Provided macros

11th January 2023

Authors:

Ada Gąssowska <ada.gassowska.stud@pw.edu.pl>

Katarzyna Solawa <katarzyna.solawa.stud@pw.edu.pl>

Kacper Kurowski <kacper.kurowski.dokt@pw.edu.pl>

Overview:

This set of macros provides an implementation of the algorithms for sampling from the Determinanatal Point Processes (DPPs) written natively in SAS. DPPs are stochastic point processes (so, their result is a subset of that set) which respect the diversity present in the set.

The set offers ways to sample from Finite, Continuous and the so-called Exotic DPPs. For Finite DPPs the set of possible values is finite, for continuous it is infinite, and the samples should approximate some continuous distribution. The exotic DPPs however, are a special kind of DPPs which were analyzed for different reasons and one needed a change-of-perspective to deduce that they can be thought of as DPPs.

The available continuous DPPs come from a family of distributions called Beta ensembles. They were initially studied in Physics, as they can be thought of as models of a Coulomb gas.

Before you use the macros, you must include the file with them:

```
%let dirPath =Path_to_repository;
%include "&dirPath.\SAS4GL\loadModules.sas";
%load_modules(&dirPath.);
```

Macros:

- Finite DPPs:
  - `sample_exact`
  - `sample_mcmc`
- Continuous DPPs:
  - `sample_from_beta_ensemble_full`
  - `sample_from_beta_ensemble_banded`
- Exotic DPPs:
  - `poiss_planch_sample`
  - `stat_1_dep_sample`

Available Finite DPP Macros:

sample_exact Macro

Syntax:

```
%sample_exact(
    kernel_type,
    K_kernel=.,
    L_kernel=.,
    K_e_vals=.,
    K_e_vecs=.,
    L_e_vals=.,
    L_e_vecs=.,
    dest=sample,
    mode=GS,
    projection=N,
    size=.,
    random_state=.
);
```

Parameters

`kernel_type` Type of kernel kind used for sampling .

`X_kernel` L - Likelihood or K - correlation matrix.

`X_e_vals` L - Likelihood or K - correlation matrix eigen values.

`X_e_vecs` L - Likelihood or K - correlation matrix eigen vectors.

`dest` destination to which the sample should be written to.

`mode` One of the following: *GS,KuTa12, Chol, KuTa12*. Determines the algorithm with which the sample is generated. Default value is *GS*.

`projection` One of the following: *0,1*. Determines whether kernel is projection correlation kernel matrix. Default value is *N*.

`size` Desired size of the output sample. Should be less than or equal to rank(kernel). If none is provided then the sample will be of size equal to rank(kernel)

`random_state` Seed for the randomness. If provided it should be a positive integer

Description

Correlation kernel is calculated based on one or two input matrcies . All method use correlation sample but with kernel_type = N some methods are not available or use generic sampling method instead.

For Miner Usage insted of pasing input matrix name as parameter we connect dataset to node:

- `X_kernel` - Select kernel_type parameter and connect to node kernel dataset with role: Train
- `X_e_vals` - Select kernel_type parameter and connect to node eigen values dataset with role: Test.
- `X_e_vecs` - Select kernel_type parameter and connect to node eigen vectors dataset with role: Validation.

Example:

```
%sample_exact(
    correlation,
    K_kernel=K,
    projection=Y
    random_state=1
);

%sample_exact(
    likelihood,
    L_kernel=L,
    dest=sample1,
    mode=chol
    projection=N
    random_state=1
);

%sample_exact(
    likelihood,
    L_e_vals=L_vecs,
    L_e_vecs=L_vals,
    mode=KuTa12
    projection=N,
    size=10
    random_state=1
);
```

sample_mcmc Macro

Syntax:

```
%sample_mcmc(
    K=.,
    s_init=.,
    dest=sample,
    sampling_mode=.,
    kernel_type=.,
    projection=.,
    random_state=.,
    nb_iter=10,
    size=.
);
```

Parameters

`K` Kernel matrix

`s_init` Initial sample from set of values represented by K. If none is provided then the algorithm will generate the initial sample.

`dest` destination to which the sample should be written to.

`kernel_type` One of the following: *correlation, likelihood.*Define the type of the given kernel.

`projection` One of the following: *0,1.* Determines whether kernel is projection correlation kernel matrix. Default value is *N*.

`random_state` Seed for the randomness. If provided it should be a positive integer

`nb_iter` Number of iterations that the algorithm will perform

`size` Desired size of the output sample. Should be less than or equal to rank(kernel). If none is provided then the sample will be of size equal to rank(kernel)

Description

For given likelihood matrix, the macro returns a sample calculated by adding or removing an element (with specific probabilities) from given samples (starting from initial s0 sample) for a number of iterations specified by the nb_iter parameter. If the initial sample - s_init is not provided, then it is generated by the algorithm . It is not possible to determine the size of the output sample, however it is possible to provide a size of the generated initial sample.

Example:

```
%sample_mcmc(
    K=kernel,
    kernel_type=correlation,
    sampling_mode=AED
);
```

Available Continuous DPP Macros:

sample_from_beta_ensemble_full Macro

Syntax:

```
%sample_from_beta_ensemble_full(
    result_eigvals,
    ensemble_version,
    M_1, M_2,
    size=10,
    beta=2,
    normalize=1,
    haar_mode="Hermite",
    heurestic_fix=1,
    random_state=1618
);
```

Parameters

`result_eigvals` destination to which the sample should be written to.

`ensemble_version` Version of Beta ensemble to use. Available values are `Hermite`, `Laguerre`, `Jacobi`, `Circular`, and `Ginibre`

`M_1` Distribution parameter for the `Laguerre` and `Jacobi` `ensemble_versions`. Should be greater or equal to `size`.

`M_1` Distribution parameter for the `Jacobi` `ensemble_version`. Should be greater or equal to `size`.

`size` Size of the sampled subset.

`beta` Beta parameter. Should be 1, 2, or 4.

`normalize` Parameter which states whether the sample should be normalized to fit one of the more known distributions.

`haar_mode` Which Haar measure mode to use. Can be `Hermite` or `QR`. Influences the result only for the Circular Ensemble. (Should be 1 or 0).

`heurestic_fix` Whether to apply the heuresis to fix the oversampling problem present in the Circular and Ginibre ensembles. Should be 1 or 0.

`random_state` Seed for the randomness. Should be a positive integer.

Description

The function provides the method for sampling from beta ensemble using the full matrix method. There are five versions of Beta ensembles that have been implemented. `Hermite`, `Ginibre`, `Jacobi`, `Circular`, and `Ginibre`. For the first three, the result is a one-column sample. For the next two, it is a two-column sample.

Example:

```
%sample_from_beta_ensemble_full(
        result_eigvals=sample,
        ensemble_version=Circular,
        M_1=10, M_2=10,
        size=10,
        beta=4,
        normalize=0,
        haar_mode=Hermite,
        heurestic_fix=0,
        random_state=1618.
);

run scatter(sample[,1], sample[,2]);
```

sample_from_beta_ensemble_banded Macro

Syntax

```
%sample_from_beta_ensemble_banded(
    result_eigvals,
    ensemble_version,
    size=10,
    beta=2,
    loc=0.0,
    scale=1.0,
    shape = 1.0,
    a = 1.0, b = 1.0,
    normalize=1,
    heurestic_fix=1,
    random_state=1618
);
```

Parameters

`result_eigvals` destination to which the sample should be written to.

`ensemble_version` Version of Beta ensemble to use. Available values are `Hermite`, `Laguerre`, `Jacobi`, and `Ginibre`

`size` Size of the sampled subset.

`beta` Beta parameter. Should be positive integer.

`loc` Location parameter for the standard deviation for the `Hermite` Beta ensemble.

`scale` Scale parameter for the expected value for the `Hermite` and `Laguerre` Beta ensembles.

`shape` Shape parameter for the `Laguerre` Beta ensemble.

`a` Parameter for the `Jacobi` Beta ensemble. Related to the Beta distribution.

`b` Parameter for the `Jacobi` Beta ensemble. Related to the Beta distribution.

`normalize` Parameter which states whether the sample should be normalized to fit one of the more known distributions.

`heurestic_fix` Whether to apply the heuresis to fix the oversampling problem present in the Circular and Ginibre ensembles. Should be 1 or 0.

`random_state` Seed for the randomness. Should be a positive integer.

Description

The function provides the method for sampling from beta ensemble using the banded matrix method. There are five versions of Beta ensembles that have been implemented. `Hermite`, `Laguerre`, `Jacobi`, and `Ginibre`. For the first three, the result is a one-column sample. For the Ginibre ensemble, it is a two-column sample.

Example

```
%sample_from_beta_ensemble_banded(
    result_eigvals=sample,
    ensemble_version=Hermite,
    size=1000,
    beta=2,
    loc=0.0,
    scale=1.0,
    shape=1.0,
    a=1.0, b=1.0,
    normalize=0,
    heurestic_fix=0,
    random_state=1618
);
```

Available Exotic DPP Macros:

sample_poissonized_plancherel Macro

Syntax

```
%sample_poissonized_plancherel(
    theta,
    dest=work.poissonized_sample,
    random_state=.);
```

Parameters

`theta` parameter of poisson distribution, must be integer > 1.

`dest` Desired destination of output sample (if none is provided then sample will be saved in work.poissonized$_{s}ample$).

`random_state` Seed for the randomness. Should be a positive integer.

Description

Generates a sample from the Poissonized Plancherel method by using RSK (Robinson-Schensted-Knuth) algorithm on a random permutation on $1, N$ where N is generated from Poisson distribution with parameter theta. It is not possible to determine the size of the output sample (it will always be <= theta). The output sample is saved as SAS dataset in the directory specified by the user.

Example

```
%sample_poissonized_plancherel(
    theta=10,
    dest=work.poissonized_sample,
    random_state=123);
```

stat_1_dep_sampler Macro

Syntax

```
%stat_1_dep_sampler(
    size=100,
    dest=work.stat_sample,
    mode=descent,
    base=.,
    x0=0.5,
    random_state=.)
```

Parameters

`size` Size of the generated list, upper bound of output sample, must be integer >1.

`dest` Desired destination of output sample (if none is provided then sample will be saved in work.stat_sample).

`mode` One of the following: *descent, carries, virtual.* Determines the algorithm with which the sample is generated. Default value is *descent.*

`base` Only used if *mode=carries* (then it is required). Specifies the number by which each element of a sequence is divided to generate the list of rests, must be integer >1.

`x0` Only used if *mode=virtual.*The parameter of the binomial distribution that will determine generation of the non-uniform selection of permutation. The default value is 0.5.

`random_state` Seed for the randomness. Should be a positive integer

Description

Generates a DPP sample by forming one of the stationary-1-dependent processes. The output sample is saved as SAS dataset in the directory specified by the user. There are three versions possible to use, specified by the *mode* parameter - *descent, carries, virtual.*

- ***mode=descent***
  Generates a sample by creating a descent process obtained from a uniformly chosen permutation of $1, .., size$. It is not possible to determine the size of the output sample (it will always be < size).

- ***mode=carries***
  Generates a sample by creating the sequence of Carries. A sequence of i.i.d. digits of a given size is generated and the cumulative sum is computed. The base parameter specifies the number by which each element of a sequence is divided to generate the rests (this parameter is required to specify by the user). It is not possible to determine the size of the output sample (it will always be <size).

- ***mode=virtual***
  Generates a sample from a mix of DPPs by obtaining a non-uniformly chosen permutation of $0, \ldots, size - 1$ (using binomial distribution with specified x0 parameter - 0.5 by default). It is not possible to determine the size of the output sample (it will always be < size).

Example

```
%stat_1_dep_sampler(
    size=100,
    dest=work.carries_sample,
    mode=carries,
    base=7,
    x0=.,
    random_state=123);
```