Dppsampl Package

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>

Description:

This package provides algorithms for sampling from the Determinantal Point Processes (DPPs) for SAS.

Version:

0.5

Overview:

The dppsampl package 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 package 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 dppsampl package, you must install it by using the `package install` statement. For example, if the ZIP file is located in the directory `C:\Packages`, then the following statement installs the package:

```
proc iml;
package install "C:\Packages\dppsampl.zip";
quit;
```

Modules:

- Finite DPPs:
  - `proj_dpp_sampler_kernel_Chol`
  - `proj_dpp_sampler_kernel_shur`
  - `proj_dpp_sampler_kernel_GS`
  - `proj_dpp_sampler_eig_GS`
  - `proj_dpp_sampl_ker_eig_KuTa12`
  - `sampler_generic`
  - `add_exchange_delete_sampler`
  - `add_delete_sampler`
  - `basic_exchange_sampler`
- Continuous DPPs:
  - `sample_from_beta_ensemble_full`
  - `sample_from_beta_ensemble_banded`
- Exotic DPPs:
  - `poiss_planch_sample`
  - `carries_sample`
  - `descent_sample`
  - `virtual_descent_sample`

Data Sets

We provide four test datasets which can be used to test the implementation of finite DPPs. All datasets have been constructed from the `k.sas7bdat` one.

- `k.sas7bdat` — correlation kernel which is a 10×10 projection matrix with rank of 8,
- `l.sas7bdat` — matrix created from `k.sas7bdat` using the relation $K = L(I + L)^{-1}$,
- `eig_vals.sas7bdat` — set of eigenvalues of `k.sas7bdat`,
- `eig_vals.sas7bdat` — set of eigenvectors of `k.sas7bdat`.

Available Finite DPP Functions:

proj_dpp_sampler_kernel_Chol Function

Syntax:

```
proj_dpp_sampler_kernel_Chol(
    kernel,
    size=.,
    random_state=.
);
```

Parameters

`kernel`	Projection correlation matrix K
`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

For given projection correlation kernel matrix (K), the function returns a sample of given size (if size is not given then the output sample will be of size=rank(kernel). The sample is calculated using Cholesky based method.

Example:

```
use data.k;
    read all into kernel;
close;
size=5;
random_state=123;

sample = proj_dpp_sampler_kernel_Chol(
    kernel,
    size,
    random_state
);
```

proj_dpp_sampler_kernel_shur Function

Syntax:

```
proj_dpp_sampler_kernel_shur(
    kernel,
    size=.,
    random_state=.
);
```

Parameters

`kernel`	Projection correlation matrix K
`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_seed`	Seed for the randomness. If provided it should be a positive integer

Description

For given projection correlation kernel matrix (K), the function returns a sample of given size (if size is not given then the output sample will be of size=rank(kernel). The sample is calculated using Shur based method.

Example:

```
use data.k;
    read all into kernel;
close;
size=5;
random_seed=123;

sample = proj_dpp_sampler_kernel_shur(
    kernel,
    size,
    random_seed
);
```

proj_dpp_sampler_kernel_GS Function

Syntax:

```
proj_dpp_sampler_kernel_GS(
    kernel,
    size=.,
    random_state=.
);
```

Parameters

`kernel`	Projection correlation matrix K
`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

For given projection correlation kernel matrix (K), the function returns a sample of given size (if size is not given then the output sample will be of size=rank(kernel). The sample is calculated using GS (Gram-Schmidt) based method.

Example:

```
use data.k;
    read all into kernel;
close;
size=5;
random_state=123;

sample = proj_dpp_sampler_kernel_GS(
    kernel,
    size,
    random_state
);
```

proj_dpp_sampler_eig_GS Function

Syntax:

```
proj_dpp_sampler_eig_GS(
    eig_vecs,
    size=.,
    random_state=.
);
```

Parameters

`eig_vecs`	Matrix of eigen vectors of a proper correlation kernel matrix.
`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

For given set of eigen vectors of a proper correlation kernel matrix, the function returns a sample of given size (if size is not given then the output sample will be of size=rank(kernel). The sample is calculated using GS (Gram-Schmidt) method.

Example:

```
use data.eig_vecs;
    read all into eig_vecs;
close;
size=5;
random_state=123;

sample = proj_dpp_sampler_eig_GS(
    eig_vecs,
    size,
    random_state
);
```

proj_dpp_sampl_ker_eig_KuTa12 Function

Syntax:

```
proj_dpp_sampl_ker_eig_KuTa12(
    eig_vecs,
    size=.,
    random_state=.
);
```

Parameters

eig_vecs	Matrix of eigen vectors of a proper correlation kernel matrix.
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

For a given set of eigen vectors of a proper correlation kernel matrix, the function returns a sample of given size (if size is not given then the output sample will be of size=rank(kernel). The sample is calculated using KuTa12 (Kulesza-Taskar) method.

Example:

```
use data.eig_vecs;
    read all into eig_vecs;
close;
size=5;
random_state=123;

sample = proj_dpp_sampl_ker_eig_KuTa12(
    eig_vecs,
    size,
    random_state
);
```

sampler_generic Function

Syntax:

```
sampler_generic(
    kernel,
    random_state=.
);
```

Parameters

`kernel`	Correlation kernel matrix K.
`random_state`	Seed for the randomness. If provided it should be a positive integer

Description

For a given correlation kernel matrix, the function returns a sample calculated with generic sampling algorithm. It is not possible to determine the size of the sample.

Example:

```
use data.k;
    read all into kernel;
close;
random_state=123;

sample = sampler_generic(
    kernel,
    random_state
);
```

add_exchange_delete_sampler Function

Syntax:

```
add_exchange_delete_sampler(
    kernel,
    s0=.,
    random_state=.,
    nb_iter=10
);
```

Parameters

`kernel`	Likelihood kernel matrix
`s0`	Initial sample from set of values represented by K. If none is provided then the algorithm will generate the initial sample.
`random_state`	Seed for the randomness. If provided it should be a positive integer
`nb_iter`	Number of iterations that the algorithm will perform
`nb_trials`	Number of trials for generating the initial sample

Description

For given likelihood matrix, the function 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 — s0 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:

```
use data.l;
    read all into kernel;
close;

sample = add_exchange_delete_sampler(kernel);
```

add_delete_sampler Function

Syntax:

```
add_delete_sampler(
    kernel,
    s0=.,
    size_s0=.,
    random_state=.,
    nb_iter=100,
    nb_trials=100
);
```

Parameters

`kernel`	Likelihood kernel matrix
`s0`	Initial sample from set of values represented by K. If none is provided then the algorithm will generate the initial sample.
`size_s0`	Expected size of the initial sample generated by the algorithm (provided the initial sample is not given)
`random_state`	Seed for the randomness. If provided it should be a positive integer
`nb_iter`	Number of iterations that the algorithm will perform.

Description

For given likelihood matrix, the function returns a sample calculated by adding, exchanging 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 — s0 is not provided, then it is generated by the algorithm. It is not possible to determine the size of the sample.

Example:

```
use data.l;
    read all into kernel;
close;

sample = add_delete_sampler(kernel);
```

basic_exchange_sampler Function

Syntax:

```
basic_exchange_sampler(
    kernel,
    s0=.,
    size_s0=.,
    random_state=.,
    nb_iter=100,
    nb_trials=100
);
```

Parameters

`kernel`	Likelihood kernel matrix
`s0`	Initial sample from set of values represented by K. If none is provided then the algorithm will generate the initial sample.
`size_s0`	Expected size of the initial sample generated by the algorithm (provided the initial sample is not given). In this case this will equal to the size of the output sample.
`random_state`	Seed for the randomness. If provided it should be a positive integer
`nb_iter`	Number of iterations that the algorithm will perform.

Description

For given likelihood matrix, the function returns a sample calculated by exchanging an element (with specific probability) from given samples (starting from initial s0 sample) for a number of iterations specified by the nb_iter parameter. If the initial sample — s0 is not provided, then it is generated by the algorithm . As the only operation performed during the algorithm is exchanging elements, the output sample will be of the same size as the initial s0 sample, thus it is possible to specify the size of the sample.

Example:

```
use data.l;
    read all into kernel;
close;

sample = basic_exchange_sampler(kernel);
```

Available Continuous DPP Functions:

sample_from_beta_ensemble_full Function

Syntax:

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

Parameters

`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_version`s. 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", "Laguerre", "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:

```
ensemble_version = "Circular";
size=10;
beta=4;
M_1=10; M_2 = 10;
haar_mode="Hermite";
normalize=0;
heurestic_fix=0;
random_state=1618;


sample = sample_from_beta_ensemble_full(
        ensemble_version,
        M_1, M_2,
        size,
        beta,
        normalize,
        haar_mode,
        heurestic_fix,
        random_state
);

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

sample_from_beta_ensemble_banded Function

Syntax

```
sample_from_beta_ensemble_banded(
    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

`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

```
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;


sample = sample_from_beta_ensemble_banded(
    ensemble_version,
    size,
    beta,
    loc,
    scale,
    shape,
    a, b,
    normalize,
    heurestic_fix,
    random_state
);

run histogram(sample);
```

Available Exotic DPP Functions:

poiss_planch_sample Function

Syntax

```
poiss_planch_sample(
    theta=10,
    random_state=.
);
```

Parameters

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

Example

```
theta = 10;
random_state = 123;


sample = poiss_planch_sample(
    theta,
    random_state);
```

carries_sample Function

Syntax

```
carries_sample(
    base=10,
     size=100,
    random_state=.
);
```

Parameters

`base`	The number by which each element of a sequence is divided to generate the list of rests, must be integer >1.
`size`	Size of the generated list, upper bound of output sample, must be integer >1.
`random_state`	Seed for the randomness. Should be a positive integer

Description

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. It is not possible to determine the size of the output sample (it will always be <size).

Example

```
base = 3;
size=20;
random_state = 123;


sample = carries_sample(
    base,
    size,
    random_state
);
```

descent_sample Function

Syntax

```
descent_sample(
     size=100,
    random_state=.
);
```

Parameters

`size`	Size of the generated list, upper bound of output sample, must be integer >1.
`random_state`	Seed for the randomness. Should be a positive integer

Description

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).

Example

```
size=20;
random_state = 123;


sample = descent_sample(
    size,
    random_state
);
```

virtual_descent_sample Function

Syntax

```
virtual_descent_sample(
    size=100,
    x0=0.5,
    random_state=.
);
```

Parameters

`size`	Size of the generated list, upper bound of output sample, must be integer >1.
`x0`	The parameter of the binomial distribution that will determine generation of the non-uniform selection of permutation.
`random_state`	Seed for the randomness. Should be a positive integer

Description

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). It is not possible to determine the size of the output sample (it will always be $<$ size).

Example

```
size=20;
x0=0.5;
random_state = 123;


sample = virtual_descent_sample(
    size,
    x0,
    random_state
);
```

Performance tests:

Tables from 1 to 6 show result of speed tests performed on SAS/IML and Python implementation. The performance was tested by generating 10000 sampls for each setting of sampler.

Size	SAS mean time [s]	Python mean time [s]	SAS/Python
10	0.000093	0.000431	0.22
50	0.001029	0.001754	0.59
100	0.003920	0.003954	0.99
200	0.053076	0.011193	4.74

Tabela 1: Performance test of Exact sampling: eig_GS

Size	SAS mean time [s]	Python mean time [s]	SAS/Python
10	0.000137	0.000460	0.30
50	0.000380	0.000550	0.69
100	0.000812	0.000674	1.20
200	0.002737	0.001061	2.58

Tabela 2: Performance test of MCMC sampling: AED

Size	SAS mean time [s]	Python mean time [s]	SAS/Python
10	0.000169	0.000155	1.09
20	0.000478	0.000183	2.61
30	0.001035	0.000229	4.52
40	0.002391	0.000300	7.96
50	0.003987	0.000395	10.09

Tabela 3: Performance test of Beta Full Ensemble: Hermite

Size	SAS mean time [s]	Python mean time [s]	SAS/Python
10	0.000063	0.000163	0.39
20	0.000107	0.000175	0.61
30	0.000190	0.000187	1.01
40	0.000221	0.000205	1.08
50	0.000286	0.000224	1.28

Tabela 4: Performance test of Beta Banded Ensemble: Hermite

Theta	SAS mean time [s]	Python mean time [s]	SAS/Python
10	0.000104	0.000133	0.78
50	0.001432	0.000240	5.95
100	0.004767	0.000395	12.07
200	0.016604	0.000747	22.22

Tabela 5: Performance test of Poissonized Plancherel

Size	SAS mean time [s]	Python mean time [s]	SAS/Python
10	0.000016	0.000124	0.13
50	0.000061	0.000193	0.32
100	0.000120	0.000287	0.42
200	0.000239	0.000481	0.50
300	0.000353	0.000644	0.55
400	0.000456	0.000828	0.57
500	0.000586	0.001021	0.58
800	0.000913	0.001583	0.61

Tabela 6: Performance test of Stationary 1-dependent processes: Descend