Function estimation with the nucdataBaynet package

Georg Schnabel

2022-11-12

This R Markdown document shows the setup of a Bayesian network with the *nucdataBaynet* package to estimate a function from a set of data points. We assume 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 eneergies between 1.0 and 1.1 MeV. In order to ensure that the estimated curve is smooth, we will specify pseudo-observations of the second derivative of the unknown function, which can be regarded as an implicit construction of a Gaussian process.

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. Finally, 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, which allow to control 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

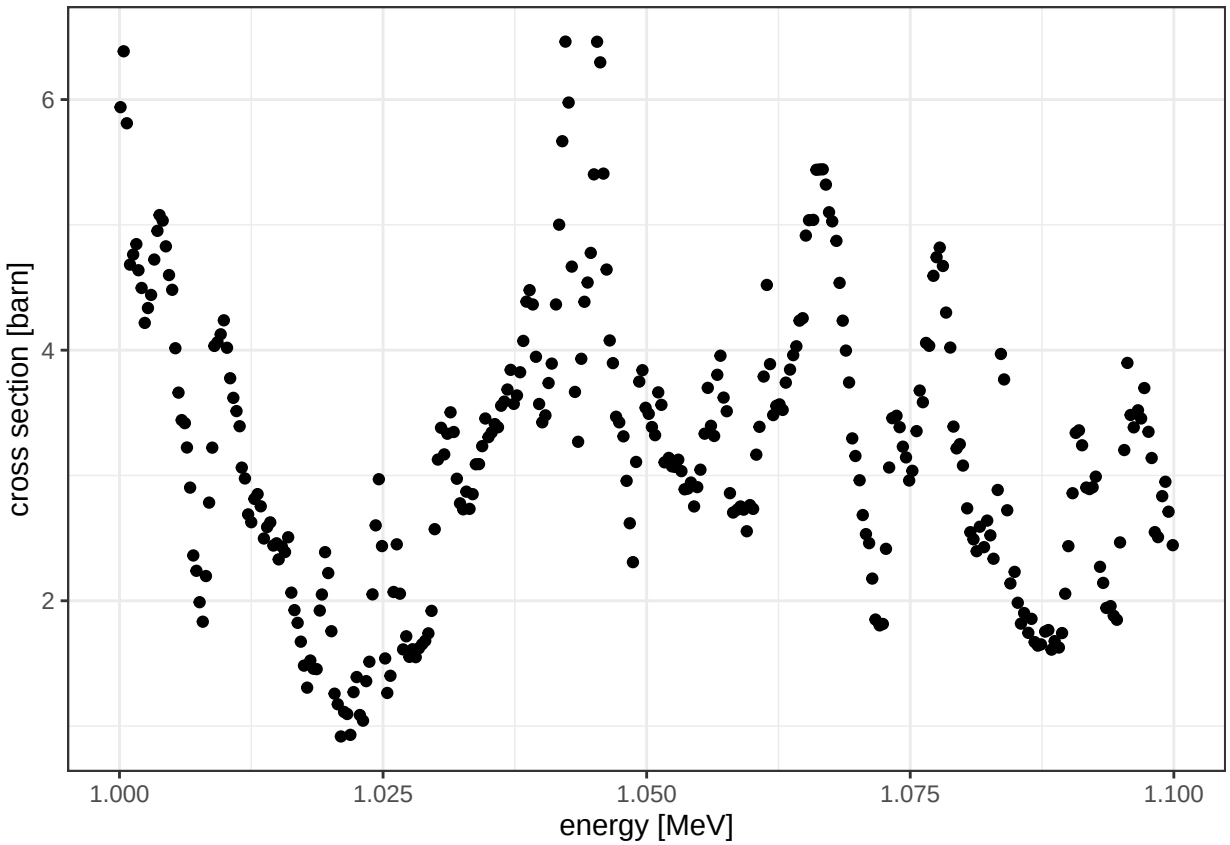
The experimental data have been prepared as a csv file along with this document. To load the dataset of Cornelis et al. (EXFOR accession number 22316001), we can use the following code:

```
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 = Inf,
  OBS = NA,
  ENERGY = seq(Emin, Emax, by=1e-5)
)
```

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. The `UNC` column contains the associated prior uncertainty, which we set to infinity to obtain a non-informative prior. 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-5` MeV (10 eV) between mesh points yields a mesh with 10001 points.

There are many possible types of Gaussian process kernels, such as the squared exponential. Here we will be using a convenient and flexible construction that permits us to control the smoothness in a fine-grained way. The idea is to create a node that is associated with the second derivatives of the cross sections assigned to the node `sigma` and pretend that we have observations for the values of this node.

Imposing uncertainties on the individual second derivatives introduces a bias to favor smoothing solutions. The smaller the values of these uncertainties, the smoother the cross section function in the node `sigma` will be.

The definition of the node `sigma_2nd` containing the second derivatives of `sigma` is done analogously to the definition of `sigma`:

```
sigma_2nd_dt <- data.table(
  NODE = "sigma_2nd",
  PRIOR = 0,
  UNC = 1e7,
  OBS = 0,
  ENERGY = sigma_dt[2:(nrow(sigma_dt)-1), ENERGY]
)
```

Using a vector with zeros for the prior values given in column `PRIOR` favors a straight line as most likely solution in the absence of data. Non-zero values of `PRIOR` would bias the solution towards a parabola. By introducing a vector of zeros for `OBS`, we implement the assumption that the values in this node have been observed. The vector of energies coincides with that for the node `sigma` except that the first and last element are missing. This is due to the implementation of the second derivative in the associated mapping, type `?create_derivative_2nd_map` on the R prompt for details.

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 = 0.05,
```

```
  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 can use of the function`rbindlist` of the `data.table` package:

```
node_dt <- rbindlist(list(sigma_dt, sigma_2nd_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 `lambda` 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 a covariance matrix.

The covariance matrix is stored as a sparse matrix. The degree of sparseness is extremely high as all non-diagonal elements are zero. Despite the covariance matrix being diagonal, the prior covariance matrix still incorporates a bias towards smooth solutions because a subset of diagonal elements is associated with the prior uncertainties for pseudo-observations of the second derivative of the unknown function.

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 a functional relationship.

Mapping to second derivatives to impose smoothness bias

To enforce smoothness, we need to link the node `sigma` to `sigma_2nd` associated with the pseudo-observation of the second derivatives. The functional relationship is given by a finite difference approximation that computes the second derivative, which is explained in the help page of `create_derivative2nd_map` (type `?create_derivative2nd_map` on an R prompt to view it).

The definition of a mapping is given by a list. Here is the definition of the mapping to associate `sigma` to `sigma_2nd` (explanation following afterwards):

```
sigma_to_sigma2nd_mapdef <- list(
  mapname = "2ndmap",
```

```
  maptype = "derivative2nd_map",
  src_idx = node_dt[NODE=="sigma", IDX],
  tar_idx = node_dt[NODE=="sigma_2nd", IDX],
  src_x = node_dt[NODE=="sigma", ENERGY],
  tar_x = node_dt[NODE=="sigma_2nd", ENERGY]
)
```

The `mapname` field contains a string chosen by the user that identifies this particular mapping. This string is helpful in debugging Bayesian networks and can be also used in visualizations, but does not have any relevance for the Bayesian inference in the network. The `maptype` field indicates the type of mapping, here a mapping to go from a function to its second derivative. Type `?create_map` to get a list with available mapping types implemented so far in the nucdataBaynet package.

The next two fields, `src_idx` and `tar_idx` identify the source variables and the target variables of the mapping respectively. Query operations on the node data table are employed to retrieve the relevant indices for the `sigma` and `sigma_2nd` node.

The last two fields `src_x` and `tar_x` provide the energies associated with the source and target mesh. For the time being, the energies of the target mesh must be a subset of those of the source mesh and for each energy of the target mesh there must be a smaller and a larger energy in the source mesh available.

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]
)
```

Apart from the `maptype` field, which indicates a linear interpolation mapping here, the meaning of the other fields is precisely the same as for the second derivative mapping explaind in the previous section.

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.

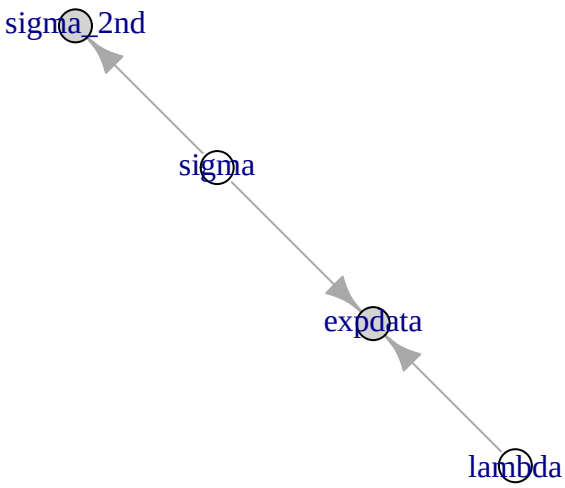
```
comp_mapdef <- list(
  mapname = "compmap",
  maptype = "compound_map",
  maps = list(
    sigma_to_exp_mapdef,
    sigma_to_sigma2nd_mapdef,
    normerr_to_exp_mapdef
  )
)
```

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)
```

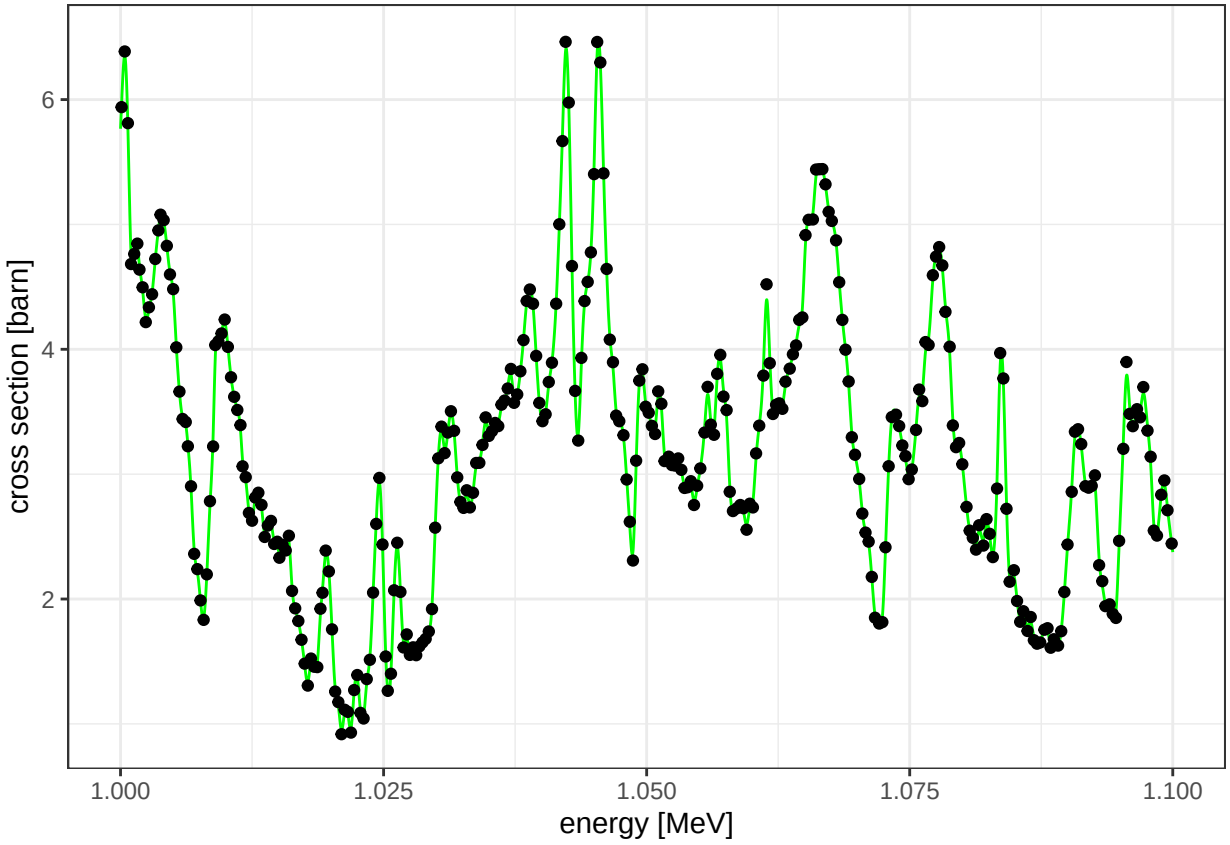
We extract the prior best estimates from the node data table and also the observed values and store it in variables `zprior` and `obs` for convenience. The covariance matrix `covmat` was defined in the section explaining the creation of the node data table. The last instruction returns the map estimates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

And here is the code to plot the result:

```
ggp <- ggplot() + theme_bw()
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
```

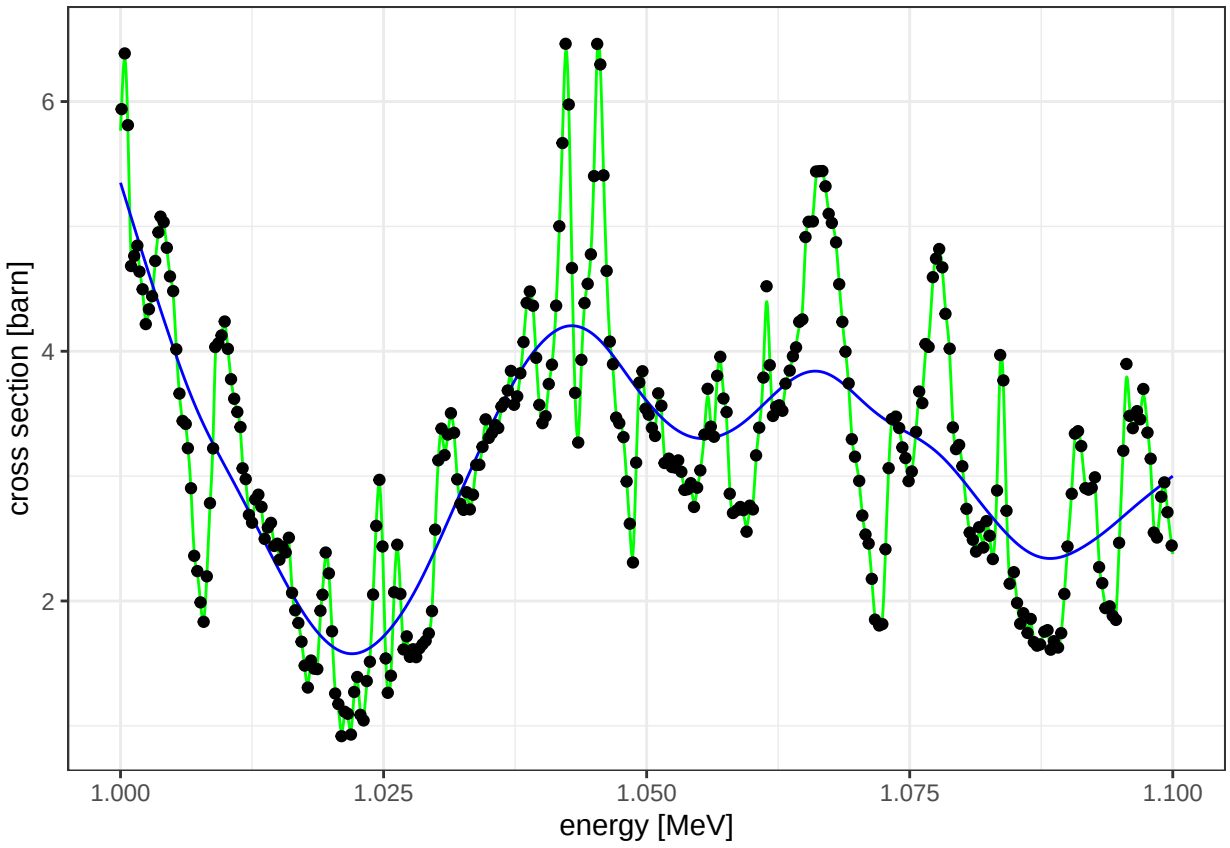


The most likely values of the cross section follow closely the data. Looking at the plot, we may be overfitting the data. Let us use this opportunity to explore the impact of the prior uncertainties associated with the second derivative node. To this end, we can reduce the uncertainties from `1e7` to `1e4` and check the impact on the MAP estimate.

```
alt_covmat <- covmat
diag(alt_covmat)[node_dt$NODE == "sigma_2nd"] <- 1e4^2
alt_U <- alt_covmat
alt_map_estimates <- glsalgo(compmap, zprior, alt_U, obs)
node_dt[, ALT_POST := alt_map_estimates]
```

And here we plot again:

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



Summary

This tutorial explained the construction of a Bayesian network to infer an unknown function from a set of data points. 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. It was demonstrated how the smoothness of solutions can be controlled by adjusting the uncertainties of the pseudo-observation of the second derivatives of the cross sections on the computational mesh.