GSPBOX: A toolbox for signal processing on graphs

Nathanael Perraudin, Johan Paratte, David I Shuman, Lionel Martin, Vassilis Kalofolias, Pierre Vandergheynst & David K.Hammond

Ecole Polytechnique Fédérale de Lausanne (EPFL), LTS2

Website: **`https://lts2.epfl.ch/gsp/`**

Abstract

The Graph Signal Processing toolbox (GSPBox) is a MATLAB/Python open-source toolbox designed for graph signal processing and data mining tasks such as filtering, de-noising, prediction, classification, data representation and visualization. Its purpose is to serve as a tool for achieving new scientific developments in a reproducible research perspective.

We propose an overview of the current features of the toolbox: graph construction, graph operators, graph learning, filter design, spectral filtering methods, graph reduction, bindings with the optimization toolbox UNLocBoX, etc.

In order to prepare future collaborations between different research groups, we additionally present the modules that are currently under development and will be released in the near future.

1 The box

The general design of the GSPBox focuses around the graph object [1], a structure containing the necessary information to use most of the algorithms.

Toolbox features

- **MATLAB** and **Python** libraries
- Efficient implementations of a large set of graph signal processing algorithms
- Documented, maintained and regularly tested
- Fast development at the state of the art of the graph signal processing field
- Binded with the UNLocBoX to solve your graph regularized problems

2 Graph

To initialize a graph from a weight matrix W, use

```
G = gsp_graph(W);
```

Alternatively, the toolbox contains a lot of synthetic graphs and an optimized nearest neighbor graph function

```
G = gsp_nn_graph(X); % X is a matrix of coordinates
```

Finally, if you do not possess any coordinates, you can build a graph using graph learning methods:

```
a = 1;       % Regularization parameter 1
b = 1.5;     % Regularization parameter 2
% For X is a matrix of smooth signals
G = gsp_learn_graph_log_degrees(X, a, b);
```

All those functions initialize the graph structure with the arguments inside Table 1.

Attribute	Format	Data type	Description
Mandatory fields			
`W`	NxN sparse matrix	double	Weight matrix $\mathbf{W}$
`L`	NxN sparse matrix	double	Laplacian matrix $\mathcal{L}$
`d`	Nx1 vector	double	The diagonal of the degree matrix
`N`	scalar	integer	Number of vertices
`Ne`	scalar	integer	Number of edges
`plotting`	**[M]: `structure` [P]: `dict`**	none	Plotting parameters
`type`	text	string	Name, type or short description
`directed`	scalar	**[M]: `logical` [P]: `boolean`**	State if the graph is directed or not
`lap_type`	text	string	Laplacian type
Optional fields			
`A`	NxN sparse matrix	**[M]: `logical` [P]: `boolean`**	Adjacency matrix
`coords`	Nx2 or Nx3 matrix	double	Vectors of coordinates in 2D or 3D.
`lmax`	scalar	double	Exact or estimated maximum eigenvalue
`U`	NxN matrix	double	Matrix of eigenvectors
`e`	Nx1 vector	double	Vector of eigenvalues
`mu`	scalar	double	Graph coherence

Table 1: Attributes of the graph object

In order to speed-up computation with MATLAB, an optional field can be pre-computed:

```
% The Fourier basis
G = gsp_compute_fourier_basis(G);
% The maximum Laplacian eigenvalues
G = gsp_estimate_lmax(G);
% The gradient operator
G = gsp_adj2vec(G);
```

3 Operators

The most central operator in graph signal processing is the Laplacian. It is stored in `G.L`. In order to select the correct definition, use:

```
lap_type = 'normalized';
G = gsp_create_laplacian(G, lap_type);
```

The available definitions are given in Table 2.

Name	Laplacian matrix (operator)
Undirected graph	
Combinatorial Laplacian	$\mathbf{D}-\mathbf{W}$
Normalized Laplacian	$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}$
Directed graph	
Combinatorial Laplacian	$\frac{1}{2}(\mathbf{D}_{+}+\mathbf{D}_{-}-\mathbf{W}-\mathbf{W}^{*})$
Degree normalized Laplacian	$\mathbf{I}-\frac{1}{2}\left(\mathbf{D}_{+}^{-\frac{1}{2}}[\mathbf{W}+\mathbf{W}^{*}]\mathbf{D}_{-}^{-\frac{1}{2}}\right)$
Distribution normalized Laplacian	$\frac{1}{2}\left(\mathbf{\Pi}^{\frac{1}{2}}\mathbf{P}\mathbf{\Pi}^{-\frac{1}{2}}+\mathbf{\Pi}^{-\frac{1}{2}}\mathbf{P}^{*}\mathbf{\Pi}^{\frac{1}{2}}\right)$

Table 2: Different definitions for graph Laplacian operators and their associated edge derivatives. (For directed graph, $d_{+}, \mathbf{D}_{+}$ and $d_{-}, \mathbf{D}_{-}$ define the out degree and in-degree of a node. $\pi, \mathbf{\Pi}$ is the stationary distribution of the graph and $\mathbf{P}$ is a normalized weight matrix $\mathbf{W}$.

Based on the Lapacian, the toolbox is able to perform

- Fourier transform **[M]: `gsp_gft`**
- Kron reduction **[M]: `gsp_kron_reduce`**
- Gradient computation **[M]: `gsp_grad`**
- Multi-resolution analysis using a pyramid transform **[M]: `gsp_pyramid_analysis`**

4 Filters

Filters are central in graph signal processing. They are implemented as:

```
g = @(x) exp(-x);
tau = 1;
h = @(x) 1./(1+tau*x);
% Filterbank composed of g and h
fb = {g, h};
```

The toolbox contains a large set of predefined designs such as:

- Wavelets (Filters are scaled version of a mother window) **[M]: `gsp_design_mexican_hat` & `gsp_design_abspline`**
- Gabor (Filters are shifted version of a mother window) **[M]: `gsp_design_itersine`**
- Low pass filter (Filters to de-noise a signal) **[M]: `gsp_design_expwin`**

5 Plotting

The toolbox contains a few plotting functions

```
gsp_plot_graph(G);        % Plot a graph
gsp_plot_signal(G, sig); % Plot a signal
gsp_plot_filter(G, g);   % Plot a filter
```

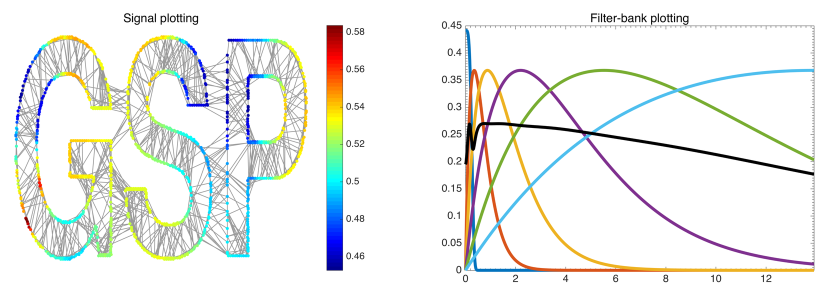

Figure 1: Visualization of graph and signals using plotting functions.

6 Python

The Python port of the library works similarly. Each package described here is a module of the library. Graph functions are in **[P]: `pygsp.graphs`**, filters in **[P]: `pygsp.filters`**, operators in **[P]: `pygsp.operators`** and so on.

All mathematical operations are performed with matrices using numpy and scipy libraries. Plotting requires either matplotlib or pyqtgraph to be installed.

7 Help

Starting with the GSPBox

1. Get a free version online:
   [M]: `https://lts2.epfl.ch/gsp`
   [P]: `pip install pygsp`
2. Do the tutorial:
   [M]: `Run gsp_demo`
   [P]: `https://lts2.epfl.ch/pygsp/tutorials`
3. Get help from the documentation, the article [2], or by contacting us `gspbox-support@groupes.epfl.ch`

If you need additional functions, please ask. Unreleased modules include:

- Machine learning / Optimization
- Clustering
- Low rank extraction
- Hypergraphs
- Bi-graphs, vertex-time signal processing

References

[1] D. I Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Process. Mag.*, 30(3):83–98, May 2013.

[2] Nathanaël Perraudin, Johan Paratte, David Shuman, Vassilis Kalofolias, Pierre Vandergheynst, and David K. Hammond. GSPBOX: A toolbox for signal processing on graphs. *ArXiv e-prints*, August 2014.

[3] Nathanael Perraudin, David Shuman, Gilles Puy, and Pierre Vandergheynst. Unlocbox a matlab convex optimization toolbox using proximal splitting methods. *arXiv preprint arXiv:1402.0779*, 2014.

Acknowledgements

We would like to thank all coding authors of the GSPBOX. The toolbox was ported in Python by Basile Châtillon, Alexandre Lafaye and Nicolas Rod. The toolbox was also improved by Nauman Shahid and Yann Schönenberger.

This work has been supported by the Swiss National Science Foundation research project *Towards Signal Processing on Graphs*, grant number: 2000_21/154350/1.

Demonstration in 7 steps

Use it as a black-box

MATLAB code

```
% 1) Start the toolbox
gsp_start;
% 2) Create a graph
N = 100; % number of nodes
G = gsp_sensor(N);
% 3) Compute the Fourier basis
G = gsp_compute_fourier_basis(G);
% 4) Create a smooth signal with noise
x = G.U(:, 2);
y = x + 1/sqrt(N)*randn(N, 1);
% 5) Select a filter
g = gsp_design_expwin(G, 0.1);
% 6) Remove the noise
s = gsp_filter(G, g, y);
% 7) Display the results
figure(1); gsp_plot_signal(G, x); title('Original signal');
figure(2); gsp_plot_signal(G, y); title('Noisy signal');
figure(3); gsp_plot_signal(G, s); title('Denoised signal');
```

Python code

```
# 1) Import package
import pygsp, numpy as np
# 2) Create a graph
N = 100  # number of nodes
G = pygsp.graphs.Sensor(N)
# 3) Compute the Fourier basis
G.compute_fourier_basis()
# 4) Create a smooth signal with noise
x = G.U[:, 1]
y = x + np.random.normal(scale=1/np.sqrt(N), size=N)
# 5) Select a filter
filter = pygsp.filters.Expwin(G, 0.1)
# 6) Remove the noise
s = filter.analysis(y)
# 7) Display the results
G.plot_signal(x, plot_name='Original signal')
G.plot_signal(y, plot_name='Noisy signal')
G.plot_signal(s, plot_name='Denoised signal')
```

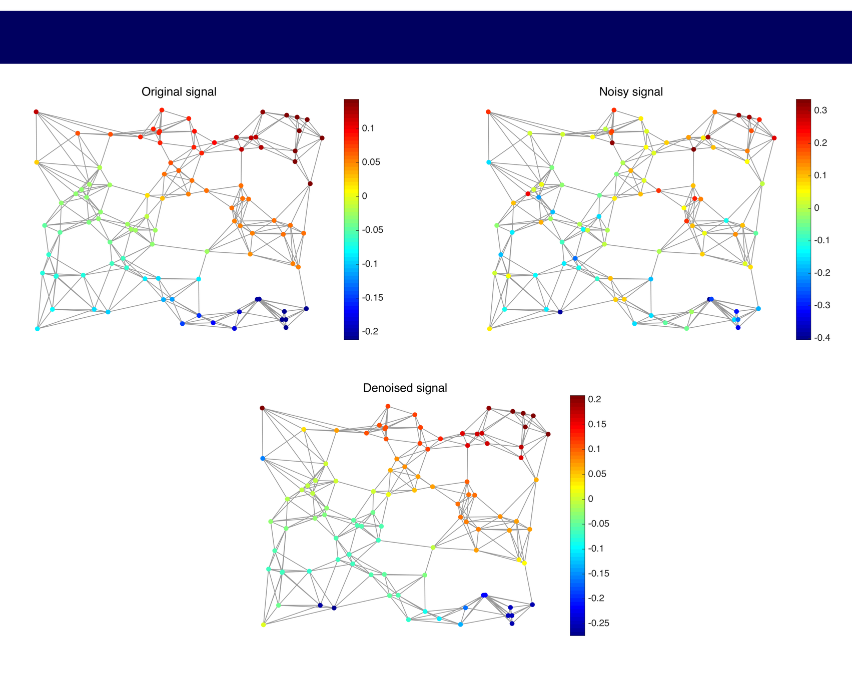

Figure 2: Resulting figures.