



MixSIAPy User Manual

Version 0.1.0
September 2026

By Jing Yang^{1,5}, Heng Dai^{2,3}, Ming Ye⁴, Su Cheng⁵, Yonghua Zhao¹, and Hui Qian⁶

¹School of Land Engineering, Chang'an University, Xi'an 710054, Shaanxi, China.

²State Key Laboratory of Geomicrobiology and Environmental Changes, China University of Geosciences, Wuhan 430074, Hubei, China. Email:

³Hubei Key Laboratory of Yangtze Catchment Environmental Aquatic Science, China University of Geosciences, Wuhan 430074, Hubei, China

⁴Department of Earth, Ocean, and Atmospheric Science, Florida State University, Tallahassee, FL 32306, USA

⁵Shaanxi Biyuan Chenchuang Technology Co., Ltd., Xi'an 710065, China

⁶School of Water and Environment, Chang'an University, Xi'an 710054, Shaanxi, China

Contents

Purpose and scope	4
1 Overview of an analysis	4
1.1 What the model estimates	4
1.2 Supported model structures	5
2 Installation	5
2.1 Recommended environment	5
2.2 Installing from the source repository	5
2.3 Optional dependency groups	6
2.4 Verifying the installation	6
2.5 Installing GPU support	7
3 Ways to run MixSIAPy	7
3.1 Python scripts and notebooks	7
3.2 Local graphical interface	8
3.3 Hosted graphical interface	8
4 Preparing input data	8
4.1 Mixture data	8
4.2 Raw source data	9
4.3 Summarized source data	9
4.4 Source stratification	9
4.5 Concentration dependence	10
4.6 Discrimination data	10
4.7 Source ordering and priors	10
5 Basic scripted analysis	11
6 Specifying covariate effects	12
6.1 Intercept-only model	12
6.2 Categorical fixed effect	12
6.3 Categorical random effect	13
6.4 Nested random effects	13
6.5 Conventional continuous effect	13
6.6 Formula-based compositional regression	14
6.7 Combining formula terms with random effects	15
6.8 How formula variables are encoded	15
7 Priors and error structures	16
7.1 Dirichlet prior for the baseline composition	16
7.2 Priors for covariate effects	16
7.3 Error-model selection	17
8 Sampling and computational settings	18
8.1 Understanding a sampling schedule	18
8.2 Run presets	18
8.3 Inference backends	19
8.4 Selecting a computational device	20
8.5 Additional sampling arguments	21
9 Statistical principles	21
9.1 Notation and compositional constraints	22
9.2 Source-distribution model	22
9.2.1 Raw source observations	22
9.2.2 Source summary statistics	23
9.2.3 Factor-stratified source distributions	23
9.3 Baseline source composition	24
9.4 Isometric log-ratio representation	24
9.5 Compositional predictor	24
9.5.1 Categorical fixed effects	25
9.5.2 Categorical random effects	25
9.5.3 Nested random effects	25
9.5.4 Conventional continuous effect	26

9.5.5 Formula-based compositional regression	26
9.6 Expected tracer composition of a mixture	27
9.6.1 Concentration-independent mixing.....	27
9.6.2 Concentration-dependent mixing.....	27
9.7 Propagation of source and discrimination uncertainty	27
9.8 Mixture-error models	28
9.8.1 Combined process and residual error.....	28
9.8.2 Residual-only error	28
9.8.3 Process-only error	29
9.9 Joint posterior distribution	29
9.10 Posterior predictions and contrasts	30
9.11 Identifiability and interpretation	30
10 Diagnosing a fitted model	30
10.1 Compact diagnostic report	31
10.2 $\hat{R}$ and effective sample size.....	31
10.3 Hamiltonian Monte Carlo diagnostics	31
10.4 Graphical diagnostic checks.....	32
10.5 Responding to diagnostic failures	32
11 Summarizing, predicting, and interpreting posterior results	33
11.1 Posterior summaries	33
11.2 Conventional continuous-effect prediction	34
11.3 Formula-based prediction	34
11.4 Interpreting contrasts and interactions	35
11.5 Combining sources after fitting.....	35
11.6 Comparing models	36
12 Visualization, export, and reproducibility	36
12.1 Saving the complete analysis	36
12.2 Reporting graphical results	37
12.3 Reproducibility checklist	37
13 Scope, limitations, and troubleshooting	38
13.1 Statistical limitations.....	38
13.2 Data and model-specification problems.....	38
13.3 Installation and environment problems	39
13.4 Performance and output problems	40
Appendix A: Public API.....	41
A.1 Data loading	41
A.2 Model construction and sampling	41
A.3 Results and utilities	42
Appendix B: Recommended workflow checklist.....	42
References.....	44

Purpose and scope

MixSIAPy is an open-source Python package for Bayesian stable isotope mixing models. It estimates the proportional contributions of candidate sources to observed mixtures while representing uncertainty in source signatures, discrimination values, covariate effects, and mixture variability. It reimplements the core model structures used by MixSIAR (<https://brianstock.github.io/MixSIAR/>) and adds formula-based compositional regression, interchangeable No-U-Turn Sampler (NUTS) backends, optional GPU execution, and graphical access through both locally installed interfaces and a hosted web application.

This manual serves two purposes. Chapters 1 through 8 show how to install the package, prepare files, specify a model, run an analysis, inspect diagnostics, and export results. Chapters 9 through 13 explain the statistical model and the computational principles behind each option. The appendices provide a compact API reference and a recommended workflow checklist.

MixSIAPy 0.1.0 is a research-preview release. Its models should be used with the same care required for any Bayesian mixing analysis. A completed run is not automatically a reliable run, source separation, prior sensitivity, posterior convergence, and the ecological meaning of the predictors must all be evaluated.

1 Overview of an analysis

A MixSIAPy analysis follows six operations:

- (1) Prepare a mixture table, a source table, and a discrimination table.
- (2) Load and validate the three tables.
- (3) Specify source-proportion priors, covariate effects, and an error model.
- (4) Build and sample the Bayesian model.
- (5) Check convergence and sampling diagnostics.
- (6) Summarize, predict, plot, and export posterior results.

The same analysis kernel is used by Python scripts, the locally installed graphical interface, and the hosted web application. The interfaces differ in how settings are entered and where computation occurs; they do not define different statistical models.

1.1 What the model estimates

For each mixture observation, the model estimates a vector of source proportions. Every component is positive and the components sum to one. Depending on the selected model, MixSIAPy can also estimate:

- global source proportions;
- source proportions for categorical fixed-factor levels;
- variation among random-factor levels;
- nested random-group effects;
- a conventional continuous-covariate relationship;

- multivariable and interaction effects defined by a composition formula;
- uncertain source means, variances, and correlations;
- residual scaling or covariance parameters for selected error structure;
- posterior predictions for specified covariate combinations.

1.2 Supported model structures

The current implementation supports raw or summarized source data, optional source stratification by one mixture factor, concentration-dependent mixing, uninformative or informative Dirichlet priors, conventional fixed and random effects, two-level nesting of random factors, one conventional continuous effect, formula-based compositional regression, and three error structures.

The conventional MixSIAR-compatible interface accepts at most two categorical grouping factors and one continuous covariate. The formula interface accepts multiple numerical and categorical predictors and two-way interactions. Formula predictors describe systematic regression relationships. Random and nested grouping effects remain specified through the conventional factor arguments and are limited to two grouping factors.

2 Installation

2.1 Recommended environment

Python 3.11 or 3.12 in a dedicated environment is recommended. Python 3.9 or newer is supported by the package metadata. A dedicated environment prevents conflicts among PyMC, JAX, NumPy, and plotting libraries.

Using conda:

```
conda create -n mixsiapy python=3.11
conda activate mixsiapy
python -m pip install --upgrade pip
```

Using the standard Python environment manager on Windows:

```
py -3.11 -m venv .venv
.\.venv\Scripts\Activate.ps1
python -m pip install --upgrade pip
```

On Linux or macOS, activate a virtual environment with:

```
python3 -m venv .venv
source .venv/bin/activate
python -m pip install --upgrade pip
```

2.2 Installing from the source repository

Clone the repository, enter its root directory, and install the required optional dependency groups:

```
git clone https://github.com/jyangfsu/MixSIAPy.git
cd MixSIAPy
python -m pip install -e ".[bayes,gui]"
```

The editable installation is useful during development because changes to the local source code become available without rebuilding the package. Remove `-e` for a conventional source installation.

When a PyPI release is available, the corresponding command is:

```
python -m pip install "mixsiapy[bayes,gui]"
```

2.3 Optional dependency groups

MixSIAPy separates optional features into dependency groups so that users need to install only the components required for their workflow. The names in square brackets in an installation command are package extras. The `bayes` extra provides the standard PyMC analysis environment, `fast` adds Nutpie for compiled CPU sampling, `jax` enables the NumPyro and BlackJAX backends, and `gui` installs the local graphical interface. Multiple extras can be installed together in a single command. The available groups are summarized below.

Extra	Purpose	Main packages
bayes	Default Bayesian inference	PyMC, ArviZ
fast	Compiled CPU sampling	PyMC, ArviZ, Nutpie
jax	JAX-based sampling	JAX, NumPyro, BlackJAX
all-backends	All four inference backends	PyMC, Nutpie, NumPyro, BlackJAX
gui	Local graphical interface	Streamlit, Plotly
test	Automated tests	pytest
dev	Package development	build, ruff

For a complete local installation:

```
python -m pip install -e ".[bayes,all-backends,gui,test]"
```

Most users do not need every optional dependency. For example, a standard analysis using the default PyMC backend requires only `bayes`, whereas a local graphical installation using the default backend requires `bayes` and `gui`.

2.4 Verifying the installation

After installation, users should verify that Python is importing the intended copy of MixSIAPy and that the required optional components are available. The following commands report the installed package version and location, identify the usable inference backends, and confirm that the bundled example data can be located. They should be run in the same Python environment that will be used for the analysis.

```
import mixsiapy
from mixsiapy import backend_status, get_resource_path

print(mixsiapy.__version__)
print(mixsiapy.__file__)
print(backend_status())
```

```
| print(get_resource_path("data"))
```

The value of `mixsiapy.__file__` identifies the installation that Python has loaded, which is useful when several Python environments are present. The output of `backend_status()` shows which inference backends can be used in the current environment. The final command returns the location of the example datasets distributed with the package.

If these checks run without an import error, the core package and its installed backends are available. Users working from a source checkout can additionally run the automated test suite from the repository root:

```
| python -m pytest
```

2.5 Installing GPU support

GPU sampling is available only through the NumPyro and BlackJAX backends. The `jax` extra installs the Python-side dependencies, but a CUDA-enabled JAX build must match the installed driver and CUDA environment. Linux or WSL2 with an NVIDIA GPU is recommended. Native Windows is appropriate for PyMC and Nutpie CPU sampling; GPU-enabled JAX is generally easier to configure in WSL2 or Linux.

After following the current JAX installation instructions, verify the device:

```
| import jax
| print(jax.devices())
```

At least one device must report platform `gpu`. MixSIAPy stops with an error if `device="gpu"` is requested and JAX cannot see a compatible GPU. It never silently labels a CPU run as a GPU run.

3 Ways to run MixSIAPy

3.1 Python scripts and notebooks

The Python interface offers full control and is the preferred route for reproducible research, repeated analyses, custom predictions, and integration with other workflows. A minimal import is:

```
| from mixsiapy import (
|     load_mix_data,
|     load_source_data,
|     load_discr_data,
|     run_model,
|     save_results,
| )
```

Installed examples can be located without assuming a particular environment directory:

```
| from mixsiapy import get_resource_path
| print(get_resource_path("examples", "wolves.py"))
| print(get_resource_path("data", "wolves_consumer.csv"))
```

Run an example from a terminal:

```
python path/to/wolves.py --run test
python path/to/wolves.py --run normal --output outputs/wolves
python path/to/wolves.py --build-only
```

3.2 Local graphical interface

Install the GUI dependency group and start the application:

```
python -m pip install -e ".[bayes,gui]"
mixsiapy-gui
```

The interface opens in the default browser, but the application and model computation remain on the local computer. The workflow contains four stages: data input, model configuration, sampling, and results. It can load an installed example or uploaded CSV files; configure effects, priors, error models, run presets, backends, and devices; display diagnostics and plots; and export results together with an editable reproduction script.

3.3 Hosted graphical interface

The hosted application is available at: <https://huggingface.co/spaces/jyang-research/MixSIAPy>.

It requires no local Python installation. Computation runs on the remote Space, so public deployment limits apply. The hosted CPU service is appropriate for learning the workflow, validating files, building models, and running small test analyses. Long schedules and demanding hierarchical models should be run locally or on managed computing resources. Download all result files after a run because hosted storage can be temporary.

4 Preparing input data

MixSIAPy uses three CSV files. Column names are case-sensitive. Tracer names supplied to `iso_names` determine the required mean, standard-deviation, and concentration column names in the other files.

4.1 Mixture data

Each row represents one observed mixture or consumer. The table must contain every tracer named in `iso_names`. It may also contain factor columns, conventional continuous covariates, and columns referenced by `composition_formula`.

Example:

```
delta2H,delta18O,species,diversity,biomass,spcomp
-45.89,-6.45,Bean,mixture,4.92,BePe
-45.95,-6.41,Pea,mixture,2.72,BePe
```

Rules:

- tracer values cannot be missing;
- each factor and covariate named in the model must exist in the table;

- a conventional continuous covariate must be numeric and have nonzero variance;
- formula variables must not be constant;
- text-valued formula variables are categorical;
- numerically coded categories are interpreted as continuous, so convert them to text labels before loading if categorical treatment is intended.

4.2 Raw source data

For raw source data, the first column contains source names and every subsequent source replicate occupies one row. The table must contain the tracer columns used in the mixture table.

```
Source,delta2H,delta18O
Shallow (0-10 cm),-30.82,-3.03
Shallow (0-10 cm),-36.37,-4.68
Middle (15-30 cm),-52.11,-7.40
```

Every source group needs at least two observations and a non-zero sample standard deviation for every tracer. With two or more tracers, raw replicates allow the model to estimate within-source covariance.

4.3 Summarized source data

When only source summaries are available, the first column contains source names. For each tracer named x , provide Mean_x and SD_x , together with a sample-size column named n .

```
Source,Meand13C,SDd13C,Meand15N,SDd15N,n
Deer,-26.88,1.10,3.07,1.35,24
Marine Mammals,-16.20,0.80,13.10,1.20,18
Salmon,-20.40,0.90,11.70,1.10,25
```

Summary inputs describe marginal means and variances. They do not contain the replicate-level information needed to reconstruct covariance among tracers, so their source covariance is diagonal. Sample sizes must be greater than one. Setting an arbitrarily large n makes the posterior source mean highly concentrated near the reported mean; this can approximate a fixed-mean analysis, but it weakens the fully Bayesian propagation of source uncertainty and should be reported explicitly.

4.4 Source stratification

Source signatures may differ among levels of a mixture factor, such as region, habitat, or season. In this case, add the factor as a column in the source table and provide a separate row for every source within every factor level. For example, the wolves source file contains region-specific summary statistics:

```
Source,Region,Meand13C,SDd13C,Meand15N,SDd15N,n
Deer,1,-26.88,1.10,3.07,1.35,24
Deer,2,-27.15,0.67,2.80,1.14,37
Deer,3,-27.47,0.75,2.76,2.32,9
Salmon,1,-18.58,1.34,12.26,1.18,6
Salmon,2,-22.38,2.85,11.92,1.12,5
```

```
|Salmon, 3, -22.38, 2.85, 11.92, 1.12, 5
```

The stratification column is then specified using `source_factors`:

```
source = load_source_data(  
    filename="wolves_sources.csv",  
    source_factors="Region",  
    conc_dep=False,  
    data_type="means",  
    mix=mix,  
)
```

For each mixture observation, MixSIAPy uses the source distributions associated with the same `Region` level. Only one source-stratification factor is currently supported. The factor must also be declared when loading the mixture data, and every mixture-factor level must occur in the source table. Each level must contain the same set of source names, although their means, standard deviations, and sample sizes may differ among levels.

4.5 Concentration dependence

When source concentrations differ, add one positive concentration column for every tracer. The required name is `Conc` followed by the tracer's name, for example `Concd13C` and `Concd15N`. Set `conc_dep=True` when loading the source table.

Concentration dependence changes the expected tracer value because a source can contribute a different proportion of the measured element than of the total material. It does not change the definition of the dietary or source proportion itself.

4.6 Discrimination data

The discrimination table contains one row per source. Its first column contains source names. For each tracer `x`, provide `Meanx` and `SDx`.

```
Source, Meandelta2H, SDdelta2H, Meandelta18O, SDdelta18O  
Deep (50-75 cm), 0, 0, 0, 0  
Middle (15-30 cm), 0, 0, 0, 0  
Shallow (0-10 cm), 0, 0, 0, 0
```

Discrimination represents a systematic shift between a source signature and the signature expected in the mixture. Trophic enrichment factors are a common example. Zero means and standard deviations are appropriate only when the underlying scientific assumption is no discrimination and no discrimination uncertainty.

4.7 Source ordering and priors

The loader sorts source names alphabetically. Elements of `alpha_prior` are assigned to sources in this sorted order. Always inspect `source["source_names"]` before specifying an informative prior:

```
|print(source["source_names"])
```

By default, `alpha_prior=1` assigns a generalist Dirichlet prior with equal concentration parameters to all sources. For three sources, it is equivalent to:

```
alpha_prior = [1, 1, 1]
```

An informative prior must contain one positive value for each source, arranged in the printed source order:

```
alpha_prior = [2, 1, 4] # Must match source["source_names"]
```

5 Basic scripted analysis

The wolves example below uses summarized sources stratified by region and two random factors, with `pack` nested within `region`.

```
from pathlib import Path
from mixsiapy import (
    get_resource_path,
    load_mix_data,
    load_source_data,
    load_discr_data,
    run_model,
    save_results,
)

data = get_resource_path("data")
output = Path("outputs/wolves")

mix = load_mix_data(
    filename=data / "wolves_consumer.csv",
    iso_names=["d13C", "d15N"],
    factors=["Region", "Pack"],
    fac_random=[True, True],
    fac_nested=[False, True],
    cont_effects=None,
)

source = load_source_data(
    filename=data / "wolves_sources.csv",
    source_factors="Region",
    conc_dep=False,
    data_type="means",
    mix=mix,
)

discr = load_discr_data(
    filename=data / "wolves_discrimination.csv",
    mix=mix,
)

fit = run_model(
    run="normal",
    mix=mix,
```

```

        source=source,
        discr=discr,
        alpha_prior=[1, 1, 1],
        process_err=True,
        resid_err=True,
        backend="pymc",
        device="cpu",
        target_accept=0.95,
        random_seed=42,
    )

save_results(fit, output, source["source_names"], mix)

```

The returned `fit` object is an ArviZ `InferenceData`. It contains labeled posterior arrays, sampler statistics, pointwise mixture log likelihood where available, and runtime metadata.

6 Specifying covariate effects

6.1 Intercept-only model

For an intercept-only model, no factors, continuous covariates, or compositional-regression formula are specified. Because the corresponding arguments default to `None`, they can be omitted:

```

mix = load_mix_data(
    filename="consumer.csv",
    iso_names=["d13C", "d15N"],
)

```

This call is equivalent to explicitly setting `factors=None`, `fac_random=None`, `fac_nested=None`, `cont_effects=None`, and `composition_formula=None`. The model estimates a single global source composition shared by all mixture observations, while the selected error model represents variation around the expected tracer values.

6.2 Categorical fixed effect

Use a categorical fixed effect when the analysis aims to estimate compositional differences among a specific set of observed groups, such as habitats or treatments. Supply the grouping column through `factors`, set the corresponding entry in `fac_random` to `False`, and use `fac_nested=False` when the factor is not nested within another factor:

```

mix = load_mix_data(
    filename="consumer.csv",
    iso_names=["d13C", "d15N"],
    factors=["Habitat"],
    fac_random=[False],
    fac_nested=[False],
)

```

The first sorted factor level is used as the reference and its isometric log-ratio transformation (ILR) offset is fixed at zero. Each remaining level is assigned a

separate coefficient vector with an independent prior. The posterior distributions of these coefficients quantify compositional contrasts with the reference level. Because the coefficients do not share a group-level distribution, they are not partially pooled across levels.

6.3 Categorical random effect

Use a categorical random effect when the observed groups are regarded as members of a broader group population and the analysis is concerned with both group-specific compositions and the overall magnitude of variation among groups. Supply the grouping column through `factors` and set its entry in `fac_random` to `True`.

```
mix = load_mix_data(  
    filename="consumer.csv",  
    iso_names=["d13C", "d15N"],  
    factors=["Site"],  
    fac_random=[True],  
    fac_nested=[False],  
)
```

Each site receives an ILR offset drawn from a shared zero-centred distribution. The standard deviation of this distribution is estimated jointly with the site offsets and the other model parameters. This hierarchical structure partially pools weakly informed site estimates toward the composition predicted by the remaining model terms.

6.4 Nested random effects

Nested random effects are appropriate when the levels of one grouping factor occur within the levels of another. For example, individual packs may be nested within regions because every pack belongs to one region. Enter the higher-level factor first and mark the lower-level factor as nested.

```
mix = load_mix_data(  
    filename="consumer.csv",  
    iso_names=["d13C", "d15N"],  
    factors=["Region", "Pack"],  
    fac_random=[True, True],  
    fac_nested=[False, True],  
)
```

Each pack (lower-level group) must belong to exactly one region (higher-level group). For every observation, the predictor contains both the offset for its region and the offset for its pack. The region effects represent compositional variation among regions, whereas the pack effects represent additional variation among packs within regions.

6.5 Conventional continuous effect

Use the conventional continuous-effect interface when the model contains one numerical covariate and no interaction involving that covariate is required. The name of the numerical column is supplied through `cont_effects`.

```
mix = load_mix_data(  
    filename="consumer.csv",
```

```
iso_names=["d13C", "d15N"],
cont_effects=["Length"],
)
```

The covariate is centered and divided by its sample standard deviation. One ILR coefficient vector describes the common compositional relationship with the covariate. The conventional interface supports one continuous effect and does not construct predictor interactions.

6.6 Formula-based compositional regression

Use the formula interface when several categorical or numerical predictors must be analyzed jointly or when interactions between predictors are required. All variables referenced in the formula must be columns in the mixture-data table. Numerical columns are treated as standardized continuous predictors, whereas nonnumeric columns are encoded as categorical predictors.

For example, the crop-water model includes categorical predictors for species and planting diversity, a numerical predictor for biomass, and interactions that allow the biomass and diversity relationships to differ among species:

```
mix = load_mix_data(
    filename="crop_water_consumer.csv",
    iso_names=["delta2H", "delta18O"],
    composition_formula="species * biomass + species *
diversity",
)
```

The `*` operator is an expansion operator, not numerical multiplication. This formula becomes:

```
species + biomass + diversity + species:biomass +
species:diversity
```

It estimates baseline differences among species, a biomass association, a diversity contrast, species-specific deviations from the biomass association, and species-specific deviations from the diversity contrast. Every term changes the complete source composition rather than fitting an independent regression to each source proportion.

The `:` operator requests an interaction without automatically adding both main effects:

```
composition_formula="species + biomass + species:biomass"
```

Only two-way interactions are supported. A response-and-predictor notation is also accepted:

```
composition_formula="composition ~ species * biomass + species
* diversity"
```

The response name is ignored because the modelled response is always the latent source composition. For clarity, the predictor-only form is generally recommended.

6.7 Combining formula terms with random effects

The formula interface can be combined with conventional grouping arguments when the analysis requires flexible regression relationships together with random or nested variation. Formula terms define the systematic relationships of source composition with the predictors, whereas factors, `fac_random`, and `fac_nested` specify the grouping structure.

In the crop-water-uptake analysis, species, biomass, planting diversity, and their interactions are included in the formula, while planting composition is included as a random grouping effect:

```
mix = load_mix_data(  
    filename="crop_water_consumer.csv",  
    iso_names=["delta2H", "delta18O"],  
    factors=["spcomp"],  
    fac_random=[True],  
    fac_nested=[False],  
    cont_effects=None,  
    composition_formula="species * biomass + species *  
diversity",  
)
```

Here, the formula-generated terms estimate differences among species, biomass-related changes, diversity contrasts, and their interactions. The `spcomp` random effect represents additional compositional variation among the particular crop combinations that is not explained by those predictors.

Variables used as random or nested grouping factors should not normally be entered again as categorical predictors in the formula. Doing so would assign both a systematic coefficient and a group-specific deviation to the same variable and should be used only when that decomposition is scientifically intended.

The `composition_formula` and conventional `cont_effects` interfaces should not be used simultaneously. When a formula is specified, all numerical regression predictors should be included in that formula. The conventional factor arguments may still be used for required fixed, random, or nested grouping effects, although the recommended combined workflow uses the formula for fixed regression relationships and the conventional arguments for random or nested grouping structures.

6.8 How formula variables are encoded

Numerical columns are centered and scaled automatically. Nonnumeric columns are sorted, the first level becomes the reference, and the remaining levels are treatment contrasts. The formula intercept is omitted because the baseline composition is represented by `p_global`.

The fitted coefficient matrix has one row for each encoded design-matrix column and one column for each of the $(K - 1)$ ILR coordinates. A categorical predictor with several levels therefore generates several design columns. An interaction can generate several additional columns. Inspect the exact matrix labels with:

```
print(mix["regression"]["columns"])
print(mix["regression"]["metadata"])
```

7 Priors and error models

Priors describe the plausible values of unknown quantities before the mixture observations are considered. In MixSIAPy, they regularize the baseline source composition, covariate coefficients, and variation among random-factor levels. The source and discrimination distributions are additionally informed by the corresponding input data, as described in Section 4.

The error model specifies how variability around the expected tracer values is represented. Prior and error-model choices affect the posterior estimates and should therefore be based on the scientific design rather than selected only to improve convergence.

7.1 Dirichlet prior for the baseline composition

The baseline source proportions are assigned a Dirichlet prior. A scalar value of one or a vector of ones is the uninformative prior:

```
alpha_prior = 1
# Equivalent for three sources
alpha_prior = [1, 1, 1]
```

An informative prior uses positive source-specific concentration parameters:

```
alpha_prior = [4, 1, 2]
```

The vector must contain one positive value for each source and follow the source order stored in `source["source_names"]`. The prior mean for each source is calculated by dividing its alpha value by the sum of all alpha values. In this example, the prior mean composition is approximately [0.57, 0.14, 0.29]. Multiplying every alpha by the same constant preserves the prior mean but increases prior concentration. Prior sensitivity should be examined when conclusions depend on strongly overlapping or poorly separated sources.

7.2 Priors for covariate effects

Covariate effects operate in ILR coordinates rather than directly on individual `source` proportions. MixSIAPy currently uses the following priors:

Effect	Prior
Conventional fixed-factor contrast	<i>Normal</i> (0, 1)
Conventional continuous slope	<i>Normal</i> (0, 1000)
Formula regression coefficient	<i>Normal</i> (0, 6.25)
Random-factor level offset	<i>Normal</i> (0, σ^2) with σ follows <i>Uniform</i> (0, 20)

A prior mean of zero represents no compositional difference from the baseline prediction. Fixed-factor levels receive separate coefficient vectors and are not pooled.

Random-factor levels share an estimated factor SD of σ , allowing partial pooling across levels. These priors are assigned automatically when the model is constructed and do not need to be supplied separately.

For hierarchical random effects, the noncentred parameterization can be requested during model fitting:

```
# Define Site as a random factor
mix = load_mix_data(
    filename="consumer.csv",
    iso_names=["d13C", "d15N"],
    factors=["Site"],
    fac_random=[True],
    fac_nested=[False],
)

# Fit the model using the noncentred parameterization
fit = run_model(
    run="normal",
    mix=mix,
    source=source,
    discr=discr,
    noncentered=True,
)
```

By default, each random-factor offset is sampled directly from $Normal(0, \sigma^2)$, where σ is the estimated factor SD. With `noncentered=True`, MixSIAPy instead samples a standardized value from $Normal(0, 1)$ and multiplies it by σ to obtain the random-factor offset. The two parameterizations define the same random-effect distribution, but the noncentred form can be easier for NUTS to sample when σ is weakly identified.

7.3 Error model selection

MixSIAPy also supports three error models through the `process_err` and `resid_err` arguments:

<code>process_err</code>	<code>resid_err</code>	Model
True	True	Combined process and residual error
False	True	Residual-only error
True	False	Process-only error
False	False	Invalid

The combined model propagates uncertainty in source and discrimination signatures and estimates an additional residual multiplier. It is appropriate when both propagated source uncertainty and unexplained variability among mixture observations are expected.

The residual-only model represents variation around the expected mixture using a residual variance or covariance structure, without adding an explicit process-variance term to the mixture likelihood.

The process-only model derives mixture variance from source and discrimination uncertainty without estimating an additional residual component. This structure is required when only one mixture observation is available because residual variability cannot be estimated from a single observation.

At least one of `process_err` and `resid_err` must be `True`. Error-structure selection is a scientific modelling decision rather than only a convergence setting. Models with different error structures may produce different posterior uncertainty and should be compared only when their assumptions are scientifically defensible.

8 Sampling and computational settings

After the data, priors, covariates, and error structure have defined the statistical model, `run_model` controls how its posterior distribution is sampled. The main computational choices are the sampling schedule, inference backend, computational device, and NUTS settings.

These choices can affect runtime and Monte Carlo efficiency. They should not alter the scientific meaning of the model, although an inadequately sampled posterior can produce unstable estimates.

8.1 Understanding a sampling schedule

A sampling schedule contains four principal quantities:

- Chains are independent sampling sequences started from different initial states. Agreement among chains is required for convergence assessment.
- Warm-up iterations adapt the NUTS step size and mass matrix. They are discarded and are not included in posterior summaries.
- Posterior draws are generated after warm-up and retained for inference.
- Thinning retains only every specified draw after sampling.

Thinning reduces the size of the stored posterior object, but it does not improve exploration of the posterior or reduce the time needed to generate the original draws. It should therefore not be used as a substitute for effective sampling. The total number of retained draws is the number of chains multiplied by the draws per chain and divided by the thinning interval.

8.2 Run presets

The `run` argument selects a predefined computational schedule:

Preset	Chains	Warm-up per chain	Draws per chain before thinning	Thinning	Retained draws in total
--------	--------	-------------------	---------------------------------	----------	-------------------------

test	2	200	200	1	400
very short	3	1,000	1,000	1	3,000
short	3	2,000	2,000	1	6,000
normal	3	5,000	5,000	1	15,000
long	4	10,000	10,000	2	20,000
very long	4	20,000	20,000	5	16,000
extreme	4	50,000	50,000	10	20,000

The test preset is intended only to detect input errors, model-construction failures, missing dependencies, and major sampling problems. Its output should not normally be used for scientific interpretation.

The test preset should be used only to check data loading, model construction, and software configuration. Scientific analyses should normally begin with normal. Longer schedules can be used when the model is correctly specified but the retained posterior sample remains insufficient.

A longer schedule does not guarantee convergence. Divergences, poor chain mixing, or persistently low effective sample sizes may indicate problems with model structure, source separation, priors, or parameterization. These problems should be examined before increasing the number of iterations.

A custom schedule can be supplied as a dictionary when the predefined options are not suitable:

```
fit = run_model(
    run={"chains": 4, "tune": 4000, "draws": 4000, "thin": 1},
    mix=mix,
    source=source,
    discr=discr,
    random_seed=42,
)
```

Use at least three chains when chain-based convergence diagnostics are required. Four chains provide a stronger basis for comparing chain behavior, although they also increase computation. Each call to `run_model` starts a new fit. For example, running long after normal does not continue the previous chains.

8.3 Inference backends

MixSIAPy constructs one PyMC model and can dispatch that model to four NUTS backends:

Backend	Device	Main characteristic
pymc	CPU	Default and broadest compatibility
nutpie	CPU	Compiled sampler that can reduce execution overhead

numpyro	CPU or GPU	JAX-compiled NUTS
blackjax	CPU or GPU	JAX-based NUTS implementation
auto	Selected automatically	Prefers installed accelerated backends under platform rules

Select a backend explicitly with:

```
fit = run_model(
    run="normal",
    mix=mix,
    source=source,
    discr=discr,
    backend="nutpie",
)
```

All backends receive the same PyMC model and target the same posterior distribution. However, their compilation, adaptation, and chain-execution strategies differ. Runtime and sampling diagnostics may therefore differ among backends. Native PyMC provides the broadest compatibility. Nutpie offers compiled CPU sampling. NumPyro and BlackJAX use JAX and support CPU or GPU execution. No backend is expected to be fastest or most stable for every model.

When `backend="auto"` is used, Windows prefers Nutpie and then PyMC. Linux and macOS prefer Nutpie, NumPyro, BlackJAX, and PyMC in that order, subject to package availability. When a GPU is explicitly requested, automatic selection considers only NumPyro and BlackJAX.

Automatic selection identifies an available backend but does not test which backend is fastest for the current model. An explicitly requested backend never silently falls back to another backend.

8.4 Selecting a computational device

The `device` argument accepts `auto`, `cpu`, or `gpu`:

```
fit = run_model(
    run="normal",
    mix=mix,
    source=source,
    discr=discr,
    backend="numpyro",
    device="gpu",
    target_accept=0.95,
)
```

With `device="auto"`, a JAX backend uses an available GPU and otherwise runs on the CPU. The setting `device="cpu"` explicitly selects CPU execution. The setting `device="gpu"` requires NumPyro or BlackJAX and a compatible GPU detected by JAX. Native PyMC and Nutpie are CPU backends in MixSIAPy and reject an explicit GPU request. If a compatible GPU is unavailable, `device="gpu"` raises an error rather than silently running on the CPU.

GPU execution is not necessarily faster for small models. Model conversion, compilation, and data transfer can dominate short analyses. GPU acceleration is more likely to be useful for large models, long sampling schedules, or many chains executed in parallel.

8.5 Additional sampling arguments

Additional sampling options can be passed through `run_model`:

```
fit = run_model(  
    run="normal",  
    mix=mix,  
    source=source,  
    discr=discr,  
    backend="pymc",  
    target_accept=0.97,  
    max_treedepth=14,  
    cores=3,  
    random_seed=42,  
)
```

The principal options are:

- `target_accept` controls the target acceptance probability. Increasing it can reduce some divergences but usually increases runtime.
- `max_treedepth` limits the maximum NUTS trajectory length.
- `cores` controls the number of CPU processes used to run chains.
- `random_seed` supports reproducible sampling.

On Windows, MixSIAPy uses `cores=1` by default to avoid multiprocessing failures in some Python environments. Users with a tested environment may override this setting.

A higher `target_accept` may help when only a small number of divergences remains. It should not be used to hide persistent sampling problems. Some arguments are backend specific, so every fit should be checked for warnings and convergence diagnostics.

The returned result records the resolved `backend`, computational device, package versions, model-construction time, sampling time, and total wall time in `fit.attrs`.

The fitted posterior is returned as an ArviZ `InferenceData` object. The statistical principles underlying the model are described in Chapter 9, followed by convergence assessment in Chapter 10 and posterior summaries, predictions, visualization, and result export in Chapter 11.

9 Statistical principles

This chapter describes the statistical model implemented in MixSIAPy. The formulation follows the hierarchical structure of MixSIAR while expressing the model using the parameterization employed by the Python implementation. It covers

source uncertainty, compositional predictors, concentration dependence, discrimination uncertainty, alternative error models, and joint Bayesian inference.

9.1 Notation and compositional constraints

Consider N mixture observations, K candidate sources, J tracers, and n_k source observations from source k . For mixture observation i , let $\mathbf{y}_i^m$ denote the i -th observed mixture observation and $\mathbf{p}_i$ its K -source proportion vector:

$$\mathbf{y}_i^m = (y_{i1}^m, \dots, y_{iJ}^m), \quad \mathbf{p}_i = (p_{i1}, \dots, p_{iK}) \quad (9.1)$$

The source proportions form a composition and must satisfy

$$p_{ik} > 0, \quad \sum_{k=1}^K p_{ik} = 1 \quad (9.2)$$

The components of $\mathbf{p}_i$ cannot be modelled independently because increasing one source contribution requires at least one other contribution to decrease. For source k , let $\boldsymbol{\mu}_k^s$ denotes its J -tracer mean vector and $\boldsymbol{\Sigma}_k^s$ its $(K \times K)$ -dimensional within-source covariance matrix. The discrimination mean is λ_k , its tracer-specific variances are ω_{kj}^2 , and q_{kj} is the concentration associated with source k and tracer j .

9.2 Source-distribution model

MixSIAPy accepts either raw source observations or source summary statistics. Both options estimate the source parameters rather than automatically fixing them at their sample values, but only raw observations contain the information required to estimate covariance among tracers.

9.2.1 Raw source observations

Let $\mathbf{y}_{kr}^s$ be the J -tracer vector measured for replicate r of source k . Raw source observations follow

$$\mathbf{y}_{kr}^s \sim \text{Normal}(\boldsymbol{\mu}_k^s, \boldsymbol{\Sigma}_k^s). \quad (9.3)$$

Before model fitting, each tracer is standardized separately using the pooled mean and standard deviation calculated from the mixture and source data. After tracer standardization, each component of the source mean receives the diffuse prior

$$\mu_{kj}^s \sim \text{Normal}(0, 1000). \quad (9.4)$$

The source covariance matrices $\boldsymbol{\Sigma}_k^s$ is constructed as:

$$\boldsymbol{\Sigma}_k^s = \mathbf{D}_k \mathbf{R}_k \mathbf{D}_k. \quad (9.5)$$

Here, $\mathbf{D}_k$ is diagonal with elements σ_{kj}^s defined as:

$$\tau_{kj}^s \sim \text{Gamma}(0.001, 0.001), \quad (9.6)$$

$$\sigma_{kj}^s = \frac{1}{\sqrt{\tau_{kj}^s}}. \quad (9.7)$$

And $\mathbf{R}_k$ is assigned a Lewandowski–Kurowicka–Joe (LKJ) prior with shape parameter one, which is uniform over valid correlation matrices. For two tracers, this prior is equivalent to assigning their correlation coefficient a *Uniform* (-1, 1) prior. For three or more tracers, the LKJ prior estimates all correlations jointly while ensuring that the resulting correlation matrix remains valid. The posterior distributions of $\boldsymbol{\mu}_k^s$ and $\boldsymbol{\Sigma}_k^s$ are estimated jointly with the remainder of the mixing model.

9.2.2 Source summary statistics

If raw observations are unavailable, source k is represented by its reported mean m_{kj} , standard deviation s_{kj} , and sample size n_k . The latent population mean is modelled as

$$\mu_{kj}^s \sim \text{Normal}\left(m_{kj}, \frac{s_{kj}^2}{n_k}\right) \quad (9.8)$$

Uncertainty in the source variance is represented by

$$X_{kj} \sim \chi^2(n_k) \quad (9.9)$$

$$(\sigma_{kj}^s)^2 = \frac{s_{kj}^2(n_k - 1)}{X_{kj}} \quad (9.10)$$

Because summary statistics contain no cross-tracer covariance information, the source covariance matrix is diagonal:

$$\boldsymbol{\Sigma}_k^s = \text{diag}\left((\sigma_{k1}^s)^2, \dots, (\sigma_{kJ}^s)^2\right) \quad (9.11)$$

Larger n_k values reduce uncertainty in $\boldsymbol{\mu}_k^s$. Setting n_k to an arbitrarily large number, such as 1000, keeps the source mean close to its reported value and effectively turns off source-mean fitting, while retaining the reported within-source variance. This approximation can aid convergence in poorly resolved systems but is not a fully hierarchical treatment of source means.

9.2.3 Factor-stratified source distributions

Source signatures may vary among levels of one categorical factor, such as region, habitat, or season. If g_i identifies the source-factor level for mixture observation i , the applicable source parameters are

$$\boldsymbol{\mu}_k^s(g_i), \quad \boldsymbol{\Sigma}_k^s(g_i) \quad (9.12)$$

Each mixture observation is matched to the source distribution for its factor level. The level-specific source distributions are estimated separately unless another hierarchical relationship is explicitly introduced.

9.3 Baseline source composition

The baseline source composition is assigned a Dirichlet prior:

$$\mathbf{p}_0 \sim \text{Dirichlet}(\alpha_1, \dots, \alpha_K) \quad (9.13)$$

Its prior mean for source k is

$$E(p_{0k}) = \frac{\alpha_k}{\sum_{k=1}^K \alpha_k} \quad (9.14)$$

A vector of ones assigns equal prior means to all sources. Larger source-specific values express prior evidence favoring particular sources. Multiplying all α values by the same constant preserves the prior mean but concentrates the prior more strongly around that mean.

9.4 Isometric log-ratio representation

A K -source composition has $(K - 1)$ independent dimensions. MixSIAPy therefore maps source proportions from the simplex to $(K - 1)$ unconstrained isometric log-ratio coordinates. Under the sequential binary partition used by MixSIAR, coordinate r is

$$\eta_r = \sqrt{\frac{r}{r+1}} \log \left[\frac{(\prod_{k=1}^r p_k)^{1/r}}{p_{r+1}} \right], \quad r = 1, \dots, K - 1 \quad (9.15)$$

In matrix notation,

$$\boldsymbol{\eta} = \mathbf{V}^T \log(\mathbf{p}) \quad (9.16)$$

where $\mathbf{V}$ is a $K \times (K - 1)$ orthonormal ILR basis. The inverse transformation is

$$p_k = \frac{\exp(\mathbf{v}_k^T \boldsymbol{\eta})}{\sum_{k=1}^K \exp(\mathbf{v}_k^T \boldsymbol{\eta})} \quad (9.17)$$

The inverse transformation guarantees positive proportions that sum to one. ILR coefficient values depend on source ordering and the selected basis, whereas reconstructed source proportions do not. Effects should therefore be interpreted primarily through predicted compositions and compositional contrasts.

9.5 Compositional predictor

For observation i , the general predictor is

$$\boldsymbol{\eta}_i = \boldsymbol{\eta}_0 + \mathbf{X}_i \mathbf{B} + \mathbf{u}_i \quad (9.18)$$

$$\mathbf{p}_i = \text{ILR}^{-1}(\boldsymbol{\eta}_i) \quad (9.19)$$

Here $\boldsymbol{\eta}_0$ is the ILR transformation of the baseline composition defined in Equation (9.13), $\mathbf{X}_i$ is the fixed-effect design row, $\mathbf{B}$ is the regression coefficient matrix, and $\mathbf{u}_i$

contains random or nested grouping effects. If $\mathbf{X}$ contains P terms, $\mathbf{B}$ has dimensions $P \times (K - 1)$. Every term can therefore modify the complete composition.

9.5.1 Categorical fixed effects

For a fixed categorical factor, the first level is the reference and its ILR offset is fixed at zero:

$$\mathbf{a}_1 = \mathbf{0} \quad (9.20)$$

Each remaining level receives an independent coefficient vector:

$$a_{lr} \sim \text{Normal}(0, 1), \quad l = 2, \dots, L \quad (9.21)$$

With one fixed factor, the observation-level predictor and fitted level composition are

$$\boldsymbol{\eta}_i = \boldsymbol{\eta}_0 + \mathbf{a}_{l_i} \quad (9.22)$$

$$\mathbf{p}_l = \text{ILR}^{-1}(\boldsymbol{\eta}_0 + \mathbf{a}_l) \quad (9.23)$$

The non-reference levels are not partially pooled. Their posterior compositions quantify contrasts with the reference composition.

9.5.2 Categorical random effects

For random factor m , the level-specific ILR offsets share a common standard deviation:

$$a_{mlr} \sim \text{Normal}(0, \gamma_m^2) \quad (9.24)$$

$$\gamma_m \sim \text{Uniform}(0, 20) \quad (9.25)$$

The shared γ_m measures compositional variation among factor levels and induces partial pooling. Weakly informed levels are consequently shrunk toward the composition predicted by the remaining model terms.

With `noncentered=True`, the same prior is written as

$$z_{mlr} \sim \text{Normal}(0, 1) \quad (9.26)$$

$$a_{mlr} = \gamma_m z_{mlr} \quad (9.27)$$

The centred and non-centred forms imply the same random-effect distribution. The non-centred form can reduce sampling difficulties when γ_m is small or weakly identified.

9.5.3 Nested random effects

If lower-level group l belongs to higher-level group $g(l)$, both offsets enter the predictor:

$$\boldsymbol{\eta}_i = \boldsymbol{\eta}_0 + \mathbf{a}_{g(i)}^G + \mathbf{a}_{l(i)}^L \quad (9.28)$$

For example, with packs nested within regions,

$$\boldsymbol{\eta}_i = \boldsymbol{\eta}_0 + \mathbf{a}_{\text{region}(i)}^{\text{region}} + \mathbf{a}_{\text{pack}(i)}^{\text{pack}} \quad (9.29)$$

The higher-level effect represents variation among regions, whereas the lower-level effect represents additional variation among packs within their assigned regions.

9.5.4 Conventional continuous effect

A numerical covariate x is centered and divided by its sample standard deviation:

$$x_i^* = \frac{x_i - \bar{x}}{s_x} \quad (9.30)$$

The predictor and coefficient prior are

$$\boldsymbol{\eta}_i = \boldsymbol{\eta}_0 + x_i^* \mathbf{b} \quad (9.31)$$

$$b_r \sim \text{Normal}(0, 1000) \quad (9.32)$$

The baseline composition corresponds to the mean observed value of x . The coefficient vector describes the common compositional association with a one-standard-deviation increase in the covariate.

9.5.5 Formula-based compositional regression

The formula interface constructs X from multiple categorical and numerical predictors. Its regression coefficients receive the prior

$$B_{pr} \sim \text{Normal}(0, 6.25) \quad (9.33)$$

For example, the formula

species * biomass + species * diversity

expands to

species + biomass + diversity + species:biomass + species:diversity

The corresponding predictor can be written as

$$\boldsymbol{\eta}_i = \boldsymbol{\eta}_0 + \mathbf{B}^s s_i + \mathbf{b}^b x_i + \mathbf{b}^d d_i + \mathbf{B}^{sb}(s_i x_i) + \mathbf{B}^{sd}(s_i d_i) + \mathbf{u}_i \quad (9.34)$$

In this compact expression, s_i , x_i , and d_i denote the encoded species, standardized biomass, and encoded diversity predictors, respectively. Categorical variables are encoded using treatment contrasts and numerical variables are standardized. The species terms describe baseline differences among species. The biomass coefficient is the association for the reference species, and the species-by-biomass terms describe how that association differs among the other species. The diversity term is the planting-diversity contrast for the reference species, while the species-by-diversity terms allow that contrast to differ among species.

9.6 Expected tracer composition of a mixture

For source k and tracer j , define the discrimination-adjusted source mean as

$$\theta_{kj} = \mu_{kj}^s + \lambda_{kj} \quad (9.35)$$

9.6.1 Concentration-independent mixing

When tracer concentrations are treated as equal across sources, the expected mixture value is

$$\mu_{ij}^m = \sum_{k=1}^K p_{ik} \theta_{kj} \quad (9.36)$$

The expected tracer value is therefore the source-proportion-weighted mean of the discrimination-adjusted source signatures.

9.6.2 Concentration-dependent mixing

When concentrations differ among sources, the source proportions are converted into tracer-specific weights:

$$w_{ikj} = \frac{p_{ik} q_{kj}}{\sum_{h=1}^K p_{ih} q_{hj}} \quad (9.37)$$

The expected mixture value becomes

$$\mu_{ij}^m = \sum_{k=1}^K w_{ikj} \theta_{kj} \quad (9.38)$$

or, equivalently,

$$\mu_{ij}^m = \frac{\sum_{k=1}^K p_{ik} q_{kj} (\mu_{kj}^s + \lambda_{kj})}{\sum_{k=1}^K p_{ik} q_{kj}} \quad (9.39)$$

A source with a greater concentration of tracer j contributes more strongly to the expected value of that tracer. Setting every q_{kj} to one returns the concentration-independent expression.

9.7 Propagation of source and discrimination uncertainty

The discrimination covariance matrix for source k is diagonal because the input provides tracer-specific variances rather than cross-tracer covariances:

$$\Sigma_k^c = \text{diag}(\omega_{k1}^2, \dots, \omega_{kJ}^2) \quad (9.40)$$

For raw multi-tracer source data, the process covariance for observation i is

$$\mathbf{\Omega}_i = \sum_{k=1}^K p_{ik}^2 (\mathbf{\Sigma}_k^s + \mathbf{\Sigma}_k^e) \quad (9.41)$$

For source summary data, covariance is unavailable and the process variance for tracer j is

$$(\omega_{ij}^p)^2 = \sum_{k=1}^K p_{ik}^2 [(\sigma_{kj}^s)^2 + \omega_{kj}^2] \quad (9.42)$$

The squared proportions arise from the variance of a weighted sum. To preserve MixSIAR compatibility, the process component uses the squared dietary proportions p_{ik}^2 even when concentration dependence is included in the mixture mean.

9.8 Mixture-error models

MixSIAPy implements combined process and residual error, residual-only error, and process-only error. At least one error component must be included.

9.8.1 Combined process and residual error

For each tracer, the process variance is multiplied by an estimated residual parameter:

$$e_j \sim \text{Uniform}(0, 20) \quad (9.43)$$

For raw multi-tracer source data, define the scaling matrix

$$E_{jh} = \sqrt{e_j e_h} \quad (9.44)$$

The total mixture covariance is

$$\mathbf{\Sigma}_i^m = \mathbf{E} \odot \mathbf{\Omega}_i \quad (9.45)$$

where $\odot$ denotes element-by-element multiplication. The mixture likelihood is

$$\mathbf{y}_i^m \sim \text{Normal}(\boldsymbol{\mu}_i^m, \mathbf{\Sigma}_i^m) \quad (9.46)$$

For source summary data, the covariance is diagonal and

$$y_{ij}^m \sim \text{Normal}(\mu_{ij}^m, e_j (\omega_{ij}^p)^2) \quad (9.47)$$

This model propagates source and discrimination uncertainty while estimating an additional multiplier for variation among mixture observations.

9.8.2 Residual-only error

The residual-only model estimates a covariance matrix shared by the mixture observations:

$$\mathbf{y}_i^m \sim \text{Normal}(\boldsymbol{\mu}_i^m, \mathbf{\Sigma}^r) \quad (9.48)$$

For multiple tracers, the MixSIAR-compatible prior is placed on the residual precision matrix:

$$\mathbf{T}^r \sim \text{Wishart}(\mathbf{I}, J + 1) \quad (9.49)$$

$$\mathbf{\Sigma}^r = (\mathbf{T}^r)^{-1} \quad (9.50)$$

MixSIAPy evaluates this prior through a Bartlett decomposition, which preserves positive definiteness during sampling. For one tracer,

$$\tau^r \sim \text{Gamma}(0.001, 0.001) \quad (9.51)$$

$$y_i^m \sim \text{Normal}\left(\mu_i^m, \frac{1}{\tau^r}\right) \quad (9.52)$$

This formulation estimates unexplained variability among mixture observations without propagating source variance into the mixture likelihood.

9.8.3 Process-only error

The process-only model attributes mixture variability entirely to uncertainty in `source` and discrimination signatures:

$$y_{ij}^m \sim \text{Normal}\left(\mu_{ij}^m, (\omega_{ij}^p)^2\right) \quad (9.53)$$

No additional residual parameter is estimated. For compatibility with the corresponding MixSIAR and MixSIR formulation, the process-only likelihood uses tracer-specific marginal variances even when raw source data contain cross-tracer covariance. It is also the supported error formulation for a single mixture observation because residual variation cannot be estimated from one observation.

9.9 Joint posterior distribution

Let Θ contain the unknown source parameters, baseline composition, regression coefficients, random-effect parameters, and error components. Let $\mathbf{Y}$ denote mixture observations and $\mathbf{S}$ the raw source observations. The joint posterior is

$$p(\Theta \mid \mathbf{Y}, \mathbf{S}) \propto p(\mathbf{Y} \mid \Theta)p(\mathbf{S} \mid \Theta)p(\Theta) \quad (9.54)$$

The corresponding log-posterior is

$$\log p(\Theta \mid \mathbf{Y}, \mathbf{S}) = \log L_{\text{mix}} + \log L_{\text{source}} + \log p(\Theta) + C \quad (9.55)$$

Here L_{mix} and L_{source} denote the mixture-data and raw-source likelihoods, respectively. When source summaries are used, the raw-source likelihood is replaced by the source-parameter distributions defined from their reported means, variances, and sample sizes. The model is hierarchical because source parameters, source proportions, covariate effects, random-effect variation, and error parameters are estimated jointly. Uncertainty at each level is therefore propagated into posterior source proportions and predictions.

9.10 Posterior predictions and contrasts

For posterior draw d , the predicted linear predictor and composition at a new design row $\mathbf{X}^*$ are

$$\boldsymbol{\eta}_*^{(d)} = \boldsymbol{\eta}_0^{(d)} + \mathbf{X}_* \mathbf{B}^{(d)} + \mathbf{u}_*^{(d)} \quad (9.56)$$

$$\mathbf{p}_*^{(d)} = \text{ILR}^{-1}(\boldsymbol{\eta}_*^{(d)}) \quad (9.57)$$

Population-level predictions set $\mathbf{u}^*$ to zero and describe the expected composition for an average random-effect level. Group-specific predictions include the posterior offset for the selected group.

For predictor settings A and B , a source-specific contrast is calculated within every posterior draw:

$$\Delta_k^{(d)} = p_{Ak}^{(d)} - p_{Bk}^{(d)} \quad (9.58)$$

Posterior medians and credible intervals for Δ_k quantify differences in source contribution. Continuous prediction curves are generated by evaluating the predictor across a covariate grid. Because the inverse ILR transformation is nonlinear, an ILR coefficient is not a constant change in one source proportion. Predicted compositions and their contrasts provide the clearest physical interpretation.

9.11 Identifiability and interpretation

Bayesian computation quantifies uncertainty but cannot create information absent from the tracer data. Source proportions may remain weakly identified when:

- candidate sources have strongly overlapping tracer signatures;
- the number of sources is large relative to the number of informative tracers;
- source or discrimination variances are large;
- relevant sources are omitted from the mixing system;
- covariate combinations are sparsely represented; or
- complex interactions or random effects are supported by few observations.

Under these conditions, posterior estimates may depend strongly on the Dirichlet prior or hierarchical shrinkage assumptions. Markov chain convergence does not by itself establish that the ecological source proportions are well resolved. Model assessment should therefore combine posterior diagnostics with source separation, prior sensitivity, posterior uncertainty, covariate coverage, and the ecological plausibility of fitted compositions.

10 Diagnosing a fitted model

Posterior results should not be interpreted until sampling quality has been assessed. Model assessment should begin with numerical diagnostics and then proceed to graphical checks and, when necessary, investigation of the model specification.

10.1 Compact diagnostic report

MixSIAPy provides a compact diagnostic summary:

```
from mixsiapy import diagnostics

table, report = diagnostics(
    fit,
    rhat_limit=1.05,
    ess_limit=400,
)
print(report)
```

The function evaluates rank-normalized $\hat{R}$, bulk effective sample size, tail effective sample size, and divergent transitions. The returned table contains parameter-level diagnostics, whereas report summarizes the number of failures and gives an overall convergence status.

The default ESS threshold is 400. A different threshold may be adopted for a particular study, but it should be specified before interpreting the results and applied consistently to all prespecified parameters.

Maximum-tree-depth warnings and the Bayesian fraction of missing information are not currently included in the compact report. They should be inspected separately from the sampler statistics when relevant.

10.2 $\hat{R}$ and effective sample size

Rank-normalized $\hat{R}$ compares variation within chains with variation among chains after rank normalization and chain splitting. Values close to one indicate that the chains sampled similar distributions. Values above the chosen threshold may indicate incomplete mixing, non-stationarity, or separate posterior modes.

$\hat{R}$ is a sampling diagnostic rather than evidence that the statistical model is scientifically correct. A model can have satisfactory $\hat{R}$ values while its parameters remain weakly identified by the tracer data.

MCMC draws are usually autocorrelated. Effective sample size estimates how many independent draws would contain approximately the same amount of information as the correlated posterior draws.

Bulk ESS measures sampling efficiency in the central part of the posterior distribution. Tail ESS measures sampling efficiency in the distribution tails and is particularly relevant to credible-interval limits. The minimum ESS among the parameters required for scientific interpretation provides a conservative assessment of posterior precision.

10.3 Hamiltonian Monte Carlo diagnostics

A divergent transition indicates that the numerical Hamiltonian trajectory could not accurately explore part of the posterior geometry. Divergences may bias posterior estimates even when $\hat{R}$ and ESS appear acceptable.

When divergences occur, first identify the affected parameters and examine their joint posterior distributions. Possible responses include:

1. checking data scaling and model inputs;
2. increasing `target_accept` when divergences are few;
3. using `noncentered=True` for hierarchical random effects;
4. applying stronger scientifically justified priors;
5. combining sources that cannot be distinguished;
6. simplifying unsupported interactions or hierarchical structures.

A maximum-tree-depth warning means that NUTS reached the permitted trajectory length before meeting its termination criterion. Occasional warnings may have little practical effect, but persistent warnings indicate inefficient exploration or difficult posterior geometry. Increasing the maximum tree depth may increase computation without addressing the underlying problem.

The Bayesian fraction of missing information assesses movement through the posterior energy distribution. It can be calculated with ArviZ:

```
import arviz as az
print(az.bfmi(fit))
```

Values below approximately 0.3 warrant further investigation. BFMI should be interpreted together with divergences, tree depth, trace plots, and effective sample sizes.

10.4 Graphical diagnostic checks

Numerical diagnostics should be supplemented with graphical checks:

```
import arviz as az
az.plot_trace(fit, var_names=["p_global"])
az.plot_rank(fit, var_names=["p_global"])
az.plot_energy(fit)
```

Trace plots should show stable and overlapping chains without persistent trends or chains confined to separate regions. Rank plots should be approximately uniform across chains. Energy plots help identify poor exploration of the Hamiltonian energy distribution.

10.5 Responding to diagnostic failures

Diagnostic problems should be addressed in the following order:

- (1) Confirm that files, units, factors, nesting, concentration columns, and priors are correct.
- (2) Plot mixtures and sources to assess whether the mixing system is geometrically plausible.
- (3) Identify the parameters associated with poor diagnostics.
- (4) Increase `target_accept` when divergences are few and geometry otherwise appears reasonable.
- (5) Try `noncentered=True` for hierarchical random effects.

- (6) Consider source aggregation when sources are not distinguishable for the scientific question.
- (7) Add scientifically justified prior information.
- (8) Increase the sampling schedule only after addressing structural problems.

A parameter should not be removed from the diagnostic assessment solely to declare the model converged. If scientifically important quantities remain unreliable after reasonable attempts to improve sampling, the analysis should be reported as inconclusive.

11 Summarizing, predicting, and interpreting posterior results

Once sampling quality is acceptable, posterior draws can be summarized as parameter distributions, predicted source compositions, and contrasts between scientifically relevant conditions.

11.1 Posterior summaries

Posterior summaries are generated with:

```
from mixsiapy import summary_stat
global_summary = summary_stat(fit, var_names=["p_global"])
print(global_summary)
```

The output includes posterior means, standard deviations, credible intervals, Monte Carlo standard errors, bulk and tail effective sample sizes, and $\hat{R}$ where these quantities are defined.

The principal posterior variables are:

Variable	Meaning
p_global	Baseline source composition
ilr_global	Baseline composition in ILR coordinates
p_ind	Observation-specific source compositions
ilr_fac1, ilr_fac2	Conventional factor offsets in ILR coordinates
p_fac1, p_fac2	Factor-level source compositions
p_both	Composition for combinations of two factors
fac1_sig, fac2_sig	Random-factor standard deviations
ilr_cont1	Conventional continuous-effect coefficient
ilr_beta	Formula-regression coefficient matrix
src_mu, src_var	Latent source means and variances
mix_mean	Expected tracer value for each mixture
resid_prop or Sigma	Residual-error parameters

11.2 Conventional continuous-effect prediction

For a model fitted using `cont_effects`, source proportions can be predicted across the continuous covariate:

```
from mixsiapy import continuous_effect_prediction,
plot_continuous_effect

prediction = continuous_effect_prediction(
    fit,
    mix,
    source_names=source["source_names"],
)
plot_continuous_effect(prediction, "continuous_effect.png")
```

The prediction grid is expressed on the original covariate scale. Values are standardized internally using the center and scale stored during model fitting.

Optional factor-level indices may be supplied when predictions should include conventional factor offsets. If no level is supplied, predictions describe the baseline factor condition.

The plotted curves show posterior medians and credible intervals for every source proportion. Because the proportions sum to one, a change in one source necessarily corresponds to changes in one or more other sources.

11.3 Formula-based prediction

Formula-based models are predicted from a data frame containing the desired predictor combinations:

```
import pandas as pd
from mixsiapy import compositional_regression_prediction

new_data = pd.DataFrame({
    "species": ["Bean", "Bean", "Pea", "Pea"],
    "biomass": [2.0, 6.0, 2.0, 6.0],
    "diversity": ["monoculture", "mixture", "monoculture",
"mixture"],
})

prediction = compositional_regression_prediction(
    fit,
    mix,
    new_data,
    source_names=source["source_names"],
)
```

The prediction data must contain every variable referenced in the fitted formula. Categorical values must match levels observed during model fitting. Numerical variables are transformed using the center and scale estimated from the training data.

The current helper combines `p_global` with the formula coefficients but does not add conventional random-factor offsets. It therefore produces population-level predictions with random deviations set to zero. Conditional predictions for a particular random-effect level require direct calculation from the corresponding posterior draws.

Predictions should normally remain within the support of the observed data. Interaction models can extrapolate strongly when a species or treatment has few observations over part of a numerical predictor range.

11.4 Interpreting contrasts and interactions

Compositional effects should be interpreted through predictions rather than isolated ILR coefficients. For a numerical predictor, a useful summary is the change in every source proportion between two observed values, such as the 10th and 90th percentiles.

For example, the posterior contrast for source k may be defined as $\Delta_k = p_k(\text{high biomass}) - p_k(\text{low biomass})$. A positive value means that the contribution of source k increases across the selected biomass range. A credible interval containing zero means that the available data do not clearly resolve the direction of the change.

For a categorical predictor, levels should be compared at a common value of the numerical covariates. Comparisons should be restricted to covariate ranges represented in all conditions being contrasted.

The formula “species * biomass + species * diversity” addresses four related questions:

- (1) Do baseline source compositions differ among species?
- (2) Is biomass associated with a redistribution among sources?
- (3) Does the biomass association differ among species?
- (4) Does the monoculture–mixture contrast differ among species after accounting for biomass?

A conventional model containing species and biomass without an interaction estimates one common biomass relationship across species. The species-by-biomass interaction instead estimates how this relationship differs among species within one joint mixing model.

A relationship with an observational predictor such as biomass is a conditional association. It does not imply that experimentally increasing biomass would cause a change in source use.

11.5 Combining sources after fitting

Posterior source proportions can be combined into broader reporting categories:

```
from mixsiapy import combine_sources

combined = combine_sources(
    fit,
    groups={
        "Terrestrial": ["Deer"],
        "Marine": ["Marine Mammals", "Salmon"],
    },
    source_names=source["source_names"],
)
```

Every original source must appear exactly once, and groups cannot overlap. The default operation combines `p_global`, although another posterior composition variable can be selected with the `variable` argument.

Post-fitting combination is appropriate when broader source categories are required for presentation. It does not reproduce the likelihood that would have resulted from combining source signatures before fitting. If scientifically indistinguishable sources affect identifiability, they should be combined before model fitting rather than only in the output.

11.6 Comparing models

Alternative fitted models can be compared using predictive information criteria:

```
from mixsiapy import compare_models

comparison = compare_models(
    [fit_a, fit_b],
    names=["process_residual", "residual_only"],
    ic="loo",
)
```

MixSIAPy uses ArviZ to calculate Pareto-smoothed importance-sampling leave-one-out cross-validation or WAIC when pointwise mixture log likelihood is available. Historical DIC output is not reproduced.

Information criteria compare expected predictive performance while accounting for model complexity. They do not determine whether source definitions, priors, covariates, or ecological assumptions are scientifically appropriate. Models with poor convergence should not be compared using these criteria.

12 Visualization, export, and reproducibility

Posterior draws, diagnostic information, summaries, and figures should be retained together so that the analysis can be inspected and reproduced.

12.1 Saving the complete analysis

A complete output bundle is created with:

```
from mixsiapy import save_results

save_results(
    fit,
    "outputs/my_analysis",
    source["source_names"],
    mix,
)
```

Depending on the fitted model, the output directory may contain:

- `posterior_complete.nc`, containing the complete ArviZ object;

- `summary_all_parameters.csv`;
- `summary_source_proportions.csv`;
- `summary_compositional_regression.csv`;
- factor-level posterior summaries;
- continuous-effect predictions;
- posterior-density and interval figures;
- trace and posterior plots;
- diagnostic reports in CSV, JSON, and text formats;
- LOO and WAIC output where these quantities can be calculated.

Runtime metadata are stored in `fit.attrs` and retained in the NetCDF file. These metadata include the requested and resolved inference backends, computational device, build time, sampling time, total time, software environment, and selected error model.

Formula-based predictions for a user-defined `new_data` table are not generated automatically by `save_results`. They should be saved separately when they form part of the reported analysis.

12.2 Reporting graphical results

Figures should emphasize source proportions and scientifically relevant contrasts rather than raw ILR coefficients.

Recommended outputs include:

- posterior distributions of the global source proportions;
- factor-level source compositions;
- prediction curves for continuous covariates;
- predicted compositions for selected formula-based scenarios;
- contrasts between treatments or predictor values;
- trace, rank, and energy plots for diagnostic review.

Each figure should state whether predictions include group-specific random effects or set their deviations to zero. Prediction grids, factor reference levels, and contrast directions should also be documented.

12.3 Reproducibility checklist

A final analysis should record:

- MixSIAPy, Python, PyMC, ArviZ, backend, and JAX versions;
- operating system and CPU or GPU device;
- archived copies or checksums of the mixture, source, and discrimination files;
- source stratification, factor, nesting, formula, concentration, prior, and error-model settings;
- run preset or custom chains, warm-up iterations, retained draws, and thinning;
- random seed, target acceptance probability, and any tree-depth setting;

- convergence thresholds and the parameters to which they were applied;
- $\hat{R}$, bulk ESS, tail ESS, divergences, tree-depth warnings, and BFMI where relevant;
- the prediction data or grid and whether random effects were included;
- complete posterior draws in NetCDF format.

Graphically configured analyses should retain the exported configuration and reproduction script. Outputs from hosted sessions should be downloaded before the session ends because remote storage may be temporary.

13 Scope, limitations, and troubleshooting

13.1 Statistical limitations

Mixing models cannot recover information that is absent from the tracer system. Strong overlap among `source` signatures can allow several `source` compositions to explain the same mixture observations. Longer sampling cannot resolve this fundamental non-identifiability.

Posterior results should therefore be evaluated using `source` separation, uncertainty intervals, prior sensitivity, predictor coverage, and ecological plausibility in addition to convergence diagnostics.

Important limitations of the current implementation include:

- at most two conventional categorical grouping factors;
- at most one source-stratification factor;
- only two-way interactions in the formula interface;
- predictor type inferred from the input-column data type;
- formula prediction restricted to population-level predictions without conventional random offsets;
- raw-source correlation priors for more than two tracers that are not algebraically identical to historical independent pairwise JAGS priors;
- no historical DIC calculation;
- backend and GPU performance that depends on model structure, posterior geometry, and hardware.

MixSIAPy remains a research software implementation. Each scientific application should therefore be checked using appropriate validation, sensitivity analysis, and posterior diagnostics.

13.2 Data and model-specification problems

(1) A formula category is treated as continuous

Numerically stored columns are interpreted as continuous predictors. Convert a numerically coded category to text before loading the data:

```
import pandas as pd

data = pd.read_csv("consumer.csv")
data["Treatment"] = data["Treatment"].astype(str)
```

(2) Prediction reports an unknown category

Formula-based prediction cannot introduce a categorical level that was absent during fitting. Check spelling, capitalization, whitespace, and the levels represented in the training data. Refit the model if the missing level should be included.

(3) Nesting validation fails

Each lower-level group must belong to exactly one higher-level group. Inspect the factor relationship with:

```
import pandas as pd

data = pd.read_csv("consumer.csv")

print(
    pd.crosstab(
        data["Pack"],
        data["Region"],
    )
)
data["Treatment"] = data["Treatment"].astype(str)
```

A row containing observations in more than one region indicates that the proposed nesting relationship is invalid or that the group labels require correction.

13.3 Installation and environment problems

(1) Import fails in Spyder

Spyder must use the Python interpreter in which MixSIAPy was installed:

```
import sys
import mixsiapy

print(sys.executable)
print(mixsiapy.__file__)
```

Restart the Spyder kernel after installing or updating the package.

(2) A GPU is not detected

Check the devices visible to JAX in the same environment used to run MixSIAPy:

```
import jax

print(jax.devices())
```

A CPU-only JAX installation cannot use a CUDA GPU. WSL2 or Linux is the recommended CUDA environment on Windows.

(3) PyTensor compilation fails on Windows

MixSIAPy can retry using PyTensor’s pure-Python linker for known incompatible compiler environments. This route is statistically equivalent but slower. A clean Python 3.11 or 3.12 environment with compatible package versions is preferable for production analyses.

13.4 Performance and output problems

(1) The model is slow

Inspect `fit.attrs` to determine whether time is being spent in model construction, compilation, warm-up, or retained sampling. The `test` schedule should be used only to validate the data and model configuration.

Nutpie may reduce CPU overhead for some models. JAX compilation can make short analyses slower even when retained sampling is efficient. A GPU is most likely to help when sufficient parallel work is available. Statistical structure should be simplified only when scientifically justified.

(2) A NetCDF file cannot be written

Confirm that the destination exists, is writable, and is not open in another application. Cloud-synchronized directories should be avoided during long analyses. `save_results` writes to a temporary file before replacing the final NetCDF output, reducing the risk of a partial file if writing is interrupted.

Appendix A: Public API

A.1 Data loading

```
load_mix_data(  
    filename,  
    iso_names,  
    factors=None,  
    fac_random=None,  
    fac_nested=None,  
    cont_effects=None,  
    composition_formula=None,  
)  
  
load_source_data(  
    filename,  
    source_factors=None,  
    conc_dep=False,  
    data_type="raw",  
    mix=None,  
)  
  
load_discr_data(filename, mix=None)
```

A.2 Model construction and sampling

```
build_model(  
    mix,  
    source,  
    discr,  
    alpha_prior=1,  
    process_err=True,  
    resid_err=True,  
    noncentered=False,  
)  
  
run_model(  
    run,  
    mix,  
    source,  
    discr,  
    model_filename=None,  
    alpha_prior=1,  
    process_err=True,  
    resid_err=True,  
    random_seed=None,  
    backend="pymc",  
    device="auto",  
    noncentered=False,  
    **sample_kwargs,  
)
```

`model_filename` is retained for interface compatibility and is not used to generate a JAGS model in the native Python pathway.

A.3 Results and utilities

```
summary_stat(result, var_names=None, hdi_prob=0.95)

diagnostics(result, var_names=None, rhat_limit=1.05,
ess_limit=400)

save_diagnostics(result, directory, var_names=None)

save_results(fit, output, source_names, mix)

compare_models(models, names=None, ic="loo")

combine_sources(result, groups, source_names=None,
variable="p_global")

continuous_effect_prediction(
    result, mix, source_names=None, values=None,
    factor1_level=None, factor2_level=None,
)

compositional_regression_prediction(result, mix, new_data,
source_names=None)

plot_continuous_effect(prediction, filename=None, hdi_prob=0.95)

calc_area(source, mix, discr)

backend_status()

benchmark_summary(results)

resolve_backend(backend="pymc", device="auto")

resolve_device(backend, device="auto")

get_resource_path(collection=None, *parts, must_exist=True)

list_resources(collection)
```

Appendix B: Recommended workflow checklist

Before sampling:

- confirm tracer names and units;
- confirm source and discrimination row names;
- inspect source ordering before defining an informative prior;
- plot the isotope space and prior;
- verify factor levels and nesting;
- confirm whether concentration dependence is required;
- document the error-model rationale;
- build the model or run the test preset.

Before interpreting results:

- inspect $\hat{R}$, bulk ESS, tail ESS, divergences, tree depth, and energy behaviour;
- confirm that predictions stay within observed covariate support;
- interpret formula effects using proportion-scale predictions;
- distinguish associations from causal effects;
- examine prior sensitivity and source identifiability;
- retain complete posterior draws and the resolved configuration.

References

- Abril-Pla, O., Andreani, V., Carroll, C., Dong, L., Fonnesbeck, C. J., Kochurov, M., Kumar, R., Lao, J., Luhmann, C. C., Martin, O. A., Osthege, M., Vieira, R., Wiecki, T., and Zinkov, R.: PyMC: a modern and comprehensive probabilistic programming framework in Python, *PeerJ Computer Science*, 9, e1516, 2023.
- Aitchison, J.: *The Statistical Analysis of Compositional Data*, Chapman and Hall, London, 1986.
- Betancourt, M.: A conceptual introduction to Hamiltonian Monte Carlo, *arXiv:1701.02434*, 2017.
- Egozcue, J. J., Pawlowsky-Glahn, V., Mateu-Figueras, G., and Barcelo-Vidal, C.: Isometric logratio transformations for compositional data analysis, *Mathematical Geology*, 35, 279-300, 2003.
- Hoffman, M. D. and Gelman, A.: The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo, *Journal of Machine Learning Research*, 15, 1593-1623, 2014.
- Kumar, R., Carroll, C., Hartikainen, A., and Martin, O.: ArviZ a unified library for exploratory analysis of Bayesian models in Python, *Journal of Open Source Software*, 4, 1143, 2019.
- Moore, J. W. and Semmens, B. X.: Incorporating uncertainty and prior information into stable isotope mixing models, *Ecology Letters*, 11, 470-480, 2008.
- Parnell, A. C., Inger, R., Bearhop, S., and Jackson, A. L.: Source partitioning using stable isotopes: coping with too much variation, *PLOS ONE*, 5, e9672, 2010.
- Phillips, D. L., Inger, R., Bearhop, S., Jackson, A. L., Moore, J. W., Parnell, A. C., Semmens, B. X., and Ward, E. J.: Best practices for use of stable isotope mixing models in food-web studies, *Canadian Journal of Zoology*, 92, 823-835, 2014.
- Stock, B. C., Jackson, A. L., Ward, E. J., Parnell, A. C., Phillips, D. L., and Semmens, B. X.: Analyzing mixing systems using a new generation of Bayesian tracer mixing models, *PeerJ*, 6, e5096, 2018.
- Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., and Buerkner, P. C.: Rank-normalization, folding, and localization: an improved R-hat for assessing convergence of MCMC, *Bayesian Analysis*, 16, 667-718, 2021.