Simple linear regression with the nucdataBaynet package

Georg Schnabel

October 20, 2021

In this document we show how linear regression can be done with the *nucdataBaynet* package with datapoints being affected by statistical errors and a common absolute normalization error. The purpose of this document is to demonstrate the general workflow with the *nucdataBaynet* package at a specific example, which includes the steps:

1. Define a data table with the information about the nodes including prior estimates and uncertainties
2. Define the mappings, i.e., functional relationships, between the nodes
3. Perform inference in the Bayesian network to obtain posterior estimates, uncertainties and covariance matrix blocks of variables of interest

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. In order to create the covariance matrix for the Levenberg-Marquardt algorithm, we rely on functionality of the *Matrix* package. The Bayesian network inference algorithms are implemented in the *nucdataBaynet* package.

These packages can be loaded by

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

Defining the node datatable

First we need to define a datatable that contains the information about the nodes including their prior estimates and uncertainties. We also need to specify which nodes have been observed by providing values for them. We have three nodes in this example, which are:

1. The computational mesh
2. The experimental data
3. The normalization error

The computational mesh and the normalization error represent independent nodes, i.e., they don't have any parent nodes, whereas the experimental data is given as a dependent node. All dependent nodes are functions of the values of other nodes and have an additional parentless node associated with them. This is the reason why we did not introduce the statistical error as a node in the list above because we use the parentless node associated with the experimental data to model it.

We are going to define the datatable for each of the nodes individually and then merge them together. The specification of the node with the computational mesh can be created by

```
linreg_dt <- data.table(
  NODE = "mesh",
  PRIOR = c(0, 0),
  UNC = c(1e4, 1e4),
  OBS = c(NA, NA),
  ENERGY = c(0, 10)
)
```

The first field *NODE* establishes the name of this node, which is simply called *mesh* here. The field *PRIOR* contains the prior estimate of the variables associated with this node and the field *UNC* the respective prior uncertainty. We use a large prior uncertainties to impose an uninformative prior. The field *OBS* allows to specify observed values associated with the node. As we have not observed the values at the mesh locations given in *ENERGY*, we assign NA (=Not available) values.

Next we define the node associated with the experimental data:

```
expdata_dt <- data.table(
  NODE = "expdata",
  PRIOR = c(0, 0, 0),
  UNC = c(1, 2, 3),
  OBS = c(5, 7, 9),
  ENERGY = c(2, 4, 8)
)
```

This node will be a dependent node. For a dependent node, the prior estimate and uncertainty given in *PRIOR* and *UNC* respectively, refer to the parentless node attached to this dependent node. As we model statistical errors here, we expect them a priori to be zero in average, with explains the specification of *PRIOR*. In this example, we assume the uncertainty associated with the three experimental data points to be different, which is reflected in the specification of *UNC*. We deal with observed experimental data, hence we know the values of the dependent node, which is reflected in the specification of *OBS*. The *ENERGY* field indicates at which energies the values were measured.

The node linked to the absolute normalization error is defined as follows:

```
normerr_dt <- data.table(
  NODE = "normerr",
  PRIOR = 0,
  OBS = NA,
  UNC = 0.5
)
```

This node is an independent node because it is not a function of any other node. We assume a prior uncertainty of 0.5 for the normalization error.

Now we need to glue together these individual specification to a single datatable:

```
node_dt <- rbindlist(list(linreg_dt, expdata_dt, normerr_dt), fill = TRUE)
node_dt[, IDX := seq_len(.N)]
```

The first line glues the individual datatables to a single one. The argument *fill=TRUE* means that missing columns in one datatable will be associated with *NA* values. The second line introduces a column *IDX* with an index running from 1 to the number of rows to establish an order among the rows. The created node datatable looks like this:

```
##       NODE PRIOR   UNC OBS ENERGY IDX
## 1:    mesh     0 1e+04  NA      0   1
## 2:    mesh     0 1e+04  NA     10   2
## 3: expdata     0 1e+00   5      2   3
## 4: expdata     0 2e+00   7      4   4
## 5: expdata     0 3e+00   9      8   5
## 6: normerr     0 5e-01  NA     NA   6
```

Defining the mappings

The node datatable in the previous step serves as the basis to create the mappings between nodes. We will establish two mappings. The first one is linear interpolation from the node *mesh* to the node *expdata*. The second mapping will be responsible to distribute the normalization error to the experimental datapoints. Finally we are going to merge these two mappings to a so-called *compound mapping*.

The mapping specification to perform linear interpolation from the *mesh* node to the *expdata* node is given by:

```r
linmap_spec <- list(
  mapname = "mylinmap",
  maptype = "linearinterpol_map",
  src_idx = node_dt[NODE=="mesh", IDX],
  tar_idx = node_dt[NODE=="expdata", IDX],
  src_x = node_dt[NODE=="mesh", ENERGY],
  tar_x = node_dt[NODE=="expdata", ENERGY]
)
```

The first field *mapname* provides a name to the individual mapping instance. The field *maptype* defines the type of the mapping and identifies the relevant mapping handler responsible for the mapping. The string *linearinterpol_map* specifies that linear interpolation should be used to interpolate from the source mesh to the target mesh. The field *src_idx* contains the indices associated with the source mesh and the field *tar_idx* of the target mesh. The indices are retrieved by query operation on the *node_dt* datatable created in the previous section. The fields *src_x* and *tar_x* define the x-values of the source and target mesh respectively. They are also retrieved from the *node_dt* datatable. Information about linear interpolation mappings can be retrieved by typing `?create_linearinterpol_map` in the R console.

Now we can define the mapping of the normalization error:

```r
normerrmap_spec <- list(
  mapname = "mynormerr",
  maptype = "normerr_map",
  src_idx = node_dt[NODE=="normerr", IDX],
  tar_idx = node_dt[NODE=="expdata", IDX],
  src_feat = 1,
  tar_feat = rep(1, node_dt[NODE=="expdata", .N])
)
```

The string *normerr_map* in the field *maptype* indicates the mapping of an absolute normalization error. The fields *src_idx* and *tar_idx* for the specification of source and target indices, respectively, are again determined by a query operation on the *node_dt* datatable. The fields *src_feat* and *tar_feat* establish the association of normalization errors to experiments. These values may contain identification numbers or strings of experimental datasets. In this case we have only a single normalization error so we can hard-code an identification number of one in *src_feat*. To specify that the normalization error should be distributed to all datapoints, we create a vector of ones with a size determined by the number of datapoints. More information on normalization error mappings can be found by typing `?create_normerr_map` in the R console.

Finally, we need to combine the individual mapping specifications to a so-called compound mapping specification, which is done by:

```
compmap_spec <- list(
  mapname = "mycompmap",
  maptype = "compound_map",
  maps = list(linmap_spec, normerrmap_spec)
)
```

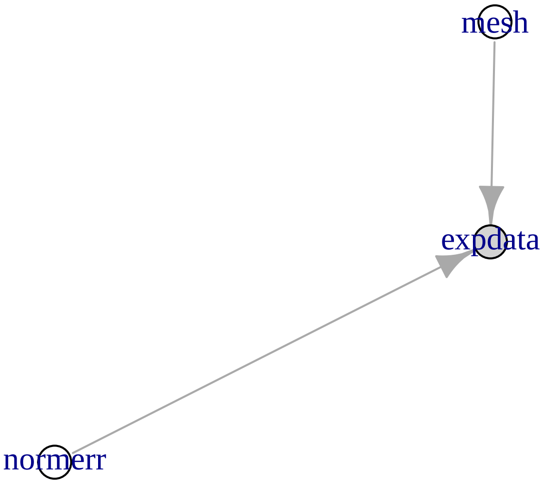
The individual mapping specifications are provided as a list in the field *maps*. For more information on compound mappings and their creation, type `?create_compound_map` in the R console.

The variable `compmap_spec` contains just the specification of the mapping and we need to create a mapping object from it:

```
compmap <- create_map(compmap_spec)
```

The *node_dt* datatable and the *compmap* object fully characterize the Bayesian network. We can plot it now by executing:

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



We can verify that the *expdata* node is indeed a function of the nodes *mesh* and *normerr*. The parentless node associated with *expdata* that contains the statistical error is not displayed, but it should always be kept in mind that every dependent node contains a parentless node directly attached to it.

Inference in the Bayesian network

To perform inference, i.e., obtain a MAP estimate, we need to pass the prior estimates, the prior covariance matrix, and values of observed nodes as arguments to the *LMalgo* function. The prior covariance matrix can be constructed by

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

The prior estimates and observed values are already defined in the *PRIOR* and *OBS* column, respectively, of the *node_dt* datatable. The posterior estimates can be obtained by invoking the LM algorithm:

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

More information on the LMalgo function can be retrieved by typing `?LMalgo`. The variable optres is a list with the posterior estimates in *zpost* and some other fields with additional information:

```
## $zpost
## [1]  3.653062e+00  1.079592e+01 -8.163269e-02  4.897962e-01 -3.673460e-01
## [6]  3.612245e-08
##
## $init_val
## [1] 46.25
##
## $final_val
## [1] 0.08163395
##
## $numiter
## [1] 10
```

Now we can augment the *node_dt* datatable by the found MAP estimate:

```
node_dt[, POST := optres$zpost]
```

The values in *zpost* are the posterior estimates of variables associated with independent nodes and the values of the parentless nodes associated with depedent nodes. To obtain the posterior estimates of the variables associated with the dependent nodes, we need to invoke the propagate function:

```
node_dt[, PRED := compmap$propagate(optres$zpost)]
```

Before we plot the regression result, let us compute a posterior uncertainty band.

```
sel_idcs <- node_dt[NODE=="mesh", IDX]
U_post <- get_posterior_cov(compmap, optres$zpost, U_prior, node_dt$OBS, sel_idcs, sel_idcs)
```

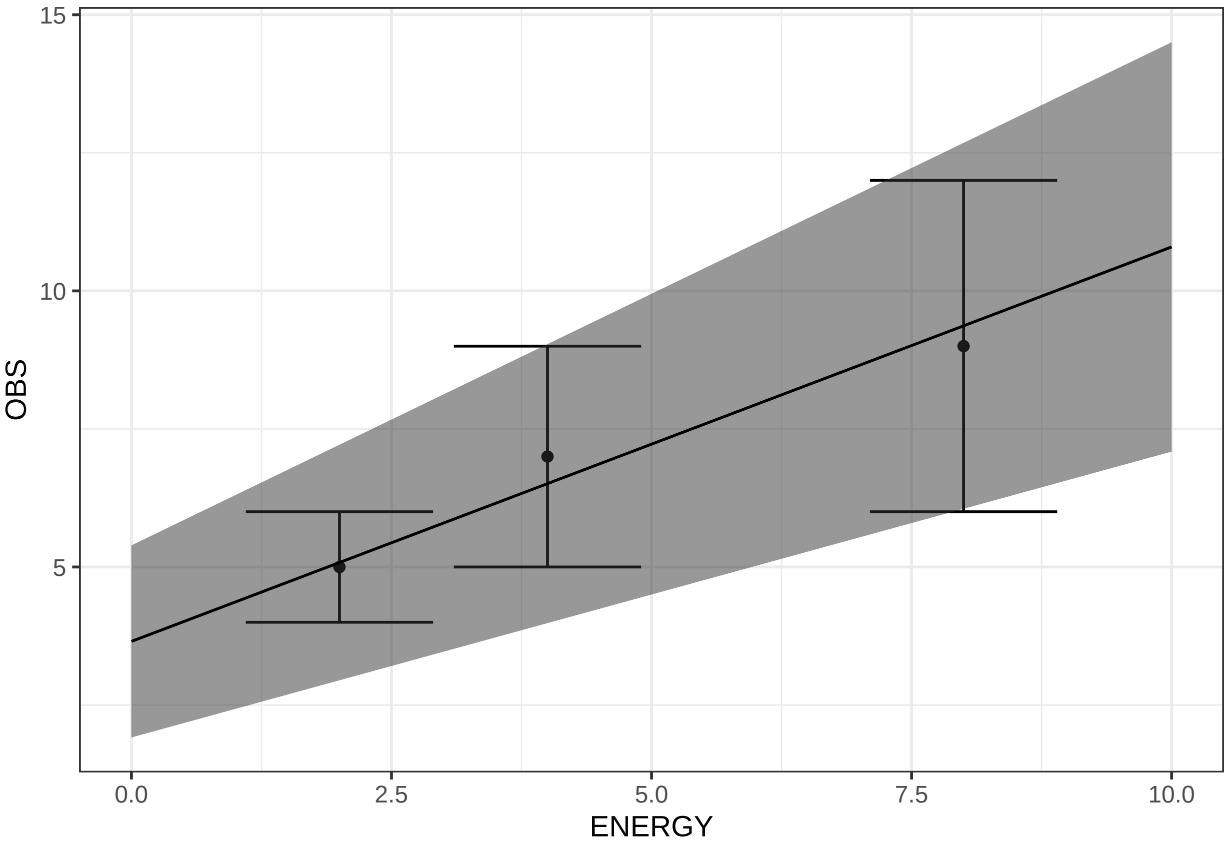
We extract the diagonal of the posterior covariance matrix and take the square root to obtain the posterior uncertainties:

```
post_unc <- sqrt(diag(U_post))
node_dt[sel_idcs, UNC_POST := post_unc]
```

Plotting the results

The results can be conveniently displayed by relying on functionality of the `ggplot2` package.

```
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_point(aes(x=ENERGY, y=OBS), data=node_dt[NODE=="expdata"])
ggp <- ggp + geom_errorbar(aes(x=ENERGY, ymin=OBS-UNC, ymax=OBS+UNC), data=node_dt[NODE=="expdata"])
ggp <- ggp + geom_ribbon(aes(x=ENERGY, ymin=PRED-UNC_POST, ymax=PRED+UNC_POST), alpha=0.5, data=node_dt
ggp <- ggp + geom_line(aes(x=ENERGY, y=PRED), data=node_dt[NODE=="mesh"])
print(ggp)
```



Summary

This document discussed the creation of a simple Bayesian network with three nodes given by a computational mesh, experimental data and a normalization error. In the first step a *node_dt* datatable was created summarizing the information about the nodes. In the second step, mappings were established between the nodes by using query operation on *node_dt*. Afterwards, Bayesian inference was performed on the Bayesian network by relying on a customized Levenberg-Marquardt algorithm, which amounted to simple linear regression in this example. Finally, the results were displayed as a plot.

The purpose of this document was to demonstrate the the essential steps to create a Bayesian networks and perform inference in them. These steps remain the same for larger networks with nested and non-linear relationships, only the number of nodes and mappings that need to be defined increases.