Appendix S8: Computing confidence distributions for model-averaged predictions

Dormann et al.

March 27, 2018

1 Introduction

As the main document outlines, there are five different ways to compute confidence distributions for model-averaged predictions. They differ in the assumptions they make, and the degree to which these assumptions are likely to be violated.

Propagation Error-propagartion based confidence distributions build on the equations for Gaussian error propagation (a.k.a. uncertainty propagation), with bias estimates added. This equation is 'derived' in the paper, but essentially it is the combination of the rules for error propagation as found, e.g., on wikipedia,[1] and the equation defining MSE as sum of squared bias and variance (see main text). While this equation doesn't assume that the outcome is a normal distribution, we only have the first and second moment of the confidence distribution thus derived. Hence, we use a normal distribution to represent it.

Buckland Equation proposed by Buckland et al. (1997, see main text for citation and equation). This is a simplification of the first approach, achieved by assuming perfect correlation between predictions. While unlikely to be true, this is a conservative assumption, leading to wider confidence distributions and a simpler equation. As for the first case, the outcome is not necessarily a normal distribution, but we only have mean and variance for this distribution and hence represent it as a normal.
This approach's theoretical justification has been criticised (see main text), and to our knowledge no comparison with simulated data is available, so we do not know how generalisable our (supportive) findings for this approach are.

Convolution The (weighted) sum of *independent* distributions is mathematically equivalent to a (weighted) convolution. The convolution of normal distributions is again a normal distribution (and so for many other simple cases). Since the assumption of independent distributions is virtually almost violated, this approach will lead to confidence distributions that are (far) too narrow. **Strongly discouraged.**

Mixing Mixing describes the drawing from independent distributions according to their weight. It is essentially a weighted overlay of the confidence distributions of the single models. The assumption likely to be violated is that of independence of model predictions. For positively correlated model predictions, this method should thus lead to overly conservative estimates. It is the only approach here that (potentially) yields multi-model confidence distributions.

Full model When a full model is available, it is the preferred option for prediction of several statisticians (e.g. Harrell 2001). While potentially overparameterised, it does not suffer from model selection bias and distorted confidence distributions for its prediction. The full model can easily be build from various GLMs, but not across model types (e.g. a randomForest and a GAM). So, for mixing modelling algorithms, no obvious full model exists.

[1] `https://en.wikipedia.org/wiki/Propagation_of_uncertainty#Example_formulas`

2 Computing confidence distributions for four linear model's averaged prediction

We start by simulating a small data set, fitting four inadequate models to it, use equal weights and construct the model-averaged prediction's confidence distributions according to the five methods above. (Thus, coverage estimation is conditional on equally-weighted models and will hence change with the actual weighting scheme employed.)

```
set.seed(2)
N <- 70 # number of data points
Y <- rnorm(N, sd=1)
X <- as.data.frame(matrix(NA, ncol=20, nrow=N))
for (i in 1:20) X[,i] <- runif(N) # 20 uncorrelated predictors
colnames(X) <- paste0("X", 1:20)
X <- cbind(Y, as.data.frame(scale(X, scale=F))) # center all predictors
rm(Y)

fm1 <- lm(Y ~ 1, data=X)
fm2 <- lm(Y ~ ., data=X[, 1:6])
fm3 <- lm(Y ~ ., data=X[, c(1, 7:13)])
fm4 <- lm(Y ~ ., data=X[, c(1, 14:21)])

# compute AIC-weights (as those were used by Fletcher/Turek):
AICs <- sapply(list(fm1, fm2, fm3, fm4), AIC)
round(w <- exp(0.5*AICs) / sum(exp(0.5*AICs)), 3)

## [1] 0.002 0.006 0.003 0.989

# for the sake of argument, let's assume equal weights for all four models:
w <- rep(0.25, 4)
```

Now we predict, with each model, to a single new data point.

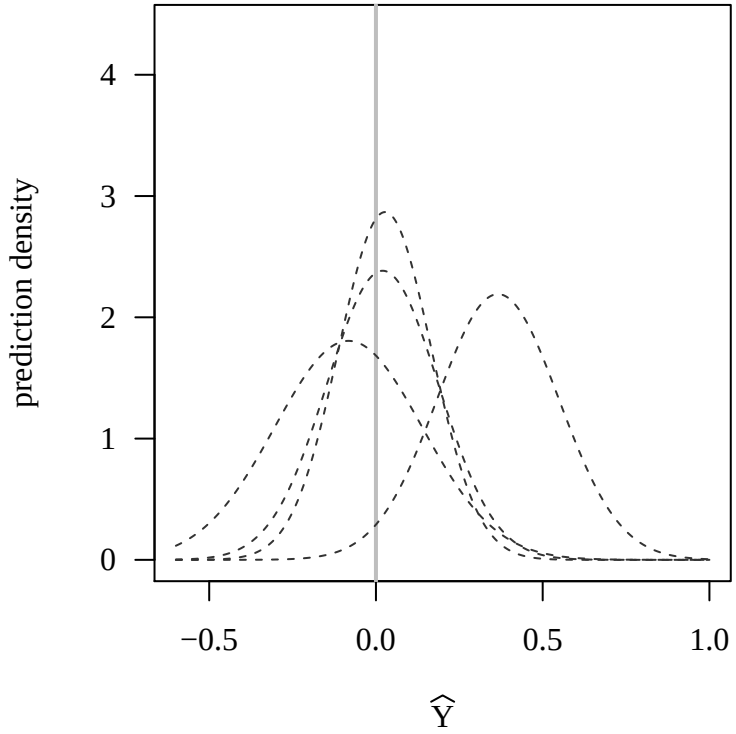
```
newpoint <- as.data.frame(t(rep(.1, 20)))
colnames(newpoint) <- paste0("X", 1:20)
(preds <- lapply(list(fm1, fm2, fm3, fm4), predict, newdata=newpoint, se.fit=T))

## [[1]]
## [[1]]$fit
##         1
## 0.0271926
##
## [[1]]$se.fit
## [1] 0.1391161
##
## [[1]]$df
## [1] 69
##
## [[1]]$residual.scale
## [1] 1.163929
##
##
```

```
## [[2]]
## [[2]]$fit
##          1
## 0.02001059
##
## [[2]]$se.fit
## [1] 0.1673532
##
## [[2]]$df
## [1] 64
##
## [[2]]$residual.scale
## [1] 1.141219
##
##
## [[3]]
## [[3]]$fit
##        1
## 0.365974
##
## [[3]]$se.fit
## [1] 0.1820847
##
## [[3]]$df
## [1] 62
##
## [[3]]$residual.scale
## [1] 1.117693
##
##
## [[4]]
## [[4]]$fit
##           1
## -0.08250375
##
## [[4]]$se.fit
## [1] 0.2209388
##
## [[4]]$df
## [1] 61
##
## [[4]]$residual.scale
## [1] 1.205489
```

We start by plotting these four distributions, over which we shall next be averaging.

```
cols <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

par(mar=c(4,4,1,1))
plot(1, 1, xlim=c(-0.6, 1), ylim=c(0, 4.4), type="n", las=1, xlab=expression(widehat(Y)),
     ylab="prediction density")
```

```
abline(v=0, lwd=2, col="grey") # there is no effect of any X!
for (i in 1:4) curve(dnorm(x, preds[[i]]$fit, preds[[i]]$se.fit), add=T, n=501, col="grey20", lty=2)
```



Now we come to the first method.

2.1 Propagation

For error propagation, we need to estimate several parameters, in particular bias and covariance between predictions of the different models.

From the main text, the relevant equation is eqn 5, where we replace the correlation matrix-expression with the equivalent covariance expression (eqn 4):

$$\mathrm{MSE}(\widetilde{Y}) = \left(\sum_{m=1}^{M} w_m \left(E(\widehat{Y}_m) - y^* \right) \right)^2 + \sum_{m=1}^{M} \sum_{n=1}^{M} w_m w_n \mathrm{cov}(\widehat{Y}_m, \widehat{Y}_n) \tag{1}$$

First, we bootstrap the analysis by the four models:

```
## 1. bootstrapping to compute elements of equation "eqn:varest":
if (!exists("bspred.mat")){
        Nbs <- 500
        bspred.mat <- matrix(NA, nrow=Nbs, ncol=4)
        for (i in 1:Nbs){
                Xbs <- X[sample(nrow(X), replace=T), ]
                fm1bs <- lm(Y ~ 1, data=Xbs)
                fm2bs <- lm(Y ~ ., data=Xbs[, 1:6])
                fm3bs <- lm(Y ~ ., data=Xbs[, c(1, 7:13)])
                fm4bs <- lm(Y ~ ., data=Xbs[, c(1, 14:21)])
                bspred.mat[i,] <- sapply(list(fm1bs, fm2bs, fm3bs, fm4bs), predict,
                                         newdata=newpoint)
        }
}
```

The question then is, what do we compute from the bootstrap, and what from the original data? We can compare the estimates from the bootstrap with those from the data. Let's see; bias from original data (note that the weights enter here!):

```
predsMeans <- c(preds[[1]]$fit, preds[[2]]$fit, preds[[3]]$fit, preds[[4]]$fit)
(gamma <- predsMeans - predsMeans %*% w )

## [1] -0.05547577 -0.06265777  0.28330566 -0.16517211
```

Note that the last term (`predsMeans %*% w`) is $\widetilde{Y}$.

Alternatively, we compute model misspecification bias as deviation of each bootstrapped prediction from the mean of that bootstrap:

```
biasBS <- (bspred.mat - matrix(bspred.mat %*% w, ncol=4, nrow=nrow(bspred.mat)))
(gamma_m <- colMeans(biasBS))   # mean for each run of the bootstrap

## [1] -0.05531237 -0.05942203  0.28163100 -0.16689661

sum(w*gamma_m)

## [1] 3.469447e-18
```

In this case, the difference between computation from data and from bootstrap is negligible. For the estimation we thus do not need the bootstrap.

Just to make this point explicit again: The "bias" estimated here is of course *not* the correct bias of the models and the model average. We *assume* that the model average is unbiased (see also main text), and hence use the individual models to compute what we hope is a good estimate of the truth. In real life, we have no way of knowing how close to the actual truth this estimate is. (In our simulation we do, and we see that the averaged estimate is indeed *not* the truth.)

Next, we need to estimate the variance of the prediction. Again, we can use the predictions from the linear model (their squared standard error, to be precise), or the variance of the bootstrap.

```
# estimate variance of prediction from prediction se:
(predsVars <- c(preds[[1]]$se.fit, preds[[2]]$se.fit, preds[[3]]$se.fit, preds[[4]]$se.fit)^2)

## [1] 0.01935329 0.02800708 0.03315485 0.04881393

# now from bootstrap:
(varY_m <- apply(bspred.mat, 2, var)) # from across the bootstraps

## [1] 0.01981484 0.04109116 0.03655210 0.06207153
```

Again, estimates are similar, but now differences are larger than for the estimate of $\widetilde{Y}$. We expect for small data sets bootstrap estimates to be less biased than their analytical counterparts. Also, we suggest using the bootstrap in general, as non-parametric approaches do not provide analytical estimates for the standard error of predictions.

The final parameters we need are the covariances (computed in either case from the bootstraps).

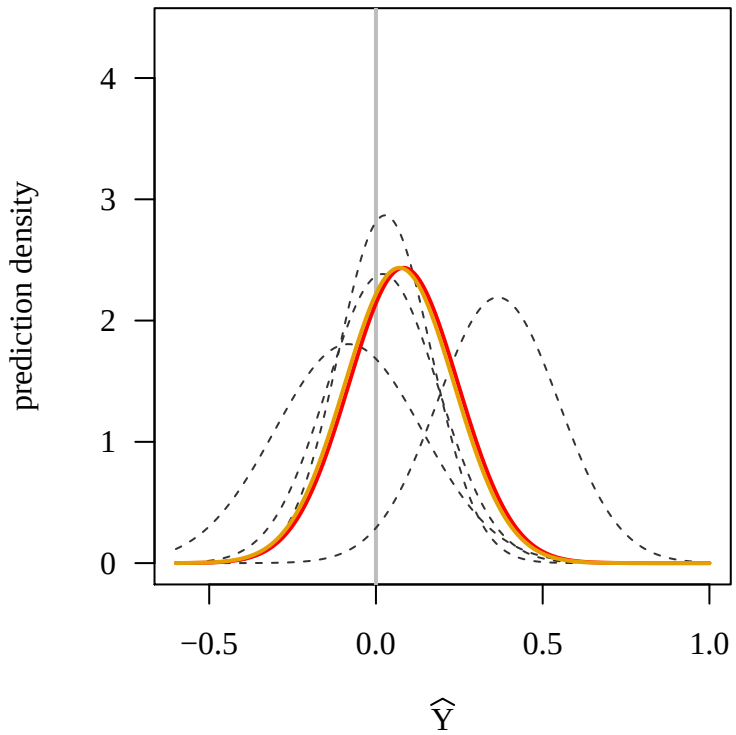
```
(COV <- cov(bspred.mat))

##            [,1]       [,2]       [,3]       [,4]
## [1,] 0.01981484 0.02203224 0.01765071 0.02415945
## [2,] 0.02203224 0.04109116 0.02156726 0.02901665
## [3,] 0.01765071 0.02156726 0.03655210 0.02054667
## [4,] 0.02415945 0.02901665 0.02054667 0.06207153

# the weights-weights matrix is:
ww <- tcrossprod(w)
```

With these ingredients, we can now compute the variance of model-averaged predictions according to uncertainty propagation.

```
# eqn 4 with data-based estimates:
(MSE <- (sum(w*gamma))^2 + sum(ww * COV) )

## [1] 0.02684222

# eqn 4, using gamma_m from bootstrapped data:
(MSE_m <- (sum(w*gamma_m))^2 + sum(ww * COV) )

## [1] 0.02684222
```

So, overall these two estimates (one based on the observed data, one entirely on the bootstraps) are about the same, simply because the bias isn't very different and small relative to the covariances. We shall briefly plot them for comparison, and then in the following use the wider variance, based on the bootstrap (in orange).

```
# add this to the plot:
par(mar=c(4,4,1,1))
plot(1, 1, xlim=c(-0.6, 1), ylim=c(0, 4.4), type="n", las=1, xlab=expression(widehat(Y)),
     ylab="prediction density")
abline(v=0, lwd=2, col="grey") # there is no effect of any X!
for (i in 1:4) curve(dnorm(x, preds[[i]]$fit, preds[[i]]$se.fit), add=T, n=501, col="grey20", lty=2)
curve(dnorm(x, mean=predsMeans %*% w, sd=sqrt(MSE)), add=T, n=501, col="red", lwd=2)
curve(dnorm(x, mean=mean(bspred.mat %*% w), sd=sqrt(MSE_m)), add=T, n=501, col=cols[1], lwd=2)
```



Thus, we proceed in general as follows: run bootstraps to compute the covariances. Compute weighted average from model predictions directly. Then equation 5 (main text) is used to compute the MSE. (Note that its first term is actually 0, as we use exactly this weighted sum of averages to estimate the truth, and hence averaged model predictions are cancelling with individual model biases.)

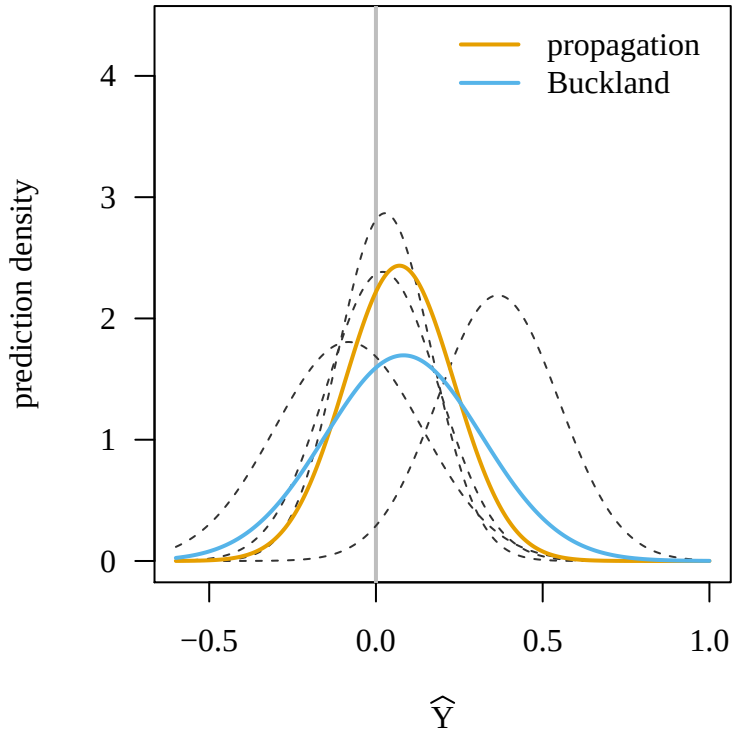
2.2 Buckland et al.'s equation

Using the equation proposed by Buckland et al. (1997) is much simpler, as it does not require the covariances to be estimated. The equation is $var_{\widehat{Y}} = \left(\sum_{m \in \mathscr{M}} w_m \sqrt{var(Y_m) + \gamma_m^2}\right)^2$. As in the previous case, we compute model misspecification bias γ_m and predicted variances for each prediction from the data/fits. In the more general case, both could as well be derived from bootstraps.

```
fits <- c(preds[[1]]$fit, preds[[2]]$fit, preds[[3]]$fit, preds[[4]]$fit)
vars <- c(preds[[1]]$se.fit^2, preds[[2]]$se.fit^2, preds[[3]]$se.fit^2, preds[[4]]$se.fit^2)
gamma <- fits - mean(fits %*% w)
sum(w*sqrt(vars + gamma^2))^2 # var according to Buckland et al.

## [1] 0.05535392
```

This value is slightly higher than that of the previous method (0.043). Adding the Buckland et al. distribution to the previous plot (omitting the non-bootstrapped uncertainty distribution):

```
par(mar=c(4,4,1,1))
plot(1, 1, xlim=c(-0.6, 1), ylim=c(0, 4.4), type="n", las=1, xlab=expression(widehat(Y)),
     ylab="prediction density")
abline(v=0, lwd=2, col="grey") # there is no effect of any X!
for (i in 1:4) curve(dnorm(x, preds[[i]]$fit, preds[[i]]$se.fit), add=T, n=501, col="grey20", lty=2)
curve(dnorm(x, mean(bspred.mat %*% w), sqrt(MSE_m)), add=T, n=501, col=cols[1], lwd=2)
curve(dnorm(x, mean(fits %*% w), sum(w*sqrt(vars + gamma^2))), add=T, n=501, col=cols[2], lwd=2)
legend("topright", bty="n", col=cols[1:2], legend=c("propagation", "Buckland"), lwd=2)
```



Omitting the information on the correlation of predictions, and assuming them to be perfect, yields a wider confidence distribution for the Buckland et al. approach than for the uncertainty propagation.

2.3 Convolution

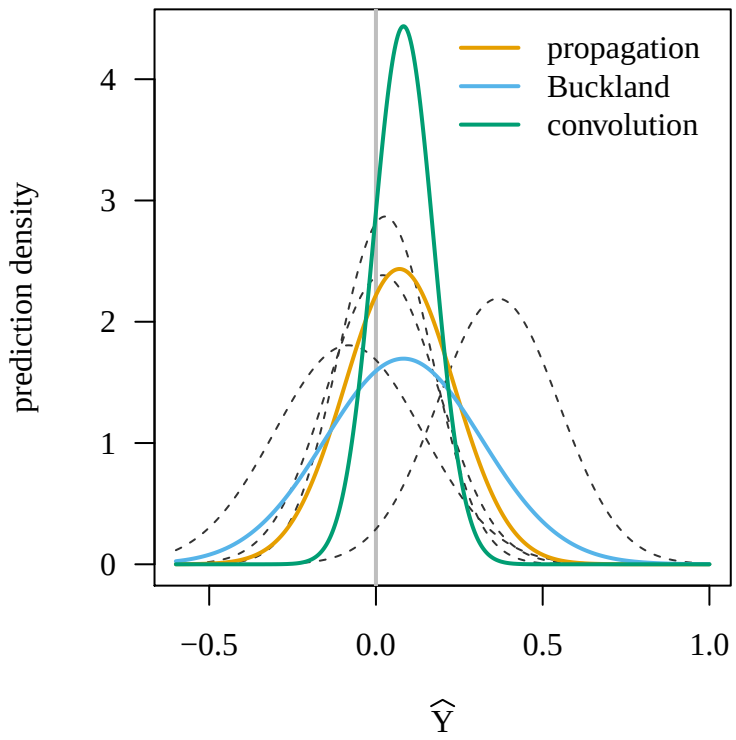
Making the liberal assumption of independence of predictions (which we have already seen in the first method to be violated), we could compute a new normal distribution based on the convoluted contributing four normal

confidence distributions (see main text for details).

```
(meanConv <- c(preds[[1]]$fit, preds[[2]]$fit, preds[[3]]$fit, preds[[4]]$fit) %*% w)

##            [,1]
## [1,] 0.08266836

(sdConv <- sqrt(sum(w^2*c(preds[[1]]$se.fit, preds[[2]]$se.fit, preds[[3]]$se.fit,
                          preds[[4]]$se.fit)^2)))^2

## [1] 0.008083073
```

We squared the last line to get the variance estimate, comparable with the 0.043 of the propagation and the 0.055 of Buckland et al.'s approach. This is clearly *much* narrower, indicating the effect of ignoring the positive correlation among predictions. Let's put the convolution into the plot:

```
par(mar=c(4,4,1,1))
plot(1, 1, xlim=c(-0.6, 1), ylim=c(0, 4.4), type="n", las=1, xlab=expression(widehat(Y)),
     ylab="prediction density")
abline(v=0, lwd=2, col="grey") # there is no effect of any X!
for (i in 1:4) curve(dnorm(x, preds[[i]]$fit, preds[[i]]$se.fit), add=T, n=501, col="grey20", lty=2)
curve(dnorm(x, mean(bspred.mat %*% w), sqrt(MSE_m)), add=T, n=501, col=cols[1], lwd=2)
curve(dnorm(x, mean(fits %*% w), sum(w*sqrt(vars + gamma^2))), add=T, n=501, col=cols[2], lwd=2)
curve(dnorm(x, meanConv, sdConv), add=T, n=501, col=cols[3], lwd=2)
legend("topright", bty="n", col=cols[1:3], legend=c("propagation", "Buckland", "convolution"), lwd=2)
```



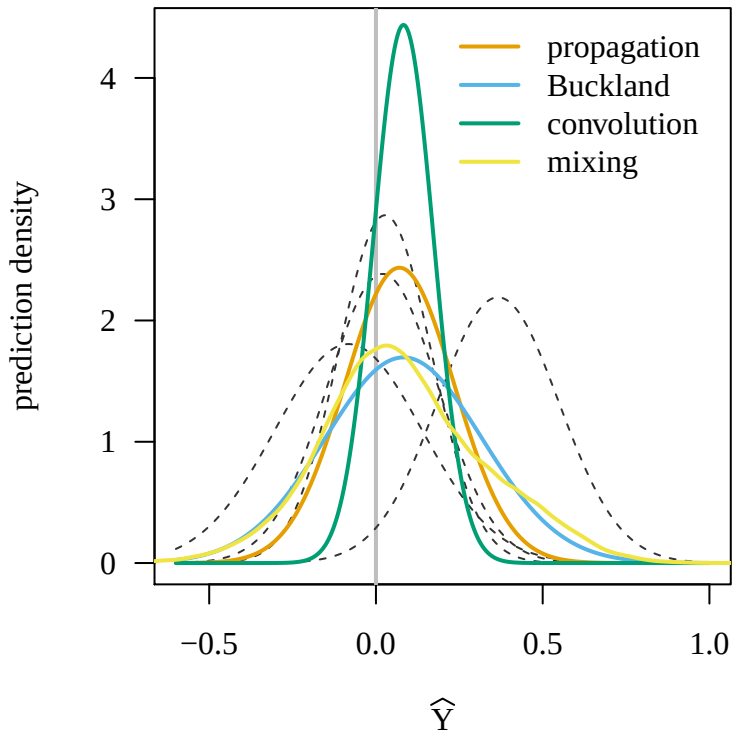
As expected, the convolution yields a much narrower confidence distribution, not including the mean of the right-most model prediction at all, and barely the left-most. This suggests that the convolution will have too narrow confidence intervals, as confirmed by the simulation in the main text.

2.4 Mixing of confidence distributions

Technically, mixing is the easiest way to compute the confidence distribution of the averaged prediction. It is simply the weighted overlay of the individual models, realised by brute-force repeated drawing. We can then

immediately draw the result as a density plot.

```
draws <- 1E5 # a large number of draws across all models
mix <- as.vector(sapply(1:4, function(i) rnorm(round(w[i]*draws), preds[[i]]$fit, preds[[i]]$se.fit)))

par(mar=c(4,4,1,1))
plot(1, 1, xlim=c(-0.6, 1), ylim=c(0, 4.4), type="n", las=1, xlab=expression(widehat(Y)),
     ylab="prediction density")
abline(v=0, lwd=2, col="grey") # there is no effect of any X!
for (i in 1:4) curve(dnorm(x, preds[[i]]$fit, preds[[i]]$se.fit), add=T, n=501, col="grey20", lty=2)
curve(dnorm(x, mean(bspred.mat %*% w), sqrt(MSE_m)), add=T, n=501, col=cols[1], lwd=2)
curve(dnorm(x, mean(fits %*% w), sum(w*sqrt(vars + gamma^2))), add=T, n=501, col=cols[2], lwd=2)
curve(dnorm(x, meanConv, sdConv), add=T, n=501, col=cols[3], lwd=2)
lines(density(mix), col=cols[4], lwd=2)
legend("topright", bty="n", col=cols[1:4], legend=c("propagation", "Buckland", "convolution",
                                                    "mixing"), lwd=2)
```



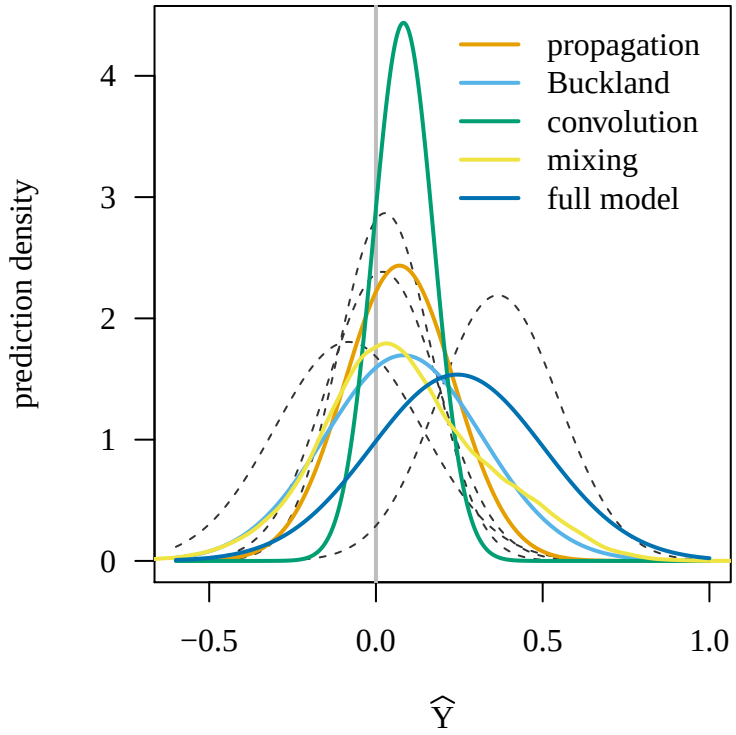
A pleasing feature of mixing is that it relaxes the assumption that the resulting distribution is normal. We had to make this (or another) assumption for the previous cases, as we only estimate mean and variance for each approach. Relying on the Central Value Theorem we hoped that the resulting distribution would be best described by a normal. Mixing allows the confidence distribution to be entirely driven by the contributing distributions, and have any shape, including multi-modality.

2.5 Full model predictions

When a full model exists, we can add its prediction's confidence distribution to the choir. In the present simple situation, the full model does exist, while when using very different model types, it may not (see further below for such a case).

```
fmfull <- lm(Y ~ ., data=X) # use all predictors simultaneously
predfull <- predict(fmfull, newdata=newpoint, se.fit=T)
```

```
par(mar=c(4,4,1,1))
plot(1, 1, xlim=c(-0.6, 1), ylim=c(0, 4.4), type="n", las=1, xlab=expression(widehat(Y)),
     ylab="prediction density")
abline(v=0, lwd=2, col="grey") # there is no effect of any X!
for (i in 1:4) curve(dnorm(x, preds[[i]]$fit, preds[[i]]$se.fit), add=T, n=501, col="grey20", lty=2)
curve(dnorm(x, mean(bspred.mat %*% w), sqrt(MSE_m)), add=T, n=501, col=cols[1], lwd=2)
curve(dnorm(x, mean(fits %*% w), sum(w*sqrt(vars + gamma^2))), add=T, n=501, col=cols[2], lwd=2)
curve(dnorm(x, meanConv, sdConv), add=T, n=501, col=cols[3], lwd=2)
lines(density(mix), col=cols[4], lwd=2)
curve(dnorm(x, predfull$fit, predfull$se.fit), add=T, n=501, col=cols[5], lwd=2)
legend("topright", bty="n", col=cols[1:5], legend=c("propagation", "Buckland", "convolution",
                                                    "mixing", "full model"), lwd=2)
```



The full model's prediction distribution is shifted noticeably to the right. It is also slightly wider than the Buckland et al. approach. Since the full model is unbiased (under repeated sampling, and only if all true predictors are in the model), it should at least include the true value of 0 in 95% of repetitions in its 95% confidence interval. For this single run, truth is certainly within that interval of this and all other methods. It requires repeated simulations, and possibly a less trivial data simulation with actual effects of X on Y, to evaluate the coverage of each method.

3 An R helper function for four methods (not the full model)

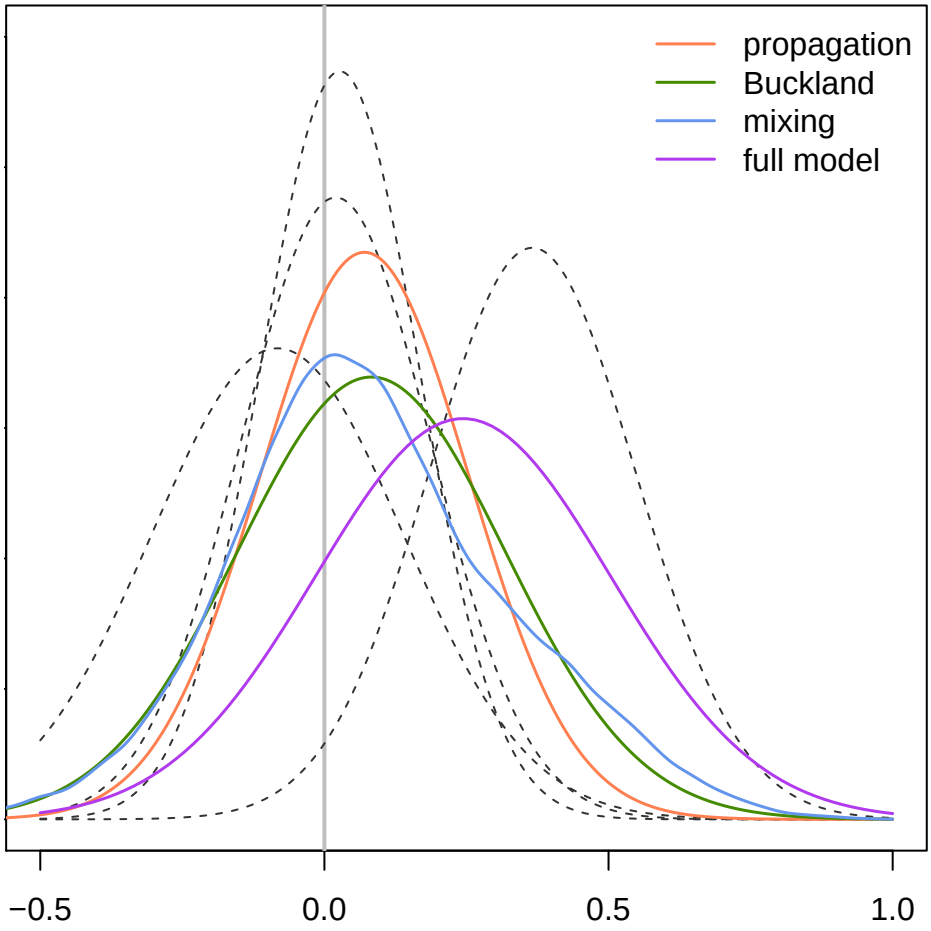
Before we turn to repeated simulations, we can define a helper function for the first four approaches presented above. The full model is not part of this function.

The function accepts, as input, predictions, weights, standard errors of predictions and bootstrap estimates of predictions. It returns a function, which can be used to compute the density for each prediction value. here is an illustration.

```
predictMA <- function(Preds, weights, type, PredsSE = NULL, N = 1e+05,
    PredsBS = NULL) {
    # function to compute confidence distributions from averaged
```

```
    # predictions if used for non-normal data, all data have to
    # be provided AT THE LINK SCALE!
    if (length(Preds) != length(weights))
        stop("The number of models in preds is different from the number of weights.")
    if (type == "propagation") {
        if (is.null(PredsBS))
            stop("Please provide matrix with bootstrap estimates for Preds from each model
                                (at least 500)!")
        ww <- tcrossprod(weights)
        # we estimate model misspecification bias gamma as the
        # weighted mean bias of model predictions:
        gamma_m <- as.vector(Preds %*% w)  # mean for each run of the bootstrap
        # estimate variance of prediction from bootstrap:
        COV <- cov(PredsBS)  # from across the bootstraps
        # now fill in eqn 5, using gamma_m from bootstrapped data!:
        MSE <- (sum(weights * gamma_m))^2 + sum(ww * COV)
        out <- function(x) dnorm(x, mean = mean(PredsBS %*% weights),
            sd = sqrt(MSE))
    }
    if (type == "Buckland") {
        vars <- PredsSE^2
        gamma <- Preds - as.vector(Preds %*% weights)  # model misspecification bias
        varBuckland <- sum(weights * sqrt(vars + gamma^2))^2  # var according to Buckland et al.
        out <- function(x) dnorm(x, mean = mean(Preds %*% weights),
            sd = sqrt(varBuckland))
    }
    if (type == "mixing") {
        warning("You requested mixing. At present, this function assumes your predictions
            are AT THE LINK SCALE and normally distributed.")
        mix <- unlist(sapply(seq_along(Preds), function(i) rnorm(round(N *
            weights)[i], mean = Preds[i], sd = PredsSE[i])))
        dx <- density(mix, n = N/20)
        out <- approxfun(dx)
    }
    if (type == "convolution") {
        warning("You requested a convolution. At present, this function assumes your
            predictions are normally distributed.")
        if (is.null(PredsSE))
            stop("For convolution, please provide estimates for the prediction standard
                                error of each prediction, akin to preds).")
        meanconv <- Preds %*% weights
        # vars <- redsSE^2# c(preds[[1]]£se.fit^2,
        # preds[[2]]£se.fit^2, preds[[3]]£se.fit^2,
        # preds[[4]]£se.fit^2)
        sdconv <- sqrt(sum(weights^2 * PredsSE^2))
        out <- function(x) dnorm(x, mean = meanconv, sd = sdconv)
    }
    return(out)
}
# and now apply it:
```

```r
Preds <- sapply(preds, function(x) x[[1]])
PredsSE <- sapply(preds, function(x) x[[2]])
weights <- w
PredsBS <- bspred.mat
cols <- c("coral", "chartreuse4", "cornflowerblue", "darkorchid2")
xseq <- seq(-1, 1, len = 300)  # sequence along which to plot the confidence distribution
par(mar = c(2, 0.035, 1, 1), family = "sans")
plot(1:2, 1:2, type = "n", col = cols[1], xlim = c(-0.5, 1),
    ylim = c(0, 3), las = 1, xlab = "", ylab = "")
abline(v = 0, lwd = 2, col = "grey")  # there is no effect of any X!
for (i in 1:4) curve(dnorm(x, preds[[i]]$fit, preds[[i]]$se.fit),
    add = T, n = 501, col = "grey20", lty = 2)
propagPred.fun <- predictMA(Preds, weights, PredsSE = PredsSE,
    type = "propagation", PredsBS = PredsBS)
lines(xseq, propagPred.fun(xseq), type = "l", col = cols[1],
    lwd = 1.5)
BuckPred.fun <- predictMA(Preds, weights, PredsSE = PredsSE,
    type = "Buckland")
lines(xseq, BuckPred.fun(xseq), type = "l", col = cols[2], lwd = 1.5)
mixPred.fun <- predictMA(Preds, weights, PredsSE = PredsSE, type = "mixing")

## Warning in predictMA(Preds, weights, PredsSE = PredsSE, type = "mixing"): You requested mixing.
At present, this function assumes your predictions
##            are AT THE LINK SCALE and normally distributed.

lines(xseq, mixPred.fun(xseq), type = "l", col = cols[3], lwd = 1.5)
fullpred <- predict(fmfull, newdata = newpoint, se.fit = T)
curve(dnorm(x, mean = fullpred$fit, sd = fullpred$se.fit), type = "l",
    col = cols[4], lwd = 1.5, add = T)
legend("topright", bty = "n", col = cols[1:4], legend = c("propagation",
    "Buckland", "mixing", "full model"), lwd = 1.5)
```



4 Computing coverage of all five approaches

Using a slight modification of the simulation above, we here repeat the procedure 1000 times and assess the proportion of times the true value was in the 95% confidence interval of each method. If these work out to be 95%, then the nominal coverage is equal to the actual coverage.

The simulation contains again 70 data points, 20 predictors (half of them with a coefficient of -1, the other with one of 1), and the four models are different levels of model simplification (with penalisation factors for the stepwise selection of $k = 0.2, 0.5, 2$ and 4, where 2 is the AIC. Weights are based on AIC, simply because this is the most common approach in the literature to date. It would be simple to change that to any other approach outlined in the main text.

Note that the evaluation of this code takes several hours, and we hence provide the output of our simulation alongside the code.

```
R <- 1000
var.mat <- quant.mat <- matrix(NA, nrow=R, ncol=5) # variance and quantile of truth
nobias.quant.mat <- quant.mat # for the hypothetical case of unbiased model averages
model.estimates.mat <- matrix(NA, nrow=R, ncol=6) # models' point predictions
colnames(model.estimates.mat) <- c("fm1", "fm2", "fm3", "fm4", "fmfull", "avg")
colnames(var.mat) <- colnames(quant.mat) <- c("propagation", "Buckland", "convolution",
                                              "mixing", "full model")
cols <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
# from color-blind scheme
verbose=F

for (r in 1:R){
```

```
        set.seed(r)
        N <- 70 # number of data points
X <- as.data.frame(matrix(NA, ncol=20, nrow=N))
        for (i in 1:20) X[,i] <- runif(N) # 20 more or less uncorrelated predictors
        colnames(X) <- paste0("X", 1:20)
        X <- as.data.frame(scale(X, scale=F)) # center all predictors
        Y <- 1 + as.matrix(X) %*% rep(c(-1,1), 10) + rnorm(N, sd=1)
        dats <- cbind(Y, X)
        rm(X, Y)

        fm1 <- step(lm(Y ~ ., data=dats), k=0.2, trace=F)
        fm2 <- step(fm1, k=0.5, trace=F)
        fm3 <- step(fm2, k=2, trace=F)
        fm4 <- step(fm3, k=4, trace=F)

        fmfull <- lm(Y ~ ., data=dats)

        # compute AIC-weights (as those were used by Fletcher/Turek):
        AICs <- sapply(list(fm1, fm2, fm3, fm4), AIC)
        round(w <- exp(0.5*AICs) / sum(exp(0.5*AICs)), 3)

        newpoint <- as.data.frame(t(rep(.1, 20))) # a single point of evaluation
        colnames(newpoint) <- paste0("X", 1:20)
        preds <- lapply(list(fm1, fm2, fm3, fm4), predict, newdata=newpoint, se.fit=T)
        predAvg <- sapply(list(fm1, fm2, fm3, fm4), predict, newdata=newpoint, se.fit=F) %*% w

        model.estimates.mat[r, ] <- unlist(c(lapply(list(fm1, fm2, fm3, fm4), predict,
                newdata=newpoint, se.fit=F), predict(fmfull, newdata=newpoint), predAvg))

        truth <- 1 + as.matrix(newpoint) %*% rep(c(-1,1), 10)

        # bootstrap for estimating ingredients for var-est-formula:
        Nbs <- 500
        bspred.mat <- matrix(NA, nrow=Nbs, ncol=4)
        for (i in 1:Nbs){
                Xbs <- dats[sample(nrow(dats), replace=T), ]
                fm1bs <- lm(Y ~ 1, data=Xbs)
                fm2bs <- lm(Y ~ ., data=Xbs[, 1:6])
                fm3bs <- lm(Y ~ ., data=Xbs[, c(1, 7:13)])
                fm4bs <- lm(Y ~ ., data=Xbs[, c(1, 14:21)])
                bspred.mat[i,] <- sapply(list(fm1bs, fm2bs, fm3bs, fm4bs), predict,
                                        newdata=newpoint)
        }
        # now compute model misspecification bias as deviation of each prediction from the mean:
        # biasBS <- (bspred.mat - matrix(bspred.mat %*% w, ncol=4, nrow=Nbs))
        # we estimate model misspecification bias gamma as the mean bias (colMeans(bias)):
        # gamma_m <- colMeans(biasBS)   # mean across bootstraps

        # Compute "bias" on ORIGINAL predictions:
        # (Note that this is not the actual bias, because truth is unknown in real life.)
```

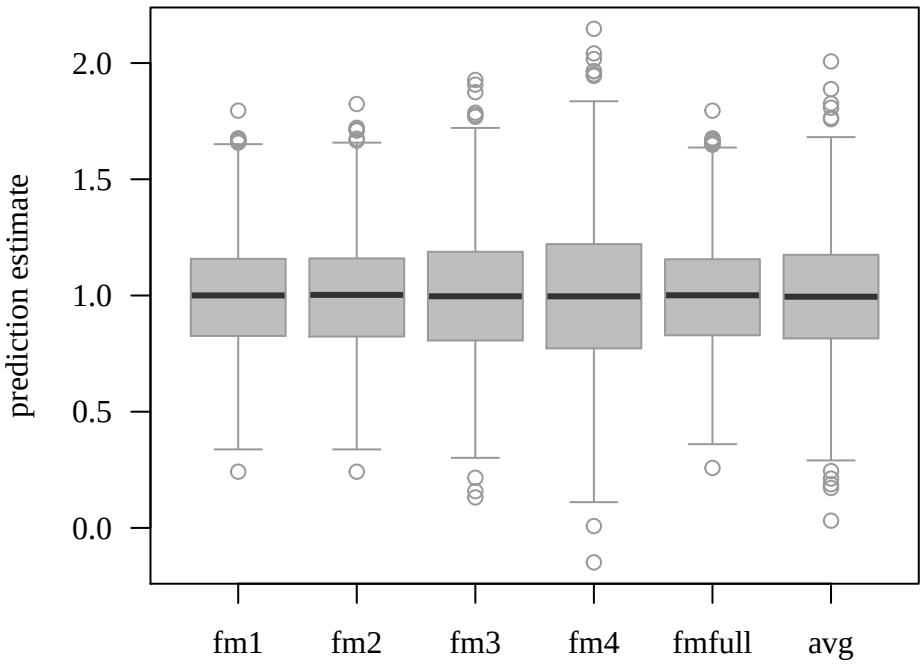
```
      gamma_m <- sapply(list(fm1, fm2, fm3, fm4), predict, newdata=newpoint) - rep(predAvg, 4)

      # the covariance term in eqn 4 is:
      COV <- cov(bspred.mat)
      ww <- tcrossprod(w)
      # now fill in eqn 1 of this document, using gamma_m from bootstrapped data!:
      MSE <- (sum(w * gamma_m))^2 + sum(ww*COV)
      # (Note that since we assume the model average is unbiased, the first term is actually 0.)

# store var and compute 95%CI, using as model average the ORIGINAL model predictions:
      var.mat[r, 1] <- MSE
      quant.mat[r, 1] <- pnorm(truth, mean=predAvg, sd=sqrt(MSE)) #old: bs.means: mean(bspred.mat %*% w)

      # now assume that averaged predictions were unbiased (which of course we could never
      # know in real life):
      gamma_m <- sapply(list(fm1, fm2, fm3, fm4), predict, newdata=newpoint) - rep(truth, 4)
      MSE <- (sum(w * gamma_m))^2 + sum(ww*COV)
      nobias.quant.mat[r, 1] <- pnorm(truth, mean=predAvg, sd=sqrt(MSE))


      ## 2. Buckland et al. correction:
      fits <- c(preds[[1]]$fit, preds[[2]]$fit, preds[[3]]$fit, preds[[4]]$fit)
      vars <- c(preds[[1]]$se.fit^2, preds[[2]]$se.fit^2, preds[[3]]$se.fit^2,
                preds[[4]]$se.fit^2)
      gamma <- fits - rep(predAvg, 4)
      varBuckland <- sum(w*sqrt(vars + gamma^2))^2 # var according to Buckland et al.
      var.mat[r, 2] <- varBuckland
      quant.mat[r, 2] <- pnorm(truth, predAvg, sqrt(varBuckland))

      # now assume that averaged predictions were unbiased (which of course we could never
      # know in real life):
      gamma <- fits - rep(truth, 4)
      varBuckland <- sum(w*sqrt(vars + gamma^2))^2
      nobias.quant.mat[r, 2] <- pnorm(truth, mean=predAvg, sd=sqrt(varBuckland))


      ## 3. convolution as sum of independent normals:
      var.mat[r, 3] <- sum(w^2*c(preds[[1]]$se.fit^2, preds[[2]]$se.fit^2,
                                 preds[[3]]$se.fit^2, preds[[4]]$se.fit^2))
      quant.mat[r, 3] <- pnorm(truth, mean(c(preds[[1]]$fit, preds[[2]]$fit, preds[[3]]$fit,
                       preds[[4]]$fit) %*% w), sqrt(sum(w^2*c(preds[[1]]$se.fit^2,
                       preds[[2]]$se.fit^2, preds[[3]]$se.fit^2, preds[[4]]$se.fit^2))))

      # now assume that averaged predictions were unbiased (which of course we could never
      # know in real life):
# (does not apply, as there is no bias-correction involved!)
      nobias.quant.mat[r, 3] <- quant.mat[r, 3]


      ## 4. mixing = overlay
```

```
        howOften <- round(w*400000) # draw a lot of times ...
        mix <- as.vector(sapply(1:4, function(i) rnorm(howOften[i], preds[[i]]$fit,
                                                       preds[[i]]$se.fit)))
        var.mat[r, 4] <- var(unlist(mix))
        #quant.mat[r, 4] <- ecdf(unlist(mix))(truth) # from mix, approximated
  quant.mat[r, 4] <- w[1] * pnorm(truth, preds[[1]]$fit, preds[[1]]$se.fit) +
    w[2] * pnorm(truth, preds[[2]]$fit, preds[[2]]$se.fit) +
    w[3] * pnorm(truth, preds[[3]]$fit, preds[[3]]$se.fit) +
    w[4] * pnorm(truth, preds[[4]]$fit, preds[[4]]$se.fit) # "analytical" estimate

        # now assume that averaged predictions were unbiased (which of course we could never
  # know in real life):
  # (does not apply, as there is no bias-correction involved!)
        nobias.quant.mat[r, 4] <- quant.mat[r, 4]



        ## 5. full model:
  predfull <- predict(fmfull, newdata=newpoint, se.fit=T)
        var.mat[r, 5] <- predfull$se.fit^2
        quant.mat[r, 5] <- pnorm(truth, mean=predfull$fit, sd=predfull$se.fit)

        # now assume that averaged predictions were unbiased (which of course we could never
        # know in real life):
  # (does not apply, as there is no bias-correction involved!)
        nobias.quant.mat[r, 5] <- quant.mat[r, 5]

        print(r)
  #save(model.estimates.mat, var.mat, quant.mat, nobias.quant.mat, file="coverageSimu.Rdata")
}
```

```
load("coverageSimu.Rdata")
round(head(quant.mat), 3)

##      propagation Buckland convolution mixing full model
## [1,]       0.377    0.400       0.346  0.394      0.429
## [2,]       0.012    0.052       0.006  0.053      0.057
## [3,]       0.075    0.156       0.024  0.160      0.285
## [4,]       0.002    0.008       0.000  0.010      0.016
## [5,]       0.819    0.760       0.913  0.759      0.780
## [6,]       0.062    0.117       0.019  0.121      0.169
```

First, we may want to have a look at the predictions, across the 1000 repetitions.

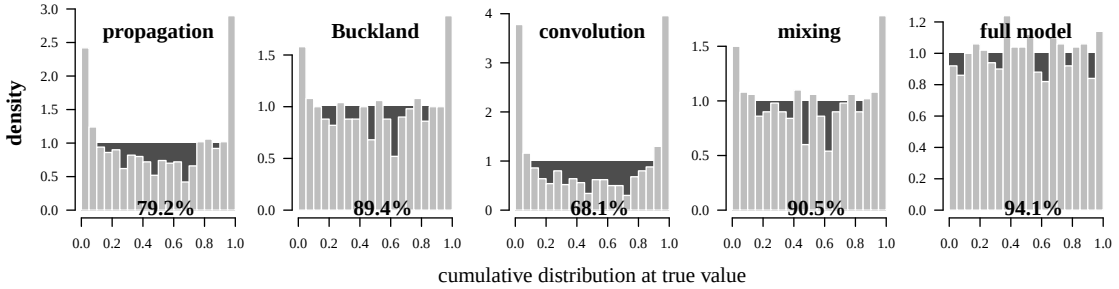
```
par(mar=c(4,4,1,1))
boxplot(model.estimates.mat, col="grey", border="grey60", whisklty=1, pch=1,
        medcol="grey20", las=1, ylab="prediction estimate")
```



On average, all six ways to predict yield the same mean (which correctly lies on 1, indicating that all methods are unbiased). Variance is slightly reduced in the full model, while averaging yields no obvious benefit for the estimate in terms of spread.

Let's look at the actual coverage.

```
#summary(quant.mat)
(coverage <- apply(quant.mat, 2, function(x) sum(x < 0.975 & x > 0.025))/nrow(quant.mat))

## propagation    Buckland convolution      mixing  full model
##       0.792       0.894       0.681       0.905       0.941

titles <- c("propagation", "Buckland", "convolution", "mixing", "full model")
par(mfrow=c(1,5), mar=c(4,2,1,1), oma=c(0, 4, 0, 0))
for (i in 1:5){
        hist(quant.mat[,i], col="grey", border="white", main="", freq=F, las=1, xlab="",
             breaks=seq(0, 1, by=0.05), ylab="")
        polygon(x=c(0, 1, 1, 0), y=c(0, 0, 1, 1), col="grey30", border="transparent")
        hist(quant.mat[,i], col="grey", border="white", main="", freq=F, las=1, xlab="", ylab="",
             breaks=seq(0,1,by=0.05), add=T)
        mtext(titles[i], side=3, line=-2, font=2)
        mtext(paste0(coverage[i]*100, "%"), side=1, line=-1.0, at=0.55, font=2, cex=1)
        if (i==3) mtext("cumulative distribution at true value", side=1, line=3, font=1)
        if (i==1) mtext("density", side=2, line=3, font=2)
}
```



In this simulation, the full model alone has the nominal coverage: in 95% of the cases is the truth in the 95% confidence interval. Mixing is a very close second, followed by Buckland et al.'s correction formula. Uncertainty propagation is too narrow, covering the truth in only 80% of cases, while convolution is clearly unacceptable.

The reason for the poor performance of eqn 5-based error propagation is the estimation of bias. Equation 5 assumes that bias is known, and we approximate that by *assuming* that the averaged prediction is unbiased (see main text, section 2.4, point 1 in the final list of options). If it isn't, this is what we get.

Apparently, Buckland et al.'s approach, which is weird in its derivation, has much less of a problem with this. (Note on "weird": Their equation 6 is clearly wrong. It states that their estimator for the variance of the prediction by model k depends on the truth, whereas in their eqn 5 it correctly depended on the prediction mean. The variance of an estimator describes just that: the variance of the estimator around the estimate, but not around the truth. The latter would be quantified by the MSE, not the variance. Their eqn 6 - 9 should refer instead to the mean squared error of the estimator, not the variance of the estimator.)

In a similiar way, we could investigate how the methods would fare IF (big if) the model average was unbiased. Remember, we don't know the truth in real life, so there we can never make this analysis. Also note that only "error propagation" and "Buckland" have a term specifying model bias; hence only these two will look different to the previous analysis.

```
(coverage <- apply(nobias.quant.mat, 2, function(x) sum(x < 0.975 & x > 0.025))/nrow(nobias.quant.mat))

## [1] 1.000 1.000 0.681 0.905 0.941
```

Unsurprisingly, setting the bias-correction to the correct term makes all predictions to fall into the 95% quantiles of the uniform distribution, the predictions have "full" coverage (not only the intended nominal coverage). This only means that both methods *would* work if the truth was known (an academic point).