Vignette

Brody Erlandson

Introduction

In this vignette, we walk through how to use the `FunCZIDM` package and the associated shipy app. We use the case study data in the paper "A Bayesian Functional Concurrent Zero-Inflated Dirichlet Multinomial Regression Model" to provide examples of how you would use and gain inference from the model. This package contains the data from the case study, and the samples from the case study used in the paper. Along with the scripts used to run the model and simulations from the paper. More details on reproducing the results in the paper can be found in the `README.md` file in the packages root directory.

Data Set Up

We can load the data from the case study using `data` function.

```
library(FunCZIDM)

data(infantData)
```

This loads two objects: `infantCovariates` and `infantCounts`. The first step is to ensure the covariates and counts data are aligned. We can check the two data sets are aligned by checking the subject id's and times are aligned.

```
# Check count and covariate data are aligned
(all(infantCounts$Subject == infantCovariates$Subject) &&
 all(infantCounts$Day.of.life.sample.obtained
     == infantCovariates$Day.of.life.sample.obtained))
```

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

The `FunCZIDM` function requires the categorical covariates to be factors, and will use the reference level as the reference category for the variable. Below, we convert the categorical variables to factors, and ensure the reference level is the one we want.

```
# Convert the categorical variables to factors
infantCovariates$gender <- as.factor(infantCovariates$gender)
infantCovariates$mode.of.birth <- as.factor(infantCovariates$mode.of.birth)
infantCovariates$Room.category <- as.factor(infantCovariates$Room.category)
infantCovariates$milk <- as.factor(infantCovariates$milk)
infantCovariates$milk <- relevel(infantCovariates$milk, ref="< 10%")
infantCovariates$Period.of.study <- as.factor(infantCovariates$Period.of.study)
infantCovariates$Period.of.study <- relevel(infantCovariates$Period.of.study,
                                            ref="before")

# Check the types of the covariates
str(infantCovariates)
```

```
## 'data.frame':    922 obs. of  9 variables:
```

```
##  $ Subject                          : int  1 1 1 1 1 2 2 2 2 3 ...
##  $ Day.of.life.sample.obtained      : num  54.1 56 59.1 60 67.5 ...
##  $ Gestational.age.at.birth...weeks: num  26 26 26 26 26 26 26 26 26 25 ...
##  $ gender                           : Factor w/ 2 levels "female","male": 1 1 1 1 1 1 1 1 1 1 ...
##  $ mode.of.birth                    : Factor w/ 2 levels "caesarian section",..: 1 1 1 1 1 1 1 1 1 1
##  $ Period.of.study                  : Factor w/ 2 levels "before","after": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Room.category                    : Factor w/ 2 levels "Single ","open": 2 2 2 2 2 2 2 2 2 2 ...
##  $ milk                             : Factor w/ 3 levels "< 10%","10-50%",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Proportion.days.on.antibiotics   : num  0.314 0.304 0.288 0.283 0.252 ...
```

The continuous variables will be standardized automatically, and the output of `FunCZIDM` will contain the means and standard deviations used for standardization. Other functions in this package will use this to standardized automatically, so you will always input the original scale and location of the continuous variables.

Now we can remove the subject id's and time variables from the covariates and counts data. The `FunCZIDM` function will use the subject id's and time variables as the `ids` and `varyingCov` arguments, respectively. Lastly, the package requires the counts data to be a matrix.

```
# getting the index of the subject id's and the time-varying variable
idIdx <- which(colnames(infantCovariates) == "Subject")
timeIdx <- which(colnames(infantCovariates) == "Day.of.life.sample.obtained")
# extracting the subject ids and tv variable
ids <- infantCovariates[, idIdx]
time <- infantCovariates[, timeIdx]
infantCovariates <- infantCovariates[, -c(idIdx, timeIdx), drop = FALSE]

idIdx <- which(colnames(infantCounts) == "Subject")
timeIdx <- which(colnames(infantCounts) == "Day.of.life.sample.obtained")
infantCounts <- infantCounts[, -c(idIdx, timeIdx), drop = FALSE]
infantCounts <- as.matrix(infantCounts)
```

Now we can double check the data is set up correctly.

```
str(infantCovariates)
```

```
## 'data.frame':    922 obs. of  7 variables:
##  $ Gestational.age.at.birth...weeks: num  26 26 26 26 26 26 26 26 26 25 ...
##  $ gender                           : Factor w/ 2 levels "female","male": 1 1 1 1 1 1 1 1 1 1 ...
##  $ mode.of.birth                    : Factor w/ 2 levels "caesarian section",..: 1 1 1 1 1 1 1 1 1 1
##  $ Period.of.study                  : Factor w/ 2 levels "before","after": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Room.category                    : Factor w/ 2 levels "Single ","open": 2 2 2 2 2 2 2 2 2 2 ...
##  $ milk                             : Factor w/ 3 levels "< 10%","10-50%",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Proportion.days.on.antibiotics   : num  0.314 0.304 0.288 0.283 0.252 ...
```

```
if (is.matrix(infantCounts)) {
  print("Counts data is a matrix.")
}
```

```
## [1] "Counts data is a matrix."
```

```
colnames(infantCounts)
```

```
##  [1] "Actinobacteria"        "Alphaproteobacteria"   "Bacilli"
##  [4] "Bacteroidia"           "Betaproteobacteria"    "Clostridia"
##  [7] "Cyanobacteria"         "Epsilonproteobacteria" "Erysipelotrichi"
## [10] "Flavobacteria"         "Fusobacteria"          "Gammaproteobacteria"
## [13] "Sphingobacteria"       "Deltaproteobacteria"   "OD1_no_class"
```

```
## [16] "unclassified"
```

Running the Model

Once the data are set up, one can sample from the FunC-ZIDM model using the `FunCZIDM` function. The first four inputs are what we prepared in the pervious section. Next we specify the number of total iterations (`iter`), the burn-in iterations (`burnIn`), and how much to thin by (`thin`). To allow for efficient knitting, we only do 8500 iterations with a 7500 burn-in while thinning to every 10th iteration. If you need to assess convergence, you can return the burn-in samples by setting `returnBurnIn=TRUE`.

By default, the function will return the β^*_{jpd} samples, r_{jp} means, the acceptance rates for the β^*_{jpd} and r_{jp} after burn-in, and the average level of zero-inflation. One can specify other parameters to be returned using the `toReturn` argument. Commonly, one may want the relative abundance samples for each individual. This can be specified by adding `"RA"` to the `toReturn` argument.

The model has several priors that can be adjusted using the `priors` argument. Additionally, the proposal standard deviations for the β^*_{jpd} and r_{jp} can be initialized using the `proposalVars` argument. By default, the prioirs and proposal standard deviations are set to the values used in the case study and simulations of the paper.

For the splines, one can specify the degrees of freedom (`df`), the degree of the splines (`degree`), and the basis function to use (`basisFunc`). The default values (4, 3, and `splines::bs`) are the ones used in the case study and simulations of the paper.

If there are covariates that is assumed to not have a time-varying effect, one can specify the indices of these covariates in the `covWithoutVC` argument. In the case study, we assume the period of study to be time-invariant.

Lastly, one can save the output to a file using the `saveToFile` argument. If `TRUE`, the output will be saved to a file with the name specified by `fileName` or "output.rds" by default. This is useful when running multiple chains to be able to combine after all chains have finished sampling. One can also save a csv file indicating the proportion of mean $\beta_{jp}(t)$ functions that have the null function in the 95% credible interval using the `saveNullInBetaCI` argument. This saves to a file named "nullInBetaCI.csv" by default. This can be useful to see which covariates may have meaningful effects, specifally when there are a large number of covariates and/or categories. It is important to note that if the output contains the burn-in, then the mean $\beta_{jp}(t)$ will be estimated using the burn-in samples as well.

There are some other inputs that can be specified, however we recommend keeping them at their defaults unless a strong reason to change. A full description of the arguments and options can be found in the reference manual.

```
output <- FunCZIDM(counts=infantCounts,
                   covariates=infantCovariates,
                   ids=ids,
                   varyingCov=time,
                   iter=85,
                   burnIn=75,
                   thin=10,
                   returnBurnIn=FALSE,
                   toReturn=c("RA"),
                   priors=list("first beta sd"=1, "a"=3, "b"=3, "alpha"=.01,
                               "beta"=10, "kappaShape"=100, "kappaRate"=900),
                   covWithoutVC=c(which(grepl("Period", colnames(infantCovariates)))),
                   saveToFile=FALSE,
                   saveNullInBetaCI=FALSE)
```

```
## [==>                                              ] 5% - 5/85 iterations[=====>
```

```
## [1] "Total time taken: 1.167 seconds"
```

If multiple chains are run, one can combine the output files with the `combineOutputs` function. An example is provided below.

```
outputFiles <- c("output1.rds", "output2.rds", "output3.rds", "output4.rds")
combinedOutput <- combineOutputs(outputFiles, saveToFile=FALSE)
```

If the output files contain the burn-in, then the combined output will too. One can resave the `rds` without the burn-in or take the last `(iter - burnIn)/thin` times the number of chains samples from the combined output. An example of resaving the output without the burn-in is provided below.

```
files <- c("output1.rds", "output2.rds", "output3.rds", "output4.rds")
for (file in files) {
  output <- readRDS(file)
  output$beta <- output$beta[,, 751:850] # only keeping samples after burn-in
  saveRDS(output, file=file, compress="xz")
}
```

The package includes the combined output from the case study in the paper under `inst/extdata`, so it is not necessary to run the sampling. We will use this output for the inference section. We load the combined output below.

```
combinedOutput <- readRDS(system.file("extdata", "caseStudySamples.rds",
                          package = "FunCZIDM"))
names(combinedOutput)
```

```
##  [1] "interiorKnots"     "boundaryKnots"     "varyingCov"
##  [4] "colMapping"        "XvartoXColMapping" "basisFunc"
##  [7] "df"                "catNames"          "centerScaleList"
## [10] "beta"              "betaAcceptProp"    "rAcceptProp"
## [13] "rMeans"            "etaMeanPropZeros"
```

The output object contains the samples and other information used in the other functions in the package. Most of these variables are not needed for direct use by the user, but are used in the other functions. One can read the description of each variable in the reference manual.

Inference

We demonstrate how to use the functions to obtain inference from the samples using the preloaded samples. The package has the ability to calculate the relative abundance ($\text{RA}_j(t)$) and α-diversity functions ($\alpha(t)$), the multiplicative change in relative abundance ($\Delta_v \text{RA}_{jp}(t)$) and α-diversity functions ($\Delta_v \alpha_p(t)$), and the functional coefficients ($\beta_{jp}(t)$).

We can begin by looking at the proportion of the credible interval of $\beta_{jp}(t)$ that contains the null function. By looking at the lowest proportions, we can narrow down which covariates and categories to look into. allowing for easier exploration of the α-diversity and relative abundance.

```
propNullInCI <- getPropNullInCI(combinedOutput,
                                varCovRange=c(0, 50)) # look at first 50 days
```

```
## Warning in basisFunc(testPoints, df = df, intercept = FALSE, knots =
## interiorKnots, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases
```

```
covToFocusOn <- c("gestational", "milk")
propNullInCI <- propNullInCI[grep(paste(covToFocusOn, collapse="|"),
```

```
                                    propNullInCI[,2], ignore.case=T), ]
head(propNullInCI[order(propNullInCI[,3]), ])
```

```
##      category              covariate
## [1,] "Bacilli"             "milk> 50%"
## [2,] "Clostridia"          "milk10-50%"
## [3,] "Gammaproteobacteria" "Gestational.age.at.birth...weeks"
## [4,] "Actinobacteria"      "Gestational.age.at.birth...weeks"
## [5,] "Alphaproteobacteria" "Gestational.age.at.birth...weeks"
## [6,] "Bacilli"             "Gestational.age.at.birth...weeks"
##      Proportion 0 in 95 CI
## [1,] "0.284"
## [2,] "0.46"
## [3,] "0.756"
## [4,] "1"
## [5,] "1"
## [6,] "1"
```

As we did in the paper, we will focus on the gestational age at birth and the breast milk proportion in the diet covariates. We can see the top 3 category and covariate combinations are the only ones that do not contain the null function for the entire time period. This doesn't mean the other covariates multiplicative changes credible intervals do not contain the null function, but these variables are a good starting point for exploration unless there is interest particular covariates.

Relative Abundance

The relative abundance and multiplicative change in relative abundance can be calculated using the `calcRA` and `calcDeltaRA` functions, respectively. First, we need to note the index of each covariate in the model:

```
combinedOutput$colMapping
```

```
## [[1]]
## [1] "(Intercept)"
##
## [[2]]
## [1] "Gestational.age.at.birth...weeks"
##
## [[3]]
## [1] "gendermale"
##
## [[4]]
## [1] "mode.of.birthvaginal delivery"
##
## [[5]]
## [1] "Period.of.studyafter"
##
## [[6]]
## [1] "Room.categorySingle "
##
## [[7]]
## [1] "milk> 50%"
##
## [[8]]
## [1] "milk10-50%"
##
```

```
## [[9]]
## [1] "Proportion.days.on.antibiotics"
```

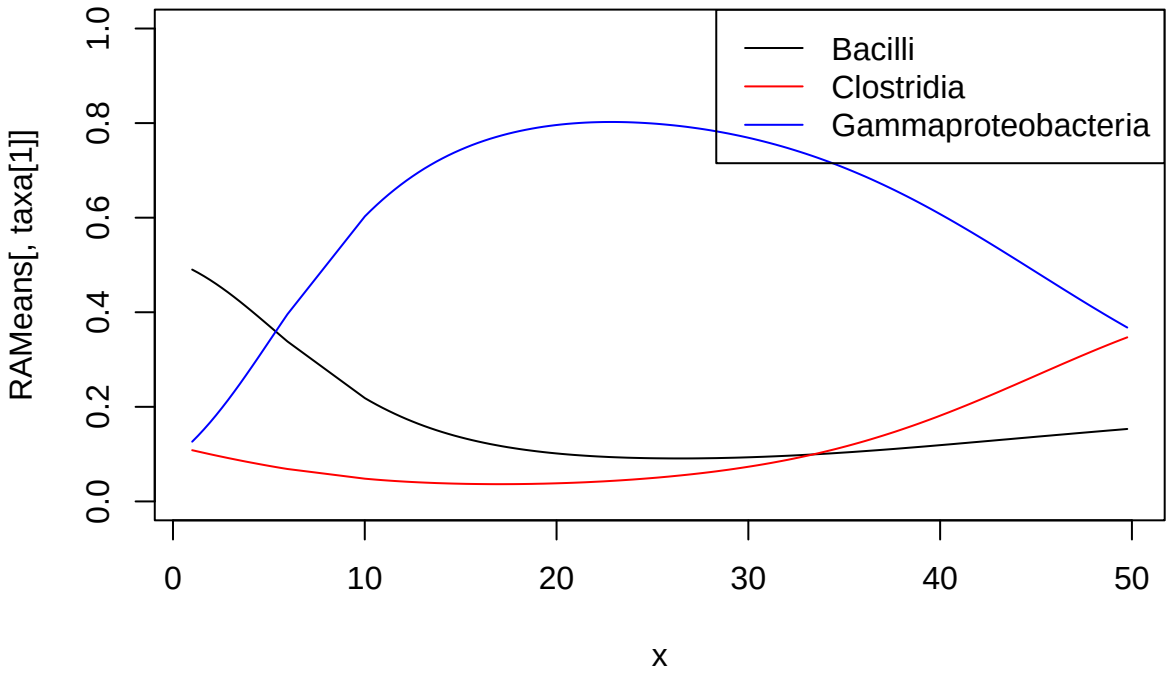
These will be used for filling in the covariate profile and indicating which covariate you want the change in. Similarly, you can view the category names in the order the model uses them with `combinedOutput$catNames`.

```
# always put 1 for the intercept
covProfile <- c(1, 27, 0, 0, 0, 0, 0, 0, .3)
# calculate the relative abundance
RA <- calcRA(combinedOutput, covProfile=NULL)
dim(RA)
```

```
## [1]  250   16 4000
```

After assigning the desired covariate profile, we get all the categories relative abundance samples. Where we have 250 timepoints, 16 categories, and 4000 samples. With these samples, we can visualize the mean relative abundance over time for tha taxa. Below, we provide an example of plotting the mean relative abundance over time for Bacilli, Clostridia, and Gammaproteobacteria.

```
taxa <- c("Bacilli", "Clostridia", "Gammaproteobacteria")
times <- rownames(RA) <= 50
RAMeans <- apply(RA[times, taxa, ], c(1, 2), mean)
x <- rownames(RAMeans)
plot(x, RAMeans[, taxa[1]], ylim=c(0, 1), type="l")
lines(x, RAMeans[, taxa[2]], col="red")
lines(x, RAMeans[, taxa[3]], col="blue")
legend("topright", taxa, col=c("black", "red", "blue"), lty=1)
```

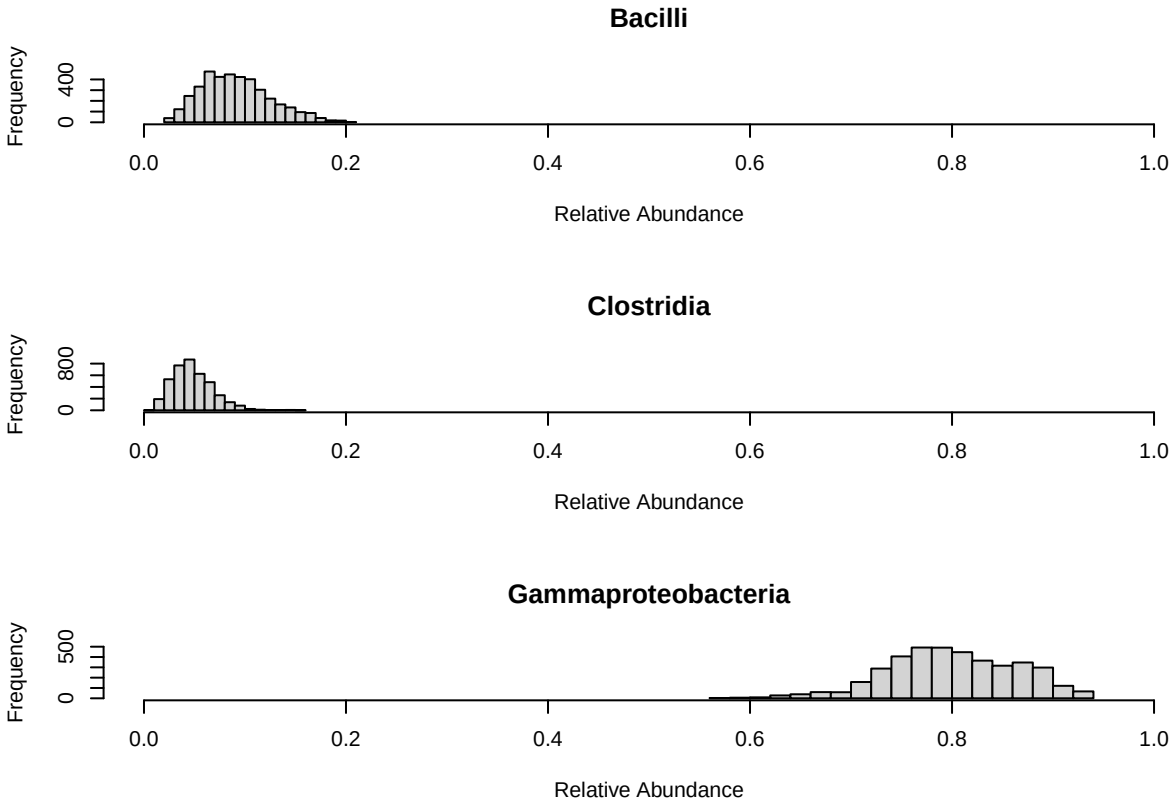


We can also view the distribution of the relative abundance at a particular time.

```
print(paste("Time point", rownames(RA)[77]))
```

```
## [1] "Time point 24.6027309236948"
```

```
par(mfrow=c(3, 1))
hist(RA[77, taxa[1], ], main=taxa[1], xlab="Relative Abundance",
     xlim=c(0, 1), breaks=20)
hist(RA[77, taxa[2], ], main=taxa[2], xlab="Relative Abundance",
     xlim=c(0, 1), breaks=20)
hist(RA[77, taxa[3], ], main=taxa[3], xlab="Relative Abundance",
     xlim=c(0, 1), breaks=20)
```




```
library(ggtern)
```

```
## Loading required package: ggplot2
```

```
## Registered S3 methods overwritten by 'ggtern':
##   method           from
##   grid.draw.ggplot ggplot2
##   plot.ggplot      ggplot2
##   print.ggplot     ggplot2
```

```
## --
## Remember to cite, run citation(package = 'ggtern') for further info.
## --
```

```
##
## Attaching package: 'ggtern'
```

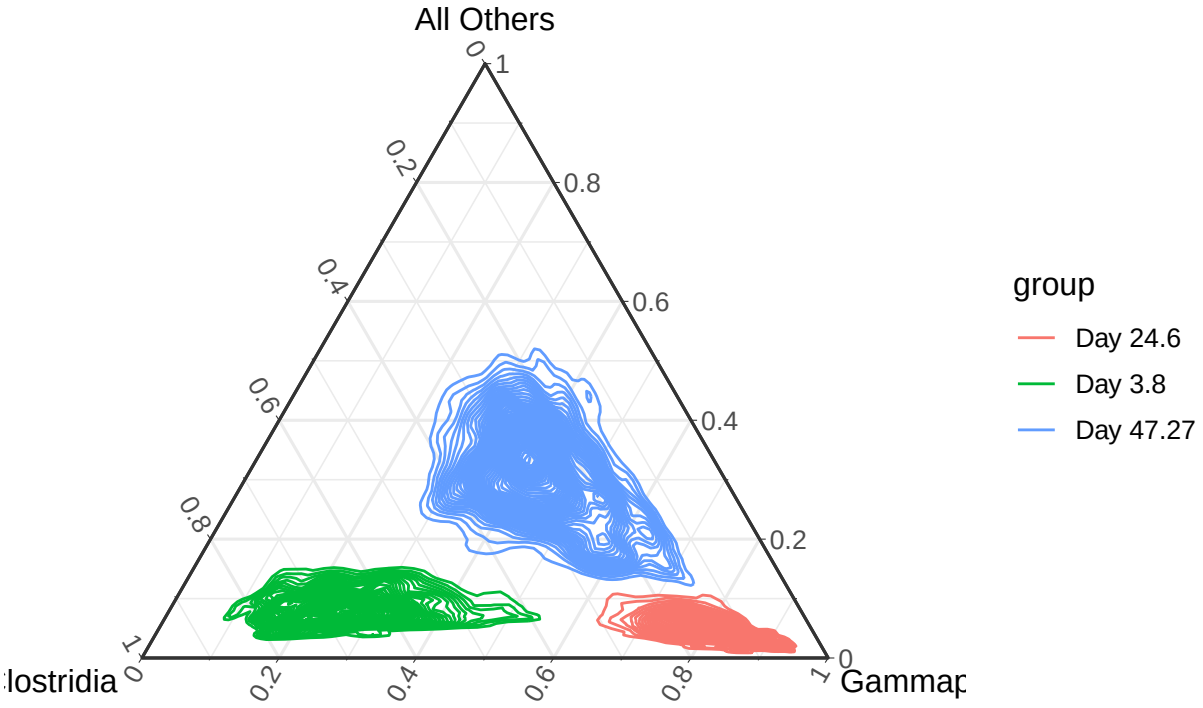
```
## The following objects are masked from 'package:ggplot2':
##
##     aes, annotate, ggplot, ggplotGrob, ggplot_build, ggplot_gtable,
##     ggsave, layer_data, theme_bw, theme_classic, theme_dark,
##     theme_gray, theme_light, theme_linedraw, theme_minimal, theme_void
```

```
S1 <- as.data.frame(t(RA[10,,]))
S1["oneMinusSumClostAndGamma"] <- 1 - rowSums(S1[c("Clostridia", "Gammaproteobacteria")])
S2 <- as.data.frame(t(RA[77,,]))
S2["oneMinusSumClostAndGamma"] <- 1 - rowSums(S2[c("Clostridia", "Gammaproteobacteria")])
S3 <- as.data.frame(t(RA[150,,]))
S3["oneMinusSumClostAndGamma"] <- 1 - rowSums(S3[c("Clostridia", "Gammaproteobacteria")])
S1$group <- paste("Day", round(as.numeric(rownames(RA)[10]), 2))
S2$group <- paste("Day", round(as.numeric(rownames(RA)[77]), 2))
S3$group <- paste("Day", round(as.numeric(rownames(RA)[150]), 2))
S_all <- rbind(S1, S2, S3)

ggtern(S_all, aes(oneMinusSumClostAndGamma, Clostridia, Gammaproteobacteria, color = group)) +
  stat_density_tern(
    aes(fill = after_stat(level)),
    geom = "path", bins = 50
  ) +
  scale_fill_gradient(low = "white", high = "darkblue") +
  scale_T_continuous(labels = seq(0, 1, 0.2), limits = c(0,1)) +
  scale_L_continuous(labels = seq(0, 1, 0.2), limits = c(0,1)) +
  scale_R_continuous(labels = seq(0, 1, 0.2), limits = c(0,1)) +
  labs(T = "All Others", L = "Clostridia", R = "Gammaproteobacteria") +
  theme_bw()
```

```
## Warning in stat_density_tern(aes(fill = after_stat(level)), geom = "path", :
## Ignoring unknown aesthetics: fill
```

```
## Warning: stat_density_tern: You have not specified a below-detection-limit (bdl) value (Ref. 'bdl' ar
## You can either:
## 1. Ignore this warning,
## 2. Set the bdl value appropriately so that fringe values are omitted from the ILR calculation, or
## 3. Accept the high density values if they exist, and manually set the 'breaks' argument
##    so that the countours at lower densities are represented appropriately.
```



Now, we can look into the multiplicative change using the same covariate profile.

```
# calculate the change in relative abundance
change <- c(2, 1, 1) # two week increase in gestational age and a 1 for indicating
                     # the change from <10% to >50% breast milk in the diet
forCovs <- c(2, 7, 8) # GA is 2nd idx and >50% BM in diet is 7th idx
forCats <- match(c("Bacilli", "Clostridia", "Gammaproteobacteria"),
                 combinedOutput$catNames) # gets the idx of these taxa
deltaRA <- calcDeltaRA(combinedOutput, change, covProfile = covProfile,
                       forCovs = forCovs, forCats = forCats)
names(deltaRA)
```

```
## [1] "Gestational.age.at.birth...weeks" "milk> 50%"
## [3] "milk10-50%"
```

```
colnames(deltaRA[[1]]) # GA is 1
```

```
## [1] "Bacilli"             "Clostridia"          "Gammaproteobacteria"
```

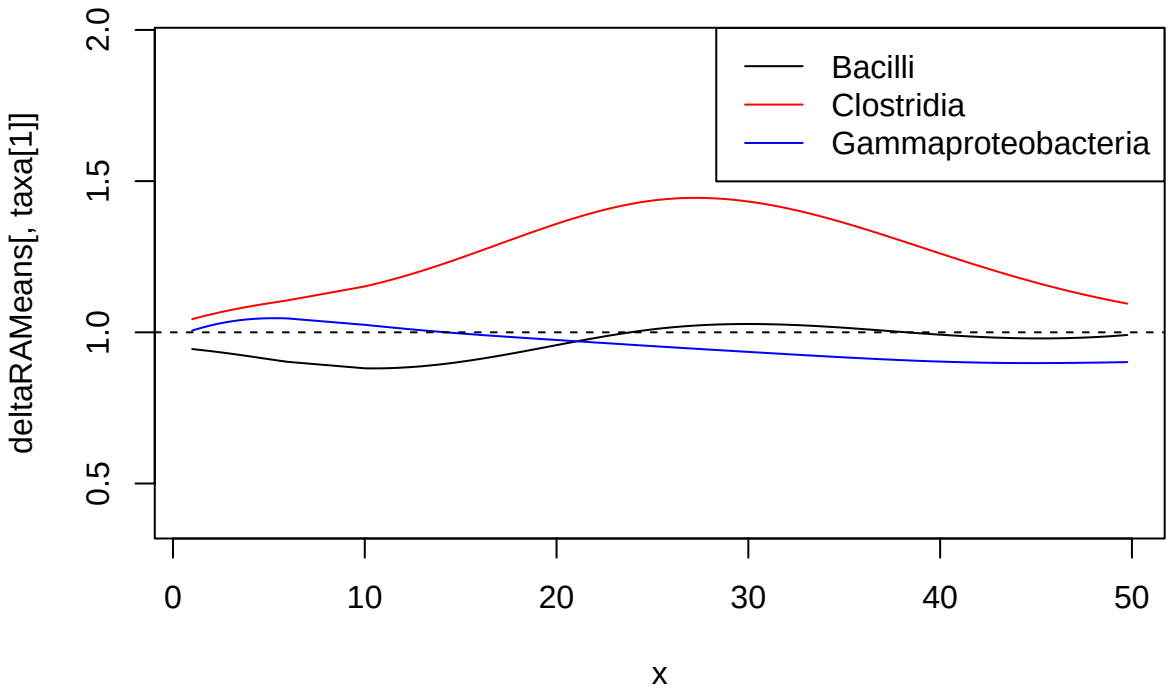
```
dim(deltaRA[[1]])
```

```
## [1]  250    3 4000
```

We can see the two covariates specified in the List, and for each covariate we have an array with the categories as the columns. The array has 250 timepoints, 3 categories, and 4000 samples. We can visualize the mean multiplicative change over time for the taxa we looked at before for the gestational age covariate below.

```
taxa <- c("Bacilli", "Clostridia", "Gammaproteobacteria")
times <- rownames(deltaRA[[1]]) <= 50
deltaRAMeans <- apply(deltaRA[[1]][times,,], c(1, 2), mean)
x <- rownames(deltaRAMeans)
ylim <- c(min(deltaRAMeans) - 0.5, max(deltaRAMeans) + 0.5)

plot(x, deltaRAMeans[, taxa[1]], ylim=ylim, type="l")
lines(x, deltaRAMeans[, taxa[2]], col="red")
lines(x, deltaRAMeans[, taxa[3]], col="blue")
abline(h=1, lty=2)
legend("topright", taxa, col=c("black", "red", "blue"), lty=1)
```

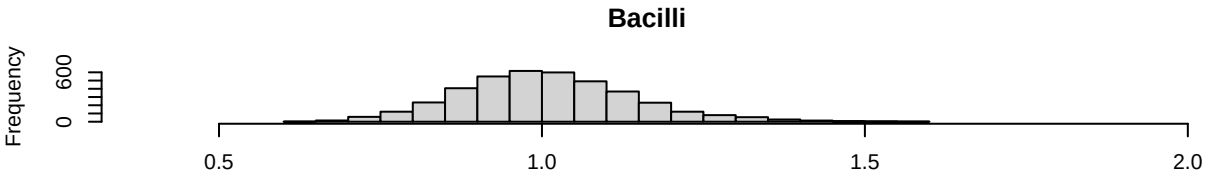


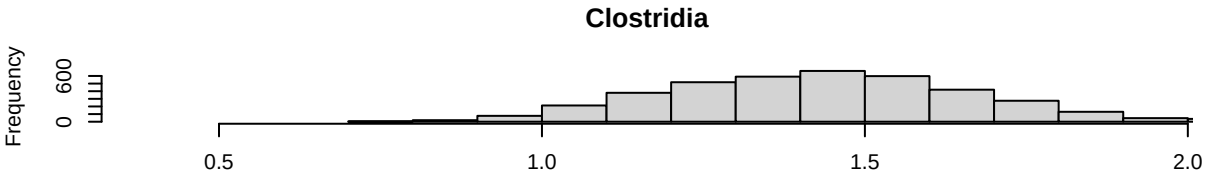
Additionally, we can find the distribution of the multiplicative change at a particular time.

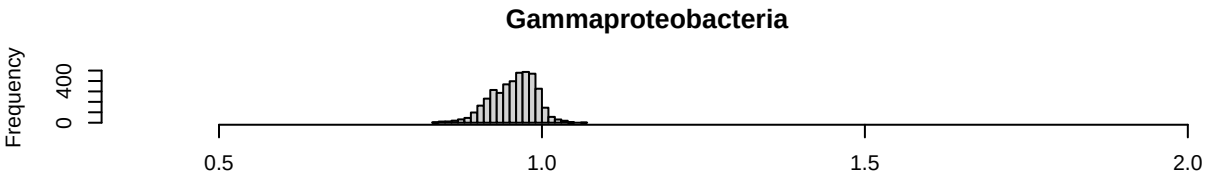
```
print(paste("Time point", rownames(deltaRA[[1]])[77]))
```

```
## [1] "Time point 24.6027309236948"
```

```
par(mfrow=c(3, 1))
hist(deltaRA[[1]][77, taxa[1], ], main=taxa[1], xlab="Multiplicative Change",
     xlim=ylim, breaks=20)
hist(deltaRA[[1]][77, taxa[2], ], main=taxa[2], xlab="Multiplicative Change",
     xlim=ylim, breaks=20)
hist(deltaRA[[1]][77, taxa[3], ], main=taxa[3], xlab="Multiplicative Change",
     xlim=ylim, breaks=20)
```







One may want to see what proportion of the multiplicative change credible interval contains the null effect (1). This can be done using the samples to estimate the credible interval.

```
# calculate the change in relative abundance for all cats
change <- c(2) # two week increase in gestational age
forCovs <- c(2) # GA is 2nd idx

deltaRA <- calcDeltaRA(combinedOutput, change, covProfile=covProfile,
                       forCovs=forCovs)

lowerCI <- apply(deltaRA[[1]][times,,], c(1, 2), quantile, probs=c(.025))
upperCI <- apply(deltaRA[[1]][times,,], c(1, 2), quantile, probs=c(.975))

oneInCI <- (lowerCI <= 1)*(upperCI >= 1)
colMeans(oneInCI)
```

```
##        Actinobacteria   Alphaproteobacteria              Bacilli
##             1.0000000             1.0000000           1.0000000
##           Bacteroidia    Betaproteobacteria           Clostridia
##             0.7945205             1.0000000           0.7465753
##         Cyanobacteria Epsilonproteobacteria       Erysipelotrichi
##             0.8287671             0.8287671           0.8013699
##         Flavobacteria          Fusobacteria   Gammaproteobacteria
##             1.0000000             0.7876712           1.0000000
##       Sphingobacteria   Deltaproteobacteria          OD1_no_class
##             0.8150685             0.8698630           0.7808219
##          unclassified
##             0.8013699
```

We can see that many categories do not contain the null effect over the whole time period, whereas the $\beta_{jp}(t)$

credible intervals do contain the null effect over the whole time period (with exception of Gammaproteobacteria). Additionally, this is only for a 2 week increase in gestational age from 27 weeks of gestational age at birth. With the associated shiny app, one can easily plot the mean multiplicative change over a range of changes.

```
deltaRAMeans <- apply(deltaRA[[1]][times,,], c(1, 2), mean)

oneInCI <- (lowerCI <= 1)*(upperCI >= 1)
colMeans(oneInCI)
```

```
##        Actinobacteria   Alphaproteobacteria               Bacilli
##             1.0000000             1.0000000             1.0000000
##           Bacteroidia    Betaproteobacteria            Clostridia
##             0.7945205             1.0000000             0.7465753
##         Cyanobacteria Epsilonproteobacteria       Erysipelotrichi
##             0.8287671             0.8287671             0.8013699
##         Flavobacteria          Fusobacteria   Gammaproteobacteria
##             1.0000000             0.7876712             1.0000000
##       Sphingobacteria   Deltaproteobacteria          OD1_no_class
##             0.8150685             0.8698630             0.7808219
##          unclassified
##             0.8013699
```

```
getMinMaxDeltaAlphaDiv(combinedOutput, 27, 2, 2, infantCovariates, time,
                       seq(0, 80, 40), l=.75)
```

```
## [1] 922   3
## [1] 695
## [1] 695   3
## [1] 492
## [1] 645
## NULL
## [1] 7
## [1] 227
## [1] 227   3
## [1] 175
## [1] 715
## NULL
## [1] 7
```

```
## $maxCov
##      [,1] [,2]   [,3]              [,4]   [,5]    [,6]   [,7]
## [1,] NA   female caesarian section before Single  10-50% 0
## [2,] NA   female caesarian section before open    < 10%  0.3035714
##
## $minCov
##      [,1] [,2] [,3]              [,4]   [,5] [,6]  [,7]
## [1,] NA   male caesarian section before open < 10% 0.6195787
## [2,] NA   male caesarian section after  open < 10% 0.2217841
```

α-diversity

This works similarly to the relative abundance functions. Using the same covariate profile, we can find the diversity over time:

```
l <- 0.75 # Hill's diversity parameter
alphaDiv <- calcAlphaDiv(combinedOutput, l=l, covProfile=covProfile)
```

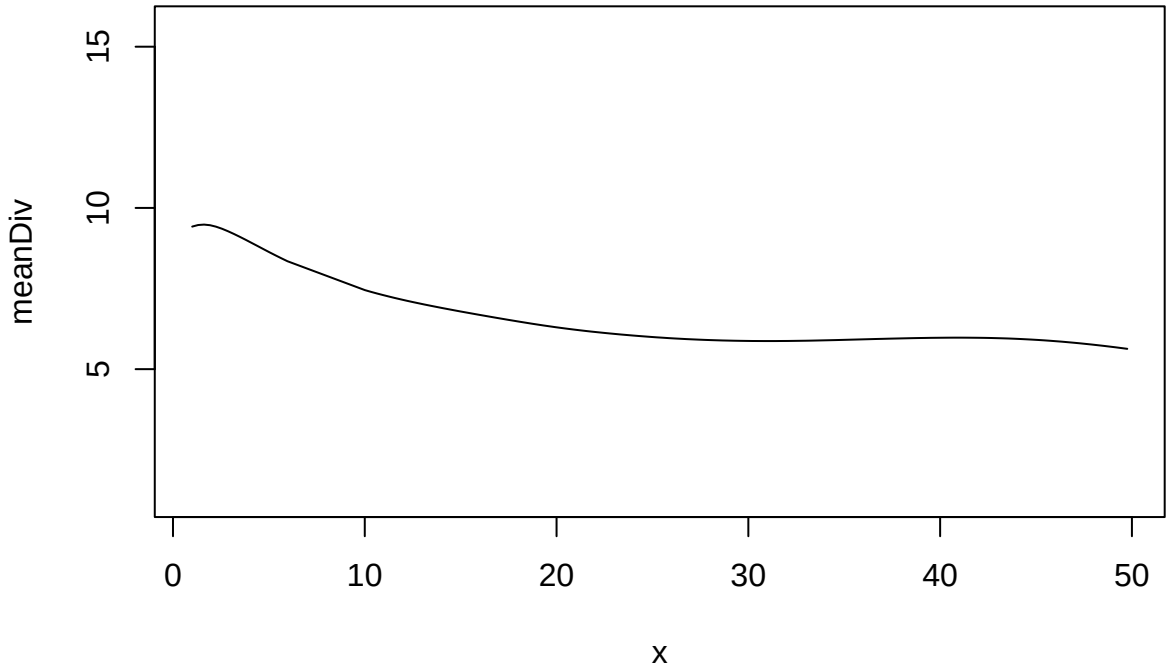
```
dim(alphaDiv)
```

```
## [1]  250 4000
```

The diversity uses all the categories, so we just have 250 timepoints and 4000 samples. For the multiplicative change in α-diversity, we only need to specify the covariates of interest and the amount of change. Again, one can plot the mean change over time.

```
times <- rownames(alphaDiv) <= 50
meanDiv <- rowMeans(alphaDiv[times,])
ylim <- c(min(alphaDiv), max(alphaDiv))
x <- rownames(alphaDiv[times,])

plot(x, meanDiv, ylim=ylim, type="l")
```



We can calculate the multiplicative change from this α-diversity function for a change in a covariate. We can calculate the change for all covariates, and find each covariates proportion of credible intervals that contain the null effect (1).

```
# calculate the change in relative abundance
change <- c(2, # two week increase in gestational age
            1, # indicating a change from female to male
            1, # indicating a change from c-section to vaginal delivery
            1, # indicating a change from before to after study change
            1, # indicating a change from multipatient room to single
            1, # indicating a change from <10% to >50% breast milk
            1, # indicating a change from <10% to 10-50% breast milk
           .3)# .3 increase in proportion of days on antibiotics
forCovs <- 2:9 # all covariates (not intercept)

deltaAlphaDiv <- calcDeltaAlphaDiv(combinedOutput, change, forCovs, l=1,
                                   covProfile=covProfile)
names(deltaAlphaDiv)
```

```
## [1] "Gestational.age.at.birth...weeks" "gendermale"
## [3] "mode.of.birthvaginal delivery"    "Period.of.studyafter"
## [5] "Room.categorySingle "             "milk> 50%"
## [7] "milk10-50%"                       "Proportion.days.on.antibiotics"
```

```
dim(deltaAlphaDiv[[1]]) # 1 is GA
```

```
## [1]  250 4000
```

```
times <- rownames(deltaAlphaDiv[[1]]) <= 50

for (cov in 1:8) {
  CI <- apply(deltaAlphaDiv[[cov]][times,], 1, quantile, probs=c(.025, .975))
  oneInCI <- (CI[1,] <= 1)*(CI[2,] >= 1)
  print(paste0(combinedOutput$colMapping[[cov+1]], ": ", mean(oneInCI)))
}
```

```
## [1] "Gestational.age.at.birth...weeks: 0.773972602739726"
## [1] "gendermale: 0.657534246575342"
## [1] "mode.of.birthvaginal delivery: 1"
## [1] "Period.of.studyafter: 0.917808219178082"
## [1] "Room.categorySingle : 0.410958904109589"
## [1] "milk> 50%: 1"
## [1] "milk10-50%: 1"
## [1] "Proportion.days.on.antibiotics: 0.554794520547945"
```

Again, all categories are used in calculating the diversity, so we just have 250 timepoints and 4000 samples. We find that the mode of birth and breast milk proportion are the only covariates that contain the nul effect in the credible interval over all the first 50 days.

$\beta_{cp}(t)$

At times, the covariate itself is of interest. These are particularly useful for seeing which covariate is causing the estimated changes.

```
betas <- getBetaFunctions(combinedOutput)
dim(betas)
```

```
## [1]  250   16 4000
```

This gives us the intercept for each category, and with `cov` you can specify which covariate you want. For example,

```
betas <- getBetaFunctions(combinedOutput, cov=2)
dim(betas)
```

```
## [1]  250   16 4000
```

provides the gestational age $\beta_{jp}(t)$ samples for each category. We can rank the average mean $\beta_{jp}(t)$ over time, and rank the maximum mean $|\beta_{jp}(t)|$.

```
times <- rownames(betas) <= 50
meanBetas <- apply(betas[times,,], c(1,2), mean)
# average mean B_jp(t)
sort(colMeans(meanBetas))
```

```
##   Gammaproteobacteria               Bacilli        Actinobacteria
##          -0.221389104          -0.207623650          -0.067049117
##   Alphaproteobacteria    Betaproteobacteria         Flavobacteria
##          -0.022520646          -0.017662369          -0.016562493
```

```
##         Cyanobacteria          Fusobacteria          OD1_no_class
##          -0.004651921          -0.003606308          -0.002844067
##       Erysipelotrichi       Sphingobacteria   Deltaproteobacteria
##          -0.002696590          -0.002472377          -0.001244403
## Epsilonproteobacteria          unclassified           Bacteroidia
##          -0.000737008           0.003457280           0.074021133
##            Clostridia
##           0.108195436
```

```
# max mean abs(B_jp(t))
sort(apply(abs(meanBetas), 2, max))
```

```
##   Deltaproteobacteria Epsilonproteobacteria          OD1_no_class
##           0.002008211           0.003163086           0.003549920
##       Erysipelotrichi       Sphingobacteria          unclassified
##           0.003751179           0.003825799           0.004312284
##         Cyanobacteria          Fusobacteria   Alphaproteobacteria
##           0.006191627           0.006217845           0.030109986
##         Flavobacteria    Betaproteobacteria           Bacteroidia
##           0.030825081           0.047030987           0.087916247
##        Actinobacteria            Clostridia               Bacilli
##           0.120518126           0.152506368           0.272297825
##   Gammaproteobacteria
##           0.342490756
```

This shows the coefficient of Gammaproteobacteria for gestational age has the smallest average mean $\beta_{jp}(t)$ over time and

Generating Datasets

We provide a function to easily generate a dataset if needed for testing or otherwise.

```
n <- 50 # Number of individuals
c <- 10 # Number of categories
p <- 10 # Number of functional covariates
dataList <- generateData(n, c, p)
```

`dataList` contains the generated data: counts, covariates, ids, timepoints and the true parameters. To recover the true function for each covariate, we use the following code:

```
cov <- 1 # the first covariate, not the intercept.
cat <- 1
f <- dataList$funcList[[round(dataList$betaFuncMat[cov, cat])]]
plusMinus <- dataList$betaFuncMat[cov, cat] - round(dataList$betaFuncMat[cov, cat])
betaFunc <- ifelse(plusMinus >= 0, 1, -1)*f(dataList$timePoints)
```

This is only for the covariates. The intercept functions can be found in `dataList$betaIntercepts`.

Conclusion

While we covered all the functions in the `FunCZIDM` package, the functionality is not entirely covered. The reference manual contains all of the function parameters, so please refer to the reference for a further description of the functions.

For plotting, the shiny app associated with the package can be used to generate plots with ease. This app should be downloaded locally, then the samples can be uploaded securely.