# Package: CNVRG (via r-universe)

September 13, 2024

**Title** Dirichlet-Multinomial Modelling of Relative Abundance Data

**Version** 1.0.0

**Maintainer** Joshua Harrison <joshua.grant.harrison@gmail.com>

**Description** Implements Dirichlet multinomial modelling of relative abundance data using functionality provided by the 'Stan' software. The purpose of this package is to provide a user friendly way to interface with 'Stan' that is suitable for those new to modelling. For more regarding the modelling mathematics and computational techniques we use see our publication in Molecular Ecology Resources titled ``Dirichlet multinomial modelling outperforms alternatives for analysis of ecological count data" (Harrison et al. 2020 <doi:10.1111/1755-0998.13128>).

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.1.1

**Biarch** true

**Depends** R (>= 3.4.0)

**Imports** methods, Rcpp (>= 0.12.0), rstan (>= 2.18.1), vegan, rstantools (>= 2.1.1), tibble

**LinkingTo** BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), RcppParallel (>= 5.0.1), rstan (>= 2.18.1), StanHeaders (>= 2.18.0)

**SystemRequirements** GNU make

**NeedsCompilation** yes

**Repository** https://jharrisonecoevo.r-universe.dev

**RemoteUrl** https://github.com/jharrisonecoevo/cnvrg

**RemoteRef** HEAD

**RemoteSha** ba34997803768657605 2945fe555f5df6a88bdbb

# Contents



---

`CNVRG-package` *The 'CNVRG' package.*

---

## Description

This package implements Dirichlet multinomial modeling of relative abundance data using functionality provided by the 'Stan' software. The purpose of this package is to provide a user friendly way to interface with 'Stan' that is suitable for those new modelling.

## References

Stan Development Team (2018). RStan: the R interface to Stan. R package version 2.18.2.

---

`cnvrg_HMC` *Perform Hamiltonian Monte Carlo sampling*

---

## Description

This function uses a compiled Dirichlet multinomial model and performs Hamiltonian Monte Carlo sampling of posteriors using 'Stan'. After sampling it is important to check convergence. Use the summary function and shinystan to do this. If you use this function then credit 'Stan' and 'RStan' along with this package.

## Usage

```
cnvrg_HMC(
  countData,
  starts,
  ends,
  algorithm = "NUTS",
  chains = 2,
```

```
  burn = 500,
  samples = 1000,
  thinning_rate = 2,
  cores = 1,
  params_to_save = c("pi", "p")
)
```

## Arguments

| | |
|---|---|
| countData | A matrix or data frame of counts.The first field should be sample names and the subsequent fields should be integer data. Data should be arranged so that the first n rows correspond to one treatment group and the next n rows correspond with the next treatment group, and so on. The row indices for the first and last sample in these groups are fed into this function via 'starts' and 'ends'. |
| starts | A vector defining the indices that correspond to the first sample in each treatment group. The indexer function can help with this. |
| ends | A vector defining the indices that correspond to the last sample in each treatment group. The indexer function can help with this. |
| algorithm | The algorithm to use when sampling. Either 'NUTS' or 'HMC' or 'Fixed_param'. If unsure, then be like a squirrel. This is "No U turn sampling". The abbreviation is from 'Stan'. |
| chains | The number of chains to run. |
| burn | The warm up or 'burn in' time. |
| samples | How many samples from the posterior to save. |
| thinning_rate | Thinning rate to use during sampling. |
| cores | The number of cores to use. |
| params_to_save | The parameters from which to save samples. Can be 'p', 'pi', 'theta'. |

## Details

It can be helpful to use the indexer function to automatically identify the indices needed for the 'starts' and 'ends' parameters. See the vignette for an example.

Warning: data must be input in the correct organized format or this function will not provide accurate results. See vignette if you are unsure how to organize data. Warning: depending upon size of data to be analyzed this function can take a very long time to run.

## Value

A fitted 'Stan' object that includes the samples from the parameters designated.

## Examples

```
#simulate an OTU table
com_demo <-matrix(0, nrow = 10, ncol = 10)
com_demo[1:5,] <- c(rep(3,5), rep(7,5)) #Alternates 3 and 7
com_demo[6:10,] <- c(rep(7,5), rep(3,5)) #Reverses alternation
fornames <- NA
```

```
for(i in 1:length(com_demo[1,])){
fornames[i] <- paste("otu_", i, sep = "")
}
sample_vec <- NA
for(i in 1:length(com_demo[,1])){
sample_vec[i] <- paste("sample", i, sep = "_")
}
com_demo <- data.frame(sample_vec, com_demo)
names(com_demo) <- c("sample", fornames)

#These are toy data, many more samples, multiple chains, and a longer burn
#are likely advisable for real data.
fitstan_HMC <- cnvrg_HMC(com_demo,starts = c(1,6),
ends=c(5,10),
chains = 1,
burn = 100,
samples = 150,
thinning_rate = 2)
```

---

`cnvrg_VI` *Perform variational inference sampling*

---

## Description

This function uses a compiled Dirichlet multinomial model and performs variational inference estimation of posteriors using 'Stan'. Evaluating the performance of variational inference is currently under development per our understanding. Please roll over to the 'Stan' website and see if new diagnostics are available. If you use this function then credit 'Stan' and 'RStan' along with this package.

## Usage

```
cnvrg_VI(
  countData,
  starts,
  ends,
  algorithm = "meanfield",
  output_samples = 500,
  params_to_save = c("pi", "p")
)
```

## Arguments

| | |
|---|---|
| `countData` | A matrix or data frame of counts.The first field should be sample names and the subsequent fields should be integer data. Data should be arranged so that the first n rows correspond to one treatment group and the next n rows correspond with the next treatment group, and so on. The row indices for the first and last sample in these groups are fed into this function via 'starts' and 'ends'. |

| | |
|---|---|
| starts | A vector defining the indices that correspond to the first sample in each treatment group. The indexer function can help with this. |
| ends | A vector defining the indices that correspond to the last sample in each treatment group. The indexer function can help with this. |
| algorithm | The algorithm to use when performing variational inference. Either 'meanfield' or 'fullrank'. The former is the default. |
| output_samples | The number of samples from the approximated posterior to save. |
| params_to_save | The parameters from which to save samples. Can be 'p', 'pi', 'theta'. |

## Details

It can be helpful to use the indexer function to automatically identify the indices needed for the 'starts' and 'ends' parameters. See the vignette for an example.

Warning: data must be input in the correct organized format or this function will not provide accurate results. See vignette if you are unsure how to organize data. Warning: depending upon size of data to be analyzed this function can take a very long time to run.

## Value

A fitted 'Stan' object that includes the samples from the parameters designated.

## Examples

```
#simulate an OTU table
```

---

# diff_abund *Calculate features with different abundances between treatment groups*

---

## Description

This function determines which features within the matrix that was modeled differ in relative abundance among treatment groups. Pass in a model object, with samples for pi parameters. This function only works for pi parameters.

## Usage

```
diff_abund(model_out, countData, prob_threshold = 0.05)
```

## Arguments

| | |
|---|---|
| model_out | Output of CNVRG modeling functions, including cnvrg_HMC and cnvrg_VI |
| countData | Dataframe of count data that was modeled. Should be exactly the same as those data modeled! The first field should be sample name and integer count data should be in all other fields. This is passed in so that the names of fields can be used to make the output of differential relative abundance testing more readable. |
| prob_threshold | Probability threshold, below which it is considered that features had a high probability of differing between groups. Default is 0.05. |

## Details

The output of this function gives the proportion of samples that were greater than zero after subtracting the two relevant posterior distributions. Therefore, values that are very large or very small denote a high certainty that the distributions subtracted differ. If this concept is not clear, then read Harrison et al. 2020 'Dirichlet-multinomial modeling outperforms alternatives for analysis of microbiome and other ecological count data' in Molecular Ecology Resources. For a simple explanation, see this video: https://use.vg/OSVhFJ

The posterior probability distribution of differences is also output. These samples can be useful for plotting or other downstream analyses. Finally, a list of data frames describing the features that differed among treatment comparisons is output, with the probability of differences and the magnitude of those differences (the effect size) included.

## Value

A dataframe with the first field denoting the treatment comparison (e.g., treatment 1 vs. 2) and subsequent fields stating the proportion of samples from the posterior that were greater than zero (called "certainty of diffs"). Note that each treatment group is compared to all other groups, which leads to some redundancy in output. A list, called ppd_diffs, holding samples from the posterior probability distribution of the differences is also output. Finally, a list of dataframes describing results for only those features with a high probability of differing is output (this list is named: features_that_differed).

## Examples

```
#simulate an OTU table
com_demo <-matrix(0, nrow = 10, ncol = 10)
com_demo[1:5,] <- c(rep(3,5), rep(7,5)) #Alternates 3 and 7
com_demo[6:10,] <- c(rep(7,5), rep(3,5)) #Reverses alternation
fornames <- NA
for(i in 1:length(com_demo[1,])){
fornames[i] <- paste("otu_", i, sep = "")
}
sample_vec <- NA
for(i in 1:length(com_demo[,1])){
sample_vec[i] <- paste("sample", i, sep = "_")
}
com_demo <- data.frame(sample_vec, com_demo)
names(com_demo) <- c("sample", fornames)

out <- cnvrg_VI(com_demo,starts = c(1,6), ends=c(5,10))
diff_abund_test <- diff_abund(model_out = out, countData = com_demo)
```

---

# diversity_calc *Calculate diversity entropies for each replicate*

---

## Description

Calculate Shannon's or Simpson's indices for each replicate while propagating uncertainty in relative abundance estimates through calculations.

## Usage

```
diversity_calc(
  model_out,
  countData,
  params = "pi",
  entropy_measure = "shannon",
  equivalents = T
)
```

## Arguments

| | |
|---|---|
| model_out | Output of CNVRG modeling functions, including cnvrg_HMC and cnvrg_VI or isd_transform |
| countData | Dataframe of count data that was modeled. Should be exactly the same as those data modeled! The first field should be sample name and integer count data should be in all other fields. This is passed in so that the names of fields can be used to make the output of differential relative abundance testing more readable. |
| params | Parameter for which to calculate diversity, can be 'p' or 'pi' or both (e.g., c("pi","p")) |
| entropy_measure | Diversity entropy to use, can be one of 'shannon' or 'simpson' |
| equivalents | Convert entropies into number equivalents. Defaults to true. See Jost (2006), "Entropy and diversity" |

## Details

Takes as input either a fitted Stan object from the cnvrg_HMC or cnvrg_VI functions, or the output of isd_transform. As always, doublecheck the results to ensure the function has output reasonable values. Note that because there are no zero values and all proportion estimates are non zero there is a lot of information within the modeled data. Because diversity entropies are measures of information content, this means there will be a much higher entropy estimate for modeled data than the raw count data. However, patterns of variation in diversity should be similar among treatment groups for modeled and raw data.

## Value

A list that has samples from posterior distributions of entropy metrics

## Examples

```
#simulate an OTU table
com_demo <-matrix(0, nrow = 10, ncol = 10)
com_demo[1:5,] <- c(rep(3,5), rep(7,5)) #Alternates 3 and 7
com_demo[6:10,] <- c(rep(7,5), rep(3,5)) #Reverses alternation
fornames <- NA
for(i in 1:length(com_demo[1,])){
fornames[i] <- paste("otu_", i, sep = "")
}
```

```
sample_vec <- NA
for(i in 1:length(com_demo[,1])){
sample_vec[i] <- paste("sample", i, sep = "_")
}
com_demo <- data.frame(sample_vec, com_demo)
names(com_demo) <- c("sample", fornames)

out <- cnvrg_VI(com_demo,starts = c(1,6), ends=c(5,10))
diversity_calc(model_out = out,params = c("pi","p"),
countData = com_demo, entropy_measure = 'shannon')
```

---

`extract_pi_quantiles` *Extract quantiles of pi parameters*

---

## Description

Provides quantiles of pi parameters for each feature and treatment group.

## Usage

```
extract_pi_quantiles(model_out, probs = c(0.05, 0.5, 0.95))
```

## Arguments

| | |
|---|---|
| `model_out` | Output of CNVRG modeling functions, including cnvrg_HMC and cnvrg_VI |
| `probs` | A vector of quantiles |

## Value

A list specifying quantiles for each feature in each treatment group.

## Examples

```
#simulate an OTU table
com_demo <-matrix(0, nrow = 10, ncol = 10)
com_demo[1:5,] <- c(rep(3,5), rep(7,5)) #Alternates 3 and 7
com_demo[6:10,] <- c(rep(7,5), rep(3,5)) #Reverses alternation
fornames <- NA
for(i in 1:length(com_demo[1,])){
fornames[i] <- paste("otu_", i, sep = "")
}
sample_vec <- NA
for(i in 1:length(com_demo[,1])){
sample_vec[i] <- paste("sample", i, sep = "_")
}
com_demo <- data.frame(sample_vec, com_demo)
names(com_demo) <- c("sample", fornames)

out <- cnvrg_VI(com_demo,starts = c(1,6), ends=c(5,10))
extract_pi_quantiles(model_out = out, probs = c(0.05,0.5,0.95))
```

`extract_point_estimate` *Extract point estimates of multinomial and Dirichlet parameters*

## Description

Provides the mean value of posterior probability distributions for parameters.

## Usage

```
extract_point_estimate(model_out, countData, params = c("pi", "p"))
```

## Arguments

| | |
|---|---|
| `model_out` | Output of CNVRG modeling functions, including cnvrg_HMC and cnvrg_VI |
| `countData` | The count data modeled. |
| `params` | Parameters to be extracted, either pi (Dirichlet) or p (multinomial). |

## Value

A list of of point estimates for model parameters. If both multinomial and Dirichlet parameters are requested then they will be named elements of a list.

## Examples

```
#simulate an OTU table
com_demo <-matrix(0, nrow = 10, ncol = 10)
com_demo[1:5,] <- c(rep(3,5), rep(7,5)) #Alternates 3 and 7
com_demo[6:10,] <- c(rep(7,5), rep(3,5)) #Reverses alternation
fornames <- NA
for(i in 1:length(com_demo[1,])){
fornames[i] <- paste("otu_", i, sep = "")
}
sample_vec <- NA
for(i in 1:length(com_demo[,1])){
sample_vec[i] <- paste("sample", i, sep = "_")
}
com_demo <- data.frame(sample_vec, com_demo)
names(com_demo) <- c("sample", fornames)

out <- cnvrg_VI(com_demo,starts = c(1,6), ends=c(5,10))
extract_point_estimate(model_out = out, countData = com_demo)
```

fungi *Fungal endophytes of Astragalus lentiginosus grown near Reno, NV*

## Description

Fungal endophytes of Astragalus lentiginosus grown near Reno, NV

## Usage

```
fungi
```

## Format

A data frame with columns:

**treatment** A categorical variable describing if a plant was treated with a slurry of endophyte inoculum and whether it was positive or negative for Alternaria fulva.

**Otu10** Contains count data.

**Otu100** Contains count data.

**Otu11** Contains count data.

**Otu12** Contains count data.

**Otu4** Contains count data.

**Otu40** Contains count data.

**Otu42** Contains count data.

**Otu54** Contains count data.

**Otu58** Contains count data.

**Otu6** Contains count data.

**Otu62** Contains count data.

**Otu7** Contains count data.

**Otu70** Contains count data.

**Otu71** Contains count data.

**Otu72** Contains count data.

**Otu74** Contains count data.

**Otu76** Contains count data.

**Otu77** Contains count data.

**Otu79** Contains count data.

**Otu86** Contains count data.

**Otu9** Contains count data.

**Otu92** Contains count data.

**Otu94** Contains count data.

**Otu96** Contains count data.

**Otu97** Contains count data.

**Otu99** Contains count data.

## Source

Joshua G. Harrison

## Examples

```
## Not run:
 fungi

## End(Not run)
```

---

`indexer` *Determine indices for treatment groups*

---

## Description

This function determines the indices for the first and last replicates within a vector describing treatment group.

## Usage

```
indexer(x)
```

## Arguments

| | |
|---|---|
| `x` | Vector input. |

## Value

A list with two named elements that contain start and end indices.

## Examples

```
indexer(c(rep("treatment1",5), rep("treatment2",5)))
```

---

`isd_transform` *Transform data into estimates of absolute abundances using an ISD*

---

## Description

If an internal standard (ISD) has been added to samples such that the counts for that standard are representative of the same absolute abundance, then the ISD can be used to transform relative abundance data such that they are proportional to absolute abundances (Harrison et al. 2020). This function performs this division while preserving uncertainty in relative abundance estimates of both the ISD and the other features present.

## Usage

```
isd_transform(model_out, isd_index, countData, format = "stan")
```

## Arguments

| | |
|---|---|
| model_out | Output of CNVRG modeling functions, including cnvrg_HMC and cnvrg_VI |
| isd_index | The index for the field with information for the internal standard. |
| countData | The count data modeled. |
| format | The output format. Can be either 'or 'samples' or 'ml'. "samples" outputs samples from the posterior probability distribution, the last option ("ml") outputs the mean of posterior samples for each parameter. |

## Details

An index for the ISD must be provided. This should be the field index that corresponds with the ISD. Remember that the index should mirror what has been modeled. Also, note that this function subtracts one from this index because the modeled data have a non integer sample field. If the wrong index is passed in, the output of this function will be incorrect, but there will not be a fatal error or warning.

A simple check that the correct index has been passed to the function is to examine the output and make sure that the field that should correspond with the ISD is one (signifying that the ISD was divided by itself).

Output format can either as means of the samples for each pi parameter or the transformed samples from the posterior distribution for that parameter. Harrison et al. 2020. 'The quest for absolute abundance: the use of internal standards for DNA based community ecology' Molecular Ecology Resources.

## Value

A dataframe, or list, specifying either point estimates for each feature in each treatment group (if output format is 'ml') or samples from the posterior (if output format is 'samples').

## Examples

```
#simulate an OTU table
com_demo <-matrix(0, nrow = 10, ncol = 10)
com_demo[1:5,] <- c(rep(3,5), rep(7,5)) #Alternates 3 and 7
com_demo[6:10,] <- c(rep(7,5), rep(3,5)) #Reverses alternation
fornames <- NA
for(i in 1:length(com_demo[1,])){
fornames[i] <- paste("otu_", i, sep = "")
}
sample_vec <- NA
for(i in 1:length(com_demo[,1])){
sample_vec[i] <- paste("sample", i, sep = "_")
}
com_demo <- data.frame(sample_vec, com_demo)
names(com_demo) <- c("sample", fornames)
```

```
#Model the data
out <- cnvrg_VI(com_demo,starts = c(1,6), ends=c(5,10))
#Transform the data
transformed_data <- isd_transform(model_out = out, countData = com_demo,
isd_index = 3, format = "ml")
```

# Index