SPiCT

Martin W. Pedersen

07-11-2016

Contents

1 Basic functionality

1.1 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("mawp/spict/spict")
```

installs the stable version of `spict` and

```
devtools::install_github("mawp/spict/spict", ref = "dev")
```

installs the current development version that has new features, but it is not fully tested yet. When loading the package you are notified which version of the package you have installed:

```
library(spict)
#> Loading required package: TMB
#> Welcome to spict_v1.0@434a796de7d5dc406e8466d7456f340a8a32df80
```

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

1.2 Loading built-in example data

The package contains the catch and index data analysed in Polacheck, Hilborn, and Punt (1993). This data can be loaded by typing

```
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
#> [15] 23.6 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
#> [15] 1981 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
#> [12] 38.82 34.32 37.64 34.01 32.16 26.88 36.61 30.07 30.75 23.36 22.36
#> [23] 21.91
#>
#> $timeI
#>  [1] 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980
#> [15] 1981 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

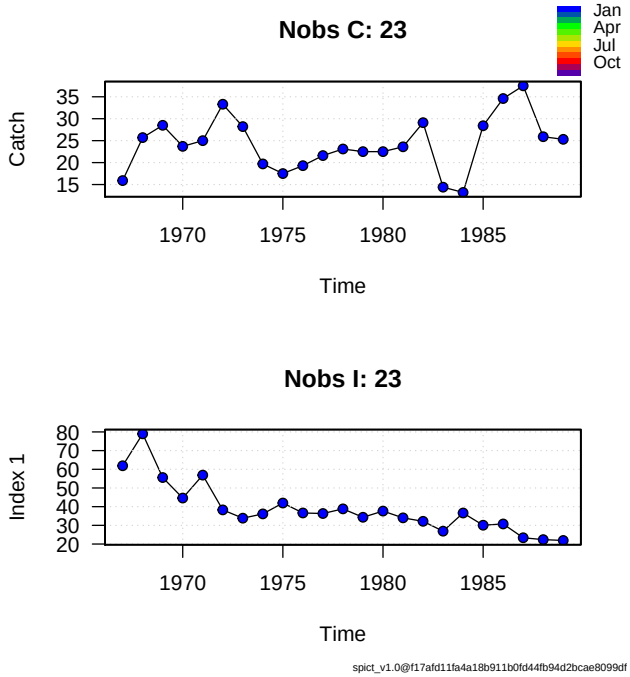
```
inp <- check.inp(pol$albacore)
inp$dtc
#>  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

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

1.3 Plotting data

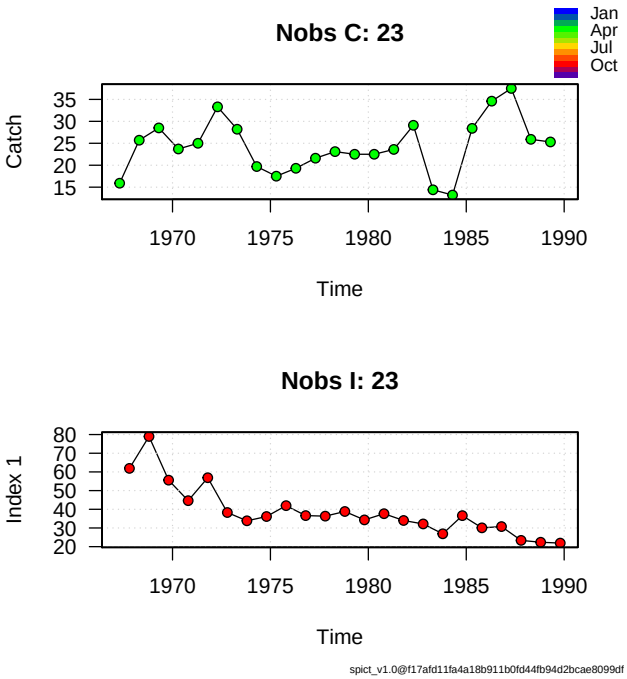
The data can be plotted using the command

```
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 left 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)
```

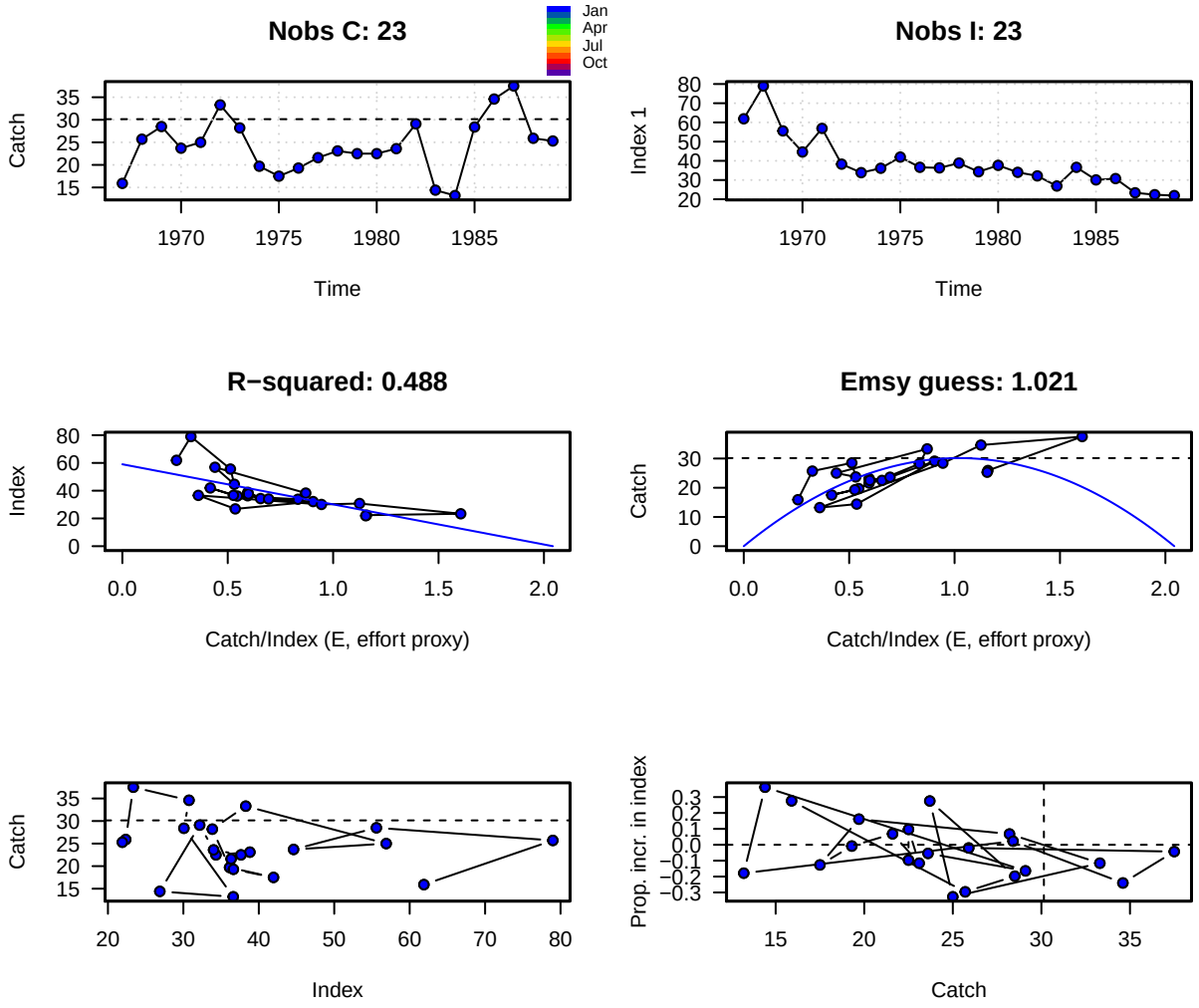


Now the colours show that catches are observed in the spring and index in the autumn.

1.4 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.

1.5 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"
```

```
#>  [4] "par.fixed"       "cov.fixed"       "pdHess"
#>  [7] "gradient.fixed"  "par.random"      "diag.cov.random"
#> [10] "env"             "inp"             "obj"
#> [13] "opt"             "pl"              "Cp"
#> [16] "report"          "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.

1.6 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  59.5934563  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 350.3057059 -0.2009019  "
#> [18] " m       22.5827819  17.0681833  29.8791050  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 239.490699  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 112.9279357  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   4.3199660  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 106.3183529  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  "
```

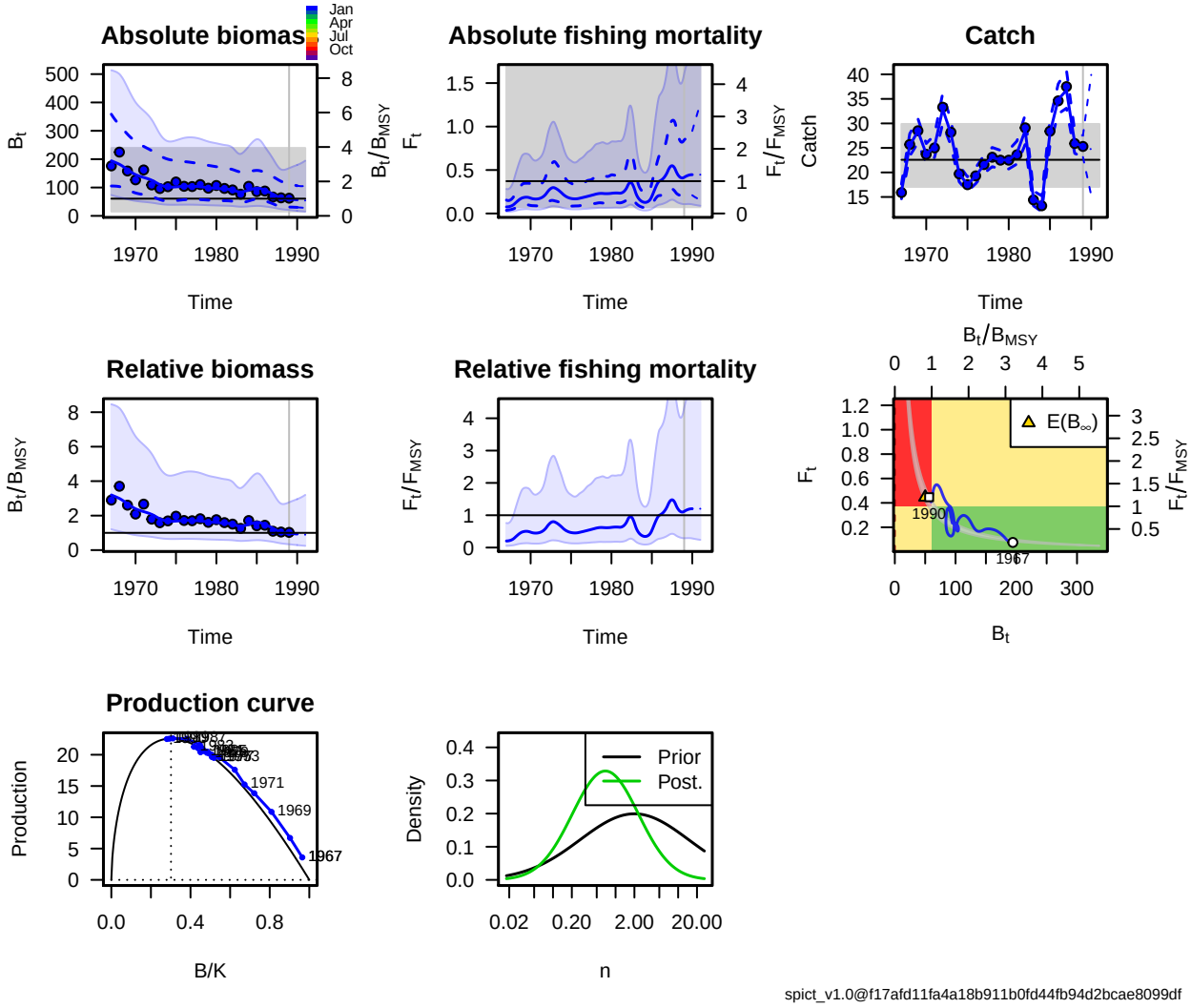
Here the `capture.output()` is only used to provide line numbers for easier reference, but the `summary()` command works without this.

- 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)`.

1.7 Interpreting plots of results

`spict` comes with several plotting abilities. The basic plottin 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 coloured 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	

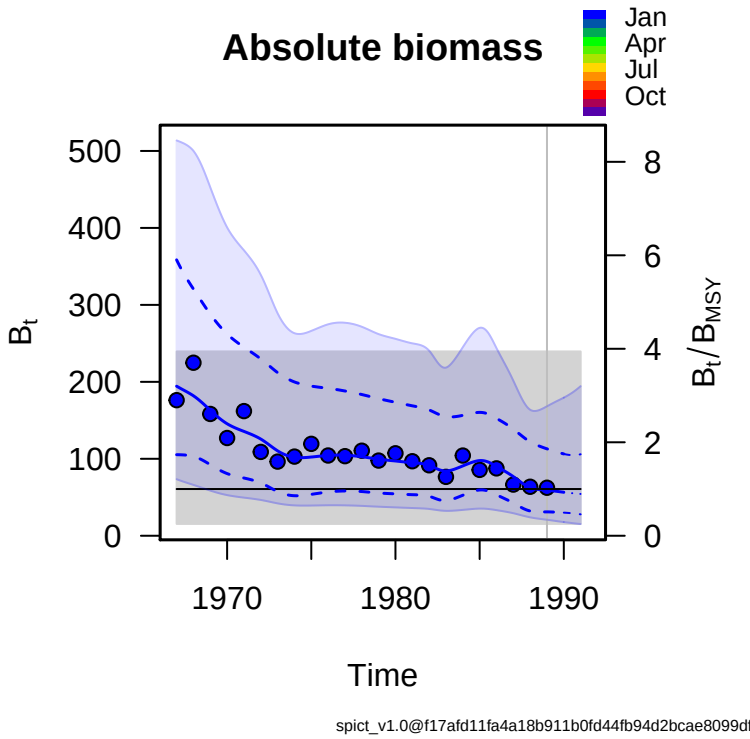
Function	Plot
`plotspict.ci`	Basic data plotting (see section 1.3)
`plotspict.data`	Advanced data plotting (see section 1.4)
Estimates	
`plotspict.bbmsy`	Relative biomass B/B_{MSY} estimates with uncertainty
`plotspict.biomass`	Absolute (and relative) biomass estimates with uncertainty
`plotspict.btrend`	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

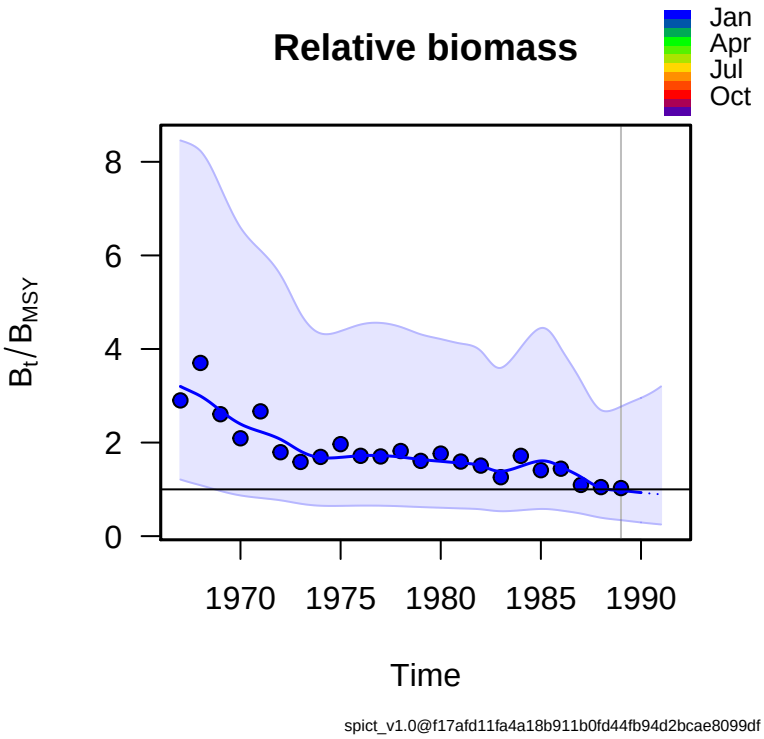
```
plotspict.biomass(res)
```



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

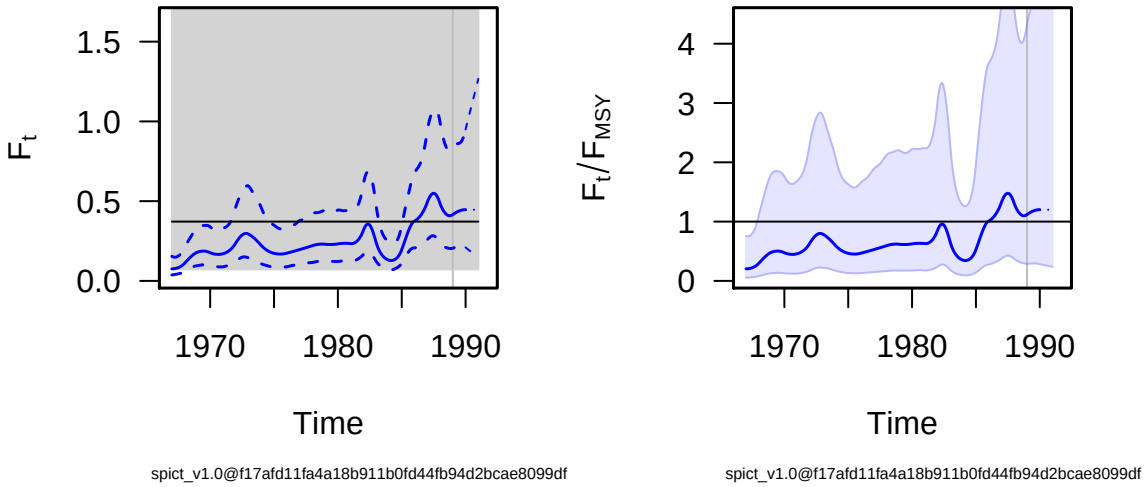
```
plotspict.bbmsy(res)
```



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

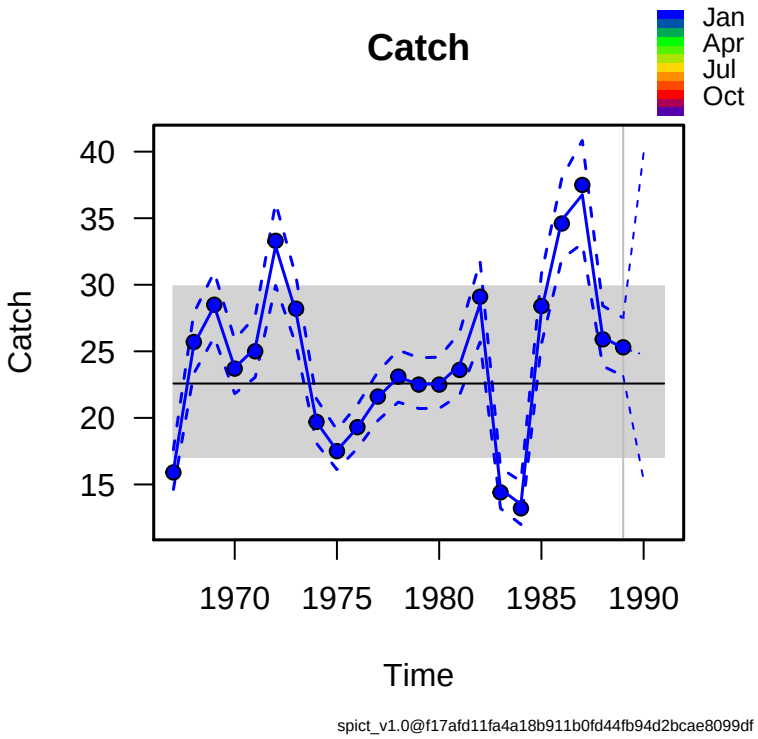
```
plotspict.f(res, main='', qlegend=FALSE, rel.axes=FALSE, rel.ci=FALSE)
plotspict.ffmsy(res, main='', qlegend=FALSE)
```



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)
```

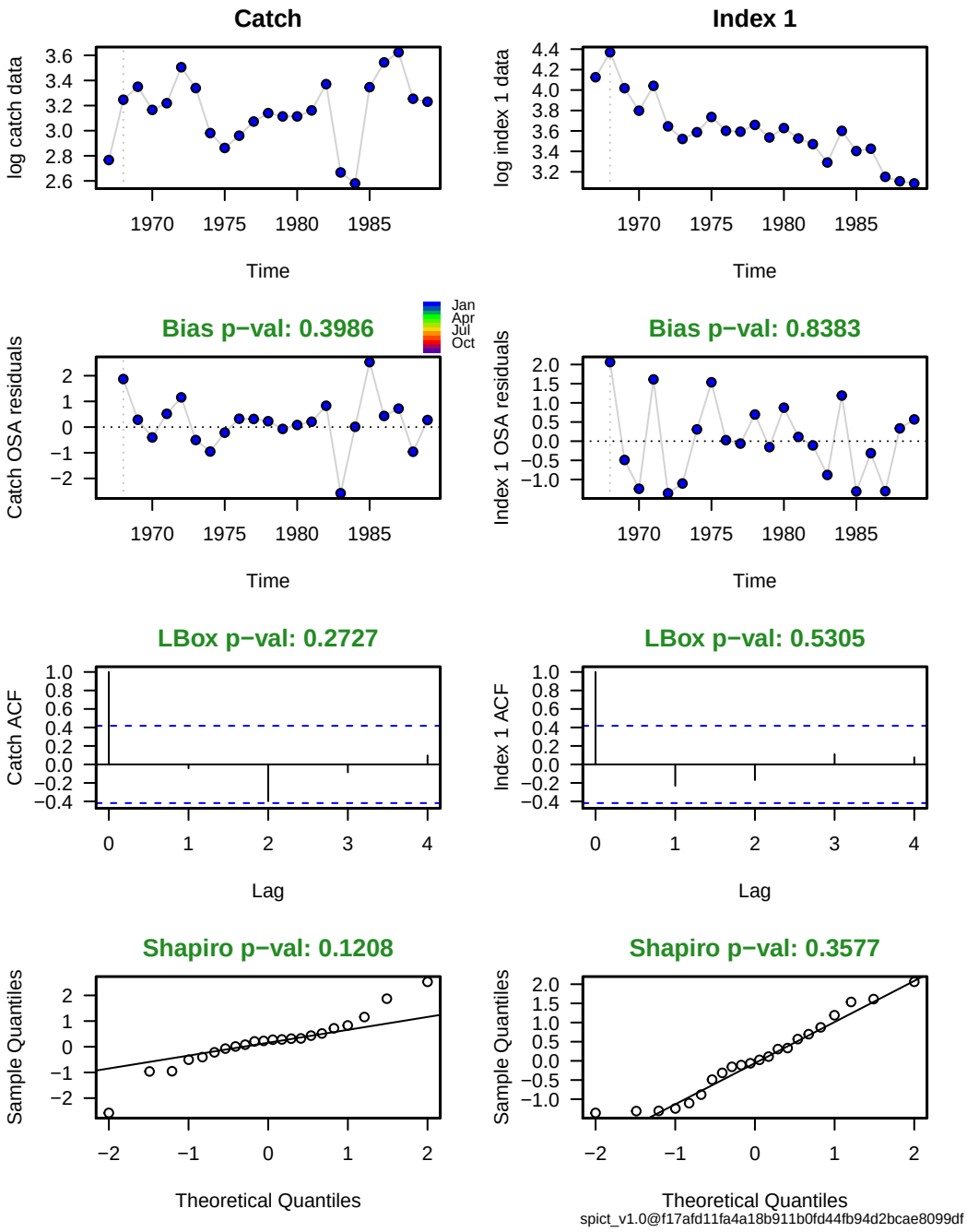


This plot shows estimated catches (blue line) versus observed catches (points) with the estimate of MSY plotted as a horizontal black line with its 95% CI given by the grey region.

1.8 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 (2016). 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 (2016).

1.9 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

```
sort(unique(c(names(res$value), names(res$par.fixed), names(res$par.random))))
#>  [1] "Bmsy"             "Bmsy2"            "Bmsyd"
#>  [4] "Bmsys"            "Cp"               "Emsy"
#>  [7] "Emsy2"            "Fmsy"             "Fmsyd"
#> [10] "Fmsys"            "gamma"            "isdb2"
#> [13] "isdc2"            "isde2"            "isdf2"
#> [16] "isdi2"            "K"                "logalpha"
#> [19] "logB"             "logBBmsy"         "logbeta"
#> [22] "logbkfrac"        "logBl"            "logBlBmsy"
#> [25] "logBlK"           "logBmsy"          "logBmsyd"
#> [28] "logBmsys"         "logBp"            "logBpBmsy"
#> [31] "logBpK"           "logCp"            "logCpred"
#> [34] "logEmsy"          "logEmsy2"         "logEp"
#> [37] "logF"             "logFFmsy"         "logFl"
#> [40] "logFlFmsy"        "logFmsy"          "logFmsyd"
#> [43] "logFmsys"         "logFp"            "logFpFmsy"
#> [46] "logFs"            "logIp"            "logIpred"
#> [49] "logK"             "logm"             "logMSY"
#> [52] "logMSYd"          "logMSYs"          "logn"
#> [55] "logq"             "logq2"            "logr"
#> [58] "logrc"            "logrold"          "logsdb"
#> [61] "logsdc"           "logsdf"           "logsdi"
#> [64] "m"                "MSY"              "MSYd"
#> [67] "MSYs"             "p"                "q"
#> [70] "r"                "rc"               "rold"
#> [73] "sdb"              "sdc"              "sde"
#> [76] "sdf"              "sdi"              "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 and it is preferred to transform the quantity from log as most quantities are estimated on the log scale.

1.9.1 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    0.0204040311 -0.005706714  0.026834550 -0.156740737  0.032240104
#> logK   -0.0057067142  0.037105155 -0.038074890 -0.011343168 -0.021784063
#> logq    0.0268345499 -0.038074890  0.091311212 -0.227633743  0.040958411
#> logn   -0.1567407368 -0.011343168 -0.227633743  1.473743323 -0.190011612
#> logsdb  0.0322401039 -0.021784063  0.040958411 -0.190011612  0.980089751
#> logsdf  0.0014253353 -0.002407435  0.003270276 -0.006648026 -0.002144227
#> logsdi -0.0007603784 -0.002521057  0.002848540  0.007158237  0.010535580
#> logsdc  0.0015535777  0.001300308 -0.008392544  0.001997814  0.137920100
#>               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   -0.0066480259  0.0071582370  0.001997814
#> logsdb -0.0021442267  0.0105355798  0.137920100
#> logsdf  0.0262881975 -0.0002784384 -0.035159455
#> logsdi -0.0002784384  0.0237200085  0.005885858
#> logsdc -0.0351594554  0.0058858584  0.846803904
```

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 -0.207401077  0.62169056 -0.903884565  0.22798460
#> logK   -0.20740108  1.000000000 -0.65412309 -0.048507213 -0.11423228
#> logq    0.62169056 -0.654123090  1.00000000 -0.620531253  0.13691408
#> logn   -0.90388456 -0.048507213 -0.62053125  1.000000000 -0.15810161
#> logsdb  0.22798460 -0.114232279  0.13691408 -0.158101610  1.00000000
#> logsdf  0.06154300 -0.077082727  0.06674857 -0.033775468 -0.01335849
#> logsdi -0.03456323 -0.084978313  0.06120724  0.038285800  0.06909842
#> logsdc  0.01181908  0.007335635 -0.03018145  0.001788351  0.15139210
#>              logsdf       logsdi       logsdc
#> logm    0.06154300 -0.03456323  0.011819077
#> logK   -0.07708273 -0.08497831  0.007335635
#> logq    0.06674857  0.06120724 -0.030181454
#> logn   -0.03377547  0.03828580  0.001788351
#> logsdb -0.01335849  0.06909842  0.151392101
#> logsdf  1.00000000 -0.01115042 -0.235651555
#> logsdi -0.01115042  1.00000000  0.041529907
#> logsdc -0.23565155  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
```

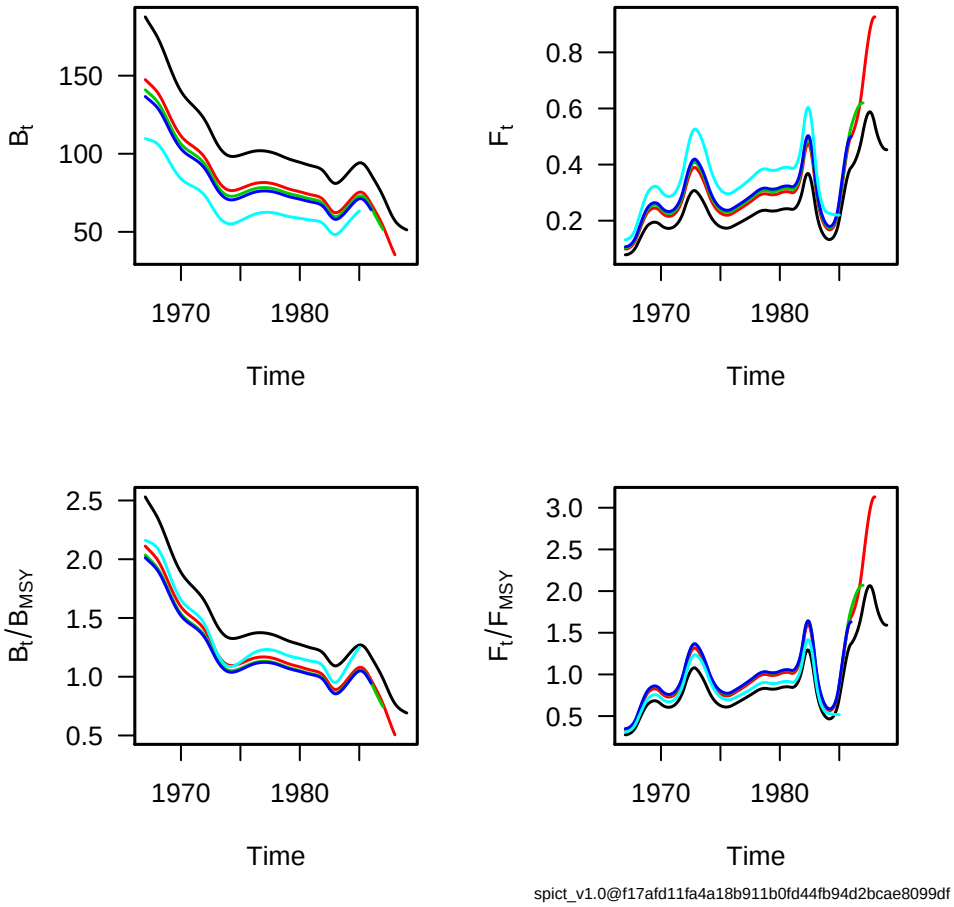
This reveals that for this data set the estimates of log Fmsy and log Bmsy are highly correlated. This is often the case and the reason why the model is reparameterised.

2 Advanced functionality

2.1 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

```
rep <- fit.spict(pol$albacore)
rep <- retro(rep)
plotspict.retro(rep)
```

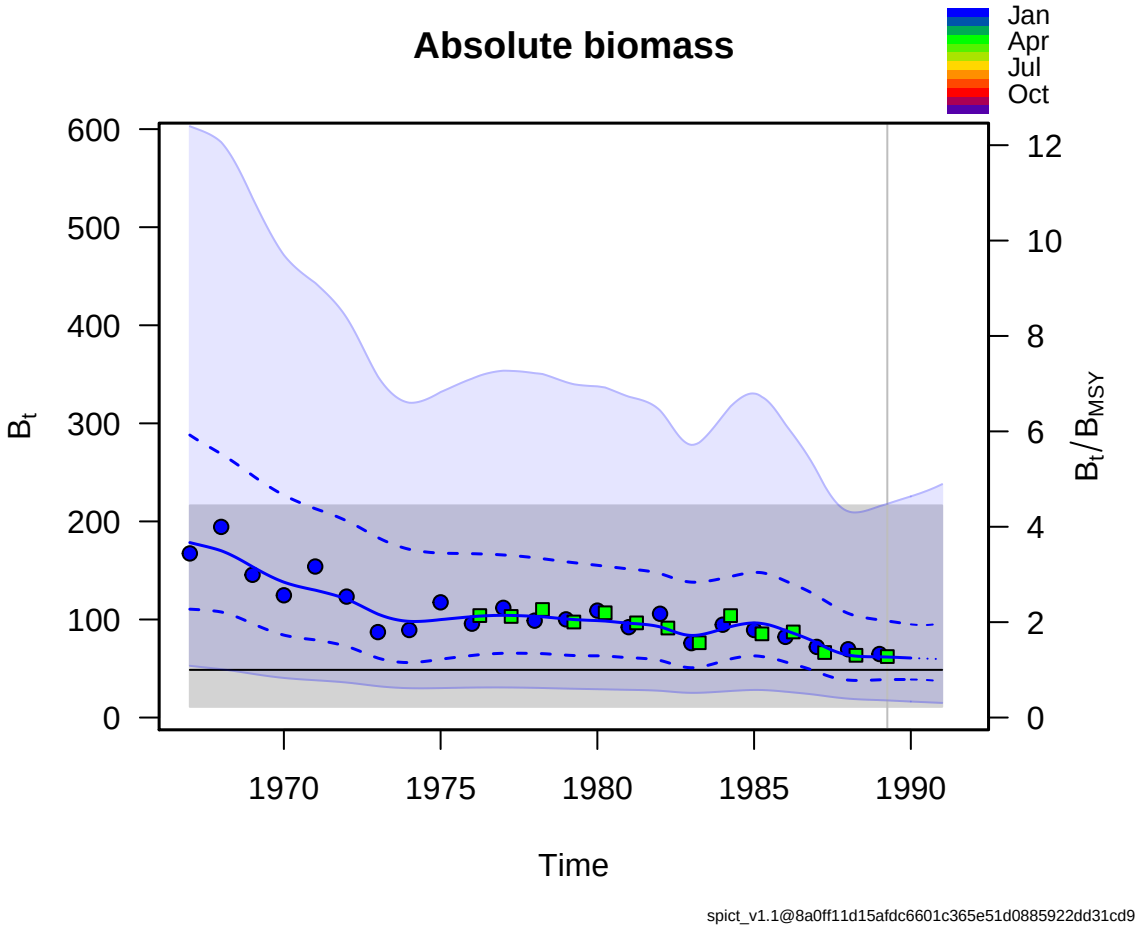


2.2 Estimation using two or more biomass indices

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

```
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   4.42580480   0.81555879  24.0175797  1.48745214
#> alpha2   3.01233058   0.51739118  17.5382493  1.10271406
#> beta     0.11812488   0.01931969   0.7222416 -2.13601289
#> r        0.22879005   0.09331027   0.5609767 -1.47495049
#> rc       1.00284912   0.16532792   6.0831006  0.00284507
#> rold     0.42078662   0.02718093   6.5141761 -0.86562942
#> m       24.39400768  17.53452170  33.9369172  3.19433752
#> K      205.97422592 131.81456247 321.8565608  5.32775104
#> q1       0.34814129   0.21582491   0.5615773 -1.05514687
```

```
#> q2       3.52337364   2.21557593   5.6031309  1.25941895
#> n        0.45628011   0.04216818   4.9371711 -0.78464839
#> sdb      0.02293207   0.00462212   0.1137745 -3.77521902
#> sdf      0.37021222   0.27062280   0.5064506 -0.99367888
#> sdi1     0.10149285   0.07435185   0.1385413 -2.28776688
#> sdi2     0.06907897   0.04505067   0.1059231 -2.67250496
#> sdc      0.04373127   0.00768991   0.2486928 -3.12969177
plotspict.biomass(res)
```



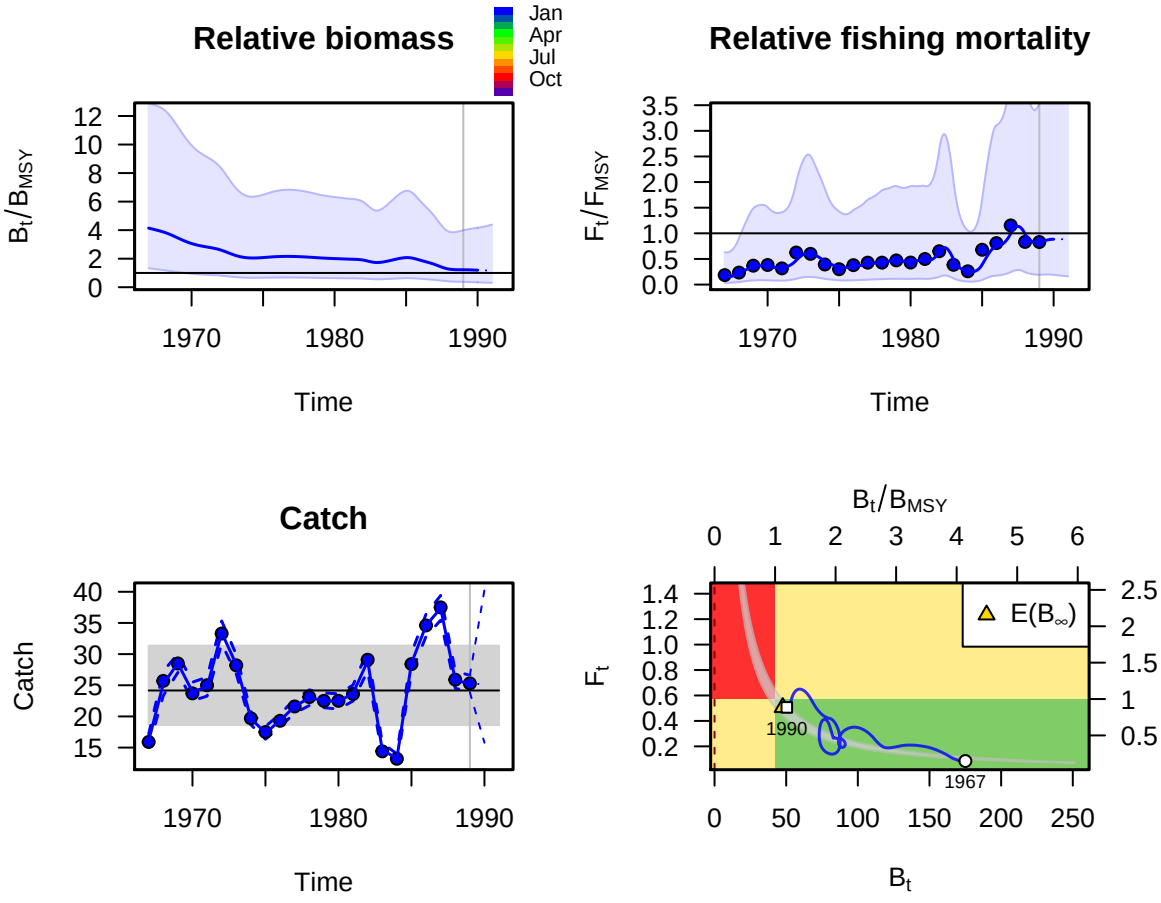
The model estimates seperate observation noises and finds that the first index (`sdi1`) is more noisy than the second (`sdi2`). It is furthermore estimated that the catchabilities are different by a factor 10 (`q1` versus `q2`). The biomass plot shows both indices with circles indicating the first index and squares indicating the second index (the two series can also be distringuished by their colours).

2.3 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   0.37238017 -2.6056723
#> r        0.23822939   0.10656855   0.53255150 -1.4345212
#> rc       1.14399163   0.21452271   6.10059821  0.1345236
#> rold     0.40826819   0.03851360   4.32789743 -0.8958310
#> m       24.16376130  18.62117905  31.35608969  3.1848540
#> K      189.53177410 132.49527136 271.12132399  5.2445567
#> qf       0.41117179   0.25562626   0.66136492 -0.8887442
#> n        0.41648800   0.04180192   4.14962359 -0.8758976
#> sdb      0.01430671   0.00236752   0.08645421 -4.2470266
#> sdf      0.37625014   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)
```



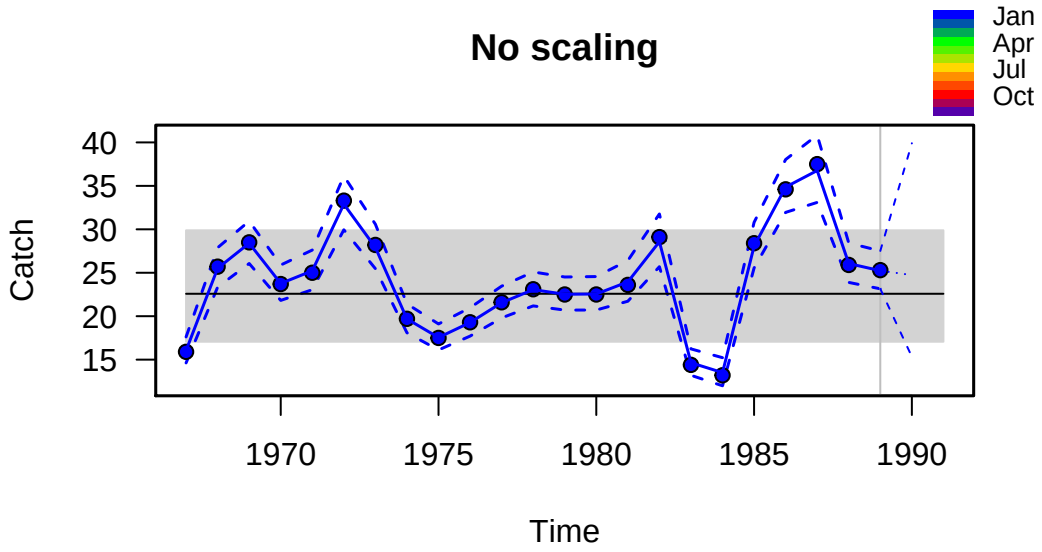
Here the model runs without and index of biomass and instead uses effort as an index of fishing mortality Note that index observations are missing from the biomass plot, but effort observations are present in the plot of fishing mortality. Note also that `q` is missing from the summary of parameter estimates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

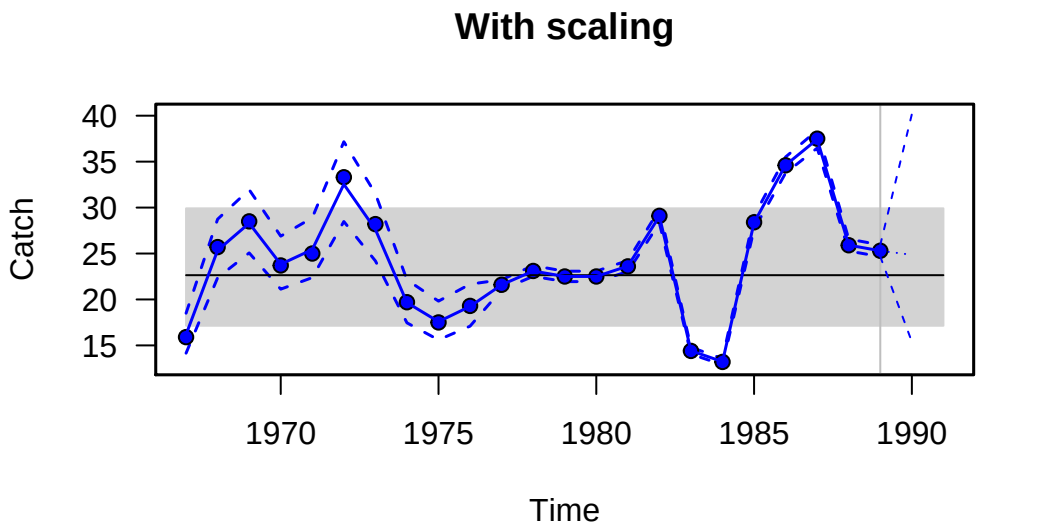
Overall for this data set the results in terms of stock status etc. do not change much, and this will probably often be the case, however using effort data directly instead of commercial CPUE is cleaner and avoids inputting the same data twice.

2.4 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.

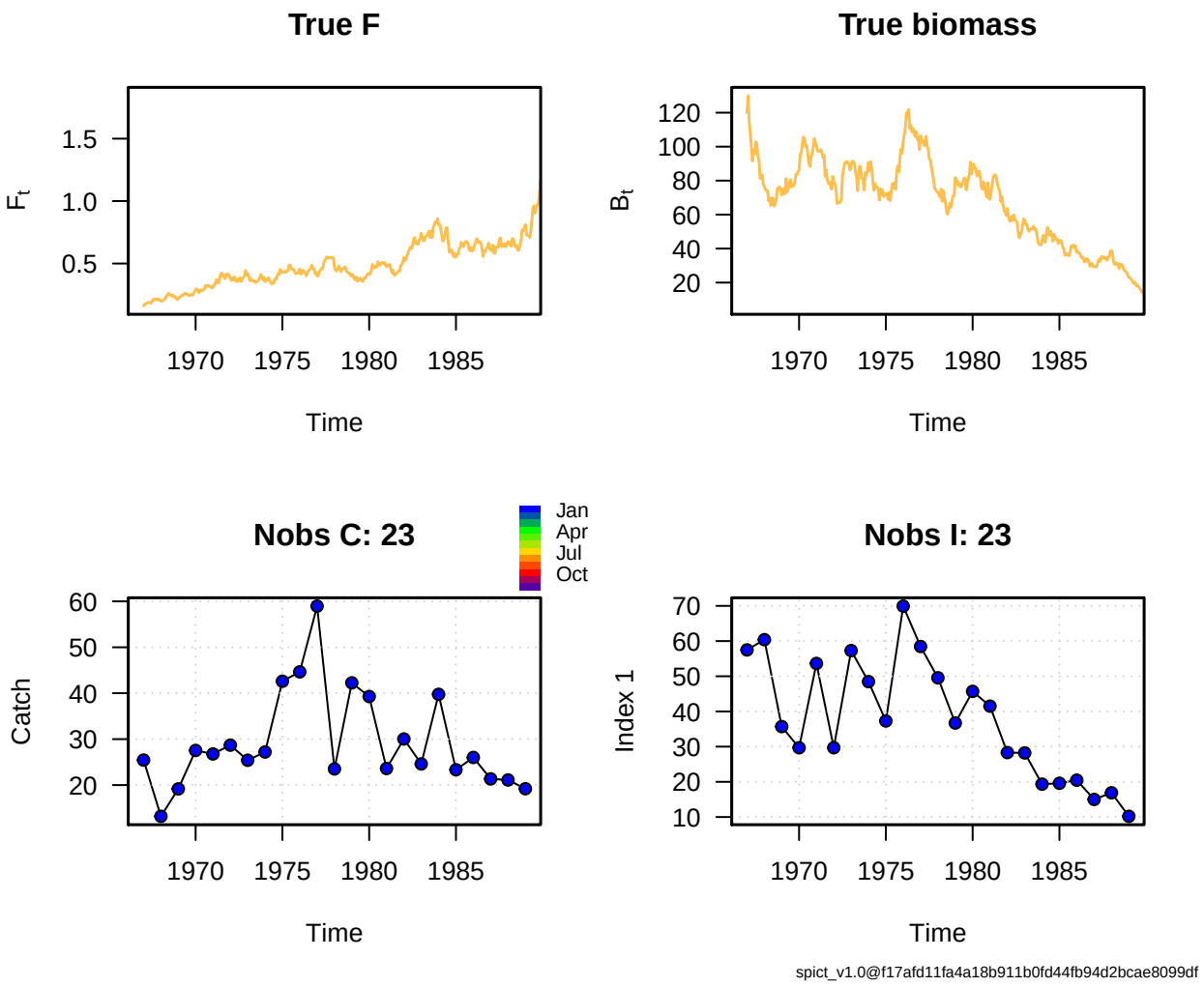
2.5 Simulating data

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

2.5.1 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

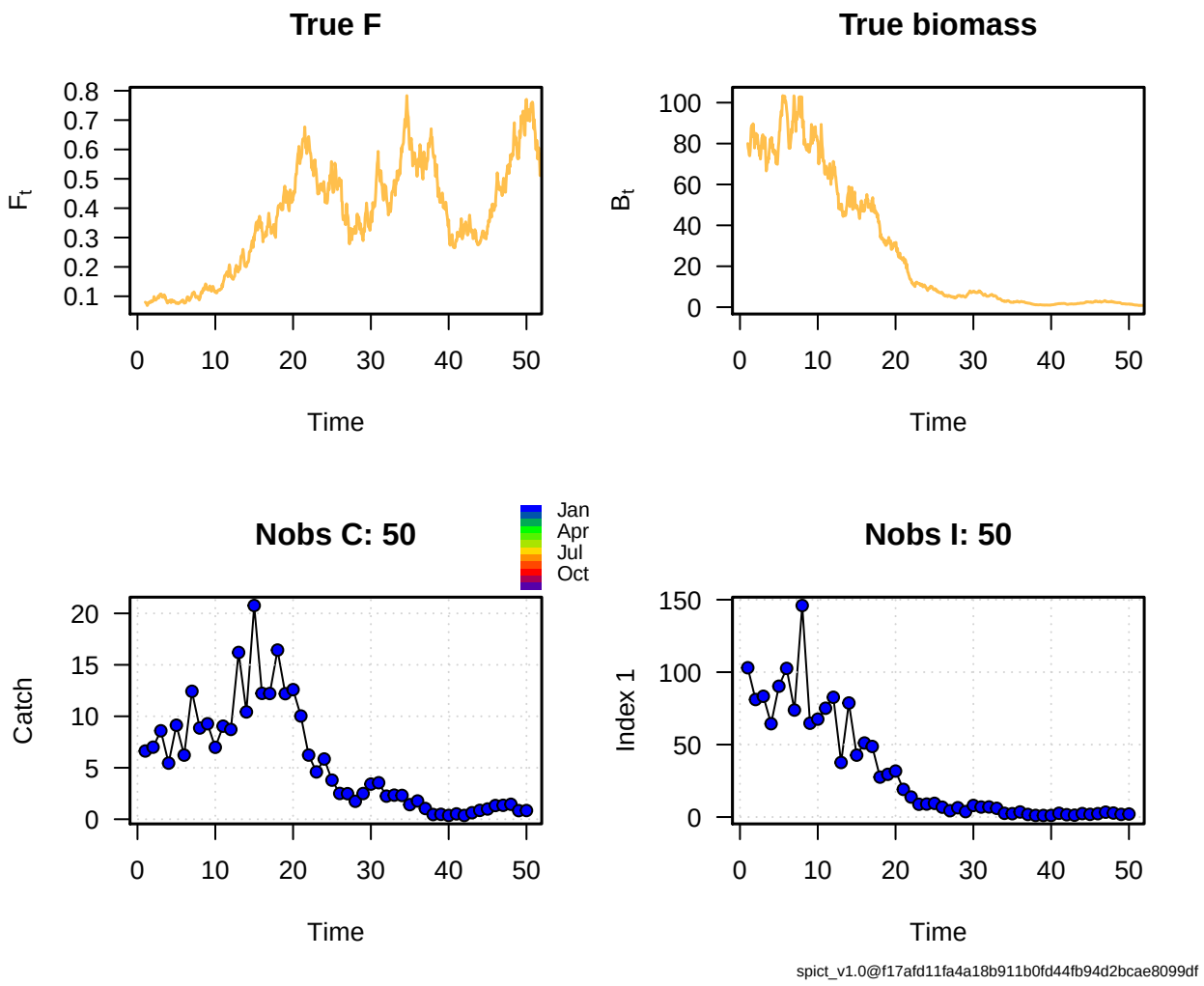
```
inp <- check.inp(pol$albacore)
sim <- sim.spict(inp)
plotspict.data(sim)
```



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)
```

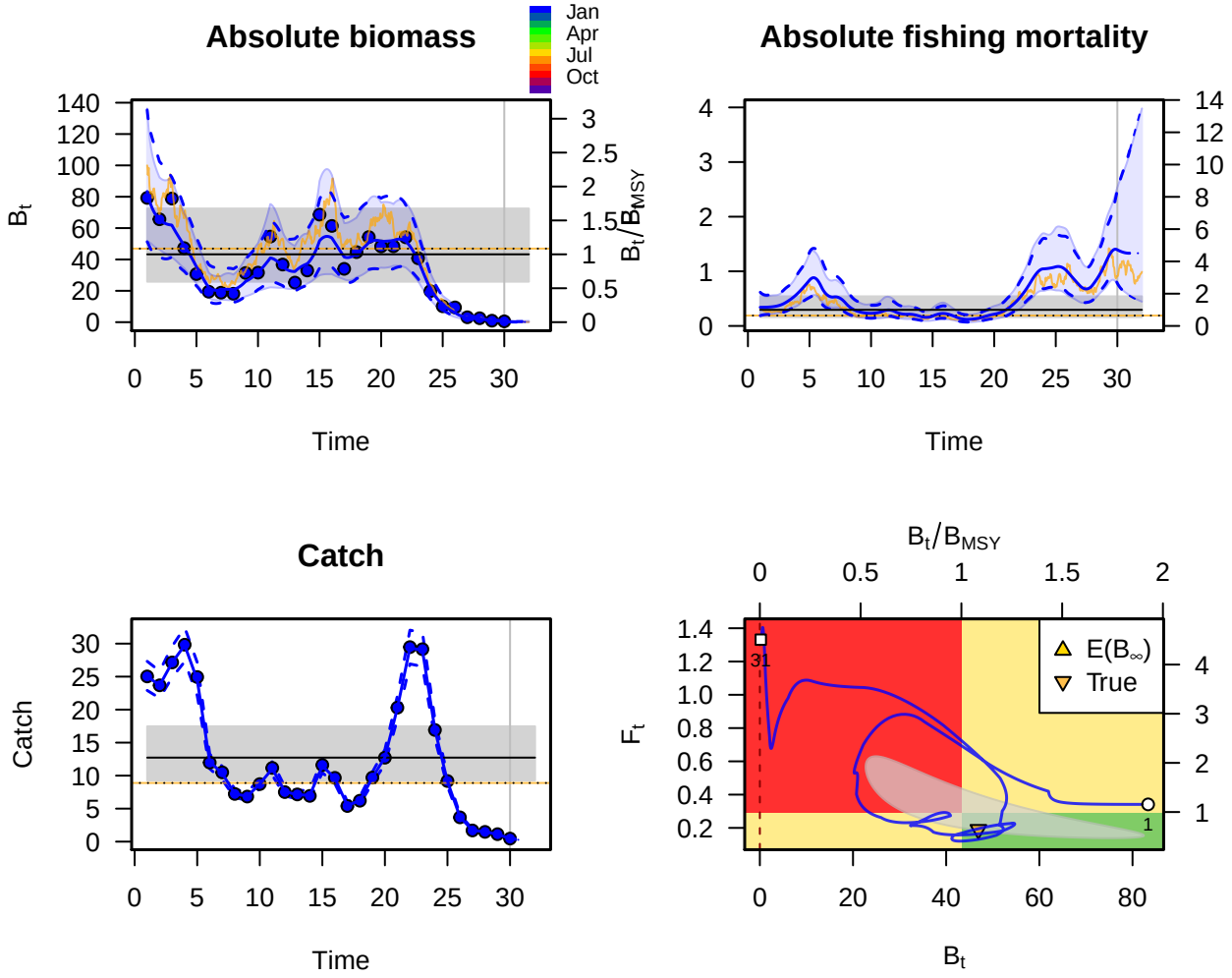


Here the required parameters are specified (the rest use default values), and the number of observations is specified as an argument to `sim.spict()`.

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  0.04504708
#> beta    0.13191757  -9.0  0.02269878   0.7666599         -9 -2.02557800
#> 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      76.02606884 100.0 46.11318791 125.3429530          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`.

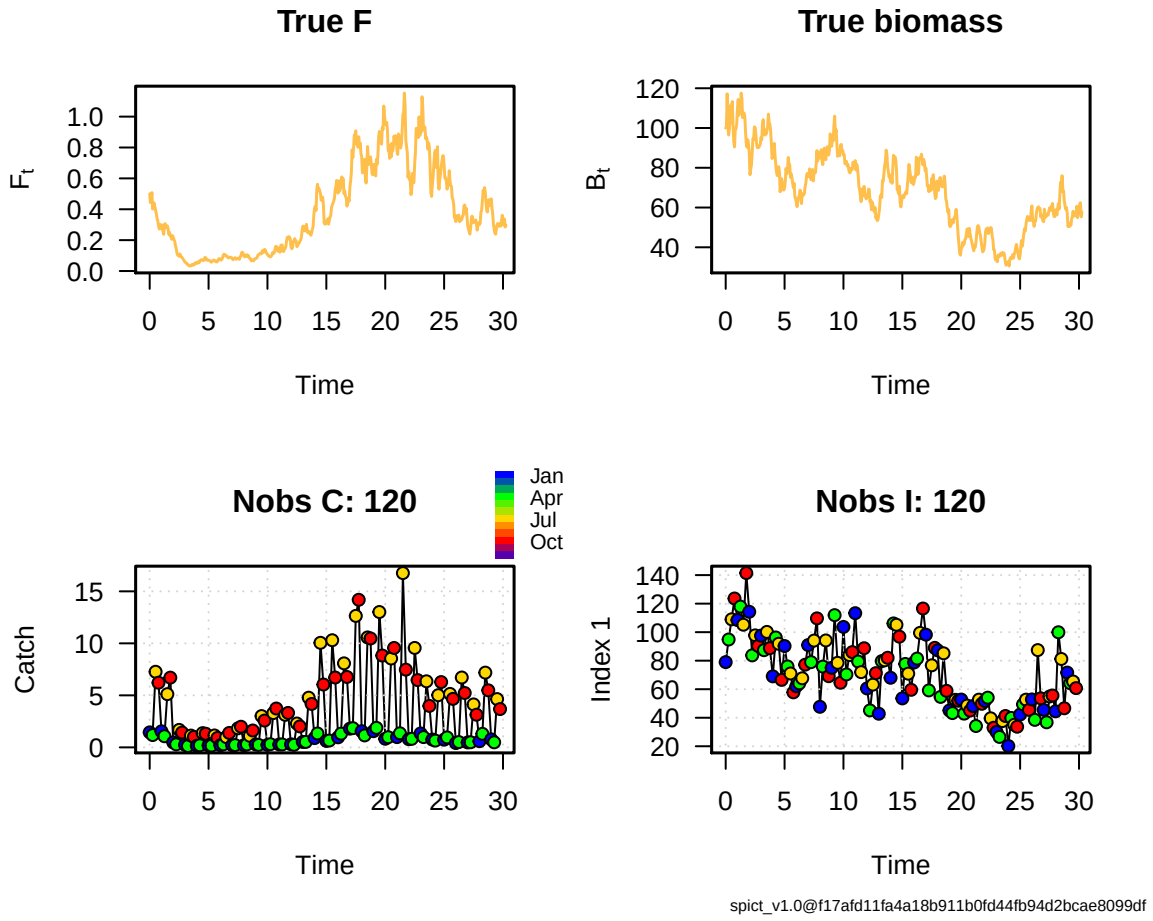
When printing the summary of the parameter estimates the true values are included as well as a check whether the true value was inside the 95% CIs. Similarly, the true biomass, fishing mortality, and reference points are included in the results plot using a yellow/orange colour.

2.5.2 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 need not 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

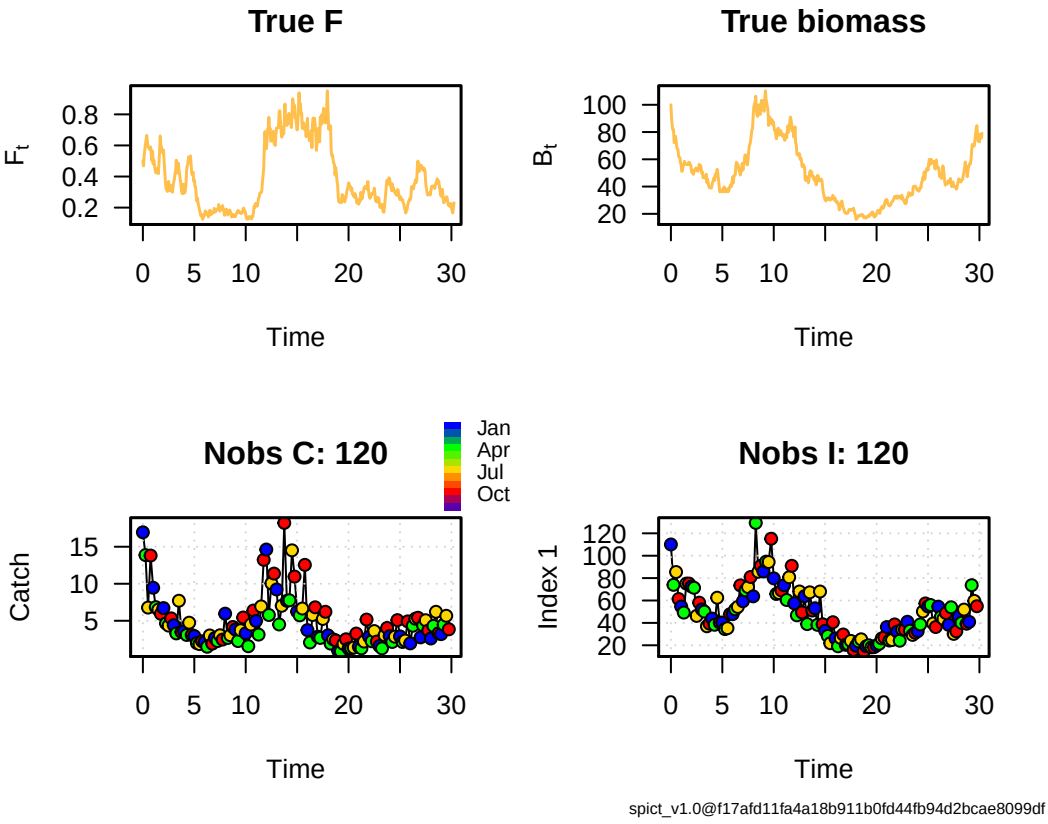
```
set.seed(1234)
inp <- list(nseasons=4, splineorder=3)
inp$timeC <- seq(0, 30-1/inp$nseasons, by=1/inp$nseasons)
inp$timeI <- seq(0, 30-1/inp$nseasons, by=1/inp$nseasons)
inp$ini <- list(logK=log(100), logm=log(20), logq=log(1),
                logbkfrac=log(1), logsdf=log(0.4), logF0=log(0.5),
                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)
```



2.6 Estimation using quarterly data

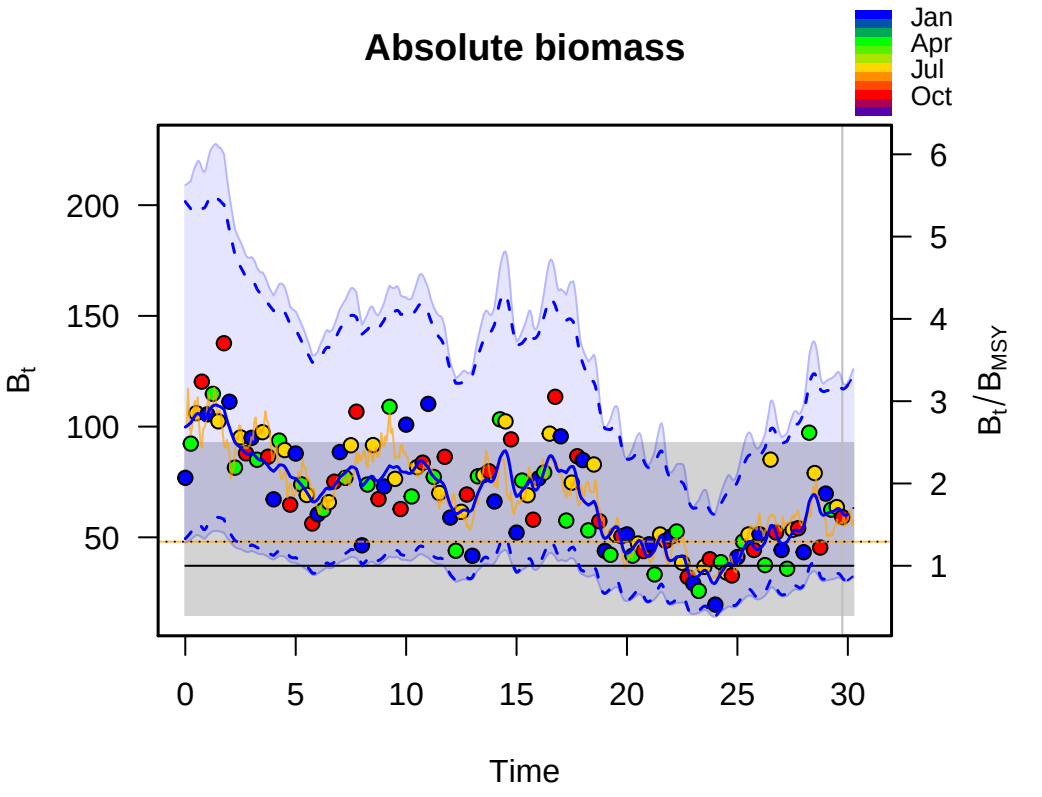
Catch information available in sub-annual aggregations, e.g. quarterly catch, can be used to estimate the seasonal pattern of the fishing mortality. The user can choose between two types of seasonality by setting `seasontype` to 1 or 2:

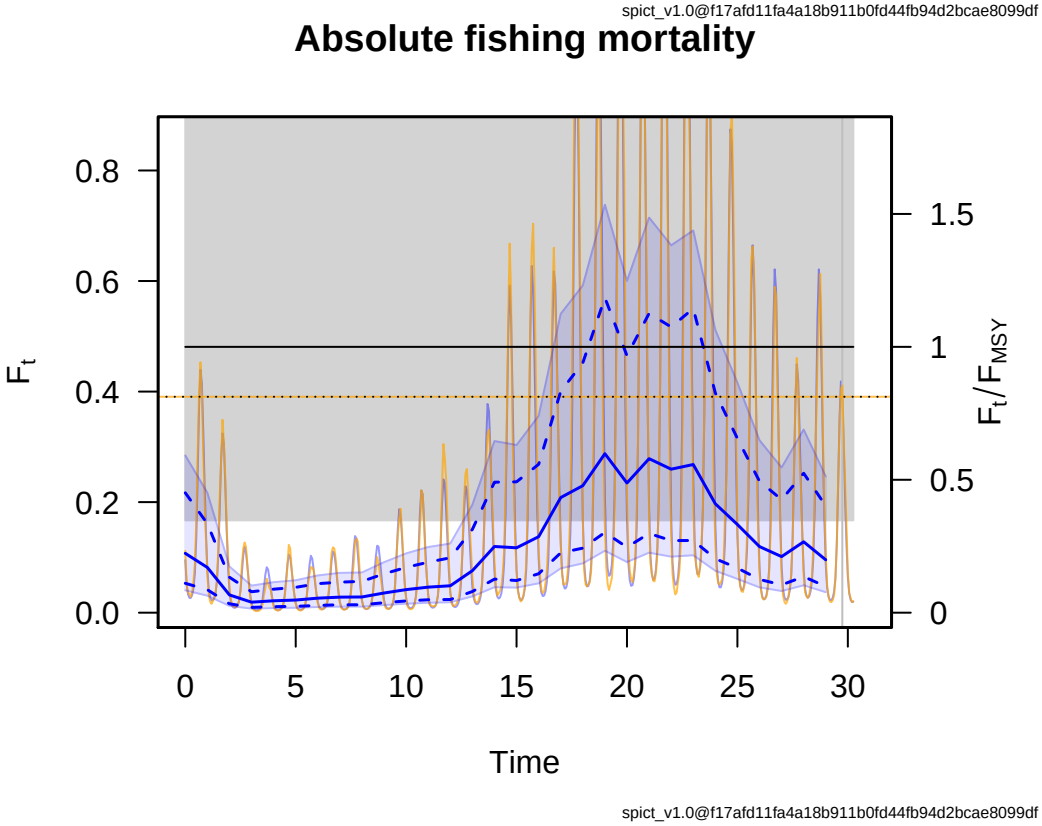
1. using cyclic B-splines.
2. using coupled stochastic differential equations (SDEs).

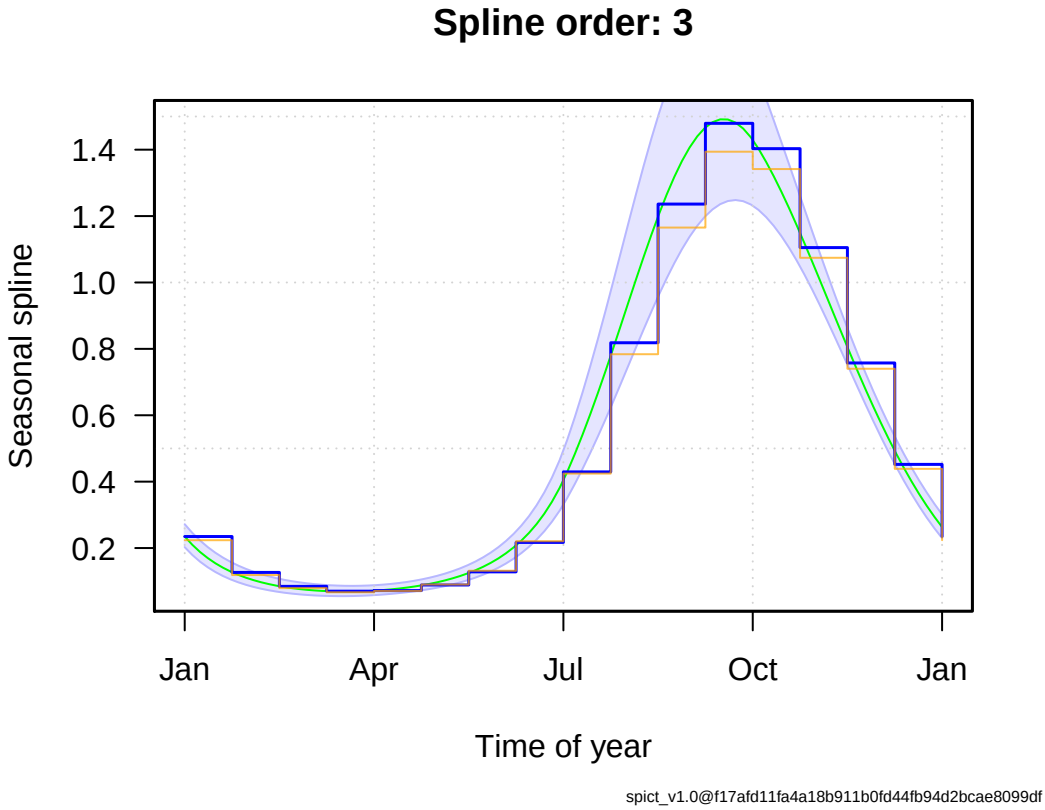
Technical description of the season types is found in Pedersen and Berg (2016).

Here is shown an example of a spline-based model fitted to quarterly data simulated in section 2.5.2

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


Absolute biomass
Jan
Apr
Jul
Oct
200
150
100
50
B_t
6
5
4
3
2
1
B_t/B_{MSY}
0
5
10
15
20
25
30
Time
spict_v1.0@f17afd11fa4a18b911b0fd44fb94d2bcae8099df

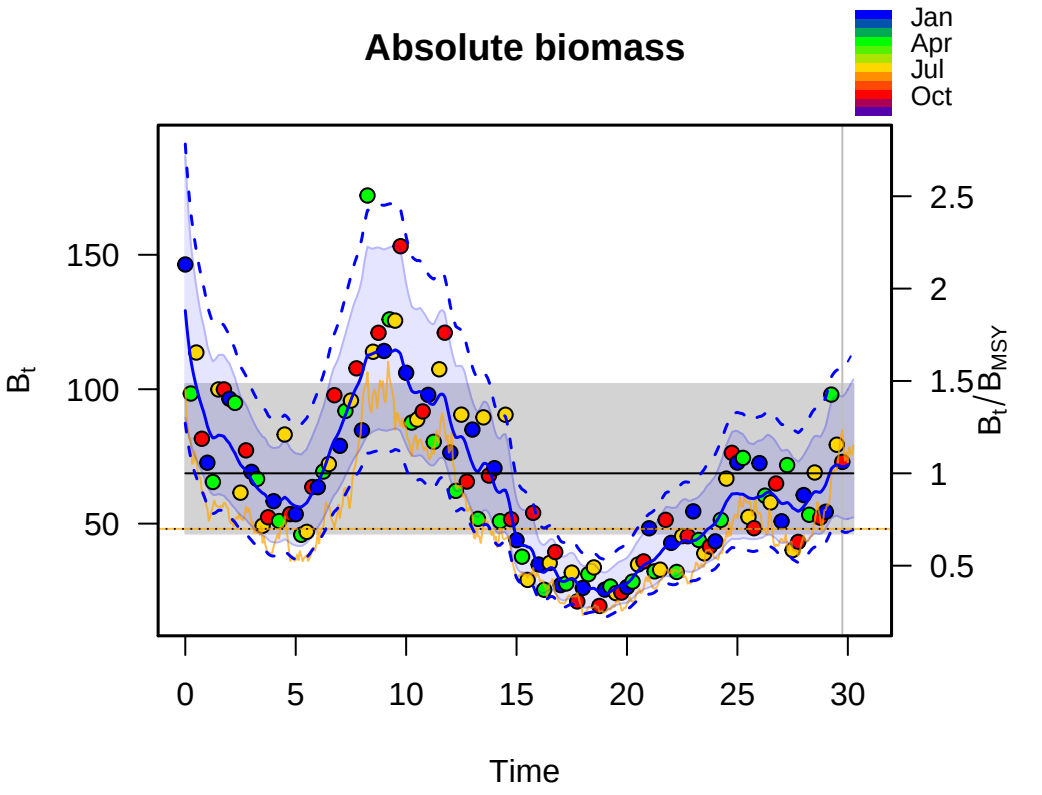

Absolute fishing mortality
0.8
0.6
0.4
0.2
0.0
F_t
1.5
1
0.5
0
F_t/F_{MSY}
0
5
10
15
20
25
30
Time
spict_v1.0@f17afd11fa4a18b911b0fd44fb94d2bcae8099df

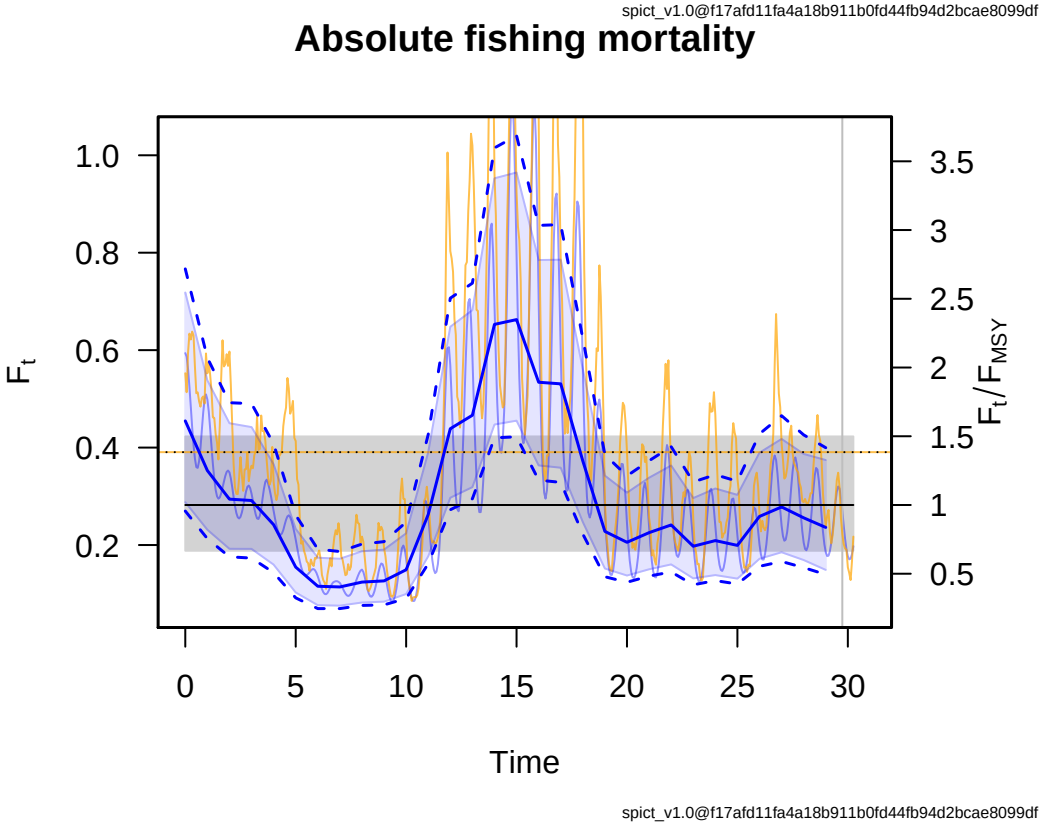


In the results plots, the model is able to estimate the seasonal variation in fishing mortality as seen both in the plot of `F` and in the plot of the estimated spline, where blue is the estimated spline, orange is the true spline, and green is the spline if time were truly continuous (it is discretised with the Euler steps show 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    1.38654513  -9.0  0.74646650   2.5754772          -9  0.3268151
#> beta     0.69812209  -9.0  0.46862508   1.0400093          -9 -0.3593613
#> r        0.75461230  -9.0  0.19653877   2.8973404          -9 -0.2815512
#> rc       0.57786149  -9.0  0.38165597   0.8749343          -9 -0.5484211
#> rold     0.46819698  -9.0  0.22851759   0.9592627          -9 -0.7588662
#> m       20.30400509  20.0 15.39627327  26.7761305           1  3.0108182
#> K      127.48850152 100.0 77.62144027 209.3921211           1  4.8480262
#> q        0.75175503   1.0  0.51602963   1.0951612           1 -0.2853448
#> n        2.61174109   2.0  0.81672644   8.3518681           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)
```

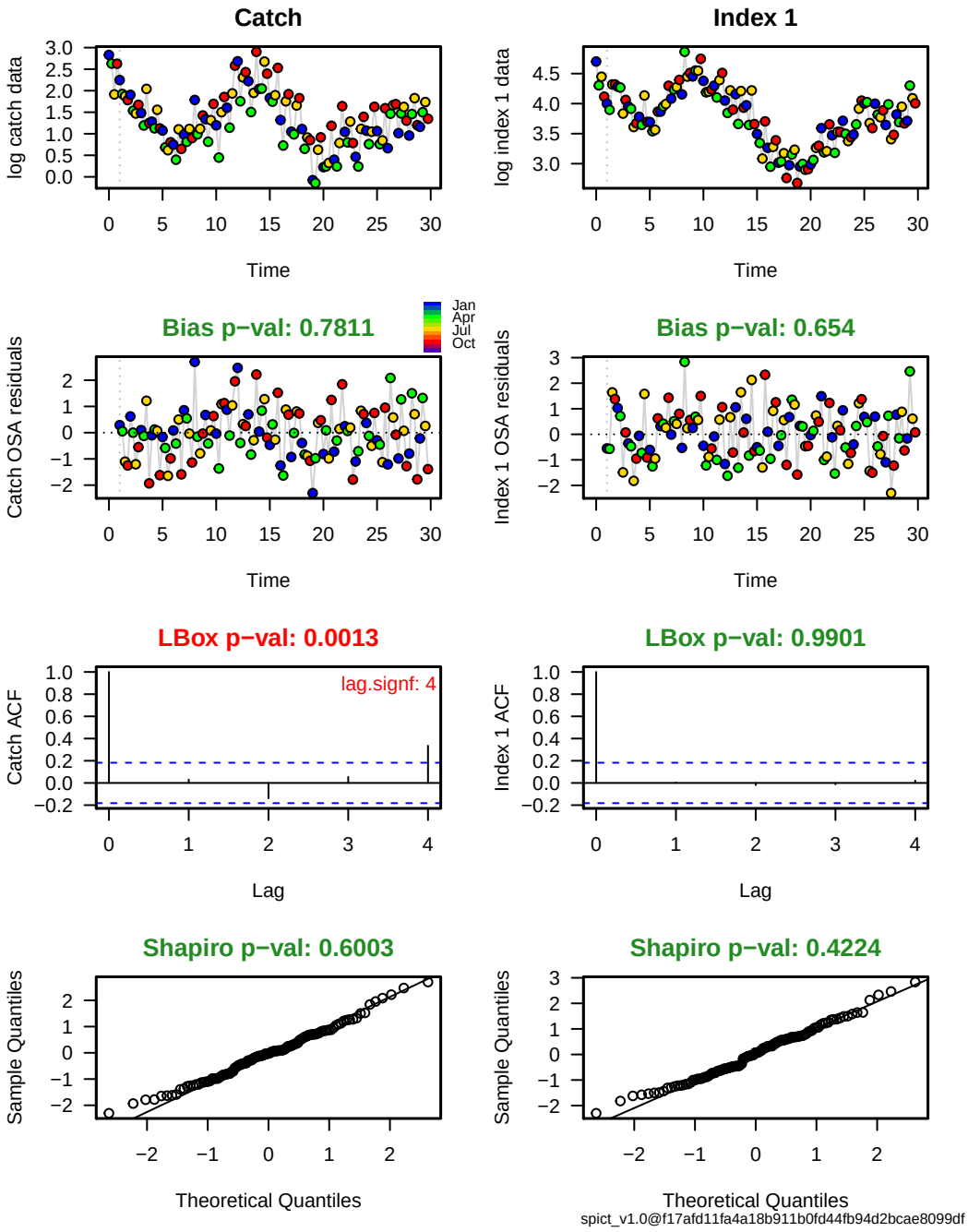




Two parameters related to the coupled SDEs are estimated (`sdu` and `lambda`) as evident from the summary of estimated parameters. In the plot of fishing mortality it is noted that the amplitude of the seasonal pattern

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 if in reality the seasonal pattern shifts a bit, will lead to biased estimates and autocorrelation in residuals. 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.

2.7 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`.

2.7.1 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 ... convergence not obtained!
#>  Trial 4 ... model fitted!
#> $propchng
#>          logn  logK  logm  logq logqf logsdb logsdf logsdi logsdc logsde
#> Trial 1 -1.41  0.26 -0.12 -2.75 -9.29   1.30  -0.08  -1.12  -0.15   0.12
#> Trial 2  3.04 -0.04  0.24 -0.52  8.37  -1.14   0.73   1.31   0.49  -1.30
#> Trial 3  2.59  0.18  0.19 -3.55 -3.28  -0.60  -0.13  -0.27   0.60   1.01
#> Trial 4  3.08  0.37  0.26 -2.12 10.02   0.06  -0.74   0.81   0.52   0.77
#>
#> $inimat
#>         Distance  logn logK logm  logq logqf logsdb logsdf logsdi logsdc
#> Basevec     0.00  0.69 5.01 3.41 -0.64 -0.22  -1.61  -1.61  -1.61  -1.61
#> Trial 1     4.23 -0.29 6.34 2.99  1.12  1.81  -3.70  -1.48   0.20  -1.37
#> Trial 2     4.78  2.80 4.80 4.22 -0.31 -2.05   0.23  -2.78  -3.72  -2.40
#> Trial 3     3.85  2.49 5.90 4.05  1.63  0.50  -0.65  -1.41  -1.18  -2.58
#> Trial 4     4.56  2.83 6.86 4.28  0.72 -2.41  -1.71  -0.42  -2.92  -2.45
#>         logsde
#> Basevec  -1.61
#> Trial 1  -1.81
#> Trial 2   0.48
#> Trial 3  -3.23
#> Trial 4  -2.85
#>
#> $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    NA     NA   NA   NA   NA   NA   NA   NA
#> 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.

2.8 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
```

2.9 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 a catch and in 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.

2.9.1 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$logbeta <- c(1, 1, 0)
fit.spict(inp)
#> Convergence: 0  MSG: relative convergence (4)
#> Objective function at optimum: 5.0598288
#> Euler time step (years):  1/16 or 0.0625
#> Nobs C: 23,  Nobs I1: 23
#>
#> Model parameter estimates w 95% CI
#>            estimate       cilow        ciupp    log.est
#>  alpha   39.0512852   0.0402369 3.790058e+04  3.6648758
#>  beta     0.0245071   0.0000265 2.269299e+01 -3.7087912
#>  r        0.1955750   0.0313372 1.220581e+00 -1.6318113
#>  rc       1.2604800   0.0097765 1.625129e+02  0.2314926
#>  rold     0.2835728   0.0024028 3.346681e+01 -1.2602862
#>  m       24.3112479  12.0981960 4.885330e+01  3.1909391
#>  K      210.4516033 123.9786492 3.572379e+02  5.3492557
#>  q        0.3842438   0.2070701 7.130113e-01 -0.9564779
#>  n        0.3103183   0.0004170 2.309135e+02 -1.1701567
#>  sdb      0.0028040   0.0000029 2.728471e+00 -5.8767196
#>  sdf      0.3809116   0.2827808 5.130959e-01 -0.9651879
#>  sdi      0.1094986   0.0812449 1.475777e-01 -2.2118438
#>  sdc      0.0093351   0.0000103 8.475656e+00 -4.6739791
#>
#> Deterministic reference points (Drp)
#>        estimate      cilow      ciupp    log.est
#>  Bmsyd 38.57459  0.5969775 2492.55451  3.6525937
#>  Fmsyd  0.63024  0.0048883   81.25643 -0.4616546
#>  MSYd  24.31125 12.0981960   48.85330  3.1909391
#> Stochastic reference points (Srp)
#>          estimate      cilow      ciupp    log.est  rel.diff.Drp
#>  Bmsys 38.5744682  0.5969870 2492.49896  3.6525906 -3.108161e-06
#>  Fmsys  0.6302411  0.0048883   81.25594 -0.4616529  1.767717e-06
#>  MSYs  24.3112135 12.0980443   48.85377  3.1909377 -1.413256e-06
#>
#> States w 95% CI (inp$msytype: s)
#>                   estimate      cilow       ciupp    log.est
#>  B_1989.00      54.0126818 28.3907712 102.7576805  3.9892189
#>  F_1989.00       0.4528693  0.2235663   0.9173593 -0.7921517
#>  B_1989.00/Bmsy  1.4002184  0.0314790  62.2830913  0.3366283
#>  F_1989.00/Fmsy  0.7185652  0.0079263  65.1417184 -0.3304988
#>
#> Predictions w 95% CI (inp$msytype: s)
```

```
#>                 prediction      cilow      ciupp    log.est
#>  B_1990.00      52.5650582 29.3174859 94.2470086  3.9620516
#>  F_1990.00       0.4858020  0.2370661  0.9955179 -0.7219542
#>  B_1990.00/Bmsy  1.3626904  0.0266106 69.7815404  0.3094610
#>  F_1990.00/Fmsy  0.7708193  0.0076917 77.2474057 -0.2603013
#>  Catch_1990.00  25.2158689 15.5407631 40.9143386  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.

2.9.2 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"
#>  [6] "logm"       "logq"       "iqgamma"    "logqf"      "logbkfrac"
#> [11] "logB"       "logF"       "logBBmsy"   "logFFmsy"   "logsdb"
#> [16] "isdb2gamma" "logsdf"     "isdf2gamma" "logsdi"     "isdi2gamma"
#> [21] "logsde"     "isde2gamma" "logsdc"     "isdc2gamma" "logsdm"
#> [26] "logpsi"
```

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  59.6064738  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 358.7037399 -0.2081839
#>  m       22.5701177  17.0283130  29.9154832  3.1166268
#>  K      202.2160641 138.6995451 294.8195436  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
```

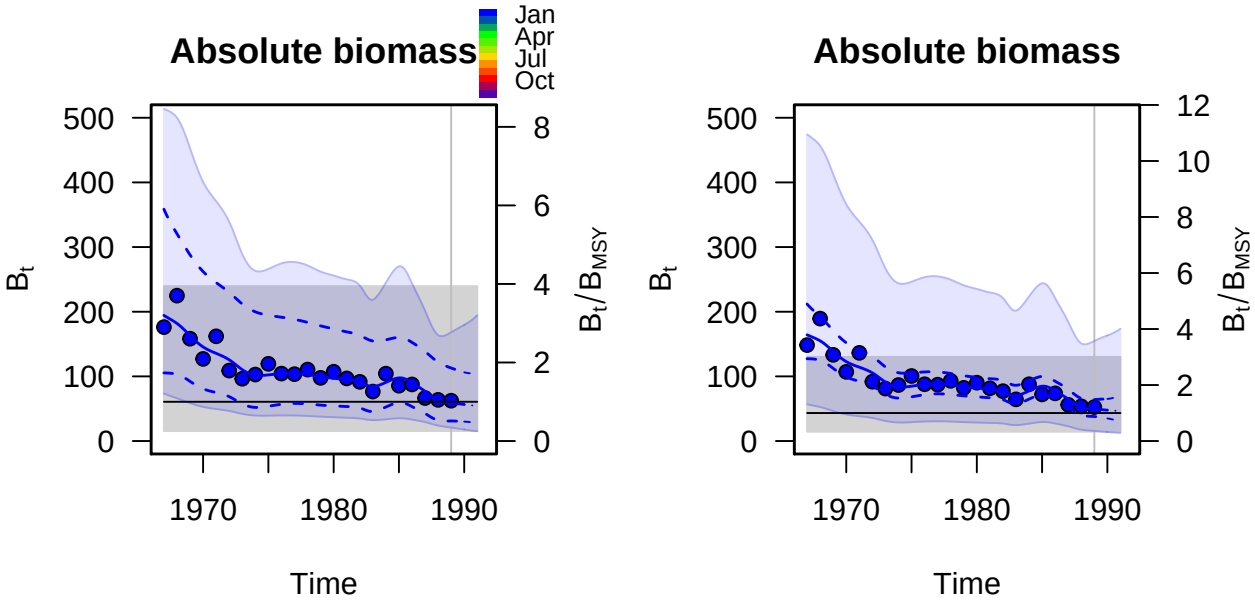
```
#>
#> Deterministic reference points (Drp)
#>         estimate      cilow      ciupp    log.est
#>  Bmsyd 60.927699 15.2912403 242.765427  4.1096879
#>  Fmsyd  0.370441  0.0711624   1.928357 -0.9930611
#>  MSYd  22.570118 17.0283130  29.915483  3.1166268
#> Stochastic reference points (Srp)
#>          estimate      cilow      ciupp    log.est  rel.diff.Drp
#>  Bmsys 60.9200355 15.2914369 242.701241  4.1095621 -1.257924e-04
#>  Fmsys  0.3704531  0.0711556   1.928668 -0.9930283  3.266222e-05
#>  MSYs  22.5680187 17.0227429  29.919706  3.1165338 -9.300659e-05
#>
#> States w 95% CI (inp$msytype: s)
#>                    estimate      cilow       ciupp    log.est
#>  B_1989.00      59.4317253 31.0677432 113.6912309  4.0848282
#>  F_1989.00       0.4143985  0.2035123   0.8438121 -0.8809271
#>  B_1989.00/Bmsy  0.9755694  0.3412397   2.7890532 -0.0247339
#>  F_1989.00/Fmsy  1.1186260  0.2875234   4.3520786  0.1121012
#>
#> Predictions w 95% CI (inp$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.

2.9.3 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 is propagated 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.

2.9.4 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 17.0613915  28.3905610  3.0914374
#>  K      174.4569109 74.7554134 407.1305658  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 87.2283940 37.377636 203.5653828  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 199.7713777  4.456347 -0.012257037
#>  Fmsys  0.2500268  0.0921861   0.678122 -1.386187 -0.009136168
#>  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      54.5156896 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)       20.7049019         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.

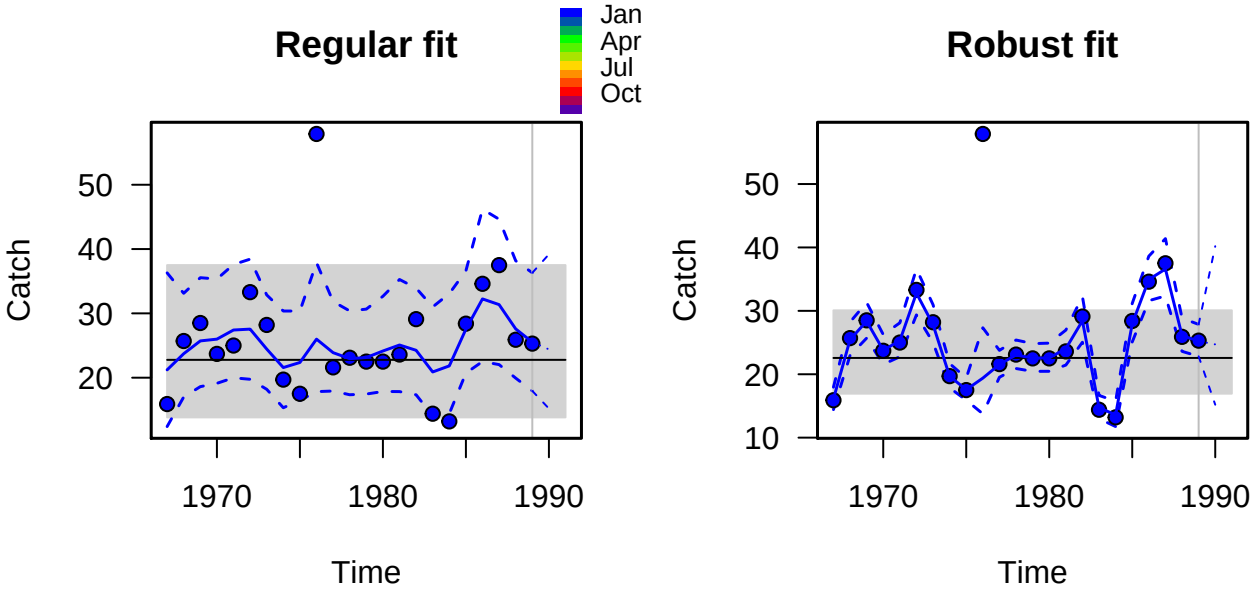
2.9.5 Pitfalls when fixing parameters and specifying priors

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.

2.10 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 index 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   1.14064157  56.30299055  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   3.84722638 -0.3101324
#> rold     0.85759756   0.00140610 523.05883524 -0.1536203
#> m       22.57344335  16.94359772  30.07391660  3.1167741
#> K      202.06196157 137.06067490 297.89023252  5.3085744
#> q        0.34766878   0.18846113   0.64137140 -1.0565050
#> n        0.70049565   0.06408927   7.65641661 -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   2.67330916 236.13438045  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 (2016). 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`.

2.11 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`). 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:

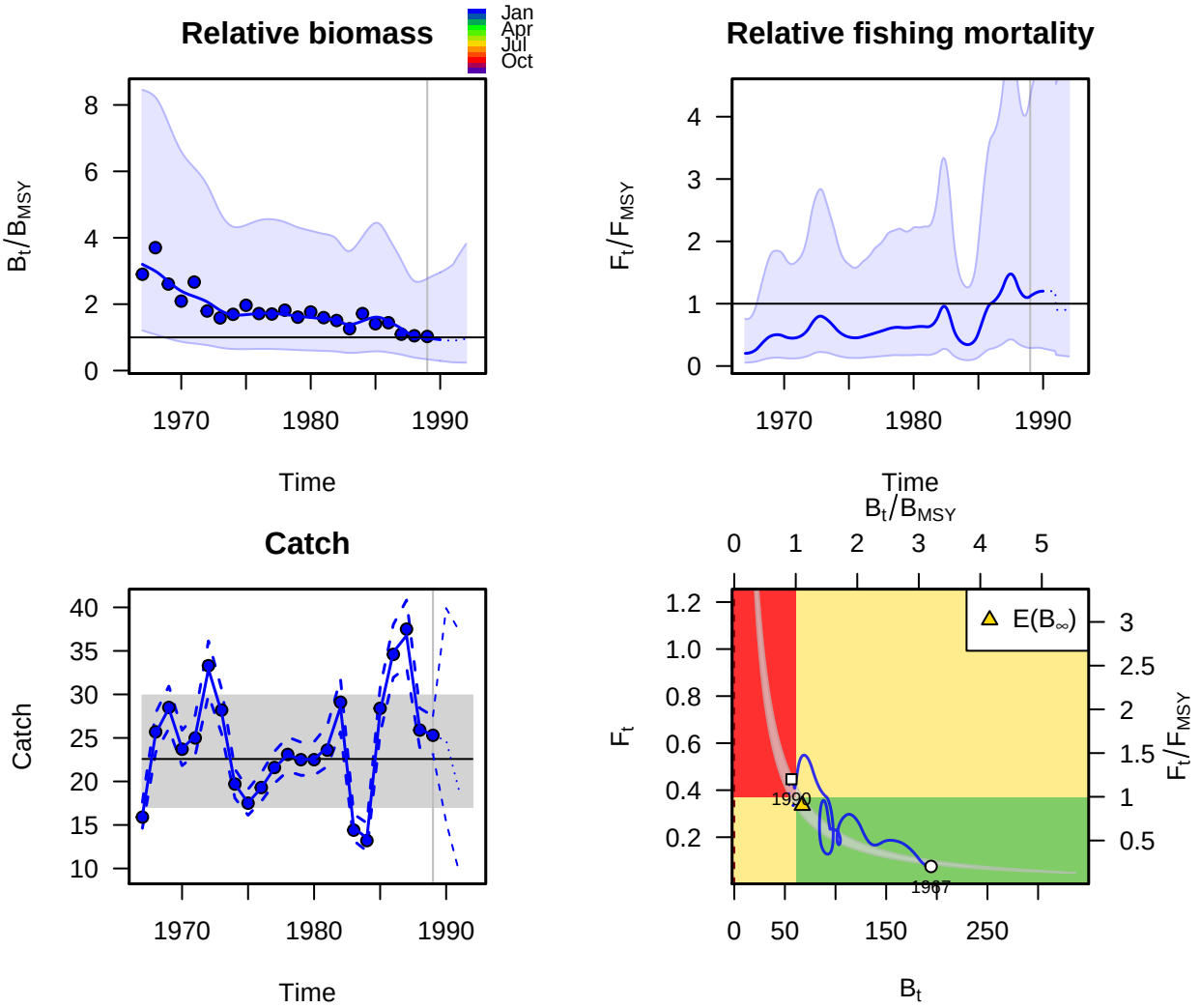
```
library(spict)
data(pol)
inp <- pol$albacore
inp$manstart <- 1991
inp$timepredc <- 1991
inp$dtpredc <- 1
inp$timepredi <- 1992
inp$ffac <- 0.75
res <- fit.spict(inp)
```

To specifically show forecast results use

```
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

```
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.

2.11.1 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
#>
```

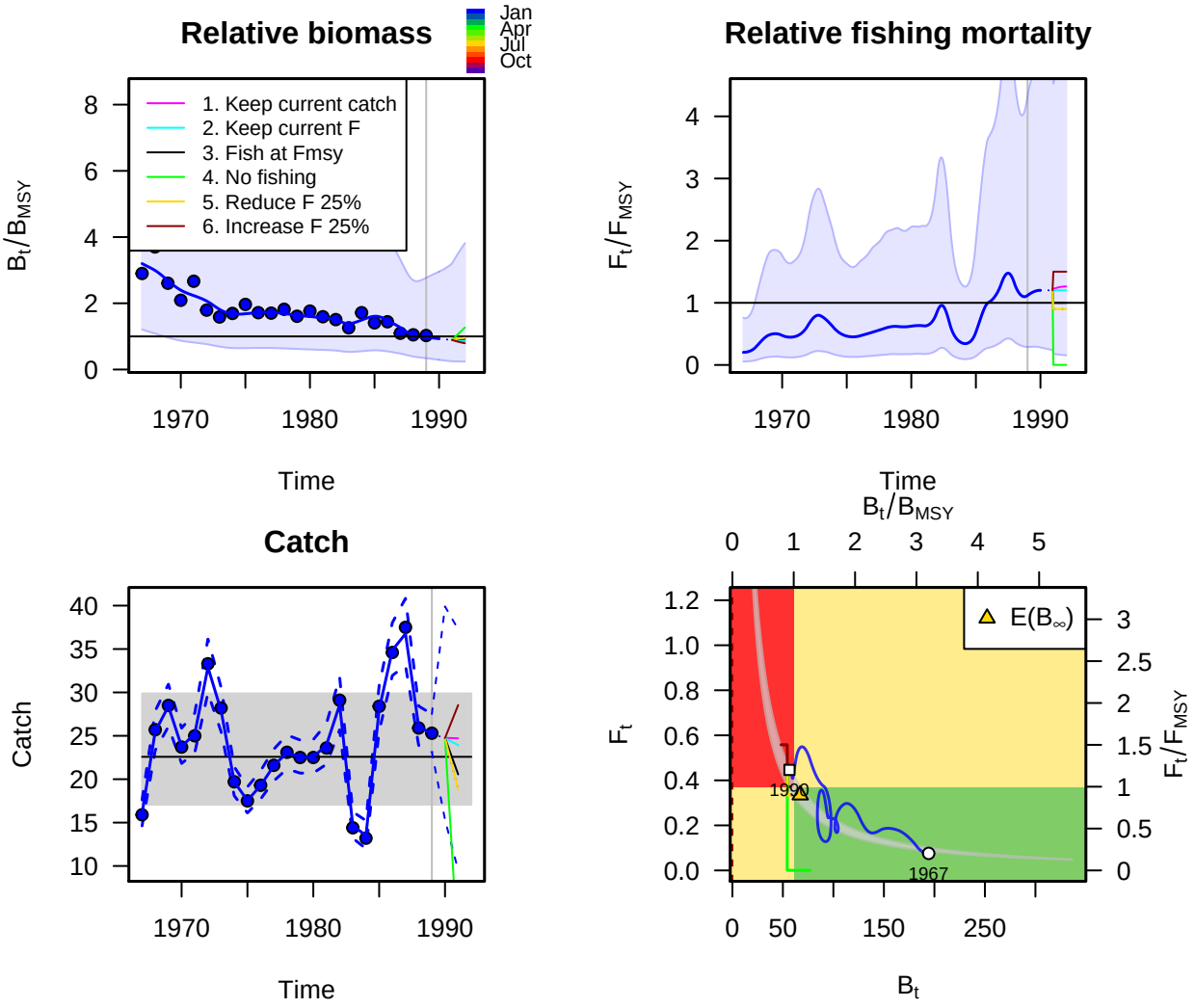
```
#> Predictions
#>                          C    B     F B/Bmsy F/Fmsy perc.dB perc.dF
#> 1. Keep current catch 24.7 52.0 0.470  0.857  1.263    -4.2     5.2
#> 2. Keep current F     23.9 52.9 0.446  0.870  1.201    -2.7     0.0
#> 3. Fish at Fmsy       20.6 56.3 0.372  0.926  1.000     3.6   -16.7
#> 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
#>
#> 95% CIs of absolute predictions
#>                       C.lo C.hi B.lo  B.hi  F.lo  F.hi
#> 1. Keep current catch 22.7 26.9 26.0 104.2 0.214 1.032
#> 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
#>
#> 95% CIs of relative predictions
#>                       B/Bmsy.lo B/Bmsy.hi F/Fmsy.lo F/Fmsy.hi
#> 1. Keep current catch     0.202     3.634     0.231     6.911
#> 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
```

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 24.7 52.0 0.470  0.857  1.263    -4.2     5.2
#> 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)
```



3 Other model settings and options

3.1 `catchunit` - Define unit of catch observations

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

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

3.2 `dteuler` - 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 time step. The influence of `dteuler` 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.

3.3 `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'`.

3.4 `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.

3.5 `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.

3.6 `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. 2016. "A Stochastic Surplus Production Model in Continuous Time." *Fish and Fisheries*. Wiley Online Library. doi: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). NRC Research Press: 2597–2607.