A guide to the use of tRophicPosition

Claudio Quezada-Romegialli, Andrew L Jackson & Chris Harrod

12th of June 2016

Contents

Introduction **2**

What is TP? . . . 2

The Bayesian model behind `tRophicPosition` . . . 2

Future releases and how to get support . . . 2

Installing `tRophicPosition` **2**

From GitHub . . . 3

From a package archive file . . . 3

From CRAN . . . 3

Working with `tRophicPosition` **3**

Loading and saving data **5**

Loading data from the clipboard . . . 5

Loading data from a csv/txt file . . . 6

Saving the data . . . 7

Isotope Data **8**

Screening your isotope data – visual exploration . . . 8

Working with data frames . . . 9

Generating TEFs . . . 11

Defining the Bayesian model for trophic position **13**

Only one baseline . . . 14

Two baselines model, only using the TEF for N . . . 15

Two baselines model, including TEFs for both N and C . . . 15

Initializing the model and sampling the posterior estimates **15**

Initializing the Bayesian model . . . 16

Sampling and plotting the posterior of trophic position . . . 18

Resuming: trophic position in a nutshell **20**

Comparing two trophic positions . . . 23

Introduction

In this vignette, we will guide you through the necessary steps to estimate trophic position (TP) of groups of consumers using stable isotope ratios, in a Bayesian framework.

What is TP?

Stable isotopes have proven ability to capture the complexity of trophic interactions, to identify energy sources and to infer consumer trophic position. **Trophic position** is an important concept used to describe the ecological role of consumers in food webs – it tells us on average how many tropic levels are there between the consumer and the primary producers fueling the food web. However, current methods used to estimate TP using δ^{15}N stable isotopes are limited and do not fulfil the full potential of the isotopic approach.

For instance, researchers typically use a single point estimate for the key parameters involved in estimating TP such as trophic enrichment factors (isotopic difference between the consumer's prey and its tissues) and isotopic baselines (the isotopic value of the basal indicator – typically primary producers or a primary consumer), and often do not explicitly include within-population variation i.e. that shown by individual consumers. There is a marked need to include such variation both in the calculation of trophic position, but also to provide robust estimates of trophic position (e.g. measures of central tendency and variance) to allow e.g. statistical comparisons between species or across space and time.

The Bayesian model behind `tRophicPosition`

In this vignette we present an R package `tRophicPosition`, incorporating a Bayesian model for the calculation of trophic position using stable isotopes with one or multiple baselines. This uses the powerful approach of Markov Chain Monte Carlo simulations provided by JAGS and the statistical language R. We model consumer and baseline observations using relevant statistical distributions, allowing them to be treated as random variables. The calculation of trophic position – a random parameter– for one baseline follows standard equations linking δ^{15}N enrichment per trophic level and the trophic position of the baseline (e.g. a primary producer or primary consumer). In the case of two baselines, a simple mixing model incorporating δ^{13}C allows for the differentiation between two distinct sources of nitrogen, thus including spatial heterogeneity derived from alternatives sources of δ^{15}N.

Future releases and how to get support

As of 2016-05-26, we have released a beta-testing version of `tRophicPosition` (version 0.3.5.9000). You are encouraged to use it with your own data, test the package and see if there are any issues or problems. You can send your questions or commentaries to the google group tRophicPosition-support or directly to the email trophicposition-support@googlegroups.com. You can send your questions to http://stackexchange.com/ http://stackoverflow.com/ or even Facebook.

We are working on future releases of `tRophicPosition`, so feedback is very much appreciated.

Installing `tRophicPosition`

In order to use `tRophicPosition`, first you have to install it. There are a range of options including using GitHub, a package archive file or CRAN (not currently available). Once installed you need to load the library into memory.

From GitHub

The easiest way to install the latest version of `tRophicPosition` is to install it from GitHub. In this way you always will have the most up to date code. You must install the release version of devtools from CRAN, and RTools. If you are using Windows, install RTools from here: http://cran.r-project.org/bin/windows/Rtools/. For other operating systems (Linux, Mac) refer to the README of devtools.

To install the devtools package from CRAN:

```
install.packages(devtools)
```

As of 2016-06-12, the current version of tRophicPosition available on GitHub is 0.3.8.9000. To install `tRophicPosition` use this code:

```
library(devtools)
install_github("AndrewLJackson/tRophicPosition")
```

From a package archive file

Another option is to install `tRophicPosition` from a previously saved release version, or a beta-testing version, available from a file stored in your computer.

```
install.packages(file.choose(), repos = NULL)
```

From CRAN

When the package is finally available on CRAN, you will be able to install it using this code:

```
install.packages("tRophicPosition")
```

Working with `tRophicPosition`

Once you have installed the package, you have to load it into memory:

```
library(tRophicPosition)
```

And then you can check the version you have installed:

```
packageDescription("tRophicPosition")$Version
```

```
## [1] "0.3.5.9000"
```

The basic use of `tRophicPosition` is outlined in Figure 1. Briefly, it involves:

1. Load isotope data into R: from csv, txt (both common extensions of text files) or xlsx/ods (spreadsheet) files, the clipboard or other preferred source you like. Data can be organized either as a data frame with 4 variables (δ^{13}C, δ^{15}N, Functional Group and Species) or a named list (N and C isotope values for your consumer, one or more baselines, and trophic enrichment factors). We will cover how to load and manipulate data in detail in this vignette.

2. Define a Bayesian model for the calculation of trophic position. Here you have to select a model that reflects your scientific question from the following options:

- One baseline model, using the `jagsOneBaseline()` function,
- Two baselines model using only the trophic enrichment factor (TEF) for N, using the `jagsTwoBaselines()` function, or
- Two baselines model incorporating TEFs for both C and N, using the `jagsTwoBaselinesFull()` function.

3. Initialize the model: feed the Bayesian model defined in (2) with data. Optionally you can define the number of parallel chains and the number of adaptive iterations you want. These options (as well as the Bayesian model and the data) are entered as arguments to the function `TPmodel()`.

4. Once the model has run, sample the posterior Trophic Position: get the posterior estimates of trophic position with the function `posteriorTP()`. Here you can define others parameters you want to examine, as well as trophic position.

5. Analyse and plot the data, using `summary()`, `plot()`, `trophicDensityPlot()` and other functions.

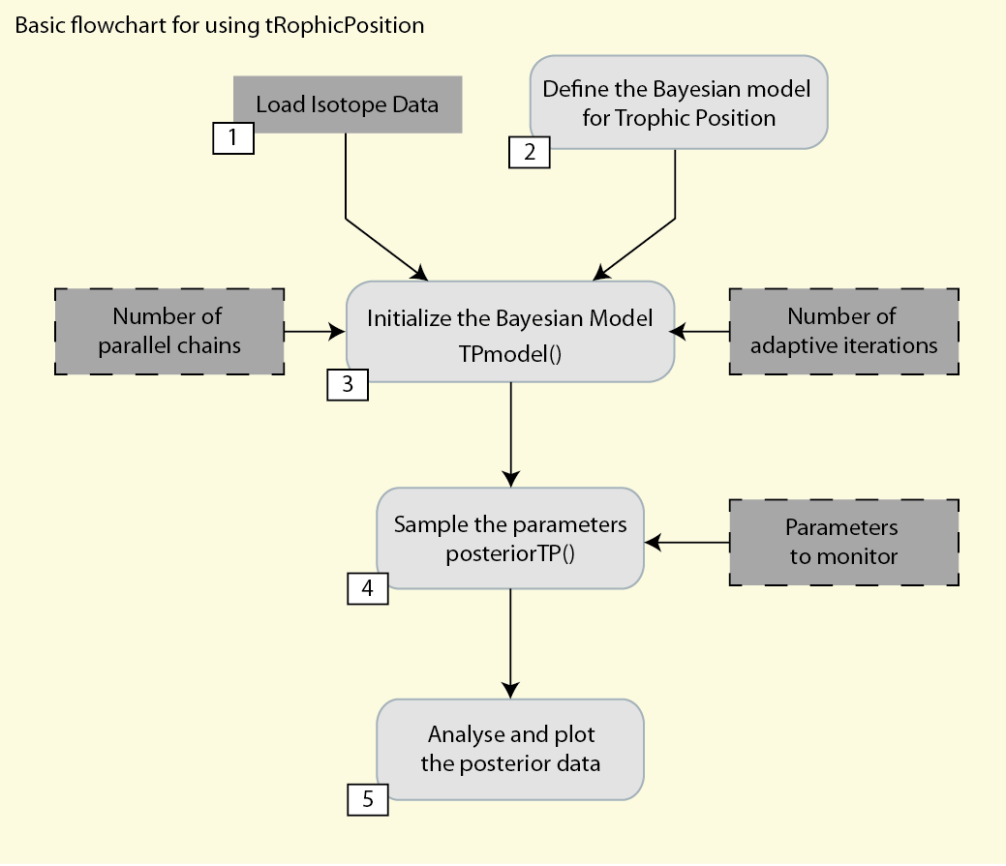


Figure 1: Basic flowchart of trophic position estimation.

Figure 1 shows a basic flow chart describing these steps. The rounded rectangles (light grey) refers to functions inside or outside the package, the grey rectangle refers to input data, and the grey rectangles with broken lines refer to optional arguments for each function.

As an example, in order to initialize the Bayesian model you just need to load isotope data and have previously defined the Bayesian model itself. Establishing the number of baselines and defining the number of parallel

chains is optional. These two arguments are defined by default in the `TPmodel()` function. Each of the functions of `tRophicPosition` have a help page, that you can refer to if you are not sure how to proceed. To access the relevant help page, simply write a question mark before the function name and press enter in the console: e.g. `?TPmodel`.

Loading and saving data

Although loading/saving data for all the packages available in R would be a desirable task, most packages use their own data formats. This is especially true with isotope data. We are working in a standardized way to load/save isotope data, but in the meantime, here we provide some examples on how to load isotope data into R, and get it ready to work within `tRophicPosition`. Also, if you want to reproduce the examples, or conduct your own data analysis, this task will be much easier if you save your data properly.

Loading data from the clipboard

In order to work with `tRophicPosition`, you can have your isotope data in a spreadsheet, organized like this:

dCc	dNc	dCb1	dNb1	dCb2	dNb2	deltaC	deltaN
-12,8	9,1	-18	4,1	-6,8	2,9	2,6	1,1
-12,9	9,3	-15,7	3,9	-12,6	4,2	2,6	1,2
-13,3	8,8	-18,8	4,5	-8,6	2,7	2,6	2,3
NA	NA	NA	NA	-9,5	3,4	2,6	1,9

In this example, each column of the spreadsheet may have several isotope values, and each column may have a different number of values as well. An example of a spreadsheet formatted like this can be found here.

You can copy to the clipboard a table like the one in the example, with your **own data** directly from the spreadsheet, and probably this is the easiest way to enter data into R. Besides, if you enter data like this (using the exact headers in the example above), it will be in the exact **format required** by `tRophicPosition`. The headers stands for the following, *dCc* and *dNc* are both δ^{13}C and δ^{15}N of the consumer we want to calculate the trophic position of, *dCb1* is δ^{13}C of the baseline 1, *dNb1* is δ^{15}N of the baseline 1, *dCb2* is δ^{13}C of the baseline 2, *dNb2* is δ^{15}N of the baseline 2, *deltaC* is ΔC (TEF) for carbon and *deltaN* is ΔN (TEF) for nitrogen. The order of the columns does not have to be the same, but all the columns must be there if you have two baselines available. If you have only one baseline, then you can have less columns (at least dCc, dNc, dNb1, dCb1 and deltaN if you want to plot the data).

Once you have selected all the data (remember to check that you have 8 columns in the spreadsheets example, copy it to the clipboard, and then use this code in R:

```
YourOwnDataSet <- read.table("clipboard", header=TRUE, sep="\t",
                        dec=",", strip.white=TRUE)
```

Remember to look for differences in **regional settings**. For instance, the package was written in Chile where we use a comma "," as our indicator of a decimal. In the Anglo world, a decimal point "." is used instead. You have to change the decimal (comma to point) if you are using a different locale than Spanish (change `dec=","` to `dec="."`). The **separator** `"\t"` stands for the tab key (the one that most spreadsheets uses to separate cells). **Header** is `TRUE` because we copied the headers from the spreadsheet. This is important, as we require that each variable has its own name in R.

Once you have entered the data correctly into R, try this code to see if that is true:

```
YourOwnDataSet
```

```
##       dCc  dNc   dCb1 dNb1   dCb2 dNb2 deltaN deltaC
## 1  -12.86 9.08 -18.01 4.09  -6.76 2.87   2.64   1.05
## 2  -12.94 9.27 -15.75 3.91 -12.70 4.22   2.60   1.21
## 3  -13.31 8.79 -18.89 4.45  -8.63 2.67   2.61   2.26
## 4  -13.75 8.94 -16.03 4.98  -8.65 2.90   2.69   1.07
## 5  -14.72 8.31 -15.19 4.25     NA   NA   2.10   1.11
## 6  -12.28 8.62     NA   NA     NA   NA   2.38   1.97
## 7  -13.23 8.39     NA   NA     NA   NA   3.09   0.90
## 8  -16.09 7.96     NA   NA     NA   NA   3.00   0.35
## 9  -14.91 8.14     NA   NA     NA   NA   2.48   0.76
## 10 -13.42 8.61     NA   NA     NA   NA   2.86   0.81
## 11 -15.08 8.57     NA   NA     NA   NA     NA     NA
## 12 -13.62 8.85     NA   NA     NA   NA     NA     NA
## 13 -13.26 8.56     NA   NA     NA   NA     NA     NA
## 14 -14.49 8.53     NA   NA     NA   NA     NA     NA
## 15 -13.16 8.34     NA   NA     NA   NA     NA     NA
## 16 -14.26 9.21     NA   NA     NA   NA     NA     NA
## 17 -11.96 7.99     NA   NA     NA   NA     NA     NA
## 18 -14.40 8.44     NA   NA     NA   NA     NA     NA
## 19 -14.91 8.65     NA   NA     NA   NA     NA     NA
```

You can access every column from "YourOwnDataSet" adding a "$" and the name of the variable at the end, for example try:

```
YourOwnDataSet$dCc
```

Loading data from a csv/txt file

Another possibility is to have the data in a plaintext ASCII file, either as a comma separated value (csv) or as a plaintext (txt) file. It is worth noting that the filename extension not necesarilly reflects the format of the file. If you change the extension of a ".txt" file into ".csv", the content and the format of the file will remain as plaintext and will not change into a "comma separated value (csv)" file. Here we mention "txt file" not referring to its extension, but to the format of the file. Same applies for csv or xls/xlsx files (in the latter, we refer to xls/xlsx as a standard spreadsheet, not necesarilly referring to the propietary format/file extension of Microsoft Excel).

In the following example, we will enter data into R from a txt tab delimited file that has the following structure (download this txt example from here):

Species	FG	d13C	d15N
Amphipod	Baseline	-18.0	4.1
Amphipod	Baseline	-15.7	3.9
Bivalve	Baseline	-6.8	2.9
Bivalve	Baseline	-12.7	4.2
Bivalve	Baseline	-8.6	2.7
Bivalve	Baseline	-8.7	2.9

This file includes data from two species of fish - one endemic native (killifish *Orestias chungarensis*) and one invasive (juvenile rainbow trout *Oncorhynchus mykiss*) that live sympatrically in a high altitude small river

(Río Chungará) in the North of Chile. Also included are invertebrates that we will use to form an isotopic baseline (primary consumers with a putative trophic level of 2). We will use these data to estimate the trophic position of the two fishes and to examine if there are differences between the native and invasive species.

As you see, this txt file has 4 columns. The first column details the species, then the second column has the functional group, and the last two columns have the isotope values. In order to load a txt file organized like this txt file we will use the following code:

```
# This code will open up a dialog to choose the file you just downloaded
Altiplano <- read.table(file.choose(), header=TRUE, sep="\t",
                        dec=".", strip.white=TRUE)
```

The function `file.choose()` opens a dialog to choose a file interactively rather than typing the file name itself. If you do not like to use the dialog to open the file, you can use the same code as above, but changing `file.choose()` to the name of the file:

```
Altiplano <- read.table("Altiplano.txt", header=TRUE, sep="\t", dec=".", strip.white=TRUE)
```

If everything went well, we can check the structure of the data:

```
str(Altiplano)
```

```
## 'data.frame':    62 obs. of  4 variables:
##  $ Species: Factor w/ 7 levels "Amphipod","Bivalve",..: 1 1 2 2 2 2 3 4 7 5 ...
##  $ FG     : Factor w/ 2 levels "Baseline","Consumer": 1 1 1 1 1 1 1 1 1 2 ...
##  $ d13C   : num  -18.01 -15.75 -6.76 -12.7 -8.63 ...
##  $ d15N   : num  4.09 3.91 2.87 4.22 2.67 ...
```

When you open a txt file with the `read.table()` function, you get a data.frame object, and in this case it has 62 observations of 4 variables. In this example `Species` provides information on the species examined, `FG` stands for the functional group of each species (i.e. baselines or consumers), and `d13C` and `d15N` are the stable isotope values associated with each sample.

Saving the data

If you could copy some data from the spreadsheet to the clipboard, and then you were able to enter it into R (with the `read.table("clipboard")` function), you will have an object stored in the *R Environment*. If you did not change the name, that object is called `YourOwnDataSet`. This object is a data frame.

If you downloaded the txt file and you could open it (using `read.table(file.choose())`), you will also have an object called `Altiplano`, which is also a data frame.

If you want to save the data from the spreadsheet to reuse it later, you can use the `save()` function. In order to save an object you have to use a code like this:

```
save(Altiplano, file = "Altiplano_fish_from_txt_file")
```

If you try to open the saved file in a text editor, you will see that it is a binary and compressed file, so you may not be able to see it easily outside R. So you have to check and double check what you are saving to a file when you do this.

Another important point when you save (and afterwards load) a file, is that you have to know **where** you are saving your files. So, you must know previously which is your working directory, and probably you might want to change it as well.

To know which is your working directory use `getwd()`. To change your working directory, use `setwd("C:/path/to/your/directory")`. Once you change the working directory to your preferred directory, you can save your data, and then open it at a later date. Probably you will want to save all your work in a script, making further analysis easier.

If you want to save `Altiplano` to a tab separated (tsv) file, you can use this code:

```
write.table(Altiplano, "mydata.txt", sep="\t")
```

In this case "mydata.txt" file has a ".txt" extension, but it is a tab delimited plaintext file. If you want to save `Altiplano` as a csv file, use `write.csv()` instead.

Isotope Data

In the above examples we learned to load data from the clipboard/spreadsheet, and also we learned to load some data from a text file.

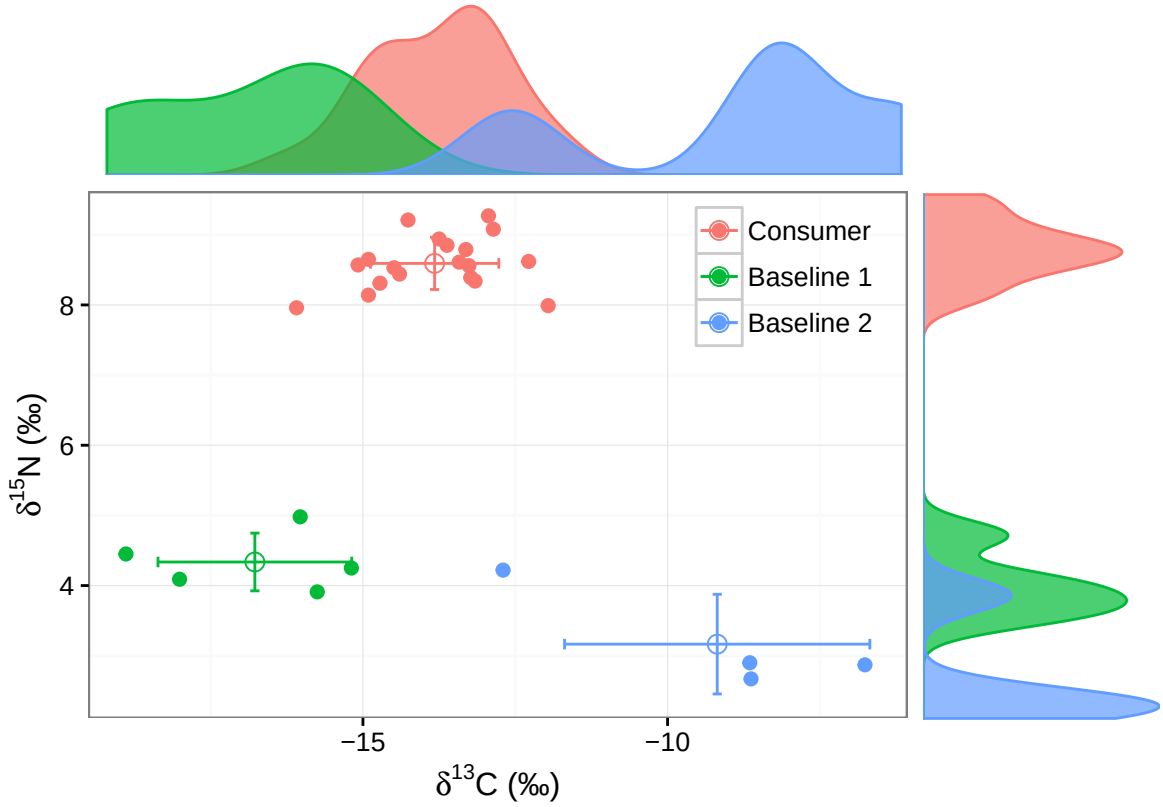
If you have previously organized your isotope data as seen in the spreadsheet in the example above, it should run smoothly in R, and you can use all the functions within the package. However, first we need to remove `NA` values (R's code for missing data). This is done using this code:

```
IsotopeData <- lapply(YourOwnDataSet, na.omit)
```

Screening your isotope data – visual exploration

After removing `NA` values, you can use the `tRophicPosition` function `screenIsotopeData()` to undertake some initial graphical analysis:

```
screenIsotopeData(IsotopeData)
```



This stable isotope biplot is largely self-explanatory. The main plot includes values for the consumer, the baseline 1 and the baseline 2. This plot also includes the centroid (bivariate mean) ± standard deviation bars to provide a feel for the average and distribution of the data. At the top and to the right, you will see a density function representing a smoothed density estimate. The key information from the density plot above (representing the δ^{13}C) is that you will clearly see which one of the two putative baselines is the likely source of the energy that is fuelling your species. The density to the right, represents the degree of overlap between your two sources in δ^{15}N space.

In this example, the isotope data comes from the High Andes of Chile, the Altiplano. The secondary consumer here is the native fish *Orestias chungaraensis*, and the two baselines represent 2 distinct groups of macroinvertebrate baselines (lambda = 2 or putative primary consumers) available at the time of sampling. We will return to this example later.

Working with data frames

If you load data from a txt/csv file or other source, and have your species and baselines stacked, then the data will require some manipulation before it is ready for analysis. Using the Altiplano.txt example above, we can summarise its content:

```
summary(Altiplano)
```

```
##           Species          FG          d13C              d15N
##  Amphipod     : 2   Baseline: 9   Min.   :-22.187   Min.   :2.675
##  Bivalve      : 4   Consumer:53   1st Qu.:-15.588   1st Qu.:6.207
##  Coleoptra    : 1                 Median :-14.473   Median :7.192
```

```
##  Corixid      : 1                Mean   :-14.353   Mean   :6.969
##  Orestias     :19                3rd Qu.:-13.179   3rd Qu.:8.374
##  Rainbow trout:34                Max.   : -6.757   Max.   :9.273
##  Stonefly     : 1
```

We will split this data frame into 2 named lists (one for the *Orestias*, and one for the rainbow trout), so we can use them with the functions available from `tRophicPosition`. First, we will extract δ^{13}C and δ^{15}N values for baseline 1 and baseline 2:

```
# We need to separate the baselines
# First we select d15N and d13C for baseline 1
dNb1 <- Altiplano$d15N[which(Altiplano$FG == "Baseline" & Altiplano$Species != "Bivalve")]
dCb1 <- Altiplano$d13C[which(Altiplano$FG == "Baseline" & Altiplano$Species != "Bivalve")]

# Then we select d15N and d13C for baseline 2
dNb2 <- Altiplano$d15N[which(Altiplano$Species == "Bivalve")]
dCb2 <- Altiplano$d13C[which(Altiplano$Species == "Bivalve")]
```

In the code above we selected some values from the column `Altiplano$d15N` considering functional group and the species identity. So, `dNb1` now has the values of δ^{15}N associated with "Baseline" functional group (`Altiplano$FG == "Baseline"`) and (`&`) at the same time whose identity is different of "Bivalve" (`Altiplano$Species != "Bivalve"`).

```
dNb1
```

```
## [1] 4.090708 3.914633 4.451877 4.976481 4.245888
```

You can see the values for `dCb1`, `dNb2` and `dCb2` as well:

```
dCb1
```

```
## [1] -18.01197 -15.74571 -18.88836 -16.02859 -15.19338
```

```
dNb2
```

```
## [1] 2.871996 4.223670 2.674563 2.896213
```

```
dCb2
```

```
## [1]  -6.756632 -12.696298  -8.627630  -8.654645
```

Now we want to extract the fish species. This is done with a similar code, but changing the Species value. So, for *Orestias* we write:

```
dN_Orestias <- Altiplano$d15N[which(Altiplano$Species == "Orestias")]
dC_Orestias <- Altiplano$d13C[which(Altiplano$Species == "Orestias")]
```

And now we combine these variables together, in a named list (could be a data frame as well):

```
Orestias <- list("dNb1" = dNb1, "dCb1" = dCb1, "dNb2" = dNb2,
                 "dCb2" = dCb2, "dNc" = dN_Orestias, "dCc" = dC_Orestias)
```

Why is important to make a named list? Because when we feed the Bayesian model, we have to explicitly state which data we have. With this approach, we know exactly what the values are for each variable.

Now that we have extracted the isotope values for the *Orestias*, we need to do the same for the rainbow trout:

```
# Here we extract all the isotope values filtering for Species == "Rainbow trout"
dN_Trout <- Altiplano$d15N[which(Altiplano$Species == "Rainbow trout")]
dC_Trout <- Altiplano$d13C[which(Altiplano$Species == "Rainbow trout")]

# And we create a named list
Trout <- list("dNb1" = dNb1, "dCb1" = dCb1, "dNb2" = dNb2,
              "dCb2" = dCb2, "dNc" = dN_Trout, "dCc" = dC_Trout)
```

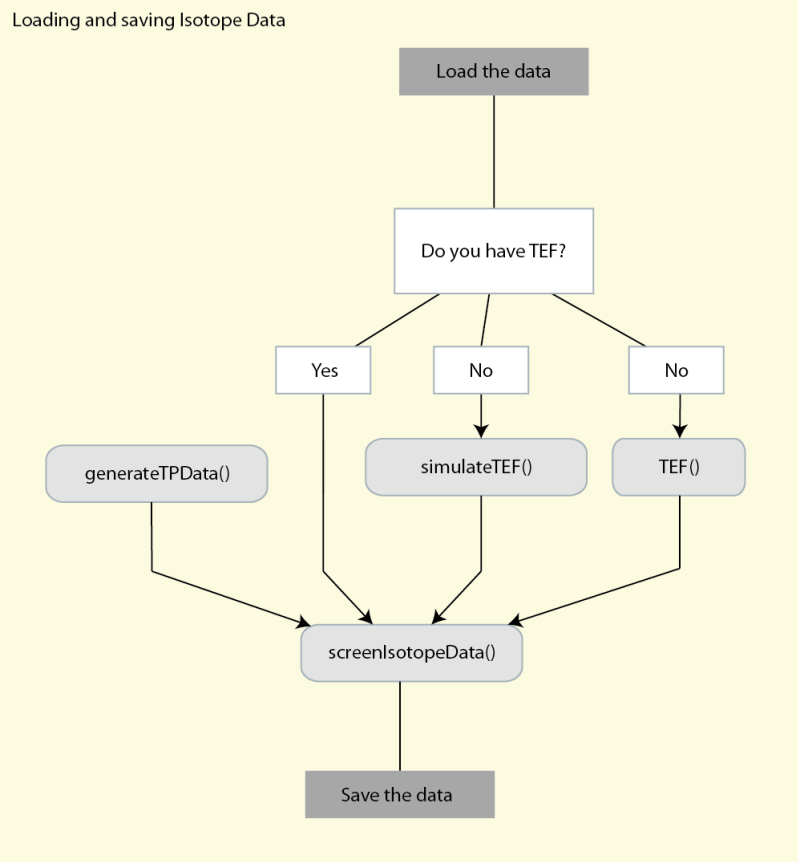


Figure 2: Basic flowchart of loading and saving isotope data.

Generating TEFs

At this point we have manipulated a data frame that has stacked information of two fish species, and two baselines. we are missing an essential component of the model to estimate TP, namely the trophic enrichment factor (TEF). We have several options – we can simply use ΔN (the TEF for nitrogen) when using a single

or dual baseline model, or we can include ΔC to account for shifts in C between baseline and consumers (as Post 2002).

As to the actual TEFs used in the model, there are 3 options (Figure 2):

1. Use your own empirical values for TEFs, that correspond to an experiment for your model organism;
2. Use 'standard' TEFs from the inbuilt bibliography in `tRophicPosition` (e.g. Post 2002, McCutchan et al. 2003); or
3. Use a mean, SD and n to simulate some TEF values, i.e. to follow a published TEF for your taxa of interest.

In this flow chart, light grey rectangles refer to functions within `tRophicPosition`, grey rectangles are input/output data, and white rectangles are questions on how to proceed.

Generating TEFs from meta-analysis

Getting accurate and relevant TEFs for your taxa of interest can be sometimes difficult. You can have them in your text file or spreadsheet and load them into R, or you can use two functions we have built into `tRophicPosition` to provide values.

If you do not have an approximate idea of the mean and standard deviation of the TEF for your organism, then you can use the function `TEF()`. This provides the overall mean ± SD values provided from Post's (2002) meta-analysis, which are widely used in the literature. Without giving any argument (i.e. using `TEF()`), this function returns 56 values with a mean 3.4 ± 0.98 SD values for ΔN, and 107 values with a mean 0.39 ± 1.3 SD for ΔC.

We have also included information from another major review (McCutchan et al. 2003), which includes estimates of TEFs for different tissues and trophic functional groups. To use these TEFs we have to state the `author` argument as `"McCutchan"`, the tissue type argument (`type` can be `"muscle"`, `"whole"`, `"acidified"`, `"unacidified"`, `"Rainbow Trout"` or `"Brook Trout"`) and if we want TEFs for both N and C (`element = "both"`) or by element, in which case `element` would be either `"N"` or `"C"`. As usual, as this function returns a list, you have to assign it to a variable:

```
McCutchanTEF <- TEF(author = "McCutchan", type = "muscle", element = "both")
```

```
## You select McCutchan's et al (2003) muscle tissue d15N: 73 values with 2.3 mean +- 0.18 se
```

```
## and also McCutchan's et al (2003) muscle tissue d13C: 18 values with 1.3 mean +- 0.3 se
```

If you want to add them to a data frame or to a named list you have previously created, the procedure is quite simple. For example:

```
str(Orestias)
```

```
## List of 6
##  $ dNb1: num [1:5] 4.09 3.91 4.45 4.98 4.25
##  $ dCb1: num [1:5] -18 -15.7 -18.9 -16 -15.2
##  $ dNb2: num [1:4] 2.87 4.22 2.67 2.9
##  $ dCb2: num [1:4] -6.76 -12.7 -8.63 -8.65
##  $ dNc : num [1:19] 9.08 9.27 8.79 8.94 8.31 ...
##  $ dCc : num [1:19] -12.9 -12.9 -13.3 -13.7 -14.7 ...
```

Orestias is a named list with 6 variables: dNb1, dCb1, dNb2, dCb2, dNc and dCc. As that list doesn't have TEF (what we refer to as deltaN or deltaC), we may want to add one, e.g. the one we previously generated from the bibliography:

```
# In this line we add deltaN of McCutchanTEF to Orestias
Orestias$deltaN <- McCutchanTEF$deltaN

# and with this line we do the same but with deltaC
Orestias$deltaC <- McCutchanTEF$deltaC
```

And now the named list `Orestias` has now 8 variables and is ready for subsequent modelling of TP:

```
str(Orestias)
```

```
## List of 8
##  $ dNb1  : num [1:5] 4.09 3.91 4.45 4.98 4.25
##  $ dCb1  : num [1:5] -18 -15.7 -18.9 -16 -15.2
##  $ dNb2  : num [1:4] 2.87 4.22 2.67 2.9
##  $ dCb2  : num [1:4] -6.76 -12.7 -8.63 -8.65
##  $ dNc   : num [1:19] 9.08 9.27 8.79 8.94 8.31 ...
##  $ dCc   : num [1:19] -12.9 -12.9 -13.3 -13.7 -14.7 ...
##  $ deltaN: num [1:73] 0.738 2.258 2.097 1.345 4.862 ...
##  $ deltaC: num [1:18] -0.00452 1.69535 3.39986 0.81836 1.18088 ...
```

Simulating TEF given a mean and standard deviation

We might want to add TEF for a species based on a particular study that provides mean ± SD TEFs values rather than using those calculated from a particular meta-analyses (i.e. Post 2002 or McCutchan et al. 2003). In this case, we can simulate data with the function `simulateTEF()`. If you do not state any arguments, by default this function returns 56 values for nitrogen and 107 for carbon - which are the same values used by Post. However, as an example we will use a mean ± SD TEF of 1.5 ± 1 for ΔN and 0.25 ± 0.5 for ΔC (n = 15 for both):

```
simulatedTEFValues <- simulateTEF(nN = 15, meanN = 1.5, sdN = 1,
                                  nC = 15, meanC = 0.25, sdC = 0.5)
```

In case we want to include those values to a named list that doesn't have TEFS values, or if we want to replace them, we can do the following:

```
YourOwnNamedList$deltaN <- simulatedTEFValues$deltaN
YourOwnNamedList$deltaC <- simulatedTEFValues$deltaC
```

In this example we have replaced the `deltaN` and `deltaC` values of `YourOwnNamedList` (that does not exists) with those simulated above and saved into the data frame `simulatedTEFValues`. If you try to run this code it will not work, obviously.

Defining the Bayesian model for trophic position

Now that we have introduced the basics of loading and saving isotope data, and you are able to generate or simulate trophic enrichment factors values, we are ready to get into the core of `tRophicPosition`. Depending on how many baselines you have, and if you want to use the simple (ΔN only), dual (ΔN, two baselines) or the full (both ΔN and ΔC, two baselines) model, you may choose among the three Bayesian models we have implemented in `tRophicPosition`. This is shown in Figure 3.

Only one baseline

If you want to implement the simplest model, then `jagsOneBaseline()` is the function you need to use. This function by default returns a string defining a Bayesian model with uninformative *priors*. This model implements an estimation of trophic position using the following equation:

$$(1)\ \ \delta^{15}N_c = \delta^{15}N_b + \Delta N(TP - \lambda)$$

where $\delta^{15}N_c$ refers to the consumer that we want to estimate TP for, $\delta^{15}N_b$ refers to the baseline, ΔN is the TEF for nitrogen, TP is the trophic position of the target consumer, and λ is the trophic position of the baseline.

Both $\delta^{15}N_c$ and $\delta^{15}N_b$ are modelled as having a normally distributed mean μ_c and μ_b respectivelly. They are both also modelled with a uniform *prior* for the standard deviation (σ) between 0 and 100. The standard deviation is modelled (because of the implementation of BUGS) as tau (τ), which in turn is calculated as ($\tau_b = 1/\sigma_b$).

Without stating any argument, `jagsOneBaseline()` returns a jags model as a character string that you have to assign to a variable. As such, we can use the following:

```
model.string <- jagsOneBaseline()
```

By default, `jagsOneBaseline()` (and the other more complex models) uses $\lambda = 2$ if you do not explicitly define it (i.e. that we use a primary consumer as our isotopic baseline). If, for example, you want to calculate the trophic position for a species whose baseline is a primary producer (e.g. phytoplankton), then you have to define $\lambda = 1$. This is done with:

```
model.string <- jagsOneBaseline(lambda = 1)
```

If you have some information for your organism (given stomach content analysis for example), you may define a *prior* distribution for the trophic position of it. By default trophic position is calculated as a random parameter with an uniform prior with lower and upper boundaries between λ and 10. If $\lambda = 2$, then by default this *prior* gives you an uniform *prior* distribution between 2 and 10.

If you have information that your species has a trophic position of 3, given the stomach content analysis, then you can define a *prior* distribution that includes your previous knowledge in order to improve the calculation of the trophic position. This is done either with:

```
# defining a uniform prior between lambda and 4
model.string <- jagsOneBaseline(TP = "dunif(lambda, 4)", lambda = 1)
```

or giving a normal prior (or other distribution you may think appropriate)

```
# defining a normal prior distribution with mean 3 and sd 0.5, with lambda = 1.
model.string <- jagsOneBaseline(TP = "dnorm(3, 0.5)", lambda = 1)
```

You can define the *priors* for the mean of the baseline (`muB`), the standard deviation for the baseline (`sigmaB`), the mean of the ΔN (`muDeltaN`), standard deviation for ΔN (`sigmaDeltaN`), and both trophic position (`TP`) and λ (`lambda`).

Two baselines model, only using the TEF for N

This model implements a different equation for the calculation of trophic position, for a situation where there are two distinct sources of C and N, e.g. pelagic vs benthic or terrestrial vs aquatic:

$$(2)\ \ \delta^{15}N_c = \Delta N(TP + \lambda) + \alpha(\delta^{15}N_{b1} + \delta^{15}N_{b2}) - \delta^{15}N_{b2}$$

And also includes a simple mixing model to calculate α, which allows to estimate the relative contribution of each source to the consumer's trophic position:

$$(3)\ \ \delta^{13}C_c = \alpha(\delta^{13}C_{b1} - \delta^{13}C_{b2}) + \delta^{13}C_{b2}$$

This Bayesian model needs thus two baselines. In order to use this model with your data, you need to use the following function:

```
model.string <- jagsTwoBaselines()
```

As usual, you can define the *prior* distribution for all of the variables included within the equations (2) and (3), the parameter `TP` and the constant λ. For example, if you want to define a *prior* normal distribution for the mean of ΔN, then you would have to do this:

```
model.string <- jagsTwoBaselines(muDeltaN = "dnorm(3.4, 1)")
```

Two baselines model, including TEFs for both N and C

Finally, the most complex model included in `tRophicPosition` is `jagsTwoBaselinesFull()`. This model implements the same equation as above (2), allowing the calculation of TP from two baselines, but the calculation of α is a little bit different, because it includes ΔC:

$$(4)\ \ \alpha = (\delta^{13}C_{b2} - (\delta^{13}C_c + \Delta C))/(\delta^{13}C_{b2} + \delta^{13}C_{b1})$$

In order to use this Bayesian model, you have to use it like this:

```
model.string <- jagsTwoBaselinesFull()
```

By default, as the simplest models, the *prior* distributions defined are uninformative, and you can improve the estimation of trophic position by including your *prior* knowledge. The random variables you can define are the same of the `jagsTwoBaselines()` model, plus $\mu_{\Delta C}$ (muDeltaC) and $\sigma_{\Delta C}$ (sigmaDeltaC).

Initializing the model and sampling the posterior estimates

Once you have loaded the isotope data and you have formatted it accordingly, you can run a suitable Bayesian model to generate large numbers of posterior estimated values of trophic position. Once this is done, it is necessary to summarize the posterior sample of trophic position and other variables in order to calculate some statistics. As such, we therefore have to initialize the Bayesian model, and then sample the posterior estimates of trophic position.

Initializing the Bayesian model

In order to initialize the Bayesian model, we need to feed the Bayesian model we have selected before with data. This is done with the function `TPmodel()`.

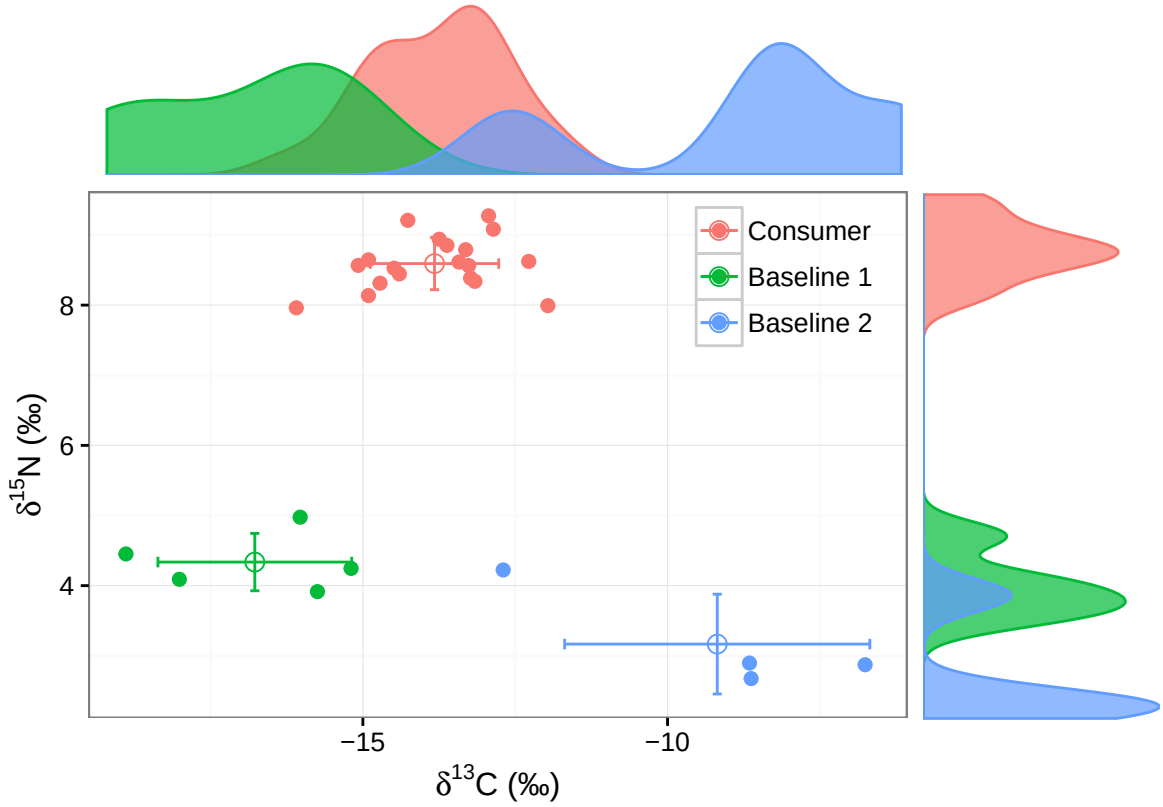
If you were practicing with data while reading this vignette, you can use your own data. In the following example we will use a dataset that is included within `tRophicPosition`, and we will see its structure:

```
# Here we load the data set included within tRophicPosition
data(Orestias)

# And here we see the structure of the data set
str(Orestias)
```

```
## List of 8
##  $ dCc   : num [1:19] -12.9 -12.9 -13.3 -13.7 -14.7 ...
##  $ dNc   : num [1:19] 9.08 9.27 8.79 8.94 8.31 ...
##  $ dCb1  : num [1:19] -18 -15.7 -18.9 -16 -15.2 ...
##  $ dNb1  : num [1:19] 4.09 3.91 4.45 4.98 4.25 ...
##  $ dCb2  : num [1:19] -6.76 -12.7 -8.63 -8.65 NA ...
##  $ dNb2  : num [1:19] 2.87 4.22 2.67 2.9 NA ...
##  $ deltaN: num [1:19] 2.64 2.6 2.61 2.69 2.1 ...
##  $ deltaC: num [1:19] 1.05 1.21 2.26 1.07 1.11 ...
```

As you see, it is a named list with 8 variables: δ^{13}C and δ^{15}N of the consumer that we want to estimate TP for (*Orestias chungarensis*), and of the two baselines (baseline 1 and baseline 2). Also included are the TEFs (ΔN and ΔC). If you look closer, there are some `NA` values we need to omit:

```
# Here we omit the NA values, and assign the output to the IsotopeData variable
Isotope.Orestias <- lapply(Orestias, na.omit)

# And then we can screen the dataset, after having omited the NA values
screenIsotopeData(Isotope.Orestias)
```



And now we are ready to initialize the Bayesian model. We will use the `jagsTwoBaselinesFull()` model, following this simple process:

```
# Here we define the Bayesian model
model.string <- jagsTwoBaselinesFull()
```

```
# And here we initialize the model
model.Orestias <- TPmodel(data = Isotope.Orestias,
                  model.string = model.string)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 76
##    Unobserved stochastic nodes: 16
##    Total graph size: 170
##
## Initializing model
```

You can optionally define the number of chains sampled, and the number of adaptive iterations in order to further customise the model. This is done by including the following arguments:

```
# By default, TPmodel() samples 2 chains with 10000 adaptive iterations
# Here we will sample 4 chains with 20000 adaptive iterations each
```

```
model.Orestias <- TPmodel(data = Isotope.Orestias,
                  model.string = model.string,
                  n.chains = 4,
                  n.adapt = 20000)
```

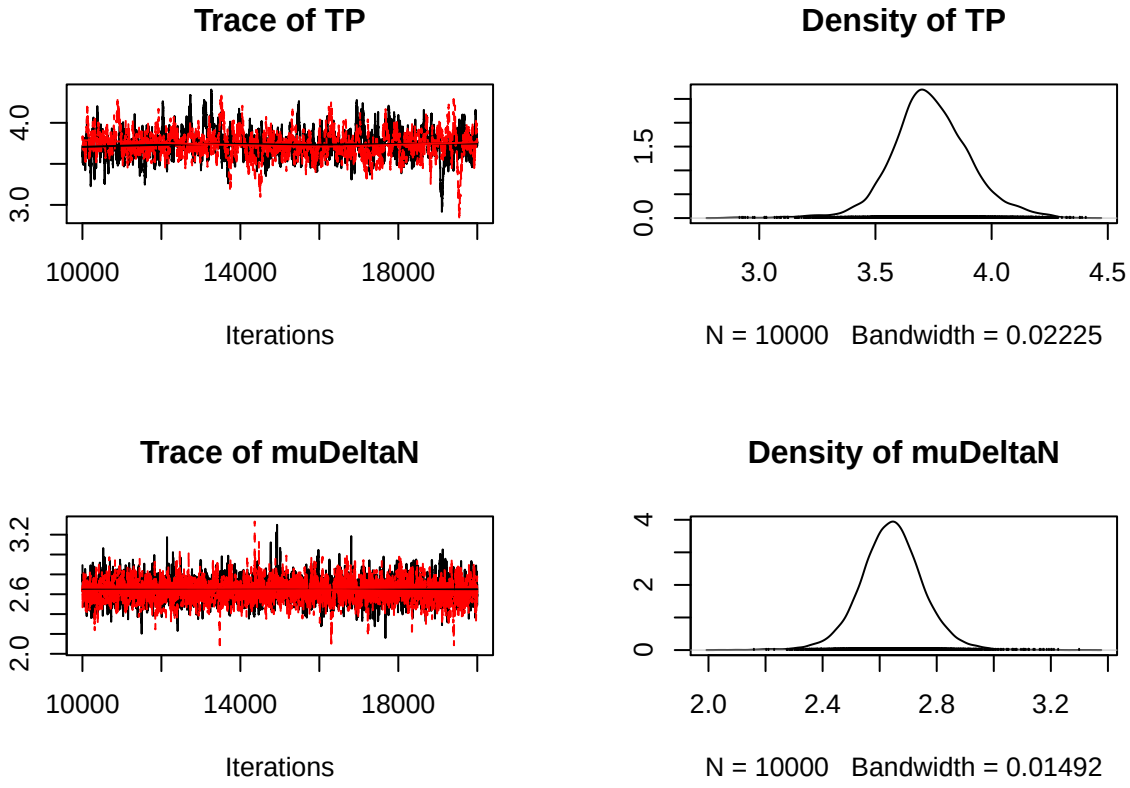
Sampling and plotting the posterior of trophic position

The task of sampling the posterior of trophic position estimates is simple, and uses the Bayesian model you previously initialised. If you do not explicitly define the variables you want to monitor, `posteriorTP()` will monitor by default `TP` and `muDeltaN`.

```
samples.Orestias <- posteriorTP(model.Orestias)
```

When you have a posterior sample of trophic position, you can plot it with:

```
plot(samples.Orestias)
```



In the graph above you have four subgraphs. You will see the trace and density of posterior estimates of `TP` and `muDeltaN`. As we set up 4 chains, you should see 4 colours in the trace of both variables, with each colour representing one chain sampled.

You can also plot the data with the function `trophicDensityPlot()`. This functions receives a data frame with two variables: `TP` and `Species`, where `TP` are the posterior samples of trophic position, and `Species` is a factor. We can plot the quantiles (95% of credibility interval, plus median and mean). In order to plot the data with this function, we first have to extract the data:

```r
# Here we save the posterior samples of trophic position (only first chain)
Orestias.TP <- as.data.frame(samples.Orestias[[1]][,"TP"])$var1

# And then we combine it with the posterior samples from the second chain
Orestias.TP <- c(Orestias.TP,as.data.frame(samples.Orestias[[2]][,"TP"])$var1)

# And we check the length of the variable we created
length(Orestias.TP)
```

```
## [1] 20000
```

Now, we create the data frame:

```r
# With this code we make the Species variable, including the species name
# "Orestias chungarensis"
Species <- c(rep("Orestias chungarensis", length(Orestias.TP)))

# and then we combine it with the posterior samples of Orestias trophic position
df <- data.frame(Orestias.TP, Species)
```

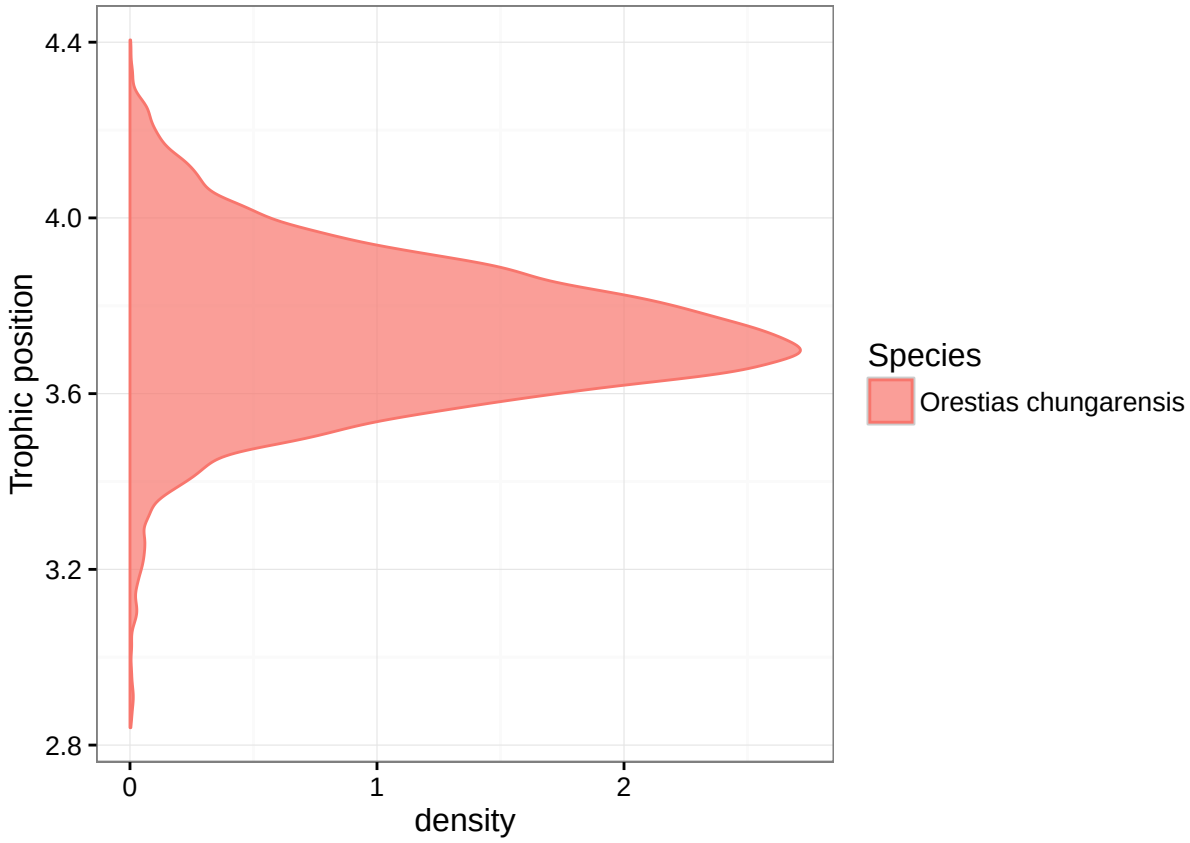
We need the data frame to include the variable names `TP` and `Species`, so we need to change the names accordingly:

```r
# Changing the variable names of the dataframe "df"
colnames(df) <- c("TP", "Species")

# We check that everything worked well
summary(df)
```

```
##        TP                          Species
##  Min.   :2.840   Orestias chungarensis:20000
##  1st Qu.:3.637
##  Median :3.731
##  Mean   :3.738
##  3rd Qu.:3.840
##  Max.   :4.405
```

And finally, plot it:

```
trophicDensityPlot(df)
```



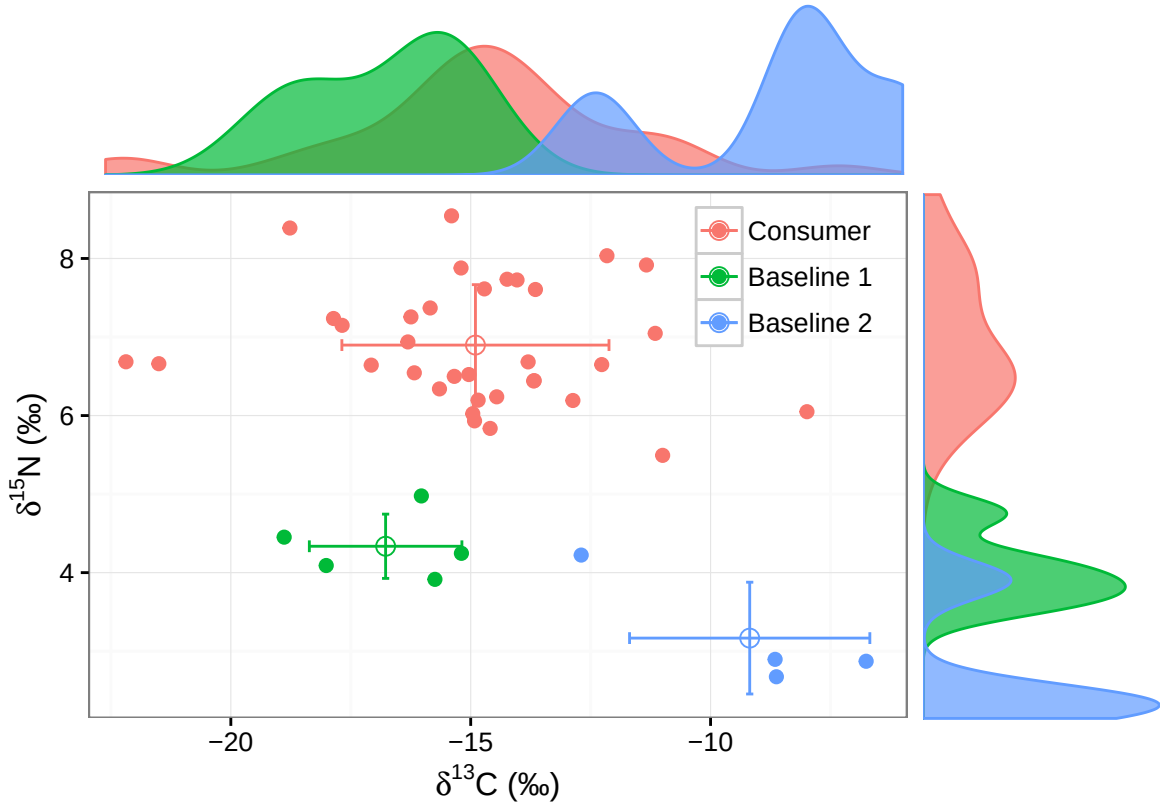
Resuming: trophic position in a nutshell

So, in a few words, in order to use `tRophicPosition` you have to:

```
# Load data into R
data(Trout)
```

The Trout dataset is included with `tRophicPosition`. These isotope values correspond to the juvenile invasive rainbow trout *Oncorhynchus mykiss* that lives sympatrically with the *Orestias chungarensis* in the River Chungará, in the Chilean Altiplano.

```
# Omit the NA values (if there are)
Isotope.Trout <- lapply(Trout, na.omit)

# Screen the isotope data
screenIsotopeData(Isotope.Trout)
```



As you see, the rainbow trout is much more variable than the *Orestias* in isotopic biplot space. We will use the `jagsTwoBaselinesFull()` model to calculate the rainbow trout trophic position.

```
# Here we choose the Bayesian model
model.string <- jagsTwoBaselinesFull()

# And then initialize the Bayesian model with data
model.Trout <- TPmodel(Isotope.Trout, model.string)
```

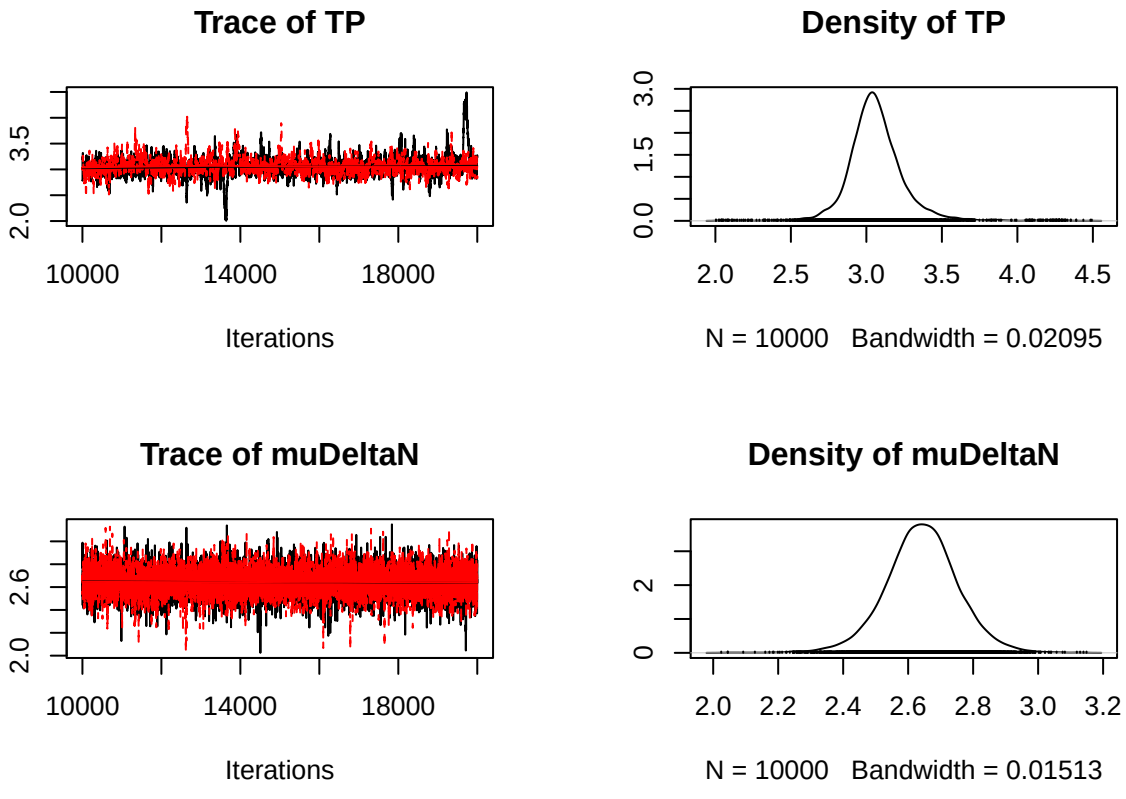
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 106
##    Unobserved stochastic nodes: 16
##    Total graph size: 215
##
## Initializing model
```

Now that we have initialized the model, we sample the posterior and plot the data.

```
# Here we sample the posterior trophic position
samples.Trout <- posteriorTP(model.Trout)

# Then we summarize the posterior data
summary(samples.Trout)
```

```
##
## Iterations = 10001:20000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean     SD Naive SE Time-series SE
## TP       3.066 0.1930 0.001365        0.01031
## muDeltaN 2.642 0.1131 0.000800        0.00147
##
## 2. Quantiles for each variable:
##
##            2.5%   25%   50%   75% 97.5%
## TP       2.733 2.961 3.051 3.153 3.482
## muDeltaN 2.413 2.573 2.643 2.712 2.865
```

```
# And plot it
plot(samples.Trout)
```



Finally, we extract the posterior trophic position samples, and save them into a variable:

```r
# Here we extract the first chain
Trout.TP <- as.data.frame(samples.Trout[[1]][,"TP"])$var1

# Here we extract the second chain and combine it with the first chain
Trout.TP <- c(Trout.TP,as.data.frame(samples.Trout[[2]][,"TP"])$var1)

# And then we check the length of the variable, i.e. how many posterior values we have
length(Trout.TP)
```

```
## [1] 20000
```

Comparing two trophic positions

Now we have posterior samples of two sympatric species living in the Chilean Altiplano: *Orestias chungarensis* and *Oncorhynchus mykiss*. If you want to compare the trophic position of both species you can use the function `compareTwoDistributions()`. This function receives 2 posterior distributions and a test, and returns the probability of ocurring that comparison, in a Bayesian way. Check `?compareTwoDistributions()` for other details.

We have the *Orestias* posterior trophic position:

```r
summary(Orestias.TP)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.840   3.637   3.731   3.738   3.840   4.405
```

And also we have the rainbow trout posterior trophic position:

```r
summary(Trout.TP)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.004   2.961   3.051   3.066   3.153   4.493
```

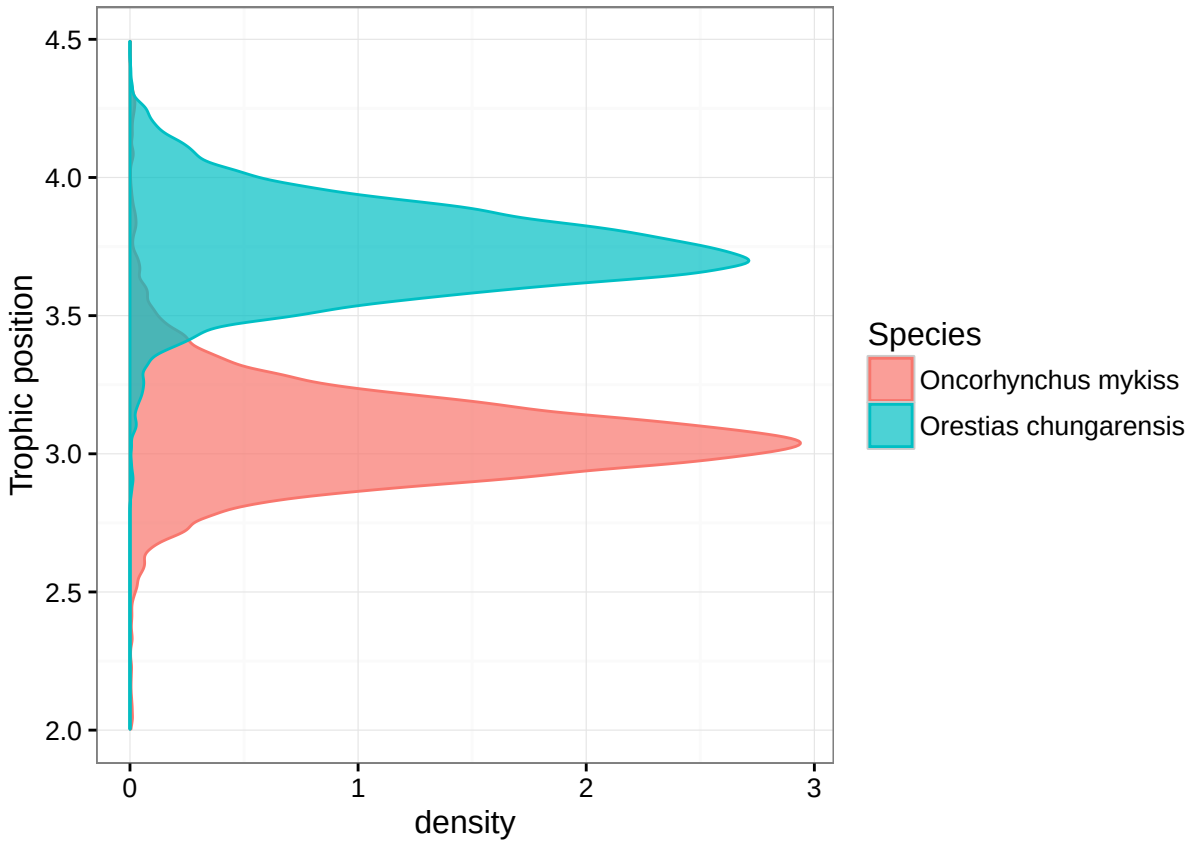
From the summary statistics, *Orestias* has a median estimated TP of 3.74 and rainbow trout of 3.05 (your results may be slightly different). We want to know whether there is any statistical support for differences in TP between *Orestias* and the juvenile rainbow trout. We can simply inquire wheter "the posterior distribution of trophic position of *Orestias* is higher than the posterior distribution of rainbow trout". With the `compareTwoDistributions` function, we ask the folowing:

```r
compareTwoDistributions(Orestias.TP, Trout.TP, test = ">")
```

```
## [1] 0.9869
```

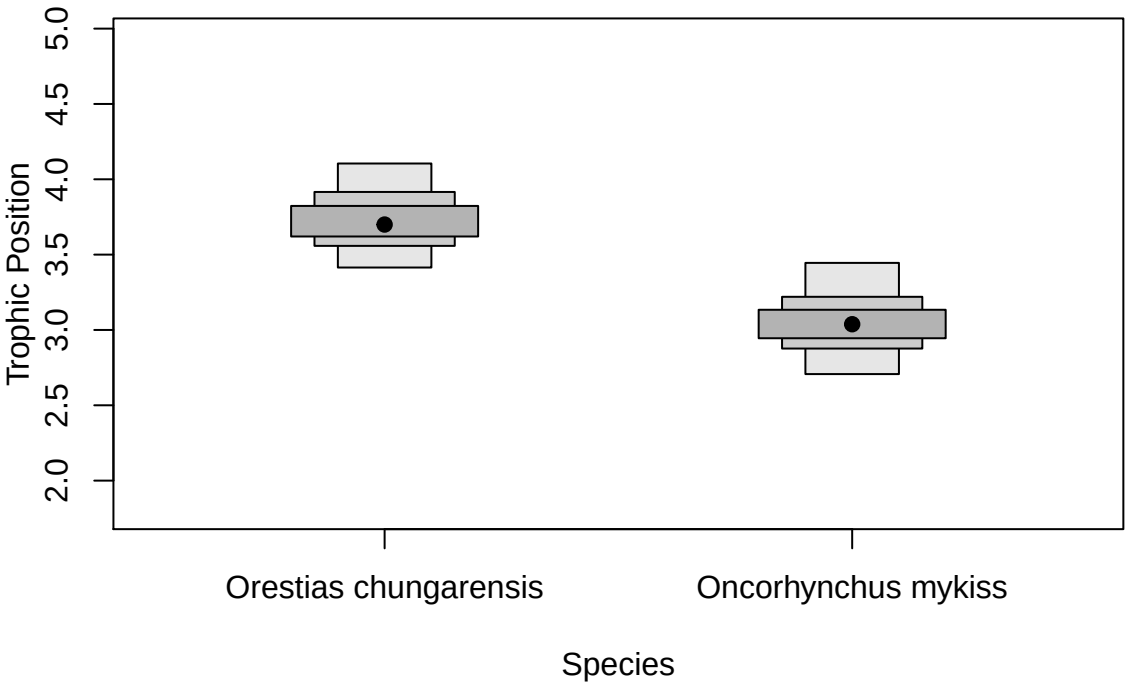
And according to the result, it is very, very likely (more than 99.5 %) that *Orestias* is predating on a higher trophic position than rainbow trout. Remember that your values will be slightly different. We can plot the results to visualise the differences:

```r
#Now we build a data frame combining both TP and creating a Species factor
TP <- c(Orestias.TP, Trout.TP)
Species <- c(rep("Orestias chungarensis", length(Orestias.TP)),
             rep("Oncorhynchus mykiss", length(Trout.TP)))

df <- data.frame(TP, Species)

# Here we use the function for plotting trophic position grouped by the Species factor
# and without quantiles (the default)
trophicDensityPlot(df)
```



Because we have posterior data in the same format to that used by Jackson et al.'s (2011) SIBER package, we can borrow the `siberDensityPlot()` function:

```r
# First we combine the two posterior samples into a data frame
df2 <- data.frame(Orestias.TP, Trout.TP)

# Then we change the name of each column
colnames(df2) <- c("Orestias chungarensis", "Oncorhynchus mykiss")

# And then we plot the data
SIBER::siberDensityPlot(df2, xlab = "Species",
                        ylab = "Trophic Position")
```


Trophic Position
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Orestias chungarensis
Oncorhynchus mykiss
Species