EuroHPC
Joint Undertaking

Data driven Computational Mechanics at EXascale

DCoMEX

Data driven Computational Mechanics at EXascale

Work program topic: EuroHPC-01-2019
Type of action: Research and Innovation Action (RIA)

Report on DMAP algorithm prototype

DELIVERABLE D2.1

Version No 1

http://www.dcomex.eu/
This project has received funding from the European High-Performance Computing Joint Undertaking Joint Undertaking ('the JU'), under Grant Agreement No 956201

DOCUMENT SUMMARY INFORMATION

Project Title	Data driven Computational Mechanics at EXascale
Project Acronym	DCoMEX
Project No:	956201
Call Identifier:	EuroHPC-01-2019
Project Start Date	01/04/2021
Related work package	WP 2
Related task(s)	Task 2.1
Lead Organisation	NTUA
Submission date	24/01/2022
Re-submission date	
Dissemination Level	PU

Quality Control:

	Who	Affiliation	Date
Checked by internal reviewer	George Stavroulakis	NTUA	22/01/2022
Checked by WP Leader	Vissarion Papadopoulos	NTUA	22/01/2022
Checked by Project Coordinator	Vissarion Papadopoulos	NTUA	22/01/2022

Document Change History:

Version	Date	Author (s)	Affiliation	Comment
1.0	22.01.2022	Ioannis Kalogeris	ETHZ	

Description

Deliverable 2.1 illustrates the theoretical background preceding the development of the Diffusion Maps algorithm. An algorithmic implementation at a prototyping level is available at:
https://github.com/mgroupntua/MSolve.MachineLearning[1]

The Diffusion maps algorithm

Let $\boldsymbol{U} = [\boldsymbol{u_1}, \cdots, \boldsymbol{u_N}]$ be a data set consisting of vectors $\boldsymbol{u_i} \in R^d$, which can be seen as N distinct realizations of an R^d-valued random variable and sampled independently with density $q(\boldsymbol{u})$. Next, assume a connectivity measure K between data pairs $\boldsymbol{u}_i, \boldsymbol{u}_j$ such as the Gaussian kernel

$$K_\epsilon(\boldsymbol{u_i}, \boldsymbol{u_j}) = exp\left(\frac{-\left(\|\boldsymbol{u_i} - \boldsymbol{u_j}\|^2\right)}{4\varepsilon}\right)$$

Next, a discrete approximation to the Laplacian L_ε is constructed as follows:

- Estimate the densities q_ε at the sample points $\boldsymbol{u}_i$ as

$$q_\varepsilon(\boldsymbol{u}_i) = \frac{1}{N}\sum_{j=1}^{N} K_\varepsilon(\boldsymbol{u}_i, \boldsymbol{u}_j)$$

- Normalize the previously defined kernel K_ε as

$$\widehat{K_\varepsilon}(\boldsymbol{u}_i, \boldsymbol{u}_j) = \frac{K_\varepsilon(\boldsymbol{u}_i, \boldsymbol{u}_j)}{q_\varepsilon(\boldsymbol{u}_i)^\alpha q_\varepsilon(\boldsymbol{u}_j)^\alpha}$$

  Where for $\alpha = 1$ the discrete Laplacian approximates the Laplace-Beltrami operator, while $\alpha = 1/2$ approximates a diffusion operator.

- Estimate the new densities

$$\widehat{q_\varepsilon}(\boldsymbol{u}_i) = \frac{1}{N}\sum_{j=1}^{N} \widehat{K_\varepsilon}(\boldsymbol{u}_i, \boldsymbol{u}_j)$$

- If we define the matrix $\boldsymbol{K} = [K_{ij}] = \widehat{K_\varepsilon}(\boldsymbol{u}_i, \boldsymbol{u}_j)$ and the diagonal matrix $\boldsymbol{D} = [D_{ii}] = q_\varepsilon(\boldsymbol{u}_i)$, then the discrete approximation of the weighted Laplacian is given by the expression:

$$\boldsymbol{L}_\varepsilon = \frac{\boldsymbol{D}^{-1}\boldsymbol{K} - \boldsymbol{I_N}}{\varepsilon}$$

The solution to the eigenvalue problem $\boldsymbol{L}_\varepsilon\boldsymbol{\psi} = \lambda\boldsymbol{\psi}$ will produce the sequence of eigenvalues $0 = \lambda_0 \geq \lambda_1 \geq \lambda_2 \geq \cdots$ and right eigenvectors $\boldsymbol{\psi}_j$ for the operator. In practice, only the first n non-trivial eigenvectors are kept with n obtained from the expression

$$n = argmin_{n,n\geq 2}\left(\frac{\lambda_1}{\lambda_n} < tol\right)$$

[1] The code has originally been submitted in the repo https://github.com/YiannisKalogeris/MSolve.MachineLearning

Then, the diffusion map operator $\Psi_\varepsilon: u \to R^n$can be defined as

$$\Psi_\varepsilon(\boldsymbol{u}) = \left[e^{\lambda_1 \varepsilon}\psi_1(\boldsymbol{u}), e^{\lambda_2 \varepsilon}\psi_2(\boldsymbol{u}), \dots, e^{\lambda_n \varepsilon}\psi_n(\boldsymbol{u})\right]$$

Diffusion Maps with variable-bandwidth kernels

In several data-driven applications, the samples follow some distribution which is unknown a priori. It is expected that the samples belonging to the tails of the distribution will be fewer and, thus, there will be regions on the manifold that will be more sparsely delineated. To address this issue in classical kernel methods the idea of the variable-bandwidth (or self-tuning) kernels has been proposed and illustrated herein. The main differentiation with respect to the classical DMAP algorithm lies in the form of the kernel used, which in this setting becomes:

$$K_\varepsilon^{VB}(\boldsymbol{u_i}, \boldsymbol{u_j}) = exp\left(\frac{-\left(\|\boldsymbol{u_i} - \boldsymbol{u_j}\|^2\right)}{\boldsymbol{4}\varepsilon\rho(\boldsymbol{u_i})\rho(\boldsymbol{u_j})}\right)$$

Following the construction for the graph Laplacian of the previous sections, in this case the sample densities are

$$q_\varepsilon^{VB}(\boldsymbol{u}_i) = \sum_{j=1}^{N} \frac{K_\varepsilon(\boldsymbol{u}_i, \boldsymbol{u}_j)}{\rho(\boldsymbol{u}_i)^m}$$

which are used to construct the kernel

$$K_{\varepsilon,\alpha}^{VB}(\boldsymbol{u}_i, \boldsymbol{u_j}) = \frac{K_\varepsilon^{VB}(\boldsymbol{u}_i, \boldsymbol{u}_j)}{q_\varepsilon^{VB}(\boldsymbol{u}_i)^\alpha q_\varepsilon^{VB}(\boldsymbol{u}_j)^\alpha}$$

Setting $q_{\varepsilon,\alpha}^{VB}(\boldsymbol{u}_i) = \sum_{j=1}^{N} K_{\varepsilon,\alpha}^{VB}(\boldsymbol{u}_i, \boldsymbol{u}_j)$, we can obtain the normalized kernel

$$\widehat{K_{\varepsilon,\alpha}^{VB}}(\boldsymbol{u}_i, \boldsymbol{u}_j) = \frac{\widehat{K_{\varepsilon,\alpha}^{VB}}(\boldsymbol{u}_i, \boldsymbol{u}_j)}{q_{\varepsilon,\alpha}^{VB}(u_i)}$$

and the weighted Laplacian for this formulation becomes

$$L_{\varepsilon,\alpha}^{VB}(\boldsymbol{u}_i, \boldsymbol{u}_j) = \frac{\widehat{K_{\varepsilon,\alpha}^{VB}}(\boldsymbol{u}_i, \boldsymbol{u}_j) - \delta_{ij}}{\varepsilon\rho(\boldsymbol{u}_i)^2}$$

Algorithmic implementation in the MSolve software

The code for implementing the variable-bandwidth diffusion maps algorithm can be found in https://github.com/mgroupntua/MSolve.MachineLearning [2]. In particular, the C# class DiffusionMapsAlgorithm.cs in the MGroup.MachineLearning folder implements the aforementioned procedure for an input data set. An example illustrating the use of this class is provided in the MGroup.MachineLearning.Tests folder, called DMAPexample.cs.

In this particular example, an initial data set is considered which consists of 2000 points in R^2, generated from a 2-dimensional Gaussian distribution centered at zero with covariance $C = 0.04I_2$. Using the syntax outlined below, a new object called DMAP from the DiffusionMapsAlgorithm class is generated, taking as input from the user a specified set of variables. Then the method ProcessData() applies the DMAP algorithm and computes the member variables DMAP.DMAPeigenvalues[·] and DMAP.DMAPeigenvalues[·].

- `dataSet` : the initial data set
- `numberOfKNN`: number of k-nearest neighbors used in the evaluation of the kernel $K_{\varepsilon,\alpha}^{VB}(\boldsymbol{u}_i, \boldsymbol{u}_j)$
- `numberOfKDE`: number of k-nearest neighbors required to estimate the kernel parameter ε
- `differentialOperator`: 1 – Laplace Beltrami operator, 2- generator of grad systems
- `numberOfEigenvectors`: The number of eigenvectors requested by the user

```
DiffusionMapsAlgorithm DMAP = new DiffusionMapsAlgorithm(dataSet,
numberOfKNN, NNofKDE, differentialOperator, numberOfEigenvectors);

DMAP.ProcessData();
```

The data used in this particular example are shown in figure 1, while figure 2 depicts the first 10 non-trivial DMAP eigenvalues.

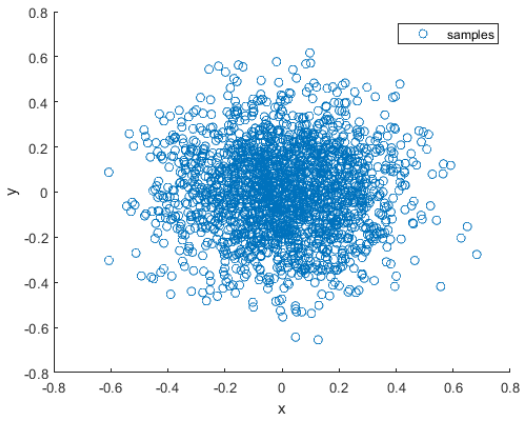

Figure 1: initial data samples

[2] The code has originally been submitted in the repo https://github.com/YiannisKalogeris/MSolve.MachineLearning

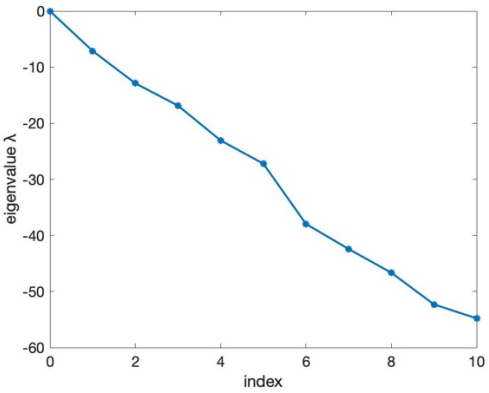

Figure 2: The first 10 diffusion map eigenvalues