# PyGauss Documentation

***Release 0.1.1***

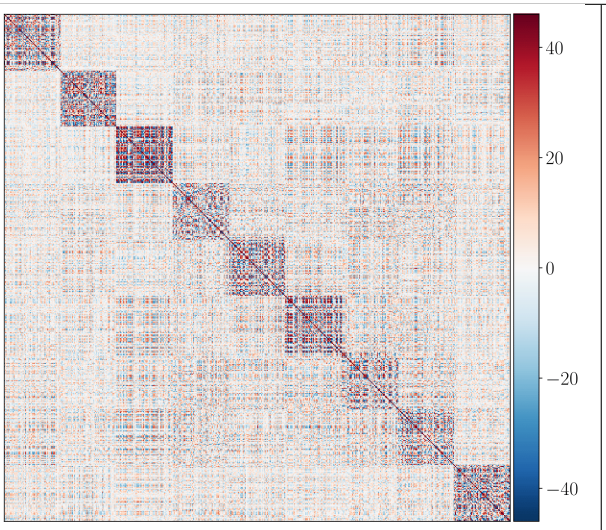
**Maxime Vono**

**Feb 05, 2022**

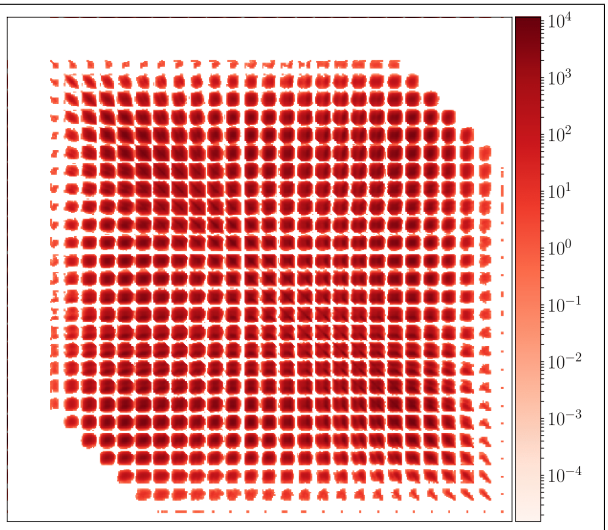
# Contents

As a contraction of **Python and Gaussian**, **PyGauss** is the companion package associated to the paper entitled *High-dimensional Gaussian sampling: A review and a unifying approach based on a stochastic proximal point algorithm* [1] which is publicy available on arXiv.

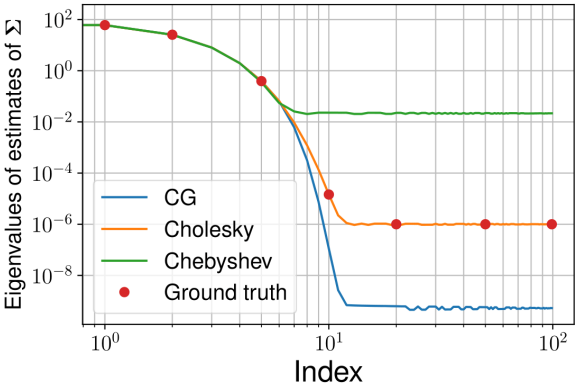
This package, written in PYTHON, aims at both reproducing the illustrations and experiments of [1] and providing the readers implementations of the Gaussian sampling approaches reviewed in [1].



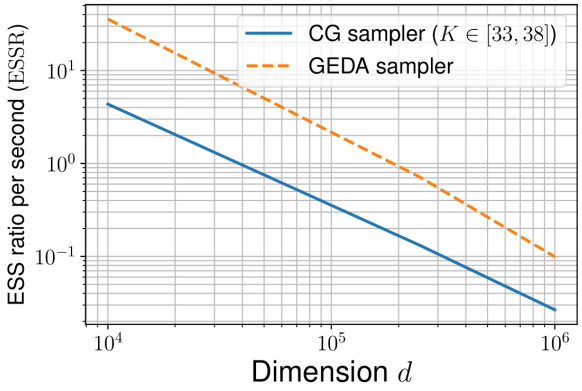

Precision matrix for Coepra dataset




Precision matrix for MNIST dataset




Eigenvalues of estimated covariance matrices




ESS ratio for two samplers

CHAPTER 1

# Installation instructions

See the installation instructions on GitHub.

CHAPTER 2

# Documentation contents

## 2.1 Direct sampling

### 2.1.1 Description

This Python module implements existing approaches, directly derived from numerical linear algebra, to sample from high-dimensional Gaussian probability distributions. The latter can be divided into three groups, namely:

- factorization approaches (e.g., Cholesky or square-root samplers),
- square-root approximation approaches (e.g., Chebyshev and Lanczos samplers),
- conjugate-gradient samplers.

For more details, we refer the interested reader to Section 3 of the companion paper.

### 2.1.2 API

Implementation of direct approaches to sample from multivariate Gaussian distributions.

**See also:**

Documentation on ReadTheDocs

pygauss.direct_sampling.**sampler_band**(*mu*, *A*, *b*, *mode='precision'*, *seed=None*, *size=1*)

Algorithm dedicated to sample from a multivariate real-valued Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A})$ or $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A}^{-1})$ when $\mathbf{A}$ is a band matrix.

**Parameters**

- **mu** (*1-D array_like, of length d*) – Mean of the d-dimensional Gaussian distribution.
- **A** (*2-D array_like, of shape (d, d)*) – Covariance or precision matrix of the distribution. It must be symmetric and positive-definite for proper sampling.
- **b** (*int*) – Bandwidth of A.

- **mode** (*string, optional*) – Indicates if A refers to the precision or covariance matrix of the Gaussian distribution.
- **seed** (*int, optional*) – Random seed to reproduce experimental results.
- **size** (*int, optional*) – Given a size of for instance T, T independent and identically distributed (i.i.d.) samples are returned.

**Returns** **theta** – The drawn samples, of shape (d,size), if that was provided. If not, the shape is (d,1).

**Return type** ndarray, of shape (d,size)

**Raises** `ValueError` – If mode is not included in ['covariance','precision'].

### Examples

```
>>> d = 2
>>> mu = np.zeros(d)
>>> A = np.eye(2)
>>> b = 0
>>> mode = "covariance"
>>> size = 1
>>> theta = sampler_band(mu,A,b,mode=mode,seed=2022,size=size)
```

pygauss.direct_sampling.**sampler_circulant**(*mu*, *a*, *M*, *N*, *mode='precision'*, *seed=None*, *size=1*)

Algorithm dedicated to sample from a multivariate real-valued Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A})$ or $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A}^{-1})$ when $\mathbf{A}$ is a block-circulant matrix with circulant blocks.

**Parameters**

- **mu** (*1-D array_like, of length d*) – Mean of the d-dimensional Gaussian distribution.
- **a** (*2-D array_like, of shape (N, M)*) – Vector built by stacking the first columns associated to the $M$ blocks of size $N$ of the matrix $\mathbf{A}$.
- **M** (*int*) – Number of different blocks.
- **N** (*int*) – Dimension of each block.
- **mode** (*string, optional*) – Indicates if A refers to the precision or covariance matrix of the Gaussian distribution.
- **seed** (*int, optional*) – Random seed to reproduce experimental results.
- **size** (*int, optional*) – Given a size of for instance T, T independent and identically distributed (i.i.d.) samples are returned.

**Returns** **theta** – The drawn samples, of shape (d,size), if that was provided. If not, the shape is (d,1).

**Return type** ndarray, of shape (d,size)

**Raises** `ValueError` – If mode is not included in ['covariance','precision'].

### Examples

```
>>> d = 2
>>> mu = np.zeros(d)
>>> a = np.matrix([1,0]).T
>>> M = 1
>>> N = 2
>>> mode = "covariance"
>>> size = 1
>>> theta = sampler_circulant(mu,a,M,N,mode=mode,seed=2022,size=size)
```

pygauss.direct_sampling.**sampler_factorization**(*mu*, *A*, *mode='precision'*, *method='Cholesky'*, *seed=None*, *size=1*)

Algorithm dedicated to sample from a multivariate real-valued Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A})$ or $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A}^{-1})$ based on matrix factorization (e.g., Cholesky or square root).

**Parameters**

- **mu** (*1-D array_like, of length d*) – Mean of the d-dimensional Gaussian distribution.
- **A** (*2-D array_like, of shape (d, d)*) – Covariance or precision matrix of the distribution. It must be symmetric and positive-definite for proper sampling.
- **mode** (*string, optional*) – Indicates if A refers to the precision or covariance matrix of the Gaussian distribution.
- **method** (*string, optional*) – Factorization method. Choose either 'Cholesky' or 'square-root'.
- **seed** (*int, optional*) – Random seed to reproduce experimental results.
- **size** (*int, optional*) – Given a size of for instance T, T independent and identically distributed (i.i.d.) samples are returned.

**Returns** **theta** – The drawn samples, of shape (d,size), if that was provided. If not, the shape is (d,1).

**Return type** ndarray, of shape (d,size)

**Raises** `ValueError` – If A is not positive definite and symmetric. If mode is not included in ['covariance','precision']. If method is not included in ['Cholesky','square-root'].

### Examples

```
>>> d = 2
>>> mu = np.zeros(d)
>>> A = np.eye(d)
>>> mode = "covariance"
>>> method = "Cholesky"
>>> size = 1
>>> theta = sampler_factorization(mu,A,mode=mode,method=method,seed=2022,
↪size=size)
```

pygauss.direct_sampling.**sampler_squareRootApprox**(*mu*, *A*, *lam_l*, *lam_u*, *tol*, *K=100*, *mode='precision'*, *seed=None*, *size=1*, *info=False*)

Algorithm dedicated to sample from a multivariate real-valued Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A})$ or $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A}^{-1})$ based on matrix square root approximation using Chebychev polynomials.

**Parameters**

- **mu** (*1-D array_like, of length d*) – Mean of the d-dimensional Gaussian distribution.
- **A** (*function*) – Linear operator returning the matrix-vector product $\mathbf{A}\mathbf{x}$ where $\mathbf{x}) \in \mathbb{R}^d$.
- **lam_l** (*float*) – Lower bound on the eigenvalues of $\mathbf{A}$.
- **lam_u** (*float*) – Upper bound on the eigenvalues of $\mathbf{A}$.
- **tol** (*float*) – Tolerance threshold used to optimize the polynomial order $K$. This threshold stands for the Euclidean distance between the vector computed using order $K$ and the one computed using order $L \leq K$.
- **K** (*int, optional*) – Polynomial order of the approximation.
- **mode** (*string, optional*) – Indicates if A refers to the precision or covariance matrix of the Gaussian distribution.
- **seed** (*int, optional*) – Random seed to reproduce experimental results.
- **size** (*int, optional*) – Given a size of for instance T, T independent and identically distributed (i.i.d.) samples are returned.
- **info** (*boolean, optional*) – If info is True, returns the order $K$ used in the polynomial approximation.

**Returns** **theta** – The drawn samples, of shape (d,size), if that was provided. If not, the shape is (d,1).

**Return type** ndarray, of shape (d,size)

**Raises** `ValueError` – If mode is not included in ['covariance','precision'].

### Examples

```
>>> d = 2
>>> mu = np.zeros(d)
>>> def A(x):
    return np.eye(d).dot(x)
>>> lam_l = 0
>>> lam_u = 1
>>> tol = 1e-4
>>> mode = "covariance"
>>> size = 1
>>> theta = sampler_squareRootApprox(mu,A,lam_l=lam_l,lam_u=lam_u,tol=tol,
mode=mode,seed=2022,size=size)
```

pygauss.direct_sampling.**sampler_CG**(*mu*, *A*, *K*, *init*, *tol=0.0001*, *mode='precision'*, *seed=None*, *size=1*, *info=False*)

Algorithm dedicated to sample from a multivariate real-valued Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A})$ or $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A}^{-1})$ based on the conjugate gradient algorithm.

**Parameters**

- **mu** (*1-D array_like, of length d*) – Mean of the d-dimensional Gaussian distribution.
- **A** (*function*) – Linear operator returning the matrix-vector product $\mathbf{A}\mathbf{x}$ where $\mathbf{x} \in \mathbb{R}^d$.
- **K** (*int, optional*) – Number of conjugate gradient iterations.
- **init** (*1-D array_like, of length d*) – Vector used to initialize the CG sampler.

- **tol** (*float, optional*) – Tolerance threshold used to stop the conjugate gradient sampler.
- **mode** (*string, optional*) – Indicates if A refers to the precision or covariance matrix of the Gaussian distribution.
- **seed** (*int, optional*) – Random seed to reproduce experimental results.
- **size** (*int, optional*) – Given a size of for instance T, T independent and identically distributed (i.i.d.) samples are returned.
- **info** (*boolean, optional*) – If info is True, returns the number of iterations $K$.

**Returns** **theta** – The drawn samples, of shape (d,size), if that was provided. If not, the shape is (d,1).

**Return type** ndarray, of shape (d,size)

**Raises** `ValueError` – If mode is not included in ['covariance','precision'].

### Examples

```
>>> d = 2
>>> mu = np.zeros(d)
>>> def A(x):
    return np.eye(d).dot(x)
>>> K = 2
>>> init = mu
>>> theta = sampler_CG(mu,A,K,init)
```

**class** pygauss.direct_sampling.**sampler_PO**(*mu1*, *mu2*, *K*, *init*, *tol=0.0001*, *seed=None*, *size=1*)

Algorithm dedicated to sample from a multivariate real-valued Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{Q}^{-1})$ where $\mathbf{Q}$ is a symmetric and positive definite precision matrix. We assume here that $\mathbf{Q} = \mathbf{G}_1^T \boldsymbol{\Lambda}_1^{-1} \mathbf{G}_1 + \mathbf{G}_2^T \boldsymbol{\Lambda}_2^{-1} \mathbf{G}_2$. The mean vector is assumed to have the form $\boldsymbol{\mu} = \mathbf{G}_1^T \boldsymbol{\Lambda}_1^{-1} \boldsymbol{\mu}_1 + \mathbf{G}_2^T \boldsymbol{\Lambda}_2^{-1} \boldsymbol{\mu}_2$. Sampling from the corresponding multivariate Gaussian distribution is done with the perturbation-optimization sampler.

**__init__**(*mu1*, *mu2*, *K*, *init*, *tol=0.0001*, *seed=None*, *size=1*)

**Parameters**

- **mu1** (*1-D array_like, of length d*) –
- **mu2** (*1-D array_like, of length d*) –
- **K** (*int, optional*) – Number of conjugate gradient iterations to solve the linear system $\mathbf{Q}\boldsymbol{\theta} = \boldsymbol{\eta}$.
- **init** (*1-D array_like, of length d*) – Vector used to initialize the CG algorithm.
- **tol** (*float, optional*) – Tolerance threshold used to stop the conjugate gradient algorithm.
- **seed** (*int, optional*) – Random seed to reproduce experimental results.
- **size** (*int, optional*) – Given a size of for instance T, T independent and identically distributed (i.i.d.) samples are returned.

**circu_diag_band**(*Lamb1*, *g*, *M*, *N*, *Q2*, *b2*)

We assume here that $\mathbf{G}_1$ is a circulant matrix, $\boldsymbol{\Lambda}_1$ is diagonal, $\mathbf{G}_2$ is the identity matrix and $\mathbf{Q}_2 = \boldsymbol{\Lambda}_2^{-1}$ is a band matrix.

**Parameters**

- **Lamb1** (*1-D array_like, of length d*) – Diagonal elements of $\mathbf{\Lambda}_1$.
- **g** (*2-D array_like, of shape (N, M)*) – Vector built by stacking the first columns associated to the $M$ blocks of size $N$ of the matrix $\mathbf{G}_1$.
- **M** (*int*) – Number of different blocks in $\mathbf{G}_1$.
- **N** (*int*) – Dimension of each block in $\mathbf{G}_1$.
- **Q2** (*2-D array_like, of shape (d, d)*) – Precision matrix $\mathbf{Q}_2$.
- **b2** (*int*) – Bandwidth of $\mathbf{Q}_2$.

**Returns** **theta** – The drawn samples, of shape (d,size), if that was provided. If not, the shape is (d,1).

**Return type** ndarray, of shape (d,size)

### Examples

```
>>> d = 15
>>> mu1 = np.zeros(d)
>>> mu2 = np.zeros(d)
>>> K = 15
>>> init = np.zeros(d)
>>> Lamb1 = np.random.normal(2,0.1,d)
>>> g =  np.reshape(np.random.normal(2,0.1,d),(d,1))
>>> M = 1
>>> N = d
>>> Q2 = np.diag(np.random.normal(2,0.1,d))
>>> b2 = 0
>>> size = 10000
>>> S = sampler_PO(mu1,mu2,K,init,size=10000)
>>> theta = S.circu_diag_band(Lamb1,g,M,N,Q21,b2)
```

## 2.2 MCMC sampling

### 2.2.1 Description

This Python module implements existing approaches, based on Markov chain Monte Carlo (MCMC) schemes, to sample from high-dimensional Gaussian probability distributions. The latter can be divided into two groups, namely:

- matrix splitting approaches,
- data augmentation approaches.

For more details, we refer the interested reader to Section 4 of the companion paper.

### 2.2.2 API

Implementation of Markov chain Monte Carlo (MCMC) approaches to sample from multivariate Gaussian distributions.

**See also:**

Documentation on ReadTheDocs

**class** pygauss.mcmc_sampling.**sampler_MS**(*mu*, *Q*, *ini*, *b*, *band=True*, *seed=None*, *size=1*)

Algorithm dedicated to sample from a multivariate real-valued Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{Q}^{-1})$ where $\mathbf{Q}$ is a symmetric and positive definite precision matrix. We assume here that the matrix splitting scheme $\mathbf{Q} = \mathbf{M} - \mathbf{N}$ holds.

**__init__**(*mu*, *Q*, *ini*, *b*, *band=True*, *seed=None*, *size=1*)

**Parameters**

- **mu** (*1-D array_like, of length d*) –
- **Q** (*2-D array_like, of shape (d,d)*) – Precision matrix.
- **ini** (*1-D array_like, of length d. Initialization of the Markov chain.*) –
- **b** (*int*) – Bandwidth of the precision matrix Q.
- **band** (*boolean, optional*) – Indicates if the precision matrix is banded with bandwidth b.
- **seed** (*int, optional*) – Random seed to reproduce experimental results.
- **size** (*int, optional*) – Given a size of for instance T, T independent and identically distributed (i.i.d.) samples are returned.

**exact_MS**(*method='Gauss-Seidel'*)

The samplers considered here are exact.

**Parameters** **method** (*string, optional*) – Matrix splitting approach to choose within ['Gauss-Seidel','Richardson','Jacobi','SOR','SSOR','Cheby-SSOR'].

**Returns** **theta** – The drawn samples, of shape (d,size), if that was provided. If not, the shape is (d,1).

**Return type** ndarray, of shape (d,size)

### Examples

```
>>> import mcmc_sampling as mcmc
>>> d = 10
>>> mu = np.zeros(d)
>>> ini = np.zeros(d)
>>> Q = np.eye(d)
>>> b = 1
>>> band = True
>>> S = mcmc.sampler_MS(mu,Q,ini=ini,b=b,band=True,seed=2022,size=1)
>>> theta = S.exact_MS(method="Gauss-Seidel")
```

**approx_MS**(*method='Clone-MCMC'*, *omega=1*)

The samplers considered here are approximate.

**Parameters**

- **method** (*string, optional*) – Matrix splitting approach to choose within ['Clone-MCMC','Hogwild'].
- **omega** (*float, optional*) – Tuning parameter appearing in some approximate matrix splitting Gibbs samplers.

**Returns** **theta** – The drawn samples, of shape (d,size), if that was provided. If not, the shape is (d,1).

**Return type** ndarray, of shape (d,size)

### Examples

```
>>> import mcmc_sampling as mcmc
>>> d = 10
>>> mu = np.zeros(d)
>>> ini = np.zeros(d)
>>> Q = np.eye(d)
>>> b = 1
>>> band = True
>>> S = mcmc.sampler_MS(mu,Q,ini=ini,b=b,band=True,seed=2022,size=1)
>>> theta = S.approx_MS(method="Gauss-Seidel",omega=1)
```

**class** pygauss.mcmc_sampling.**sampler_DA**(*mu*, *seed=None*, *size=1*)

Algorithm dedicated to sample from a multivariate real-valued Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{Q}^{-1})$ where $\mathbf{Q}$ is a symmetric and positive definite precision matrix. We assume here that $\mathbf{Q} = \mathbf{G}_1^T \boldsymbol{\Lambda}_1^{-1} \mathbf{G}_1 + \mathbf{G}_2^T \boldsymbol{\Lambda}_2^{-1} \mathbf{G}_2$. Sampling from the corresponding multivariate Gaussian distribution is done with an MCMC algorithm based on a data augmentation scheme.

**__init__**(*mu*, *seed=None*, *size=1*)

**Parameters**

- **mu** (*1-D array_like, of length d*) –
- **seed** (*int, optional*) – Random seed to reproduce experimental results.
- **size** (*int, optional*) – Given a size of for instance T, T independent and identically distributed (i.i.d.) samples are returned.

**exact_DA_circu_diag_band**(*Lamb1*, *g*, *M*, *N*, *Q2*, *b2*, *method='GEDA'*)

The samplers considered here are exact. We further assume here that $\mathbf{G}_1$ is a circulant matrix, $\boldsymbol{\Lambda}_1$ is diagonal, $\mathbf{G}_2$ is the identity matrix and $\mathbf{Q}_2 = \boldsymbol{\Lambda}_2^{-1}$ is a band matrix.

**Parameters**

- **Lamb1** (*1-D array_like, of length d*) – Diagonal elements of $\boldsymbol{\Lambda}_1$.
- **g** (*2-D array_like, of shape (N, M)*) – Vector built by stacking the first columns associated to the $M$ blocks of size $N$ of the matrix $\mathbf{G}_1$.
- **M** (*int*) – Number of different blocks in $\mathbf{G}_1$.
- **N** (*int*) – Dimension of each block in $\mathbf{G}_1$.
- **Q2** (*2-D array_like, of shape (d, d)*) – Precision matrix $\mathbf{Q}_2$.
- **b2** (*int*) – Bandwidth of $\mathbf{Q}_2$.
- **method** (*string, optional*) – Data augmentation approach to choose within ['EDA','GEDA'].

**Returns** **theta** – The drawn samples, of shape (d,size), if that was provided. If not, the shape is (d,1).

**Return type** ndarray, of shape (d,size)

### Examples

```
>>> import mcmc_sampling as mcmc
>>> d = 15
>>> Lamb1 = np.random.normal(2,0.1,d)
>>> g =  np.reshape(np.random.normal(2,0.1,d),(d,1))
>>> M = 1
>>> N = d
>>> Q2 = np.diag(np.random.normal(2,0.1,d))
>>> b2 = 0
>>> S = mcmc.sampler_DA(mu,seed=2022,size=1)
>>> theta = S.exact_DA_circu_diag_band(Lamb1,g,M,N,
                                       Q2,b2,method="EDA")
```

**exact_DA_circu_diag_circu** (*Lamb1*, *LambG1*, *LambG2*, *A*, *method='GEDA'*)

The samplers considered here are exact. We further assume here that $\mathbf{G}_1$ is a circulant matrix, $\mathbf{\Lambda}_1$ is diagonal, $\mathbf{\Lambda}_2$ is the identity matrix and $\mathbf{G}_2$ is a circulant matrix.

**Parameters**

- **Lamb1** (*1-D array_like, of length d*) – Diagonal elements of $\mathbf{\Lambda}_1$.
- **LambG1** (*1-D array_like, of length d*) – Diagonal elements of the Fourier counterpart matrix associated to the matrix $\mathbf{G}_1$.
- **LambG2** (*1-D array_like, of length d*) – Diagonal elements of the Fourier counterpart matrix associated to the matrix $\mathbf{G}_2$.
- **A** (*function*) – Linear operator returning the matrix-vector product $\mathbf{Qx}$ where $\mathbf{x} \in \mathbb{R}^d$.
- **method** (*string, optional*) – Data augmentation approach to choose within ['EDA','GEDA'].

**Returns** **theta** – The drawn samples, of shape (d,size), if that was provided. If not, the shape is (d,1).

**Return type** ndarray, of shape (d,size)

### Examples

```
>>> import mcmc_sampling as mcmc
>>> d = 15
>>> Lamb1 = np.random.normal(2,0.1,d)
>>> g =  np.reshape(np.random.normal(2,0.1,d),(d,1))
>>> M = 1
>>> N = d
>>> Q2 = np.diag(np.random.normal(2,0.1,d))
>>> b2 = 0
>>> S = mcmc.sampler_DA(mu,seed=2022,size=1)
>>> theta = S.exact_DA_circu_diag_band(Lamb1,g,M,N,
                                       Q2,b2,method="EDA")
```

# Python Module Index

## p

# Index

## Symbols



## A



## C



## E



## P



## S