BHPMF - Bayesian Hierarchical Probabilistic Matrix Factorization. Filling gaps in hierarchical data

Franziska Schrodt, Farideh Fazayeli

June 2017

BHPMF is a machine learning (recommender system based) gap filling technique which has been developed specifically to make use of information within hierarchical data structures. In other words, where individual observations can be nested within higher orders (e.g. data collected on living organisms with hierarchial information from the taxonomy, soil data with hierarchical information from different soil layers or car manifacturing data with hierarchical information from car models, manufacturers and countries of origin.

BHPMF enables imputation (gap filling) where at least one measurement is available per observation. The advantages of BHPMF compared to other gap filling techniques are three fold. BHPMF (1) fills gaps at the individual observation level (e.g. individual plant rather than species), (2) makes use of correlations between measured variables and within hierarchical structures and (3) provides estimates of uncertainties for each imputed value.

The code below demonstrates the different steps necessary to implement BHPMF, take full advantage of all its features and appropriately analyse the quality of the gap filling.

For more information on the algorithm used and technical details, please refer to:

BHPMF for ecological applications:

F. Schrodt, J. Kattge, H. Shan, F. Fazayeli, J. Joswig, A. Banerjee, M. Reichstein, M. Boenisch, S. Diaz, J. Dickie, A. Gillison, A. Karpatne, S. Lavorel, P.W. Leadley, C. Wirth, I. Wright, S.J. Wright, P.B. Reich (2015) BHPMF – a hierarchical Bayesian approach to gap-filling and trait prediction for macroecology and functional biogeography. Global Ecology and Biogeography 24: 1510–1521 link

Technical descriptions:

F. Fazayeli, A. Banerjee, J. Kattge, F. Schrodt, P.B. Reich (2014) Uncertainty quantified matrix completion using Bayesian Hierarchical Matrix factorization. International Conference on Machine Learning and Applications (ICMLA)

H. Shan, J. Kattge, P. B. Reich, A. Banerjee, F. Schrodt, M. Reichstein. (2012) Gap Filling in the Plant Kingdom – Trait Prediction Using Hierarchical Probabilistic Matrix Factorization. International Conference on Machine Learning (ICML)

BHPMF package installation

If you're working on Windows, you will need to have Rtools installed. For instructions, see here

```
library(devtools)
install_github("fisw10/BHPMF")
library(BHPMF)
```

Data preparation

Specify the working directory where your data is stored and where you want the `BHPMF` results to be placed.

```
setwd("Your/working/directory/")
```

Specify a temporary output directory - this will store the intermediate results from gap filling. Note, the folder has to be called "tmp" and needs to be empty (!) the first time BHPMF is run for a new dataset. Each time a function from BHPMF is called, the preprocessing files saved in this directory will be used. This helps to avoid having to re-run preprocessing functions for the same dataset. Note: if this is not specified, BHPMF will create a tmp folder automatically.

```
tmp.dir <- dirname("Your/working/directory/tmp/")
```

BHPMF comes with example data from the TRY database of plant functional traits (Kattge et al. 2011) with has been randomized to adhere to data protection requirements. Things to consider when using your own data:

1. You need two data sets:
    - one containing the hierarchical information (here: `hierarchy.info`) which should not contain any missing values
    - one dataset containing the measurements (here: `trait.info`) which are to be gap filled with empty cells being coded as "NA"
2. The hierarchical data set should contain the hierarchical information in ascending order with the first column containing your data code, the second column containing the lowest hierarchical level (here: species), the third column the next highest level in which the lowest level is nested (here: genus) and so on. The class of the `hierarchy.info` file is `data.frame`.
    - Load the hierarchical data file (in this case, plant taxonomy), consisting of 20,000 rows (individual plant species names) and 4 columns (hierarchical levels (here: species nested in genus nested in family nested in phylogenetic group)) with an additional column containing the measurement code.

```
data(hierarchy.info)
head(hierarchy.info)
```

```
##   plant_id  species genus    family
## 1        1 Acer cac  Acer Aceraceae
## 2        2 Acer cac  Acer Aceraceae
## 3        3 Acer sac  Acer Aceraceae
## 4        4 Acer sac  Acer Aceraceae
## 5        5 Acer sac  Acer Aceraceae
## 6        6 Acer sac  Acer Aceraceae
```

3. The data set containing the missing values should be in the same order as the hierarchical dataset with measured variables (e.g. `traits`) in the columns and individuals (e.g. `species`) in the rows. The class of the `trait.info` file is `matrix`.
    - Load the measurement data file (in this case, plant functional traits), consisting of 73,300 rows (individual plants) and 13 columns (functional traits).

```
data(trait.info)
head(trait.info)
```

```
##      Trait1 Trait2 Trait3 Trait4 Trait5 Trait6     Trait7 Trait8 Trait9
## [1,]     NA     NA     NA     NA     NA     NA  1.0644797     NA     NA
## [2,]     NA     NA     NA     NA     NA     NA -0.7959262     NA     NA
## [3,]     NA     NA     NA     NA     NA     NA  1.3739679     NA     NA
## [4,]     NA     NA     NA     NA     NA     NA  1.3383991     NA     NA
```

```
## [5,]     NA     NA     NA     NA     NA     NA -0.7345956     NA     NA
## [6,]     NA     NA     NA     NA     NA     NA  1.3312182     NA     NA
##      Trait10 Trait11 Trait12    Trait13
## [1,]      NA      NA      NA  0.5076937
## [2,]      NA      NA      NA -0.3405634
## [3,]      NA      NA      NA  0.6250250
## [4,]      NA      NA      NA  0.6305120
## [5,]      NA      NA      NA  0.1231677
## [6,]      NA      NA      NA  0.3565508
```

4. So far, BHPMF can not deal with categorical variables. If your dataset contains categories, consider transforming where possible.

Although we provide example data which is already in the correct format, it is good practice to double check that all individuals in the trait matrix are accounted for in the hierarchy matrix, e.g. by keeping the species information in the trait data and using:

```
sum(row.names(traits) %in% colnames(hier)) == ncol(hier)
```

Here, we just check if the number of rows is the same in both datasets

```
nrow(hierarchy.info) == nrow(trait.info)
```

```
## [1] TRUE
```

Another feature of the provided `trait.info` data is that it has been log- and z-transformed to improve normality and equalize measurements in the cost function during optimization. If this should not be the case for your dataset, test it for normality and perform z-transformation to get the data to a mean of 0 and standard deviation of sigma squared. The following code performes the necessary transformation (assuming log transformation is appropriate for your data) and stores the variables necessary to back-transform you data after performing BHPMF in the output directory:

```
back_trans_pars <- list()
    rm_col <- c()
    for(i in 1:ncol(trait.info)){
      x <- trait.info[,i] # goes through the columns
      min_x <- min(x,na.rm = T) # takes the min of each column
        if(min_x < 0.00000000001){
        x <- x - min_x + 1  # make this optional if min x is neg
        }
      logx <- log(x)
      mlogx <- mean(logx, na.rm = T)
      slogx <- sd(logx, na.rm = T)
      x <- (logx - mlogx)/slogx # Z transformation
      back_trans_pars[[i]] <- list(min_x = min_x,
                                   mlogx = mlogx,
                                   slogx = slogx)
      trait.info[,i] <- x
      }
write.csv(back_trans_pars, paste0(tmp.dir,"back_trans_pars.csv"))
```

Performing the gap filling

The most simple means of performing the gap filling, using all standard . . . settings is run as follows - note that you should remove `tmp.dir` if you want `BHPMF` to automatically produce a temporary directory:

```
GapFilling(trait.info, hierarchy.info,
           mean.gap.filled.output.path = paste0(tmp.dir,"/mean_gap_filled.txt"),
           std.gap.filled.output.path= paste0(tmp.dir,"/std_gap_filled.txt"), tmp.dir=tmp.dir)
```

This will produce two files: `mean_gap_filled.txt` and `std_gap_filled.txt` which contain the mean gap filled data (including gap filling for existing datapoints which were removed as part of the validation process (see `Validation of the gap filling` below)) and the standard deviation ("uncertainty") of each imputation respectively. In other words, even instances where you do have measurements are imputed within `mean_gap_filled.txt` and you will need to filter these out if you want to use the data for subsequent analyses. These "artificially imputed" data instances are also used to calculate the root mean squared error (RMSE) and to produce an xy plot of standard deviation versus RMSE provided at the end of each `GapFilling` run (see also `Validation of the gap filling output` below)

Specifying other settings

Use of prediction.level and used.num.hierarchy.levels

Depending on your hierarchical information, you might want to fill gaps not on the lowest available hierarchical level (e.g. here: `species`) but aggregated at a higher hierarchical level (e.g. here: `genus`) and/or not use information from all levels of the hierarchy (e.g. not consider information on the `family` level but only at the `species` and `genus` level). These factors can be adjusted using `prediction.level` and `used.num.hierarchy.levels` within the `GapFilling` function. The parameter `used.num.hierarchy.levels` is also useful to generate results similar to Figure 4 in Schrodt et al. (2015) and Figure 4 and Table 2 in Shan et al. (2012).

1. Figure 1 shows the inputs and output of the `GapFilling` function. Here the hierarchy has 4 levels where the observation level (e.g. plants) is the 4th level. In this example, the `GapFilling` function will predict the missing values at the observation level i.e. filling the missing values in matrix X by setting the parameter `prediction.level` to 4 (observation (e.g. species) level - default value). Also, here we use all the hierarchy information (information in levels 1, 2 and 3, e.g. phylogenetic group, genus and family) by setting the parameter `used.num.hierarchy.levels` to 3 (the default value). The usage is shown in the examples.

```
## Loading required package: jpeg
```

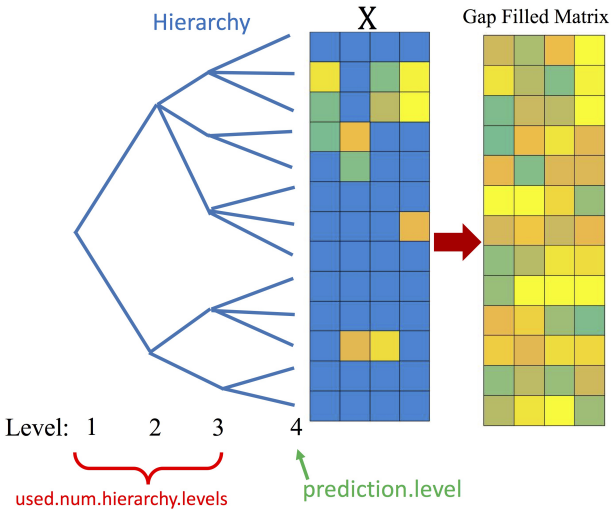

Figure 1

- Corresponding gap filling code

```
GapFilling(trait.info, hierarchy.info,
           mean.gap.filled.output.path = paste0(tmp.dir,"/mean_gap_filled.txt"),
           std.gap.filled.output.path= paste0(tmp.dir,"/std_gap_filled.txt"))
```

2. Figure 2 shows the inputs and output of the `GapFilling` function where we use only part of the hierarchy information (information in levels 1, 2 (e.g. using only information from the phylogenetic group and family)) by setting the parameter `used.num.hierarchy.levels` to 2. In this example, the GapFilling function will predict the missing values at the observation level (e.g. species) i.e., filling the missing values in matrix X since we set the parameter `prediction.level` to 4 (default value). The usage is shown in the examples.

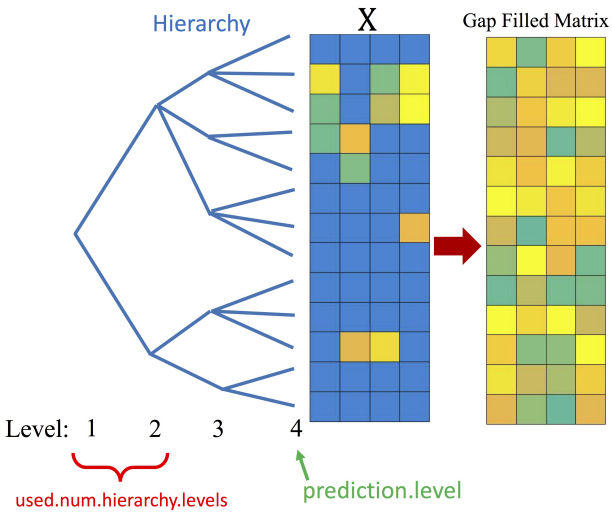

Figure 2

- Corresponding gap filling code

```
GapFilling(trait.info, hierarchy.info,
           prediction.level = 4,
           used.num.hierarchy.levels = 2,
           mean.gap.filled.output.path = paste0(tmp.dir,"/mean_gap_filled.txt"),
           std.gap.filled.output.path=paste0(tmp.dir,"/std_gap_filled.txt"))
```

Figure 3 shows the inputs and output of the GapFilling function where we use part of the hierarchy information (information in levels 1, 2) by setting the parameter `used.num.hierarchy.levels` to 2. In this example, the GapFilling function will predict the missing values at level 3 i.e., filling the missing values in matrix at level 3 (smaller dimension, e.g. at the genus level) by setting the parameter `prediction.level` to 3. The usage is shown in the examples.

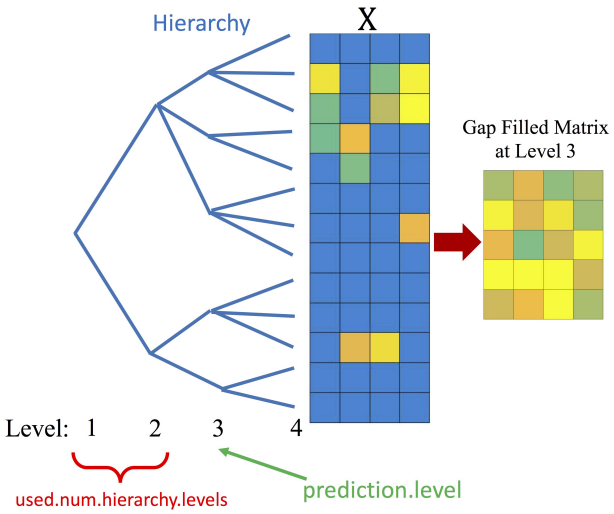

Figure 3

- Corresponding gap filling code

```
GapFilling(trait.info, hierarchy.info,
           prediction.level = 3,
           used.num.hierarchy.levels = 2,
           mean.gap.filled.output.path = paste0(tmp.dir,"/mean_gap_filled.txt"),
           std.gap.filled.output.path= paste0(tmp.dir,"/std_gap_filled.txt"))
```

- In order to run PMF (BHPMF without any hierarchical information):

```
GapFilling(trait.info, hierarchy.info,
           used.num.hierarchy.levels = 0,
           mean.gap.filled.output.path = paste0(tmp.dir,"/mean_gap_filled.txt"),
           std.gap.filled.output.path= paste0(tmp.dir,"/std_gap_filled.txt"))
```

Tuning

Tuning optimises parameters which can impact the model performance, such as the length of the latent vectors, and can help to minimize the RMSE. This step can be slightly time consuming depending on the size of your data matrix but is highly recommended to ensure the best possible model is applied.

Tuning is initiated by setting `tuning` to `TRUE` within `GapFilling` or by running `TuneBhpmf` separately. The number of cross validation runs or "folds" can be specified using `num.folds.tuning`. The default value for this parameter is 10.

If `tuning` is set to `FALSE` (automatic setting), `num.latent.feats` is automatically set to 10.

Output from tuning is a list with `best.number.latent.features` and `min.rmse` which give the best length for the latent vectors and minimum RMSE achieved with tuning, respectively. Results of the tuning are stored in the "fold" folders within `tmp.dir`.

Gibbs sampler

Uncertainties for each imputation are caluclated using a Gibbs sampler. This is a Markov Chain Monte Carlo (MCMC) technique which generates posteriors by sampling from each variable's conditional distribution and keeps the remaining variables fixed to their current values.

Gibbs sampler related factors are adjusted using `num.samples`, `burn` and `gaps`.

- `num.samples` specifies the number of samples generated at each fold. This should never need to be higher than 10000 and is automatically set to 1000
- `burn` sets the "burn-in" period. The Gibbs sampler might take some time to reach a point where all samples are coming from a stationary Markov chain distribution. In order to avoid sampling from iterations before that point, the number of samples to discard (the "burn-in") needs to be specified. The default value is 200 but it should always (!) be smaller than `num.samples`
- `gaps` specifies the gaps between the sampled parameters. The default value is 2.

Validation of the gap filling output

In order to validate the quality of the performed gap filling, 80% of entries are randomly selected for training (parameter setting), 10% for validation (parameter adjustment by optimizing performance) and 10% for testing (independent performance testing after parameter adjustment and learning). This cross-validation improves model fidelity by ensuring that none of the observations are known by the model when performing new predictions. Test entries without training data in the same row would have highly inflated variance. Such cases are prevented by adjusting the splitting into training, test and validation sets. For details, see Appendix S1 in Schrodt et al. (2015). Several ways of evaluating the quality of the imputations are built into the `BHPMF` package:

1. Plotting root mean squared error (RMSE) versus the standard deviation (SD): Specifying `rmse.plot.test.data=TRUE` in the `GapFilling` function produces an xy plot of the root mean squared error (RMSE) versus the standard deviation (SD). This can be used to evaluate model fidelity and should show increasing RMSE with increasing SD.
   - e.g:

```
GapFilling(trait.info, hierarchy.info,
              mean.gap.filled.output.path = paste0(tmp.dir,"/mean_gap_filled.txt"),
          std.gap.filled.output.path= paste0(tmp.dir,"/std_gap_filled.txt"),
              rmse.plot.test.data=TRUE)
```

2. Calculating the cross validation RMSE: The cross valiadation RMSE provides the average root mean squared error and standard deviation of the average root mean squared error and can serve as a good indication about the quality of your gap filling.
   - using the `CalculateCvRmse` function:

```
out1 <- CalculateCvRmse(trait.info, hierarchy.info)
                        avg.rmse <- out1$avg.rmse
                        std.rmse <- out1$std.rmse
```

3. Analysing the imputation uncertainties: There are essentially two means of analysing uncertainties in the imputations:
   - evaluating the RMSE of all imputations in the `std_gap_filled.txt` file

- comparing the imputed with original data (as mentioned above, due to the cross-validation procedure, data is randomly removed from the `trait.info` matrix and all data in the `mean_gap_filled.txt` file are imputed, including where original measurements were available)