Exploring the impact of penalties on derivatives with the nucdataBaynet package

Georg Schnabel

2023-01-10

Employing pseudo-observations of derivatives is a powerful method to bias the posterior to concentrate on functions with desirable shapes. For instance, pseudo-observations of the second derivative of a function can bias the posterior towards smooth functions despite discrepant data, which would otherwise cause kinks or abrupt changes.

In this tutorial, we are going to explore the possibilities of pseudo-observations of derivatives of various orders in more detail to develop a feeling of the impact of these assumptions.

For this purpose, we will be setting up a toy scenario with synthetic data points, create a Bayesian network for the analysis, and then modify the Bayesian network to explore the impact of the pseudo-observations of derivatives.

Before we start, we remark that the approach of pseudo-observations scales well to large data sets, which is an advantage over Gaussian process regression with popular kernels, such as the squared exponential. Importantly, the construction of Bayesian networks with pseudo-observations is an implicit way to define a Gaussian process kernel without explicitly defining a covariance function.

Loading the required packages and defining the energy range

We will make use of the functionality of the *data.table* package to build up a data table with the information about the nodes present in the network. The *Matrix* package will be used to create a sparse covariance matrix that contains the covariance matrix blocks associated with the nodes. We also need to load the *nucdataBaynet* package to establish the links between the nodes and perform Bayesian inference in the Bayesian network. The *ggplot2* package will be pertinent for plotting experimental data and the estimated function.

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

Generating the synthetic data

Let us assume that the true function $f(x)$ is given by a parabola: $f(x) = x^2$.

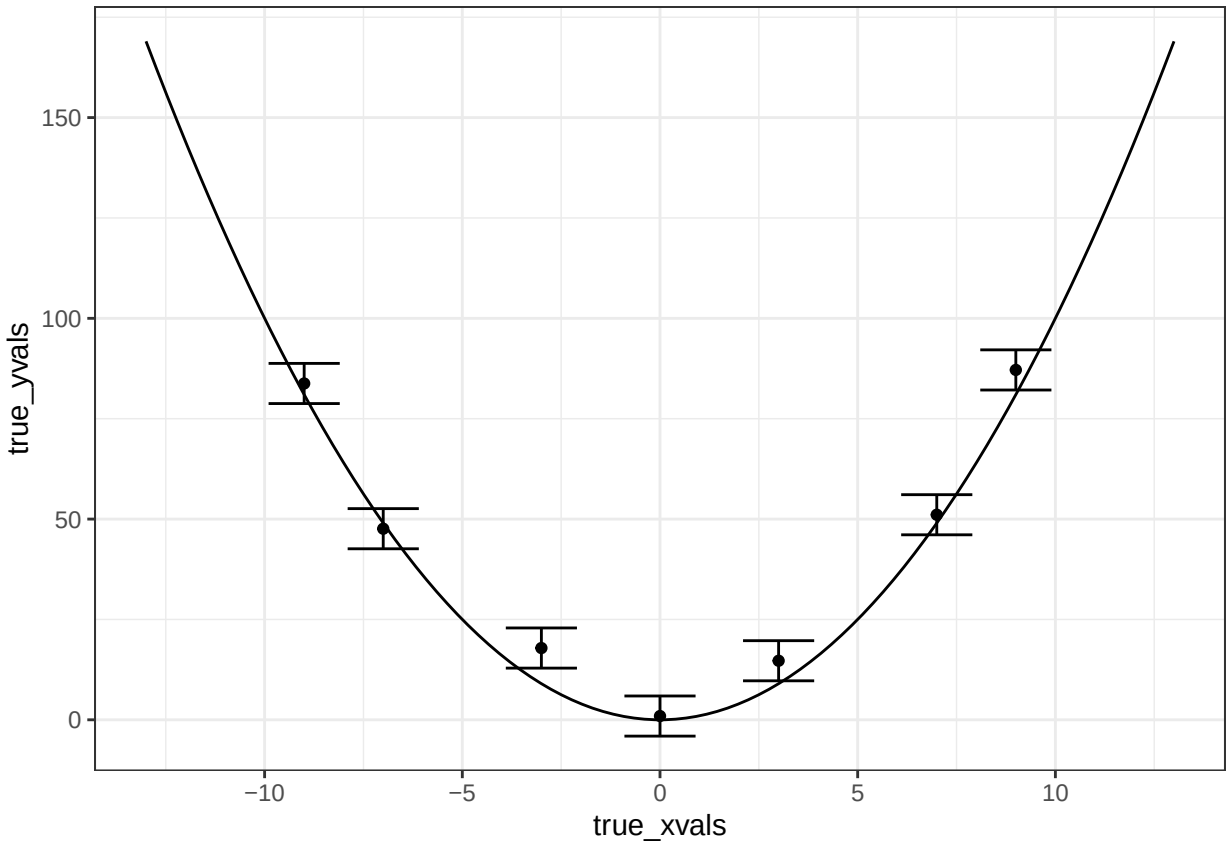
```
truefun <- function(x) { x^2 }
```

We further assume that we don't know this function but have made some measurements at locations x_1, x_2, etc. Due to imprecision in the measurement process, we did not observe $f(x_1)$, $f(x_2)$, ... but values $y_i = f(x_i) + \varepsilon_i$ with an unknown statistical error. In this notebook, we assume that we know that each error contribution ε_i was generated from a normal distribution with zero mean and standard deviation 5. Following we create the synthetic data according to these assumptions:

```
unc_obs <- 5
x_obs <- c(-9, -7, -3, 0, 3, 7, 9)
y_obs <- truefun(x_obs) + rnorm(length(x_obs), 0, unc_obs)
```

Before we will be moving to the construction of the Bayesian network, let's plot the data in comparison to the true function.

```
true_xvals <- seq(-13, 13, length=100)
true_yvals <- truefun(true_xvals)
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_line(aes(x=true_xvals, y=true_yvals))
ggp <- ggp + geom_errorbar(aes(x=x_obs, ymin=y_obs-unc_obs, ymax=y_obs+unc_obs))
ggp <- ggp + geom_point(aes(x=x_obs, y=y_obs))
ggp
```



Constructing the basic Bayesian network skeleton

Defining the nodes

Now we will be assembling one data table that contains the information about the nodes. To this end, we will define each node as its own data table and afterwards glue them all together to a single data table.

Node associated with the true function First, we define a node that is associated with the true function to be estimated. We discretize the function on a computational mesh of x-values and assume that values in-between mesh points can be obtained by linear interpolation. This is a simple and flexible construction, which can be made arbitrarily precise by increasing the number of mesh points.

The definition of the data table linked to the true function looks as follows:

```
truefun_dt <- data.table(
  NODE = "truefun",
  PRIOR = 0,
  UNC = 1e30,
  OBS = NA,
  X = seq(-13, +13, length=100)
)
```

The `NODE` field specifies the label for the node. With `PRIOR=0` we specify that the most likely value of the function without considering the data is zero. However, the huge prior uncertainty of `1e30` makes the prior uninformative.

Node associated with the synthetic datapoints Next, we make use of the variables associated with the synthetic data to create a data table for the observation node:

```
obs_dt <- data.table(
  NODE = "obs",
  PRIOR = rep(0, length(x_obs)),
  UNC = unc_obs,
  OBS = y_obs ,
  X = x_obs
)
```

Merging the individual data tables Finally, we merge the two data tables defined above to a single one and augment it with an index column:

```
node_dt <- rbindlist(list(truefun_dt, obs_dt))
node_dt[, IDX:=seq_len(.N)]
```

Creating the basic mapping

To obtain estimates of the function associated with the node `truefun`, we need to define a mapping between `truefun` and the observation node `obs`. We will be using an interpolation mapping to propagate the values from the `truefun` node to values that can be compared with the observed values of the `obs` node.

The following list contains all required information for the setup of the linear interpolation mapping. The precise meaning of the field names can be looked up by executing `?create_linearinterpol_map` in the R console to open the help page.

```
truefun_to_obs_mapdef <- list(
  mapname = "truefun_to_obs",
  maptype = "linearinterpol_map",
  src_idx = node_dt[NODE=="truefun", IDX],
  tar_idx = node_dt[NODE=="obs", IDX],
  src_x = node_dt[NODE=="truefun", X],
  tar_x = node_dt[NODE=="obs", X]
)
```

Putting everything together

The basic nodes and the essential link between the `truefun` node and the `obs` node are defined now. Later on, we will be adding additional nodes to introduce pseudo-observations of derivatives. Before that, let's put everything together and explore what Bayesian inference yields.

To do so, we first create a mapping object based on the mapping definition above:

```
basicmap <- create_map(truefun_to_obs_mapdef)
```

Now we construct the prior covariance matrix by making use of the information in the node data table:

```
U_prior <- Diagonal(x= node_dt$UNC^2)
```

We specify the covariance matrix to be diagonal because the additive error terms ε_i contributing to the observed values were generated from independent normal distributions. In other words, knowledge of the value of one ε_i would not help us in obtaining a better idea about the value of any another ε_j, which is reflected in zero off-diagonal elements.

The inference can be performed by

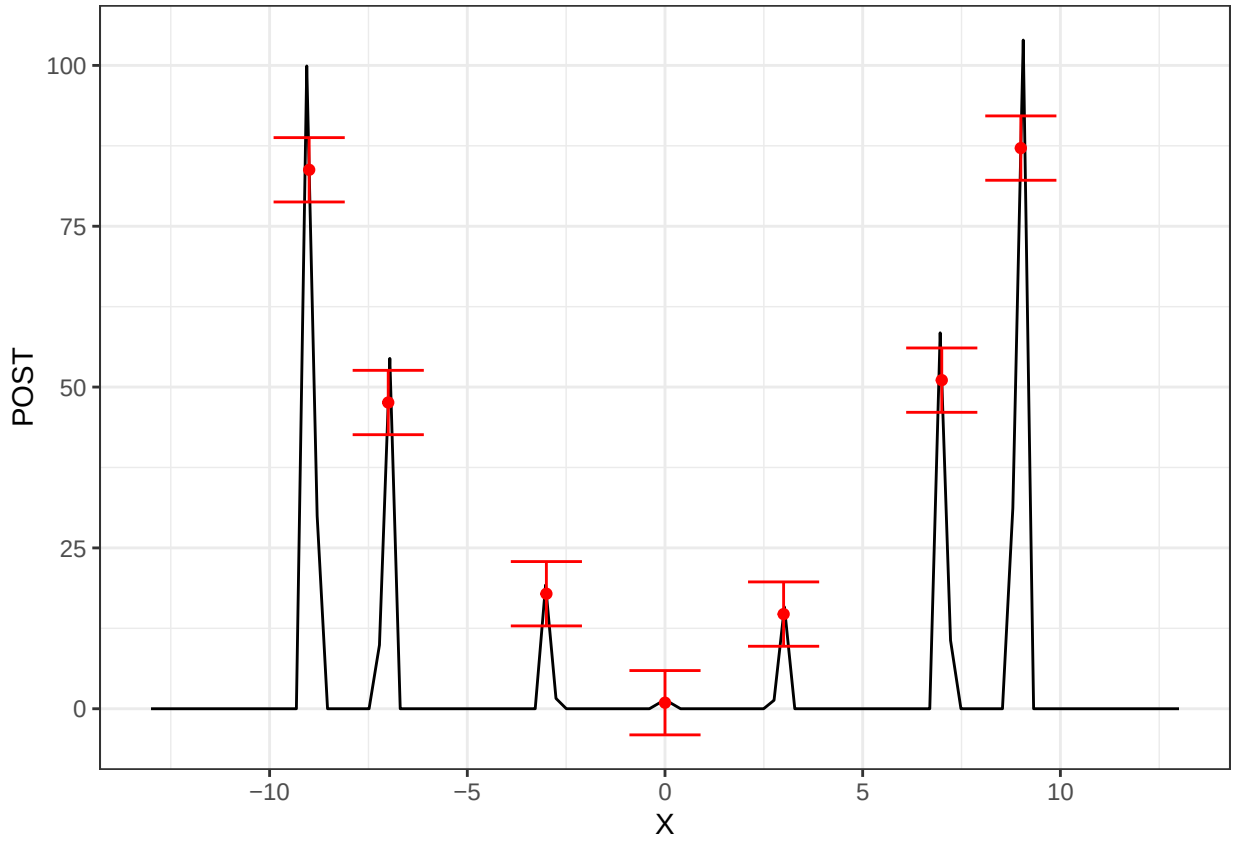
```
optres <- LMalgo(basicmap, zprior= node_dt$PRIOR, U=U_prior, obs=node_dt$OBS)
```

For convenience, we augment the node data table with the results of the Bayesian inference,

```
node_dt[, POST:=basicmap$propagate(optres$zpost)]
```

so that we can plot the most likely estimate of `truefun` in comparison with the observations:

```
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_line(aes(x=X, y=POST), data=node_dt[NODE=='truefun',])
ggp <- ggp + geom_errorbar(aes(x=X, ymin=OBS-UNC, ymax=OBS+UNC), data=node_dt[NODE=='obs'], col='red')
ggp <- ggp + geom_point(aes(x=X, y=OBS), data=node_dt[NODE=='obs'], col='red')
ggp
```



The inferred function looks very strange. It is certainly not resembling a parabola in any way. What happened here?

The reason for this shape is that we specified a diagonal covariance matrix as prior for the function values in `truefun`. With this prior, we are expressing the belief that knowing one function value at some x-value does not give us any information about the function values associated with x-values in vicinity. If we believe the function to exhibit a certain degree of smoothness, we would expect nearby function values not to be too different from the observed one.

We could go back and modify the prior covariance of the `truefun` node so that nearby function values become correlated, which is at the heart of Gaussian process regression. In this tutorial, however, we will be taking another route. We leave the prior covariance matrix of `truefun` as it is and add pseudo-observations of derivatives of this function instead.

Case study 1: Introducing a pseudo-observation on the first derivative

In our first case study, we are going to study the impact of pseudo-observations of the first derivative of the function associated with the node `truefun`. To this end, we construct a data table for a new node named `truefun_1st_deriv`:

```
truefun_1st_deriv_dt <- data.table(
  NODE = "truefun_1st_deriv",
  PRIOR = 0,
  UNC = 20,
  OBS = 0,
  X = node_dt[NODE=='truefun', head(X, n=-1)]
)
```

In this data table, we define a pseudo-observation of the first derivative for each x-value of the node `truefun` except the last one. The reason for excluding the last x-value being that the first derivative at each x-value is constructed based on a finite difference approximation that involves for each mesh point `x_i` also the next one, `x_{i+1}`. For details, consult the help, i.e., type `?create_derivative_map` in the R console. All observed values of the pseudo-observations are taken to be zero, as indicated by the `OBS` field. The field `UNC` contains the uncertainty (or standard deviation) of this observation. The zero value in the `PRIOR` field can be interpreted as a preference for a specific slope and is taken to be zero here as well. We shall explore its impact a bit later.

Now we create a new data table that combines the definition of the basic nodes discussed above with the new node associated with the second derivative:

```
ext_node_dt1 <- copy(node_dt)
ext_node_dt1[, POST := NULL]
ext_node_dt1[, IDX := NULL]
ext_node_dt1 <- rbindlist(list(ext_node_dt1, truefun_1st_deriv_dt))
ext_node_dt1[, IDX:=seq_len(.N)]
```

We also need to introduce the link from the node `truefun` to the node `truefun_to_1st_deriv`, for which we are going to use a first derivative mapping whose required parameters can be looked up by `?create_derivative_map`.

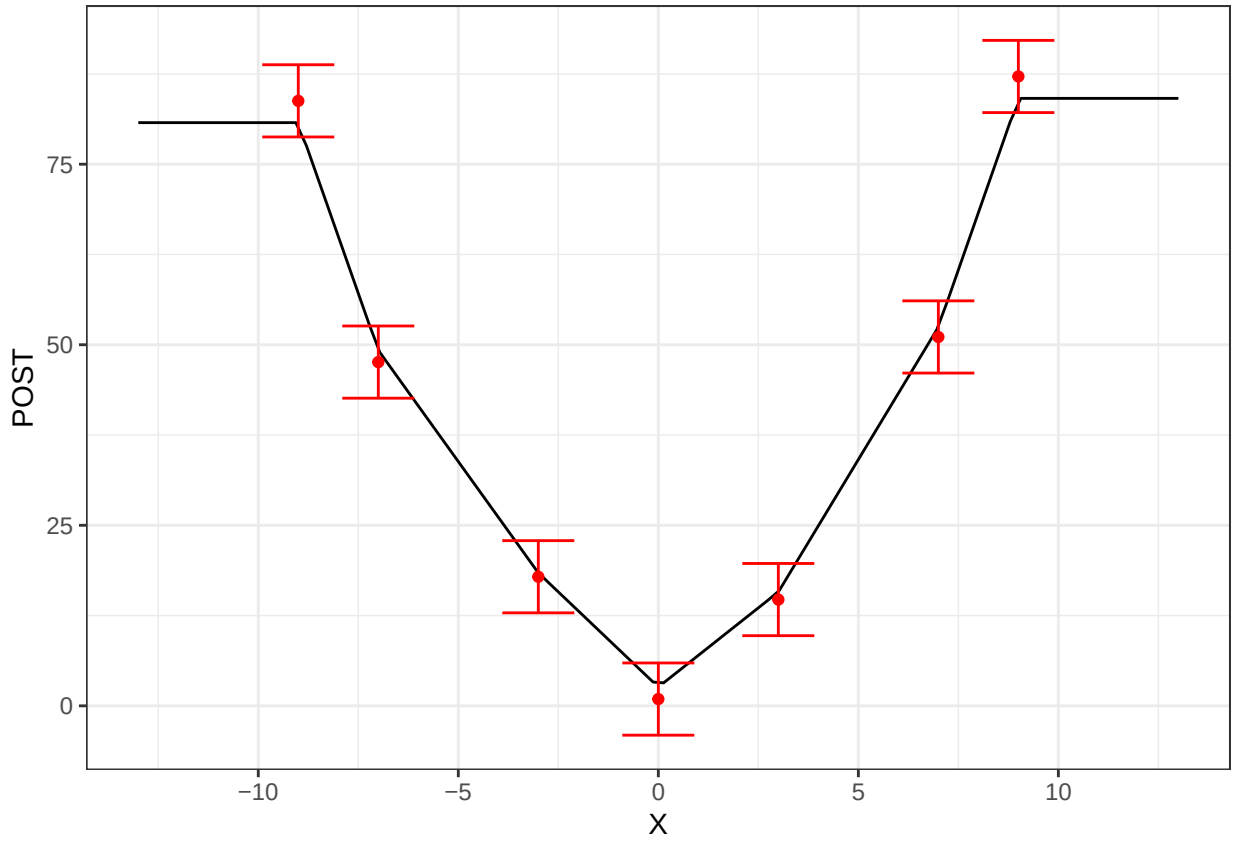
```
truefun_to_1st_deriv_mapdef <- list(
  maptype = "derivative_map",
  mapname = "truefun_to_1st_deriv",
  src_idx = ext_node_dt1[NODE=="truefun", IDX],
  tar_idx = ext_node_dt1[NODE=="truefun_1st_deriv", IDX],
  src_x = ext_node_dt1[NODE=="truefun", X],
  tar_x = ext_node_dt1[NODE=="truefun_1st_deriv", X]
)
```

Finally, the basic mapping from `truefun` to `obs` whose parameters are stored in `truefun_to_obs_mapdef` needs to be combined with the derivative mapping defined in `truefun_to_1st_deriv_mapdef`:

```
compmap1_def <- list(
  maptype = "compound_map",
  mapname = "notimportant",
  maps = list(truefun_to_obs_mapdef, truefun_to_1st_deriv_mapdef)
)
compmap1 <- create_map(compmap1_def)
```

As we have already discussed this step in more detail above, here is the code without any comment to do the Bayesian inference and plot the result:

```
U_prior <- Diagonal(x=ext_node_dt1$UNC^2)
optres <- LMalgo(compmap1, zprior=ext_node_dt1$PRIOR, U=U_prior, obs=ext_node_dt1$OBS)
ext_node_dt1[, POST:=compmap1$propagate(optres$zpost)]
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_line(aes(x=X, y=POST), data=ext_node_dt1[NODE=='truefun',])
ggp <- ggp + geom_errorbar(aes(x=X, ymin=OBS-UNC, ymax=OBS+UNC), data=ext_node_dt1[NODE=='obs'], col='re
ggp <- ggp + geom_point(aes(x=X, y=OBS), data=ext_node_dt1[NODE=='obs'], col='red')
ggp
```



Several important conclusions can be drawn from this plot. The pseudo-observations of the first derivative seem to favor solutions that connect observations by straight lines. This behavior is already a significant improvement over the case without pseudo-observations (we may say without regularization). If we recall that the spacing between mesh points is about `0.3` arbitrary units, therefore many dozens of mesh points lie between some of the observations, we realize that even though the pseudo-observations are independent in the second derivative space (are associated with a diagonal prior covariance matrix), they still introduce prior correlations between the function values themselves.

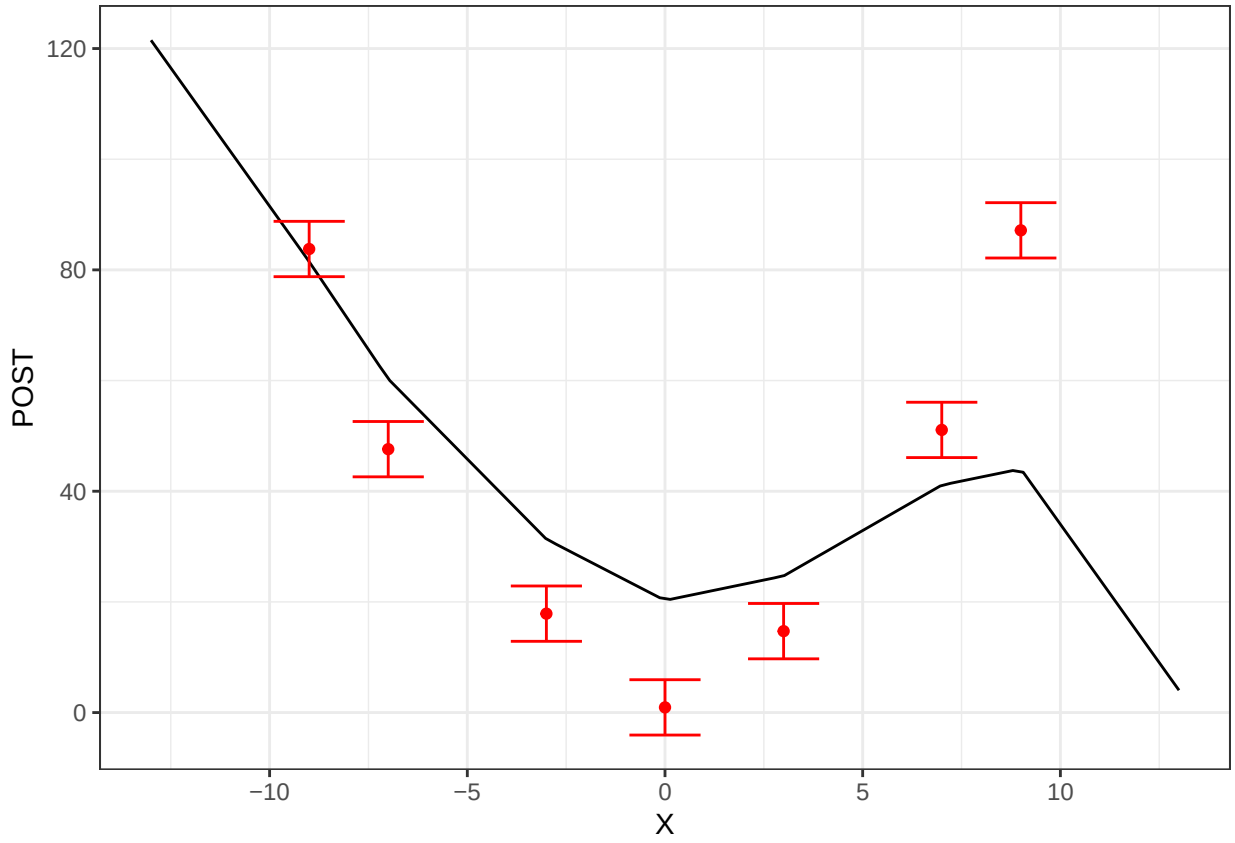
We also see that in the extrapolation region outside the range covered by the synthetic data points, the estimated function immediately changes to a horizontal line. This is due to the fact that the specification of `PRIOR=0` and `OBS=0` imposes a prior preference for horizontal lines. Because the pseudo-observations in the first derivative space are assumed to be uncorrelated, the prior doesn't penalize abrupt changes of the slope from one mesh point to another one. Furthermore, we are overfitting the data because the estimated curve runs to close to too many data points. According to statistics, we would expect that for 1/3 of the data points, the fitted curve should be outside the 1-sigma error bar.

Following up on the insights of the last paragraph, we are going to reduce the uncertainties of the pseudo-observations and adopt `PRIOR` values different from zero:

```
ext_node_dt1a <- copy(ext_node_dt1)
ext_node_dt1a[NODE=='truefun_1st_deriv', PRIOR:=10]
ext_node_dt1a[NODE=='truefun_1st_deriv', UNC:=5]
```

And again do the inference and plotting:

```
U_prior <- Diagonal(x=ext_node_dt1a$UNC^2)
optres <- LMalgo(compmap1, zprior=ext_node_dt1a$PRIOR, U=U_prior, obs=ext_node_dt1a$OBS)
ext_node_dt1a[, POST:=compmap1$propagate(optres$zpost)]
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_line(aes(x=X, y=POST), data=ext_node_dt1a[NODE=='truefun',])
ggp <- ggp + geom_errorbar(aes(x=X, ymin=OBS-UNC, ymax=OBS+UNC), data=ext_node_dt1a[NODE=='obs'], col='
ggp <- ggp + geom_point(aes(x=X, y=OBS), data=ext_node_dt1a[NODE=='obs'], col='red')
ggp
```



Even with a reduced uncertainty on pseudo-observations, the estimated curve is still composed of straight line segments between synthetic data points. However, due to the reduced uncertainty of the pseudo-observations,

we are now underfitting the data points as the estimated curve is in most cases outside the 1-sigma error bars.

The specification `PRIOR=10` biases the slope towards `-10`. This slope is immediately adopted by the estimated curve as soon as we enter the extrapolation range. However, we clearly see that this bias is also affecting the interpolation range. Please note that due to symmetry we also could have chosen `OBS=10` and kept `PRIOR=0` to achieve the same biasing effect.

It is left as an exercise for the interested notebook reader, to find a value for the uncertainties `UNC` and `PRIOR` of the pseudo-observations to get the best fit to the synthetic data.

Summarizing this case study, pseudo-observations of the first derivative favor solutions that are piecewise linear functions. The knot points of this piecewise linear function are located approximately at the locations of the data points.

Case study 2: Introducing a pseudo-observation on the second derivative

Now we are going to explore the impact of pseudo-observations of the second derivative of `truefun`. The basic steps to set up the Bayesian network are the same as for the case for the first derivative.

We define a data table for the node corresponding to the second derivative:

```
truefun_2nd_deriv_dt <- data.table(
  NODE = "truefun_2nd_deriv",
  PRIOR = 0,
  UNC = 5,
  OBS = 0,
  X = node_dt[NODE=='truefun', head(X[-1], n=-1)]
)
```

The second derivative at a mesh point x_i is computed by an expression involving finite differences to x_{i-1} and x_{i+1} so the first and last x-value of `truefun` are not included in the field `X` here. Doing so would raise an error later on in the setup of the mapping.

Then we combine this data table with the basic node data table:

```
ext_node_dt2 <- copy(node_dt)
ext_node_dt2[, POST := NULL]
ext_node_dt2[, IDX := NULL]
ext_node_dt2 <- rbindlist(list(ext_node_dt2, truefun_2nd_deriv_dt))
ext_node_dt2[, IDX:=seq_len(.N)]
```

We define the mapping from `truefun` to the node `truefun_2nd_deriv` (see `?create_derivative2nd_map` for an explanation of the parameters):

```
truefun_to_2nd_deriv_mapdef <- list(
  maptype = "derivative2nd_map",
  mapname = "truefun_to_2nd_deriv",
  src_idx = ext_node_dt2[NODE=="truefun", IDX],
  tar_idx = ext_node_dt2[NODE=="truefun_2nd_deriv", IDX],
  src_x = ext_node_dt2[NODE=="truefun", X],
  tar_x = ext_node_dt2[NODE=="truefun_2nd_deriv", X]
)
```

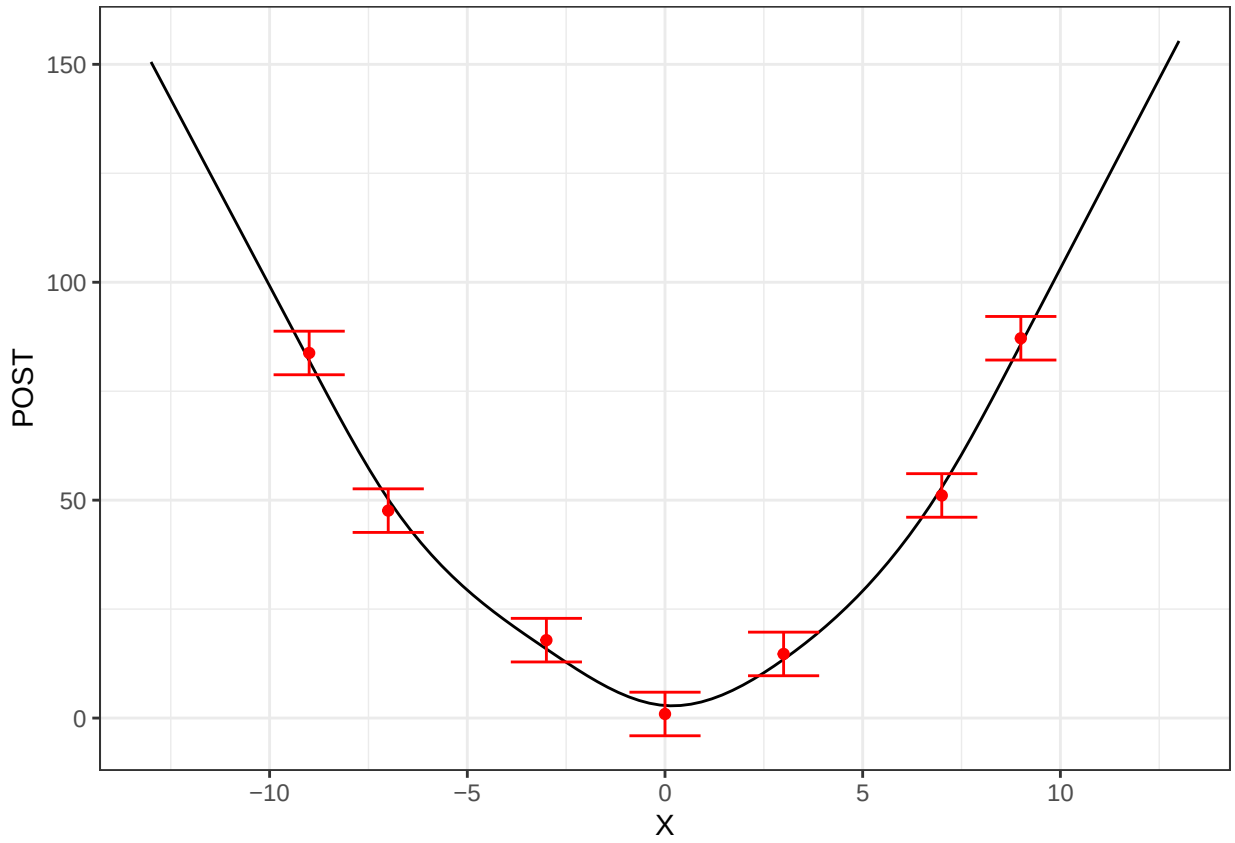
This new mapping is combined with the basic mapping to a compound mapping:

```
compmap2_def <- list(
  maptype = "compound_map",
  mapname = "notimportant",
  maps = list(truefun_to_obs_mapdef, truefun_to_2nd_deriv_mapdef)
```

```
)
compmap2 <- create_map(compmap2_def)
```

Inference and plotting is analogous to the previous case for the first derivative:

```
U_prior <- Diagonal(x=ext_node_dt2$UNC^2)
optres <- LMalgo(compmap2, zprior=ext_node_dt2$PRIOR, U=U_prior, obs=ext_node_dt2$OBS)
ext_node_dt2[, POST:=compmap2$propagate(optres$zpost)]
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_line(aes(x=X, y=POST), data=ext_node_dt2[NODE=='truefun',])
ggp <- ggp + geom_errorbar(aes(x=X, ymin=OBS-UNC, ymax=OBS+UNC), data=ext_node_dt2[NODE=='obs'], col='r
ggp <- ggp + geom_point(aes(x=X, y=OBS), data=ext_node_dt2[NODE=='obs'], col='red')
ggp
```



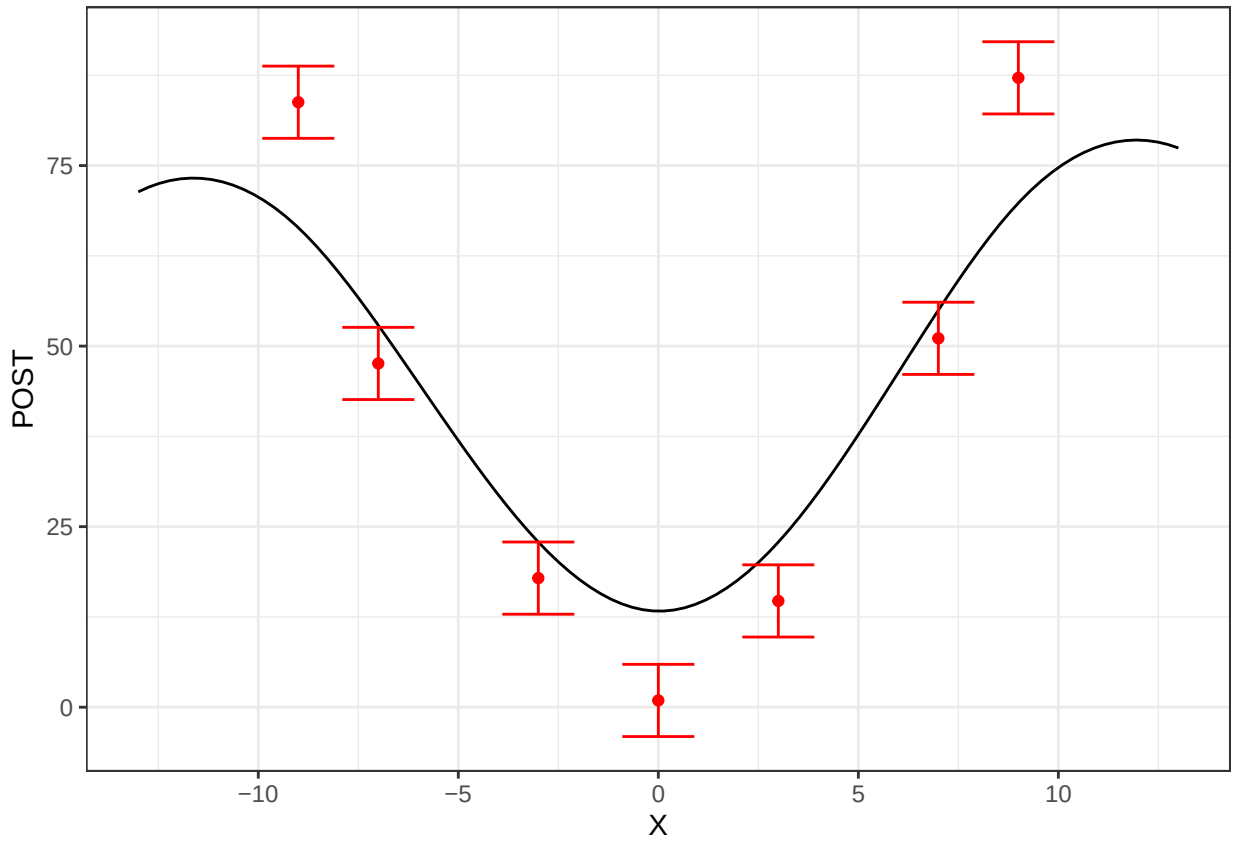
We see that pseudo-observations of the second derivative lead to a solution curve that smoothly changes its slope between data points in a way that locally resembles the shape of a parabola. We can draw from this investigation that the shape of the estimated function in the interpolation range (between data points) are straight lines for pseudo-observations of the first derivative and parabolic segments for the second derivative. Due to the `PRIOR=0` and `OBS=0` specifications, the prior is biased towards solutions with vanishing second derivatives, in other words straight lines. This preference is clearly visible in the extrapolation range, where the estimated curve immediately becomes a straight line after passing by the most outer data points.

Next, we are doing the same exercise as for the first derivative, which is decreasing the uncertainty of the pseudo-observations and using a non-zero value for the `PRIOR` field:

```
ext_node_dt2a <- copy(ext_node_dt2)
ext_node_dt2a[NODE=='truefun_2nd_deriv', PRIOR:=1]
ext_node_dt2a[NODE=='truefun_2nd_deriv', UNC:=1]
```

And again do the inference and plotting:

```
U_prior <- Diagonal(x=ext_node_dt2a$UNC^2)
optres <- LMalgo(compmap2, zprior=ext_node_dt2a$PRIOR, U=U_prior, obs=ext_node_dt2a$OBS)
ext_node_dt2a[, POST:=compmap2$propagate(optres$zpost)]
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_line(aes(x=X, y=POST), data=ext_node_dt2a[NODE=='truefun',])
ggp <- ggp + geom_errorbar(aes(x=X, ymin=OBS-UNC, ymax=OBS+UNC), data=ext_node_dt2a[NODE=='obs'], col='
ggp <- ggp + geom_point(aes(x=X, y=OBS), data=ext_node_dt2a[NODE=='obs'], col='red')
ggp
```



The segments between data points in the interpolation range are still given by parabolic shapes.
The non-zero `PRIOR` values introduces a prior biased to solution curves where the second derivative is 1. Immediately after entering the extrapolation region, the estimated curve adopts the prior preference and resembles the shape of a parabola.

Case study 3: Introducing a pseudo-observation on the third derivative

In the last case study of this tutorial, we are going to explore the impact of pseudo-observations on the third derivative. We may already anticipate the result regarding the features of the solution. However, this case study is also interesting because a third derivative mapping is not implemented at the time of writing (Jan 2023) in the *nucdataBaynet* package. Therefore we are going to implement this case by nesting two additional nodes corresponding to the second derivative and third derivative respectively.

The definition of the node `truefun_2nd_deriv` looks like this:

```
truefun_2nd_deriv_det_dt <- data.table(
  NODE = "truefun_2nd_deriv",
  PRIOR = 0,
  UNC = 0,
  OBS = NA,
  X = node_dt[NODE=='truefun', head(X[-1], n=-1)]
)
```

There are two important differences to `truefun_2nd_deriv_dt` defined at the beginning of the second case study. First, the column `OBS` is filled with `NA` values because we assume that the second derivative has not been observed here and we don't want to specify any pseudo derivatives for it. Second, the values in the column `UNC` are set to zero, which means that we establish a deterministic mapping between `truefun` and `truefun_2nd_deriv`.

Additionally, we define a new node `truefun_3rd_deriv` that will represent the first derivative of the `truefun_2nd_deriv` node, and therefore also corresponds to the third derivative of `truefun`:

```
truefun_3rd_deriv_dt <- data.table(
  NODE = "truefun_3rd_deriv",
  PRIOR = 0,
  UNC = 20,
  OBS = 0,
  X = node_dt[NODE=='truefun', head(X[-1], n=-2)]
)
```

The introduction of a pseudo-observations for this node, the value of `OBS` is not `NA` anymore but chosen to be zero. The uncertainty of the pseudo observation is given by `UNC`. Importantly, due to the way how the second and first derivative mappings are implemented (`?create_derivative_map` and `?create_derivative2nd_map`), we need to be careful with the specification of the x-values in the column `X`.

Now we combine again all the data tables to a full node data table:

```
ext_node_dt3 <- copy(node_dt)
ext_node_dt3[, POST := NULL]
ext_node_dt3[, IDX := NULL]
ext_node_dt3 <- rbindlist(list(ext_node_dt3, truefun_2nd_deriv_det_dt, truefun_3rd_deriv_dt))
ext_node_dt3[, IDX:=seq_len(.N)]
```

The definition of the mappings between nodes is analogous to the previous case studies. Here is the mapping defintion to go from `truefun` to the second derivative node `truefun_2nd_deriv_det`:

```
truefun_to_2nd_deriv_det_mapdef <- list(
  maptype = "derivative2nd_map",
  mapname = "truefun_to_2nd_deriv_det",
  src_idx = ext_node_dt3[NODE=="truefun", IDX],
  tar_idx = ext_node_dt3[NODE=="truefun_2nd_deriv", IDX],
  src_x = ext_node_dt3[NODE=="truefun", X],
  tar_x = ext_node_dt3[NODE=="truefun_2nd_deriv", X]
)
```

The next mapping definition establishes the link from the second derivative `truefun_2nd_deriv_det` to the node with the third derivative by a (first) derivative mapping (`?create_derivative_map`):

```
deriv2nd_det_to_deriv3rd_mapdef <- list(
  maptype = "derivative_map",
  mapname = "2nd_deriv_det_to_3rd_deriv",
  src_idx = ext_node_dt3[NODE=="truefun_2nd_deriv", IDX],
  tar_idx = ext_node_dt3[NODE=="truefun_3rd_deriv", IDX],
```

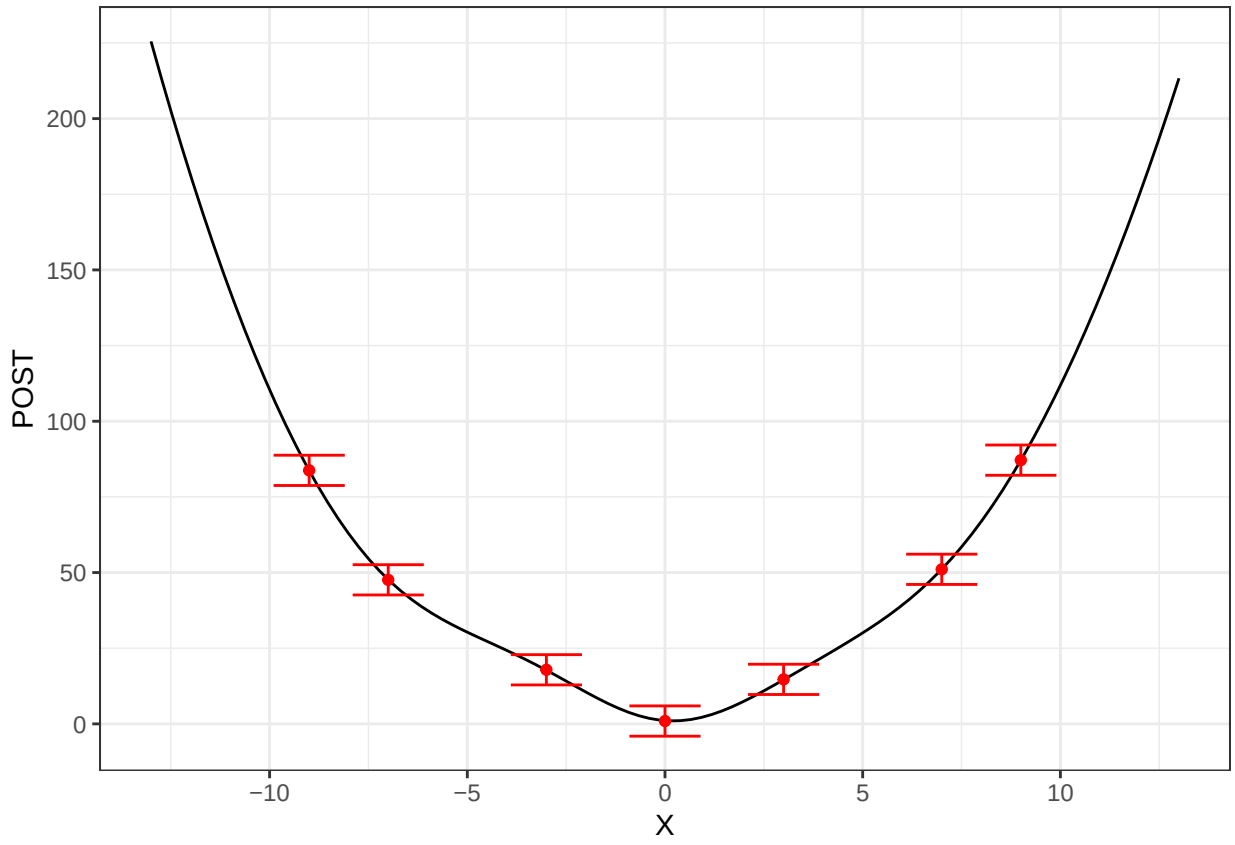
```
  src_x = ext_node_dt3[NODE=="truefun_2nd_deriv", X],
  tar_x = ext_node_dt3[NODE=="truefun_3rd_deriv", X]
)
```

As usual, let's create the compound mapping:

```
compmap3_def <- list(
  maptype = "compound_map",
  mapname = "notimportant",
  maps = list(truefun_to_obs_mapdef,
              truefun_to_2nd_deriv_det_mapdef,
              deriv2nd_det_to_deriv3rd_mapdef)
)
compmap3 <- create_map(compmap3_def)
```

Inference and plotting can be done analogously to the previous case studies:

```
U_prior <- Diagonal(x=ext_node_dt3$UNC^2)
optres <- LMalgo(compmap3, zprior=ext_node_dt3$PRIOR, U=U_prior, obs=ext_node_dt3$OBS)
ext_node_dt3[, POST:=compmap3$propagate(optres$zpost)]
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_line(aes(x=X, y=POST), data=ext_node_dt3[NODE=='truefun',])
ggp <- ggp + geom_errorbar(aes(x=X, ymin=OBS-UNC, ymax=OBS+UNC), data=ext_node_dt3[NODE=='obs'], col='re
ggp <- ggp + geom_point(aes(x=X, y=OBS), data=ext_node_dt3[NODE=='obs'], col='red')
ggp
```



Looks nice, doesn't it? The solution is even smoother than for the case with pseudo-observations of the second derivative. Furthermore, the estimated curve is of parabolic shape even in the extrapolation region. The

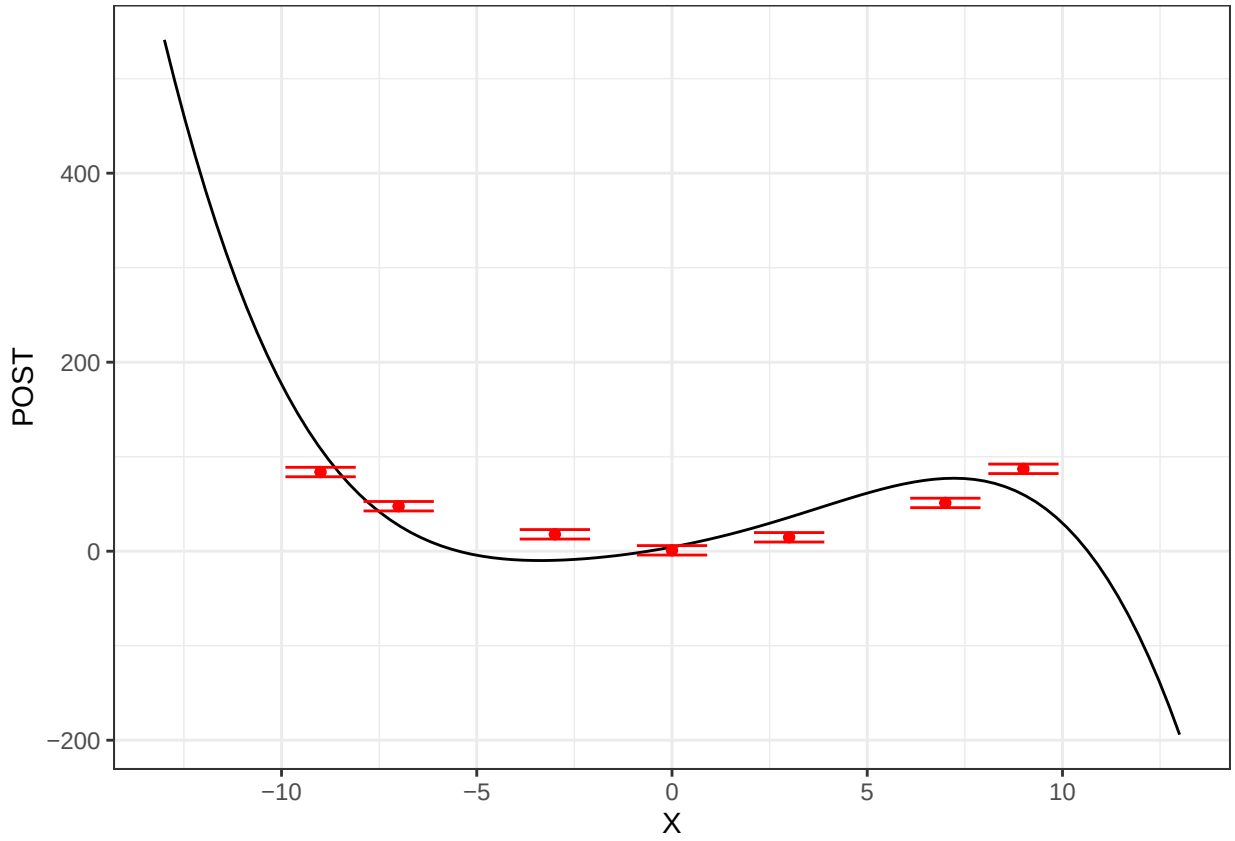
reason being that the pseudo-observations on the third derivative induce the bias that the second derivative should be constant, hence the underlying function biased towards being a parabola.

For completeness, we will be also reducing the uncertainty and introduce a non-zero bias on the third derivative, analogously to the previous cases:

```
ext_node_dt3a <- copy(ext_node_dt3)
ext_node_dt3a[NODE=='truefun_3rd_deriv', PRIOR:=2]
ext_node_dt3a[NODE=='truefun_3rd_deriv', UNC:=0.5]
```

Here is the result:

```
U_prior <- Diagonal(x=ext_node_dt3a$UNC^2)
optres <- LMalgo(compmap3, zprior=ext_node_dt3a$PRIOR, U=U_prior, obs=ext_node_dt3a$OBS)
ext_node_dt3a[, POST:=compmap3$propagate(optres$zpost)]
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_line(aes(x=X, y=POST), data=ext_node_dt3a[NODE=='truefun',])
ggp <- ggp + geom_errorbar(aes(x=X, ymin=OBS-UNC, ymax=OBS+UNC), data=ext_node_dt3a[NODE=='obs'], col='
ggp <- ggp + geom_point(aes(x=X, y=OBS), data=ext_node_dt3a[NODE=='obs'], col='red')
ggp
```



Summary

This tutorial explored the impact of pseudo-observations of derivatives of various order in the Bayesian inference. To this end, synthetic data were generated from a parabola and a simple Bayesian network constructed that connected a node `truefun` associated with the unknown function to a node `obs` associated with the synthetic data points.

The Bayesian network was then extended in three case studies to accommodate pseudo-observations of the first, second and third derivative, respectively. The impact of the pseudo-observations was clearly visible and very easy to understand:

1. Pseudo-observations on the first derivative biased the estimated curve towards a piecewise linear function in the interpolation region and towards a horizontal line in the extrapolation region.
2. For the case with pseudo-observations on the second derivative, the estimated curve was biased towards a piecewise parabolic function in the interpolation region and to straight lines in the extrapolation region .
3. Finally, pseudo-observations on the third derivative led to an even smoother solution curve. By considering the results of the other two cases, we can reason by induction that the estimated curve is a piecewise cubic function in the interpolation range and has a parabolic shape in the extrapolation region.

We started out in each case study with pseudo-observations with zero values but also investigated how pseudo-observations with non-zero values would impact solutions (we changed `PRIOR` instead of `OBS` but due to symmetry this has the same effect on the solution up to a sign).

In summary, pseudo-observations of derivatives are a powerful approach to introduce biases in statistical analyses whose impact can be intuitively understood. The abstraction of nodes and links put forward by the Bayesian network paradigm makes it very easy to integrate pseudo-observations in a statistical analysis.