presamples

Release 0.2.6

Mar 10, 2021

Contents

Presamples is used to write, load, manage and verify *presample arrays*.

Presample arrays refer to arrays of values that specific parameters or matrix elements can take on. The presamples package allows these arrays to be generated *ahead* of their use in a particular model. This is useful if:

- Generating these values is computationally expensive and there is no need to recalculate them with each model run;
- We want to reuse the *same* values every time a model is solved.

Presamples was initially built specifically for parameters and matrix elements used in life cycle assessment (LCA), and hence has many methods specifically geared at making the integration of presamples in LCA models easy. However, it can be used in any other type of model.

Presample's source code is hosted on github.

Contents:

CHAPTER 1

Installing

1.1 Installing with pip

The presamples package is hosted on Pypi and can be installed using pip:

```
pip install presamples
```

1.2 Installing with conda

The presamples package is hosted on a conda channel and can be installed using conda:

```
conda install --channel pascallesage presamples
```

1.3 Install from github

The latest version of the presamples package is hosted on github and can be installed using pip:

```
https://github.com/PascalLesage/presamples/archive/master.zip
git clone https://github.com/PascalLesage/presamples.git
cd presamples
python setup.py install
```

Note: On some systems you may need to use `sudo python setup.py install` to install presamples system-wide.

1.4 Brightway2 dependencies and integration

Presamples was initially developed to work with the Brightway2 LCA framework and hence inherits some requirements from that framework. While it is possible to use with presamples without ever using Brightway, these dependencies still get installed. They won't be a bother, though.

Note: If you *do* want to use presamples with Brightway2, then great! You'll find many classes and methods that will take your LCA game to the next level. Make sure you have Brightway2 installed in your environment, see **'here<https://docs.brightwaylca.dev/installation.html>'_** for more details.

CHAPTER 2

Quickstart

This section provides a brief overview of the use of presamples for named parameters via a simple example.

Note: While presamples are application-agnostic, the package was developed in the context of the Brightway2 LCA framework. If you are interested in using presamples for LCA and are new to presamples, you should move on to *Using presamples with brightway2* after reading this section.

Note: This section really only provides a brief overview. For a more in-depth presentation of the API, read the *Technical reference* section. Also, the examples in this section are minimalistic. For some more concrete examples, refer to the *Examples* section.

2.1 The objectives of presamples

Presamples was written to meet two specific needs: - the need to store and provide access to arrays of data that are inputs to a model; - the need to have a flexible data hierarchy so that some arrays can be replaced by other arrays when running a model.

Presamples is used to write, load, manage and verify *presample arrays*, which are simply arrays of values specific parameters can take. These are stored in *presample packages*, which are based on the datapackage standard by the Open Knowledge Foundation.

These presample arrays can be based on any source:

- Measured data;
- Time series data from a statistical agency;
- Array of random values generated from a given distribution;
- The output from a MonteCarlo Simulation from a model;
- A hat.

Presamples allows these arrays to be generated *ahead* of their use in a particular model. This is useful if:

- Generating these values is computationally expensive and there is no need to recalculate them with each model run;
- We want to reuse the *same* values every time a model is solved.

Also, when multiple presample packages are accessed for a single parameter, only the last values are used. This allows a baseline model to be modulated with different input data (scenarios) without actually making changes to the baseline data.

2.2 Simple example: Fertilizer inputs to cereal production in Canada

For illustration, let's suppose you have a simple model that calculates the amount of fertilizer used to grow 1 kg of cereals in Canada. The model has three inputs:

- Total amount of fertilizers used per km2 for a given year
- The total land under cultivation for the same year
- The total output of cereals for the same year

The model is simply:

```
>>> def fert_per_kg(fert_kg_per_km2, land_ha, cereal_t):
...     return fert_kg_per_km2 * (land_ha / 100) / (cereal_t / 1000)
```

The following data, stored as arrays, were collected for years 2003-2015 from the World Bank website:

```
>>> import numpy as np

# Cereal production, in metric tons
>>> cereal_production_array = np.array(
...     [
...         49197200, 50778200, 50962400, 48577300, 48005300, 56030400,
...         49691900, 45793400, 47667200, 51799100, 66405701, 51535801, 53361100
...     ], dtype=np.int64
... )
```

(continues on next page)

(continued from previous page)

```
# Fertilizer consumption, in kg/km^2
>>> fertilizer_consumption_array = np.array(
...     [57.63016664,   58.92761065,   54.63277483,   61.82127866,   46.99494591,
...      68.60414475,   63.96407104,   62.20875736,   62.26266793,   77.0963275 ,
...      94.15242211,   96.13617882,   115.82229301
...     ], dtype=np.float64
... )
# Land used for cereal production, in hectares
>>> land_for_cereals_array = np.array(
...     [
...         17833000, 16161700, 15846800, 15946100, 16145100, 16519700,
...         15060300, 13156000, 13536700, 14981496, 15924684, 14023084, 14581100
...     ], dtype=np.int64
... )
```

2.3 Creating presample packages for data inputs

To create a presamples package for the input data described above:

```
>>> import presamples

# Stack arrays of data.
# The number of columns equals the number of observations
# The number of rows equals the number of parameters
>>> ag_sample_arr = np.stack(
...     [
...         cereal_production_array,
...         fertilizer_consumption_array,
...         land_for_cereals_array
...     ], axis=0
... )

# Create a list of your parameter names
>>> ag_names = ['cereal production [t]', 'fert consumption [kg/km2]', 'land [ha]']

>>> pp_id, pp_path = presamples.create_presamples_package(
...     parameter_data = [(ag_sample_arr, ag_names, "Agri baseline data")],
...     name="Baseline agri data - presample package"
... )
```

This function does several things:

1) It stores the samples to a numpy array and the parameter names as a json file to disk, at the location `pp_path`
2) It generates a file `datapackage.json` that contains metadata on the presamples package.

```
>>> import os
>>> os.listdir(pp_path)
['datapackage.json',
 'dbc8f4abb93540f9a1ab040b8923672f.0.names.json',
 'dbc8f4abb93540f9a1ab040b8923672f.0.samples.npy']
```

The datapackage has the following structure:

```
>>> import json
>>> with open(pp_path/'datapackage.json', 'rb') as f:
...     datapackage = json.load(f)
>>> print(json.dumps(datapackage, indent=4))
{
    "name": "Baseline agri data - presample package",
    "id": "dbc8f4abb93540f9a1ab040b8923672f",
    "profile": "data-package",
    "seed": null,
    "resources": [
        {
            "samples": {
                "filepath": "dbc8f4abb93540f9a1ab040b8923672f.0.samples.npy",
                "md5": "58978441f250cadca1d5829110d23942",
                "shape": [
                    3,
                    13
                ],
                "dtype": "float64",
                "format": "npy",
                "mediatype": "application/octet-stream"
            },
            "names": {
                "filepath": "dbc8f4abb93540f9a1ab040b8923672f.0.names.json",
                "md5": "c2202d5f8fd5fd9eb11e3cd528b6b14d",
                "format": "json",
                "mediatype": "application/json"
            },
            "profile": "data-resource",
            "label": "Agri baseline data",
            "index": 0
        }
    ],
    "ncols": 13
}
```

See the *Technical reference* for more detail and for a list of other arguments.

2.4 Direct interface to presamples package

To interact directly with a single presamples package :

```
>>> package = presamples.PresamplesPackage(pp_path)
```

The entire content of the `datapackage.json` file is returned by `package.metadata`.

The package can also be used to directly return several properties contained in the datapackage, for example:

```
>>> package.name  # Name passed as optional argument in ``create_presamples_package``
'Agri example - baseline data'

>>> package.ncols # Number of columns, i.e. number of observations stored in the␣
↪presamples array
13
```

(continues on next page)

(continued from previous page)

```
>>> package.id
'2a31aa637f564618bf5e606333ffb7fc'
```

Accessing the package's `resources` provides metadata on the stored data and filepaths to access it. `packages.resources` returns a list with as many resources as were passed in `create_presamples_package`. In our simple example, only one set of parameter data was passed, so `packages.resources` only contains one element.

```
>>> package.resources # List of resources, in simple example there is one
[{'samples': {'filepath': 'dbc8f4abb93540f9a1ab040b8923672f.0.samples.npy',
   'md5': '58978441f250cadca1d5829110d23942',
   'shape': [3, 13],
   'dtype': 'float64',
   'format': 'npy',
   'mediatype': 'application/octet-stream'},
  'names': {'filepath': 'dbc8f4abb93540f9a1ab040b8923672f.0.names.json',
   'md5': 'c2202d5f8fd5fd9eb11e3cd528b6b14d',
   'format': 'json',
   'mediatype': 'application/json'},
  'profile': 'data-resource',
  'label': 'Agri baseline data',
  'index': 0}]
```

The PresamplesPackage also provides a `ParametersMapping` interface to access named parameter data:

```
>>> package.parameters
<presamples.package_interface.ParametersMapping at 0x2136d4de5f8>

>>> list(package.parameters.keys())
['cereal production [t]', 'fert consumption [kg/km2]', 'land [ha]']

>>> list(package.parameters.values()) # Note that the arrays are memory mapped
 [memmap([49197200., 50778200., 50962400., 48577300., 48005300., 56030400.,
          49691900., 45793400., 47667200., 51799100., 66405701., 51535801.,
          53361100.]),
 memmap([ 57.63016664,  58.92761065,  54.63277483,  61.82127866,
          46.99494591,  68.60414475,  63.96407104,  62.20875736,
          62.26266793,  77.0963275 ,  94.15242211,  96.13617882,
          115.82229301]),
 memmap([17833000., 16161700., 15846800., 15946100., 16145100., 16519700.,
         15060300., 13156000., 13536700., 14981496., 15924684., 14023084.,
         14581100.])]

 >>> {k:v for k, v in package.parameters.items()}
 {'cereal production [t]': memmap([49197200., 50778200., 50962400., 48577300.,␣
↪48005300., 56030400.,
         49691900., 45793400., 47667200., 51799100., 66405701., 51535801.,
         53361100.]),
 'fert consumption [kg/km2]': memmap([ 57.63016664,  58.92761065,  54.63277483,  61.
↪82127866,
          46.99494591,  68.60414475,  63.96407104,  62.20875736,
          62.26266793,  77.0963275 ,  94.15242211,  96.13617882,
         115.82229301]),
 'land [ha]': memmap([17833000., 16161700., 15846800., 15946100., 16145100., 16519700.
↪,
         15060300., 13156000., 13536700., 14981496., 15924684., 14023084.,
         14581100.])}
```

You can also get a specific array directly from the parameter name:

```
>>> package.parameters['land [ha]']
memmap([17833000., 16161700., 15846800., 15946100., 16145100., 16519700.,
        15060300., 13156000., 13536700., 14981496., 15924684., 14023084.,
        14581100.])
```

Note that the values from **all** columns are returned, which makes the `PresamplePackages` a useful interface for models that accept arrays as inputs:

```
>>> fert_per_kg(
        fert_kg_per_km2=package.parameters['fert consumption [kg/km2]'],
        land_ha=package.parameters['land [ha]'],
        cereal_t=package.parameters['cereal production [t]']
    )
array([208.89781567, 187.55496749, 169.88106058, 202.93599925,
       158.05298607, 202.26874876, 193.85817388, 178.71973075,
       176.8157259 , 222.98038423, 225.78597129, 261.59013441,
       316.48831014])
```

2.5 Loading packages for use one column at a time

Presamples also allows accessing parameter data one observation at a time. This is useful to feed data from the presample arrays in Monte Carlo Simulations.

This is done via the `PackagesDataLoader`. A `PackagesDataLoader` is instantiated with a list of presamples package paths. In our simple example, we just have one path:

```
>>> ag_loader = presamples.PackagesDataLoader([pp_path])
```

One of the important things the `PackagesDataLoader` does in create an `Indexer` for each presamples package. This indexer simply returns an integer representing the column number of the presamples array from which data should be taken. By default, the `Indexer` returns indices at random. An `Indexer` can also be seeded for reproducibility (see *Creating presample packages with seeded indexers*), and can also return values sequentially (see *Creating presample packages with sequential indexers*).

The `PackagesDataLoader` has an interface to access named parameters, one observation at a time:

```
>>> ag_loader.parameters['land [ha]']
17833000.0
```

It is also possible to return values for all parameters using the `consolidated_arrays` property:

```
>>> ag_loader.parameters.consolidated_array
array([4.91972000e+07, 5.76301666e+01, 1.78330000e+07])
```

The order of the values is identical to the order of names: .. code-block:: python

```
>>> ag_loader.parameters.names
['cereal production [t]', 'fert consumption [kg/km2]', 'land [ha]']
```

The array is considered "consolidated" because it uses values from all packages passed to the `PackagesDataLoader`. In this simple example, only one was passed, so not much was consolidated, but the interest of consolidating is explained in *Using presamples to override input values*.

To move to the next (random) observation:

```
>>> ag_loader.update_package_indices()
>>> ag_loader.parameters.consolidated_array
array([5.33611000e+07, 1.15822293e+02, 1.45811000e+07])
```

The index value of each package's `Indexer` can be returned using the `consolidated_index`:

```
>>> for _ in range(4):
...     print(
...         "indices:",
...         ag_loader.parameters.consolidated_indices,
...         "values:",
...         ag_loader.parameters.consolidated_array
...     )
...     ag_loader.update_package_indices() # Move to the next (random) index
indices: [12, 12, 12] values: [5.33611000e+07 1.15822293e+02 1.45811000e+07]
indices: [8, 8, 8] values: [4.76672000e+07 6.22626679e+01 1.35367000e+07]
indices: [5, 5, 5] values: [5.60304000e+07 6.86041448e+01 1.65197000e+07]
indices: [0, 0, 0] values: [4.91972000e+07 5.76301666e+01 1.78330000e+07]
```

The indices are all the same because the `PackagesDataLoader` was populated with a single presamples package.

To use these in our model described in the *simple_example* section:

```
>>> for run_nb in range(5): # Run the model 5 times
...     print("Run number:", run_nb)
...
...     # Update the index, i.e. move to the next random index
...     ag_loader.update_package_indices()
# Calculate the model output using sampled parameter values
...     fertilizer_amount = fert_per_kg(
...         fert_kg_per_km2=ag_loader.parameters['fert consumption [kg/km2]'],
...         land_ha=ag_loader.parameters['fert consumption [kg/km2]'],
...         cereal_t=ag_loader.parameters['cereal production [t]']
...         )
...     # print the sampled column index and the model output for each run
...     print("\tindices:", ag_loader.parameters.consolidated_indices)
...     print("\tresult:",  '{:.2e}'.format(fertilizer_amount))
Run number: 0
    indices: [1, 1, 1]
    result: 6.84e-04
Run number: 1
    indices: [4, 4, 4]
    result: 4.60e-04
Run number: 2
    indices: [9, 9, 9]
    result: 1.15e-03
Run number: 3
    indices: [5, 5, 5]
    result: 8.40e-04
Run number: 4
    indices: [9, 9, 9]
    result: 1.15e-03
```

2.6 Storing a model's output as a presample package

The calculated model output (in the example, kg fertilizer per kg cereal) may be an input to another model. It would be possible to store the calculated output of our model as yet another presample package, and to use this directly in the other model.

While this example is simple, it is rather obvious that this can be a great advantage for larger models that take take a lot of computing resources.

```
>>> iterations = 100 # Number of iterations to store.
>>> model_output = np.zeros(shape=(1, iterations))
>>> for i in range(iterations):
...     ag_loader.update_package_indices()
...     model_output[0, i] = fert_per_kg(
...         fert_kg_per_km2=ag_loader.parameters['fert consumption [kg/km2]'],
...         land_ha=ag_loader.parameters['fert consumption [kg/km2]'],
...         cereal_t=ag_loader.parameters['cereal production [t]']
...     )
>>> model_output
array([[0.00133493, 0.00078676, 0.00081327, ..., 0.00078676, 0.00058567,
        0.00084508  ]])

>>> ag_result_pp_id, ag_result_pp_fp = presamples.create_presamples_package(
...     parameter_data = [(model_output, ['fert_input_per_kg_cereal'], "Agri model␣
↪output baseline")],
        name="baseline_model_output"
... )
```

This presample package can then be accessed or used as described above.

2.7 Creating presample packages with seeded indexers

Indexers are by default random. To force the indices to be returned in the same order everytime a presamples package is used, it is possible to specify a `seed` when creating the presamples package. This will ensure repeatability across uses of the presample package.

Reusing the original data, we simply pass a seed when using `create_presamples_package`:

```
>>> pp_id_seeded, pp_path_seeded = presamples.create_presamples_package(
...     parameter_data = [(ag_sample_arr, ag_names, "Agri baseline data")],
...     seed=42
... )
```

We can test that this worked by creating two `PackagesDataLoader` objects and making sure they return samples in the same order:

```
# Create a first loader and print indices and values
>>> ag_loader_seeded_1 = presamples.PackagesDataLoader([pp_path_seeded])
>>> ag_loader_seeded_2 = presamples.PackagesDataLoader([pp_path_seeded])
>>> ag_loader_seeded_1 is ag_loader_seeded_2
False
>>> ag_loader_seeded_1 == ag_loader_seeded_2
False
```

The two loaders are distinct, and yet:

```
>>> for _ in range(4):
...     ag_loader_seeded_1.update_package_indices()
...     print(
...         "indices:",
...         ag_loader_seeded_1.parameters.consolidated_indices,
...         "values:",
...         ag_loader_seeded_1.parameters.consolidated_array
...     )
indices: [5, 5, 5] values: [5.60304000e+07 6.86041448e+01 1.65197000e+07]
indices: [10, 10, 10] values: [6.64057010e+07 9.41524221e+01 1.59246840e+07]
indices: [8, 8, 8] values: [4.76672000e+07 6.22626679e+01 1.35367000e+07]
indices: [4, 4, 4] values: [4.80053000e+07 4.69949459e+01 1.61451000e+07]
```

```
>>> for _ in range(4):
...     ag_loader_seeded_2.update_package_indices()
...     print(
...         "indices:",
...         ag_loader_seeded_2.parameters.consolidated_indices,
...         "values:",
...         ag_loader_seeded_2.parameters.consolidated_array
...     )
indices: [5, 5, 5] values: [5.60304000e+07 6.86041448e+01 1.65197000e+07]
indices: [10, 10, 10] values: [6.64057010e+07 9.41524221e+01 1.59246840e+07]
indices: [8, 8, 8] values: [4.76672000e+07 6.22626679e+01 1.35367000e+07]
indices: [4, 4, 4] values: [4.80053000e+07 4.69949459e+01 1.61451000e+07]
```

2.8 Creating presample packages with sequential indexers

It can often be useful to sample values sequentially. To do so, pass `seed=sequential` when creating the presamples package.

```
>>> pp_id_seq, pp_path_seq = presamples.create_presamples_package(
...     parameter_data = [(ag_sample_arr, ag_names, "Agri baseline data")],
...     seed='sequential'
... )
>>> ag_loader_seq = presamples.PackagesDataLoader([pp_path_seq])
>>> for _ in range(4):
...     print(
...         "indices:",
...         ag_loader_seq.parameters.consolidated_indices,
...         "values:",
...         ag_loader_seq.parameters.consolidated_array
...     )
...     ag_loader_seq.update_package_indices()
indices: [0, 0, 0] values: [4.91972000e+07 5.76301666e+01 1.78330000e+07]
indices: [1, 1, 1] values: [5.07782000e+07 5.89276106e+01 1.61617000e+07]
indices: [2, 2, 2] values: [5.09624000e+07 5.46327748e+01 1.58468000e+07]
indices: [3, 3, 3] values: [4.85773000e+07 6.18212787e+01 1.59461000e+07]
```

2.9 Using presamples to override input values

Multiple presamples packages can be passed to a single `DataPackageLoader`. When a named parameted is present in more than one presamples package, only the value in the last package to have the named parameter is used. This allows for easily updating input data.

In our example, say we want to fix the fertilizer use parameter to an amount representing a specific scenario:

```
>>> new_fertilizer_amount = np.array([60]).reshape(1,1) # The array MUST have one row,
↪ as we only have one parameter
>>> fert_scenario_id, fert_scenario_path = presamples.create_presamples_package(
...     parameter_data=[(new_fertilizer_amount, ['fert consumption [kg/km2]'], 'ag␣
↪scenario 1')],
...     name="Scenario 1 agri data - presample package"
... )
```

We can now create a loader where both the baseline and the scenario packages are passed:

```
>>> ag_loader_scenario = presamples.PackagesDataLoader([pp_path, fert_scenario_path])
```

We can see that the original values for fertilizer use have been replaced by those in the new package.

```
>>> for _ in range(4):
...     print(
...         "indices:",
...         ag_loader_scenario.parameters.consolidated_indices,
...         "values:",
...         ag_loader_scenario.parameters.consolidated_array
...     )
...     ag_loader_scenario.update_package_indices()
indices: [6, 0, 6] values: [4.96919e+07 6.00000e+01 1.50603e+07]
indices: [11, 0, 11] values: [5.1535801e+07 6.0000000e+01 1.4023084e+07]
indices: [1, 0, 1] values: [5.07782e+07 6.00000e+01 1.61617e+07]
indices: [11, 0, 11] values: [5.1535801e+07 6.0000000e+01 1.4023084e+07]
```

Notice that the index for the second parameter ('fert consumption [kg/km2]') is always 0: this is because the package for this named parameter only has one column.

You can pass as many packages as required, and each package can have any number of named parameters and observations (columns).

Important: When passing multiple presamples package paths to a single `DataPackageLoader`, named parameters get their values and indices from the last presamples package that contains data on this named parameter.

2.10 Storing a presample resource

In order to facilitate their retrieval for reuse, references to presamples packages can be stored in a database `campaigns.db`. Interaction with this database is based on the Peewee ORM.

The first table of this database is the `PresampleResource` table, used to store references to presamples packages.

To store a reference to a presample package in the database:

```
>>> pr_baseline = presamples.PresampleResource.create(
...     name="Baseline agri data",
...     path=pp_path
... )
```

The resource has a few useful properties, such as `name` and `path`.

One can then retrieve a presample resource based on the name:

```
>>> pr_baseline_retrieved = presamples.PresampleResource.get(
...     presamples.PresampleResource.name=="Baseline agri data"
... )
>>> pr_baseline == pr_baseline_retrieved
True
```

and then use the associated presample package:

```
>>> other_loader = presamples.PackagesDataLoader([pr_baseline.path])
```

2.11 Using `Campaigns` to manage sets of presample packages

The `Campaign` database also has a table called `Campaign`, used to store information about ordered collections of `PresampleResources`.

To create a new campaign:

```
>>> ag_campaign_baseline = presamples.Campaign.create(name="Agricultural baseline␣
↪campaign")
>>> ag_campaign_baseline.save()
1
```

The 1 indicates that one row was changed in the `Campaign` table.

We can add our samples of baseline values using the `PresampleResource` that was created earlier:

```
>>> ag_campaign_baseline.add_presample_resource(pr_baseline)

>>> ag_campaign_baseline #Get some information on the campaign
<Campaign: Campaign Agricultural baseline campaign with no parent and 1 packages>

>>> [p.name for p in ag_campaign_baseline.packages] # List packages used in campaign
['Baseline agri data']
```

A `Campaign` can be passed directly to a `PackagesDataLoader`:

```
>>> loader = presamples.PackagesDataLoader(ag_campaign_baseline)
>>> loader.parameters.consolidated_array
array([6.64057010e+07, 9.41524221e+01, 1.59246840e+07])
```

More interestingly, a `Campaign` can point to multiple `PresampleResources` in the desired order. Let's add the scenario data we had above to a `PresampleResource`:

```
>>> pr_scenario = presamples.PresampleResource.create(
...     path=fert_scenario_path,
...     name="Scenario 1 agri data"
... )
```

Next we create a *child* Campaign based on the baseline campaign we created above. This child campaign will automatically point to all the resources of the parent Campaign. Note that you can have an arbitrary number of descendents.

```
>>> ag_campaign_scenario1 = ag_campaign_baseline.add_child("Agricultural scenario 1␣
↪campaign") # Create a child campaign
>>> ag_campaign_scenario1.save()
```

(continues on next page)

(continued from previous page)

```
1
>>> ag_campaign_scenario1.add_presample_resource(pr_scenario) # Add the scenario␣
↪presample resource
```

The ag_scenario Campaign has the baseline data as parent (it will use all its presample packages) and another package.

```
>>> ag_campaign_scenario1
<Campaign: Campaign Agricultural scenario 1 campaign with parent Agricultural␣
↪baseline campaign and 2 packages>

>>> [p.name for p in ag_campaign_scenario1.ancestors]
['Agricultural baseline campaign']

>>> [p.name for p in ag_campaign_scenario1.packages]
['Baseline agri data', 'Scenario 1 agri data']
```

Using the campaign in a PackagesDataLoader will call the presample packages in the expected order, i.e. from the package with baseline data (added first) to the scenario data (added after):

```
>>> loader_scenario = presamples.PackagesDataLoader(ag_campaign_scenario1) # Load
>>> for _ in range(4): # Check values for 4 iterations
...     loader_scenario.update_package_indices()
...     print(
...         "Indices:",
...         loader_scenario.parameters.consolidated_indices,
...         "Values: ",
...         loader_scenario.parameters.consolidated_array
...     )
Indices: [9, 0, 9] Values:  [5.1799100e+07 6.0000000e+01 1.4981496e+07]
Indices: [6, 0, 6] Values:  [4.96919e+07 6.00000e+01 1.50603e+07]
Indices: [2, 0, 2] Values:  [5.09624e+07 6.00000e+01 1.58468e+07]
Indices: [2, 0, 2] Values:  [5.09624e+07 6.00000e+01 1.58468e+07]
```

The scenario data overwrote the fertiliser use data on each iteration.

CHAPTER 3

Using presamples with brightway2

The presamples package is useful for writing, loading, managing and verifying presample arrays.

As shown elsewhere, this is useful to store reusable arrays of values for named parameters. Presamples can also be used to store **arrays of values for given matrix elements**, and to inject these in matrices when using them in calculations.

It was specifically geared towards matrices built by and used in the Brightway2 framework, though the code could be extended to work with any defined matrix.

This section provides an overview of the use of presamples in LCA. The first three sections provide a bit of context. If you are more of a hands-on type of person, you can jump straight to the examples. There is also a Notebook version of this documentation chapter, here. It contains extra code for e.g. importing the data used the examples and formatting the outputs.

There are also more detailed use cases in the *Examples* section.

3.1 LCA matrices and the case for using presamples

At its barest expression, LCA models can be represented with three matrices and a vector:

- the technosphere matrix **A**, describing the links among activities in the technosphere (technosphere exchanges)
- the biosphere matrix **B**, satellite matrix describing the exchanges between the activities and the environment (elementary flows)
- the characterization matrix **C**, giving unit impact factors for elementary flows with the environment (characterisation factors)
- the final demand vector **f**

An impact score per functional unit is given by $\mathbf{g} = \mathbf{CBA}^{-1}\mathbf{f}$

Presamples can replace values in any these matrices as calculations are carried out. Storing and injecting specific values in LCA matrices can improve LCA calculations in many ways:

- Storing and reusing data characterizing given scenarios makes scenario analysis much easier.
- It can easily integrate time series.
- It can use pre-generated static or stochastic values that were generated by complex, non-linear models, allowing the LCA model to capture system dynamics more accurately.
- It is possible to account to correlation across parameters during Monte Carlo Simulations (e.g. for correlation between characterization factors, between fuel use and CO2 emissions, etc.
- Since sampled data can be used directly, it is unnecessary to fit data to a distribution.

3.2 Creating and using presample packages with matrices in Brightway2

3.2.1 Defining the input matrix_data

presamples packages contain the information needed to overwrite matrix elements with new values. To create a presamples package, one passes `matrix_data` to `create_presamples_package`. The structure of `matrix_data` is the `matrix_data` is a list of tuples containing **samples**, **indices** and a **matrix label**.

Where:

1. **samples** are a two-dimensional numpy array containing the actual data to be stored in a presamples package.
2. Each row of the **samples** array contains data (any number of values) for a given matrix element.
3. Each column of the **samples** array contains values that should be used together for all matrix elements. For example, one column can contain data for the different matrix elements for one scenario, for one time step or one Monte Carlo iteration.
4. **indices** is a list of tuples containing information that will allow the presamples package to map the data in samples to the correct matrix elements. How indices are defined will depend on the type of matrix.
5. The nth element in **indices** refers to the nth row in **samples**.
6. Finally, the **matrix label** is a string giving the name of the matrix to be modified (e.g. 'technosphere', 'biosphere', 'cf').

3.2.2 Creating the presamples package

Suppose you have already correctly defined some `matrix_data` (see *here*), creating a presamples package is just as easy as for *named parameters*.

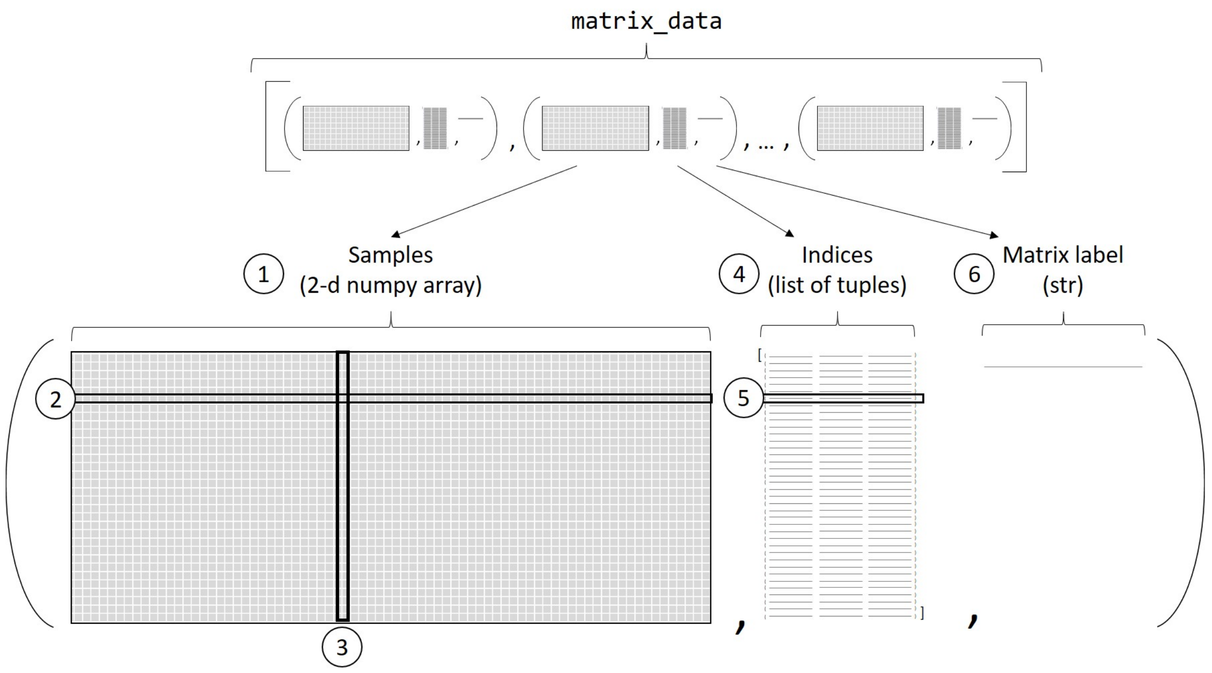


Fig. 1: Structure of `matrix_data` passed to `create_presamples_package`

```
>>> import presamples as ps
>>> pp_id, pp_path = ps.create_presamples_package(
...     matrix_data = some_well_defined_matrix_data,
... )
```

See *here* for a description of all arguments.

3.2.3 Using a presamples package in LCA

Presample packages can directly be passed to Brightway2 `LCA` (and, by extension, `MonteCarloLCA`) objects. Brightway2 will replace the appropriate matrix elements with the data contained in the presample package. This is done by simply passing a list of *paths* to presample packages in the `LCA` object:

```
>>> import brightway2 as bw
>>> LCA = bw.LCA({activity: amount}, presamples=[pp_path])
```

That is it.

3.3 Example 1 - Static scenario analysis: changing supplier by modifying the technoshere matrix

3.3.1 Context

Take the example of producer `('bd1', 'b')` that purchases product *a* from two suppliers, `('bd1', 'a1')` and `('bd1', 'a2')`.

Note: To better align with Brightway2, the activities are referred to by their activity *key*, which is a tuple made up of:

- their database (in this case, `'bd1'`)
- their activity code (in this case, one of `'a1'`, `'a2'` or `'b'`).

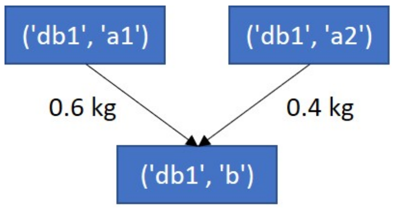


Fig. 2: Initial product system

Suppose that producer `('bd1', 'b')` decides to change (in simulation or in reality) its supplier of *a* to `('bd1', 'a2')`.

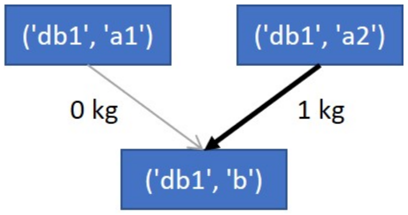
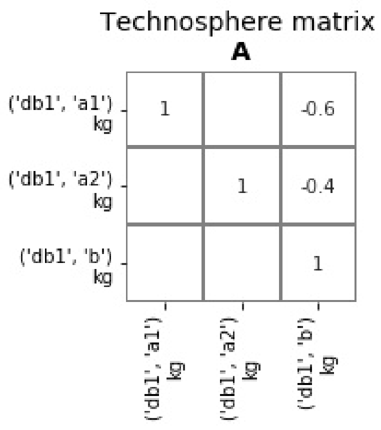


Fig. 3: Product system after change of supplier

In the LCA model, these changes occur in the technosphere matrix **A**. Originally, the **A** matrix looks like this:


Technosphere matrix
A

	('db1', 'a1') kg	('db1', 'a2') kg	('db1', 'b') kg
('db1', 'a1') kg	1		-0.6
('db1', 'a2') kg		1	-0.4
('db1', 'b') kg			1

Fig. 4: Technosphere matrix before changing supplier

The decision to start purchasing *a* exclusively from `('bd1', 'a2')` translates to the following changes in the technosphere matrix:

- changing the -0.6 value at element `('bd1', 'a1'), ('bd1', 'b')` to 0
- changing the -0.4 value at element `('bd1', 'a2'), ('bd1', 'b')` to 1

We can create a presamples package with this new data in order to override existing data with these new values in the context of an LCA.

3.3.2 Creating the `matrix_data`

The first step is to create the `matrix_data`:

1) The samples array is simply a numpy array with one column (we are modelling one new scenario) and two rows (we want to modify two values in the **A** matrix). Note that the values are positive, even if the values in the **A** matrix are actually negative. Presamples will know to flip the signs:

```
>>> import numpy as np
>>> scenario_array = np.array(
    [
        1,     # New value for exchange between ('bd1', 'a2') and ('bd1', 'b')
        0      # New value for exchange between ('bd1', 'a1') and ('bd1', 'b')
    ]).reshape(-1, 1)
```

2) The indices are defined as a list of tuples, with each tuple containing:

   - the activity key of the input (the supplier)
   - activity key of the output (the consumer)
   - the type of exchange, necessary for elements ot the technosphere matrix because exchanges can be of various types ('technosphere', 'production' or 'substitution'). In this case, the exchange type is 'technosphere', and this is how presamples knows to flip the signs of the values in the samples.

```
>>> scenario_indices = [
        (('bd1', 'a2'), ('bd1', 'b'), 'technosphere'),
        (('bd1', 'a1'), ('bd1', 'b'), 'technosphere')
    ]
```

3) The name of the matrix, simply `'technosphere'`.

The `matrix_data` is therefore:

```
>>> scenario_matrix_data = [(scenario_array, scenario_indices, 'technosphere')]
```

3.3.3 Creating the presamples package

The presamples package is created as follows:

```
>>> import presamples as ps
>>> scen_pp_id, scen_pp_path = ps.create_presamples_package(
...     matrix_data = scenario_matrix_data,
... )
```

3.3.4 Using the presamples package

The presamples package can directly be passed to an LCA object.

```
>>> import brightway2 as bw
>>> lca = bw.LCA(demand={('bd1', 'b'): 1}, presamples=[scen_pp_path])
```

This will load the presample packages, which will be accessible as an attribute of the lca object, `lca.presamples`.

```
>>> lca.presamples
<presamples.loader.PackagesDataLoader at 0x17eac8790b8>
```

However, you will probably never have to interact with the PackagesDataLoader directly.

The technosphere and biosphere matrices are built with the `LCA.load_lci_data` method, normally invoked by the `LCA.lci` method.

```
>>> lca.load_lci_data()
```

The technosphere matrix in this specific LCA is:

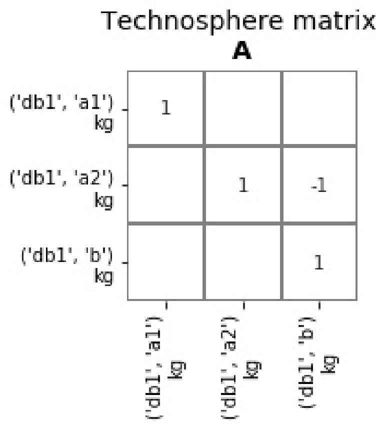


Fig. 5: Technosphere matrix after changing supplier with presamples package

The changes made are not persistent: they are only applied to this `LCA` object, and not to the database proper.

3.4 Example 2: Using presamples for time series

3.4.1 Context

Using the same product system as in Example 1, suppose that the sourcing of product *a* from the two suppliers varies in time. This type of situation happens e.g. with electricity grid mixes, where the contribution to the grid of different technologies varies in time.

Suppose we want to calculate LCA results per month, knowing the following:

Table 1: Purchases of product *a* from two suppliers

Date	('db1', 'a1')	('db1', 'a2')	Sum
Jan-19	0.9	0.1	1.0
Feb-19	0.8	0.2	1.0
Mar-19	0.6	0.4	1.0
Apr-19	0.3	0.7	1.0
May-19	0.6	0.4	1.0
Jun-19	0.5	0.5	1.0

3.4.2 Creating the `matrix_data`

We first reorganize the data as a numpy array:

```
>>> time_array = np.array(
...     [
...         [0.9, 0.8, 0.6, 0.3, 0.6, 0.5],
...         [0.1, 0.2, 0.4, 0.7, 0.4, 0.5]
...     ]
... )
>>> time_array.shape
(2, 6)
```

The indices are the same as in the previous example:

```
>>> time_indices = [
        (('bd1', 'a2'), ('bd1', 'b'), 'technosphere'),
        (('bd1', 'a1'), ('bd1', 'b'), 'technosphere')
    ]
```

The name of the matrix is again simply `'technosphere'`.

The `matrix_data` is therefore:

```
>>> time_matrix_data = [(time_array, time_indices, 'technosphere')]
```

3.4.3 Creating the presamples package

To create the presamples package, we again use `create_presamples_package`:

```
>>> time_pp_id, time_pp_path = ps.create_presamples_package(
...     matrix_data = time_matrix_data,
...     seed='sequential'
... )
```

3.4.4 Using the presamples package in an LCA

The presamples package is again passed directly to the `LCA` object:

```
>>> lca = bw.LCA(demand={('bd1', 'b'): 1}, presamples=[time_pp_path])
```

The first time the `lci` method is invoked, the first column of the array is injected in the matrices (because the presamples package seed is `'sequential'`, and so the column `index` starts at 0). The technology matrix is therefore:

If the `lci` method is called again, the values are not modified, and the `index` remains 0. To advance to the next index, the `lca.presamples.update_matrices()` needs to be called:

```
>>> lca.presamples.update_matrices()
```

Each time `lca.presamples.update_matrices()` is called, the index goes up by 1 (again, because the seed is sequential) and the matrices are updated.

```
>>> lca = bw.LCA({('db1', 'b'):1}, presamples=[time_pp_path])
>>> for i in range(10):
...     if i == 0: # Don't update the first time around, since indexer already at 0th␣
↪column
...         lca.lci() # Builds matrices, and injects values from first column of␣
↪presamples package
```

(continues on next page)

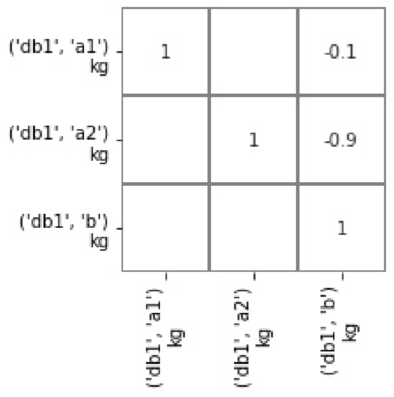


Fig. 6: Technosphere matrix after invoking `lca.lci()` once

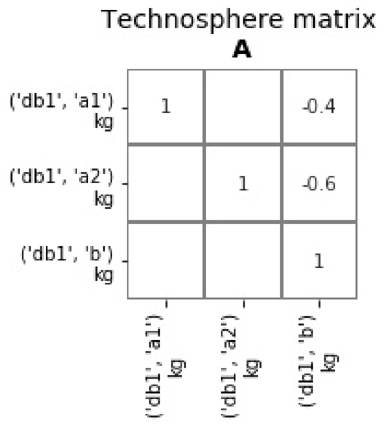


Fig. 7: Technosphere matrix after updating matrices once

(continued from previous page)

```
...     else:
...         lca.presamples.update_matrices() # Move to next column and update matrices
...     # Get value of input to ('db1', 'b') from supplier ('db1', 'a1')
...     from_a1 = lca.technosphere_matrix[
...         lca.product_dict[('db1', 'a1')], # row index in A for ('db1', 'a1')
...         lca.activity_dict[('db1', 'b')]  # col index in A for ('db1', 'b')
...     ]
...     # Get value of input to ('db1', 'b') from supplier ('db1', 'a2')
...     from_a2 = lca.technosphere_matrix[
...         lca.product_dict[('db1', 'a2')], # row index in A for ('db1', 'a2')
...         lca.activity_dict[('db1', 'b')]  # col index in A for ('db1', 'b')
...     ]
...     # Get index value for printing - normally one doesn't interact with lca.
↪presamples
...     index_value = lca.presamples.matrix_indexer[0].index
...     print(i,"\t\t", index_value,"\t\t", from_a1,"\t\t", from_a2)


Times updated       Index value         Input from a1       Input from a2
0                   0                       -0.1                -0.9
1                   1                       -0.2                -0.8
2                   2                       -0.4                -0.6
3                   3                       -0.7                -0.3
4                   4                       -0.4                -0.6
5                   5                       -0.5                -0.5
6                   0                       -0.1                -0.9
7                   1                       -0.2                -0.8
8                   2                       -0.4                -0.6
9                   3                       -0.7                -0.3
```

Note that the indexer starts back at 0 when all columns have been consumed.

The typical use would therefore be something like this:

```
>>> lca = bw.LCA({('db1', 'b'):1}, presamples=[time_pp_path], method=("mock method",
↪"pollutant emission"))
>>> for i in range(6):
...     if i == 0:
...         lca.lci()
...         lca.lcia()
...     else:
...         lca.presamples.update_matrices()
...         lca.redo_lci()
...         lca.redo_lcia()
...     print(i, lca.score)

0 1.1
1 1.2
2 1.4
3 1.7
4 1.4
5 1.5
```

Note that behaviour presented in this example will only work with presample packages that have the `'sequential'` seed. Here is an example where the seed is instead some integer, 42.

```
>>> time_not_seq_pp_id, time_not_seq_pp_path = ps.create_presamples_package(
...     matrix_data=time_matrix_data,
...     seed=42
... )
>>> lca = bw.LCA({('db1', 'b'):1}, presamples=[time_not_seq_pp_path])
>>> for i in range(10):
...     if i == 0: # Don't update the first time around, since indexer already at 0th␣
↪column
...         lca.lci() # Builds matrices, and injects values from first column of␣
↪presamples package
...     else:
...         lca.presamples.update_matrices() # Move to next column and update matrices
...     # Get value of input to ('db1', 'b') from supplier ('db1', 'a1')
...     from_a1 = lca.technosphere_matrix[
...         lca.product_dict[('db1', 'a1')], # row index in A for ('db1', 'a1')
...         lca.activity_dict[('db1', 'b')]  # col index in A for ('db1', 'b')
...     ]
...     # Get value of input to ('db1', 'b') from supplier ('db1', 'a2')
...     from_a2 = lca.technosphere_matrix[
...         lca.product_dict[('db1', 'a2')], # row index in A for ('db1', 'a2')
...         lca.activity_dict[('db1', 'b')]  # col index in A for ('db1', 'b')
...     ]
...     # Get index value for printing - normally one doesn't interact with lca.
↪presamples
...     index_value = lca.presamples.matrix_indexer[0].index
...     print(i,"\t\t", index_value,"\t\t", from_a1,"\t\t", from_a2)


Times updated       Index value         Input from a1       Input from a2
0                   0                       -0.1                -0.9
1                   3                       -0.7                -0.3
2                   2                       -0.4                -0.6
3                   4                       -0.4                -0.6
4                   4                       -0.4                -0.6
```

3.5 Example 3 - Modifying emission data by modifying the biosphere matrix

3.5.1 Context

Say a producer ('bd2', 'b') wants to estimate the environmental impacts of a process change that will affect both the amount of an input ('bd2', 'a') and the amount of emission ('bio', 'emission').

Note: for simplicity, the emissions of unit process ('bd2', 'a') (0.5 kg/kg a) are not shown in the diagram.

3.5.2 Creating the `matrix_data`

We can model this scenario using presamples. This time, we need to provide `matrix_data` for both the technosphere and biosphere exchanges.

```
>>> eg3_matrix_data = [
...     (
```

(continues on next page)

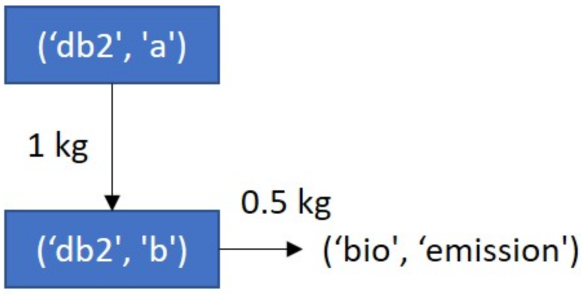


Fig. 8: Example 3, before emission reduction strategy

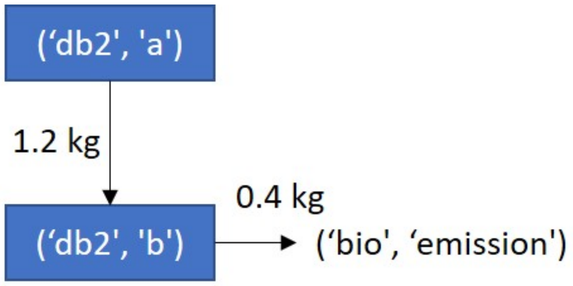


Fig. 9: Example 3, after emission reduction strategy

(continued from previous page)

```
...         np.array([1.2]).reshape(1, 1), # Only one value, but array still needs to␣
↪have two dimensions
...         [(('db2', 'a'), ('db2', 'b'), 'technosphere')],
...         'technosphere'
...     ),
...     (
...         np.array([0.4]).reshape(1, 1), # Again, only one value
...         [(('bio', 'emission'), ('db2', 'b'))], # No need to specify the exchange␣
↪type
...         'biosphere'
...     ),
... ]
```

Note that there was no need to specify the exchange type in the indices for the biosphere matrix because there is only one type of exchange in the biosphere matrix.

3.5.3 Creating the presamples package

To create the presamples package, we again use `create_presamples_package`:

```
>>> eg3_pp_id, eg3_pp_path = ps.create_presamples_package(
...     matrix_data = eg3_matrix_data,
... )
```

3.5.4 Using the presamples package in an LCA

Passing the path to the presamples package when instantiating an `LCA` object will automatically ready the package data, and calling the `lci` method will inject the presample package value in the appropriate matrices:

```
>>> lca1 = bw.LCA(
...     {('db2', 'b'):1},
...     method=('mock method', 'pollutant emission'),
...     presamples=[eg3_pp_path]
... )
>>> lca1.lci()
>>> lca1.lcia()
```

To compare with LCA result of the baseline model, simply create another LCA object without presamples:

```
>>> lca0 = bw.LCA({('db2', 'b'):1}, method=('mock method', 'pollutant emission'))
...     {('db2', 'b'):1},
...     method=('mock method', 'pollutant emission')
... )
>>> lca0.lci()
>>> lca0.lcia()
>>> lca1.score / lca0.score
0.909
```

3.6 Example 4 - Balancing sampled exchange values

3.6.1 Context

To account for variability in particular and uncertainty in general, exchanges can be represented by probability functions. These probability functions can then be propagated using e.g. Monte Carlo Simulations.

Sometimes, exchanges that are correlated in reality are independently sampled. This is the case for fuel consumption and combustion emissions if are represented by probability functions.

Take the following example, loosely based on a dataset from ecoinvent v2.2, where the fuel is diesel and the emission is CO2:

- for every kg of fuel input, there are 3 kg of emission
- fuel input is lognormally distributed with a GSD2 of 1.2
- emissions are lognormally distributed with a GSD2 of 1.2
- the cradle-to-gate emissions of fuel production are also lognormally distributed with a GSD2 of 1.2

If we sample all these values independently, the ratio of combustion emission to fuel input will always be off:

```
>>> mc = bw.MonteCarloLCA({('db3', 'a'):1}, method=("mock method", "pollutant emission
↪"))
>>> print("Fuel\t\tEmission\tRatio")
>>> for _ in range(10):
...     arr[_]=next(mc)
...     fuel = mc.technosphere_matrix[
...         mc.product_dict[('db3', 'fuel')],
...         mc.activity_dict[('db3', 'a')],
...     ]
...     emission = mc.biosphere_matrix[
```

(continues on next page)

(continued from previous page)

```
...         mc.biosphere_dict[('bio', 'emission')],
...         mc.activity_dict[('db3', 'a')],
...     ]
...     print("{:.3}\t\t{:.3}\t\t{:.6}".format(-fuel, emission, -emission/fuel))
Fuel            Emission        Ratio
1.11            3.32            3.0
1.02            2.91            2.84968
0.969           3.5             3.61066
0.855           2.92            3.42095
1.17            2.77            2.37436
1.13            3.04            2.69255
0.975           2.78            2.84856
0.881           3.35            3.79877
0.923           2.81            3.03841
0.966           3.04            3.14529
```

3.6.2 Defining matrix data

We can create samples for fuel consumption, and then calculate combustion emissions as a function of fuel consumption.

```
>>> import numpy as np
>>> fuel_consumption = np.random.lognormal(mean=np.log(1), sigma=np.log(np.sqrt(1.2)),
↪ size=1000)
>>> emissions = fuel_consumption * 3
```

We then define `matrix_data` for these exchanges. Here, we use the `split_inventory_presamples` helper function, which allows us to pass biosphere and technosphere samples together. Note that you need to specify the exchange type for biosphere exchanges when using this function.

```
>>> balanced_samples = np.stack([fuel_consumption, emissions], axis=0)
>>> balanced_indices = [
...     (('db3', 'fuel'), ('db3', 'a'), 'technosphere'),
...     (('bio', 'emission'), ('db3', 'a'), 'biosphere'),
... ]
>>> matrix_data = ps.split_inventory_presamples(balanced_samples, balanced_indices)
>>> bio_data = matrix_data[0] # matrix_data for biosphere exchanges comes first
>>> bio_data[0][0, 0:10], bio_data[1], bio_data[2] # Show that we get what we expected
(array([3.30923763, 2.88238829, 2.82005106, 2.92788522, 2.86454275,
        3.41350596, 3.03328453, 2.97565299, 2.9725899 , 2.91354418]),
[(('bio', 'emission'), ('db3', 'a'))],
'biosphere')
>>> techno_data = matrix_data[1]
>>> techno_data[0][0, 0:10], techno_data[1], techno_data[2] # Show that we get what␣
↪we expected
(array([1.10307921, 0.9607961 , 0.94001702, 0.97596174, 0.95484758,
        1.13783532, 1.01109484, 0.99188433, 0.9908633 , 0.97118139]),
[(('db3', 'fuel'), ('db3', 'a'), 'technosphere')],
'technosphere')
```

3.6.3 Creating and using the presamples package matrix data

```
>>> balanced_id, balanced_path = ps.create_presamples_package(
        matrix_data=ps.split_inventory_presamples(balanced_samples, balanced_indices)
    )
>>> mc_balanced = bw.MonteCarloLCA({('db3', 'a'):1}, presamples=[balanced_path])
```

Two `MonteCarloLCA` objects were generated: one without presamples (unbalanced), where fuel consumption and combustion emissions are independently sampled, and one with presamples (balanced), defined as above. For each, 1000 iterations were taken, and for each iteration, the fuel consumption and combustion emission were lifted from technosphere and biosphere matrices. We can see that the emission to fuel ratio is respected in all iterations for the balanced `MonteCarloLCA` object, but never for the unbalanced `MonteCarloLCA` object.

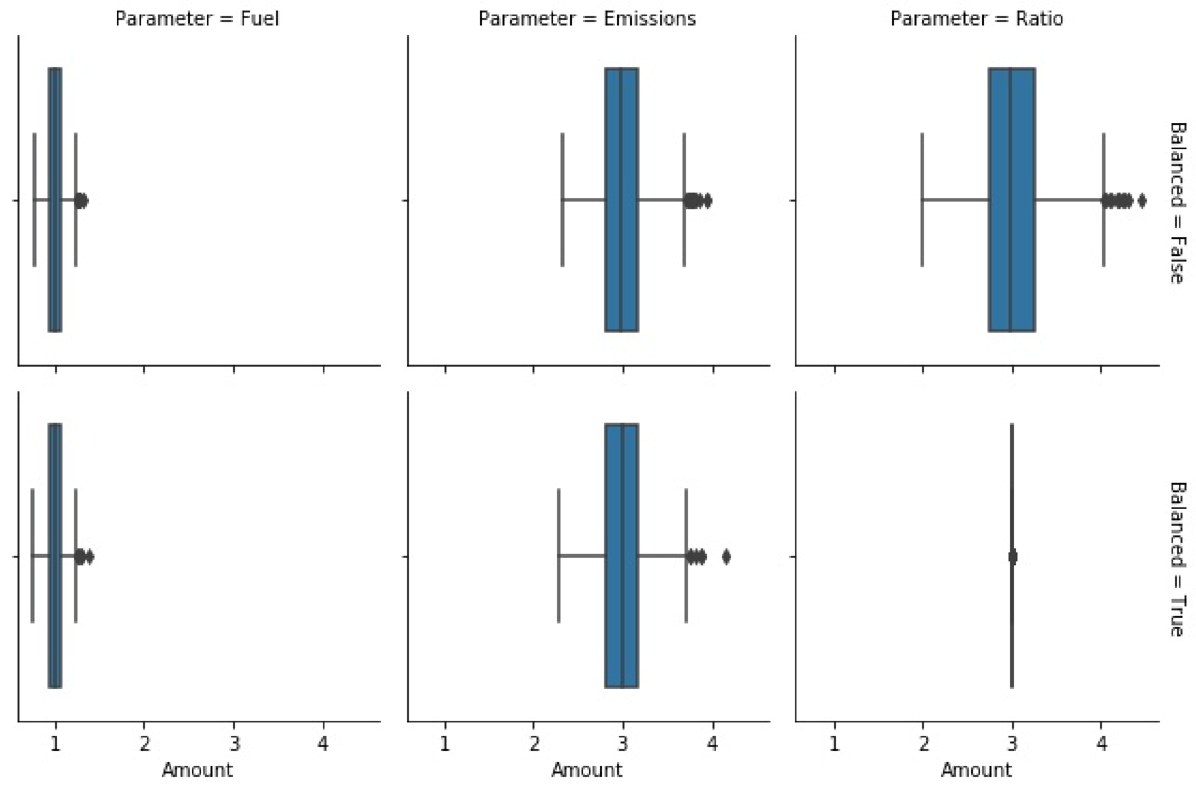


Fig. 10: Example 4, Fuel consumption and combustion emissions for balanced (using presamples) and unbalanced `MonteCarloLCA` objects

3.6.4 Real cases

This type of balancing is especially relevant for exchanges that are independently sampled and for which the ratio or balance is key to assessing the environmental impacts. Two cases are water exchanges and land transformation exchanges. Specific modules, built on brightway2 and presamples, have been developed for these cases specifically: bw2waterbalancer and bw2landbalancer

3.7 Other uses to document

The documentation is not complete. The following items are still missing:

- Kronocker delta helper function, useful for sampling one supplier from a market at each Monte Carlo iteration
- Fixed sums, useful for balancing markets
- Using presamples to store dependently sampled, preaggregated LCI for a whole LCI database
- Using presamples *resources* to store scenarios for easy reuse
- Using campaigns to efficiently manage presample resources
- Using Parameterized brightway models to combine parameter and matrix presamples in actual brightway2 models

CHAPTER 4

Technical reference

4.1 Presample packages

Presamples packages are directories that **must** minimally contain a `datapackage.json` file, based on the datapackage standard. It may contain other files, depending on the types of resources contained in the presamples package. The file contents are described here *Presample package contents*.

`presamples.packaging.`**`create_presamples_package`**`(`*matrix_data=None, parameter_data=None, name=None, id_=None, overwrite=False, dirpath=None, seed=None, collapse_repeated_indices=True*`)`

Create and populate a new presamples package

The presamples package minimally contains a datapackage file with metadata on the datapackage itself and its associated resources (stored presample arrays and identification of what the values in the arrays represent).

matrix_data: list, optional list of tuples containing raw matrix data (presamples array, indices, matrix label)

parameter_data: list, optional list of tuples containing raw parameter data (presamples array, names, label)

name: str, optional A human-readable name for these samples.

id_: str, optional Unique id for this collection of presamples. Optional, generated automatically if not set.

overwrite: bool, default=False If True, replace an existing presamples package with the same `\id_` if it exists.

dirpath: str, optional An optional directory path where presamples can be created. If None, a subdirectory in the `project` folder.

seed: {None, int, "sequential"}, optional Seed used by indexer to return array columns in random order. Can be an integer, "sequential" or None.

collapse_repeated_indices: bool, default=True Indicates whether samples for the same matrix cell in a given array should be summed. If False then only the last sample values are used.

Notes

Both `matrix_data` and `parameter_data` are optional, but at least one needs to be passed. The documentation gives more details on these input arguments.

Both matrix and parameter data should have the same number of possible values (i.e same number of samples).

The documentations provide more information on the format for these two arguments.

Returns

- **id_** (*str*) – The unique `id_` of the presamples package
- **dirpath** (*str*) – The absolute path of the created directory.

4.1.1 Description of the `parameter_data` argument

The `parameter_data` argument in `create_presamples_package` is a list (or other iterable) of tuples containing the following three objects:

- `samples`: are a two-dimensional numpy array, where each row contains values for a specific named parameter, and columns represent possible values for these parameters. It is possible to have samples arrays with only one column (i.e. only one observation for the named parameters), and only one row (data on only one named parameter).
- `names`: list of parameter names, as strings. The order of the names should be the same as the rows in `samples`, i.e. the first name corresponds to data in the first row of `samples`.
- `label`: a string which will be used to name the resource. The presamples package does not presently use this label.

Important: There must necessarily be as many named parameters in `names` as there are rows in `samples`.

Note: It is possible to pass an arbitrary amount of (`samples`, `names`, `label`) tuples in `parameter_data`. Each will be contained in a distinct resource of the presamples package. However,

1. the names in each tuple *must* be unique, and
2. the number of columns in each `samples` must be identical.

Hint: While there are no restrictions on the string, using strings that are valid names in AST evaluators can prevent problems down the line.

4.1.2 Description of the `matrix_data` argument

The `matrix_data` argument in `create_presamples_package` is a list (or other iterable) of tuples containing the following three objects:

- `samples`: are a two-dimensional Numpy array, where each row contains values for a specific matrix element that will be replaced and each column contains values for a given realization of the LCA model. It is possible to have samples arrays with only one column (i.e. only one observation for the matrix elements), and only one row (data on only one named parameter).
- `indices`: is an iterable with row and (usually) column indices. The *ith* element of indices refers to the *ith* row of the samples. The exact format of indices depends on the matrix to be constructed. These indices tell us where exactly to insert the samples into a specific matrix.
- `matrix label`: is a string giving the name of the matrix to be modified in the LCA class. Strings that are currently supported are 'technosphere', biosphere' and 'cf'.

Important: The number of rows in `samples` and `indices` must be identical.

Note: It is possible to pass an arbitrary amount of (`samples`, `indices`, `matrix label`) tuples in `matrix_data`. Each will be contained in a distinct resource of the presamples package. However, the number of columns in each `samples` must be identical.

4.1.3 Presample package contents

datapackage.json

Presample packages are directories that **must** minimally contain a `datapackage.json` file, based on the datapackage standard. It may contain other files, depending on the types of resources contained in the presamples package.

The format for the datapackage.json file is:

There can be an arbitrary number of resources. Each resource is represented in the datapackage by a dictionary.

Named parameters

Resources for named parameters have the following format:

Where: - the id is based on the `\id_` argument passed to `create_presamples_package` - the data package index indicates the position (index) of the resource in the list of resources

Matrices

The last elements ("row from label", "row to label", etc.) are used by the *`presamples.loader.PackagesDataLoader.index_arrays()`* method to map the resource elements to the LCA matrices.

4.2 Loading multiple presample packages

Loading multiple presample packages for use in models requiring one value at a time is done using the `PackagesDataLoader` class.

class `presamples.loader.`**`PackagesDataLoader`**(*dirpaths*, *seed=None*, *lca=None*)

Load set of presample packages and ready underlying data for use

Named parameters found in presample packages will be assembled into a `ConsolidatedIndexedParameterMapping`, accessed through the `parameters` property.

Matrix data found in presample packages are readied to be mapped and inserted into LCA matrices using the corresponding methods.

In both cases, elements (named parameter or matrix element) repeated in multiple presample packages will take the value (and the `Indexer`) of the last presample package in the list that contains data on the element.

Parameters

- **dirpaths** (*iterable of paths to presample packages*) – See notes below for information on expected contents of directories.
- **seed** (*{None, int, array_like, "sequential"}, optional*) – Seed value to use for index RNGs. Default is to use the seed values in each package, only specify this if you want to override the default.
- **lca** (*Brightway2 LCA object*) – Used when `PackagesDataLoader` instantiated from LCA (or MonteCarloLCA) object.

Notes

1. Accessing and using loaded named parameters

The returned `PackagesDataLoader` instance allows access to loaded parameter data via the `parameters` property.

2. Using loaded matrix data in LCA

When used for LCA within the Brightway2 framework, the `PackagesDataLoader` instance is an attribute of the `LCA` (or `MonteCarloLCA`) instance. The `LCA` instance will call the method `index_arrays` in order to identify the matrix indices of values that will be overwritten, and the `update_matrices` method to update values.

Warning: The order of the passed presample package dirpaths is key! Every matrix element or named parameter that is repeated in multiple presample packages will take the value of the *last* presample package that is passed. All former values will not be used.

Warning: Note that we currently assume that all index values in matrix data will be present in the built matrices of the LCA instance. Silent errors or losses in efficiency could happen if this assumption does not hold.

When creating a `PackagesDataLoader` instance, parameter and matrix data automatically loaded by invoking the method `load_data` to each path in `dirpaths`:

classmethod `PackagesDataLoader.`**`load_data`**(*dirpath*, *seed=None*)

Load data and metadata from a directory containing a presamples package

Parameters

- **`dirpath`** (*`str or Pathlike object`*) – path to a presamples package
- **`seed`** (*`{None, int, array_like, "sequential"}, optional`*) – Only specify this if you want to override seed value in presamples package.

Returns Dictionary with loaded data

Return type dict

Loaded data can then be parsed for accessing *consolidated* parameters or for injecting data in LCA matrices.

4.2.1 Using named parameters in `PackagesDataLoader`

With `load_data`, all presample packages are loaded. However, to ensure that only the **last** presample package with data on a specific named parameter is used, parameters are *consolidated*. The consolidated parameters are available via the `parameters` property. `parameters` points to a `ConsolidatedIndexedParameterMapping` object.

class `presamples.loader.`**`ConsolidatedIndexedParameterMapping`**(*list_IPM*)

Interface for consolidated named parameters in set of presample packages

Map all named parameters in a list of IndexedParameterMapping objects to presample arrays and Indexers identified in the **last** presample package that contains data on the named parameter.

This allows named parameters to be overwritten by successive presample packages.

Typically called directly from a *`PackagesDataLoader`* instance.

Parameters **`list_IPM`** (*`list`*) – List of IndexedParameterMapping objects. The IndexedParameterMapping (IPM) objects are typically created by a `PackagesDataLoader` instance.

Important: The order of the IPMs is crucial, as named parameters in later IPMs overwrites data from earlier IMPs.

Notes

The CIPM instance can be used to access the following properties:

- `names`: names of all *n* named parameters

- `ipm_mapper`: dict {parameter name: IndexedParameterMapping}, identifying the IndexedParameterMapping used for a given named parameter.
- `consolidated_array`: array of shape (n,) values, giving access to the values for the *n* named parameters
- `consolidated_index`: array of shape (n,) values, giving access to the index values for the *n* named parameters in their respective IndexedParameterMapping
- `ids`: dict of format {named parameters: ids}, where `ids` are tuples of (presamples package path, presamples package name, name of parameter). `ids` only contains information about the *last* presamples package with data on the named parameter.
- `replaced`: dict of format {named parameters: ids of presample packages that were overwritten}

`consolidated_array`

Array of values for named parameter

Each value is taken from the last IndexedParameterMapping object that contains data on the named parameter. The used IndexedParameterMapping contains information about the path to the presamples array, the corresponding mapping for the named parameter and the current Indexer value.

`consolidated_indices`

Return the index value for the IndexedParameterMapping used for each name

4.2.2 Using `PackagesDataLoader` with LCA

Brightway `LCA` (and `MonteCarloLCA`) objects can seamlessly integrate presample packages.

```
>>> from brightway2 import LCA
>>> lca = LCA(demand={product:1}, presamples=[pp_path1, pp_path2, pp_path3])
```

This instantiates a `PackagesDataLoader` as described above.

It then indexes arrays:

`PackagesDataLoader.`**`index_arrays`**(*lca*)

Add row and column values to the indices.

As this function can be called multiple times, we check for each element if it has already been called, and whether the required mapping dictionary is present.

Finally, data from the correct columns in the presamples arrays are inserted in the LCA matrices:

`PackagesDataLoader.`**`update_matrices`**(*lca=None*, *matrices=None*, *advance_indices=True*)

Update the LCA instance matrices from presamples

CHAPTER 5

Examples

CHAPTER 6

Publications

Here are some examples of publications or tools where Presamples have been used.

If you use Presamples in one of your projects, feel free to add it to this list.

6.1 Scientific papers

[1] Lesage, P., Mutel, C., Schenker, U. and Margni, M. Uncertainty analysis in LCA using precalculated aggregated datasets. Int J Life Cycle Assess (2018) 23: 2248. https://doi.org/10.1007/s11367-018-1444-x

6.2 Conference presentations

[1] Mutel, C. and Lesage, P. Direct sampling to improve accuracy of Monte Carlo uncertainty analysis SETAC 2019. Presentation and code available here

6.3 Software

[1] Canadian Analytical Framework for the Environmental Evaluation of Electricity (CAFE3) Developped by the CIRAIG for Environment and Climate Change Canada. Not publically available.

CHAPTER 7

Contributing

Contributions are welcome! Presamples is true open source software - our code and development process is as open as we can make it.

To contribute code, please follow the standard Github workflow. Make sure to add tests for any contribution you make.

Once your contribution has been merged, add yourself to the authors list on this page.

CHAPTER 8

Authors

The core Presamples team is Pascal Lesage from CIRAIG and Chris Mutel from PSI.

Index