Gaussian process regression with the nucdataBaynet package

Georg Schnabel

2023-01-20

This R Markdown document shows the setup of a Bayesian network with the *nucdataBaynet* package to perform Gaussian process regression using a set of data points as input. It is assumed that each data point is affected by a statistical error and that all data points share a common normalization error. The example data are taken from the EXFOR database and represent measurements of the neutron-induced total cross section of Fe-56 for incident energies between 1.0 and 1.1 MeV.

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. We will also be using the *ggplot2* package to plot experimental data and the estimated function. Finally, we also load the `igraph` package to plot the Bayesian network.

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

We are introducing two variables `Emin` and `Emax` here for controlling the range of incident energies. The computational mesh will be limited to this energy range and experimental data points outside this range will be discarded. The energies are specified in MeV.

```
Emin <- 1
Emax <- 1.1
```

Loading the data

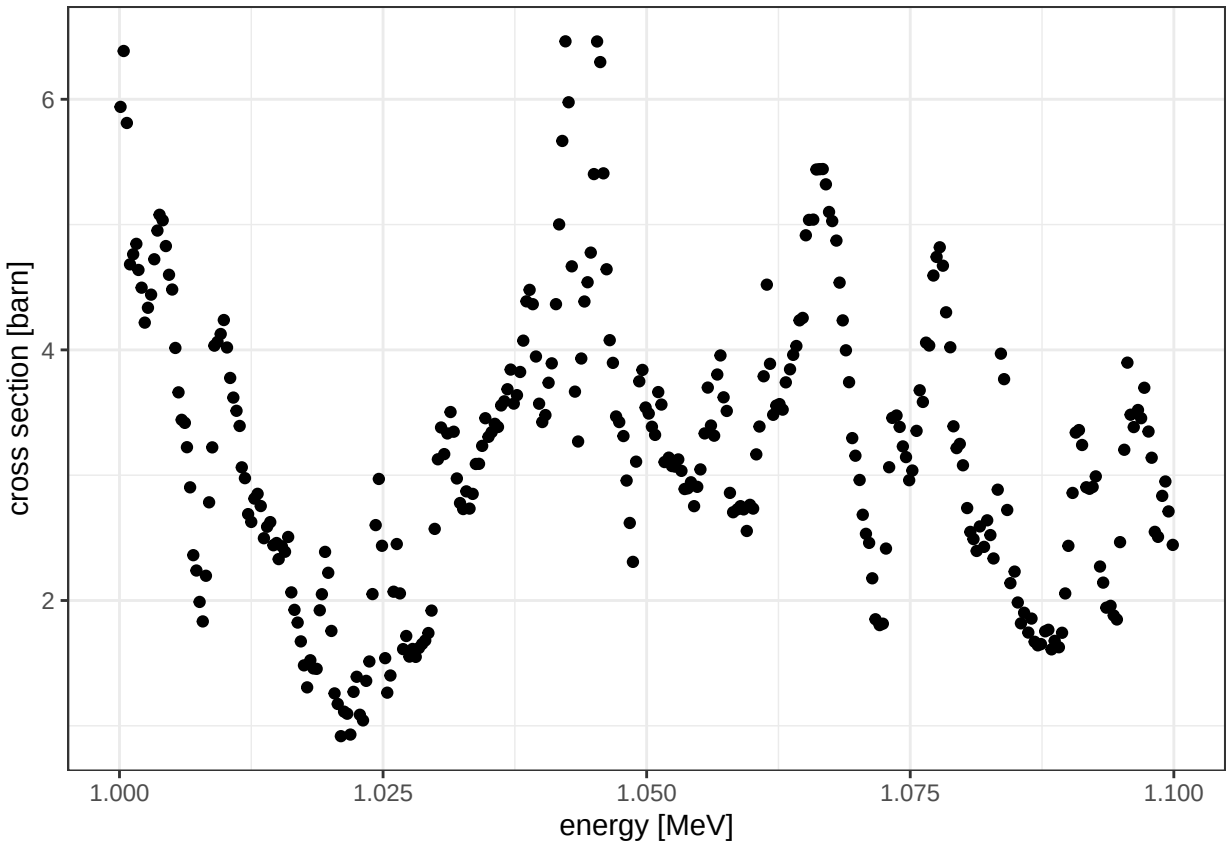
The experimental data have been prepared as a csv file along with this document. To load the dataset of Cornelis et al. (EXFOR accession number 22316001), we can use the following code:

```
rawexpdata <- fread(file.path("data", "22316.csv"))
rawexpdata$EN <- rawexpdata$EN / 1e6
rawexpdata <- rawexpdata[EN >= Emin & EN <= Emax]
```

The second instruction converts the energy column originally given in eV to MeV and the third instruction discards data outside the defined energy interval.

To get an impression of the data, let's plot them.

```
ggp <- ggplot() + theme_bw() + geom_point(aes(x=EN, y=DATA), data=rawexpdata)
ggp <- ggp + xlab("energy [MeV]") + ylab("cross section [barn]")
ggp
```



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.

Nodes associated with computational cross section mesh

First, we define a node that is associated with the cross section function to be estimated. This function is characterized by the cross sections on a computational mesh of incident energies. Linear interpolation will be used to compute values in-between the energies of the mesh.

```
sigma_dt <- data.table(
  NODE = "sigma",
  PRIOR = 0,
  UNC = 0,
  OBS = NA,
  ENERGY = seq(Emin, Emax, by=1e-4)
)
```

The name of the node is provided in the `NODE` field. We assume as prior best estimate a vector of zeros stored in the `PRIOR` column. We set the `UNC` column to zero because we will construct the covariance matrix of this node using a covariance function of a Gaussian process. The `OBS` column is given by `NA` values as we do not have any measured values for this node. Finally, the `ENERGY` column defines the energies associated with the cross sections on the computational mesh. Using a spacing of `1e-4` MeV (10 eV) between mesh points yields a mesh with 1001 points.

As an aside, persons familiar with Gaussian process regression may find the indexing of the input space by introducing a mesh unusual. The philosophy of the *nucdataBaynet* package is that functions are always

represented by a finite number of node points covering the full range of interest, i.e., training points and query/test points. The mesh points do not need and usually do not perfectly coincide with the training and test points, as linear interpolation is used to obtain the function values in-between mesh points. This construction has advantages if several nodes representing different functions are linked due to linear constraints on their values. Also, by making the mesh dense enough, any sufficiently smooth Gaussian process can be arbitrarily well approximated.

There are many possible types of Gaussian process kernels, such as the squared exponential. Here, we will be using the squared exponential as an example. At the time of writing, the *nucdataBaynet* package does not contain covariance functions, so the user has to define them herself or load them from another package. Here we define the squared exponential covariance function ourselves:

```
covfun <- function(x, y, delta=1, lambda=1, nugget=1e-6) {
  delta^2 * exp(-1/lambda^2 * outer(x, y, `-`)^2) +
    outer(x, y, `==`) * nugget
}
```

We will be using this covariance function later in this tutorial to construct the covariance matrix of the node `sigma` defined above.

Nodes associated with experimental data

Now we need to define the nodes to capture the information about the experimental data, which also includes statistical and systematic uncertainties of the measurements.

The experimental data we are going to use are already available in the variable `rawexpdata`. As above, we create a data table with the information about the node and use the information in `rawexpdata` to assign the measured values to the column `OBS` and the associated statistical uncertainty to the column `UNC`. The incident energies of the data points are stored in the column `ENERGY`:

```
expdata_dt <- data.table(
  NODE = "expdata",
  PRIOR = rep(0, nrow(rawexpdata)),
  UNC = rawexpdata[["ERR-S"]],
  OBS = rawexpdata[["DATA"]],
  ENERGY = rawexpdata[["EN"]]
)
```

It is very uncommon that data points obtained in one experiment have only independent error contributions. For instance, any calibration error of the detector will have an impact on all the data points.

Therefore we are going to introduce a normalization error that affects all the data points in the same way, which means as an offset. The corresponding data table can be defined like this:

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

This node comprises a single variable. The value in `PRIOR` is zero, which is usually the most reasonable assumption for nodes that capture experimental errors. If we had known that a non-zero value is required, we would have corrected the experimental data point already to remove this bias. As the experimental errors, irrespective of whether they are statistical or systematic in nature, are not directly observable, the `OBS` value is consequently `NA` (=Not Answered).

Glueing everything together to obtain a node data table

Finally, we need to assemble one data table with the information about all the nodes. We make use of the function `rbindlist` of the `data.table` package:

```
node_dt <- rbindlist(list(sigma_dt, norm_dt, expdata_dt), fill = TRUE)
```

The `fill=TRUE` argument means that columns not present in one of the data tables will be created and filled with `NA` values for them. This is necessary as not all columns are meaningful for all node types. For example, the node `norm_dt` contains a single number representing the normalization error affecting all data points, hence this number cannot be assigned to a single incident energy in a meaningful way.

We also must add the column `IDX` to `node_dt` to establish an order among all the variables. This is important as Bayesian inference with the `nucdataBaynet` package constructs vectors and matrices for the Bayesian inference for which the assignment of variables to positions in those vectors is essential. The index column can be added in this way:

```
node_dt[, IDX := seq_len(.N)]
```

All nodes are defined now. We still need to create the covariance matrix blocks for all the nodes. To this end, we can use the information in the `UNC` column in combination with the `Diagonal()` function from the `Matrix` package:

```
covmat <- Diagonal(n=nrow(node_dt), x=node_dt$UNC^2)
```

Please note that it is **very important** to square the uncertainties before storing them in the diagonal of the covariance matrix.

The elements in the diagonal of `covmat` associated with the node `sigma` are zero, recall the definition of `sigma_dt`. This does not matter, as we will now use the covariance function to set up the covariance matrix associated with the node `sigma`.

```
tmp_x_mesh <- node_dt[NODE=='sigma', ENERGY]
tmp_idx <- node_dt[NODE=='sigma', IDX]
gp_covmat <- covfun(tmp_x_mesh, tmp_x_mesh, delta=10,
                    lambda=0.01, nugget=1e-6)
covmat[tmp_idx, tmp_idx] <- gp_covmat
```

Later, you may want to explore different choices of the `lambda` parameter and their impact on the estimated curve.

The covariance matrix is stored in a sparse matrix format. It is desirable to keep this covariance matrix as sparse as possible for good scalability. Regarding this aspect, the squared exponential covariance function yields dense matrices and is regarding sparseness therefore not really a good choice. Preferable are covariance functions where correlations become exactly zero if the distance between x-values exceeds a certain distance. Pseudo-observations of derivatives are another pertinent approach to enforce a certain degree of smoothness while at the same time keeping the covariance matrix sparse.

Establishing links

The next step is the creation of links between the various nodes. Links are established by mappings that connect source nodes to target nodes by functional relationships.

Mapping from computational mesh to experimental data

To obtain estimates of the cross sections associated with the node `sigma`, we need to define a mapping between `sigma` and the experimental data node `expdata`. Here we will go for an interpolation mapping, as the incident energies of the data points do not perfectly coincide with the energies of the energies of the computational mesh of `sigma`. The definition of this linear interpolation mapping is given by

```
sigma_to_exp_mapdef <- list(
  mapname = "sigma_to_exp",
  maptype = "linearinterpol_map",
  src_idx = node_dt[NODE=="sigma", IDX],
  tar_idx = node_dt[NODE=="expdata", IDX],
  src_x = node_dt[NODE=="sigma", ENERGY],
  tar_x = node_dt[NODE=="expdata", ENERGY]
)
```

The `maptype` field indicates a linear interpolation mapping. The fields `src_idx` and `tar_idx` contain the indices of the source and target node, respectively. Similarly, the fields `src_x` and `tar_x` contain the x-mesh of the source and target node, respectively. For more information on a linear interpolation mapping, you can execute `?create_linearinterpol_map` on the R prompt.

Mapping of normalization error to experimental data

We need one more mapping to establish the link from the node with the normalization error `lambda` to the experimental data node `expdata`. The following list defines this mapping:

```
normerr_to_exp_mapdef <- list(
  mapname = "mynormerr",
  maptype = "normerr_map",
  src_idx = node_dt[NODE=="lambda", IDX],
  tar_idx = node_dt[NODE=="expdata", IDX],
  src_feat = "normerr0",
  tar_feat = rep("normerr0", node_dt[NODE=="expdata", .N])
)
```

The fields specific to this mapping are `src_feat` and `tar_feat`. If several different datasets are present and every dataset should be associated with its own normalization error, these two fields provide a mechanism to do it with a single mapping. The field `src_feat` establishes the names associated with normalization errors and the field elements in `tar_feat` refer to these names. In the current Bayesian network, we have only a single dataset and therefore a single name `normerr0`. Consequently, the `tar_feat` field contains a repetition of the same string because every single datapoint of the dataset should be associated with the same normalization error. The variable `.N` contains the number of rows of the filtered data table, hence the number of experimental data points here.

Creating the compound mapping object

Now we need to combine the mapping definitions to create a compound mapping that defines the full structure of the Bayesian network.

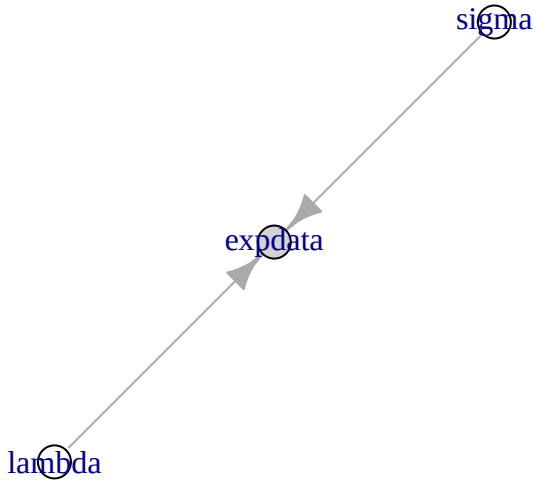
```
comp_mapdef <- list(
  mapname = "compmap",
  maptype = "compound_map",
  maps = list(
    sigma_to_exp_mapdef,
    normerr_to_exp_mapdef
  )
)
```

The `map` field contains the list of the mapping lists we have defined above. It remains to create a compound mapping object from the definition list:

```
compmap <- create_map(comp_mapdef)
```

As a quick check, we can plot the Bayesian network to see if the link structure in the Bayesian network is as expected.

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



Bayesian inference

With the setup of the Bayesian network completed, which included the definition of nodes and the associated covariance matrix as well as the definition of the link structure, we are ready to do Bayesian inference. Two functions are available in the nucdataBaynet package, which are `glsalgo` and `lmalgo`, to obtain Maximum A Posteriori (MAP) estimates. The function `glsalgo` can be used if all mappings in the network are linear. The `lmalgo` function implements an iterative optimization strategy and works for Bayesian networks with linear and non-linear mappings. For this example, we will be using the `glsalgo` function, whose help can be retrieved by typing `?glsalgo` in an R prompt.

Here is the code to compute a MAP estimate:

```
zprior <- node_dt[, PRIOR]
U <- covmat
obs <- node_dt[, OBS]
map_estimates <- glsalgo(compmap, zprior, U, obs)
```

We extract the prior best estimates from the node data table and also the observed values and store it in variables `zprior` and `obs` for convenience. The covariance matrix `covmat` was defined in the section explaining the creation of the node data table. The last instruction returns the map estimates of the variables in the Bayesian network.

For the purpose of plotting, we augment the node data table with the results:

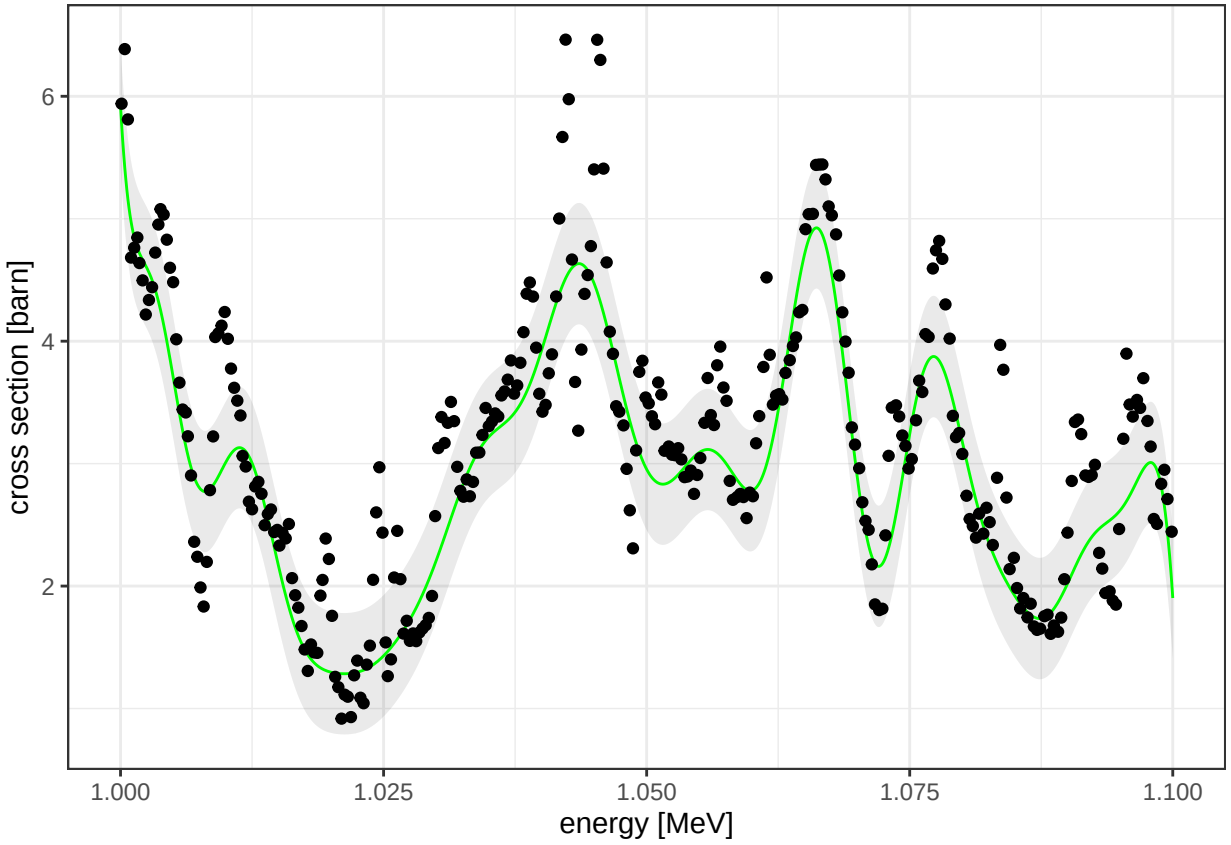
```
node_dt[, POST := map_estimates]
```

Let us also compute the posterior uncertainties to plot an uncertainty band around the estimated curve:

```
sel_idcs <- node_dt[NODE=="sigma", IDX]
U_post <- get_posterior_cov(compmap, map_estimates, U,
                            node_dt$OBS, sel_idcs, sel_idcs)
post_unc <- sqrt(diag(U_post))
node_dt[sel_idcs, UNC_POST := post_unc]
```

And here is the code to plot the result:

```
ggp <- ggplot() + theme_bw()
ggp <- ggp + geom_ribbon(aes(x=ENERGY, ymin=POST-UNC_POST, ymax=POST+UNC_POST),
                         alpha=0.1, data=node_dt[NODE=="sigma"])
ggp <- ggp + geom_line(aes(x=ENERGY, y=POST), data=node_dt[NODE=="sigma"], col="green")
ggp <- ggp + xlab("energy [MeV]") + ylab("cross section [barn]")
ggp <- ggp + geom_point(aes(x=ENERGY, y=OBS), data=node_dt[NODE=="expdata"])
ggp
```



The smoothness of the solution could be adjusted by changing the `lambda` parameter to construct the covariance matrix of the `sigma` node. Let us inspect the posterior uncertainties indicated by the pale gray band in the plot:

```
head(node_dt[NODE=='sigma'])
```

```
##     NODE PRIOR UNC OBS ENERGY IDX     POST  UNC_POST
```

```
## 1: sigma     0   0  NA 1.0000   1 5.920279 0.4968250
## 2: sigma     0   0  NA 1.0001   2 5.776044 0.4965825
## 3: sigma     0   0  NA 1.0002   3 5.645069 0.4964034
## 4: sigma     0   0  NA 1.0003   4 5.526462 0.4962722
## 5: sigma     0   0  NA 1.0004   5 5.419765 0.4961789
## 6: sigma     0   0  NA 1.0005   6 5.323140 0.4961161
```

The posterior uncertainties at all energies are about the same and a bit below 0.5. Because we have introduced a normalization uncertainty of 0.5 for the complete experimental dataset, it is from a logics point of view impossible to have a lower posterior uncertainty than that just from the data. The reason for the slightly smaller uncertainty is the prior uncertainty of `delta=10` used in the covariance function to specify the prior covariance matrix. This extra bit of imposed prior information pulls the posterior uncertainties slightly down. If one adoped an increased value `delta=1e3`, the resulting posterior uncertainties would be slightly larger than 0.5. The excess can be attributed to the statistical uncertainties of the experimental data points that are not completely washed out by the smoothing of the Gaussian process.

Summary

This tutorial explained the construction of a Bayesian network to perform inference using a Gaussian process. The data points were given by measurements of the energy-dependent total cross section of Fe-56 for incident neutrons available in the EXFOR database. It was shown how besides statistical uncertainties also a normalization uncertainty can be taken into account by the introduction of a dedicated node.