Model evaluation and selection

1 Required packages

```r
# install.packages(c("corrplot", "doRNG", "dplyr", "forecast", "GA", "ggplot2", "glmmTMB", "icesAdvice"

library(corrplot)
library(doRNG)
library(dplyr)
library(forecast)
library(GA)
library(ggplot2)
library(glmmTMB)
library(icesAdvice)
library(knitr)
library(memoise)
library(MuMIn)
library(parallel)
library(patchwork)
library(tidyr)
```

2 Introduction

The following examples focus on some strategies for selecting statistical models for use in management strategy evaluation (MSE).

Some general criteria for model selection:

- **We want a model that makes biological sense**
  - Includes plausible covariates, even if the precise mechanism is not known
- **We want a model that fits the observed data well**
  - But, careful not to overfit data
- **We want a model that can make good predictions**
  - Can the model also predict well into the future?
  - Is it better than a simpler (naive) assumption of future values

3 Synthetic data creation

We will generate synthetic data for the following examples, both for simplicity and to gain insight into the model selection process, as we will know the "true" underlying relationship.

The example will be to fit an environmentally-mediated stock-recruitment relationship (EMSRR), using a modified Ricker (e.g. "controlling"-type as described by Levi et al. 2003; Iles and Beverton 1998):

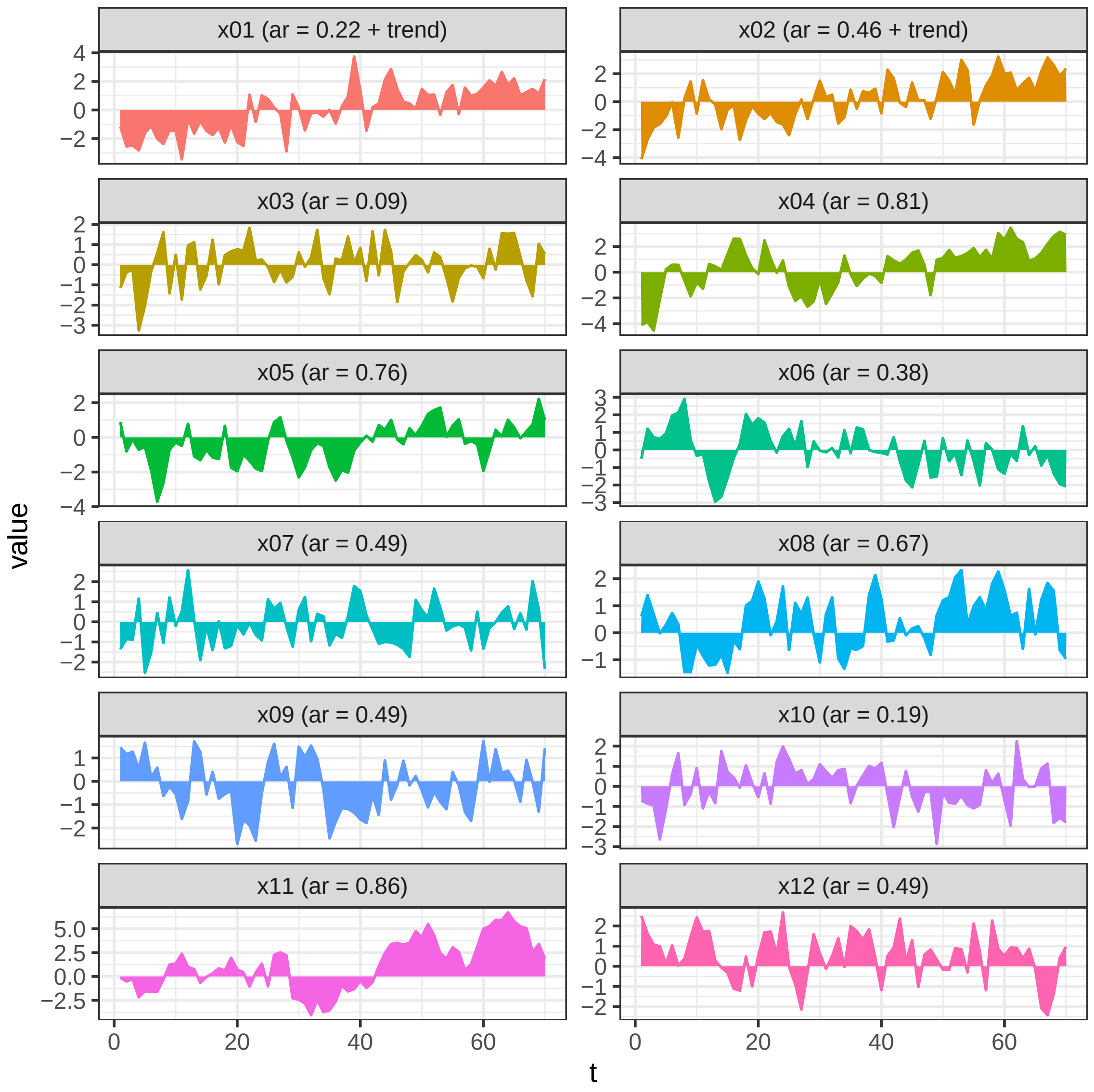
- $Ricker$: $R = \alpha SSB e^{-\beta SSB}$; $log(R) = log(\alpha) + log(SSB)\check{}\beta SSB$
- $Ricker\ wih\ env.:$ $R = \alpha SSB e^{-\beta SSB} e^{cE}$; $log(R) = log(\alpha) + log(SSB)\check{}\beta SSB + cE$

3.1 Environmental covariates

First, we define some environmental covariates using an autoregressive integrated moving average (ARIMA) model. Only AR1 processes are included, which define the the correlation between prior and subsequent values.

In a real modelling exploration, these covariates should be chosen with some plausible connection to the biological process at hand; for example, a temperature signal corresponding to the main spawning period. However, there may be many potential environmental drivers.

```r
set.seed(1235)
n <- 70
ncov <- 12
envvars <- paste0("x", formatC(x = seq(ncov), digits = 1, flag = 0))
AR <- runif(ncov, min = 0, max = 0.9) # auto-regression coefficients
names(AR) <- envvars

tmp <- lapply(seq(ncov), FUN = function(i){
  # Generate the time series
  ts_data <- arima.sim(model = list(ar = AR[i]), n = n)
  return(ts_data)
})

tmp <- as.data.frame(matrix(unlist(tmp), nrow = n, ncol = ncov))
names(tmp) <- envvars
dat <- cbind(data.frame(t = seq(n)), tmp)

# add a trend to x01 and x02
dat$x01 <- dat$x01 + diff(range(dat$x01))*0.01*(dat$t-mean(dat$t)) # 1% of sd per year
dat$x02 <- dat$x02 + diff(range(dat$x02))*0.01*(dat$t-mean(dat$t)) # 1% of sd per year

# plot time series
df <- tidyr::pivot_longer(dat, cols = seq(ncol(dat))[-1])
trend <- seq(envvars)*0
trend[1:2] <- 1
labs <- paste0(envvars, " (ar = ", as.character(round(AR, 2)),
  ifelse(trend, " + trend", ""), ")")
names(labs) <- envvars
ggplot(df) + aes(x = t, y = value, fill = name, color = name) +
  geom_area(show.legend = F) +
  facet_wrap(~ name, ncol = 2, scales = "free_y", labeller = labeller(name = labs)) +
  theme_bw()
```

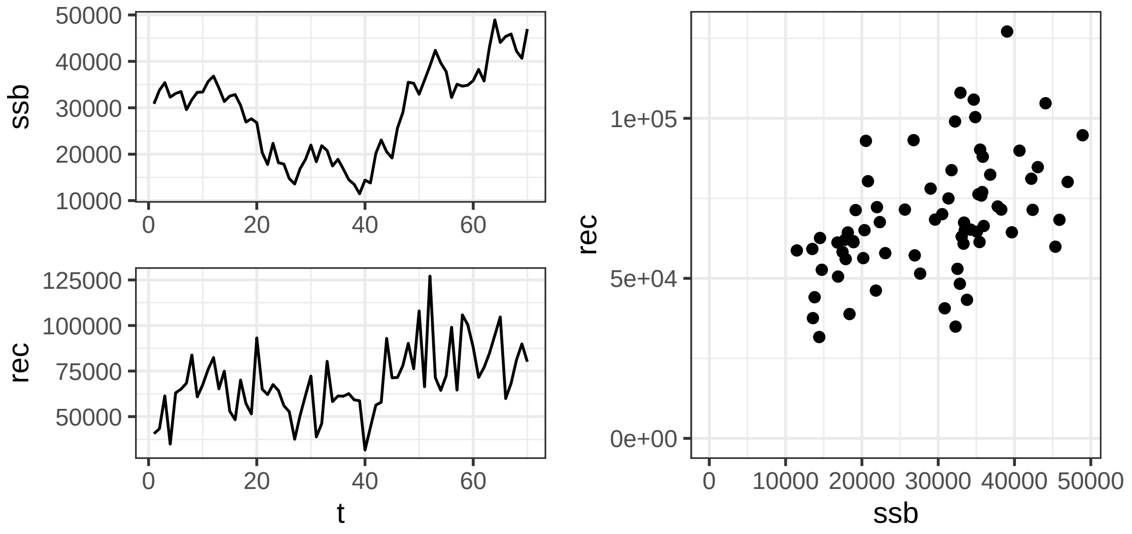


3.2 Biological covariates and response

Spawning stock biomass (SSB) is generated here as a random walk process, and recruitment is predicted using SSB and the mediation of two of the above environmental covariates ("x01" and "x03"). Additional noise is also added (random log-normal with a coefficient of variation of 20%).

Note that in reality, recruitment is often lagged behind SSB and depends on the minimum age included in the assessment. In this data set, let's assume that t refers to the time when spawning biomass was assessed.

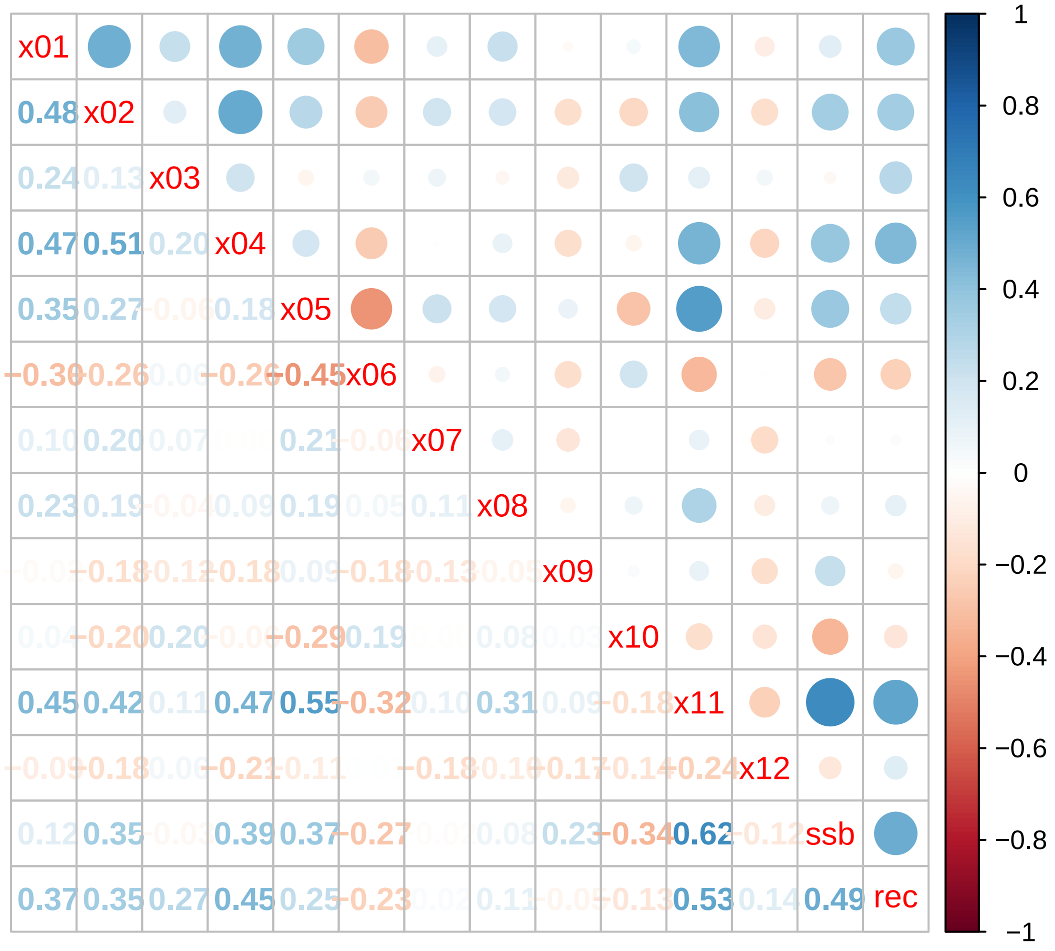
```
# generate ssb (random walk)
set.seed(236)
dat$ssb <- 30000 - cumsum(rnorm(n, 0, 3000))

# generate recruitment (Ricker model + env mediation from x01 & x03)
dat$rec <- exp(log(5) + log(dat$ssb) - 2.5e-5*dat$ssb + 0.05*dat$x01 + 0.05*dat$x03)
dat$rec <- dat$rec * rlnorm(n, 0, 0.2) # additional random noise

p1 <- ggplot(dat) + aes(x = t, y = ssb) +
  geom_line()
p2 <- ggplot(dat) + aes(x = t, y = rec) +
  geom_line()
p3 <- ggplot(dat) + aes(x = ssb, y = rec) +
  geom_point() +
  lims(x = c(0, NA), y = c(0, NA))

layout <- "
AC
BC
"
p <- p1 + p2 + p3 + plot_layout(design = layout, axis_titles = "collect") &
  theme_bw()
p
```



3.3 Resulting correlations among variables

The resulting correlations among environmental and biological covariates is shown below.

```
corrplot.mixed(cor(dat[,-1]))
```



3.4 Full model definition

We will start with a full model that includes all environmental covariates (`fit0`). This is clearly more complicated than our true relationship, and the task below will be to try and eliminate insignificant covariates that just add noise to the prediction.

We will be fitting a Generalized Linear Model (GLM) (`glm()`), where we define a specific error distribution and link function to be used in the model. A log-link is sensible given that we are using the log-transformed version of the Ricker, and the fact that our response variable (recruitment) can only be positive.

A `summary` of the model already provides us with information about the potential significance of each term.

Based on these p-values, one could start removing terms that have lowest significance, and try to arrive to a simpler model; but, with so many covariates, this could be a time-consuming endeavor when done manually.

```
fmla <- formula(paste(c("rec ~ offset(log(ssb)) + ssb", envvars), collapse = " + "))
fmla
```

```
## rec ~ offset(log(ssb)) + ssb + x01 + x02 + x03 + x04 + x05 +
##     x06 + x07 + x08 + x09 + x10 + x11 + x12
```

```
fit0 <- glm(fmla, data = dat, family = gaussian(link = "log"))
summary(fit0)
```

```
##
## Call:
```

```
## glm(formula = fmla, family = gaussian(link = "log"), data = dat)
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.627e+00  1.211e-01  13.435  < 2e-16 ***
## ssb         -2.712e-05  4.029e-06  -6.731 9.64e-09 ***
## x01          3.734e-02  2.257e-02   1.654   0.1036
## x02          1.110e-02  2.171e-02   0.511   0.6111
## x03          5.202e-02  2.645e-02   1.967   0.0542 .
## x04          7.466e-03  2.032e-02   0.367   0.7147
## x05         -9.060e-03  2.905e-02  -0.312   0.7563
## x06         -1.113e-02  2.481e-02  -0.448   0.6556
## x07          1.608e-02  2.521e-02   0.638   0.5262
## x08          1.345e-02  2.557e-02   0.526   0.6009
## x09         -2.467e-02  2.603e-02  -0.948   0.3474
## x10          9.328e-03  2.499e-02   0.373   0.7104
## x11          2.051e-02  1.568e-02   1.308   0.1961
## x12          5.378e-02  2.422e-02   2.220   0.0305 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 199130548)
##
##     Null deviance: 3.0082e+10  on 69  degrees of freedom
## Residual deviance: 1.1151e+10  on 56  degrees of freedom
## AIC: 1550.7
##
## Number of Fisher Scoring iterations: 6
```

4 Model selection with Akaike information criterion

A large part of model building is selecting the best model to describe the response variable. An important concept to remember when looking for the "best" model is whether the addition of terms improves the predictive power of the model.

Generally, the more explanatory variables that one has, the more variation will be explained in the response variable, but one has to be careful not to include variables that do not explain a *significant* portion of the variation. The inclusion of these terms might help you achieve a higher R^2 value for your specific data set, but are likely to be a poor predictor of new data that was not seen during the fitting. This problem is referred to as "*overfitting*".

The Akaike information criterion (AIC) is an estimator of prediction error and thereby relative quality of statistical models for a given set of data: $AIC = 2k - ln(\hat{L})$, where k is the number of estimated parameters and $\hat{L}$ is the maximized value of the likelihood function for the model. By considering the model complexity with k, AIC penalizes more complex models. Using this criteria, we would select a model based on the lowest AIC.

The strategy presented here relies on the comparison of "*nested models*" in determining whether a term should be kept or not. Nested models differ by a single term, whereby the larger model contains all of the terms of the smaller model, plus an additional one. One commonly used strategy, is to start with your most complex model and work backwards, sequentially removing variables that are not significant.

As a general rule, one should start by removing higher-order terms first (e.g. interaction terms) before moving on main terms or lower-level nested terms. Furthermore, some believe that there is a general rule stating that the main term should always be included if the term is also used in an interaction (see discussion here).

This logic is used within the automated stepwise search routine found in the function `step()` (introduced below).

4.1 Stepwise search

By default the `step` function uses a `direction = "backward"` search, which starts with the largest model and removes terms gradually according to the largest improvements (reductions) to the Akaike information criteria (AIC). When `direction = "both"`, terms may come back into the model at later steps.

The following example also specifies the simplest allowed model with the `lower` argument, which is defined here as the standard Ricker stock-recruitment relationship; e.g. we assume that a minimal mechanistically-based model exists where SSB is required to produce eggs and, thus, recruits.

Note that a general rule-of-thumb for significant improvements (e.g. $p < 0.05$) is that the change in AIC should be at least two units ($\delta AIC \geq 2$); otherwise, take the simpler of the two models. The `step` function does not use this criteria, and any reduction in AIC is considered as an improvement.

A comparison of AIC between the resulting model (`fit`) to the full one (`fit0`) shows the degree of improvement. The model was reduced to 5 environmental covariates before the search stopped (and some of these look to be less significant).

```r
# model with all original variables in fit0
all_variables <- "rec ~ ."
# Variables that should always be in the model (Ricker without env.)
fixed_variables <- "rec ~ offset(log(ssb)) + ssb"

fit <- step(fit0,
  scope = list(
    lower = as.formula(fixed_variables),
    upper = as.formula(all_variables)
  ),
  direction = "both",
  trace = FALSE # set to TRUE if you would like to see the model selections steps
)

summary(fit)
```

```
##
## Call:
## glm(formula = rec ~ ssb + x01 + x03 + x09 + x11 + x12 + offset(log(ssb)),
##     family = gaussian(link = "log"), data = dat)
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.624e+00  1.001e-01  16.214  < 2e-16 ***
## ssb         -2.663e-05  3.431e-06  -7.762 9.29e-11 ***
## x01          4.595e-02  1.819e-02   2.526   0.0141 *
## x03          5.323e-02  2.373e-02   2.243   0.0284 *
## x09         -3.247e-02  2.218e-02  -1.464   0.1482
## x11          2.296e-02  1.398e-02   1.643   0.1054
## x12          4.543e-02  2.189e-02   2.075   0.0420 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 182674743)
##
```

```
##     Null deviance: 3.0082e+10  on 69  degrees of freedom
## Residual deviance: 1.1509e+10  on 63  degrees of freedom
## AIC: 1538.9
##
## Number of Fisher Scoring iterations: 5
```

```
# comparison of AIC & R2
df <- data.frame(AIC=c(AIC(fit0), AIC(fit)),
  R2 = c(1 - fit0$deviance/fit0$null.deviance,
    1 - fit$deviance/fit$null.deviance),
  row.names = c("fit0", "fit"))
print(df)
```

```
##           AIC        R2
## fit0 1550.694 0.6293009
## fit  1538.901 0.6174258
```

4.2 Dredging

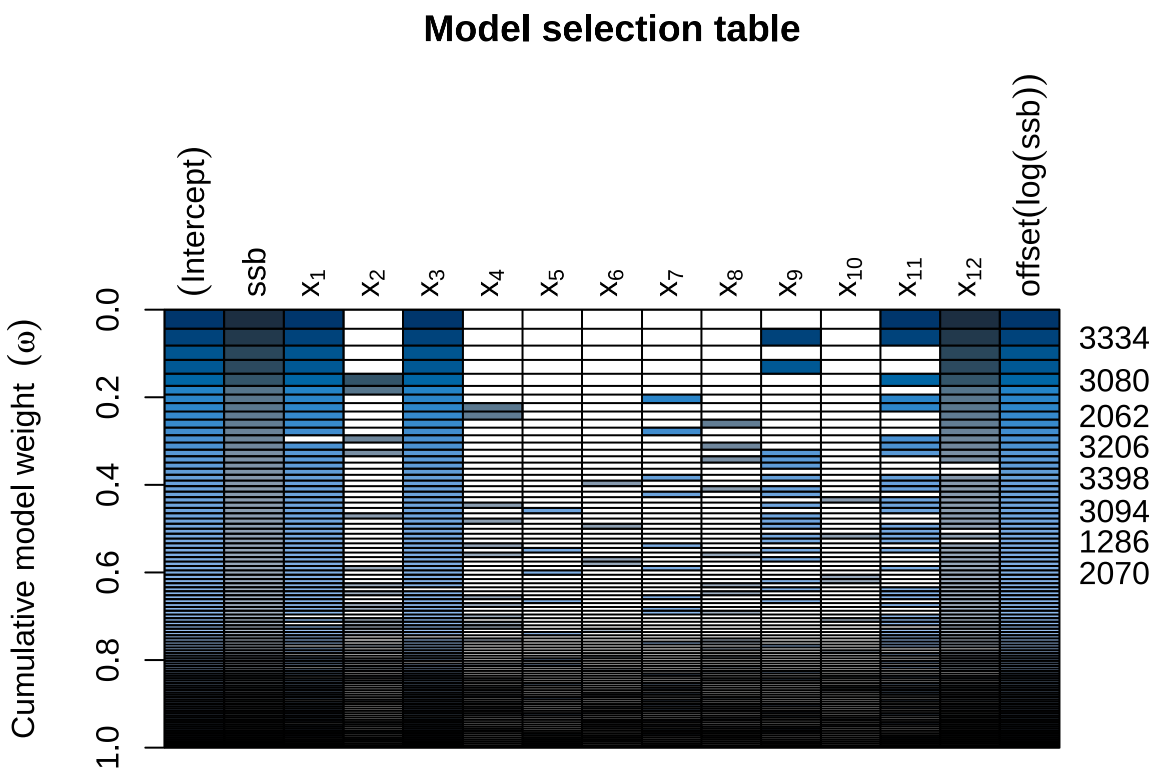
If your model fitting is fast, and there are not too many possible combinations ($combinations = 2^n - 1$, where n is the number of possible covariates), then one could try all possible models with the `dredge` function (package `MuMIn` – *Multi-Model Inference*), which can handle many model types. In our example, it's relatively quick to run through all possible models (n = 4095). The function `pdredge` can also run the process using parallel computing.

Again, we can define by the minimal model using the `fixed` argument.

```
fmla <- formula(paste(c("rec ~ offset(log(ssb)) + ssb", envvars), collapse = " + "))
# argument na.action = na.fail required for fair comparisons
fittmp <- glm(fmla, data = dat, family = gaussian(link = "log"), na.action = na.fail)

t1 <- Sys.time()
dd <- dredge(fittmp, fixed = c("offset(log(ssb))", "ssb"), trace = FALSE)
```

```
## Fixed terms are "offset(log(ssb))", "ssb" and "(Intercept)"
```

```
t2 <- Sys.time()
# t2 - t1

# subset(dd[1:100,], delta < 4)
plot(dd[1:100,], labAsExpr = TRUE)
```



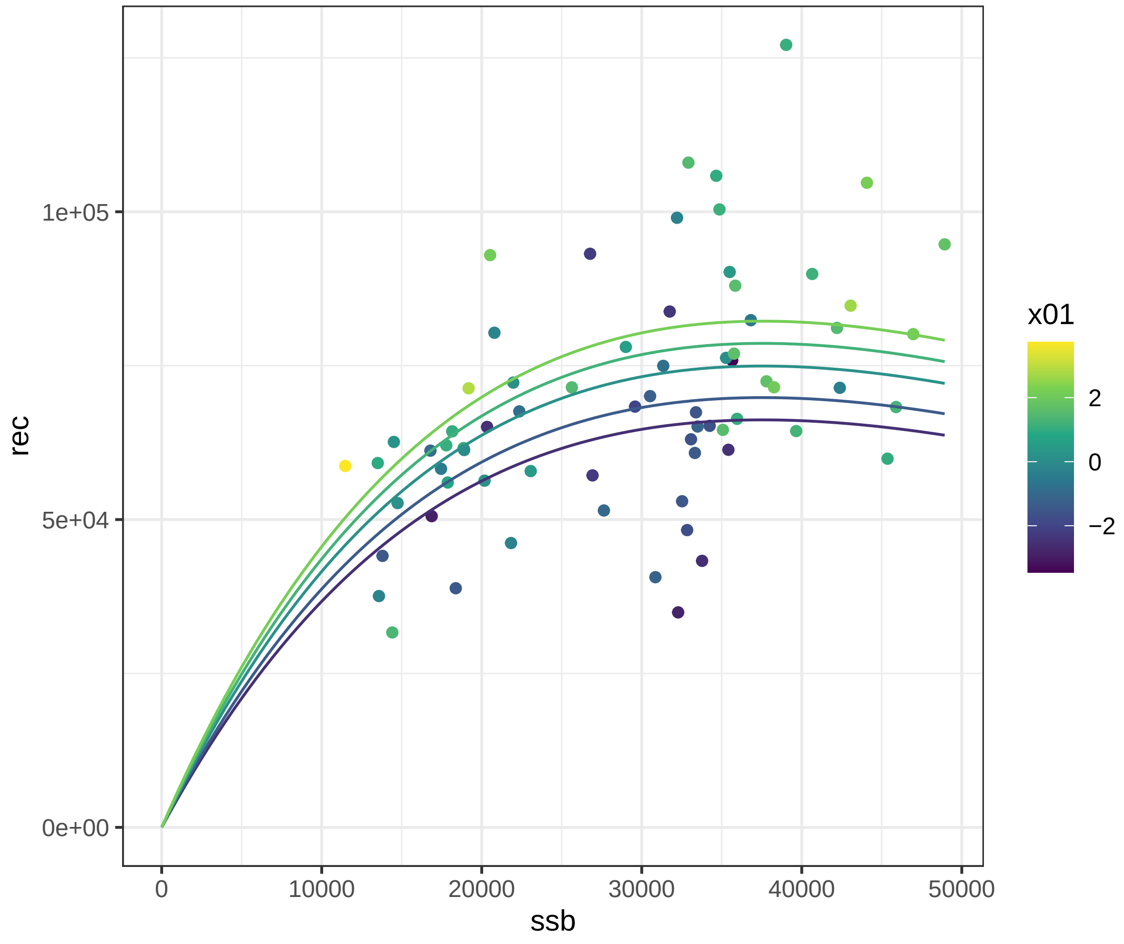
5 Visualizing marginal effects

Given the complex relationships that may result in models with many terms (and potential interactions), it may be helpful to visualize marginal effects of environmental covariates one at a time, holding others constant.

In the following example, we plot the predictions of the model, holding all but one of the environmental covariates to their long-term median values (i.e. constant). Several levels of the focus environmental covariate are used (e.g. quantiles from the historical time series) to show how the stock-recruitment relationship is influenced.

```
envvarsKeep <- intersect(names(coef(fit)), envvars) # maintained env. covariates
focusVar <- envvarsKeep[1] # choose covariate of focus

## generate data for prediction
# focus covariate generated at 5 quantile levels, other env covariates use median
L <- lapply(envvarsKeep, FUN = function(x){
  if(x == focusVar){probs = c(0.05,0.25,0.5,0.75,0.95)}else{probs <- 0.5}
  quantile(dat[,x], probs = probs)
})
names(L) <- envvarsKeep
L[["ssb"]] <-  seq(0, max(dat$ssb), len = 100)

nd <- expand.grid(L)
```

```r
nd$rec <- predict(fit, newdata = nd, type = "response")

ggplot(dat) + aes(x = ssb, y = rec, col = get(focusVar)) +
  geom_point() +
  scale_color_continuous(name = focusVar, type = "viridis") +
  geom_line(data = nd, mapping = aes(group = get(focusVar))) +
  theme_bw()
```



6 Model selection via cross validation

- Cross validation is a flexible framework that tests how well a model can predict independent (unseen) data.
- Data is typically split into two sets; one for model fitting (*training data*) and one for prediction (*validation data*).
- Can be used as a basis for selecting best performing model (e.g. inclusion of terms, model structure). Under careful design, cross validation can identify models that suffer from overfitting through decreased prediction performance.
- Performance is evaluated based on a desired metric (see below).

Cross validation differs from AIC in that it is a more general, data-driven approach that directly estimates predictive performance on new data, applicable to both parametric and non-parametric models.

AIC is a faster, theory-driven method that penalizes complexity based on the number of parameters, making it suitable for comparing parametric models.

6.1 Common metrics

Below are some common metrics that could be used for model selection. The choice of metric will depend on the goal of the model prediction.

Note that these metrics may differ from those by the model's fitting criteria to the training data set. Consistency between the two may be desirable, although one might want to emphasize other aspects of the prediction (validation) data set.

- Squared error metrics, like Mean Squared Error (MSE) and Root Mean Squared Error ($RMSE$), will more heavily penalize outliers and large errors.
- Mean Absolute Error MAE an intuitive metric that gives average error in the same units as the data.
- Mean Absolute Percentage Error ($MAPE$) and Mean Absolute Scaled Error ($MASE$) provide a scale-independent metric, useful when data varies significantly in magnitude. One may further decide whether data should be transformed (e.g. log-transformation) before evaluation by a metric.

Below are some R functions that can return these metrics, with arguments `y` and `yhat` providing the observed and predicted response values, respectively, of the validation data.

```r
# Mean Squared Error (MSE)
mse <- function(y, yhat){mean((y - yhat)^2, na.rm = TRUE)}

# Root Mean Squared Error (RMSE)
rmse <- function(y, yhat){sqrt(mean((y - yhat)^2, na.rm = TRUE))}

# Mean Absolute Percentage Error (MAPE)
mape <- function(y, yhat){mean(abs(y - yhat) / y, na.rm = TRUE)}

# Median Absolute Percentage Error (MdAPE)
mdape <- function(y, yhat){median(abs(y - yhat) / y, na.rm = TRUE)}

# Mean Absolute Error (MAE)
mae <- function(y, yhat){mean(abs(y - yhat), na.rm = TRUE)}

# Mean Absolute Scaled Error (MASE)
mase <- function(y, yhat, yhat_naive){mean(abs(y - yhat), na.rm = TRUE) /
    mean(abs(y - yhat_naive), na.rm = TRUE)}
```

6.2 Flavor 1: Repeated k-fold

Many flavors of cross validation exist, but we will focus on one of the most common and flexible ones – *k-fold cross validation*.

In this approach, k signifies the number of random groupings (i.e. *folds*) to use for the training and validation data. The groups are balanced, such that each data sample will be used once for validation.

For example, the commonly used 5-fold cross validation divides the data into 5 groups, with each fold using 4 out of 5 of the groups for training (~80% of data) and the remaining group for validation (~20% of data).

To increase the robustness of the prediction error estimates, the procedure can be repeated a number of times, with fold assignment changing randomly for each permutation.

6.2.1 Required functions

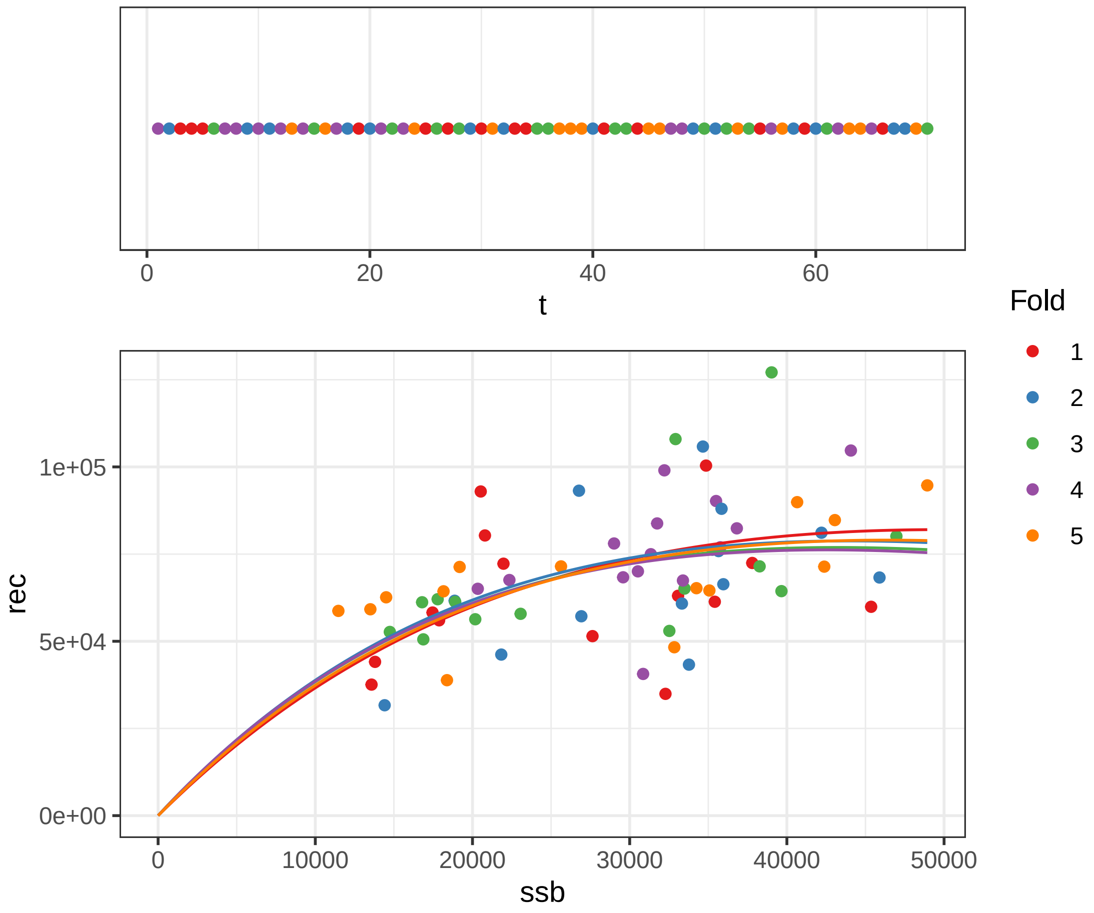
The following functions will be used in the examples:

- `kfold()` - Randomly partitions the data into the prescribed number of folds (`k`)
- `cvFun()` - Re-fits a defined model to each of the folds defined by `kfold` and records the observed and predicted response values (inner loop), which can be repeated for a defined number of permutations,`nperm` (outer loop). The setting of a random `seed` value ensures reproducibility among fold assignments, which is desired when comparing several models.

```r
# Determines k-fold partitions for a given number of samples
# n is the number of samples; k is the number of partitions
kfold <- function(n, k = NULL){
  if(is.null(k)){k <- n} # if undefined, assume leave-one-out (LOO) CV
  fold <- vector(mode="list", k)
  n.remain <- seq(n)
  for(i in seq(k)){
    samp <- sample(seq(length(n.remain)), ceiling(length(n.remain)/(k-i+1)))
    fold[[i]] <- n.remain[samp]
    n.remain <- n.remain[-samp]
  }
  return(fold)
}

# perform repeated k-fold cross validation
# n and k are passed to kfold; seed is a integer value passed to set.seed
cvFun <- function(model = NULL, nperm = 10, k = 5, seed = 1){
  modelData <- model.frame(model)
  y <- model.response(model.frame(model))
  fmla <- formula(model)
  # when k = number of sample, a single leave-one-out cross validation is performed (LOOCV)
  if(k == length(y)){nperm <- 1}
  res <- vector("list", nperm)
  set.seed(seed)
  for(j in seq(nperm)){ # for each permutation j
    fold <- kfold(n = nrow(modelData), k = k)
    res[[j]] <- data.frame(perm = j, fold = NaN, y = y, yhat = NaN)
    for(i in seq(fold)){ # for each fold
      train <- -fold[[i]]
      valid <- fold[[i]]
      # update model with training data
      fit.fold <- update(object = model, data = modelData[train,],
        formula = as.formula(fmla, env = environment()))
      # predict validation data
      res[[j]]$yhat[valid] <- predict(object = fit.fold,
        newdata = modelData[valid,], type = "response")
      res[[j]]$fold[valid] <- i
    }
  }
  res <- do.call("rbind", res)
  return(res)
}
```

6.2.2 Fitting to each fold

The following figure shows how the data samples in a 5-fold cross validation are randomly assigned to folds (colors) as validation data sets. For each fold, error is assessed based on the predicted (lines) versus observed (points) from a given fold.



6.2.3 Influence of fold number on error

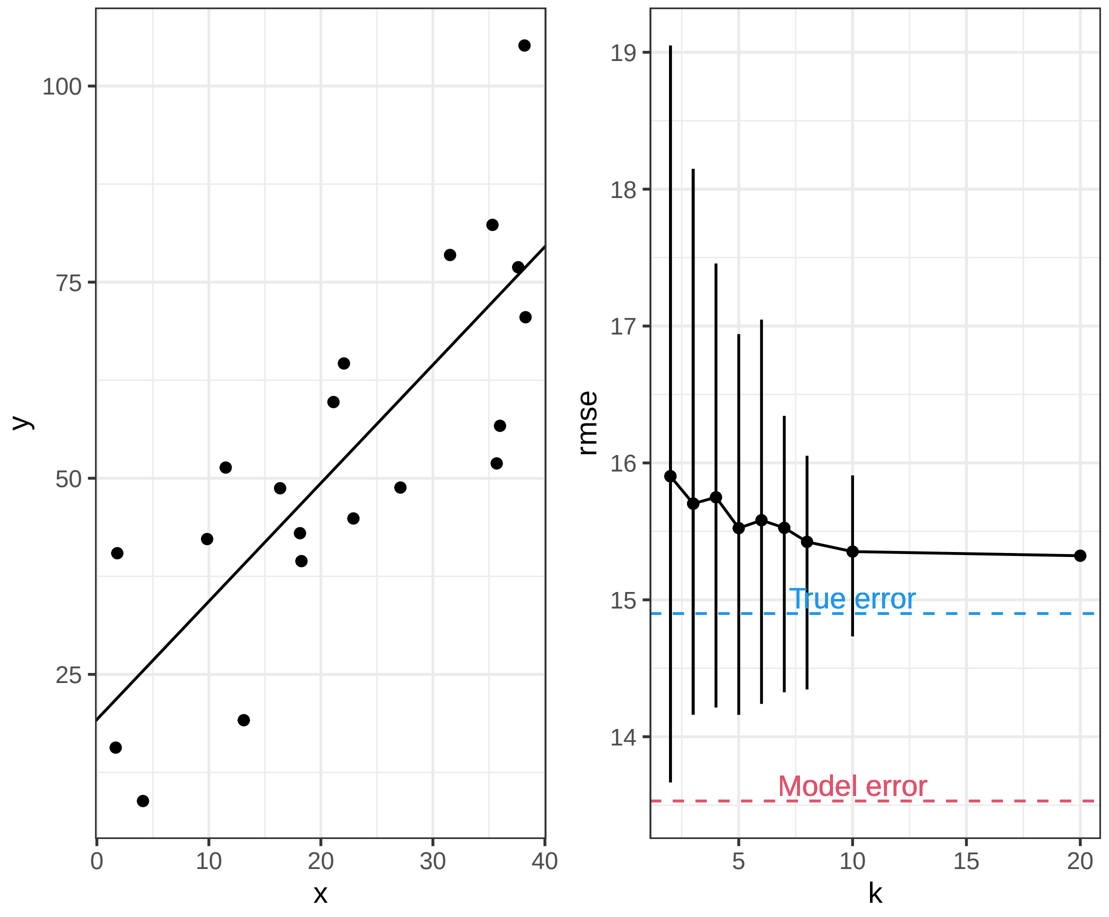
The number of folds chosen depends on several considerations; such as the variance in the data, number of samples, and computational cost. A low k (e.g. 5-fold) typically provides a good balance for the types of models that we have looked at. The larger amount of validation data (20%) provides a robust measure of prediction error, and the low number of folds is not computationally costly. That said, performing repeated k-fold cross validation (i.e. multiple permutations) will further reduce variability in error estimation.

The number of folds will influence the resulting metric of fit to the validation set. The more folds, the closer one gets to the error of the data, with a leave-one-out cross validation ($LOOCV$) being the least biased.

In the following example, a simple linear model is evaluated by repeated k-fold cross validation. As the number of folds increases, the estimated error of the validation set approaches the true error. The vertical lines shows the distribution of error values from across the permutations (5% and 95% quantiles).

Note that model error estimated by the fitting to the full data (based on residuals) tends to underestimate the true prediction error because it is calculated on the same data used to train the model. This bias can lead to overfitting, where the model appears to perform better on the training data than it would on unseen data.

LOOCV provides a *nearly* unbiased estimate of the prediction error because each data point is excluded from the training set when its error is calculated. This does not mean that the underlying model couldn't still suffer from overfitting, and a larger % holdout (e.g. 5- or 10-fold cross validation) is usually preferable.



6.3 Flexible model selection

The flexible nature of cross validation also allows for the comparison of models of different structure. For example, AIC may not be comparable between models of different structure, e.g. mixed models with random effects. Comparison should be made using same folds (e.g. use of random seed). A variety of metrics could be used for the final model selection.

```
res0 <- cvFun(model = fit0, nperm = 5, k = 5, seed = 1111)
res <- cvFun(model = fit, nperm = 5, k = 5, seed = 1111)

# mixed model with autoregressive term on time (t)
fmlatmp <- formula(fit)
fmlaMM <- as.formula(paste(c(deparse(fmlatmp), "+ ar1(t + 0 | 1)")))
fitMM <- glmmTMB(fmlaMM, data = dat, family = gaussian(link = "log"))
resMM <- cvFun(model = fitMM, nperm = 5, k = 5, seed = 1111)

cvStats0 <- with(res0, c(
  mse = mse(y = y, yhat = yhat),
  rmse = rmse(y = y, yhat = yhat),
  mape = mape(y = y, yhat = yhat),
  mdape = mdape(y = y, yhat = yhat),
  mae = mae(y = y, yhat = yhat)
))
```

```r
cvStats <- with(res, c(
  mse = mse(y = y, yhat = yhat),
  rmse = rmse(y = y, yhat = yhat),
  mape = mape(y = y, yhat = yhat),
  mdape = mdape(y = y, yhat = yhat),
  mae = mae(y = y, yhat = yhat)
))

cvStatsMM <- with(resMM, c(
  mse = mse(y = y, yhat = yhat),
  rmse = rmse(y = y, yhat = yhat),
  mape = mape(y = y, yhat = yhat),
  mdape = mdape(y = y, yhat = yhat),
  mae = mae(y = y, yhat = yhat)
))

tmp <- cbind(cvStats0, cvStats, cvStatsMM)
tmp <- as.data.frame(t(apply(tmp, 1, FUN = function(x){
  paste0(format(x, scientific = TRUE, digits = 3),
    ifelse(x == min(x, na.rm = T), "*", " "))
})))

names(tmp) <- c("res0", "res", "resMM")
tmp
```

```
##             res0       res     resMM
## mse   2.58e+08  2.04e+08* 1.62e+10
## rmse  1.61e+04  1.43e+04* 1.27e+05
## mape  2.02e-01  1.78e-01* 7.44e-01
## mdape 1.67e-01  1.56e-01* 5.53e-01
## mae   1.32e+04  1.15e+04* 5.04e+04
```

In this case, the addition of the random intercept with an autoregressive structure based on time (t) did not improve the model for any of the evaluated metrics.

7 Automated model selection with machine learning

Depending on the number of potential candidate models, as well as the complexity of the model fitting and cross validation procedures, model selection can become computationally costly. Thus, a brute force approach of evaluating all models variants with cross validation could be unrealistic.

In the example by Kühn, Taylor, and Kempf (2021), a large number of candidate covariates (n = 82) were chosen based on hypothesized biological mechanisms affecting recruitment of North Sea cod. These included time series signals extracted from different environmental fields (e.g. principal components from temperature, salinity, and currents) and at different temporal periods (seasons and lags).

The approach used an efficient search algorithm called *non-dominated sorting genetic algorithm* (NSGA-II), combined with cross validation, to explore the large number of potential model configurations (n = 4.84e+24).

In the following example, we will explore a version of the approach using a simpler implementation of a genetic algorithm.

7.1 Fitness function

One starts with a *fitness function*, whose first argument is a binomial vector indicating which covariates to include (i.e. genes that are turned on/off) in the model. The output of the function should be a single value, which the algorithm will use to rank models, and decide which covariate s will contribute to new offspring combinations in the next generation.

In the following fitness function, a repeated k-fold cross validation is run, and the resulting fitness is a single value of mean absolute percent error (MAPE). When `fitnessOnly = FALSE`, the function will return additional information about the variability of MAPE among permutations, useful in final model selection discussed below (see **Pareto front** section below).

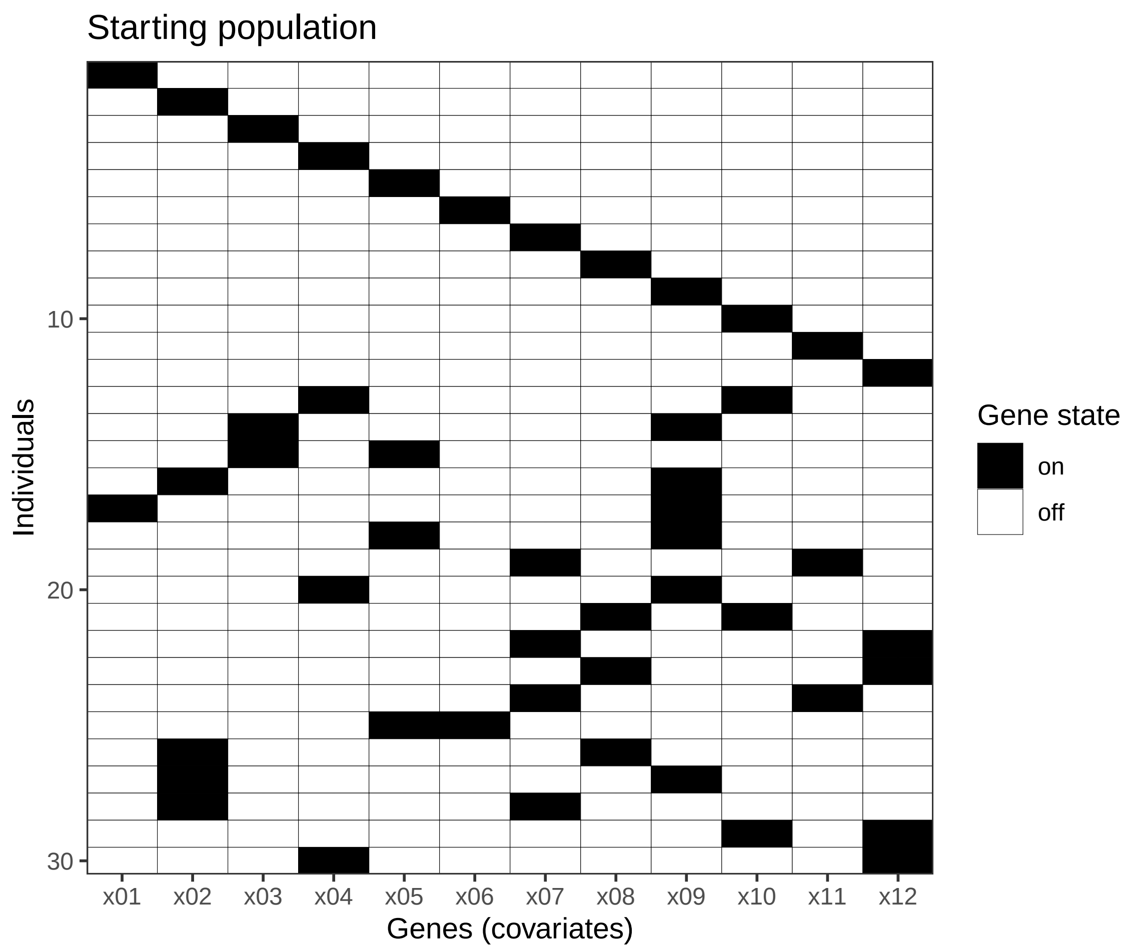
```
fitnessFun <- function(genes = NULL, data = NULL, nperm = 10, k = 5,
  ffSeed = 1, fitnessOnly = TRUE){

  fmla <- as.formula(
    paste(c("rec ~ offset(log(ssb)) + ssb", envvars[as.logical(genes)]), collapse = " + "),
    env = environment())
  fit <- glm(formula = fmla, data = data, family = gaussian(link = "log"))

  res <- cvFun(model = fit, nperm = nperm, k = k, seed = ffSeed)

  fitness <- -1 * mape(y = res$y, yhat = res$yhat) # to maximize, so reverse sign
  if(!fitnessOnly){
    byGroup <- res %>% group_by(perm) %>% summarise(mape = -1*mape(y = y, yhat = yhat))
    qs <- quantile(byGroup$mape, probs = c(0.05, 0.95))
  }
  if(fitnessOnly){
    out <- fitness
  }else{
    out <- list(fitness = fitness, byGroup = byGroup, lwr = qs[1], upr = qs[2])
  }
  return(out)
}
```

Let's create a starting population of model configurations to begin the search and visualize the initial fitness of the individuals (i.e. model configurations).

It is advisable to start off with simplest models, involving 1 or 2 covariates, and let the algorithm search more complex models later.

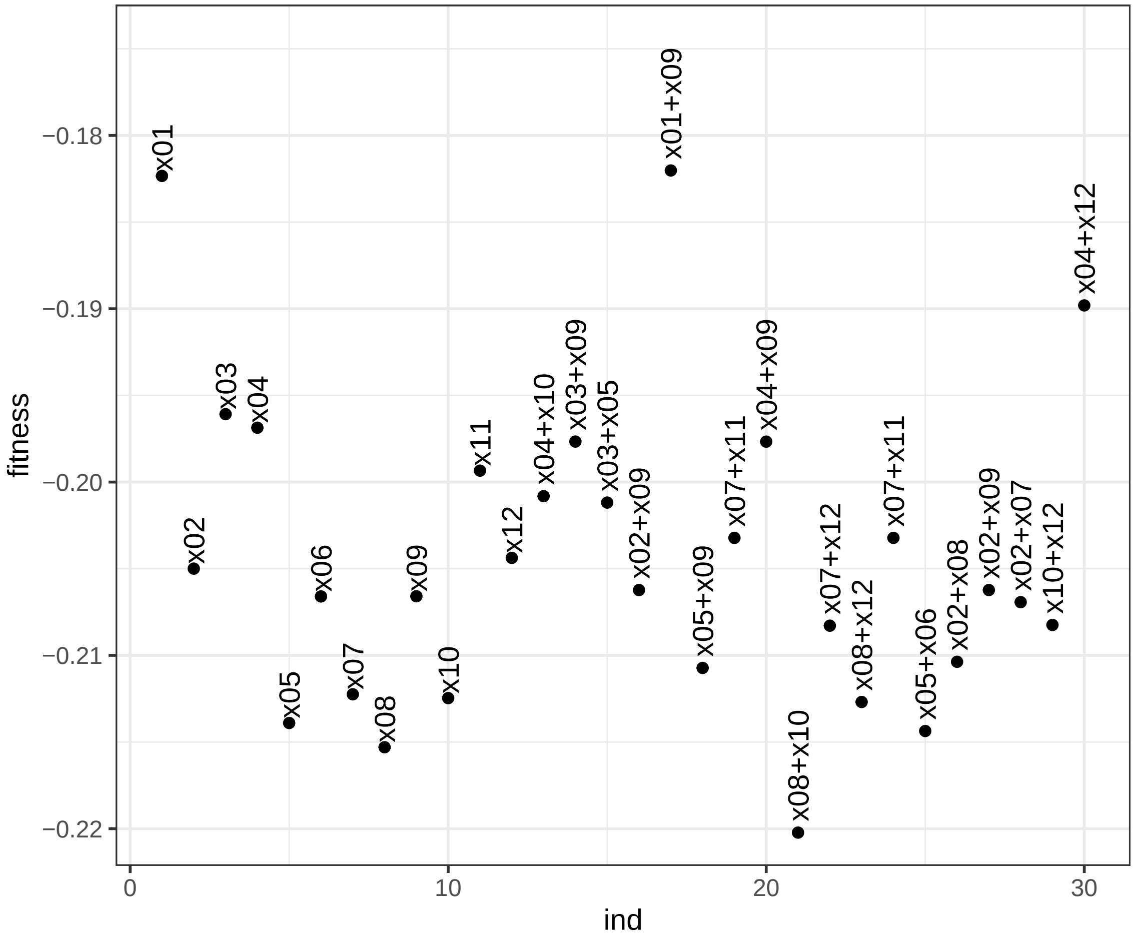
```
popSize <- max(30, length(envvars)) # number of individuals

PARS <- seq(envvars)*0
startPop <- matrix(0, ncol = length(PARS), nrow = popSize)
diag(startPop[seq(PARS),]) <- 1
if(popSize > length(PARS)){
  hit <- seq(nrow(startPop))[-seq(PARS)]
  startPop[hit, ] <- t(apply(startPop[hit, ], 1,
    FUN = function(x){x[sample(seq(PARS), size = 2)] <- 1; x}))
}

tmp <- startPop
tmp[] <- c("off", "on")[c(tmp)+1]
df <- as.data.frame(tmp)
names(df) <- envvars
df$ind <- seq(nrow(df))
```

```
df_long <- df %>%
  pivot_longer(cols = -ind, names_to = "variable", values_to = "value")

ggplot(df_long, aes(x = variable, y = ind, fill = factor(value))) +
  geom_tile(colour = 1) +
  scale_fill_manual(name = "Gene state",
    values = c("black", "white"), breaks = c("on", "off")) +
  scale_y_reverse(expand = c(0, 0)) +
  scale_x_discrete(expand = c(0, 0)) +
  labs(x = "Genes (covariates)", y = "Individuals",
    title = "Starting population") +
  theme_bw()
```



Distribution of fitness for starting population:

```
tmp1 <- apply(startPop, 1, FUN = function(x){fitnessFun(x, data = dat)})
tmp2 <- apply(startPop, 1, FUN = function(x){paste(envvars[as.logical(x)], collapse = "+")})
tmp <- data.frame(ind = seq(tmp1), fitness = tmp1, covariates = tmp2)

ggplot(tmp) + aes(x = ind, y = fitness, label = covariates) +
  geom_point() +
  geom_text(angle = 90, hjust = -0.1, vjust = 0.5) +
  scale_y_continuous(expand = expansion(mult = c(0.05,0.25))) +
  theme_bw()
```



The fitness of the model with "x01" alone is already a clear standout of the single term models with high fitness (-MAPE).

7.2 Genetic algorithm

Next, we will provide this starting population to a genetic algorithm (`GA::ga()`) of type "binary", which will turn off/on the covariates, and test the resulting fitness. According to the vignette of the GA package:

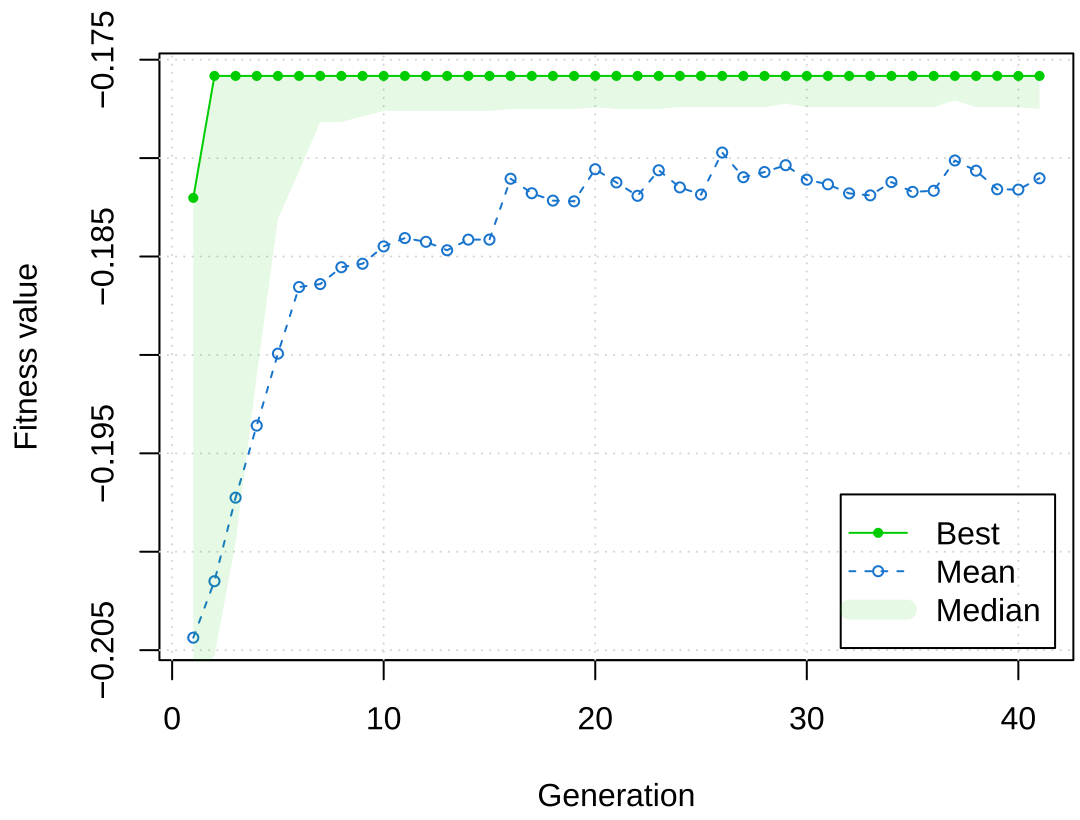
> *Genetic algorithms (GAs) are stochastic search algorithms inspired by the basic principles of biological evolution and natural selection. GAs simulate the evolution of living organisms, where the fittest individuals dominate over the weaker ones, by mimicking the biological mechanisms of evolution, such as selection, crossover and mutation.*

These aspects can be controlled by the `ga` function with specific arguments, e.g. the probability of gene crossover (`pcrossover`), the probability of mutation (`pmutation`), etc. Evolutionary settings will affect how the parameter space is searched – e.g. high mutation rates will be more likely to avoid getting trapped in local minima, while low mutation rates will provide a more refined search.

Note that genetic algorithms can also be useful for fitting other types of models (e.g. coefficient estimation), and are especially helpful in cases where the parameter space has local minima.

In the following example, a few non-standard options have been added: 1. A `postFitness` function is defined to maintain a record of all gene configurations tested, as well as their associated fitness, in order to visualize the full distribution of fitness by model complexity, and 2. Use of *memoisation*, which allows the algorithm to remember past evaluations of a particular model, so that fitness need not be recalculated in future evaluations. This can greatly reduce computation time for “binary-type” GA models.

```r
# extra diagnostic data object to save all generations
postfit <- function(object, ...){
  pop <- as.data.frame(object@population)
  names(pop) <- paste0("par", seq(ncol(pop)))
  pop$fitness <- object@fitness
  pop$iter <- object@iter
  # update info
  if(!exists(".pop", envir = globalenv()))
    assign(".pop", NULL, envir = globalenv())
  .pop <- get(".pop", envir = globalenv())
  assign(".pop", rbind(.pop, pop), envir = globalenv())
  # output the input ga object (this is needed!)
  object
}

mfitnessFun <- memoise(fitnessFun) # a memoised version of the fitness function
.pop <- NULL # make sure diagnostic output is empty
t1 <- Sys.time()
ga.fit <- ga(
  type = "binary", # for covariate selection
  fitness = mfitnessFun, # fitness function
  nBits = length(envvars), # number of genes (covariates)
  suggestions = startPop, # starting population suggestions
  popSize = popSize, # population size
  pcrossover = 0.8, # probability of gene crossover
  pmutation = 0.2,  # probability of mutation
  elitism = popSize*0.1, # number of best fitness individuals to survive at each generation
  parallel = FALSE, # set to TRUE for parallel computing (memoisation may be less effective)
  maxiter = 100, # max number of generations
  run = 40, # the max number of generations without any improvement
  monitor = FALSE, # set to TRUE to monitor fitness evolution
  data = dat, # input data
  seed = 1, # for reproducibility
  postFitness = postfit # extra diagnostics function
)
t2 <- Sys.time()
# t2 - t1 # elapsed time
tmp <- forget(mfitnessFun) # to stop memoisation

plot(ga.fit) # plot fitness development
```



Note that the best fitness model is identified relatively quickly, while the median and mean population fitness increases over time.

7.3 Pareto front

We can now visualize the resulting fitness values and identify the best models by complexity. Of these best models, those that also show increased fitness in comparison to the neighboring simpler model (i.e. one fewer covariate) are determined to lie on the Pareto front (dashed line in figure below).

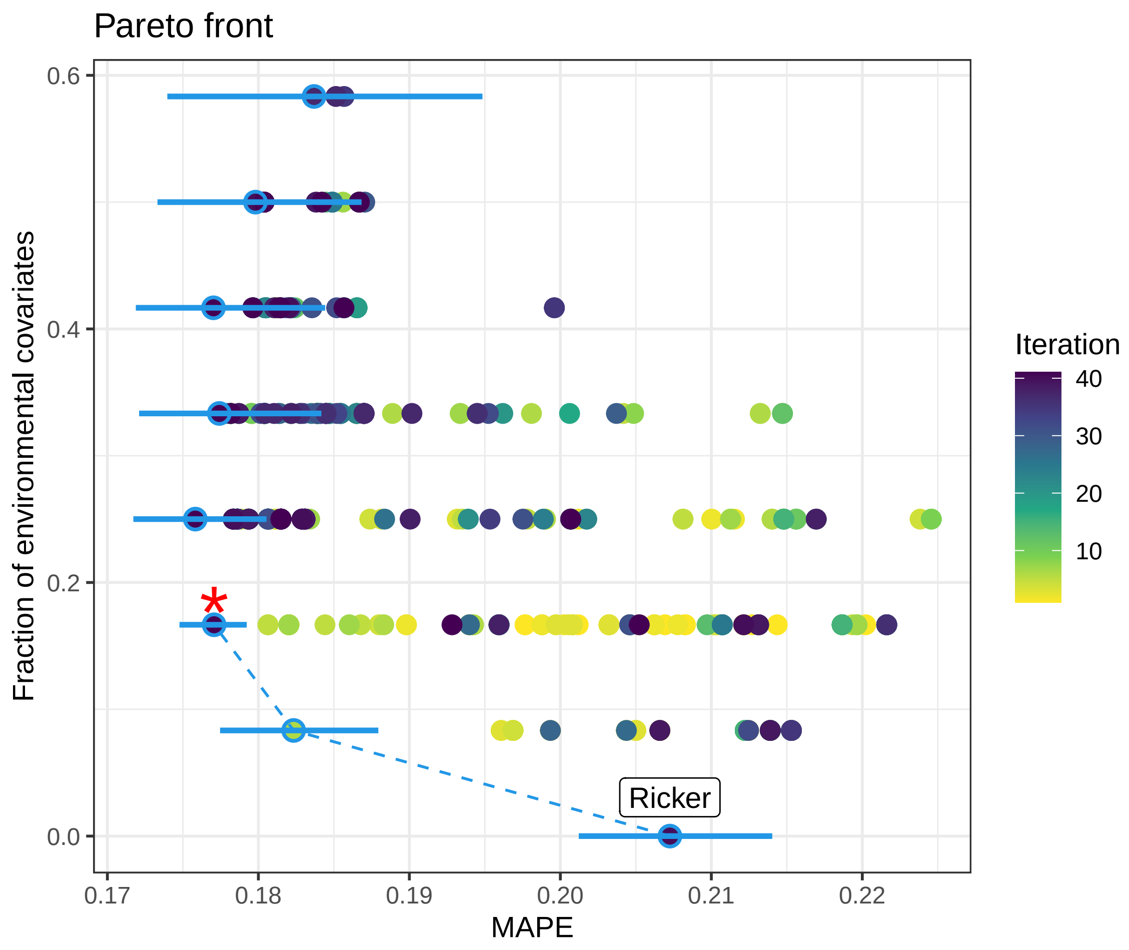
A general rule-of-thumb, the *1 Standard Error Rule* (Hastie, Tibshirani, and Friedman 2017), states that one shouold select the simplest model within one standard error of the best performance estimate. The idea is that if a simpler model performs nearly as well as a more complex one (within the margin of error), the simpler model is preferable.

In the repeated k-fold approach here, we instead use the distribution of error metric values across permutations as a proxy for this uncertainty. Only when a more complex model has a fitness value larger than the upper quantile (95%) of the simpler model, is it considered part of the Pareto front.

```r
.pop$nVar <- rowSums(.pop[,seq(envvars)])
.pop$fracVar <- .pop$nVar / length(envvars)
names(.pop)[seq(envvars)] <- envvars

# pareto front
agg <- aggregate(fitness ~ fracVar, data = .pop, FUN = max)
parFront <- merge(x = agg, y = .pop, all.x = T)
parFront <- unique(parFront[,c(envvars, "nVar", "fracVar", "fitness")])
parFront <- parFront[order(parFront$nVar),]

# determine distribution of fitness values on pareto front (by permutation)
parFront$lwr <- parFront$upr <- NaN
for(i in seq(nrow(parFront))){
  genes.i <- parFront[i,envvars]
  tmp <- fitnessFun(genes = genes.i, data = dat, fitnessOnly = F)
  parFront$lwr[i] <- tmp$lwr
  parFront$upr[i] <- tmp$upr
}

# check if model fitness is greater than the upper bound of the simpler model
better <- parFront$fitness > c(-Inf, parFront$upr[-nrow(parFront)])
parFront$best <- FALSE
hit <- max(which(better))
parFront$best[hit] <- TRUE

# get formula of best model
best <- subset(parFront, best)[envvars]
best <- paste(c(fixed_variables, names(best)[best == 1]), collapse = " + ")

# return
parFront
```

```
##     x01 x02 x03 x04 x05 x06 x07 x08 x09 x10 x11 x12 nVar    fracVar    fitness
## 1     0   0   0   0   0   0   0   0   0   0   0   0    0 0.00000000 -0.2072554
## 18    1   0   0   0   0   0   0   0   0   0   0   0    1 0.08333333 -0.1823366
## 23    1   0   1   0   0   0   0   0   0   0   0   0    2 0.16666667 -0.1770679
## 263   1   0   1   0   0   0   0   0   1   0   0   0    3 0.25000000 -0.1758235
## 388   1   0   1   0   0   0   0   0   1   0   1   0    4 0.33333333 -0.1774116
## 550   1   0   1   0   0   0   0   0   1   0   1   1    5 0.41666667 -0.1770245
## 587   1   0   1   0   0   0   1   0   1   0   1   1    6 0.50000000 -0.1798065
## 588   1   1   1   0   0   0   0   0   1   1   1   1    7 0.58333333 -0.1836888
##            upr        lwr  best
## 1   -0.2012205 -0.2140358 FALSE
## 18  -0.1774628 -0.1879445 FALSE
## 23  -0.1747733 -0.1792292  TRUE
## 263 -0.1717209 -0.1805370 FALSE
## 388 -0.1721005 -0.1841716 FALSE
## 550 -0.1718822 -0.1844287 FALSE
## 587 -0.1733157 -0.1868339 FALSE
## 588 -0.1739703 -0.1948410 FALSE
```

```r
ggplot(.pop) + aes(x = -fitness, y = fracVar, color = iter) +
  geom_point(size = 3) +
  geom_point(data = parFront, pch = 1, size = 3, color = 4, stroke = 1) +
  scale_color_continuous(name = "Iteration", type = "viridis", direction = -1) +
  geom_path(data = parFront[better,], color = 4, linetype = 2) +
  geom_label(data = parFront[1,], color = 1, label = "Ricker",
    hjust = 0.5, vjust = -0.5) +
  geom_segment(data = parFront, aes(x = -lwr, xend = -upr, y = fracVar, yend = fracVar),
    color = 4, linewidth = 1) +
  geom_text(data = subset(parFront, best), color = "red", label = "*", size = 10, vjust = 0.25) +
  labs(x = "MAPE", y = "Fraction of environmental covariates", title = "Pareto front") +
  theme_bw()
```



Results:

- Best model is rec ~ offset(log(ssb)) + ssb + x01 + x03.
- We evaluated a population of 30 individuals for 41 generations, totaling 1230 fitness evaluations.
- Memoisation evaluated 163 unique models, saving us 87% of fitness evaluations.
- Only 163 model configurations out of a total possible 4095 were evaluated (4%).

In this case, the best model is indeed the correct model, but it is only marginally better than the one containing only the environmental covariate "x01". The three term model is not part of the Pareto front as its solution is within the margin of error from the simpler 2-term model.

8 Other cross validation considerations

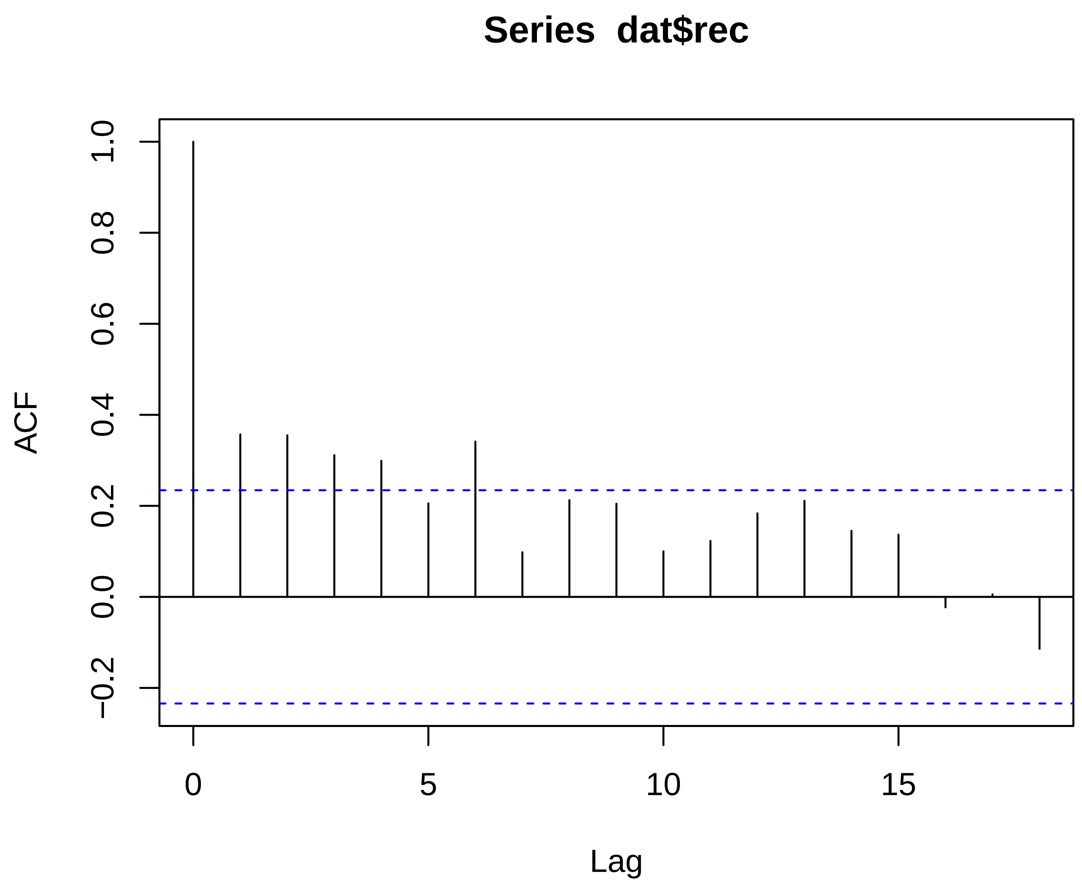
8.1 Flavor 2: Fold blocking

Until now, we have defined folds randomly. However, spatial or temporal autocorrelation may result in overly optimistic predictions if folds are not selected carefully.

For these cases, folds should be defined at scales that are larger than the residual autocorrelation.

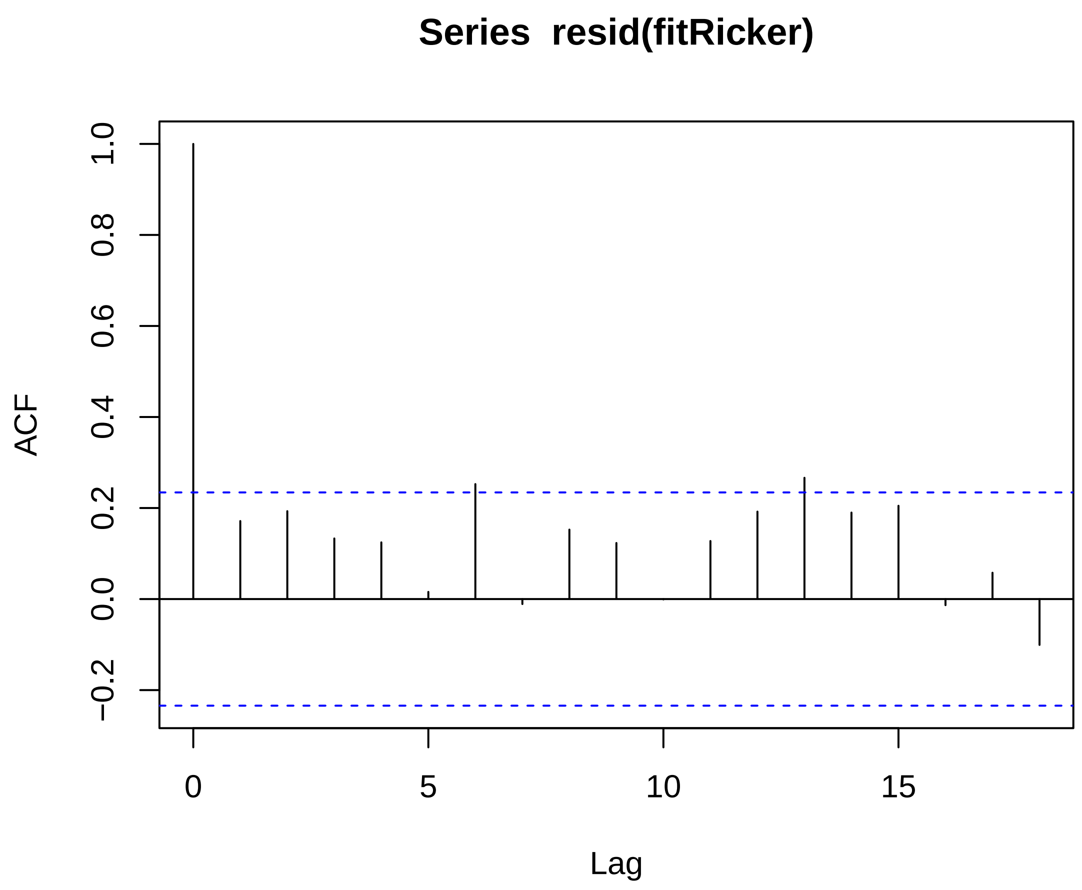
In our example data, there is autocorrelation evident in the response variable recruitment.

```r
acf(dat$rec)
```



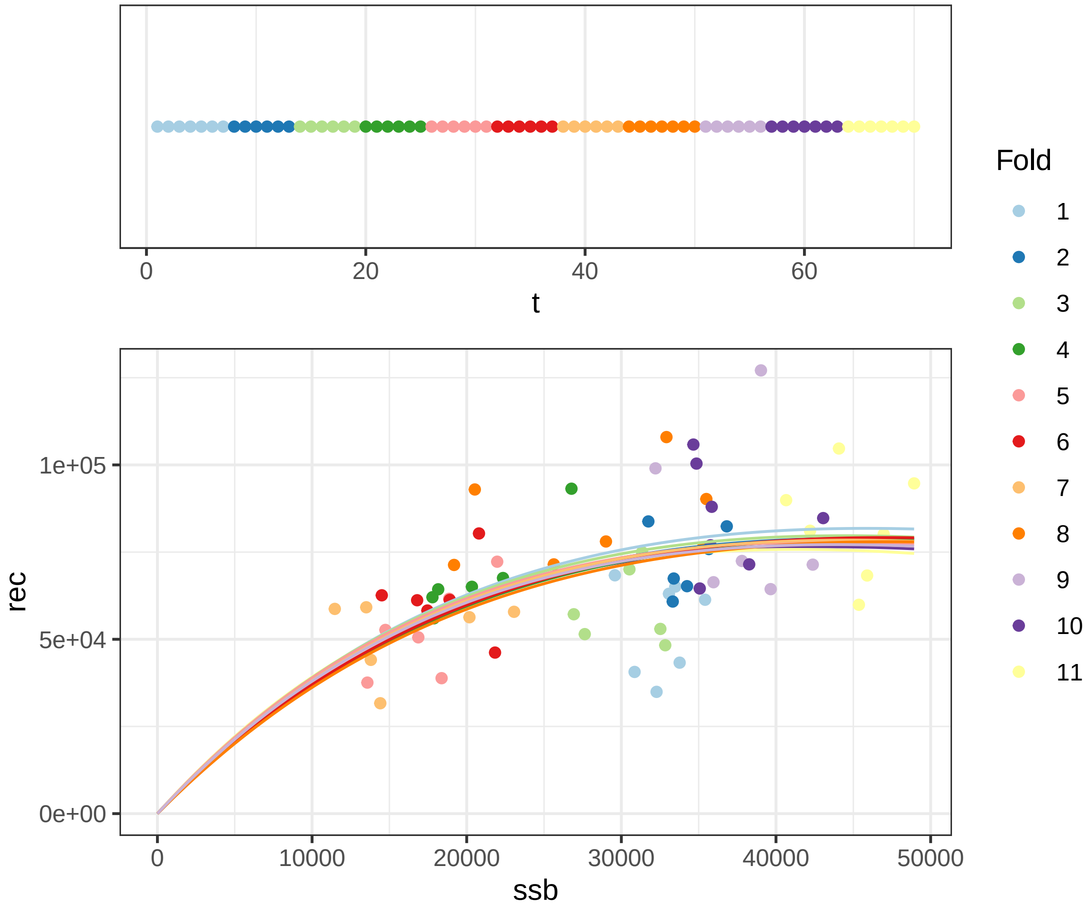
But, this is largely explained by the covariates considered (e.g. fluctuations in SSB), as indicated by an examination of autocorrelation in the residuals.

```r
fitRicker <- glm(rec ~ offset(log(ssb)) + ssb, data = dat, gaussian(link = "log"))
acf(resid(fitRicker))
```



When this is not the case, a blocking strategy could be used for the folds.

In the example below, a minimum block size is defined, corresponding to the scale where autocorrelation is no longer significant, and the data is divided accordingly.



There are indeed some folds of data that lie almost completely above/below the fitted Ricker models, but the above estimation of autocorrelation states that this was not significant.

8.2 Flavor 3: Hindcasting

When future prediction performance is the focus, you may want to select the validation data more carefully to prevent the model from "*seeing*" the future, which could potentially bias results.

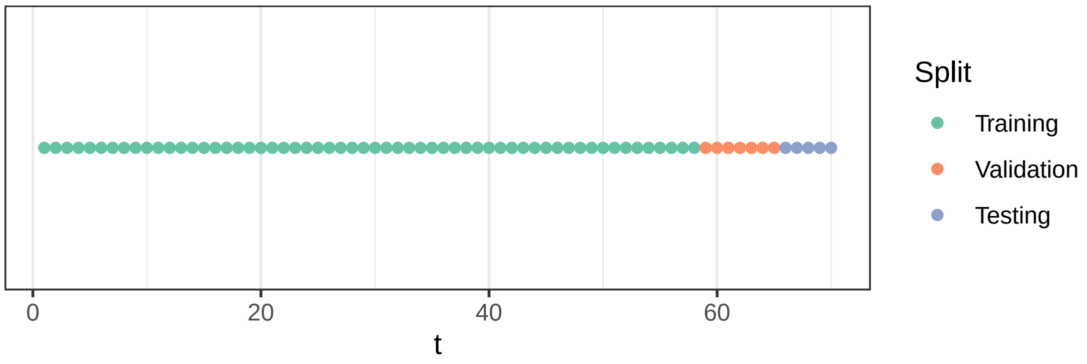
One strategy is to again fit the model to the ***training*** data set, but have two holdout sets *of future data*: 1. a ***validation*** set, used for model selection, and 2. a ***testing*** set for evaluating prediction performance after model selection.

In addition to addressing effects of autocorrelation, the set-up may also test for prediction performance when the time series is non-stationary, or subject to drift.

The amount of data used as holdouts (both for the validation and testing sets) will depend on the desired prediction performance. As with k-fold cross validation, a good rule-of-thumb would be to hold out 10–20% of the model selection data. This will hopefully deal with issues of overfitting, while also looking at future prediction performance in the validation set.

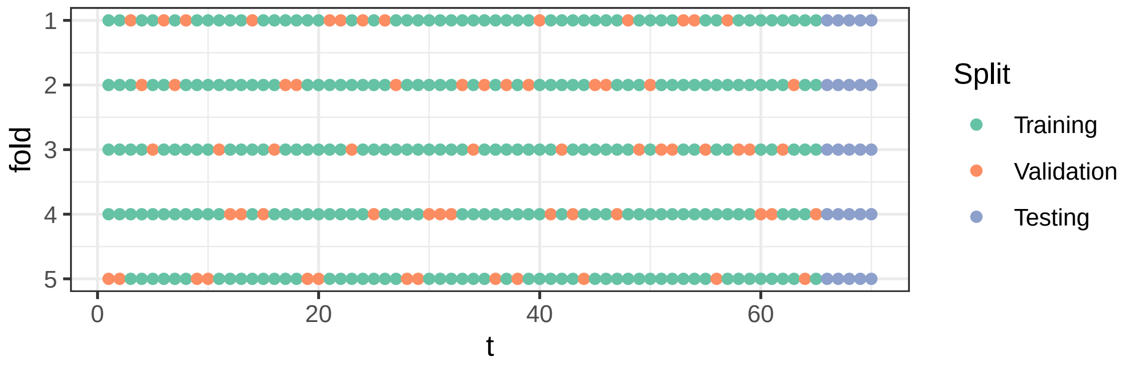
Another consideration is the forecast horizon of interest, which is a main consideration of the test data set size (e.g. a 3-year prediction might be of use for stock forecasts for defining catch advice levels). Thus, the length of both holdout sets should not differ too much, otherwise you may choose a model based on prediction at one time scale, while evaluating prediction performance at another.

Finally, the prediction performance on the test set is often compared to a *naive* prediction (e.g. mean of recent values). The *MASE* metric compares the ratio of mean absolute error of the selected model to that of the naive model (MAE_{mod}/MAE_{naive}), with a score < 1.0 being associated with a forecast model that is superior to the naive one.

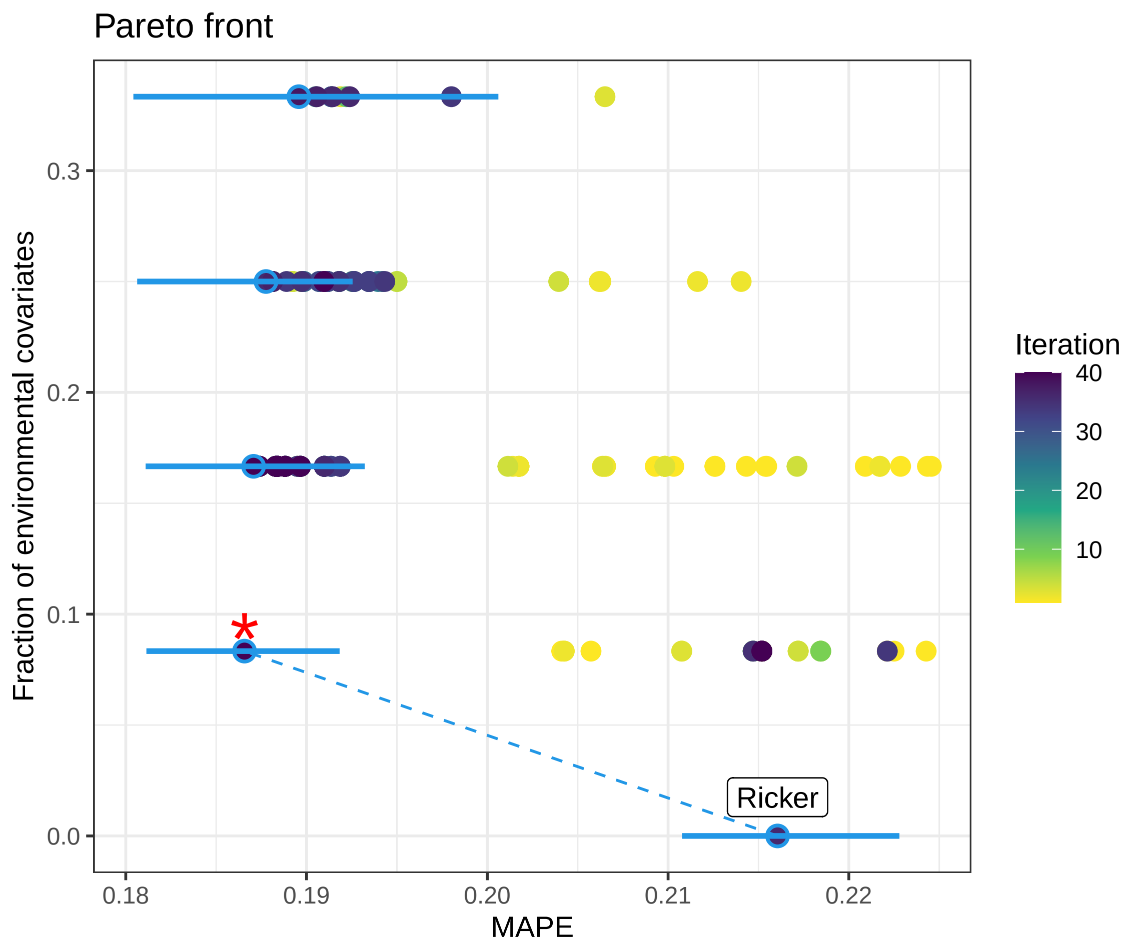


8.2.1 Model selection on training/validation data

In our example, since no strong autocorrelation was detected, lets try a k-fold approach for the model selection, leaving out the final 5 years of the data for MASE evaluation versus a naive model.



```r
# split data into training/validation & testing
nTest <- 5
datTrainValid <- subset(dat, t <= max(t)-nTest)
datTest <- subset(dat, t > max(t)-nTest)

mfitnessFun <- memoise(fitnessFun) # a memoised version of the fitness function
.pop <- NULL # make sure diagnostic output is empty
t1 <- Sys.time()
ga.fit <- ga(
  type = "binary", # for covariate selection
  fitness = mfitnessFun, # fitness function
  nBits = length(envvars), # number of genes (covariates)
  suggestions = startPop, # starting population suggestions
  popSize = popSize, # population size
  pcrossover = 0.8, # probability of gene crossover
  pmutation = 0.2,  # probability of mutation
  elitism = popSize*0.1, # number of best fitness individuals to survive at each generation
  parallel = FALSE, # set to TRUE for parallel computing (memoisation may be less effective)
  maxiter = 100, # max number of generations
  run = 40, # the max number of generations without any improvement
  monitor = FALSE, # set to TRUE to monitor fitness evolution
  data = datTrainValid, # input data *** ONLY TRAIN/VALID ***
  seed = 1, # for reproducibility
  postFitness = postfit # extra diagnostics function
)
t2 <- Sys.time()
# t2 - t1 # elapsed time
tmp <- forget(mfitnessFun) # to stop memoisation

.pop$nVar <- rowSums(.pop[,seq(envvars)])
.pop$fracVar <- .pop$nVar / length(envvars)
names(.pop)[seq(envvars)] <- envvars

# pareto front
agg <- aggregate(fitness ~ fracVar, data = .pop, FUN = max)
parFront <- merge(x = agg, y = .pop, all.x = T)
parFront <- unique(parFront[,c(envvars, "nVar", "fracVar", "fitness")])
parFront <- parFront[order(parFront$nVar),]

# determine distribution of fitness values on pareto front (by permutation)
parFront$lwr <- parFront$upr <- NaN
for(i in seq(nrow(parFront))){
  genes.i <- parFront[i,envvars]
  tmp <- fitnessFun(genes = genes.i, data = datTrainValid, fitnessOnly = F) # Only re-fit to training/v
  parFront$lwr[i] <- tmp$lwr
  parFront$upr[i] <- tmp$upr
}

# check if model fitness is greater than the upper bound of the simpler model
better <- parFront$fitness > c(-Inf, parFront$upr[-nrow(parFront)])
parFront$best <- FALSE
hit <- max(which(better))
parFront$best[hit] <- TRUE

# get formula of best model
best <- subset(parFront, best)[envvars]
best <- paste(c(fixed_variables, names(best)[best == 1]), collapse = " + ")

ggplot(.pop) + aes(x = -fitness, y = fracVar, color = iter) +
  geom_point(size = 3) +
  geom_point(data = parFront, pch = 1, size = 3, color = 4, stroke = 1) +
  scale_color_continuous(name = "Iteration", type = "viridis", direction = -1) +
  geom_path(data = parFront[better,], color = 4, linetype = 2) +
  geom_label(data = parFront[1,], color = 1, label = "Ricker",
    hjust = 0.5, vjust = -0.5) +
  geom_segment(data = parFront, aes(x = -lwr, xend = -upr, y = fracVar, yend = fracVar),
    color = 4, linewidth = 1) +
  geom_text(data = subset(parFront, best), color = "red", label = "*", size = 10, vjust = 0.25) +
  labs(x = "MAPE", y = "Fraction of environmental covariates", title = "Pareto front") +
  theme_bw()
```



8.2.2 Prediction of test data and comparison to naive model

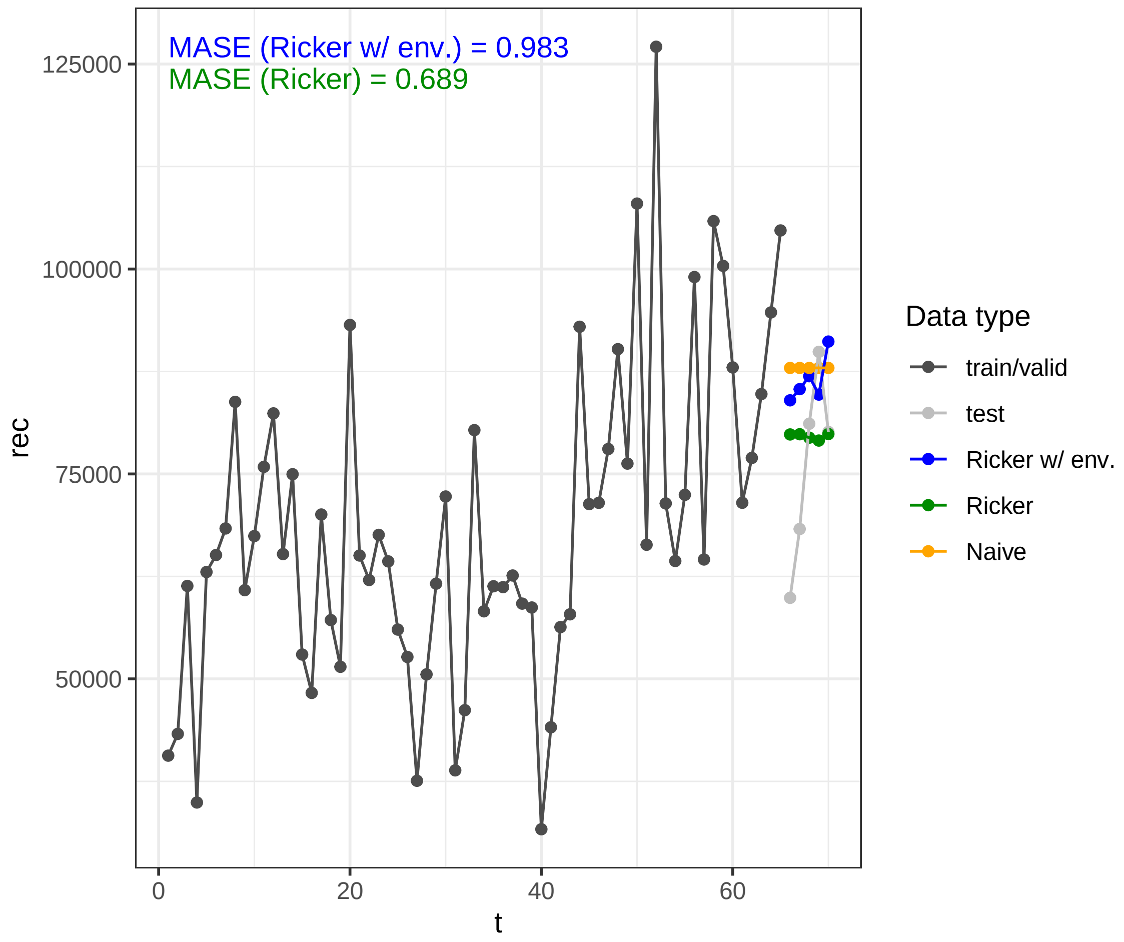
Note that the removal of the terminal 5 years already affected the definition of the best performing model, removing the term that was only marginally significant in the previous example ("x03").

The best selected model is rec ~ offset(log(ssb)) + ssb + x01, which we will then re-fit to the full training/validation data, and predict the testing holdout. The naive model prediction will be the geometric mean of recruitment from the last 10 years.

For comparison, the MASE of the simple Ricker, without environmental covariates, is also shown for comparison.

```r
# naive prediction (geometric mean of last 10 years)
yhat_naive <- rep(exp(mean(log(tail(datTrainValid$rec, 10)))), nrow(datTest))

# prediction best model
fmla <- best
fitBest <- update(fit, data = datTrainValid, # re-fit to train/valid data
  formula = as.formula(fmla, env = environment()))
yhat_mod <- predict(fitBest, newdata = datTest, type = "response")
MASE_mod <- mase(y = datTest$rec, yhat = yhat_mod, yhat_naive = yhat_naive)

# compare to non-env Ricker
fmla <- "rec ~ ssb + offset(log(ssb))"
```

```r
fitRicker <- update(fit, data = datTrainValid, # re-fit to train/valid data
  formula = as.formula(fmla, env = environment()))
yhat_ri <- predict(fitRicker, newdata = datTest, type = "response")
MASE_ri <- mase(y = datTest$rec, yhat = yhat_ri, yhat_naive = yhat_naive)

df1 <- dat[,c("t", "rec")]
df1$type = ifelse(df1$t %in% datTrainValid$t, "train/valid", "test")
df2 <- datTest[,c("t", "rec")]
df2$rec <- yhat_mod
df2$type = "Ricker w/ env."
df3 <- datTest[,c("t", "rec")]
df3$rec <- yhat_naive
df3$type = "Naive"
df4 <- datTest[,c("t", "rec")]
df4$rec <- yhat_ri
df4$type = "Ricker"

df <- rbind(df1, df2, df3, df4)
df$type <- factor(df$type, levels = c("train/valid", "test", "Ricker w/ env.", "Ricker", "Naive"))

ggplot(df) + aes(x = t, y = rec, colour = type) +
  geom_point() +
  geom_line() +
  annotate("text", x = min(df$t), y = max(df$rec),
    label = paste("MASE (Ricker w/ env.) =", round(MASE_mod, 3)),
    hjust = 0, color = "blue") +
  annotate("text", x = min(df$t), y = max(df$rec),
    label = paste("MASE (Ricker) =", round(MASE_ri, 3)),
    hjust = 0, vjust = 2, color = "green4") +
  scale_color_manual(name = "Data type",
    values = c("grey30", "grey", "blue", "green4", "orange")) +
  theme_bw()
```



The prediction performance is slightly better than the naive prediction (MASE < 1.0), but this might not be strong enough to warrant a change from the naive approach.

The simple Ricker was in fact much better, with a substantially lower MASE.

Also note that the stock recruitment models were given the correct future SSB and environmental covariates, which may not be possible (without some associated error in their predictions). A fair test of prediction performance should also take into account this uncertainty.

8.2.3 Sensitivity of prediction to data window

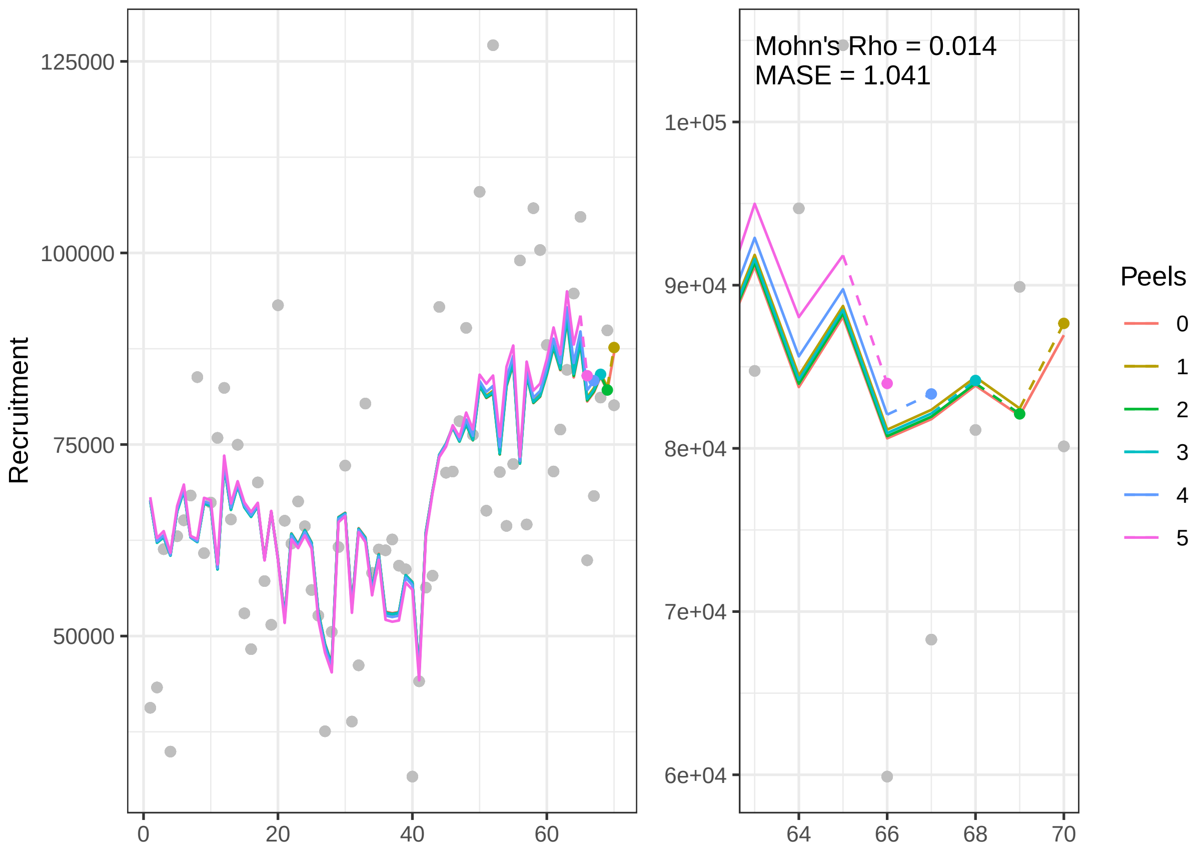
The final example will show a slightly different version of hindcasting that is slowly being adopted in the selection of stock assessment models. In particular, this approach also tests how sensitive the model fit is the to recent data.

The procedure involves the removal of recent data one year at a time, known as a "peel". This is done for several time steps into the past, followed by a refitting of the model and prediction of future values.

Two bits of insight can be gained: 1. whether retrospective bias exists in the fitted historical values – evaluated with a statistic called Mohn's Rho, and 2. the predictive performance on the future test values, which can again be assessed through a metric like MASE.

```r
nPeel <- 5
maxForecast <- 1
res <- vector("list", nPeel+1)
for(i in c(0,seq(nPeel))){
  # remove i years
  datTrain.i <- subset(dat, subset = t <= (max(t) - i))
  datTest.i <- subset(dat, subset = t > (max(t) - i))

  # refit model to data
  fmla <- best
  fit.i <- update(fit, data = datTrain.i,
    formula = as.formula(fmla, env = environment()))

  # predictions
  datTrain.i$type <- "Train"
  datTrain.i$forecast <- NaN
  datTrain.i$forecast[length(datTrain.i$forecast)] <- 0

  if(i>0){
    datTest.i$type <- "Test"
    datTest.i$forecast <- seq(nrow(datTest.i))
  }

  datPred <- rbind(datTrain.i, datTest.i)
  datPred$peel <- i
  datPred$yhat_mod <- predict(fit.i, newdata = datPred, type = "response")
  datPred$yhat_naive <- rep(exp(mean(log(tail(datTrain.i$rec, 10)))), nrow(datPred))

  # record
  res[[i+1]] <- datPred[,c("t", "type", "peel", "forecast", "rec", "yhat_mod", "yhat_naive")]
}
res <- do.call("rbind", res)
res$peel <- factor(res$peel)

dfTrain <- subset(res, type == "Train")
dfTest <- subset(res, type == "Test")
dfForecast <- subset(res, forecast <= maxForecast)
dfForecast2 <- subset(dfForecast, forecast>0)

# calculate MASE
MASE_mod <- with(dfForecast2, mase(y = rec, yhat = yhat_mod, yhat_naive = yhat_naive))

# calculate Mohn's rho
L <- split(unique(res$peel), unique(res$peel))
tmp <- lapply(L, FUN = function(x){
  subset(res, peel == x & t > (max(t)-nPeel-1) & forecast %in% c(NaN, 0))[,c("t", "peel", "yhat_mod")]
})
tmp <- do.call("rbind", tmp)
tmp2 <- tmp |> pivot_wider(
  names_from = peel,
  values_from = yhat_mod,
  names_prefix = "peel_"
)
tmp3 <- as.data.frame(tmp2[,-1])
rownames(tmp3) <- tmp2$t
RHO_mod <- icesAdvice::mohn(tmp3)
```

```r
p1 <- ggplot(data = dfTrain) + aes(x = t, y = yhat_mod, color = peel, group = peel) +
  geom_point(data = res, mapping = aes(y = rec), color = "grey") +
  geom_line() +
  geom_line(data = dfForecast, linetype = 2) +
  geom_point(data = dfForecast2, show.legend = F) +
  labs(x = NULL, y = "Recruitment", color = "Peels") +
  theme_bw()

p2 <- ggplot(data = dfTrain) + aes(x = t, y = yhat_mod, color = peel, group = peel) +
  geom_point(data = res, mapping = aes(y = rec), color = "grey") +
  geom_line() +
  geom_line(data = dfForecast, linetype = 2) +
  geom_point(data = dfForecast2, show.legend = F) +
  labs(x = NULL, y = "Recruitment", color = "Peels") +
  coord_cartesian(xlim = range(dfForecast$t)+c(-2, 0),
    ylim = range(subset(res, t >= min(dfForecast$t)-2)[,c("rec", "yhat_mod")])) +
  annotate("text", x = min(dfForecast$t)-2, y = max(dfForecast$rec),
    label = paste("Mohn's Rho =", round(RHO_mod, 3)),
    hjust = 0, color = "black") +
  annotate("text", x = min(dfForecast$t)-2, y = max(dfForecast$rec),
    label = paste("MASE =", round(MASE_mod, 3)),
    hjust = 0, vjust = 2, color = "black") +
  theme_bw()

layout = "
AAABB
AAABB
"

p <- (p1 | p2) + plot_layout(design = layout, axes = "collect", guides = "collect") &
  theme_bw()
print(p)
```



9 Concluding remarks

During the SEAwise project, similar results on the predictive power of environmentally-mediated models illustrated important considerations and trade-offs for their use in management strategy evaluations:

- Their use in the **management procedure**, e.g. as part of a harvest control rule involving short-term forecasts, requires rigorous testing with realistic information on future environmental conditions. Clear benefits over naive approaches may be rare.
- Their use in the **operating model**, e.g. as part of the biological dynamics, can provide important tests to the robustness of a given management procedure over the long-term. The ability of a given harvest control rule to produce favorable outcomes across several alternate (likely) operating models provides additional confidence in the procedure.

10 Software Versions

- R version 4.4.2 (2024-10-31 ucrt)
- corrplot: 0.95
- doRNG: 1.8.6
- dplyr: 1.1.4
- forecast: 8.23.0
- GA: 3.2.4
- ggplot2: 3.5.1
- glmmTMB: 1.1.10
- icesAdvice: 2.1.1

- knitr: 1.49
- memoise: 2.0.1
- MuMIn: 1.48.4
- parallel: 4.4.2
- patchwork: 1.3.0
- tidyr: 1.3.1
- **Compiled**: 2025-May-19

11 Author

Marc Taylor. Thünen Institute of Sea Fisheries, Marine Living Resources Unit, Herwigstraße 31, 27572 Bremerhaven, Germany. https://www.thuenen.de/en/sf/

References

Hastie, Trevor, Robert Tibshirani, and Jerome H. Friedman. 2017. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Second edition. Springer Series in Statistics. New York, NY: Springer.

Iles, T. C., and R. J. H. Beverton. 1998. “Stock, Recruitment and Moderating Processes in Flatfish.” *Journal of Sea Research*, Proceedings of the Third International Symposium on Flatfish Ecology, Part II, 39 (1–2): 41–55. https://doi.org/10.1016/S1385-1101(97)00022-1.

Kühn, Bernhard, Marc H. Taylor, and Alex Kempf. 2021. “Using Machine Learning to Link Spatiotemporal Information to Biological Processes in the Ocean: A Case Study for North Sea Cod Recruitment.” *Marine Ecology Progress Series* 664 (April): 1–22. https://doi.org/10.3354/meps13689.

Levi, Dino, M. G. Andreoli, A. Bonanno, Fabio Fiorentino, G. Garofalo, S. Mazzola, Giacomo Norrito, et al. 2003. “Embedding Sea Surface Temperature Anomalies into the Stock Recruitment Relationship of Red Mullet (Mullus Barbatus L. 1758) in the Strait of Sicily.” *Scientia Marina* 67 (S1): 259–68. http://scientiamarina.revistas.csic.es/index.php/scientiamarina/article/viewArticle/525.