Outlier detection with the nucdataBaynet package

Georg Schnabel

May 4, 2022

In this document we demonstrate how outlier detection can be done with the *nucdataBaynet* package. Importantly, statistical and systematic uncertainties are not only taken into account but play an active role in the detection of outliers.

We are going to construct a number of artificial datasets and assume that each has been measured with one out of several possible techniques. We will construct a few datasets to be outliers because the underlying technique produces biased results. Finally, we will use inference in a Bayesian network to get the posterior estimates and uncertainties of the statistical errors and errors associated with the measurement technique. We will detect outliers by checking whether the posterior estimates and uncertainties of the errors due to a measurement bias are compatible with their prior assumption, e.g., the uncertainties the evaluators has assigned to the datasets.

Even though this is not a requirement of the Bayesian network framework, we assume that all datasets belong to the same reaction channel. As a baseline model to fit the datasets, we use a Gaussian process that imposes as the only constraint a certain smoothness of the second derivative of possible solution curves.

Loading the required packages

Several packages need to be loaded. To define the datatable with the information about nodes, we make use of the *data.table* package. To plot the Bayesian network, we use the *igraph* package and to plot the experimental data together with the regression line, we use the *ggplot2* package. The *Matrix* package is used to construct sparse covariance matrices. These packages can be loaded by

```
library(data.table)
library(ggplot2)
library(igraph)
library(nucdataBaynet)
library(Matrix)
```

Definition of the truth

In this schematic example we assume that we know the true function and create artificial datasets by adding noise on top of the true function. We use a simple straight line model as the truth:

```
truefun <- function(x) {
  return(x)
}
```

Construction of artificial datasets

Now we construct artificial datasets that are perturbed around this truth. We apply two perturbations: One part of the perturbation is due to a statistical error assumed for each datapoint and the other part is due to a systematic error about the accuracy of the measurement technique, which affects all datasets associated with a specific measurement technique.

We assume that the statistician/evaluator has correctly estimated all uncertainties of the technique biases except for one technique. For the latter technique we assume that the uncertainty of the bias has been underestimated which makes all datasets measured with this technique outliers with respect to the datasets measured with one of the other techniques. The following specifications formalize our assumptions about the measurement biases:

```
set.seed(10)
num_techniques <- 5
technique_bias <- c(rnorm(4, 0, sd=1), 5)
# this is the assumption of the statistician
# and he/she messes up and underestimates the uncertainty of the third technique
technique_unc <- c(1, 1, 1, 1, 1)
```

Now we create the datasets and shift them by the measurement biases, which we assume to be known in this schematic example. We additionally perturb each datapoint to simulate statistical errors.

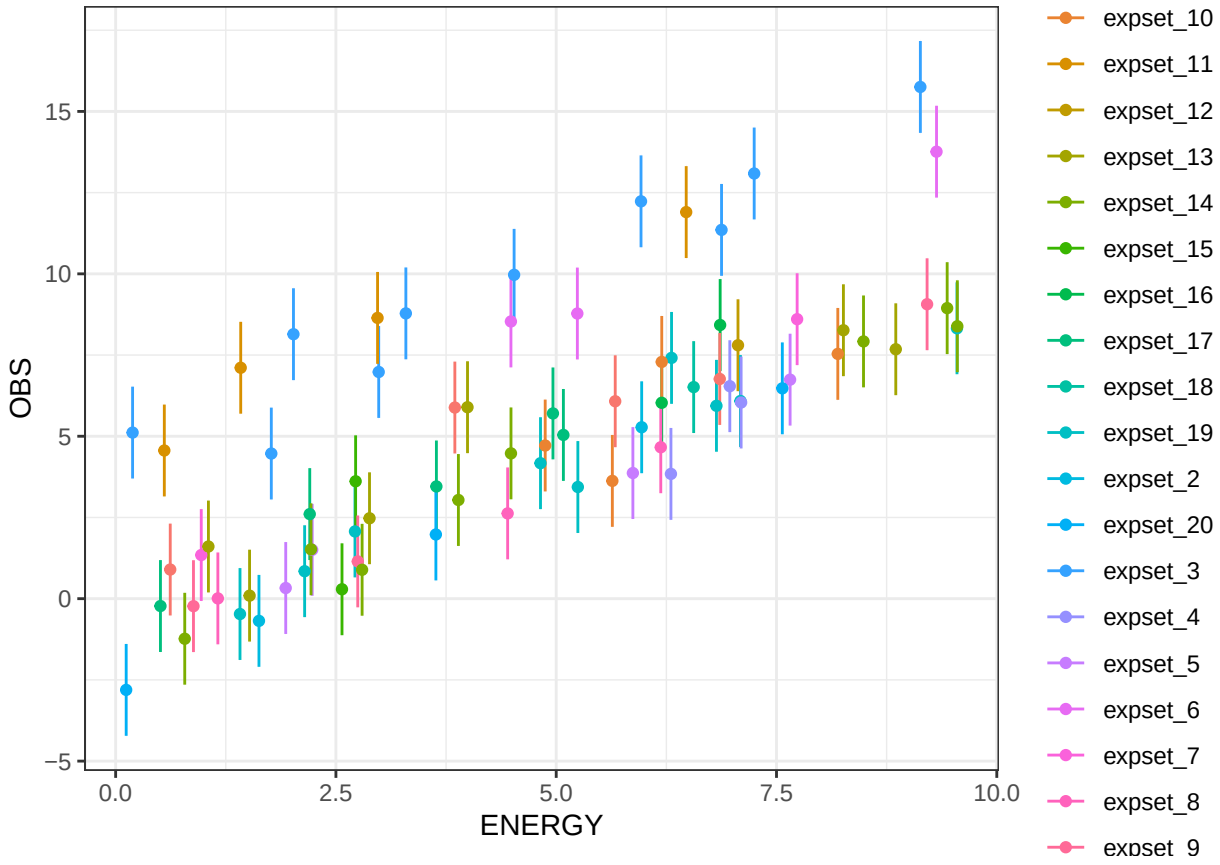
```
set.seed(11)
dataset_names <- paste0("expset_", sample(20))
measurement_technique <- sample(num_techniques, length(dataset_names), replace=TRUE)

expdata_list <- lapply(seq_along(dataset_names), function(i) {
  numpoints <- sample(10,1)
  energies <- runif(numpoints, 0, 10)
  technique <- sample(num_techniques, 1)
  statunc <- 1
  staterrors <- rnorm(numpoints, 0, statunc)
  data.table(
    NODE = dataset_names[i],
    PRIOR = 0,  # prior estimate of statistical error
    UNC = statunc,  # statistical uncertainty (=prior uncertainty of statistical error)
    OBS = truefun(energies) + staterrors + technique_bias[technique],
    ENERGY = energies,
    TECHNIQUE = measurement_technique[i],
    # for the sake of convenience. the prior uncertainty of the technique
    # bias will be stored in their own rows.
    TOTUNC = sqrt(statunc^2 + technique_unc[technique]^2)
  )}
)
# create a datatable with all datasets included
node_dt <- rbindlist(expdata_list)
node_dt[, IDX := seq_len(.N)]
```

The column `OBS` contains the observed value given as sum of true value, statistical error and measurement bias in this example.

Let's visualize the data to get an impression of them.

```
library(ggplot2)
ggp <- ggplot(data=node_dt) + theme_bw()
ggp <- ggp + geom_errorbar(aes(x=ENERGY, ymin=OBS-TOTUNC, ymax=OBS+TOTUNC,
                               col=NODE))
ggp <- ggp + geom_point(aes(x=ENERGY, y=OBS, col=NODE))
print(ggp)
```



Including the technique biases in the Bayesian network

Next, we extend the `node_dt` datatable by introducing the measurement technique errors and link them to the affected datasets. Here is the code to extend the `node_dt` datatable:

```
technique_dt <- data.table(
    NODE = paste0("technique_", seq_len(num_techniques)),
    PRIOR = 0,  # prior estimate of error due to measurement technique
    UNC = technique_unc,  # contains the assumed uncertainty due to
                          # measurement technique, see above
    OBS = NA,  # Not answered because we cannot observe technique biases
               # as they are merely a latent variable in the statistical model
    ENERGY = NA,  # Not answered because the technique is not linked to a
                  # particular energy
    TECHNIQUE = seq_len(num_techniques)
)

# append to the datatable with experimental datasets defined above
node_dt <- rbindlist(list(node_dt, technique_dt), fill=TRUE)
node_dt[, IDX := seq_len(.N)]  # reindex
```

Here is the specification that defines how the technique errors need to be mapped to the experimental datasets:

```
technique_error_to_exp_mapspec <- list(
  mapname = "technique_error_mapping",
  maptype = "normerr_map",
  src_idx = node_dt[grepl("^technique_", NODE), IDX],
```

```
  tar_idx = node_dt[grepl("^expset_", NODE), IDX],
  src_feat = node_dt[grepl("^technique_", NODE), TECHNIQUE] ,
  tar_feat = node_dt[grepl("^expset_", NODE), TECHNIQUE]
)
```

In this specification, we assume that the technique biases need to be added in order to explain the difference between the truth and the measured datapoints. Another option would be to define them as being relative to the truth. This is also possible with the Bayesian network package but not part of this tutorial.

Including the baseline Gaussian process model to fit the data

The last ingredient to complete the Bayesian network is the inclusion of the baseline model. As we normally don't know the truth, we need to define a flexible model that should be able to capture the true trend. To this end, we use a Gaussian process construction that only imposes a constraint on the smoothness.

```
# prior on the function values
gp_prior_dt <- data.table(
  NODE = 'baseline_gp',
  PRIOR = 0,
  UNC = 1e6,  # huge uncertainty to reflect ignorance about the value
  ENERGY = 0:10,  # regular spaced mesh
  OBS = NA  # no observation for this node
)

# prior on the second derivatives of the function values
gp_prior_2nd_dt <- data.table(
  NODE = 'baseline_gp_2nd',
  PRIOR = 0,
  UNC = 0.1,  # small uncertainty on 2nd derivative -> smooth function
  ENERGY = 1:9,
  OBS = 0  # we assume that a zero value has been observed.
           # this pseudo observation is responsible for the regularization
           # to make the GP solution smooth
)

# append to the datatable with experimental datasets and technique biases
node_dt <- rbindlist(list(node_dt, gp_prior_dt, gp_prior_2nd_dt), fill=TRUE)
node_dt[, IDX := seq_len(.N)]  # reindex
```

Now we create two mappings. One links the values of the GP to the second derivatives of the GP:

```
gp_to_gp2nd_mapspec <- list(
    mapname = "gp_to_gp2nd_map",
    maptype = "derivative2nd_map",
    src_idx = node_dt[NODE=="baseline_gp", IDX],
    tar_idx = node_dt[NODE=="baseline_gp_2nd", IDX],
    src_x = node_dt[NODE=="baseline_gp", ENERGY],
    tar_x = node_dt[NODE=="baseline_gp_2nd", ENERGY]
  )
```

The other one maps the values of the Gaussian process defined at the mesh points to the energies of the experimental datapoints by linear interpolation:

```
gp_to_exp_mapspec <- list(
  mapname = "gp_to_exp_map",
```

```
  maptype = "linearinterpol_map",
  src_idx = node_dt[NODE=="baseline_gp", IDX],
  tar_idx = node_dt[grepl("^expset_", NODE), IDX],
  src_x = node_dt[NODE=="baseline_gp", ENERGY],
  tar_x = node_dt[grepl("^expset_", NODE), ENERGY]
)
```

Linking everything together

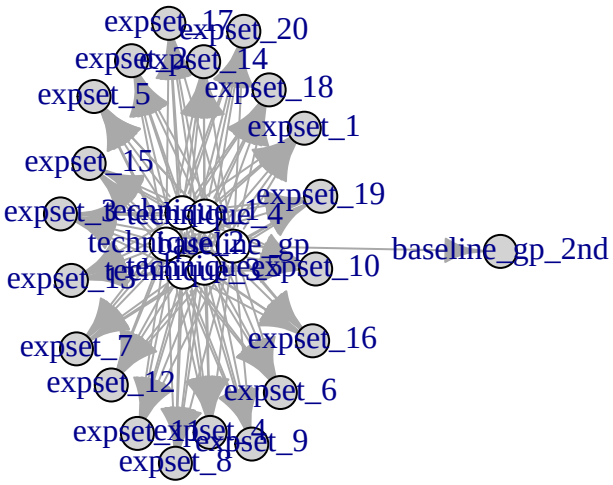
We have constructed the datatable with all variables. We also defined the necessary mappings that link the baseline GP and the technique biases to the experimental datasets. Now we produce the compound mapping that takes into account all the individual mappings we've defined up to this point in this notebook.

```
compound_mapspec <- list(
  maptype = "compound_map",
  maps = list(gp_to_exp_mapspec, technique_error_to_exp_mapspec, gp_to_gp2nd_mapspec)
)

compmap <- create_map(compound_mapspec)
```

Here is a graphical representation of the relationships between the variables. The display is obviously not very meaningful in this notebook but the code is provided nevertheless for reference. You can change the number of datasets and measurement techniques above to obtain a more pleasing visualization.

```
grph <- get_network_structure(compmap$getMaps(), node_dt$NODE, node_dt$OBS)
plot(grph)
```



Doing the inference and inspect outliers

Both our `node_dt` and our compound mapping are completely set up. We can now perform the Bayesian inference to get the posterior estimates of all the variables, which also includes posterior estimates and uncertainties of the technique biases. Here is the code to calculate the posterior estimates and covariance matrix for all quantities involved:

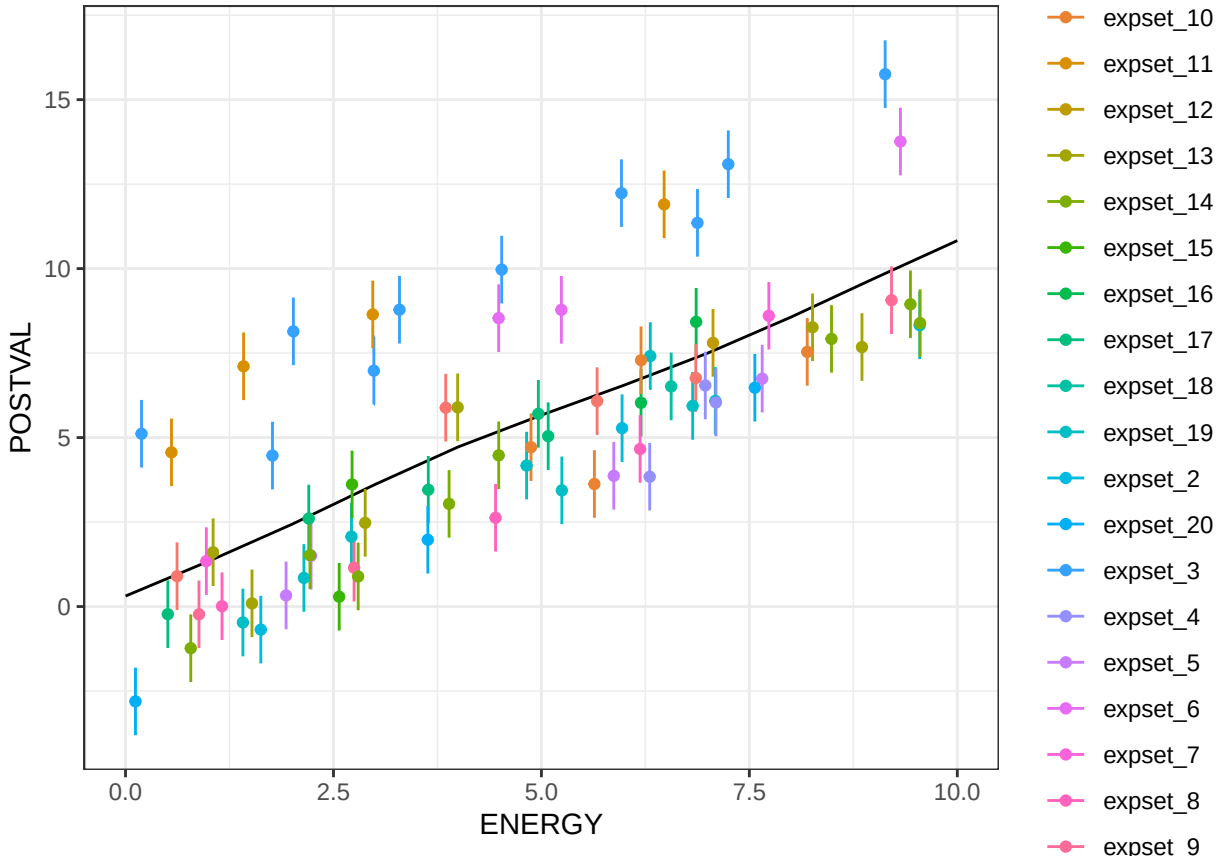
```
covmat <- Diagonal(n=nrow(node_dt), x=node_dt$UNC^2)
obs <- node_dt$OBS
zprior <- node_dt$PRIOR
postest <- glsalgo(compmap, zprior, covmat, obs)
```

```
postcov <- get_posterior_cov(compmap, postest, covmat, obs,
                             row_idcs=seq_along(zprior), col_idcs=seq_along(zprior))

node_dt[, POSTVAL := postest]
node_dt[, POSTUNC := sqrt(diag(postcov))]
```

The outlier detection relies on a flexible baseline model. Let's plot this fitted model together with the experimental data.

```
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_line(aes(x=ENERGY, y=POSTVAL), data=node_dt[NODE=="baseline_gp"])
ggp <- ggp + geom_errorbar(aes(x=ENERGY, ymin=OBS-UNC, ymax=OBS+UNC, col=NODE),
                           data=node_dt[grepl("^expset_", NODE)])
ggp <- ggp + geom_point(aes(x=ENERGY, y=OBS, col=NODE), data=node_dt[grepl("^expset_", NODE)])
print(ggp)
```

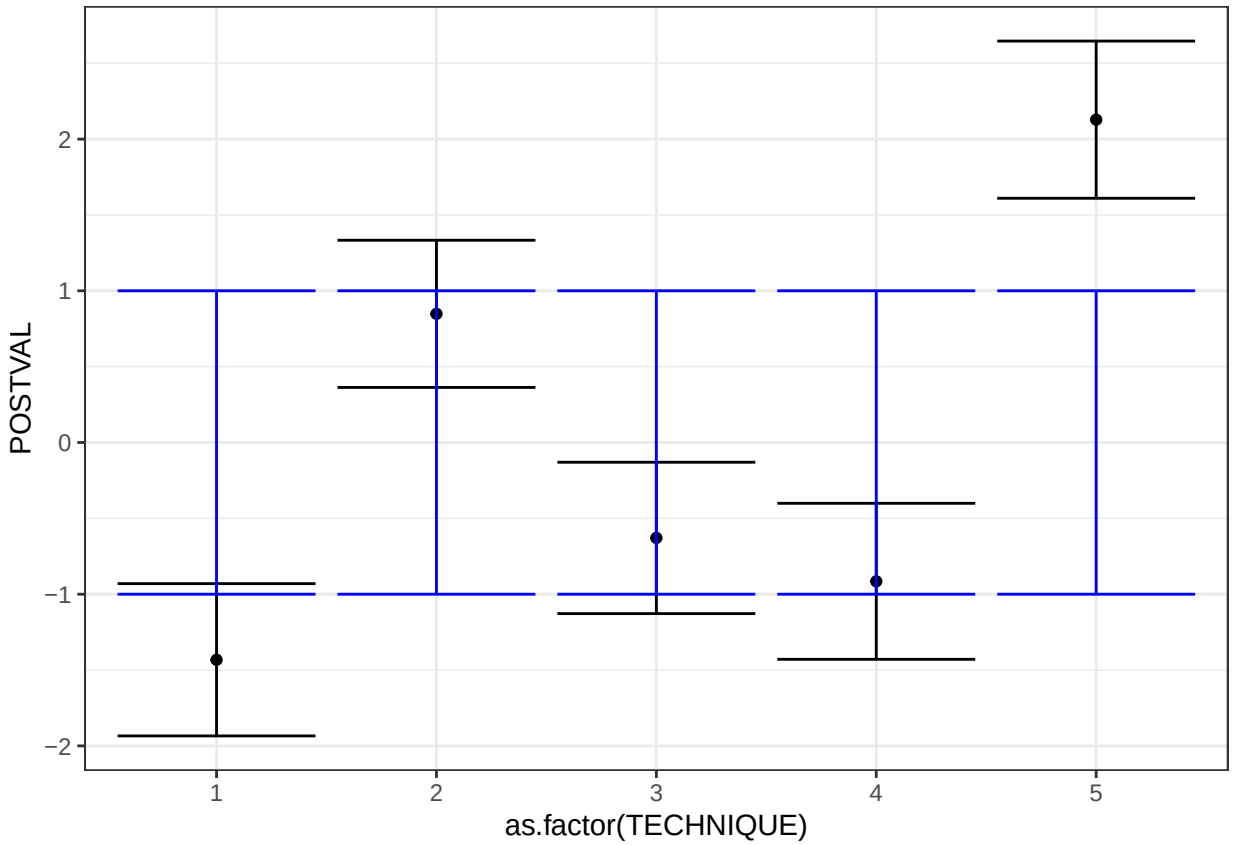


The datasets affected by the biased measurement technique appear clearly as outliers. The aim of outlier detection is to do it algorithmically without the need to visualize the data and manually remove them. The relevant numbers in the `node_dt` are the posterior estimates and uncertainties of the statistical errors and technique biases.

First we plot the technique biases, their 1-sigma posterior uncertainty and compare these estimates to the prior estimates and uncertainties. The prior estimates of all error quantities are zero.

```
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_errorbar(aes(x=as.factor(TECHNIQUE),
                               ymin=POSTVAL-POSTUNC, ymax=POSTVAL+POSTUNC),
                           data = node_dt[grepl("^technique_", NODE)])
```

```
ggp <- ggp + geom_point(aes(x=as.factor(TECHNIQUE), y=POSTVAL),
                        data = node_dt[grepl("^technique_", NODE)])
ggp <- ggp + geom_errorbar(aes(x=as.factor(TECHNIQUE), ymin=-UNC, ymax=UNC),
                         col="blue", data = node_dt[grepl("^technique_", NODE)])
print(ggp)
```



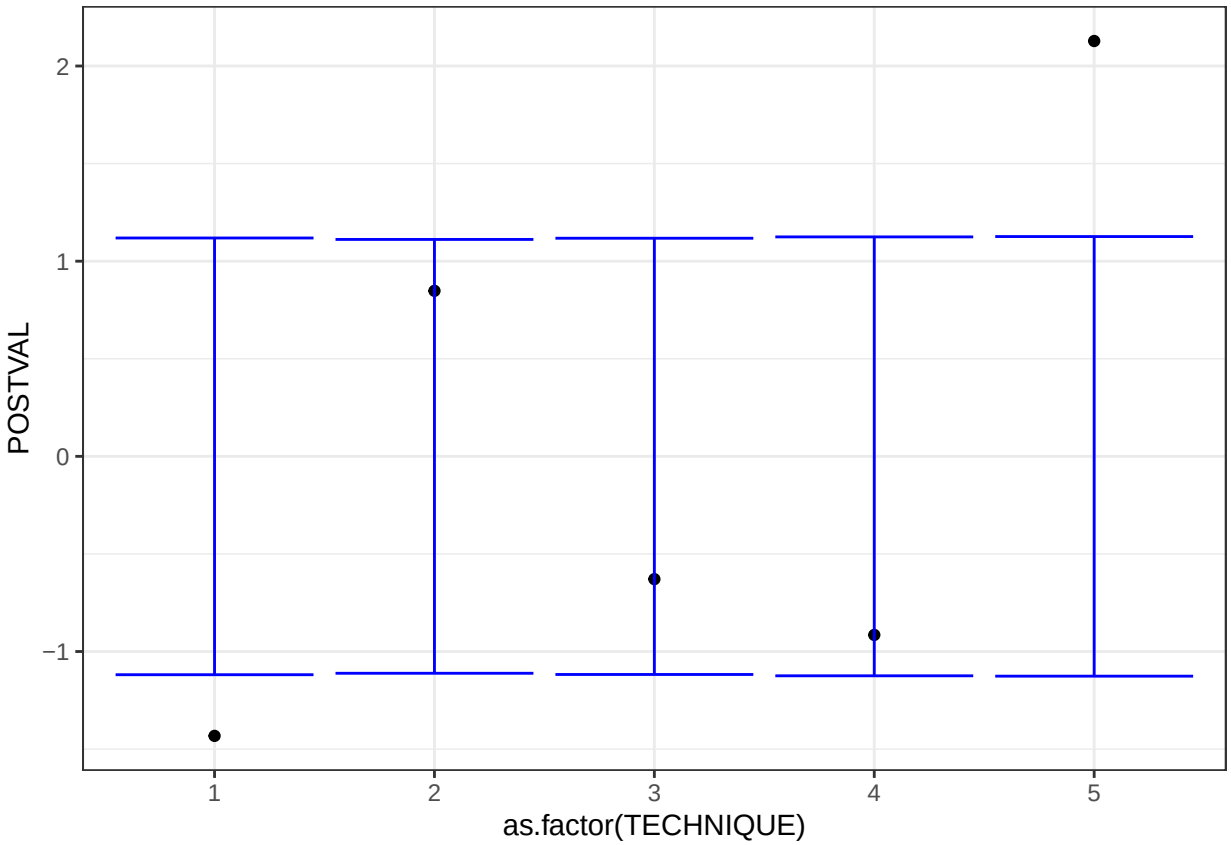
The blue line indicates the 1 sigma prior uncertainty of the measurement techniques centered at zero. This analysis shows already that technique 5 is more incompatible with the prior than the other techniques.

To get a clearer picture, we can make use of the fact that the difference of two normally distributed variables is also a normal distribution with standard deviation sqrt(unc1^2 + unc2^2)

```
# UNC is the prior uncertainty
node_dt[, DIFFUNC := sqrt(POSTUNC^2 + UNC^2)]
```

We plot the same type of plot again but now display the error bars reflecting the uncertainty of the difference between the prior and posterior estimates.

```
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_point(aes(x=as.factor(TECHNIQUE), y=POSTVAL),
                        data = node_dt[grepl("^technique_", NODE)])
ggp <- ggp + geom_errorbar(aes(x=as.factor(TECHNIQUE), ymin=-DIFFUNC, ymax=DIFFUNC),
                         col="blue", data = node_dt[grepl("^technique_", NODE)])
print(ggp)
```



We see that the posterior estimate of the bias due to technique 5 is almost two standard deviations away. We can introduce a threshold on the difference between prior estimate and posterior estimate normalized to the standard deviation of the difference and use it to automatically reject outliers.

Summary

In this notebook we constructed a number of artificial datasets and assumed that they were obtained by different measurement techniques. We introduced a large bias for one measurement technique and intentionally underestimated it's prior uncertainty. We introduced a flexible GP baseline model that imposes as only constraint smoothness on solution curves. We then applied Bayesian inference to obtain posterior estimates and uncertainties for the measurement technique biases. Finally, we explained how the uncertainty of the difference between prior and posterior estimate can be obtained on the basis of the prior and posterior uncertainty. A threshold can then be introduced on this difference normalized by the uncertainty of the difference to automatically reject outliers.