OntoPhylo Tutorial: Application 1 - Estimating Branch Rates

Diego S. Porto and Sergei Tarasov

23 August, 2023

Load packages.

If you are starting a new R session, then reload *ontophylo*.

```r
library(ontophylo)
```

Load and organize data.

Load the data from the previous tutorial.

```r
load("RData/step2_paramo.RData")
```

Split the lists of amalgamated stochastic maps to facilitate downstream analyses.

```r
stm_amalg_head <- stm_amalg_anato$head
stm_amalg_meso <- stm_amalg_anato$mesosoma
stm_amalg_meta <- stm_amalg_anato$metasoma
```

Overview on rate estimation with OntoPhylo.

OntoPhylo is able to estimate the evolutionary rate of discrete characters, allowing for variation across and within tree branches, by using a phylogenetic non-homogeneous Poisson process (pNHPP). The pNHPP method takes the state change events from the sample of stochastic maps as input and converts them into a reconstructed rate function using a non-parametric approach based on Kernel Density Estimation (KDE). The pNHPP approach is based on the concept of a time-dependent Poisson process, where a collection of state changes (Ns) in a time interval (t) occurs at a rate lambda(t). As formalized in Taddy and Kottas (2012) and Ross and Markwick (2018), in a standard NHPP, lambda(t) can be reconstructed by estimating a probability density density function (pdf) of transitions along branches (e.g., here by using KDE) and overall number of transitions across the tree (lambda0), estimated from the data.To ensure a smooth density function, particularly over branch split events, we introduce a specialized Markov kernel (MK) for KDE (see our manuscript for details).

STEP 1. Processing state bins across branches.

Before amalgamation, branches from individual stochastic character maps were discretized into episodic bins, each corresponding to a small fraction of time spent into a state. Discretization is important to allow amalgamation by stacking stochastic maps from individual characters. However, some adjacent bins are different, thus indicating that a transition has occurred, but others are identical, thus redundant. Let's merge the identical adjacent state bins across all branches and trees.

```r
# Merge state categories across branches.
cat(paste0("\n", "Starting merging state categories: ", Sys.time(), "\n"))
stm_merg_head <- lapply(stm_amalg_head, function(x) merge_tree_cat(x) )
stm_merg_head <- do.call(c, stm_merg_head)

stm_merg_meso <- lapply(stm_amalg_meso, function(x) merge_tree_cat(x) )
stm_merg_meso <- do.call(c, stm_merg_meso)

stm_merg_meta <- lapply(stm_amalg_meta, function(x) merge_tree_cat(x) )
stm_merg_meta <- do.call(c, stm_merg_meta)

stm_merg_pheno <- lapply(stm_amalg_pheno, function(x) merge_tree_cat(x) )
stm_merg_pheno <- do.call(c, stm_merg_pheno)
cat(paste0("\n", "Finished merging state categories: ", Sys.time(), "\n"))
```

STEP 2. Extracting branch data to estimate rates.

Then, let's get the information on state changes along the branches of the amalgamated stochastic maps. For binary characters, it is straightforward to check if two adjacent bins are equal or not. However, for amalgamated characters, states are combinations of the states of individual characters. Amalgamated states are strings of concatenated numbers, each position corresponding to an individual character (e.g. "3010102042" and "2000102032"). In this case, we can get the number of transitions by calculating the Hamming distances between adjacent state bins. Each chunk of code below can take up to 15 minutes. So take it easy!!! First, let's just run the chunk for the head and skip the remaining ones for now (mesosoma, metasoma, phenome). We will return to these latter.

HEAD

```r
# Calculate Hamming distances between state vectors (10_15 min).
cat(paste0("\n", "Starting calculating hamming distances: ", Sys.time(), "\n"))
path_hm_head <- path_hamming_over_trees_KDE(stm_merg_head)
cat(paste0("\n", "Finished calculating hamming distances: ", Sys.time(), "\n"))
```

MESOSOMA

```r
# Calculate Hamming distances between state vectors (10_15 min).
cat(paste0("\n", "Starting calculating hamming distances: ", Sys.time(), "\n"))
path_hm_meso <- path_hamming_over_trees_KDE(stm_merg_meso)
cat(paste0("\n", "Finished calculating hamming distances: ", Sys.time(), "\n"))
```

METASOMA

```r
# Calculate Hamming distances between state vectors (5~10 min).
cat(paste0("\n", "Starting calculating hamming distances: ", Sys.time(), "\n"))
path_hm_meta <- path_hamming_over_trees_KDE(stm_merg_meta)
cat(paste0("\n", "Finished calculating hamming distances: ", Sys.time(), "\n"))
```

PHENOME

```r
# Calculate Hamming distances between state vectors (5~10 min).
cat(paste0("\n", "Starting calculating hamming distances: ", Sys.time(), "\n"))
path_hm_pheno <- path_hamming_over_trees_KDE(stm_merg_pheno)
cat(paste0("\n", "Finished calculating hamming distances: ", Sys.time(), "\n"))
```

STEP 3. Pre-process branch data and estimate KDE.

Now, let's process again the branch data, calculate some parameters necessary for KDE, and finally estimate the KDEs.

Discretize the reference tree.

```r
hym_tree_discr <- discr_Simmap(hym_tree, res = res)
```

Again, choose one chunk below to run. Let's run the chunk for the head first. In each chunk, `make_data_NHPP_KDE_Markov_kernel` will extract the timings of changes on each branch, then `estimate_band_W` will estimate the bandwidth to be used by `estimate_edge_KDE` in the Kernel Density Estimation (KDE).

HEAD

```r
# Make path data for all tips.
path_data_head <- make_data_NHPP_KDE_Markov_kernel(path_hm_head)

# Add pseudo-data.
# path_data_head <- lapply(path_data_head, function(x) c((-1*x), x) )

# Estimate bandwidth.
bdw_hd <- estimate_band_W(hym_tree_discr, path_data_head, band.width = 'bw.nrd')
bdw_hd <- mean(bdw_hd)

# Estimate Kernel Density Estimator (KDE).
cat(paste0("\n", "Starting estimating KDEs: ", Sys.time(), "\n"))
edge_KDE_hd <- estimate_edge_KDE(hym_tree_discr, Path.data = path_data_head, h = bdw_hd)
cat(paste0("\n", "Finished estimating KDEs: ", Sys.time(), "\n"))
```

MESOSOMA

```r
# Make path data for all tips.
path_data_meso <- make_data_NHPP_KDE_Markov_kernel(path_hm_meso)

# Add pseudo-data.
# path_data_meso <- lapply(path_data_meso, function(x) c((-1*x), x) )

# Estimate bandwidth.
bdw_ms <- estimate_band_W(hym_tree_discr, path_data_meso, band.width = 'bw.nrd')
bdw_ms <- mean(bdw_ms)

# Estimate Kernel Density Estimator (KDE).
cat(paste0("\n", "Starting estimating KDEs: ", Sys.time(), "\n"))
edge_KDE_ms <- estimate_edge_KDE(hym_tree_discr, Path.data = path_data_meso, h = bdw_ms)
cat(paste0("\n", "Finished estimating KDEs: ", Sys.time(), "\n"))
```

METASOMA

```r
# Make path data for all tips.
path_data_meta <- make_data_NHPP_KDE_Markov_kernel(path_hm_meta)

# Add pseudo-data.
# path_data_meta <- lapply(path_data_meta, function(x) c((-1*x), x) )

# Estimate bandwidth.
bdw_mt <- estimate_band_W(hym_tree_discr, path_data_meta, band.width = 'bw.nrd')
bdw_mt <- mean(bdw_mt)

# Estimate Kernel Density Estimator (KDE).
cat(paste0("\n", "Starting estimating KDEs: ", Sys.time(), "\n"))
edge_KDE_mt <- estimate_edge_KDE(hym_tree_discr, Path.data = path_data_meta, h = bdw_mt)
cat(paste0("\n", "Finished estimating KDEs: ", Sys.time(), "\n"))
```

PHENOME

```r
# Make path data for all tips.
path_data_pheno <- make_data_NHPP_KDE_Markov_kernel(path_hm_pheno)

# Add pseudo-data.
# path_data_pheno <- lapply(path_data_pheno, function(x) c((-1*x), x) )

# Estimate bandwidth.
bdw_ph <- estimate_band_W(hym_tree_discr, path_data_pheno, band.width = 'bw.nrd')
bdw_ph <- mean(bdw_ph)

# Estimate Kernel Density Estimator (KDE).
cat(paste0("\n", "Starting estimating KDEs: ", Sys.time(), "\n"))
edge_KDE_ph <- estimate_edge_KDE(hym_tree_discr, Path.data = path_data_pheno, h = bdw_ph)
cat(paste0("\n", "Finished estimating KDEs: ", Sys.time(), "\n"))
```

STEP 4. Post-process branch data and calculate rates.

Now we just need to extract the overall number of transitions from the amalgamated stochastic maps and calculate some statistics to make the Poisson distributions. Again, choose one chunk below to run. Let's run the chunk for the head first.

HEAD

```r
# Calculate smoothing and normalize KDE data.
edge_KDE_hd$Maps.mean.loess <- loess_smoothing_KDE(hym_tree_discr, edge_KDE_hd)
edge_KDE_hd$Maps.mean.loess.norm <- normalize_KDE(hym_tree_discr,
                                                  edge_KDE_hd$Maps.mean.loess)

# Calculate the lambda statistics of the Non-Homogeneus Poisson distribution.
lambda_post_hd <- posterior_lambda_KDE(stm_merg_head)

# Get the posterior distribution.
edge_KDE_hd$lambda.mean <- make_postPois_KDE(edge_KDE_hd$Maps.mean.norm,
                                             lambda_post_hd, lambda.post.stat = 'Mean')
edge_KDE_hd$lambda.mean.loess <- make_postPois_KDE(edge_KDE_hd$Maps.mean.loess.norm,
                                                   lambda_post_hd, lambda.post.stat = 'Mean')
```

MESOSOMA

```r
# Calculate smoothing and normalize KDE data.
edge_KDE_ms$Maps.mean.loess <- loess_smoothing_KDE(hym_tree_discr, edge_KDE_ms)
edge_KDE_ms$Maps.mean.loess.norm <- normalize_KDE(hym_tree_discr,
                                                  edge_KDE_ms$Maps.mean.loess)

# Calculate the lambda statistics of the Non-Homogeneus Poisson distribution.
lambda_post_ms <- posterior_lambda_KDE(stm_merg_meso)

# Get the posterior distribution.
edge_KDE_ms$lambda.mean <- make_postPois_KDE(edge_KDE_ms$Maps.mean.norm,
                                             lambda_post_ms, lambda.post.stat = 'Mean')
edge_KDE_ms$lambda.mean.loess <- make_postPois_KDE(edge_KDE_ms$Maps.mean.loess.norm,
                                                    lambda_post_ms, lambda.post.stat = 'Mean')
```

METASOMA

```r
# Calculate smoothing and normalize KDE data.
edge_KDE_mt$Maps.mean.loess <- loess_smoothing_KDE(hym_tree_discr, edge_KDE_mt)
edge_KDE_mt$Maps.mean.loess.norm <- normalize_KDE(hym_tree_discr,
                                                  edge_KDE_mt$Maps.mean.loess)

# Calculate the lambda statistics of the Non-Homogeneus Poisson distribution.
lambda_post_mt <- posterior_lambda_KDE(stm_merg_meta)
```

```r
# Get the posterior distribution.
edge_KDE_mt$lambda.mean <- make_postPois_KDE(edge_KDE_mt$Maps.mean.norm,
                                             lambda_post_mt, lambda.post.stat = 'Mean')
edge_KDE_mt$lambda.mean.loess <- make_postPois_KDE(edge_KDE_mt$Maps.mean.loess.norm,
                                                   lambda_post_mt, lambda.post.stat = 'Mean')
```

PHENOME

```r
# Calculate smoothing and normalize KDE data.
edge_KDE_ph$Maps.mean.loess <- loess_smoothing_KDE(hym_tree_discr, edge_KDE_ph)
edge_KDE_ph$Maps.mean.loess.norm <- normalize_KDE(hym_tree_discr,
                                                  edge_KDE_ph$Maps.mean.loess)

# Calculate the lambda statistics of the Non-Homogeneus Poisson distribution.
lambda_post_ph <- posterior_lambda_KDE(stm_merg_pheno)

# Get the posterior distribution.
edge_KDE_ph$lambda.mean <- make_postPois_KDE(edge_KDE_ph$Maps.mean.norm,
                                             lambda_post_ph, lambda.post.stat = 'Mean')
edge_KDE_ph$lambda.mean.loess <- make_postPois_KDE(edge_KDE_ph$Maps.mean.loess.norm,
                                                   lambda_post_ph, lambda.post.stat = 'Mean')
```

STEP 5. Make data for the contmaps and edgeplots.

Now, let's just prepare the data for plotting. Again, choose one chunk below to run. Let's run the chunk for the head first.

HEAD

```r
# Make data for contmaps.
nhpp_lambda_mean_hd <- make_contMap_KDE(hym_tree_discr,
                                        edge_KDE_hd$lambda.mean.loess)

# Make data for edge profiles.
edge_profs_lambda_mean_hd <- edge_profiles4plotting(hym_tree_discr,
                                                    edge_KDE_hd$lambda.mean.loess)
```

MESOSOMA

```r
# Make data for contmaps.
nhpp_lambda_mean_ms <- make_contMap_KDE(hym_tree_discr,
                                        edge_KDE_ms$lambda.mean.loess)

# Make data for edge profiles.
edge_profs_lambda_mean_ms <- edge_profiles4plotting(hym_tree_discr,
                                                    edge_KDE_ms$lambda.mean.loess)
```

METASOMA

```r
# Make data for contmaps.
nhpp_lambda_mean_mt <- make_contMap_KDE(hym_tree_discr,
                                        edge_KDE_mt$lambda.mean.loess)

# Make data for edge profiles.
edge_profs_lambda_mean_mt <- edge_profiles4plotting(hym_tree_discr,
                                                    edge_KDE_mt$lambda.mean.loess)
```

PHENOME

```r
# Make data for contmaps.
nhpp_lambda_mean_ph <- make_contMap_KDE(hym_tree_discr,
                                        edge_KDE_ph$lambda.mean.loess)

# Make data for edge profiles.
edge_profs_lambda_mean_ph <- edge_profiles4plotting(hym_tree_discr,
                                                    edge_KDE_ph$lambda.mean.loess)
```

STEP 6. Plotting contmaps and edgeplots.

Assuming you just ran the chunks for the head, let's plot the contmap and edgeplot saving to a PNG file.

```r
# Create a folder to store figures.
dir.create("figures")

# HEAD.
# Save png.
png(paste0("figures/edgeplot_head.png"),
    units = "in", width = 7, height = 7, res = 300)

edgeplot(nhpp_lambda_mean_hd, edge_profs_lambda_mean_hd)
title(main = "HEAD", font.main = 2, line = -0.5, cex.main = 0.8)

dev.off()
```

If you already ran all the other chunks (i.e. head, mesosoma, metasoma, and phenome), then let's plot a nice figure with everything we got so far. Just run the chunk below.

```r
# MESOSOMA.
# Save png.
png(paste0("figures/edgeplot_meso.png"),
    units = "in", width = 7, height = 7, res = 300)

edgeplot(nhpp_lambda_mean_ms, edge_profs_lambda_mean_ms)
title(main = "MESOSOMA", font.main = 2, line = -0.5, cex.main = 0.8)

dev.off()
```

```r
# METASOMA.
# Save png.
png(paste0("figures/edgeplot_meta.png"),
    units = "in", width = 7, height = 7, res = 300)

edgeplot(nhpp_lambda_mean_mt, edge_profs_lambda_mean_mt)
title(main = "METASOMA", font.main = 2, line = -0.5, cex.main = 0.8)

dev.off()

# PHENOME.
# Save png.
png(paste0("figures/edgeplot_pheno.png"),
    units = "in", width = 7, height = 7, res = 300)

edgeplot(nhpp_lambda_mean_ph, edge_profs_lambda_mean_ph)
title(main = "PHENOME", font.main = 2, line = -0.5, cex.main = 0.8)

dev.off()
```

And finally, save all the results.

```r
save.image("RData/step3_nhpp.RData")
```

References

Ross, G. J. and Markwick, D. (2018). dirichletprocess: An r package for fitting complex bayesian nonparametric models.

Taddy, M. A. and Kottas, A. (2012). Mixture modeling for marked poisson processes. *Bayesian Analysis*, 7(2):335–362.

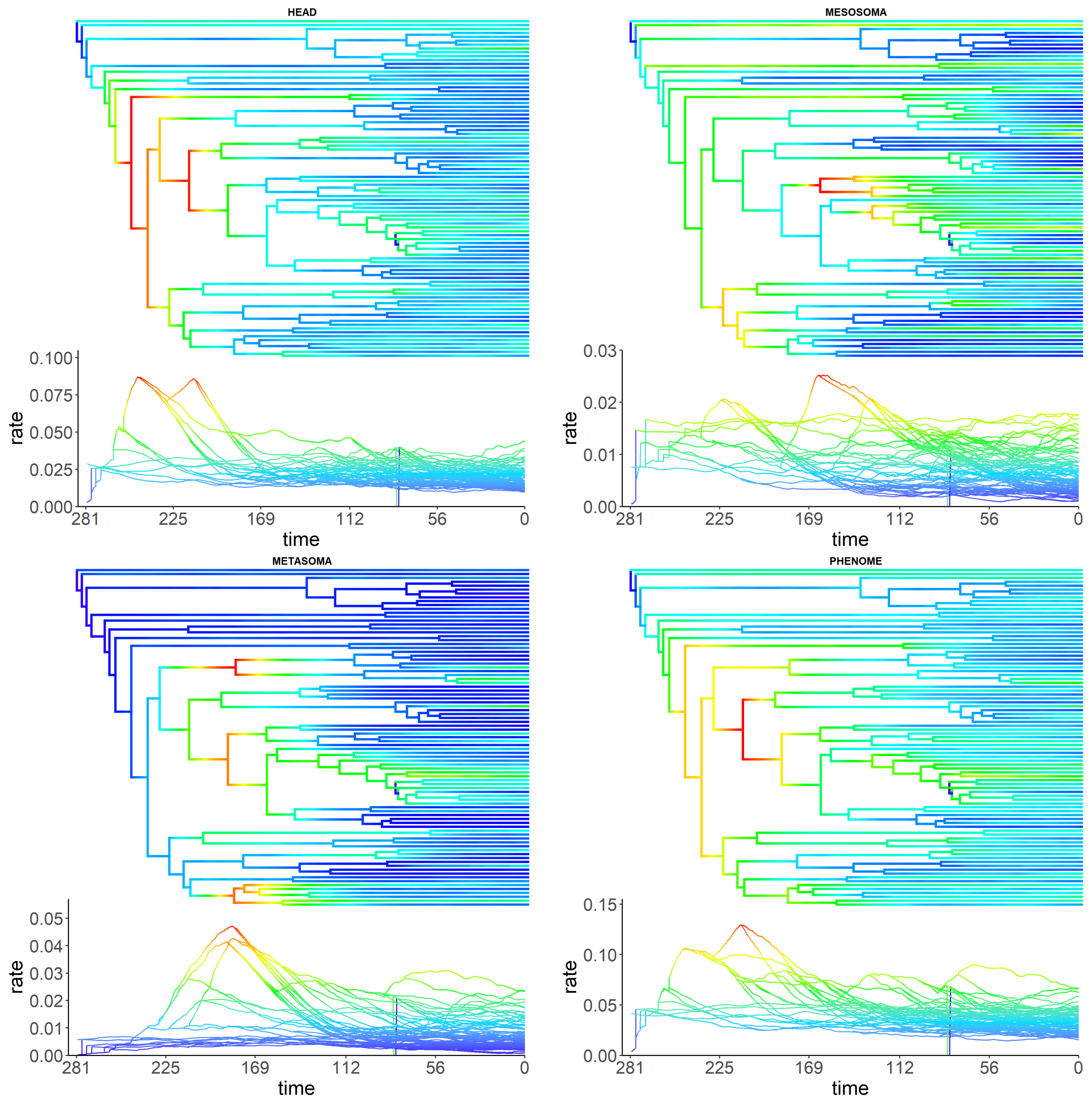

Figure 1: From left to right, top to bottom, branch rates of head, mesosoma, metasoma, and phenome.