StDoG Documentation

Release 1.0.3

Bruno Messias; Thomas Peron

Sep 18, 2019

Contents

CHAPTER 1

Strucutre and Dyanmics on Graphs

The main goal of StDoG is to provide a package which can be used to study dynamical and structural properties (like spectra) on graphs with a large number of vertices. The modules of StDoG are being built by combining codes written in *Tensorflow* + *CUDA* and *C++*.

The package is available as a pypi repository

```
$ pip install stdog
```

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

1.1 Install

The package is available as as pypi repository

```
$ pip install stdog
```

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

1.2 Examples

1.2.1 Dynamics

Kuramoto

StDoG provides two implementations of Heun's method. The first uses the TensorFlow. Therefore, it can be used with both CPU's or GPU's. The second implementation is a python wrapper to a CUDA code. The second (CUDA) is faster than TensorFlow implementation. However, a CUDA-compatible GPU is required

Creating the data and setting the variables

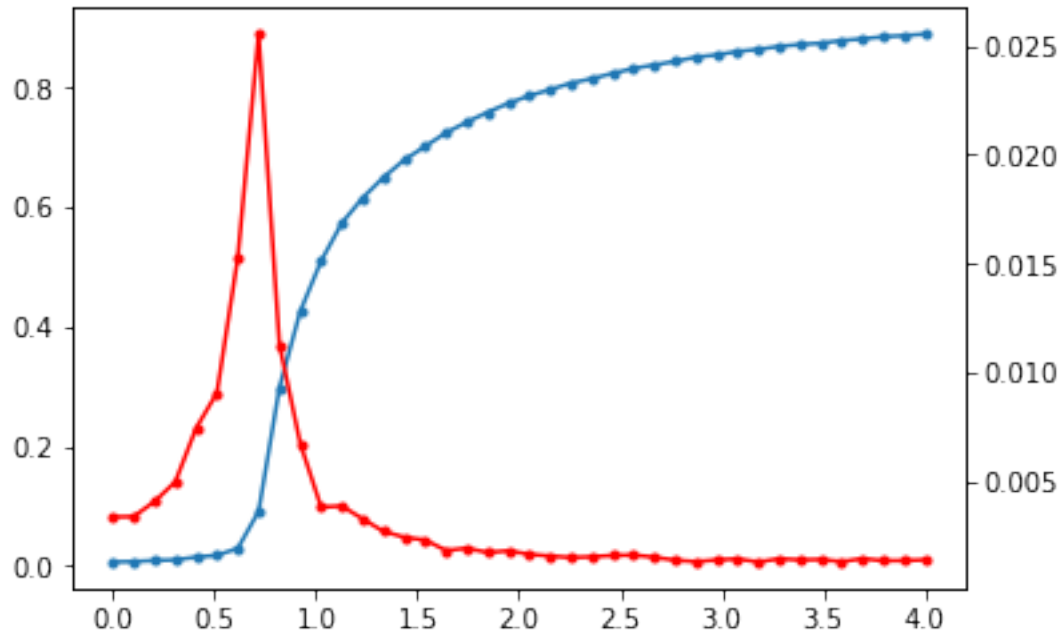
```
import numpy as np
import igraph as ig
from stdog.utils.misc import ig2sparse
num_couplings = 40
N = 20480

G = ig.Graph.Erdos_Renyi(N, 3/N)
adj = ig2sparse(G)

omegas = np.random.normal(size= N).astype("float32")
couplings = np.linspace(0.0,4.,num_couplings)
phases =  np.array([
    np.random.uniform(-np.pi,np.pi,N)
    for i_l in range(num_couplings)

],dtype=np.float32)

precision =32

dt = 0.01
num_temps = 50000
total_time = dt*num_temps
total_time_transient = total_time
transient = False
```

Tensorflow implementation

```
from stdog.dynamics.kuramoto import Heuns

heuns_0 = Heuns(adj, phases, omegas, couplings, total_time, dt,
    device="/gpu:0", # or /cpu:
    precision=precision, transient=transient)

heuns_0.run()
heuns_0.transient = True
heuns_0.total_time = total_time_transient
heuns_0.run()
order_parameter_list = heuns_0.order_parameter_list
```

Plotting the result

```
import matplotlib.pyplot as plt

r = np.mean(order_parameter_list, axis=1)
stdr = np.std(order_parameter_list, axis=1)

plt.ion()
fig, ax1 = plt.subplots()
ax1.plot(couplings,r,'.-')
ax2 = ax1.twinx()
ax2.plot(couplings,stdr,'r.-')
plt.show()
```



CUDA implementation (faster)

For that, you need to install our another package, *cukuramoto http://github.com/stdogpkg/cukuramoto*

```
$ pip install cukuramoto
```

```
from stdog.dynamics.kuramoto.cuheuns import CUHeuns as cuHeuns

heuns_0 = cuHeuns(adj, phases, omegas,  couplings,
    total_time, dt, block_size = 1024, transient = False)

heuns_0.run()

heuns_0.transient = True
heuns_0.total_time = total_time_transient
heuns_0.run()
order_parameter_list = heuns_0.order_parameter_list
```

References

[1] - Thomas Peron, Bruno Messias, Angélica S. Mata, Francisco A. Rodrigues, and Yamir Moreno. On the onset of synchronization of Kuramoto oscillators in scale-free networks. arXiv:1905.02256 (2019).

1.2.2 Spectra

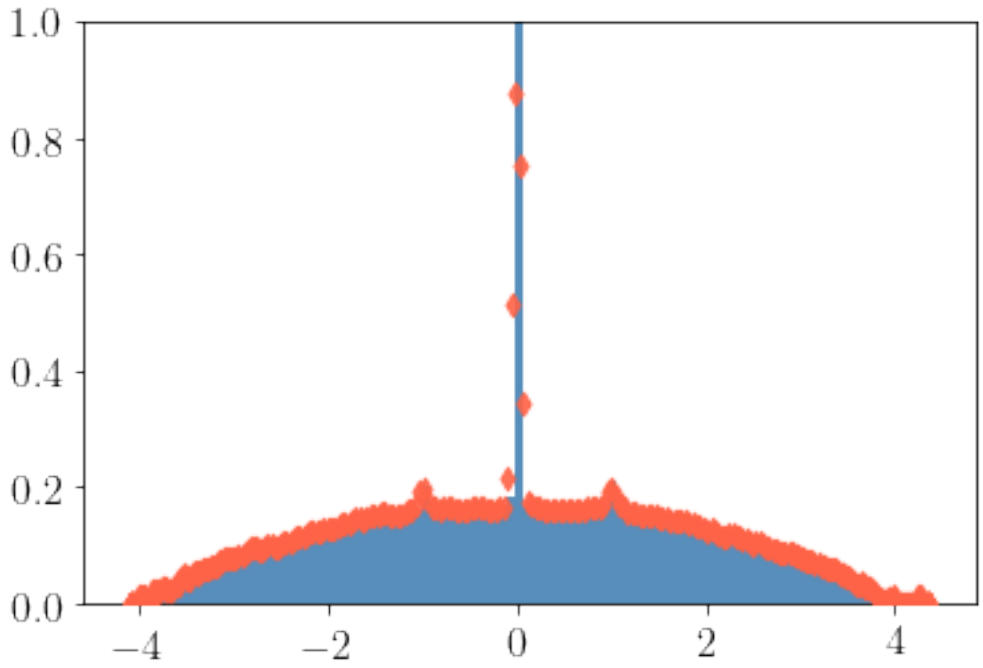
Spectral Density

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

```
import igraph as ig
import numpy as np

N = 3000
G = ig.Graph.Erdos_Renyi(N, 3/N)

W = np.array(G.get_adjacency().data, dtype=np.float64)
vals = np.linalg.eigvalsh(W).real
```

```
import stdog.spectra as spectra
from stdog.utils.misc import ig2sparse

W = ig2sparse(G)
num_moments = 300
num_vecs = 200
extra_points = 10
ek, rho = spectra.dos.kpm(W, num_moments, num_vecs, extra_points, device="/gpu:0")
```

```
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.

Trace Functions

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).

Spectral Entropy

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

(continues on next page)

(continued from previous page)

```
N = 3000
G = ig.Graph.Erdos_Renyi(N, 3/N)
```

```
from stdog.spectra.trace_function import entropy as slq_entropy

def entropy(eig_vals):
s = 0.
for val in eig_vals:
    if val > 0:
    s += -val*np.log(val)
return s

L = np.array(G.laplacian(normalized=True), dtype=np.float64)
vals_laplacian = np.linalg.eigvalsh(L).real

exact_entropy =  entropy(vals_laplacian)

L_sparse = scipy.sparse.coo_matrix(L)

num_vecs = 100
num_steps = 50
approximated_entropy = slq_entropy(
    L_sparse, num_vecs, num_steps, device="/cpu:0")

approximated_entropy, exact_entropy
```

The above code returns

```
(-509.46283, -512.5283224633046)
```

Custom Trace Function

```
import tensorflow as tf

from stdog.spectra.trace_function import slq
def trace_function(eig_vals):
    return tf.exp(eig_vals)


num_vecs = 100
num_steps = 50
approximated_trace_function, _ = slq(L_sparse, num_vecs, num_steps,  trace_function)
```

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 Dynamics

1.3.1 Kuramoto: Heun's method in Tensorflow

References

[1] - Thomas Peron, Bruno Messias, Angélica S. Mata, Francisco A. Rodrigues, and Yamir Moreno. On the onset of synchronization of Kuramoto oscillators in scale-free networks. arXiv:1905.02256 (2019).

class stdog.dynamics.kuramoto.heuns.**Heuns**(*adjacency*, *phases*, *omegas*, *couplings*, *total_time*, *dt*, *transient=False*, *frustration=None*, *precision=32*, *device='/gpu:0'*, *log=None*, *use_while=False*)

This class allow efficiently simulating phase oscillators (the Kuramoto model) on large heterogeneous networks using the Heun's method implemented in TensorFlow.

Variables

- **adjacency** (*coo matrix*) –
- **phases** (*np.ndarray*) –
- **omegas** (*np.ndarray*) –
- **couplings** (*np.ndarray*) –
- **total_time** (*float*) –
- **dt** (*float*) –
- **transient** (*bool*) –
- **frustation** (*bool*) –
- **device** (*str*) –
- **log** (*bool*) –
- **use_while** (*bool*) –
- **order_parameter_list** (*np.ndarray*) –

create_tf_graph()

This method creates the tensorflow graph

run()

This runs the algorithm and updates the phases.

If transiet is set to True, then the order parameters is calculated and the array order_parameter_list is updated.

1.3.2 Kuramoto: Heun's method in CUDA

allow efficiently simulating phase oscillators (the Kuramoto model) on large heterogeneous networks using the Heun's method with a "pure" CUDA implementation. Should be faster than tensorflow implementation. .. admonition:: References

[1] - Thomas Peron, Bruno Messias, Angélica S. Mata, Francisco A. Rodrigues, and Yamir Moreno. On the onset of synchronization of Kuramoto oscillators in scale-free networks. arXiv:1905.02256 (2019).

class stdog.dynamics.kuramoto.cuheuns.**CUHeuns**(*adjacency*, *phases*, *omegas*, *couplings*, *total_time*, *dt*, *transient=False*, *block_size=1024*)

Allow efficiently simulating phase oscillators (the Kuramoto model) on large heterogeneous networks using the Heun's method. This class uses a pure CUDA implementation of Heun's method. Therefore, should be faster than TensorFlow implementation also provided by StDoG

Variables

- **adjacency** (*coo matrix*) –
- **phases** (*np.ndarray*) –
- **omegas** (*np.ndarray*) –
- **couplings** (*np.ndarray*) –
- **total_time** (*float*) –
- **dt** (*float*) –
- **transient** (*bool*) –
- **order_parameter_list** (*np.ndarray*) –

create_simulation()

This method method crates the simulation.

run()

This runs the algorithm and updates the phases.

If transiet is set to True, then the order parameters is calculated and the array order_parameter_list is updated.

1.4 Spectra

The methods in this Spectra module uses our another package *eMaTe*, which is available at https://github.com/stdogpkg/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.

1.4.1 Trace Functions

The core module responsible to calc trace functions.

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.

stdog.spectra.trace_function.**slq**(*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.

stdog.spectra.trace_function.**entropy**(*L_sparse*, *num_vecs=100*, *num_steps=50*, *device='/gpu:0'*)

Compute the spectral entropy

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

$$f(\lambda) = \begin{cases} -\lambda \log_2 \lambda \;\; if \;\; \lambda > 0; 0, \;\; otherwise \end{cases}$$

Parameters

- **L** (*sparse matrix*) –
- **num_vecs** (*int*) – Number of random vectors used to approximate the trace using the Hutchison's trick [1]

- **num_steps** (*int*) – Number of Lanczos steps or Chebyschev's moments
- **device** (*str*) – "/cpu:int" our "/gpu:int"

Returns approximated_spectral_entropy

Return type float

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.

stdog.spectra.trace_function.**estrada_index**(*L*, *num_vecs=100*, *num_steps=50*, *device='/gpu:0'*)

Given the Laplacian matrix $L \in \mathbb{R}^{|V|\times|V|}$ s.t. for all $v \in \mathbb{R}^{|V|}$ we have $v^T L v > 0$ the Estrada Index is given by

$$\operatorname{tr}\exp(L) = \sum_{i=0}^{|V|} e^{\lambda_i}$$

Parameters

- **L** (*sparse matrix*) –
- **num_vecs** (*int*) – Number of random vectors used to approximate the trace using the Hutchison's trick [1]
- **num_steps** (*int*) – Number of Lanczos steps or Chebyschev's moments
- **device** (*str*) – "/cpu:int" our "/gpu:int"

Returns approximated_estrada_index

Return type float

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.

1.4.2 Spectral Density

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

stdog.spectra.dos.**kpm**(*H*, *num_moments*, *num_vecs*, *extra_points*, *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]

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"

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.

1.5 Utils

1.5.1 Misc

Contains tools to help some boring tasks

stdog.utils.misc.**ig2sparse**(*G*, *transpose=False*, *attr=None*, *precision=32*)

Given an igraph instance returns the sparse adjacency matrix in COO format.

Parameters

- **G** (*igraph instance*) –
- **transpose** (*bool*) – If the adjacency matrix should be transposed or not
- **attr** (*str*) – The name of weight attribute
- **precision** (*int*) – The precision used to store the weight attributes

Returns **L**

Return type COO Sparse matrix

Python Module Index

S

Index