eMaTe Documentation

Release 1.0.3

Bruno Messias; Thomas Peron

Jul 19, 2021

Contents

CHAPTER 1

eMaTe

eMaTe is a python package implemented in tensorflow which the main goal is provide useful methods capable of estimate spectral densities and trace functions of large sparse matrices.

The package is available as a pypi repository

```
$ pip install emate
```

The source code is available at <http://github.com/stdogpkg>‘.

The eMaTe package it is being built to provide our another package, stDoG (strucutre and dynamics on graphs), methods to deal with graphs with a huge amount of vertices. To install stDoG, just type in our terminal

```
$ pip install stdog
```

1.1 Install

The package is available as as pypi repository

```
$ pip install emate
```

The source code is available at <http://github.com/stdogpkg>‘.

1.2 Examples

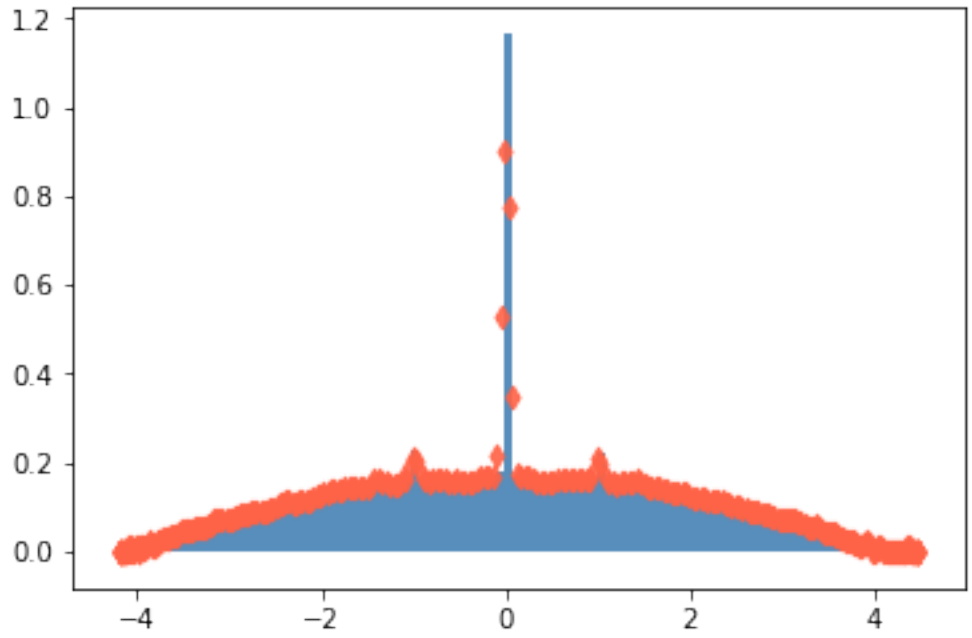
1.2.1 Kernel Polynomial Method (Chebyshev Polynomial expansion)

The Kernel Polynomial Method can estimate the spectral density of large sparse Hermitan matrices with a low computational cost. This method combines three key ingredients: the Chebyshev expansion + the stochastic trace estimator + kernel smoothing.

```
import networkx as nx
import numpy as np

n = 3000
g = nx.erdos_renyi_graph(n , 3/n)
W = nx.adjacency_matrix(g)

vals  = np.linalg.eigvals(W.todense()).real
```

```
from emate.hermitian import tfkpm


num_moments = 300
num_vecs = 200
extra_points = 10
ek, rho = tfkpm(W, num_moments, num_vecs, extra_points)
```

```
import matplotlib.pyplot as plt
plt.hist(vals, density=True, bins=100, alpha=.9, color="steelblue")
plt.scatter(ek, rho, c="tomato", zorder=999, alpha=0.9, marker="d")
plt.ylim(0, 1)
plt.show()
```



References

[1] Wang, L.W., 1994. Calculating the density of states and optical-absorption spectra of large quantum systems by the plane-wave moments method. Physical Review B, 49(15), p.10154.

[2] Hutchinson, M.F., 1990. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. Communications in Statistics-Simulation and Computation, 19(2), pp.433-450.

1.2.2 Sthocastic Lanczos Quadrature

Given a semi-positive definite matrix $A \in \mathbb{R}^{|V|\times|V|}$, which has the set of eigenvalues given by $\{\lambda_i\}$ a trace of a matrix function is given by

$$\mathrm{tr}(f(A)) = \sum_{i=0}^{|V|} f(\lambda_i)$$

The methods for calculating such traces functions have a cubic computational complexity lower bound, $O(|V|^3)$. Therefore, it is not feasible for large networks. One way to overcome such computational complexity it is use stochastic approximations combined with a mryiad of another methods to get the results with enough accuracy and with a small computational cost. The methods available in this module uses the Sthocastic Lanczos Quadrature, a procedure proposed in the work made by Ubaru, S. et.al. [1] (you need to cite them).

Estrada Index

```
import scipy
import scipy.sparse
import numpy as np
```

```
from emate.symmetric.slq import pyslq
import tensorflow as tf

def trace_function(eig_vals):
    return tf.exp(eig_vals)

num_vecs = 100
num_steps = 50
approximated_estrada_index, _ = pyslq(L_sparse, num_vecs, num_steps,  trace_function)
exact_estrada_index =  np.sum(np.exp(vals_laplacian))
approximated_estrada_index, exact_estrada_index
```

The above code returns

```
(3058.012, 3063.16457163222)
```

References

1 - Ubaru, S., Chen, J., & Saad, Y. (2017). Fast Estimation of tr(f(A)) via Stochastic Lanczos Quadrature. SIAM Journal on Matrix Analysis and Applications, 38(4), 1075-1099.

2 - Hutchinson, M. F. (1990). A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. Communications in Statistics-Simulation and Computation, 19(2), 433-450.

1.3 Symmetric

1.3.1 Sthocastic Lanczos Quadrature

Given a semi-positive definite matrix $A \in \mathbb{R}^{|V|\times|V|}$, which has the set of eigenvalues given by $\{\lambda_i\}$ a trace of a matrix function is given by

$$\mathrm{tr}(f(A)) = \sum_{i=0}^{|V|} f(\lambda_i)$$

The methods for calculating such traces functions have a cubic computational complexity lower bound, $O(|V|^3)$. Therefore, it is not feasible for large networks. One way to overcome such computational complexity it is use stochastic approximations combined with a mryiad of another methods to get the results with enough accuracy and with a small computational cost. The methods available in this module uses the Sthocastic Lanczos Quadrature, a procedure proposed in the work made by Ubaru, S. et.al. [1] (you need to cite them).

References

[1] Ubaru, S., Chen, J., & Saad, Y. (2017). Fast Estimation of tr(f(A)) via Stochastic Lanczos Quadrature. SIAM Journal on Matrix Analysis and Applications, 38(4), 1075-1099.

[2] Hutchinson, M. F. (1990). A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. Communications in Statistics-Simulation and Computation, 19(2), 433-450.

emate.symmetric.slq.**pyslq**(*A*, *num_vecs*, *num_steps*, *trace_function*, *device='/gpu:0'*, *precision=32*, *random_factory=<function radamacher>*, *parallel_iterations=10*, *swap_memory=False*, *infer_shape=False*)

Compute the approxiamted value of a given trace function using the sthocastic Lanczos quadrature using

Radamacher's random vectors.

Parameters

- **A** (*scipy sparse matrix*) – The semi-positive definite matrix
- **num_vecs** (*int*) – Number of random vectors in oder to aproximate the trace
- **num_steps** (*int*) – Number of Lanczos steps
- **trace_function** (*function*) – A function like

```
def trace_function(eig_vals)
    *tensorflow ops
    return result
```

- **precision** (*int*) –

(32) Single or (64) double precision

Returns

- **f_estimation** (*float*) – The approximated value of the given trace function
- **gammas** (*array of floats*) – See [1] for more details

References

[1] Ubaru, S., Chen, J., & Saad, Y. (2017). Fast Estimation of tr(f(A)) via Stochastic Lanczos Quadrature. SIAM Journal on Matrix Analysis and Applications, 38(4), 1075-1099.

1.4 Hermitian

1.4.1 Kernel Polynomial Method

The kernel polynomial method is an algorithm to obtain an approximation for the spectral density of a Hermitian matrix. This algorithm combines expansion in polynomials of Chebyshev [1], the stochastic trace [2] and a kernel smothing techinique in order to obtain the approximation for the spectral density

Applications

- Hamiltonian matrices associated with quantum mechanics
- Laplacian matrix associated with a graph
- Magnetic Laplacian associated with directed graphs
- etc

References

[1] Wang, L.W., 1994. Calculating the density of states and optical-absorption spectra of large quantum systems by the plane-wave moments method. Physical Review B, 49(15), p.10154.

[2] Hutchinson, M.F., 1990. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. Communications in Statistics-Simulation and Computation, 19(2), pp.433-450.

emate.hermitian.kpm.**pykpm**(*H*, *num_moments=10*, *num_vecs=10*, *extra_points=2*, *precision=32*, *lmin=None*, *lmax=None*, *epsilon=0.01*, *device='/gpu:0'*, *swap_memory_while=False*)

Kernel Polynomial Method using a Jackson's kernel.

Parameters

- **H** (*scipy sparse matrix*) – The Hermitian matrix
- **num_moments** (*int*) –
- **num_vecs** (*int*) – Number of random vectors in oder to aproximate the trace
- **extra_points** (*int*) –
- **precision** (*int*) – Single or double precision
- **limin** (*float, optional*) – The smallest eigenvalue
- **lmax** (*float*) – The highest eigenvalue
- **epsilon** (*float*) – Used to rescale the matrix eigenvalues into the interval [-1, 1]
- **device** (*str*) – '/gpu:ID' or '/cpu:ID'

Returns

- **ek** (*array of floats*) – An array with num_moments + extra_points approximated "eigenvalues"
- **rho** (*array of floats*) – An array containing the densities of each "eigenvalue"

emate.hermitian.kpm.**cupykpm**(*H*, *num_moments=10*, *num_vecs=10*, *extra_points=12*, *precision=32*, *lmin=None*, *lmax=None*, *epsilon=0.01*)

Kernel Polynomial Method using a Jackson's kernel. CUPY version

Parameters

- **H** (*scipy CSR sparse matrix*) – The Hermitian matrix
- **num_moments** (*int*) –
- **num_vecs** (*int*) – Number of random vectors in oder to aproximate the trace
- **extra_points** (*int*) –
- **precision** (*int*) – Single or double precision
- **limin** (*float, optional*) – The smallest eigenvalue
- **lmax** (*float*) – The highest eigenvalue
- **epsilon** (*float*) – Used to rescale the matrix eigenvalues into the interval [-1, 1]

Returns

- **ek** (*array of floats*) – An array with num_moments + extra_points approximated "eigenvalues"
- **rho** (*array of floats*) – An array containing the densities of each "eigenvalue"

emate.hermitian.kpm.**tfkpm**(*H*, *num_moments=10*, *num_vecs=10*, *extra_points=2*, *precision=32*, *lmin=None*, *lmax=None*, *epsilon=0.01*, *device='/gpu:0'*, *swap_memory_while=False*)

Kernel Polynomial Method using a Jackson's kernel.

Parameters

- **H** (*scipy sparse matrix*) – The Hermitian matrix

- **num_moments** (*int*) –
- **num_vecs** (*int*) – Number of random vectors in oder to aproximate the trace
- **extra_points** (*int*) –
- **precision** (*int*) – Single or double precision
- **limin** (*float, optional*) – The smallest eigenvalue
- **lmax** (*float*) – The highest eigenvalue
- **epsilon** (*float*) – Used to rescale the matrix eigenvalues into the interval [-1, 1]
- **device** (*str*) – '/gpu:ID' or '/cpu:ID'

Returns

- **ek** (*array of floats*) – An array with num_moments + extra_points approximated "eigenvalues"
- **rho** (*array of floats*) – An array containing the densities of each "eigenvalue"

Python Module Index

e

Index