Introduction to `jmpost`

Georgios Kazantzidis Daniel Sabanés Bové Xiaoyan Fang

Common usage

`jmpost` is a package aiming to optimize the procedure of developing a new Bayesian joint model. Minimal use of the package provides complete examples of joint modeling code. Here we explain the workflow of the package `jmpost` using `stan` libraries as an example.

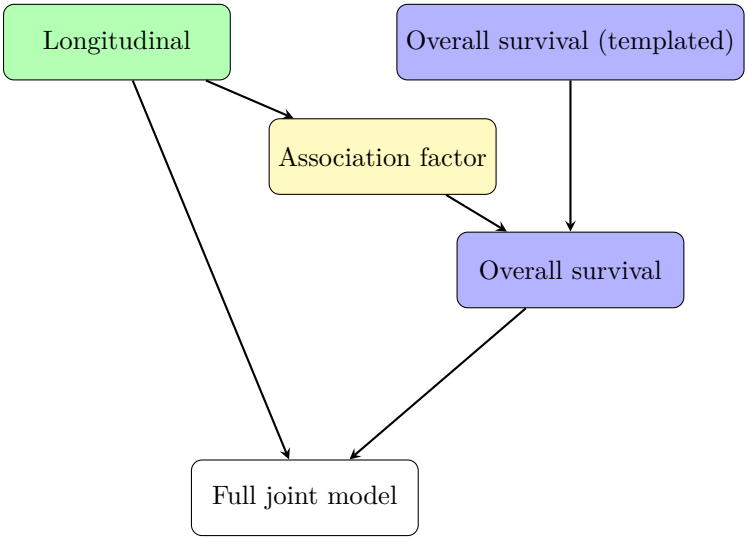

Figure 1: Package workflow. Different colours signify objects of different classes. Arrows indicate the relationships between the objects.

Figure~ illustrates the main workflow of the `jmpost` package. Three objects need to be combined for the construction of the full Bayesian joint model. The longitudinal sub-model, including all the required information for the modeling of the longitudinal measurements, the survival sub-model, including information of for the time to event data and the link or association factor, containing information for the conection of the two sub-models.

```
Long_mod <- LongModel(
    stan = StanModule(
        functions = "exp_long_functions.stan",
        data = "exp_long_data.stan",
        priors = long_prior(),
        model = "",
        inits = list(),
        parameters = "exp_long_parameters.stan",
        transformed_parameters = "exp_long_transformed_parametes.stan",
        generated_quantities = ""
    )
)
```

The construction of the different parts of the model is flexible. Here, we create the longitudinal part by calling the `LongModel` constructor. `LongModel` identifies the type of the sub-model whereas the helper constructor `StanModule` is responsible for the actual stan code. The created object contains six compartments containing different parts of the stan code (functions, data, priors, model, parameters, transformed parameters and generated quantities) while also `inits` which contains the initial values for the Markov Chain Monte Carlo run. The user is able to populate the different parts of the object with stan code either by providing the names of the stan files contained in the `jmpost` directory, the full paths of stan files outside the `jmpost` direcotry or by providing stan code in text form (code chunk below).

```
example_stan <- LongModel(
    stan = StanModule(
        functions = "type stan code here;",
        data = "type stan code here;"
    )
)
```

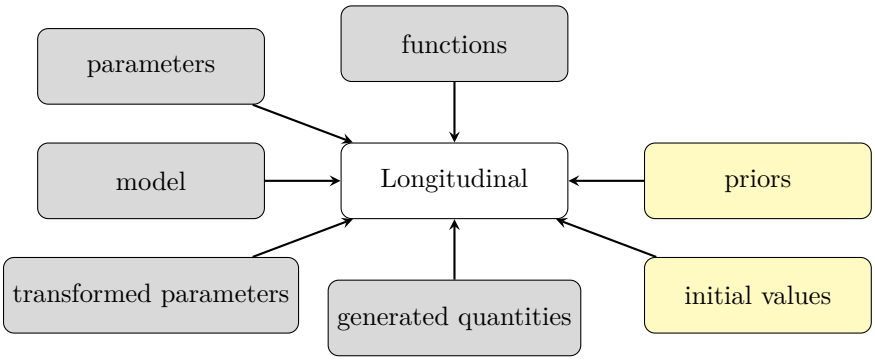

Figure 2: Structure of 'StanModule' object.

Figure~ depicts the structure of a `StanModule` object. Gray coloured cells indicate character strings containing stan code whereas yellow coloured cells mark list objects.

```
StanModule(functions = "",
           data = "",
           model = "",
           parameters = "",
           transformed_parameters = "",
           generated_quantities = "",
           priors = list(),
           inits = list()
)
```

The `jmpost` repository contains stan code for the creation of a specific stan model. The user is able to populate the longitudinal object `Long_mod` either by specifying the stan files containing the relevant code of the section or by directly typing the stan code into R. We split the model section of the stan file into two parts, the `priors` and the `model`. `Priors` contain a list with the names of the parameters and their prior distribution definitions while `model` is a character that contains model relevant information such us the definition of the likelihood. The list of priors should be a named list of strings where each element represents the stan code for the definition of the desired density.

```
prior_set <- StanModule(priors = list("param_A" = "normal(0,1);"))
```

The user can modify the prior list at any point during the workflow.

```
prior_set@priors$param_A <- "normal(1,1);"
```

Call of the object prints the different sections.

```
Long_mod
```

The overall survival object contails all ther information for the paramtetrisation of the second sub-model of the joint model. The overall survival object has identical structure as the Longitudinal object (Figure~). The slots `functions`, `parameters`, `model`, `transformed parameters` and `generated quantities` should be provided with stan code whereas `priors` and `initial values` should be filled with lists. An example with Log-logistic parametrisation of the overall survival sub-modlel object is provided in `LogLogisticOS()`.

```
temp_os <- LogLogisticOs()
```

Although the Longitudinal and the Survival object identical structures, the contained stan code has some fundamental differences. The functions contained in the overall survival object depend on the parametrisation of the longitudinal part. Thus, in order to complete the stan code of the former object, additional information is required. The required details are provided to the overall survival object through the link object. The link is an object containing the parametrisation of the longitudinal sub-model, that will be used as part of the overall survival object.

```
link <- HazardLink(
    parameters = "beta_ttg",
    contribution = "* rep_matrix(ttg(psi_ks, psi_kg, psi_phi), rows(time))",
    stan = StanModule( )
)
```

Link objects or association factor, contain the slots of `parameters`, `contribution` and `stan`. Here, the slot `parameters` is the name of the linking parameter of the two sub-models. `Contribution` contains the stan code relevant to the linking parameter. `Parameters` and `contribution` contain all the information required for the completion of the overall survival model.

The merge of the `Longitudinal`, `Overall survival` and `link` object takes place in two parts. First, the parameterisation of the overall survival sub-model needs to be completed using the information from the link object.

```
os <- parametrize(osmod = temp_os, link = link)
```

Secondly, the `Longitudinal` and `Survival` sub-models need to be combined. Function `JointModel` merges the two sub-models, creates a character containing the full stan model, creates a stan file with the produced model and checks the functionality of the produced file. Mistakes in the stan file will be reported from the `JointModel` call.

```
jm <- JointModel(long = Long_mod, os = os)
```

`jm` object contains all the complete stan code for the joint model. `jmpost` uses the package `cmdstanr` for the compilation of the stan model. For the run of the model, additional information such as the number of iterations are required.