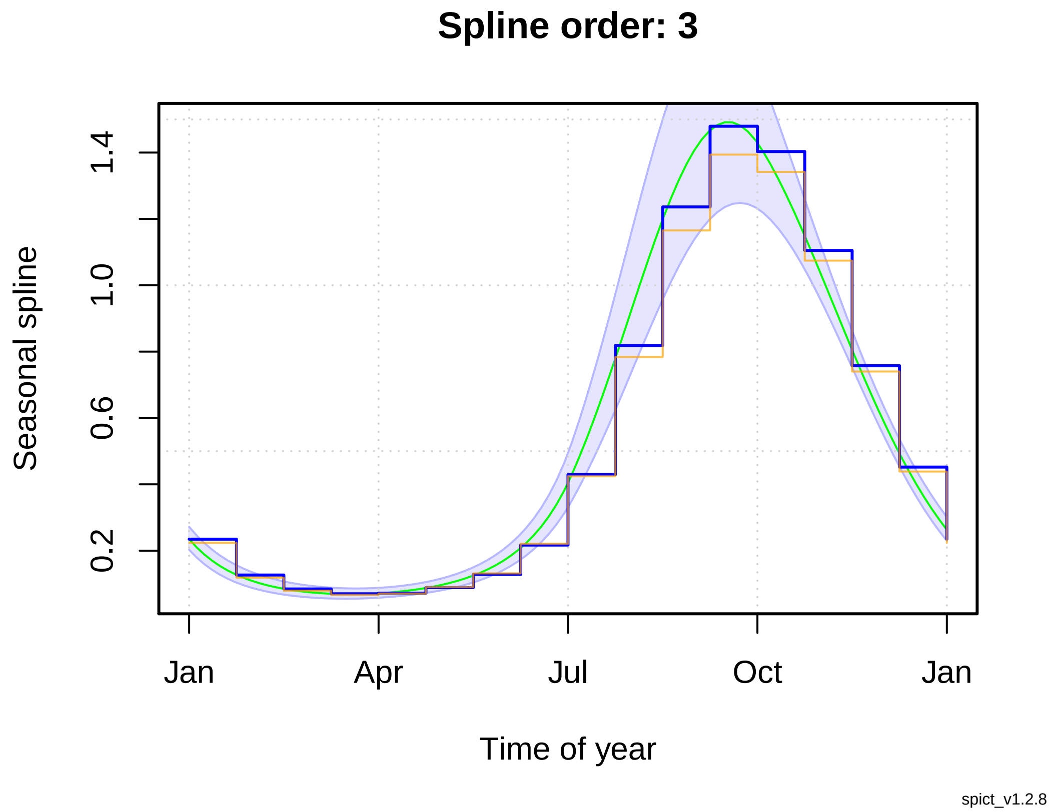
Technical description of the season types is found in Pedersen and Berg (2017).

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
Time
spict_v1.2.8

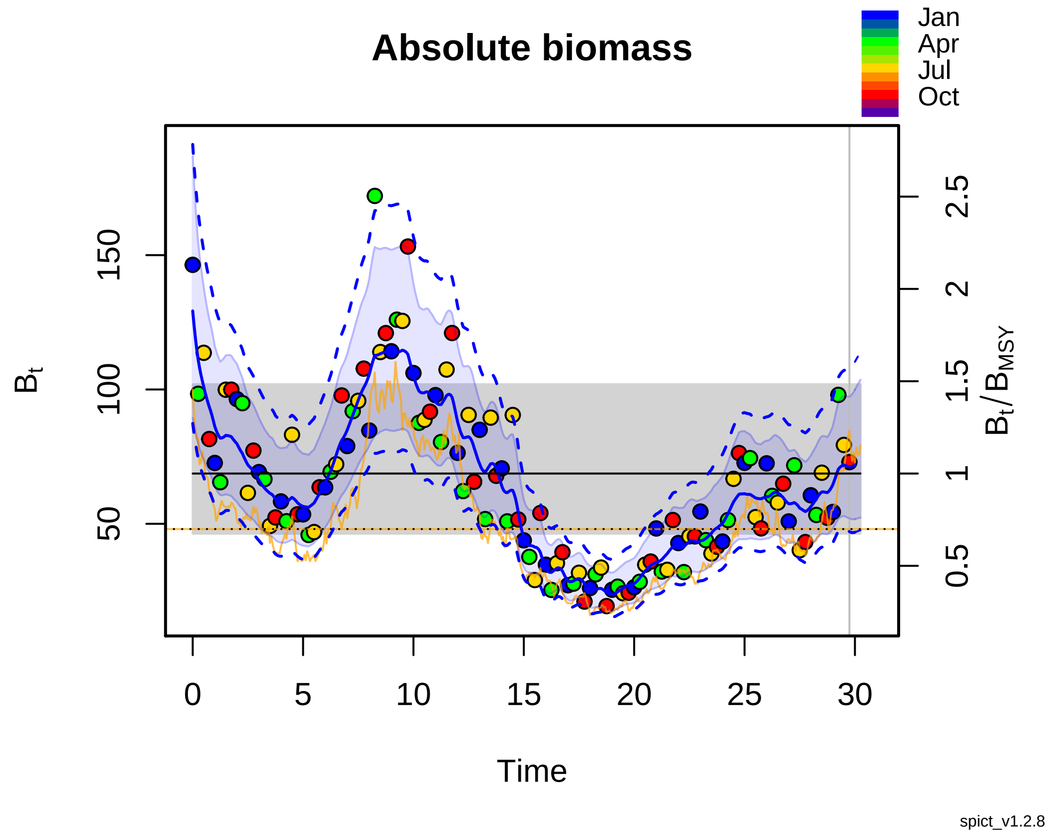

Absolute fishing mortality
F_t
F_t/F_MSY
Time
spict_v1.2.8

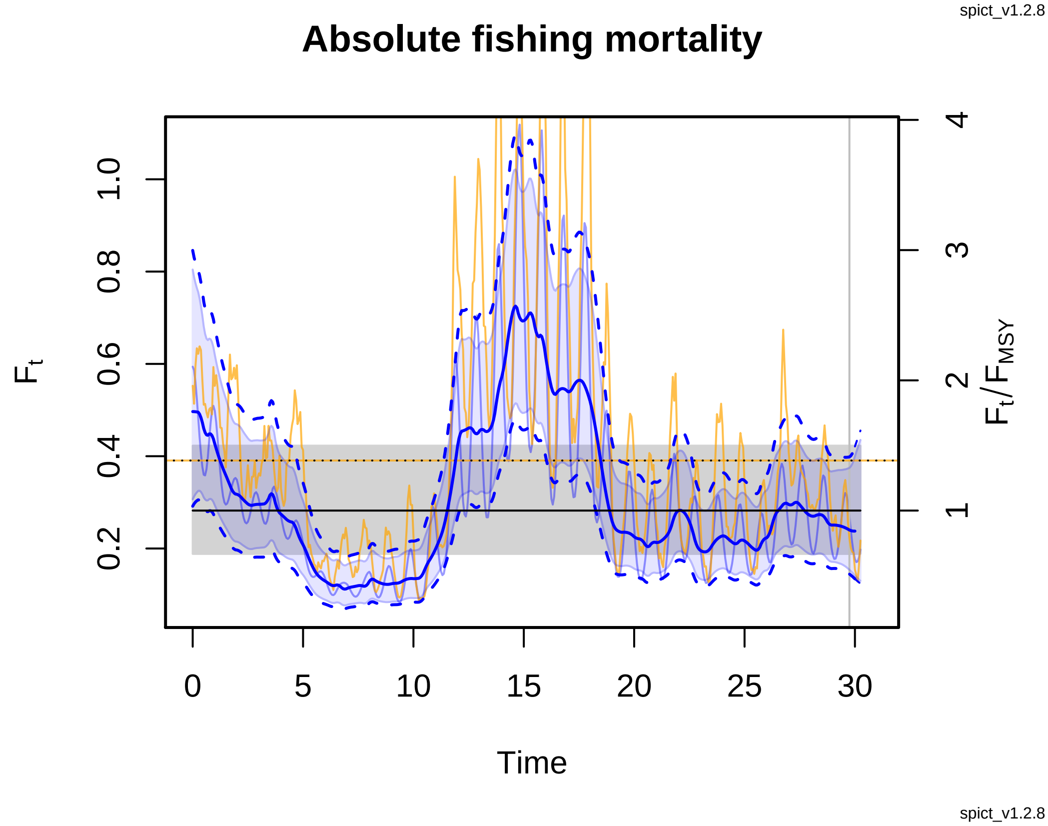


The model is able to estimate the seasonal variation in fishing mortality as seen both in the plot of `F` and in the plot of the estimated spline, where blue is the estimated spline, orange is the true spline, and green is the spline if time were truly continuous (it is discretised with the Euler steps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

To fit the coupled SDE model run

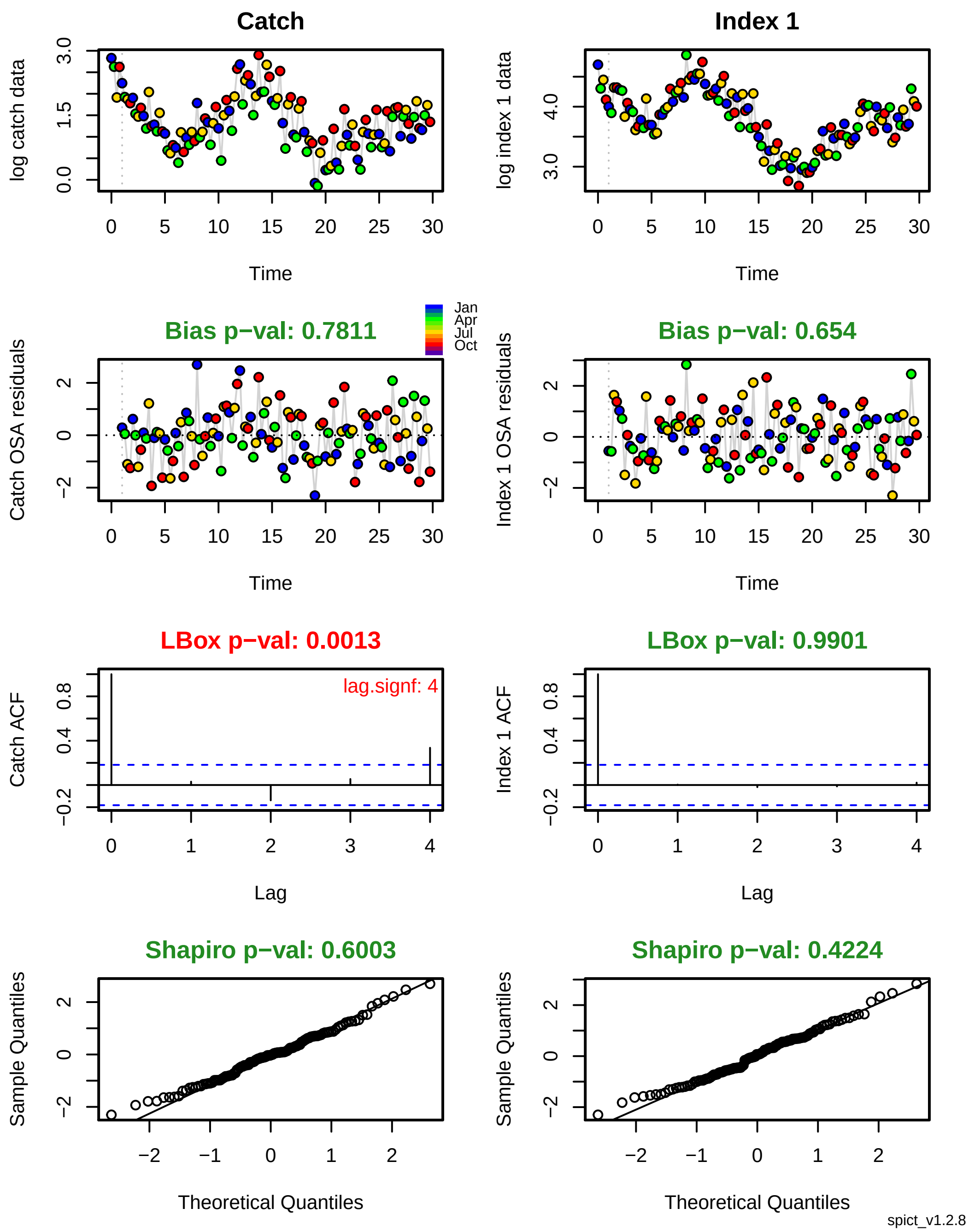
```
seasonres2 <- fit.spict(seasonsim2)
sumspict.parest(seasonres2)
              estimate  true       cilow       ciupp true.in.ci    log.est
   alpha    1.38654514  -9.0  0.74646650   2.5754772         -9  0.3268151
   beta     0.69812209  -9.0  0.46862508   1.0400093         -9 -0.3593613
   r        0.75461231  -9.0  0.19653878   2.8973404         -9 -0.2815512
   rc       0.57786149  -9.0  0.38165597   0.8749343         -9 -0.5484211
   rold     0.46819697  -9.0  0.22851759   0.9592627         -9 -0.7588662
   m       20.30400512  20.0 15.39627328  26.7761306          1  3.0108182
   K      127.48850146 100.0 77.62144049 209.3921203          1  4.8480262
   q        0.75175502   1.0  0.51602963   1.0951612          1 -0.2853448
   n        2.61174111   2.0  0.81672645   8.3518682          1  0.9600171
   sdb      0.13137017   0.2  0.07696246   0.2242408          1 -2.0297362
   sdu      0.10517086   0.1  0.05695791   0.1941944          1 -2.2521690
   sdf      0.31639458   0.4  0.23247974   0.4305989          1 -1.1507652
   sdi      0.18215067   0.2  0.15479277   0.2143438          1 -1.7029211
   sdc      0.22088205   0.2  0.18154286   0.2687458          1 -1.5101265
   lambda   0.06185887   0.1  0.00855678   0.4471918          1 -2.7828998
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: 5.8647428
   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   1.1440113 -1.7638034
    r        0.3471988  0.1465244   0.8227093 -1.0578579
    rc       1.7791121  0.2676089  11.8278559  0.5761144
    rold     0.5694636  0.0689479   4.7033865 -0.5630604
    m       27.4846402 18.9163563  39.9339827  3.3136273
    K      144.5708243 82.7912265 252.4509434  4.9737695
    q        0.4966500  0.2541097   0.9706881 -0.6998696
    n        0.3903056  0.0392947   3.8768218 -0.9408251
    sdf      0.3738951  0.2594015   0.5389235 -0.9837801
    sdi      0.1050361  0.0679152   0.1624466 -2.2534509
    sdc      0.0640825  0.0113802   0.3608516 -2.7475835

   Deterministic reference points (Drp)
            estimate      cilow      ciupp    log.est
    Bmsyd 30.8970294  6.2838311 151.917901  3.4306600
    Fmsyd  0.8895561  0.1338045   5.913928 -0.1170327
    MSYd  27.4846402 18.9163563  39.933983  3.3136273
   Stochastic reference points (Srp)
            estimate      cilow      ciupp   log.est   rel.diff.Drp
    Bmsys 30.8170212  6.3481131 149.601746  3.428067 -0.0025962349
    Fmsys  0.8901022  0.1349377   5.871466 -0.116419  0.0006135542
    MSYs  27.4303408 18.8037385  40.014575  3.311650 -0.0019795378

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

```
                 estimate      cilow     ciupp    log.est
 B_1989.00      43.5729051 21.6014377 87.892208  3.7744355
 F_1989.00       0.5677372  0.2613795  1.233170 -0.5660967
 B_1989.00/Bmsy  1.4139233  0.3592721  5.564527  0.3463683
 F_1989.00/Fmsy  0.6378337  0.1112640  3.656454 -0.4496777

Predictions w 95% CI (inp$msytype: s)
                prediction      cilow     ciupp    log.est
 B_1990.00      44.7705544 21.9026960 91.513965  3.8015507
 F_1990.00       0.5737431  0.2522597  1.304930 -0.5555735
 B_1990.00/Bmsy  1.4527866  0.3371972  6.259213  0.3734835
 F_1990.00/Fmsy  0.6445812  0.1037380  4.005138 -0.4391545
 Catch_1990.00  25.8407642 15.9509270 41.862463  3.2519533
 E(B_inf)       45.9909037         NA        NA  3.8284436
```

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: 5.0598288
   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   39.0512849   0.0402366 3.790087e+04  3.6648758
    beta     0.0245071   0.0000265 2.269298e+01 -3.7087912
    r        0.1955750   0.0313372 1.220582e+00 -1.6318113
    rc       1.2604800   0.0097765 1.625131e+02  0.2314926
    rold     0.2835728   0.0024028 3.346685e+01 -1.2602862
    m       24.3112479  12.0981938 4.885331e+01  3.1909391
```

```
 K      210.4516033 123.9786439 3.572380e+02  5.3492557
 q        0.3842438   0.2070701 7.130114e-01 -0.9564779
 n        0.3103183   0.0004170 2.309139e+02 -1.1701567
 sdb      0.0028040   0.0000029 2.728492e+00 -5.8767196
 sdf      0.3809116   0.2827808 5.130959e-01 -0.9651879
 sdi      0.1094986   0.0812449 1.475777e-01 -2.2118438
 sdc      0.0093351   0.0000103 8.475652e+00 -4.6739791

Deterministic reference points (Drp)
       estimate      cilow      ciupp    log.est
 Bmsyd 38.57459  0.5969768 2492.55706  3.6525937
 Fmsyd  0.63024  0.0048883   81.25653 -0.4616546
 MSYd  24.31125 12.0981938   48.85331  3.1909391
Stochastic reference points (Srp)
         estimate      cilow      ciupp    log.est  rel.diff.Drp
 Bmsys 38.5744682  0.5969864 2492.50152  3.6525906 -3.107788e-06
 Fmsys  0.6302411  0.0048883   81.25604 -0.4616529  1.767276e-06
 MSYs  24.3112135 12.0980421   48.85378  3.1909377 -1.413324e-06

States w 95% CI (inp$msytype: s)
                   estimate      cilow       ciupp    log.est
 B_1989.00      54.0126818 28.3907697 102.7576858  3.9892189
 F_1989.00       0.4528693  0.2235663   0.9173593 -0.7921517
 B_1989.00/Bmsy  1.4002184  0.0314790  62.2831497  0.3366283
 F_1989.00/Fmsy  0.7185652  0.0079263  65.1417912 -0.3304988

Predictions w 95% CI (inp$msytype: s)
                 prediction      cilow      ciupp    log.est
 B_1990.00      52.5650582 29.3174853 94.2470105  3.9620516
 F_1990.00       0.4858020  0.2370661  0.9955179 -0.7219542
 B_1990.00/Bmsy  1.3626904  0.0266105 69.7816085  0.3094610
 F_1990.00/Fmsy  0.7708193  0.0076917 77.2474938 -0.2603013
 Catch_1990.00  25.2158689 15.5407630 40.9143388  3.2274735
 E(B_inf)       49.5039259         NA         NA  3.9020520
```

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"        "BmsyB0"    "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: 3.697211
```

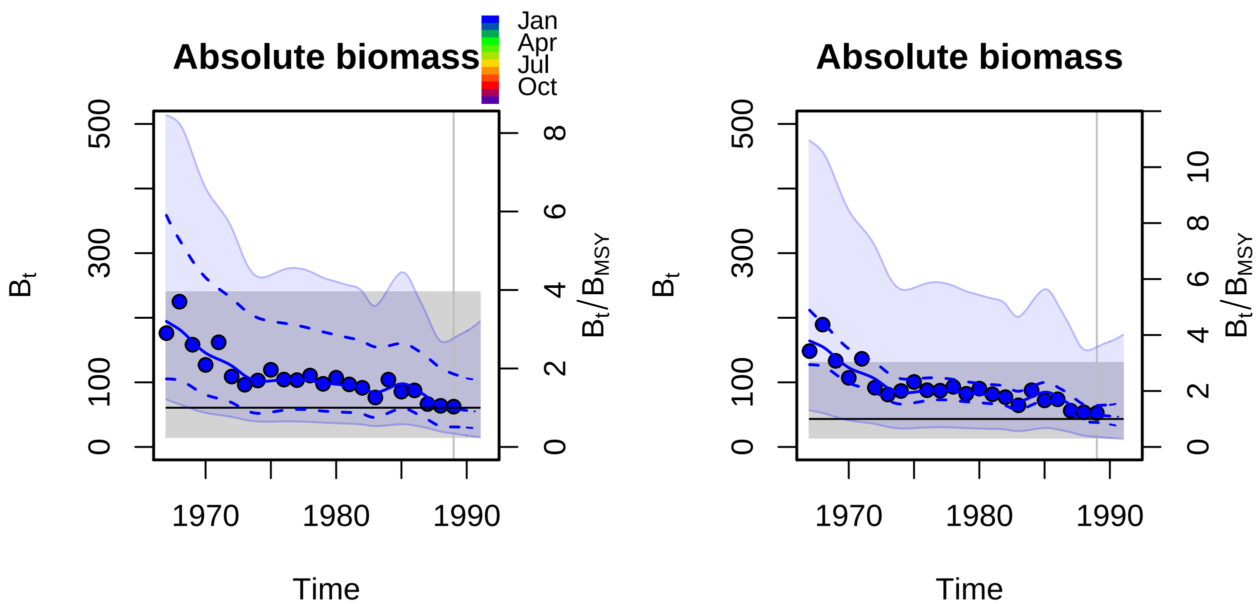
```
Euler time step (years):  1/16 or 0.0625
Nobs C: 23,  Nobs I1: 23

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    8.5541219   1.2276016  59.6064714  2.1464133
 beta     0.1213066   0.0180837   0.8137331 -2.1094342
 r        0.2543931   0.0999822   0.6472737 -1.3688747
 rc       0.7408820   0.1423248   3.8567139 -0.2999139
 rold     0.8120577   0.0018384 358.7037398 -0.2081839
 m       22.5701177  17.0283130  29.9154832  3.1166268
 K      202.2160641 138.6995451 294.8195435  5.3093367
 q        0.3499366   0.1930339   0.6343737 -1.0500034
 n        0.6867303   0.0623502   7.5637086 -0.3758136
 sdb      0.0127865   0.0018399   0.0888615 -4.3593656
 sdf      0.3672885   0.2672937   0.5046914 -1.0016078
 sdi      0.1093772   0.0808912   0.1478947 -2.2129524
 sdc      0.0445545   0.0073418   0.2703825 -3.1110420

Deterministic reference points (Drp)
        estimate      cilow      ciupp    log.est
 Bmsyd 60.927699 15.2912403 242.765427  4.1096879
 Fmsyd  0.370441  0.0711624   1.928357 -0.9930611
 MSYd  22.570118 17.0283130  29.915483  3.1166268
Stochastic reference points (Srp)
         estimate      cilow      ciupp    log.est  rel.diff.Drp
 Bmsys 60.9200355 15.2914370 242.701241  4.1095621 -1.257924e-04
 Fmsys  0.3704531  0.0711556   1.928668 -0.9930283  3.266221e-05
 MSYs  22.5680187 17.0227429  29.919706  3.1165338 -9.300660e-05

States w 95% CI (inp$msytype: s)
                   estimate      cilow       ciupp    log.est
 B_1989.00       59.4317253 31.0677432 113.6912308  4.0848282
 F_1989.00        0.4143985  0.2035123   0.8438121 -0.8809271
 B_1989.00/Bmsy   0.9755694  0.3412397   2.7890532 -0.0247339
 F_1989.00/Fmsy   1.1186260  0.2875234   4.3520786  0.1121012

Predictions w 95% CI (inp$msytype: s)
                 prediction      cilow       ciupp    log.est
 B_1990.00       56.7523819 30.1059283 106.9833428  4.0386976
 F_1990.00        0.4446518  0.2086071   0.9477876 -0.8104637
 B_1990.00/Bmsy   0.9315881  0.2917620   2.9745352 -0.0708645
 F_1990.00/Fmsy   1.2002917  0.2809798   5.1274160  0.1825646
 Catch_1990.00   24.7355116 15.3286450  39.9151741  3.2082399
 E(B_inf)        50.1634075         NA          NA  3.9152858
```

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: -13.3777183
   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 -0.6839471
    rc        0.5046222  0.1875581   1.3576783 -0.6839453
    rold      0.5046231  0.1875573   1.3576886 -0.6839435
    m        22.0086909 17.0613914  28.3905610  3.0914374
    K       174.4569110 74.7554134 407.1305663  5.1616778
    q         0.3549421  0.1278982   0.9850325 -1.0358005
    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 -1.5595721
    sdi       0.0966010  0.0704060   0.1325419 -2.3371666
    sdc       0.2102255  0.1562923   0.2827699 -1.5595745

   Deterministic reference points (Drp)
             estimate     cilow       ciupp   log.est
    Bmsyd 87.2283941 37.377636 203.5653831  4.468530
    Fmsyd  0.2523111  0.093779   0.6788392 -1.377093
    MSYd  22.0086909 17.061391  28.3905610  3.091437
   Stochastic reference points (Srp)
             estimate      cilow      ciupp   log.est rel.diff.Drp
    Bmsys 86.1721784 37.1707121 199.771377  4.456347 -0.012257038
```

```
 Fmsys  0.2500268  0.0921861   0.678122 -1.386187 -0.009136167
 MSYs  21.5429413 16.5473161  28.046743  3.070048 -0.021619592

States w 95% CI (inp$msytype: s)
                 estimate      cilow      ciupp    log.est
 B_1989.00      59.8081015 20.7927035 172.031934  4.0911411
 F_1989.00       0.4264686  0.1528595   1.189821 -0.8522164
 B_1989.00/Bmsy  0.6940535  0.4759258   1.012154 -0.3652062
 F_1989.00/Fmsy  1.7056917  1.0390604   2.800015  0.5339707

Predictions w 95% CI (inp$msytype: s)
               prediction      cilow      ciupp    log.est
 B_1990.00      54.5156897 17.2286845 172.500716  3.9984885
 F_1990.00       0.4313747  0.1427056   1.303973 -0.8407781
 B_1990.00/Bmsy  0.6326368  0.3741020   1.069840 -0.4578588
 F_1990.00/Fmsy  1.7253140  0.9361420   3.179762  0.5454091
 Catch_1990.00  22.6023846 14.8441308  34.415474  3.1180554
 E(B_inf)       20.7049018         NA         NA  3.0303705
```

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

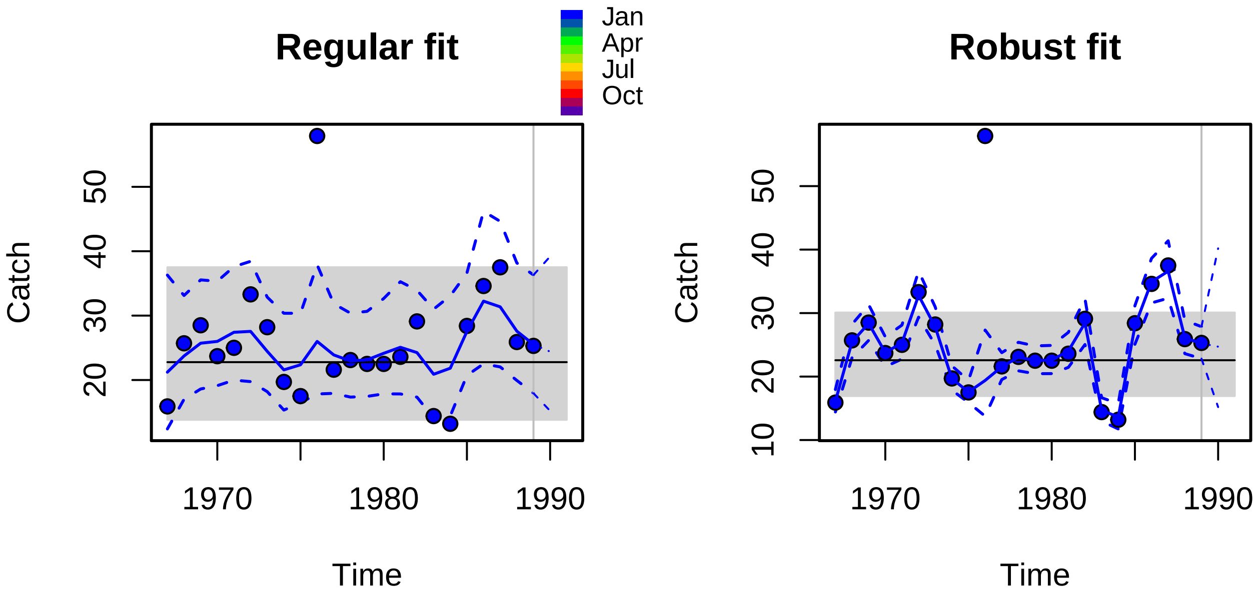
Particular caution is required when fixing a parameter that is highly correlated with other parameters because this will to some extent restrict the estimates of the correlated parameters. This could also be a problem when specifying priors depending on the amount of a priori information available.

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    8.01383375   1.14064160  56.30298905  2.0811693
   beta     0.13903414   0.02002425   0.96535421 -1.9730357
   r        0.25685420   0.10125070   0.65159133 -1.3592467
   rc       0.73334989   0.13978955   3.84722636 -0.3101324
   rold     0.85759756   0.00140610 523.05882804 -0.1536203
   m       22.57344335  16.94359774  30.07391657  3.1167741
   K      202.06196157 137.06067493 297.89023245  5.3085744
   q        0.34766878   0.18846113   0.64137140 -1.0565050
   n        0.70049565   0.06408927   7.65641657 -0.3559671
   sdb      0.01371149   0.00196485   0.09568409 -4.2895209
   sdf      0.37086365   0.26568565   0.51767886 -0.9919208
   sdi      0.10988163   0.08106901   0.14893450 -2.2083516
   sdc      0.05156271   0.00843494   0.31520244 -2.9649566
   pp       0.95304961   0.72886830   0.99351826  3.0105755
```

```
   robfac  20.83563743   2.67330915 236.13438175  2.9874802
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

To make a catch forecast a forecast interval needs to be specified. This is done by specifying the start of the interval (`inp$timepredc`) and the length of the interval in years (`inp$dtpredc`). For example, if a forecast of the annual catch of 2018 is of interest, then `inp$timepredc = 2018` and `inp$dtpredc = 1`. In addition to the forecast interval a fishing scenario needs to be specified. This is done by specifying a factor (`inp$ffac`) to multiply the current fishing mortality by (i.e. the F at the last time point of the time period where data are available) and the time that management should start (`inp$manstart`). The time point of the reported forecast of biomass and fishing mortality can be controlled by setting `inp$timepredi`. Producing short-term forecasts entails minimal additional computing time.

Forecasts are produced as part of the usual model fitting. To illustrate the procedure, a short example using the South Atlantic albacore dataset of Polacheck, Hilborn, and Punt (1993) containing catch and commercial CPUE data in the interval 1967 to 1989 is presented. The code to obtain the forecasted annual catch in the interval starting 1991 under a management scenario where the fishing pressure is reduced by 25% starting in 1991, and a forecasted index in 1992 is:

```
library(spict)
data(pol)
inp <- pol$albacore
inp$manstart <- 1991
```

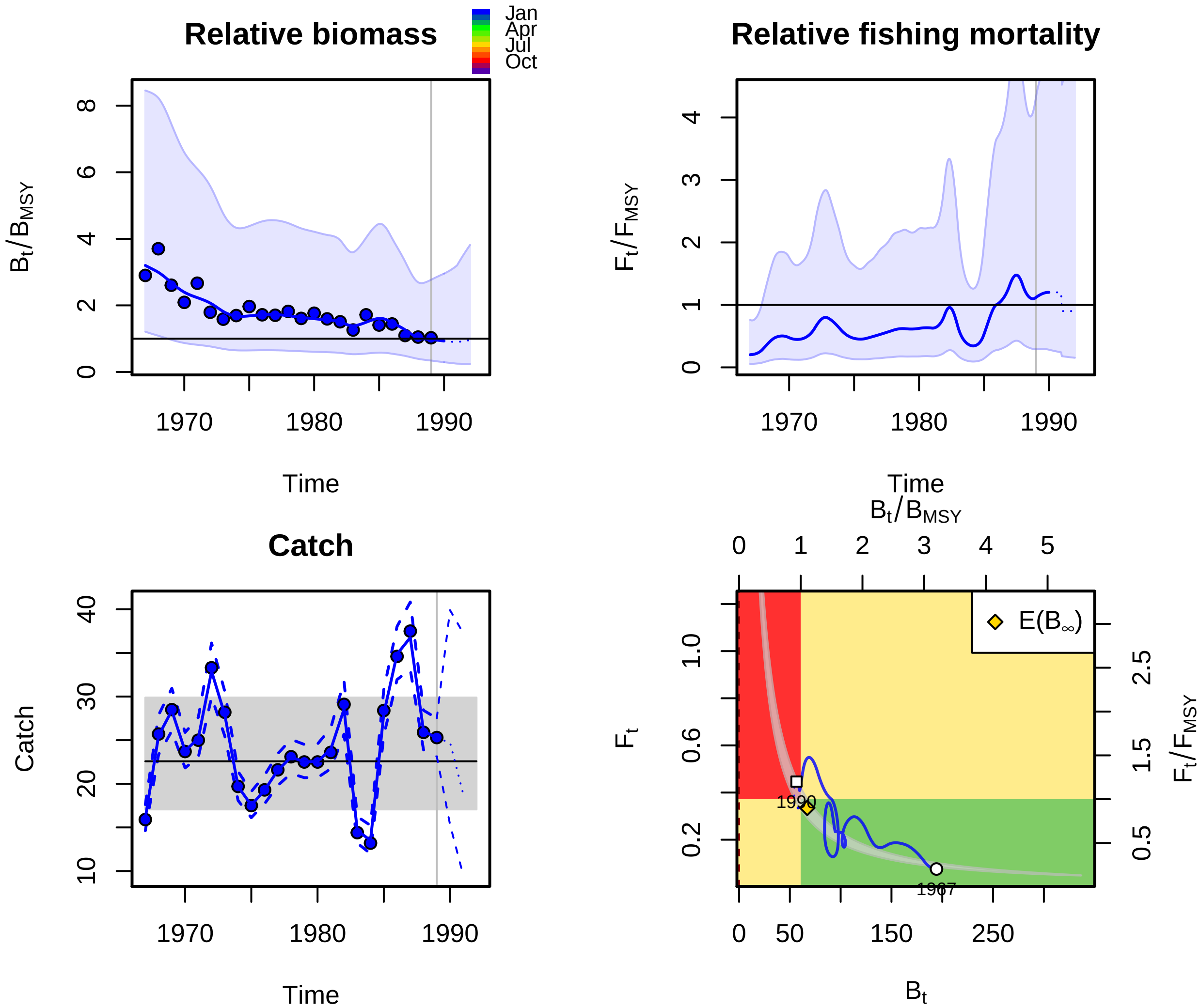
```r
inp$timepredc <- 1991
inp$dtpredc <- 1
inp$timepredi <- 1992
inp$ffac <- 0.75
res <- fit.spict(inp)
```

To specifically show forecast results use

```r
sumspict.predictions(res)
                  prediction       cilow      ciupp     log.est
   B_1992.00      58.0404387 28.74512738 117.191776  4.06113999
   F_1992.00       0.3348379  0.09426505   1.189374 -1.09410874
   B_1992.00/Bmsy  0.9556116  0.23937468   3.814912 -0.04540377
   F_1992.00/Fmsy  0.9006312  0.15380645   5.273748 -0.10465949
   Catch_1991.00  18.8028897  9.48829836  37.261545  2.93401056
   E(B_inf)       67.1854881          NA         NA  4.20745727
```

This output is also shown when using `summary(res)`. The results can be plotted using `plot(res)`, however to visualise the change in forecasted fishing mortality and associated change in forecasted catch more clearly we use

```r
par(mfrow=c(2, 2), mar=c(4, 4.5, 3, 3.5))
plotspict.bbmsy(res)
plotspict.ffmsy(res, qlegend=FALSE)
plotspict.catch(res, qlegend=FALSE)
plotspict.fb(res, man.legend=FALSE)
```



Note in the plot that the decrease in fishing pressure results in a constant biomass as opposed to the expected decrease if fishing effort had remained constant.

Management scenarios

The package has a function that runs several predefined management scenarios, which can be presented in a forecast table. To perform the calculations required to produce the forecast table run:

```
res <- manage(res)
```

where `res` is the result of `fit.spict()` from the code above. Then, the results can be summarised (and extracted) by running:

```
df <- mansummary(res)
   Observed interval, index:  1967.00 - 1989.00
   Observed interval, catch:  1967.00 - 1990.00

   Fishing mortality (F) prediction: 1992.00
   Biomass (B) prediction:           1992.00
   Catch (C) prediction interval:    1991.00 - 1992.00

   Predictions
```

```
                        C    B     F B/Bmsy F/Fmsy perc.dB perc.dF
1. Keep current catch 25.1 47.9 0.539  0.789  1.449   -11.7    20.7
2. Keep current F     23.9 52.9 0.446  0.870  1.201    -2.7     0.0
3. Fish at Fmsy       20.6 56.3 0.372  0.926  1.000     3.6   -16.7
4. No fishing          0.0 77.0 0.000  1.267  0.001    41.8   -99.9
5. Reduce F 25%       18.8 58.0 0.335  0.955  0.901     6.9   -25.0
6. Increase F 25%     28.5 48.1 0.558  0.793  1.501   -11.3    25.0
7. MSY advice rule    26.0 50.7 0.496  0.835  1.333    -6.6    11.0

95% CIs of absolute predictions
                      C.lo C.hi B.lo  B.hi  F.lo  F.hi
1. Keep current catch 23.0 27.5 23.0 100.0 0.239 1.216
2. Keep current F     12.5 45.6 24.0 116.4 0.126 1.586
3. Fish at Fmsy       10.5 40.2 27.1 116.9 0.105 1.321
4. No fishing          0.0  0.1 48.5 122.3 0.000 0.002
5. Reduce F 25%        9.5 37.3 28.7 117.2 0.094 1.189
6. Increase F 25%     15.5 52.5 20.0 115.9 0.157 1.982
7. MSY advice rule    13.9 48.8 22.1 116.2 0.140 1.761

95% CIs of relative predictions
                      B/Bmsy.lo B/Bmsy.hi F/Fmsy.lo F/Fmsy.hi
1. Keep current catch     0.234     2.660     0.350     6.002
2. Keep current F         0.211     3.595     0.205     7.032
3. Fish at Fmsy           0.230     3.737     0.171     5.856
4. No fishing             0.334     4.802     0.000     0.007
5. Reduce F 25%           0.239     3.815     0.154     5.274
6. Increase F 25%         0.184     3.413     0.256     8.790
7. MSY advice rule        0.199     3.510     0.228     7.807
```

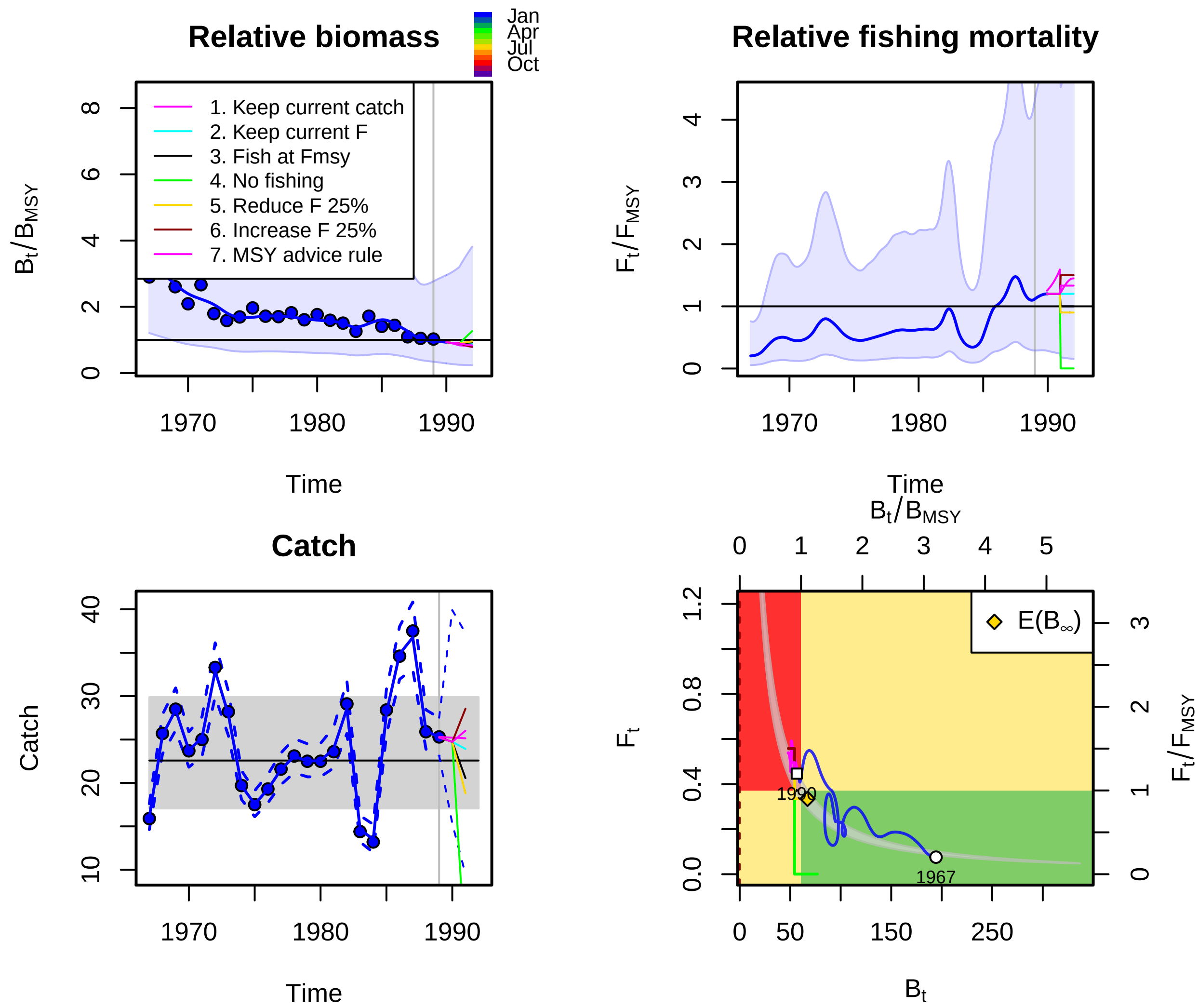
Then, `df` is a data frame with each line containing a line of the output

```
head(df)
```

```
                        C    B     F B/Bmsy F/Fmsy perc.dB perc.dF
1. Keep current catch 25.1 47.9 0.539  0.789  1.449   -11.7    20.7
2. Keep current F     23.9 52.9 0.446  0.870  1.201    -2.7     0.0
3. Fish at Fmsy       20.6 56.3 0.372  0.926  1.000     3.6   -16.7
4. No fishing          0.0 77.0 0.000  1.267  0.001    41.8   -99.9
5. Reduce F 25%       18.8 58.0 0.335  0.955  0.901     6.9   -25.0
6. Increase F 25%     28.5 48.1 0.558  0.793  1.501   -11.3    25.0
```

The resulting biomass, fishing mortality and catch of the management scenarios are included in the standard plots

```
par(mfrow=c(2, 2), mar=c(4, 4.5, 3, 3.5))
plotspict.bbmsy(res)
plotspict.ffmsy(res, qlegend=FALSE)
plotspict.catch(res, qlegend=FALSE)
plotspict.fb(res, man.legend=FALSE)
```



To obtain results for several forecast horizons without having to rerun the model each time it is required to set `inp$timepredc` equal to the longest horizon of interest. For example

```
inp <- pol$albacore
inp$timepredc <- 1991
res <- fit.spict(inp)
res <- manage(res)
```

Then the management table for 1990 is:

```
mansummary(res, ypred=1, include.unc = FALSE)
   Observed interval, index:  1967.00 - 1989.00
   Observed interval, catch:  1967.00 - 1990.00

   Fishing mortality (F) prediction: 1991.00
   Biomass (B) prediction:           1991.00
   Catch (C) prediction interval:    1990.00 - 1991.00

   Predictions
                          C    B     F B/Bmsy F/Fmsy perc.dB perc.dF
   1. Keep current catch 25.3 53.7 0.472  0.885  1.269    -4.9     5.7
   2. Keep current F     24.7 54.3 0.446  0.894  1.201    -3.9     0.0
```

```
   3. Fish at Fmsy        21.3 57.8 0.372  0.952  1.000     2.3   -16.7
   4. No fishing           0.0 79.1 0.000  1.303  0.001    40.0   -99.9
   5. Reduce F 25%        19.4 59.6 0.335  0.982  0.901     5.5   -25.0
   6. Increase F 25%      29.5 49.5 0.558  0.814  1.501   -12.5    25.0
   7. MSY advice rule     21.3 57.8 0.372  0.952  1.000     2.3   -16.7
```

and for 1991 is:

```
mansummary(res, ypred=2, include.unc = FALSE)
   Observed interval, index:  1967.00 - 1989.00
   Observed interval, catch:  1967.00 - 1990.00

   Fishing mortality (F) prediction: 1992.00
   Biomass (B) prediction:           1992.00
   Catch (C) prediction interval:    1991.00 - 1992.00

   Predictions
                           C     B     F B/Bmsy F/Fmsy perc.dB perc.dF
   1. Keep current catch 25.3  50.9 0.489  0.838  1.314   -10.0     9.5
   2. Keep current F     23.9  52.9 0.446  0.870  1.201    -6.5     0.0
   3. Fish at Fmsy       21.7  58.7 0.372  0.967  1.000     3.9   -16.7
   4. No fishing          0.0 100.4 0.000  1.652  0.001    77.6   -99.9
   5. Reduce F 25%       20.3  61.9 0.335  1.019  0.901     9.5   -25.0
   6. Increase F 25%     26.4  45.2 0.558  0.745  1.501   -20.0    25.0
   7. MSY advice rule    21.7  58.7 0.372  0.967  1.000     3.9   -16.7
```

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

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.