GSPBOX: A toolbox for signal processing on graphs

Nathanaël Perraudin
Johan Paratte
David Shuman
Vassilis Kalofolias
Pierre Vanderheynst
David K. Hammond
Contents

1 GSPBOX - Startup
  1.1 Start ................................................................. 7
    1.1.1 gsp_start ............................................. 7
    1.1.2 gsp_make ............................................. 7
    1.1.3 gsp_install ......................................... 7

2 GSPBOX - Demos
  2.1 Introduction to the GSPBox ................................. 9
    2.1.1 gsp_demo ........................................... 9
    2.1.2 gsp_demo_wavelet .................................. 12
  2.2 Convex optimization on graph ................................ 18
    2.2.1 gsp_demo_graph_tv ................................ 18
  2.3 Sparse approximation ......................................... 21
    2.3.1 gsp_demo_pyramid ................................ 21

3 GSPBOX - Graphs
  3.1 Specific graphs ............................................... 25
    3.1.1 gsp_swiss_roll ...................................... 25
    3.1.2 gsp_david_sensor_network ......................... 26
    3.1.3 gsp_ring .............................................. 26
    3.1.4 gsp_path .............................................. 27
    3.1.5 gsp_airfoil .......................................... 28
    3.1.6 gsp_comet ............................................ 28
    3.1.7 gsp_erdos_renyi ..................................... 29
    3.1.8 gsp_minnesota ....................................... 30
    3.1.9 gsp_low_stretch_tree ............................... 31
    3.1.10 gsp_sensor ......................................... 31
    3.1.11 gsp_random_regular ................................. 32
    3.1.12 gsp_full_connected ................................. 33
    3.1.13 gsp_pcl_nn_graph .................................. 34
    3.1.14 gsp_sphere ......................................... 35
    3.1.15 gsp_cube ............................................ 36
    3.1.16 gsp_logo ............................................ 37
    3.1.17 gsp_community ..................................... 38
    3.1.18 gsp_bunny .......................................... 39
  3.2 Utils .......................................................... 40
    3.2.1 gsp_graph_default_parameters ..................... 40
    3.2.2 gsp_graph ............................................ 41

4 GSPBOX - Filters
  4.1 Design - N filter ............................................. 43
    4.1.1 gsp_design_mexican_hat ............................ 43
    4.1.2 gsp_design_abspline ................................ 44
    4.1.3 gsp_design_meyer ................................... 45
    4.1.4 gsp_design_simple_tf ............................... 46
| Section | Subsection | Function |
|---------|------------|----------|
| 4.1.5   | gsp_design_itersine | 47 |
| 4.1.6   | gsp_design_half_cosine | 48 |
| 4.1.7   | gsp_design_warped_translates | 49 |
| 4.2     | Design - 2 filter (LP - HP) tight filterbank | 51 |
| 4.2.1   | gsp_design_regular | 51 |
| 4.2.2   | gsp_design_held | 53 |
| 4.2.3   | gsp_design_simoncelli | 54 |
| 4.2.4   | gsp_design_papadakis | 55 |
| 4.3     | Application | 56 |
| 4.3.1   | gsp_filter | 56 |
| 4.3.2   | gsp_filter_analysis | 56 |
| 4.3.3   | gsp_filter_synthesis | 58 |
| 4.3.4   | gsp_filter_inverse | 59 |
| 4.4     | Size Handling | 60 |
| 4.4.1   | gsp_mat2vec | 60 |
| 4.4.2   | gsp_vec2mat | 61 |
| 4.5     | Utils | 61 |
| 4.5.1   | gsp_filterbank_matrix | 61 |
| 4.5.2   | gsp_wlog_scales | 61 |
| 4.5.3   | gsp_filter_evaluate | 62 |
| 4.5.4   | gsp_filterbank_bounds | 62 |
| 4.5.5   | gsp_tighten_filter | 63 |
| 5       | GSPBOX - Operators | 65 |
| 5.1     | Localisation | 65 |
| 5.1.1   | gsp_translate | 65 |
| 5.1.2   | gsp_modulate | 65 |
| 5.1.3   | gsp_localize | 66 |
| 5.2     | Differential | 66 |
| 5.2.1   | gsp_grad_mat | 66 |
| 5.2.2   | gsp_grad | 67 |
| 5.2.3   | gsp_div | 67 |
| 5.2.4   | gsp_adj2vec | 67 |
| 5.3     | Transforms | 68 |
| 5.3.1   | gsp_gft | 68 |
| 5.3.2   | gsp_igft | 69 |
| 5.3.3   | gsp_gwft | 70 |
| 5.3.4   | gsp_ngwft | 70 |
| 5.4     | Pyramid - Reduction | 71 |
| 5.4.1   | gsp_kron_reduction | 71 |
| 5.4.2   | gsp_kron_pyramid | 72 |
| 5.4.3   | gsp_pyramid_analysis | 73 |
| 5.4.4   | gsp_pyramid_synthesis | 74 |
| 5.4.5   | gsp_pyramid_cell2coeff | 75 |
| 5.4.6   | gsp_tree_multiresolution | 75 |
| 5.5     | Utils | 76 |
| 5.5.1   | gsp_create_laplacian | 76 |
| 5.5.2   | gsp_compute_fourier_basis | 76 |
| 6       | GSPBOX - Pointclouds | 79 |
| 6.1     | Utils | 79 |
| 6.1.1   | gsp_pointcloud | 79 |
Abstract

In this document we introduce a Matlab toolbox called the Graph Signal Processing toolbox (GSPBox). This toolbox is based on spectral graph theory, more specifically graph filtering. It includes fast filtering routines using Chebychev polynomials as presented in [4].

This document is automatically generated from the source files and includes the complete documentation of the toolbox. However the most up-to-date documentation can be found on the official website http://lts2research.epfl.ch/gsp/doc
Chapter 1

GSPBOX - Startup

1.1 Start

1.1.1 GSP_START - Initialize the toolbox

Usage

    gsp_start();

Description

Initialisation script for the GSPBox. This script add the different path needed to run the toolbox.

1.1.2 GSP_MAKE - Compile the necessary toolboxes for the gspbox

Usage

    gsp_make();

Description

This function compile the routine for the gspbox:

    gsp_make();

1.1.3 GSP_INSTALL - Install third party software

Usage

    gsp_install();

Description

This function install third party software. It require an internet connection.

    It will install the gaimc toolbox and compile some functions
Chapter 2

GSPBOX - Demos

2.1 Introduction to the GSPBox

2.1.1 GSP_DEMO - Tutorial on the GSPBox

In this demo, we are going to show the basic operations of the GSPBox. To launch the toolbox, just go into the repository where the GSPBox was extracted and type:

```matlab
gsp_start;
```

A banner will popup telling you that everything happens correctly. To speedup some processing, you might want to compile some mexfile. Refer to `gsp_make` for more informations. However, if the compilation is not working on your computer, keep quiet, everything should still work and most of the routine are implemented only in matlab.

Most likely, the first thing you would like to do is to create a graph. To do so, you only need the adjacency or the weight matrix $W$. Once you have it, you can construct a graph using:

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

This function will create a full structure ready to be used with the toolbox. To know a bit more about what is in this structure, you can refer to the help of the function `gsp_graph_default_parameters`.

The GSPBox contains also a list of graph generators. To see a full list of these graphs, type:

```matlab
help graphs
```

This code produces the following output:

```
GSPBOX - Graphs

Specific graphs
  gsp_swiss_roll - Create swiss roll graph
  gsp_david_sensor_network - Create the sensor network from david
  gsp_ring - Create the ring graph
  gsp_path - Create the path graph
  gsp_airfoil - Create the airfoil graph
  gsp_comet - Create the comet graph
  gsp_erdos_renyi - Create a erdos renyi graph
  gsp_minnesota - Create Minnesota road graph
  gsp_low_stretch_tree - Create a low stretch tree graph
  gsp_sensor - Create a random sensor graph
  gsp_random_regular - Create a random regular graph
  gsp_full_connected - Create a fully connected graph
  gsp pcl_nn_graph - Create a nearest neighbors graph
  gsp_sphere - Create a spherical-shaped graph
  gsp_cube - Create a cubical-shaped graph
```

---

9
CHAPTER 2. GSPBOX - DEMOS

- gsp_logo: Create a GSP logo graph
- gsp_community: Create a community graph
- gsp_bunny: Create a bunny graph

Utils
- gsp_graph_default_parameters: Initialise all parameters for a graph
- gsp_graph: Create a graph from a weight matrix

For help, bug reports, suggestions etc. please send email to gspbox-help@lists.sourceforge.net

For this demo, we will use the graph `gsp_logo`. You can load it using:

```matlab
G = gsp_logo
```

This code produces the following output:

```plaintext
G =

W: [1130x1130 double]
coords: [1130x2 double]
info: [1x1 struct]
plotting: [1x1 struct]
limits: [0 640 -400 0]
A: [1130x1130 logical]
N: 1130
type: 'unknown'
directed: 0
lap_type: 'combinatorial'
L: [1130x1130 double]
d: [1x1130 double]
Ne: 6262
```

Here observe the attribute of the structure `G`.

- **G.W**: Weight matrix
- **G.A**: Adacency matrix
- **G.N**: Number of nodes
- **G.type**: Type of graph
- **G.directed**: 1 if the graph is directed, 0 if not
- **G.lap_type**: Laplacian type
- **G.d**: Degree vector
- **G.Ne**: Number of edges
- **G.coords**: Coordinates of the vertices
- **G.plotting**: Plotting parameters

In the folder `plotting`, the GSPBox contains some plotting routine. For instance, we can plot a graph using:

```matlab
gsp_plot_graph(G);
```

Wonderful! Isn’t it? Now, let us start to analyse this graph. To compute graph Fourier transform or exact graph filtering, you need to precompute the Fourier basis of the graph. This operation could be relatively long since it involves a full diagonalization of the Laplacian. Don’t worry, you do not need to perform this operation to filter signals on graph. The fourier basis is computed thanks to:
2.1. INTRODUCTION TO THE GSPBOX

This figure shows the result of the command 'gsp_plot_graph(G)'

\[
G = \text{gsp\_compute\_fourier\_basis}(G);
\]

The function `gsp_compute_fourier_basis` adds two new fields to the structure `G`: * `G.U`: The eigenvectors of the Fourier basis * `G.e`: The eigenvalues

The fourier eigenvectors does look like a sinusoid on the graph. Let’s plot the second and the third one. (The first one is constant!):

\[
\begin{align*}
gsp\_plot\_signal(G,G.U(:,2)); \\
title('Second eigenvector') \\
subplot(212) \\
gsp\_plot\_signal(G,G.U(:,3)); \\
title('Third eigenvector')
\end{align*}
\]

Figure 2.1: GSP graph

Figure 2.2: Eigenvectors
Now, we are going to show a basic filter operation. Filters are usually defined in the spectral domain.
To defined the following filter
\[ h(x) = \frac{1}{1 + \tau x}, \]
just write in Matlab:
```matlab
tau = 1;
flow = @(x) 1./(1+tau*x);
```
Hint: You can define filterbank using cell array!
Let’s display this filter:
```matlab
gsp_plot_filter(G,h);
```

![Filter h](image)

Figure 2.3: Low pass filter \( h \)
The filter \( h \) is plotted along all the spectrum of the graph. The black cross are the eigenvalues of the Laplacian. They are the points where the continuous filter will be evaluated to create a discrete filter.

To apply a the filter to a given signal, you only need to run a single function:
```matlab
f2 = gsp_filter(G,h,f);
```
\( gsp\_filter \) is actually a shortcut to \( gsp\_filter\_analysis \). \( gsp\_filter\_analysis \) perform the analysis operator associated to a filterbank. See the \( gsp\_demo\_wavelet \) for more information.

Finally, we display the result of this low pass filtering on the graph.
Enjoy the GSPBox...

Output

2.1.2 GSP_DEMO_WAVELET - Introduction to spectral graph wavelet with the GSPBox

Description
The wavelets are a special type of filterbank. In this demo, we will show how you can very easily construct a wavelet frame and apply it to a signal. If you want to do find an interactive demo of the wavelet, we encourage you to use the sgwt_demo2 of the sgwt toolbox. It can be downloaded at:
The noise is largely removed thanks to the filter. However, some energy is diffused between the letters. This is the typical behaviour of a low pass filter.

http://wiki.epfl.ch/sgwt/documents/sgwt_toolbox-1.02.zip

The sgwt toolbox has the same core as the GSPBox and all his functions have equivalent in the GSPBox (Except the demos ;-)).

In this demo we will show you how to compute the wavelet coefficients of a graph and visualize them. First, let’s load a graph

\[
G = \text{gsp\_bunny}();
\]

This graph is a nearest-neighbor graph of a pointcloud of the Stanford bunny. It will allow us to get interesting visual results using wavelets.

At this stage we could compute the full Fourier basis using \texttt{gsp\_compute\_fourier\_basis} but this would take a lot of time, and can be avoided by using Chebychev polynomials approximations. This operation is implemented in most function and is thus completely transparent.

**Simple filtering**

Before tackling wavelets, we can see the effect of one filter localized on the graph. So we can first design a few heat kernel filters

\[
\text{taus} = [1, 10, 25, 50];
\]

\[
\text{Hk} = \text{gsp\_design\_heat}(G, \text{taus});
\]

Let’s now create a signal as a Kronecker located on one vertex (e.g. the vertex 100)

\[
\text{S} = \text{zeros}(G.N, 1);
\]

\[
\text{vertex\_delta} = 83;
\]

\[
\text{S(\text{vertex\_delta})} = 1;
\]

\[
\text{Sf\_vec} = \text{gsp\_filter\_analysis}(G, \text{Hk}, \text{S});
\]

\[
\text{Sf} = \text{gsp\_vec2mat}(\text{Sf\_vec}, \text{length(taus)});
\]

and plot the filtered signal

\[
\text{param\_plot.cp} = [0.1223, -0.3828, 12.3666];
\]
Let’s now replace the filtering by the heat kernel by a filter bank of wavelets. We can create a filter bank using one of the design functions such as `gsp_design_mexican_hat`.

```matlab
Nf = 6;
Wk = gsp_design_mexican_hat(G, Nf);
```

We can plot the filter bank spectrum

```matlab
figure;
gsp_plot_filter(G, Wk);
```

As we can see, the wavelets atoms are stacked on the low frequency part of the spectrum. If we want to get a better coverage of the graph spectrum we can use the function `gsp_design_warped_translates`
2.1. INTRODUCTION TO THE GSPBOX

param_filter.filter = Wk;
Wkw = gsp_design_warped_translates(G,Nf,param_filter);

Now let’s plot the new filter bank

figure;
gsp_plot_filter(G,Wkw);

We can see that the wavelet atoms are much more spread along the graph spectrum. We can visualize
the filtering by one atom as we did with the heat kernel, by placing a Kronecker delta at one specific vertex
and filter using the filter bank

S = zeros(G.N*Nf,Nf);
S(vertex_delta) = 1;
for ii=1:Nf
    S(vertex_delta+(ii-1)*G.N,ii) = 1;
end

Sf = gsp_filter_synthesis(G,Wkw,S);

We can plot the resulting signal for the different scales

    figure;
    subplot(221)
    gsp_plot_signal(G,Sf(:,1), param_plot);
    axis square
    mu = mean(Sf(:,1));
    sigma = std(Sf(:,1));
    c_scale = 4;
    caxis([mu - c_scale*sigma, mu + c_scale*sigma]);
    title('Wavelet 1');

    subplot(222)
    gsp_plot_signal(G,Sf(:,2), param_plot);
    axis square
    mu = mean(Sf(:,2));
    sigma = std(Sf(:,2));
    caxis([mu - c_scale*sigma, mu + c_scale*sigma]);
    title('Wavelet 2');

    subplot(223)
    gsp_plot_signal(G,Sf(:,3), param_plot);
    axis square
    mu = mean(Sf(:,3));
    sigma = std(Sf(:,3));
    caxis([mu - c_scale*sigma, mu + c_scale*sigma]);
    title('Wavelet 3');

    subplot(224)
    gsp_plot_signal(G,Sf(:,4), param_plot);
    axis square
    mu = mean(Sf(:,4));
    sigma = std(Sf(:,4));
    caxis([mu - c_scale*sigma, mu + c_scale*sigma]);
    title('Wavelet 4');

Curvature estimation

As a last and more applied example, let us try to estimate the curvature of the underlying 3D model by only using only spectral filtering on the graph.

A simple way to accomplish that is to use the coordinates map \([x, y, z]\) and filter it using the wavelets defined above. We obtain a 3-dimensional signal \([\hat{x}, \hat{y}, \hat{z}]\) which describes variation along the 3 coordinates

    s_map = G.coords;
    s_map_out = gsp_filter_analysis(G, Wk, s_map);
    s_map_out = gsp_vec2mat(s_map_out, Nf);

Finally we can get the curvature estimation by taking the \(l_1\) or \(l_2\) norm of the filtered signal

    dd = s_map_out(:,:,1).^2 + s_map_out(:,:,2).^2 + s_map_out(:,:,3).^2;
    dd = sqrt(dd);
Let’s now plot the result to observe that we indeed have a measure of the curvature

```matlab
figure;
subplot(221)
gsp_plot_signal(G,dd(:,2), param_plot);
axis square
title('Curvature estimation scale 1');
subplot(222)
gsp_plot_signal(G,dd(:,3), param_plot);
axis square
title('Curvature estimation scale 2');
subplot(223)
gsp_plot_signal(G,dd(:,4), param_plot);
axis square
title('Curvature estimation scale 3');
subplot(224)
gsp_plot_signal(G,dd(:,5), param_plot);
axis square
title('Curvature estimation scale 4');
```

Output

```
Average degree = 5.232920e+01
The matrix W is symmetric
GSP_DESIGN_WARPED_TRANSLATES: has to compute the spectrum continuous density function
```
2.2 Convex optimization on graph

2.2.1 GSP_DEMO_GRAPH_TV - Reconstruction of missing sample on a graph using TV

Description

In this demo, we try to reconstruct missing sample of a piece-wise smooth signal on a graph. To do so, we will minimize the well-know TV norm defined on the graph.

For this example, you need the unlocbox. You can download it: [http://unlocbox.sourceforge.net/download](http://unlocbox.sourceforge.net/download)

We express the recovery problem as a convex optimization problem of the following form:

$$\arg\min_x \|\nabla x\|_1 \text{ s. t. } \|Mx - b\|_2 \leq \varepsilon$$

Where b represent the known measurments, M an operator representing the mask and $\varepsilon$ the radius of the l2 ball.

We set

- $f_1(x) = \|\nabla x\|_1$ We define the prox of $f_1$ as:

$$\text{prox}_{f_1,\gamma}(z) = \arg\min_x \frac{1}{2}\|x - z\|_2^2 + \gamma\|\nabla z\|_1$$

- $f_2$ is the indicator function of the set S define by $\|Mx - b\|_2 < \varepsilon$ We define the prox of $f_2$ as

$$\text{prox}_{f_2,\gamma}(z) = \arg\min_x \frac{1}{2}\|x - z\|_2^2 + i_S(x),$$

with $i_S(x)$ is zero if x is in the set S and infinity otherwise. This previous problem has an identical solution as:

$$\arg\min_z \|x - z\|_2^2 \text{ such that } \|Mz - b\|_2 \leq \varepsilon$$

It is simply a projection on the B2-ball.
2.2. CONVEX OPTIMIZATION ON GRAPH

This figure shows the original signal on graph.

Figure 2.10: Original signal on graph

This figure shows the signal on graph after the application of the mask and addition of noise. Half of the vertices are set to 0.

Figure 2.11: Depleted signal on graph

Results

Comparison with Tikonof regularization

We can also use the Tikonof regularizer that will promote smoothness. In this case, we solve:

$$\arg \min_x \tau \|\nabla(x)\|_2^2 \text{ s. t. } \|Mx - b\|_2 \leq \varepsilon$$

The result is presented in the following figure:
Figure 2.12: Reconstructed signal on graph using TV
This figure shows the reconstructed signal thanks to the algorithm.

Figure 2.13: Reconstructed signal on graph using Tikonof
This figure shows the reconstructed signal thanks to the algorithm.

Output

UnLocBoX version 1.3.135. Copyright 2012-2013 LTS2-EPFL, by Nathanael Perraudin
Solution found: $\|f\| = 2.164623e+02$, rel_norm = 2.590438e-04, REL_NORM
Solution found: $\|f\| = 1.803702e+02$, rel_norm = 8.234018e-04, REL_NORM
2.3 Sparse approximation

2.3.1 GSP_DEMO_PYRAMID - Demonstration of the use of the graph pyramid

In this demonstration file, we show how to reduce a graph using the GSPBox. Then we apply the pyramid to simple signal.

The function `gsp_kron_pyramid` computes the graph pyramid for you:

```matlab
param.sparsify = 1;
param.epsilon = 0.1;
param.filters = @(x) 5 ./ ( 5 + x );
Gs = gsp_kron_pyramid(G, Nlevel,param);
```

`Gs` is a cell array of graph representing the pyramid of graphs. Here all optional parameter are important:

- **param.sparsify**: When a graph is reduced, the density of edges has tendency to be amplified. One way to counterbalance this effect is to sparsify the graph for each sublevel. The function `gsp_graph_sparsify` is used to perform this operation. However, this could lead to bad graphs (disconnected for instance).

- **param.epsilon**: is the level of sparsification.

- **param.filters**: is a cell array of filters (or a single filter). Thoses filter will be used in the analysis and synthesis operator.

Let display the results:

```matlab
figure;
for ii = 1:numel(Gs)
    subplot(2,3,ii)
    gsp_plot_graph(Gs{ii})
    title([’Reduction level: ', num2str(ii-1)]);
end
```

![Reduction level images](image)

Figure 2.14: Reduction of the graph

Now, that we have precomputed the pyramid of graphs, we can apply it to a signal. Here we create a signal that is smooth over the graph but with a big discontinuity in the middle.

The graph pyramid can be simply applied thanks to the function `gsp_pyramid_analysis`
[ca, pe] = gsp_pyramid_analysis(Gs, f);

`ca` contains the coarse approximation of each level and `pe` the prediction errors. Let’s display them:

```matlab
figure
paramplot.show_edges = 0;
for ii = 1:numel(Gs)
    subplot(2, 3, ii)
    gsp_plot_signal(Gs{ii}, pe{ii}, paramplot);
    title(['P. E. level: ', num2str(ii-1)]);
end

figure
for ii = 1:numel(Gs)
    subplot(2, 3, ii)
    gsp_plot_signal(Gs{ii}, ca{ii}, paramplot);
    title(['C. A. level: ', num2str(ii-1)]);
end
```

Finally, you can perform a synthesis operation using the function `gsp_pyramid_synthesis`:

```matlab
coeff = gsp_pyramid_cell2coeff(ca, pe);
f_pred = gsp_pyramid_synthesis(Gs, coeff);
```

The function `gsp_pyramid_cell2coeff` remove all unnecessary coefficients and keep only the last sublevel and the prediction error.

Enjoy!

**Output**

The relative reconstruction error is: 1.81249e-15
2.3. SPARSE APPROXIMATION

Figure 2.16: Prediction errors

Figure 2.17: Coarse approximations
3.1 Specific graphs

3.1.1 GSP_SWISS_ROLL - Initialize a swiss roll graph

Usage

\[
G = \text{gsp\_swiss\_roll}(N, s, \text{thresh}, \text{rand\_state});
\]

Input parameters

- **N**
  - Number of vertices.
- **s**
  - \(\text{sigma}\) (default: \(\sqrt{2/N}\))
- **thresh**
  - threshold (default: 1e-6)
- **rand\_state**
  - rand seed (default: 45)

Output parameters

- **G**
  - Graph structure.

Description

'\text{gsp\_create\_swiss\_roll}(N,s,\text{thresh},\text{rand\_state})' initializes a graph structure containing the swiss roll graph.

Example:

\[
G = \text{gsp\_swiss\_roll}(200);
\]

\[
\text{gsp\_plot\_graph}(G);
\]
3.1.2 GSP_DAVID_SENSOR_NETWORK - Initialize a sensor network

Usage

G = gsp_david_sensor_network(N);

Input parameters

N  Number of vertices (default 64)

Output parameters

G  Graph structure.

Description

'gsp_david_sensor_network(N)' initializes a graph structure containing the weighted adjacency matrix (G.W), the number of vertices (G.N), the plotting coordinates (G.coords), and the plotting coordinate limits (G.limits) of a random sensor network with N vertices. The sensors are placed randomly in the unit square, and edges are placed between any sensors within a fixed radius of each other. The edge weights are assigned via a thresholded Gaussian kernel. The sensor network will be connected for N = 500 or N = 64.

Warning: this graph is not necessarily connected...

Example:

G = gsp_david_sensor_network(64);
paramplot.show_edges = 1;
gsp_plot_graph(G,paramplot);

3.1.3 GSP_RING - Initialize a ring graph

Usage

G = gsp_ring(N);
G = gsp_ring(N,k);
G = gsp_ring();

Input parameters

N  Number of vertices. (default 64)
k  Number of neighbors in each direction (default 1)
3.1. SPECIFIC GRAPHS

Output parameters

\[ G \]
Graph structure.

Description

'gsp\_ring(N)' initializes a graph structure containing the weighted adjacency matrix (G.W), the number of vertices (G.N), the plotting coordinates (G.coords), and the plotting coordinate limits (G.coord\_limits) of a ring graph with N vertices. Each vertex in the ring has 2k neighbors (maximum value of k is \( N/2 \)). The edge weights are all equal to 1.

Example:

```matlab
G = gsp\_ring(64);
param.show_edges = 1;
gsp\_plot\_graph(G,param);
```

3.1.4 GSP\_PATH - Initialize a path graph

Usage

\[ G = gsp\_path(N); \]

Input parameters

\[ N \]
Number of vertices (default 32).

Output parameters

\[ G \]
Graph structure.

Description

'gsp\_path(N)' initializes a graph structure of a path graph. The path graph correspond the graph used for the DCT. See references for more informations.

Example:

```matlab
G = gsp\_path(16);
param.show_edges = 1;
gsp\_plot\_graph(G,param);
```
### 3.1.5 GSP_AIRFOIL - Initialize the airfoil graph

**Usage**

```matlab
G = gsp_airfoil();
```

**Input parameters**

- `non`: none

**Output parameters**

- `G`: Graph structure.

**Description**

`gsp_airfoil()` initializes a graph structure containing the weighted adjacency matrix (G.W), the number of vertices (G.N), the plotting coordinates (G.coords), and the plotting coordinate limits (G.limits) of the airfoil mesh graph. All edge weights are equal to 1.

**Example:**

```matlab
G = gsp_airfoil();
paramplot.show_edges = 1;
gsp_plot_graph(G,paramplot);
```

---

### 3.1.6 GSP_COMET - Initialize a comet graph

**Usage**

```matlab
G = gsp_comet(N,k);
```

**Input parameters**

- `N`: Number of vertices. (default 32)
- `k`: Degree of center vertex. (default 12)
Output parameters

\[ G \]
Graph structure.

Description

\'gsp_comet(N,k)\' initializes the comet graph. The comet graph is a simple path graph with a star of degree \( k \) at its end.

Example:

\[
G = gsp_comet(16,8);
param.show_edges = 1;
gsp_plot_graph(G,param);
\]

3.1.7 GSP_ERDOS_RENYI - Create a random Erdos Renyi graph

Usage

\[
G = gsp_erdos_renyi( N,p,param );
G = gsp_erdos_renyi( N,p );
\]

Input parameters

\[ N \]
Number of nodes

\[ p \]
Probability of connection of a node with another

\[ param \]
Structure of optional parameter

Output parameters

\[ G \]
Graph structure.

Description

\'gsp_erdos_renyi(N,p,param)\' initializes a graph structure containing the weighted adjacency matrix \((G.W)\), the number of vertices \((G.N)\) for an Erdos Renyi graph. All edge weights are equal to 1.

The Erdos Renyi graph is constructed by connecting nodes randomly. Each edge is included in the graph with probability \( p \) independent from every other edge.
param a Matlab structure containing the following fields:

- **param.connected**: flag to force the graph to be connected. By default, it is 1.
- **param.maxit**: is the maximum number of try to connect the graph. By default, it is 10.
- **param.verbose**: 0 no log, 1 print main steps, 2 print all steps. By default, it is 1.

Example:

\[ G = \text{gsp}_\text{erdos}_\text{renyi}(100,0.05) \]

### 3.1.8 GSP_MINNESOTA - Initialize the Minnesota road network

**Usage**

\[
G = \text{gsp}_\text{minnesota}();
\]

\[
G = \text{gsp}_\text{minnesota}(\text{connect});
\]

**Input parameters**

- **connect**
  
  change the graph so that it is connected (default 1)

**Output parameters**

- **G**
  
  Graph structure.

**Description**

`gsp_minnesota()` initializes a graph structure containing the weighted adjacency matrix (G.W), the number of vertices (G.N), the plotting coordinates (G.coords), and the plotting coordinate limits (G.limits) of the Minnesota road network from the MatlabBGL library.

Remark: if connect is set to 1. We adjust the adjacency matrix so that all edge weights are equal to 1, and the graph is connected. It is the default!

To get the original disconnected graph, use:

\[
G = \text{gsp}_\text{minnesota}(\text{connect});
\]

Example:

\[
G = \text{gsp}_\text{minnesota}();
\]

\[
\text{paramplot}.\text{show_edges} = 1;
\]

\[
\text{gsp}_\text{plot}_\text{graph}(G,\text{paramplot});
\]

References: [3]
3.1.9  **GSP_LOW_STRETCH_TREE - Initialize a low stretch tree**

**Usage**

\[
G = \text{gsp\_low\_stretch\_tree}(k);
\]

\[
G = \text{gsp\_low\_stretch\_tree}();
\]

**Input parameters**

\[ k \]  
\( 2^k \) points on each side of the grid of vertices. (default 6)

**Output parameters**

\[ G \]  
Graph structure.

**Description**

`gsp_create_low_stretch_tree(k)` initializes a graph structure containing the weighted adjacency matrix (G.W), the number of vertices (G.N), the plotting coordinates (G.coords), the plotting coordinate limits (G.limits), and the root of a low stretch tree on a grid of points. There are \( 2^k \) points on each side of the grid, and therefore \( 2^{2k} \) total vertices. The edge weights are all equal to 1.

**Example:**

```matlab
G = gsp_low_stretch_tree(3);
paramplot.show_edges = 1;
gsp_plot_graph(G,paramplot);
```

![Diagram of a low stretch tree](image)

3.1.10  **GSP_SENSOR - Create a random sensor graph**

**Usage**

\[
G = \text{gsp\_sensor}( N );
\]

\[
G = \text{gsp\_sensor}( );
\]

\[
G = \text{gsp\_sensor}( N, \text{param} );
\]

**Input parameters**

- \[ N \]  
Number of nodes (default 128)

- \[ \text{param} \]  
Structure of optional parameters
### Output parameters

- **G**  Graph

### Description

This function creates a 2-dimensional random sensor graph. All the coordinates are between 0 and 1.

*param* is an optional structure with the following fields:

- *param.Nc*: Minimum number of connections (default 2)
- *param.regular*: Flag to fix the number of connections to Nc (default 0)
- *param.verbose*: display parameter - 0 no log - 1 display the errors (default 1)
- *param.N._try*: Number of attempts to create the graph (default 50)
- *param.distribute*: To distribute the points more evenly (default 0)
- *param.connected*: To force the graph to be connected (default 1)

Example:

```matlab
G = gsp_sensor(300);
paramplot.show_edges = 1;
gsp_plot_graph(G,paramplot);
```

3.1.1 GSP_RANDOM_REGULAR - Create a random regular graph

#### Usage

```matlab
G = gsp_random_regular( N,k )
G = gsp_random_regular( N )
G = gsp_random_regular();
```

**Input parameters**

- **N**  Number of nodes (default 64)
- **k**  Number of connection of each nodes (default 6)
### 3.1. SPECIFIC GRAPHS

**Output parameters**

- **G**
  - Graph structure.

**Description**

`gsp_random_regular(N,k)` initializes a graph structure containing the weighted adjacency matrix (G.W), the number of vertices (G.N) for a random regular graph. All edge weights are equal to 1.

The random regular graph has the property that every node is connected to 'k' other nodes.

Example:

```matlab
G = gsp_random_regular(100,3)
```

This code produces the following output:

```
G =

    type: 'random_regular'
    W: [100x100 double]
    A: [100x100 logical]
    N: 100
    directed: 0
    lap_type: 'combinatorial'
    L: [100x100 double]
    d: [1x100 double]
    Ne: 300
    coords: [100x2 double]
    plotting: [1x1 struct]
```

### 3.1.12 GSP_FULL_CONNECTED - Create a fully connected graph

**Usage**

```matlab
G = gsp_full_connected(N);
G = gsp_full_connected();
```

**Input parameters**

- **N**
  - Number of vertices (default 10)

**Output parameters**

- **G**
  - Graph structure.

**Description**

`gsp_full_connected(N)` initializes a graph structure representing a fully connected graph. All weight are set to 1.

Example:

```matlab
G = gsp_full_connected(5);
param.show_edges = 1;
gsp_plot_graph(G,param);
```
3.1.13 **GSP_PCL_NN_GRAPH - Create a nearest neighbors graph from a point cloud**

**Usage**

```matlab
: G = gsp_pcl_nn_graph( Xin );
G = gsp_pcl_nn_graph( Xin, param );
```

**Input parameters**

- **Xin**: Input points
- **param**: Structure of optional parameters

**Output parameters**

- **G**: Resulting graph

**Description**

'gsp_pcl_nn_graph(Xin, param)' creates a graph from positional data. The points are connected to their neighbors (either belonging to the k nearest neighbors or to the epsilon-closest neighbors).

Example:

```matlab
P = gsp_pointcloud('bunny');
param.type = 'knn';
G = gsp_pcl_nn_graph(P, param);
gsp_plot_graph(G);
```

*This code produces the following output:*

WARNING: The matrix W is not symmetric! Artificial symmetrization
3.1. SPECIFIC GRAPHS

Additional parameters

- `param.type`: ['knn', 'radius'] the type of graph
- `param.center`: [0, 1] center the data
- `param.rescale`: [0, 1] rescale the data (in a 1-ball)
- `param.sigma`: float the variance of the distance kernel
- `param.k`: int number of neighbors for knn
- `param.epsilon`: float the radius for the range search

3.1.14 GSP_SPHERE - Create a spherical-shaped graph

Usage

```python
G = gsp_sphere();
G = gsp_sphere( param );
```

Input parameters

- `param` Structure of optional parameters

Output parameters

- `G` Resulting graph

Description

`gsp_sphere( param )` creates a graph from points sampled on a hyper-sphere. The dimension of the sphere can be passed as a parameter. It can be sampled in a uniform voxel grid or randomly.

Additional parameters

- `param.radius`: float the radius of the sphere
- `param.nb_pts`: int the number of vertices
- `param.nb_dim`: int the dimension
- `param.sampling`: ['random'] the variance of the distance kernel

Example:
G = gsp_sphere();
gsp_plot_graph(G);
axis square

This code produces the following output:

WARNING : The matrix W is not symmetric ! Artificial symmetrization

3.1.15 GSP_CUBE - Create a graph corresponding to the sampling of an hyper-cube

Usage

: G = gsp_cube();
G = gsp_cube( param );

Input parameters

param Structure of optional parameters

Output parameters

G Resulting graph

Description

’gsp_cube( param )’ creates a graph from points sampled on a hyper-cube. The dimension of the cube can be passed as a parameter. It can be sampled in a uniform voxel grid or randomly.

Additional parameters

- param.radius : float the edge length
- param.nb_pts : int the number of vertices
3.1. SPECIFIC GRAPHS

- \textit{param.nb_dim} : \text{int the dimension}

- \textit{param.sampling} : \text{[‘random’] the variance of the distance kernel}

Example:

\begin{verbatim}
G = gsp_cube();
gsp_plot_graph(G);
axis square
\end{verbatim}

This code produces the following output:

WARNING : The matrix W is not symmetric ! Artificial symmetrization

---

3.1.16 GSP\_LOGO - Initialize a graph with the GSP logo

Usage

\begin{verbatim}
G = gsp_logo();
\end{verbatim}

Output parameters

- \textbf{G} Graph structure.

Description

`gsp_logo()` initializes a graph structure containing the GSP logo

Example:

\begin{verbatim}
G = gsp_logo();
gsp_plot_graph(G);
\end{verbatim}
3.1.17  GSP_COMMUNITY - Create a community graph

Usage

\[
G = \text{gsp\_community}(N);
G = \text{gsp\_community}();
G = \text{gsp\_community}(N, \text{param});
\]

Input parameters

- \textbf{N} Number of nodes (default 256)
- \textbf{param} Structure of optional parameters

Output parameters

- \textbf{G} Graph

Description

This function create a 2 dimensional random sensor graph. All the coordonates are between 0 and 1.

\textit{param} is an optional structure with the following fields

- \textit{param.Nc} : Number of communities (default round(sqrt(N)/2) )
- \textit{param.verbose}: display parameter - 0 no log - 1 display the errors (default 1)
- \textit{param.com_sizes} : size of the communities. The sum of the sizes has to be equal to \textit{N}. Leave this field empty if you want random sizes.
- \textit{param.min_comm} : Minimum size of the community (default round(N / param.Nc / 3) )
- \textit{param.min_deg} : Minimum degree of each nodes (default round(param.min_comm/2)) (NOT WORKING YET!)

\textbf{system-message}

\textbf{WARNING} in <string>, line 45

\textit{Inline emphasis start-string without end-string.}
3.1. SPECIFIC GRAPHS

- *param.size_ratio*: ratio between radius of world and radius of communities (default 1)

- *param.world_density* probability of a random edge between any pair of nodes (default 1/N)

Example:

```matlab
G = gsp_community();
paramplot.show_edges = 1;
gsp_plot_graph(G,paramplot);
```

3.1.18 GSP_BUNNY - Create a graph of the stanford bunny

**Usage**

```matlab
G = gsp_bunny();
```

**Output parameters**

- **G**
  Resulting graph

**Description**

`gsp_bunny()` creates a graph from the pointcloud of the Stanford Bunny model.

Example:

```matlab
G = gsp_bunny();
gsp_plot_graph(G);
```

*This code produces the following output:*

Average degree = 5.232920e+01
The matrix W is symmetric
3.2 Utils

3.2.1 GSP_GRAPH_DEFAULT_PARAMETERS - load default parameters for graphs

Usage

```matlab
G = gsp_graph_default_parameters( G );
G = gsp_graph_default_parameters();
```

Input parameters

- **G**: Graph (Optional)

Output parameters

- **G**: Graph

Description

This function will fill a graph with all missing parameters such that it is compatible with all functions of the GSPBox. If you create a graph manually, you need to set only the weight matrix $W$. If you have some coordinate, you can also set $G.coords$. $G.coords$ is a $N \times 2$ or a $N \times 3$ matrix with each column being the coordinates in each dimension. Finally, we recommend to set the field $G.type$ with a name that suits your graph.

Example:

```matlab
W = rand(30);
W = (W + W')/2;
G.W = W - diag(diag(W));
G = gsp_graph_default_parameters( G )
```

This code produces the following output:

```matlab
G =
    W: [30x30 double]
    A: [30x30 logical]
    N: 30
    type: 'unknown'
    directed: 0
    lap_type: 'combinatorial'
    L: [30x30 double]
    d: [1x30 double]
```
3.2. UTILS

Ne: 870
coords: [30x2 double]
plotting: [1x1 struct]

List of parameters of the graph structure

By default, a graph structure in the GSPBox contains the following parameters:

- $G.W$: Weight matrix (empty by default)
- $G.A$: Adacency matrix (constructed with $W$)
- $G.N$: Number of nodes ($\text{size}(W,1)$)
- $G.type$: Type of graph ("unknown" by default)
- $G.directed$: 1 if the graph is directed, 0 if not
- $G.lap_type$: Laplacian type (default "combinatorial") See the function `gsp_create_laplacian` for a exhaustive list of the available laplacians.
- $G.d$: Degree vector (Computed with $G.W$)
- $G.Ne$: Number of edges
- $G.coords$: Coordinates of the vertices (default (0,0))
- $G.plotting$: Plotting parameters
  - $G.plotting.edge_width$: Width of edges (default 1)
  - $G.plotting.edge_color$: Color of edges (default [255,88,41]/255)
  - $G.plotting.edge_style$: Style of edges (default \\)
  - $G.plotting.vertex_size$: Size of vertex (default 50)
  - $G.plotting.vertex_color$: Color of vertex (default 'b')

Remark: There is redundancy between $A$, $W$, $L$. However, the GSPBox is done in matlab and is not suppose yet to scale to graph sufficiently large that your matlab have memory problem. However, this will be most likely change for milestone 1.0.0. If you do have a urgent need to overcome this problem, please contact the devolper team.

3.2.2 GSP_GRAPH - Initialize a graph from a weight matrix

Usage

$G = \text{gsp\_graph}(N);$

Input parameters

- $W$: Weight matrix
- $coords$: Coordenates of the points (optional)
- $limits$: limits for the coordonates (optional)

Output parameters

- $G$: Graph structure.

Description

`gsp_graph(W,coords,limits)` initializes a graph structure with $W$ as weight matrix.

Example:

```matlab
W = rand(10);
W = W - diag(diag(W));
W = (W + W')/2;
G = gsp_graph(W);
```
Chapter 4

GSPBOX - Filters

4.1 Design - N filter

4.1.1 GSP_DESIGN_MEXICAN_HAT - Design the mexican hat filter bank

Usage

\[ g = \text{gsp\_design\_mexican\_hat}(G, Nf, \text{param}); \]
\[ \text{gsp\_design\_mexican\_hat}(G, Nf); \]
\[ \text{gsp\_design\_mexican\_hat}(G); \]

Input parameters

- **G**: Graph or upper bound on the Laplacian spectrum
- **Nf**: Number of filters to cover the interval \([0,lmax]\) (default 6)
- **param**: Structure of optional parameters

Output parameters

- **g**: A cell array of filters

Description

This function returns a array of filters designed to be mexican hat wavelet. The mexican hat wavelet is the second order derivative of a Gaussian. Since we express the filter in the Fourier domain, we find:

\[ f = \lambda^2. \]

In our convention the eigenvalues of Laplacian are equivalent to the square of vertex frequencies: \( f = \lambda^2 \).

The low pass filter is given by

**param** is an optional structure containing the following fields

- **param.t**: vector of scale to be used (default: log scale)
- **param.lpfactor**: \( \text{lmin} = \text{lmax}/\text{lpfactor} \) will be used to determine scales, then scaling function kernel will be created to fill the lowpass gap. (default 20)
- **param.verbose**: verbosity level. 0 no log - 1 display warnings. (default 1)

This function will compute the maximum eigenvalue of the laplacian. To be more efficient, you can precompute it using:

\[ G = \text{gsp\_estimate\_lmax}(G); \]

Example:
\begin{verbatim}
Nf = 4;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_mexican_hat(G, Nf);
gsp_plot_filter(G,g);
\end{verbatim}

This function is inspired by the ssgt_toolbox.

4.1.2 GSP_DESIGN_ABSPLINE - Design the mexican hat filter bank

Usage

\begin{verbatim}
g = gsp_design_abspline(G, Nf, param);
gsp_design_abspline(G , Nf);
gsp_design_abspline(G);
\end{verbatim}

Input parameters

- \textbf{G} \hspace{1cm} Graph or upper bound on the Laplacian spectrum
- \textbf{Nf} \hspace{1cm} Number of filters to cover the interval \([0, \text{lmax}]\) (default 6)
- \textbf{param} \hspace{1cm} Structure of optional parameters

Output parameters

- \textbf{g} \hspace{1cm} A cell array of filters

Description

This function return a array of filters designed to be AB spline wavelet. The AB spline wavelet is ...

In our convention the eigenvalues of Laplacian are equivalent to the square of vertex frequencies: \(f = \lambda^2\).

The low pass filter is given by

\textit{param} is an optional structure containing the following fields

- \textit{param.t}: vector of scale to be used (default: log scale)
- \textit{param.lpfactor}: \(lmin* = *lmax/\text{lpfactor}\) will be used to determine scales, then scaling function kernel will be created to fill the lowpass gap. (default 20)
• `param.verbose`: verbosity level. 0 no log - 1 display warnings. (default 1)

This function will compute the maximum eigenvalue of the laplacian. To be more efficient, you can precompute it using:

\[ G = \text{gsp\_estimate\_lmax}(G); \]

Example:

```matlab
Nf = 4;
G = gsp\_sensor(100);
G = gsp\_estimate\_lmax(G);
g = gsp\_design\_abspline(G, Nf);
gsp\_plot\_filter(G,g);
```

This function is inspired by the sgwt_toolbox.

### 4.1.3 GSP\_DESIGN\_MEYER - Design the meyer filterbank

**Usage**

```matlab
\begin{align*}
g & = \text{gsp\_design\_meyer}(G, Nf, \text{param}); \\
gsp\_design\_meyer(G, Nf); \\
gsp\_design\_meyer(G);
\end{align*}
```

**Input parameters**

- **G**: Graph or upper bound on the Laplacian spectrum
- **Nf**: Number of filters to cover the interval \([0, \text{lmax}]\) (default 6)
- **param**: Structure of optional parameters

**Output parameters**

- **g**: A cell array of filters
CHAPTER 4. GSPBOX - FILTERS

Description

This function return a array of filters designed to be meyer wavelet. 

\( \text{param} \) is an optional structure containing the following fields

- \( \text{param.t} \): vector of scale to be used (default: log scale)
- \( \text{param.verbose} \): verbosity level. 0 no log - 1 display warnings. (default 1)

This function will compute the maximum eigenvalue of the laplacian. To be more efficient, you can precompute it using:

\[ G = \text{gsp\_estimate\_lmax}(G); \]

Example:

\[
\begin{align*}
Nf &= 4; \\
G &= \text{gsp\_sensor}(100); \\
G &= \text{gsp\_estimate\_lmax}(G); \\
g &= \text{gsp\_design\_meyer}(G, Nf); \\
g &= \text{gsp\_plot\_filter}(G, g);
\end{align*}
\]

This function is inspired by the sgwt_toolbox.

4.1.4 GSP_DESIGN_SIMPLE_TF - Design a simple tight frame filterbank

Usage

\[
\begin{align*}
g &= \text{gsp\_design\_simple\_tf}(G, Nf, \text{param}); \\
g &= \text{gsp\_design\_simple\_tf}(G, Nf); \\
g &= \text{gsp\_design\_simple\_tf}(G);
\end{align*}
\]

Input parameters

- \( G \) Graph or upper bound on the Laplacian spectrum
- \( Nf \) Number of filters to cover the interval \([0, \text{lmax}]\) (default 6)
- \( \text{param} \) Structure of optional parameters

Output parameters

- \( g \) A cell array of filters
4.1. DESIGN - N FILTER

Description

This function return a array of filters designed to be simple tight frame wavelet filterbank. 

*param* is an optional structure containing the following fields

- *param.t*: vector of scale to be used (default: log scale)
- *param.verbose*: verbosity level. 0 no log - 1 display warnings. (default 1)

This function will compute the maximum eigenvalue of the laplacian. To be more efficient, you can precompute it using:

```matlab
G = gsp_estimate_lmax(G);
```

Example:

```matlab
Nf = 4;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_simple_tf(G, Nf);
gsp_plot_filter(G,g);
```

This function is inspired by the sgwt_toolbox.

4.1.5 GSP_DESIGN_ITERSINE - Create a itersine filterbanks

Usage

```matlab
  g = gsp_design_ITERSINE( G, Nf );
g = gsp_design_ITERSINE( G, Nf, param );
```

Description

**Inputs parameters:** G : Graph structure or lmax Nf : Number of filter param : Structure of optional parameters

**Outputs parameters:** g : filterbanks

This function create a itersine half overlap filterbank of Nf filters Going from 0 to $\lambda_{max}$

This filterbank is tight for an overlap of 2 and other particular values. The function normalizes the window such that the framebound is 1.

The itersine window between -0.5 and 0.5 is defined as
\[ g(t) = \sin \left( 0.5\pi \cos(\pi t)^2 \right) \]

This function will compute the maximum eigenvalue of the laplacian. To be more efficient, you can pre-compute it using:

```matlab
G = gsp_estimate_lmax(G);
```

Example:

```matlab
Nf = 20;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_itersine(G, Nf);
gsp_plot_filter(G,g);
[A,B] = gsp_filterbank_bounds(G,g)
```

This code produces the following output:

```
A =

1.0000

B =

1.0000
```

`param` is an optional structure containing the following fields:

- `param.verbose`: verbosity level. 0 no log - 1 display warnings. (default 1)
- `param.overlap`: Overlap : default 2

### 4.1.6 GSP_DESIGN_HALF_COSINE - Design uniform half cosine filterbank

Usage

```matlab
filters = gsp_design_half_cosine( Nf, UBT );
```
4.1. DESIGN - N FILTER

Description

**Inputs parameters:** G : Graph or maximum value  Nf : Number of filters

**Outputs parameters:** filters : Cell array of filters

This function generate a uniform half cosine filterbank. The main window

\[
\frac{1}{2} \left( 1 + \cos \left( 2\pi \left( \frac{x}{a} - \frac{1}{2} \right) \right) \right) \text{ for } 0 \leq x \leq a
\]

is translated uniformly to create the filterbank.

This function will compute the maximum eigenvalue of the laplacian. To be more efficient, you can precompute it using:

```matlab
G = gsp_estimate_lmax(G);
```

Example:

```matlab
figure(100);
Nf = 4;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp-design_half_cosine(G, Nf);
gsp_plot_filter(G, g);
```

*param* is an optional structure containing the following fields

- *param.verbose*: verbosity level. 0 no log - 1 display warnings. (default 1)

4.1.7 GSP_DESIGN_WARPED_TRANSLATES - Create a vertex frequency filterbank

**Usage**

```matlab
g = gsp-design_warped_translates( G, Nf );
g = gsp-design_warped_translates( G, Nf, param );
```

**Description**

**Inputs parameters:**  G : Graph structure (or lmax for 'log' and 'log_plus' only)  Nf : Number of filter

**Outputs parameters:**  g : filterbanks  wf : warped function
This function design filters that are warped versions of the uniform half cosine translates described above. This function will compute the maximum eigenvalue of the laplacian. To be more efficient, you can precompute it using:

```c
G = gsp_estimate_lmax(G);
```

Example:

```c
figure();
Nf = 10;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
G = gsp_spectrum_cdf_approx(G);
g = gsp_design_warped_translates(G, Nf);
gsp_plot_filter(G,g);
[A,B] = gsp_filterbank_bounds(G,g)
```

This code produces the following output:

```
A =

  1.1250

B =

  1.1250
```

```
param is an optional structure containing the following fields

- `param.verbose`: verbosity level. 0 no log - 1 display warnings. (default 1)
- `param.warping_type`: Create a warping function according two different methods (default 'spectrum_approximation'). Please read below for more information about this parameter.
- `param.log`: On top of the other warping add a log function. An alternative way to construct spectral graph wavelets. These are adapted to the specific spectrum, not just the length of the spectrum. The final warping function will be:

\[ \log(f(x)) \]
where the function $f(x)$ is defined by the attribute param.warping_type. Warning: Additional required inputs: param.warp_function.

- param.warp_function: To provide a special warping function. This parameter is used when param.warping_type is 'custom'.

- param.interpolation_type: select the interpolation type for the spectrum samples. You can choose 'pwl' (piece wise linear) or 'moncubic'. This attribute is used only when param.warping_type is 'spectrum_interpolation'. (default 'monocubic')

- param.filter: select the initial uniform filterbank 'half_cosine' or 'itersine'. See gsp_design_uniform_half_cosine and gsp_design_itersine for more information about those filterbank. If you want to use your personal filter, just put it there. For instance:

```matlab
param.filter = gsp_design_abspline(G,Nf);
```
(Default 'half_cosine')

- param.overlap: overlap of the initial filter. Works only with param.filter set to 'itersine'. For tight frame, input an even number (default 2).

### Warping methods

The different warping type available in param.warping_type are:

- 'spectrum_interpolation': Warping functions based on spectrum samples. From the samples, an approximation of the spectrum cdf is obtained by interpolation. Then this function is used for the warping. (i.e., like the filter banks [1] in Section 2, these are spectrum-adapted filter banks).
  
  If you use this method you need to specify the input param.approx_spectrum that contains two fields: param.approx_spectrum.x and param.approx_spectrum.y that are the point of the cumulative density distribution.

- 'spectrum_approximation': This function will compute an approximation of the cumulative density function of the graph Laplacian eigenvalues and use it as warping function. If you want to use the cdf later, you should precompute it using:

```matlab
G = gsp_spectrum_cdf_approx(G);
```

- 'custom': The user provide the warping function in the parameter: param.warp_function.

References: [10]

### 4.2 Design - 2 filter (LP - HP) tight filterbank

#### 4.2.1 GSP_DESIGN_REGULAR - Create a Simoncelli filterbank

**Usage**

```matlab
g = gsp_design_regular( G );
g = gsp_design_regular( G, param );
```

**Description**

Inputs parameters: $G$: Graph structure or lmax $\text{param}$: Structure of optional parameters

Outputs parameters: $G$: filterbank
This function creates a Parseval filterbank of 2 filters. The low-pass filter is defined by a function $f_l(x)$ between 0 and 2. For $d = 0$.

$$f_l = \sin \left( \frac{\pi}{4} x \right)$$

For $d = 1$

$$f_l = \sin \left( \frac{\pi}{4} \left( 1 + \sin \left( \frac{\pi}{2} (x - 1) \right) \right) \right)$$

For $d = 2$

$$f_l = \sin \left( \frac{\pi}{4} \left( 1 + \sin \left( \frac{\pi}{2} \sin \left( \frac{\pi}{2} (x - 1) \right) \right) \right) \right)$$

And so on for the other degrees $d$.

The high-pass filter is adapted to obtain a tight frame.

This function will compute the maximum eigenvalue of the Laplacian. To be more efficient, you can precompute it using:

```matlab
G = gsp_estimate_lmax(G);
```

Example:

```matlab
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_regular(G);
gsp_plot_filter(G,g);
[A,B] = gsp_filterbank_bounds(G,g)
```

This code produces the following output:

\[ A = 1.0000 \]

\[ B = 1.0000 \]

```
0  2  4  6  8  10  12  14  16
0.0  0.2  0.4  0.6  0.8  1.0  1.2  1.4
```

`param` is an optional structure containing the following fields:

- `param.verbose`: verbosity level. 0 no log - 1 display warnings. (default 1)
- `param.d`: Degree. See equation for more information. (default 3)
4.2. DESIGN - 2 FILTER (LP - HP) TIGHT FILTERBANK

4.2.2 GSP_DESIGN_HELD - Create a Held filterbank

Usage

\[ g = \text{gsp\_design\_held}(G); \]
\[ g = \text{gsp\_design\_held}(G, \text{param}); \]

Description

Inputs parameters: \( G \) : Graph structure or lmax \( \text{param} \) : Structure of optional parameters

Outputs parameters: \( g \) : filterbank

This function create a parseval filterbank of 2 filters. The low-pass filter is defined by a function \( f_l(x) \):

\[
f_l = \begin{cases} 
1 & \text{if } x \leq a \\
\sin(2\pi \mu(x)) & \text{if } a < x \leq 2a \\
0 & \text{if } x > 2a 
\end{cases}
\]

with

\[
\mu(x) = -1 + 24x - 144x^2 + 256x^3
\]

The high pass filter is adapted to obtain a tight frame.

This function will compute the maximum eigenvalue of the laplacian. To be more efficient, you can precompute it using:

\[
G = \text{gsp\_estimate\_lmax}(G);
\]

Example:

\[
G = \text{gsp\_sensor}(100);
G = \text{gsp\_estimate\_lmax}(G);
g = \text{gsp\_design\_held}(G);
gsp\_plot\_filter(G,g);
[A,B] = \text{gsp\_filterbank\_bounds}(G,g)
\]

This code produces the following output:

\[
A = \\
1.0000
\]

\[
B = \\
1.0000
\]
param is an optional structure containing the following fields

- param.verbose: verbosity level. 0 no log - 1 display warnings. (default 1)
- param.a: see equations above for this parameter. Note that the spectrum is scaled between 0 and 2 (default 2/3).

### 4.2.3 GSP_DESIGN_SIMONCELLI - Create a Simoncelli filterbank

**Usage**

```matlab
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_simoncelli(G);
gsp_plot_filter(G,g);
[A,B] = gsp_filterbank_bounds(G,g);
```

**Description**

**Inputs parameters:** G : Graph structure or lmax param : Structure of optional parameters

**Outputs parameters:** g : filterbank

This function creates a Parseval filterbank of 2 filters. The low-pass filter is defined by a function $f_l(x)$:

$$f_l = \begin{cases} 
1 & \text{if } x \leq a \\
\cos \left( \frac{\pi \log(\frac{x}{a})}{\log(2)} \right) & \text{if } a < x \leq 2a \\
0 & \text{if } x > 2a 
\end{cases}$$

The high-pass filter is adapted to obtain a tight frame.

This function will compute the maximum eigenvalue of the Laplacian. To be more efficient, you can precompute it using:

```matlab
G = gsp_estimate_lmax(G);
```

**Example:**

```matlab
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_simoncelli(G);
gsp_plot_filter(G,g);
[A,B] = gsp_filterbank_bounds(G,g);
```
This code produces the following output:

\[ A = 1.0000 \]

\[ B = 1.0000 \]

\[ \begin{align*}
  f_l(x) &= \begin{cases} 
    1 & \text{if } x \leq a \\
    \sqrt{1 - \frac{\sin(3\pi x)}{2}} & \text{if } a < x \leq \frac{5a}{3} \\
    0 & \text{if } x > \frac{5a}{3}
  \end{cases}
\end{align*} \]

\[ \text{The high pass filter is adapted to obtain a tight frame.} \]

\[ \text{param is an optional structure containing the following fields} \]

- \textit{param.verbose:} verbosity level. 0 no log - 1 display warnings. (default 1)
- \textit{param.a:} see equations above for this parameter. Note that the spectrum is scaled between 0 and 2 (default 2/3).

4.2.4 GSP\_DESIGN\_PAPADAKIS - Create a Simoncelli filterbank

\textbf{Usage}

\[ g = \text{gsp\_design\_papadakis}( G ); \]
\[ g = \text{gsp\_design\_papadakis}( G, \text{param} ); \]

\textbf{Description}

\textbf{Inputs parameters:} G : Graph structure or lmax \text{param} : Structure of optional parameters

\textbf{Outputs parameters:} g : filterbank

This function create a Parseval filterbank of 2 filters. The low-pass filter is defined by a function \( f_l(x) \):

\[ f_l = \begin{cases} 
  1 & \text{if } x \leq a \\
  \sqrt{1 - \frac{\sin(3\pi x)}{2}} & \text{if } a < x \leq \frac{5a}{3} \\
  0 & \text{if } x > \frac{5a}{3}
\end{cases} \]
This function will compute the maximum eigenvalue of the laplacian. To be more efficient, you can precompute it using:

```matlab
G = gsp_estimate_lmax(G);
```

Example:

```matlab
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_papadakis(G);
gsp_plot_filter(G,g);
[A,B] = gsp_filterbank_bounds(G,g)
```

This code produces the following output:

\[
\begin{align*}
A &= 1 \\
B &= 1
\end{align*}
\]

param is an optional structure containing the following fields

- **param.verbose**: verbosity level. 0 no log - 1 display warnings. (default 1)
- **param.a**: see equations above for this parameter. Note that the spectrum is scaled between 0 and 2 (default 3/4).

### 4.3 Application

#### 4.3.1 GSP_FILTER - shortcut function

This function is a shortcut to the function `gsp_filter_analysis`. Please use the documentation of `gsp_filter_analysis`.

#### 4.3.2 GSP_FILTER_ANALYSIS - Analysis operator of a gsp filterbank

Usage

```matlab
coeffs = gsp_filter_analysis(G, fi, signal);
coeffs = gsp_filter_analysis(G, fi, signal, param);
```
4.3. APPLICATION

Input parameters

- \( G \)  
  Graph structure.

- \( \mathbf{f} \)  
  Set of spectral graph filters.

- \( s \)  
  Graph signal to analyze.

- \( \text{param} \)  
  Optional parameter

Output parameters

- \( c \)  
  Transform coefficients

Description

`gsp_filter_analysis(G,\mathbf{f},\text{signal})` computes the transform coefficients of a signal \( f \), where the atoms of the transform dictionary are generalized translations of each graph spectral filter to each vertex on the graph.

\[
c = D^*f
\]

where the columns of \( D \) are \( g_{i,m} = T_i \hat{g}_m \), and \( T_i \) is a generalized translation operator applied to each filter \( \hat{g}_m(\cdot) \).

Each column of \( c \) is the response of the signal to one filter.

Example:

```matlab
Nf = 5;
param.distribute = 1;
G = gsp_sensor(256);
G = gsp_compute_fourier_basis(G);
param.log = 1;
g = gsp_design_warped_translates(G, Nf, param);
s = sign(G.U(:,2));
sf = gsp_vec2mat(gsp_filter_analysis(G,g,s),Nf);
paramplot.show_edges = 1;
figure()
subplot(221)
gsp_plot_signal(G,sf(:,2),paramplot);
subplot(222)
gsp_plot_signal(G,sf(:,3),paramplot);
subplot(223)
gsp_plot_signal(G,sf(:,4),paramplot);
subplot(224)
gsp_plot_signal(G,sf(:,5),paramplot);
```

This code produces the following output:

GSP_DESIGN_WARPED_TRANSLATES: has to compute the spectrum continuous density function.
Additional parameters

- `param.exact`: To use exact graph spectral filtering instead of the Chebyshev approximation. To use this option, the graph need the Fourier basis of the graph need to be computed (default 0).

- `param.cheb_order`: Degree of the Chebyshev approximation (default=30).

- `param.verbose`: Verbosity level (0 no log - 1 display warnings) (default 1).

References: [4]

4.3.3 GSP_FILTER_SYNTHESIS - Analysis operator of a gsp filterbank

Usage

```matlab
s = gsp_filter_synthesis(G, filter, c);
s = gsp_filter_synthesis(G, filter, c, param);
```

Input parameters

- `G`: Graph structure.
- `filter`: Set of spectral graph filters.
- `c`: Transform coefficients.
- `param`: Optional parameter.

Output parameters

- `signal`: Synthesis signal.

Description

`gsp_filter_synthesis(G, filters, c)` computes the synthesis operator for coefficient `c`, where the atoms of the transform dictionary are generalized translations of each graph spectral filter to each vertex on the graph.

\[ f = Dc \]

where the columns of \( D \) are \( g_{i,m} = T_i \hat{g}_m \), and \( T_i \) is a generalized translation operator applied to each filter \( \hat{g}_m(\cdot) \).
Each column of $c$ is the response of the signal to one filter.

Example:

```matlab
Nf = 4;
G = gsp_sensor(30);
G = gsp_estimate_lmax(G);
G = gsp_estimate_lmax(G);
g = gsp_design_mexican_hat(G, Nf);
f = zeros(G.N,1);
f(1) = 1;
f = G.L^2*f;
ff = gsp_filter_analysis(G,g,f);
f2 = gsp_filter_synthesis(G,g,ff);
paramplot.show_edges = 1;
figure()
subplot(211)
gsp_plot_filter(G,g)
subplot(223)
gsp_plot_signal(G,f,paramplot);
subplot(224)
gsp_plot_signal(G,f2,paramplot);
```

Additional parameters

- `param.exact` : To use exact graph spectral filtering instead of the Chebyshev approximation. To use this option, the graph need the Fourier basis of the graph need to be computed (default 0).

- `param.cheb_order` : Degree of the Chebyshev approximation (default=30).

- `param.verbose` : Verbosity level (0 no log - 1 display warnings) (default 1).

References: [4]

4.3.4 **GSP_FILTER_INVERSE** - Inverse operator of a gsp filterbank

Usage

```matlab
s = gsp_filter_inverse(G, filter, c);
s = gsp_filter_inverse(G, filter, c, param);
```
CHAPTER 4. GSPBOX - FILTERS

Input parameters

\[ G \quad \text{Graph structure.} \]
\[ \text{filter} \quad \text{Set of spectral graph filters.} \]
\[ c \quad \text{Transform coefficients} \]
\[ \text{param} \quad \text{Optional parameter} \]

Output parameters

\[ \text{signal} \quad \text{synthesis signal} \]

Description

'gsp_filter_inverse(G,filter,c)' computes the inverse operator for coefficients \( c \), where the atoms of the transform dictionary are generalized translations of each graph spectral filter to each vertex on the graph.

\[ f = (D^TD)^{-1}Dc \]

where the columns of \( D \) are \( g_{i,m} = T_i g_m \), and \( T_i \) is a generalized translation operator applied to each filter \( \hat{g}_m(\cdot) \).

Each column of \( c \) is the response of the signal to one filter.

Example:

\[
\begin{align*}
Nf & = 4; \\
G & = \text{gsp_sensor}(30); \\
G & = \text{gsp_estimate_lmax}(G); \\
G & = \text{gsp_estimate_lmax}(G); \\
g & = \text{gsp_design_mexican_hat}(G, Nf); \\
f & = \text{rand}(G.N,1); \\
f & = f/\text{norm}(f); \\
ff & = \text{gsp_filter_analysis}(G,g,f); \\
f2 & = \text{gsp_filter_inverse}(G,g,ff); \\
\text{norm}(f-f2)
\end{align*}
\]

This code produces the following output:

\[
\text{ans = 3.6318e-07}
\]

Additional parameters

- \text{param.cheb.order} : Degree of the Chebyshev approximation (default=30).
- \text{param.verbose} : Verbosity level (0 no log - 1 display warnings) (default 1).
- \text{param.tol} : Tolerance to stop iterating (default 1e-6)

This function is inspired by the sgwt_toolbox

References: [4]

4.4 Size Handling

4.4.1 GSP_MAT2VEC - vector to matrix transform

Usage

\[
\begin{align*}
d & = \text{gsp_mat2vec}(d); \\
[d, Nf] & = \text{gsp_mat2vec}(d);
\end{align*}
\]
4.5. UTILS

Input parameters

\[
d
\]

Description

Output parameter \( d \) : Data \( Nf \) : Number of filter

Reshape the data from the matrix form to the vector form

4.4.2 GSP_VEC2MAT - vector to matrix transform

Usage

\[
d = \text{gsp Vec2mat}( d, Nf );
\]

Input parameters

\[
d \quad \text{Data}
\]

\[
Nf \quad \text{Number of filter}
\]

Description

Output parameter \( d \) : Data

Reshape the data from the vector form to the matrix form

4.5 Utils

4.5.1 GSP_FILTERBANK_MATRIX - Create the matrix of the filterbank frame

Usage

\[
F = \text{gsp Filterbank matrix}( G, g \text{ param } );
\]

Input parameters

\[
G \quad \text{Graph}
\]

\[
g \quad \text{Filters}
\]

\[
\text{param} \quad \text{Structure of optional parameter}
\]

Output parameters

\[
F \quad \text{Frame}
\]

Description

This function create the matrix associated to the filterbank \( g \). The size of the matrix is \( MN \times N \), where \( M \) is the number of filters.

\textit{param} a Matlab structure containing the following fields:

\begin{itemize}
  \item \textit{param.verbose} : 0 no log, 1 print main steps, 2 print all steps. By default, it is 1.
\end{itemize}

4.5.2 GSP_WLOG_SCALES - compute logarithm scales for wavelet

Usage

\[
s = \text{gsp Wlog Scales}( lmin, lmax, Nscales );
\]
CHAPTER 4. GSPBOX - FILTERS

Input parameters

- \( l_{\text{min}} \): Minimum non-zero eigenvalue
- \( l_{\text{max}} \): Maximum eigenvalue
- \( \text{Nscales} \): Number of scales

Output parameters

- \( s \): Scale

Description

returns a (possibly good) set of wavelet scales given minimum nonzero and maximum eigenvalues of laplacian

returns scales logarithmically spaced between minimum and maximum "effective" scales: i.e. scales below minimum or above maximum will yield the same shape wavelet (due to homogeneity of kernel: currently assuming sgwt kernel \( g \) given as abspline with \( t_1=1, t_2=2 \))

Note that in design of transform with scaling function, \( l_{\text{min}} \) may be taken just as a fixed fraction of \( l_{\text{max}} \), and may not actually be the smallest nonzero eigenvalue

This function is inspired by the sgwt_toolbox.

4.5.3 GSP_FILTER_EVALUATE - Evaluate the filterbank

Usage

\[ fd = \text{gsp\_filter\_evaluate}(\text{filter}, x) \]

Input parameters

- \( \text{filter} \): Cell array of filter
- \( x \): Data

Output parameters

- \( fd \): Response of the filters

Description

This function apply all the filters in \( \text{filter} \) to the data \( x \). Every filter correspond to one column of the matrix \( fd \).

4.5.4 GSP_FILTERBANK_BOUNDS - Compute approximate frame bounds for a filterbank

Usage

\[
\begin{align*}
[A, B] &= \text{gsp\_filterbank\_bounds}(G, W); \\
[A, B] &= \text{gsp\_filterbank\_bounds}(G, W, \text{param}); \\
[A, B] &= \text{gsp\_filterbank\_bounds}([\text{xmin}, \text{xmax}], W); \\
[A, B] &= \text{gsp\_filterbank\_bounds}([\text{xmin}, \text{xmax}], W, \text{param});
\end{align*}
\]

Input parameters

- \( G \): Graph structure or interval to compute the bound
- \( W \): Filterbank (cell array of inline function)
- \( \text{param} \): Optional parameter
Output parameters

A  Filterbank lower bound
B  Filterbank Upper bound

Description

`param` is a Matlab structure containing the following fields:

- `param.N`: Number of point for the line search default (default 100)
- `param.use_eigenvalues`: Use eigenvalues if possible (default 1). To be used, the eigenvalues have to be computed first using `gsp_compute_fourier_basis`.

4.5.5  **GSP_TIGHTEN_FILTER - Create a function that tighten a filterbank**

**Usage**

```matlab
ftighten = gsp_tighten_filter( filters );
```

**Input parameters**

- `G`: Graph or maximum eigenvalue
- `filters`: Filters of the filterbank (cell array)

**Description**

**Output parameters**: `ftighten`: Inline function

This function will compute the maximum eigenvalue of the laplacian. To be more efficient, you can precompute it using:

```matlab
G = gsp_estimate_lmax(G);
```
Chapter 5

GSPBOX - Operators

5.1 Localisation

5.1.1 GSP_TRANSLATE - Translate the signal f to the node i

Usage

\[ ft = \text{gsp\_translate}(G, f, i); \]

Input parameters

- \( G \): Graph
- \( f \): Signal (column)
- \( i \): Indices of vertex (int)

Output parameters

- \( ft \): translate signal

Description

This function translates the column vector \( f \) onto the node \( i \). If \( f \) is a matrix, the translation will be done to each column.

5.1.2 GSP_MODULATE - Translate the signal f to the node i

Usage

\[ fm = \text{gsp\_modulate}(G, f, k); \]

Input parameters

- \( G \): Graph
- \( f \): Signal (column)
- \( k \): Indices of frequencies (int)

Output parameters

- \( fm \): Modulated signal
Description

This function modulate the column vector \( f \) onto the node \( i \). If \( f \) is a matrix, the modulation will be applied to each column.

5.1.3 GSP_LOCALIZE - Localize a kernel \( g \) to the node \( i \)

Usage

\[
ft = \text{gsp\_localize}(G, g, i);
\]

Input parameters

- \( G \): Graph
- \( g \): kernel (or filterbank)
- \( i \): Indices of vertex (int)

Output parameters

- \( gt \): translate signal

Description

This function localize the kernel \( g \) onto the node \( i \). If \( g \) is a cell array, the localization will be done to each filter.

5.2 Differential

5.2.1 GSP_GRAD_MAT - Gradient sparse matrix of the graph \( G \)

Usage

\[
D = \text{gsp\_gradient\_mat}(G);
\]

Input parameters

- \( G \): Graph structure

Output parameters

- \( D \): Gradient sparse matrix

Description

This function return the gradient matrix. To be more efficient, call the function:

\[
G = \text{gsp\_adj2vec}(G)
\]

before this function.

Example:

\[
N = 40;
G = \text{gsp\_sensor}(N);
G = \text{gsp\_adj2vec}(G);
D = \text{gsp\_grad\_mat}(G);
\]
5.2. DIFFERENTIAL

5.2.2  GSP_GRAD - Graph gradient

Usage

\[ gr = \text{gsp\_grad}(G, s) \]

Input parameters

\[ G \quad \text{Graph structure} \]
\[ s \quad \text{Signal living on the nodes} \]

Output parameters

\[ gr \quad \text{Gradient living on the edges} \]

Description

For the non normalized Laplacian, the gradient of the node signal \( f \) evaluated at the edge linking \( x \) and \( y \) is given by:

\[ \nabla f(x, y) = \sqrt{w(x, y)} (f(x) - f(y)) \]

Before using this function, you need to call the function:

\[ G = \text{gsp\_adj2vec}(G) \]

5.2.3  GSP_DIV - Graph divergence

Usage

\[ di = \text{gsp\_div}(G, s) \]

Input parameters

\[ G \quad \text{Graph structure} \]
\[ s \quad \text{Signal living on the edges} \]

Output parameters

\[ di \quad \text{Divergence} \]

Description

The divergence operator is the adjoint of the gradient operator. For graphs, the divergence of a signal residing on edges gives a signal living on the nodes. The result should be such that:

\[ \text{gsp\_div}(G, \text{gsp\_grad}(G, s)) = G.L \ast s, \]

Before using this function, you need to call the function:

\[ G = \text{gsp\_adj2vec}(G) \]

5.2.4  GSP_ADJ2VEC - Prepare the graph for the gradient computation

Usage

\[ [G] = \text{gsp\_adj2vec}(G) \]

Input parameters

\[ G \quad \text{Graph structure} \]
Output parameters

\( G \)
Graph structure

Description

This function converts adjacency matrix to edge vector form. It also add the field G.D that is the sparse gradient matrix

### 5.3 Transforms

#### 5.3.1 GSP_GFT - Graph Fourier transform

Usage

\[
\hat{f}(\lambda) = \langle f, u_\ell \rangle
\]

Input parameters

\( G \)
Graph or Fourier basis

\( f \)
f (signal)

Output parameters

\( \hat{f} \)
Graph Fourier transform of \( f \)

Description

`gsp_gft(G,f)` computes a graph Fourier transform of the signal \( f \) with respect to the Fourier basis of the graph \( G: G.U. \) Alternatively, one can provide directly the Fourier basis instead of the graph \( G \).

To compute the Fourier basis of a graph \( G \), you can use the function:

\[
G = gsp_compute_fourier_basis(G);
\]

Example:

```plaintext
N = 30;
G = gsp_sensor(N);
G = gsp_compute_fourier_basis(G);
f = sin((1:N)’*2*pi/N);
\hat{f} = gsp_gft(G,f);
gsp_plot_signal_spectral(G,\hat{f});
```
5.3.2 GSP_IGFT - Inverse graph Fourier transform

Usage

\[ f = \text{gsp\_igft}(G,f\_hat); \]

Input parameters

- \( G \) : Graph or Fourier basis
- \( f\_hat \) : Signal

Output parameters

- \( f \) : Inverse graph Fourier transform of \( f\_hat \)

Description

`gsp_igft(G,f_hat)` computes a graph Fourier transform of the signal \( f\_hat \) with respect to the Fourier basis of the graph \( G \). Alternatively, one can provide directly the Fourier basis instead of the graph \( G \).

\[ \hat{f}(\lambda_\ell) = \langle f, u_\ell \rangle \]

To compute the Fourier basis of a graph \( G \), you can use the function:

\[ G = \text{gsp\_compute\_fourier\_basis}(G); \]

Example:

\[
\begin{align*}
N &= 30; \\
G &= \text{gsp\_sensor}(N); \\
G &= \text{gsp\_compute\_fourier\_basis}(G); \\
f\_hat &= \text{zeros}(N,1); \\
f\_hat(5) &= 1; \\
f &= \text{gsp\_igft}(G,f\_hat); \\
\text{gsp\_plot\_signal}(G,f); 
\end{align*}
\]
5.3.3 GSP_GWFT - Graph windowed Fourier transform

Usage

\[ G = \text{gsp\_gwft}(G, g, f, \text{param}); \]
\[ G = \text{gsp\_gwft}(G, g, f); \]

Input parameters
\n\n- \( G \): Graph
- \( g \): Window (graph signal or kernel)
- \( f \): Graph signal (column vector)
- \( \text{param} \): Structure of optional parameter

Output parameters
\n\n- \( C \): Coefficient.

Description

This function computes the graph windowed Fourier transform of a signal \( f \) with the window \( g \). The function returns a matrix of size \( N^2 \times N \).

\( \text{param} \) is a Matlab structure containing the following fields:

- \( \text{param}.\text{verbose} \): 0 no log, 1 print main steps, 2 print all steps. By default, it is 1.
- \( \text{param}.\text{lowmemory} \): use less memory. By default, it is 1.

5.3.4 GSP_NGWFT - Normalized graph windowed Fourier transform

Usage

\[ G = \text{gsp\_ngwft}(G, f, g, \text{param}); \]
\[ G = \text{gsp\_ngwft}(G, f, g); \]
Input parameters

| Parameter | Description |
|-----------|-------------|
| \( G \)  | Graph       |
| \( f \)  | Graph signal |
| \( g \)  | Window      |
| \( \text{param} \) | Structure of optional parameter |

Output parameters

| Parameter | Description |
|-----------|-------------|
| \( C \)  | Coefficient |

Description

This function computes the normalized graph windowed Fourier transform of a signal \( f \) with the window \( g \). The function returns a matrix of size \( N^2 \times N \).

\( \text{param} \) a Matlab structure containing the following fields:

- \( \text{param.verbose} \): 0 no log, 1 print main steps, 2 print all steps. By default, it is 1.
- \( \text{param.lowmemory} \): use less memory. By default, it is 1.

5.4 Pyramid - Reduction

5.4.1 GSP_KRON_REDUCTION - Compute the kron reduction

Usage

```matlab
Gnew = gsp_kron_reduction( G, ind );
Wnew = gsp_kron_reduction( W, ind );
```

Input parameters

| Parameter | Description |
|-----------|-------------|
| \( G \)  | Graph structure or weight matrix |
| \( \text{ind} \) | Indices of the nodes to keep |

Output parameters

| Parameter | Description |
|-----------|-------------|
| \( \text{Gnew} \) | New graph structure or weight matrix |

Description

This function performs the Kron reduction of the weight matrix in the graph \( G \), with boundary nodes labeled by \( \text{ind} \). This function will create a new graph with a weight matrix \( \text{Wnew} \) that contains only boundary nodes and is computed as the Schur complement of the original matrix with respect to the selected indices.

If a matrix is given, then a matrix is returned

Example:

```matlab
N = 64;
param.distribute = 1;
param.Nc = 5;
param.regular = 1;
G = gsp_sensor(N,param);
ind = 1:2:N;
Gnew = gsp_kron_reduction( G, ind );
figure;
subplot(121)
```
```matlab
G = gsp_kron_pyramid(G, Nlevels);
Gs = gsp_kron_pyramid(G, Nlevels, param);
```

### References:
[2]

#### 5.4.2 GSP_KRON_PYRAMID - compute a pyramid of graphs using the kron reduction

**Usage**

```matlab
Gs = gsp_kron_pyramid( G, Nlevels);
Gs = gsp_kron_pyramid( G, Nlevels, param);
```

**Input parameters**

- **G**: Graph structure
- **Nlevels**: Number of level of decomposition
- **param**: Optional structure of parameters

**Output parameters**

- **Gs**: Cell array of graphs

**Description**

This function compute a pyramid of graph based on the Kron reduction. The indices are taken as the positive entry of the highest eigenvector.

*param* is a structure of optional parameters containing the following fields:

- **lambda**: Stability parameter. It add self loop to the graph to give the algorithm some stability (default: 0.025).
- **sparsify**: Sparsify the graph after the Kron reduction (default: 1).
- **epsilon**: Sparsification parameter if the sparsification is used (default: min(2/sqrt(G.N), 0.1)).
• **filters**: A cell array of filter that will be used for the analysis and synthesis operator. If only one filter is given, it will be used for all levels. You may change that later on. Default

\[
h(x) = \frac{0.5}{0.5 + x}
\]

Example:

```matlab
N = 256;
G = gsp_sensor(N);
Nlevel = 5;
Gs = gsp_kron_pyramid(G, Nlevel);
figure;
for ii = 1:numel(Gs)
    subplot(2,3,ii)
    gsp_plot_graph(Gs{ii})
    title(['Reduction level: ', num2str(ii-1)]);
end
```

Demo: gsp_demo_pyramid

References: [9]

### 5.4.3 GSP_PYRAMID_ANALYSIS - Compute the graph pyramid transform coefficients

**Usage**

```matlab
[ca,pe]=gsp_pyramid_analysis(Gs, f);
[ca,pe]=gsp_pyramid_analysis(Gs, f, param);
```

**Input parameters**

- **Gs**
  - A multiresolution sequence of graph structures.

- **f**
  - Graph signal to analyze.

- **param**
  - Structure of optional parameters
CHAPTER 5. GSPBOX - OPERATORS

Output parameters

- ca: Cell array with the coarse approximation at each level
- pe: Cell array with the prediction errors at each level

Description

'gsp_pyramid_analysis' computes the graph pyramid transform coefficients of the signal \( f \) for the pyramid structure in \( G_s \).

Demo: gsp_demo_pyramid

References: [9], [6]

5.4.4 GSP_PYRAMID_SYNTHESIS - Synthesizes a signal from its graph pyramid transform coefficients

Usage

```matlab
signal = gsp_pyramid_synthesis(Gs,coeff)
[signal, ca ] = gsp_pyramid_synthesis(Gs,coeff)
```

Input parameters

- Gs: A multiresolution sequence of graph structures.
- coeff: The coefficients to perform the reconstruction

Output parameters

- signal: The synthesized signal.
- ca: Cell array with the coarse approximation at each level

Description

This function perform the pyramid synthesis of the coefficient in \( \text{coeff} \).

The pyramid analysis operator returns two arguments:

```matlab
[ca,pe]=gsp_pyramid_analysis(Gs, f);
```

To obtain the coefficients you can call the function:

```matlab
coeff = gsp_pyramid_cell2coeff(ca,pe);
```

Example:

```matlab
N = 256;
Nl = 4;
G = gsp_sensor(N);
Gs = gsp_kron_pyramid(G,Nl);
Gs = gsp_compute_fourier_basis(Gs);
f = rand(N,1);
[ca,pe]=gsp_pyramid_analysis(Gs, f);
coeff = gsp_pyramid_cell2coeff(ca,pe);
f_pred = gsp_pyramid_synthesis(Gs,coeff);
error = norm(f-f_pred)
```

This code produces the following output:

```
error =
0
```

Demo: gsp_demo_pyramid

References: [9], [6]
5.4.5  **GSP_PYRAMID_CELL2COEFF** - Cell array to vector transform for the pyramid

**Usage**

```
coeff = gsp_pyramid_cell2coeff(ca, pe);
```

**Input parameters**

- `ca`: Cell array with the coarse approximation at each level
- `pe`: Cell array with the prediction errors at each level

**Output parameters**

- `coeff`: Vector of coefficient

**Description**

This function compress the cell array `ca` and `pe` into a single vector of coefficients. It keeps the smaller coarse approximation and the prediction errors.

Example:

```
[ca, pe] = gsp_pyramid_analysis(Gs, f);
coeff = gsp_pyramid_cell2coeff(ca, pe);
```

Demo: gsp_demo_pyramid

5.4.6  **GSP_TREE_MULTIRESOLUTION** - Compute a multiresolution of trees

**Usage**

```
[Gs, subsampled_vertex_indices] = gsp_tree_multiresolution(G, num_levels);
[Gs, subsampled_vertex_indices] = gsp_tree_multiresolution(G, num_levels, param);
```

**Input parameters**

- `G`: Graph structure of a tree.
- `Nlevel`: Number of times to downsample and coarsen the tree.

**Output parameters**

- `Gs`: Cell array, with each element containing a graph structure represent a reduced tree.
- `subsampled_vertex_indices`: Indices of the vertices of the previous tree that are kept for the subsequent tree.

**Description**

**Additional parameters:**

- `param.root`: The index of the root of the tree (default=1)
- `param.reduction_method`: The graph reduction method (default='resistance_distance')
- `param.compute_full_eigen`: To also compute the graph Laplacian eigenvalues for every tree in the sequence

`gsp_tree_multiresolution(G, num_levels)` computes a multiresolution of trees by repeatedly downsampling and performing a graph reduction. The downsampling is performed by keeping all vertices at even depths of the tree from the root vertex. Options for the graph reduction method include: 'unweighted', 'sum' (add the weight connecting a child node to its parent and the weight connecting the parent to the
grandparent and use that weight for the edge connecting the child to the grandparent in the new graph), or 'resistance_distance', which preserves the resistance distances by setting the new weights according to:

\[ W_{i,k} = \frac{1}{W_{i,j} + W_{j,k}} \]

where \( W_{i,j} \) is the weight connecting a child to its parent in the original tree, and \( W_{j,k} \) is the weight connecting the parent to the grandparent in the original tree.

### 5.5 Utils

#### 5.5.1 GSP_CREATE_LAPLACIAN - create the graph laplacian of the graph G

**Usage**

```plaintext
G = gsp_create_laplacian( G, type );
G = gsp_create_laplacian( G );
```

**Input parameters**
- `G`: Graph structure (or cell array of graph structure)
- `type`: Type of laplacian (string)

**Output parameters**
- `G`: Graph structure (or cell array of graph structure)

**Description**

This function create the graph laplacian of the graph G and store it into G. The variable `type` contains the different laplacian type.

- `combinatorial`: Non normalized laplacian. This is the default.
- `normalized`: Normalized laplacian
- `none`: No laplacian

#### 5.5.2 GSP_COMPUTE_FOURIER_BASIS - Compute the fourier basis of the graph G

**Usage**

```plaintext
G = gsp_full_eigen(G);
```

**Input parameters**
- `G`: Graph structure (or cell array of graph structure)
- `param`: structure of optional parameters

**Output parameters**
- `G`: Graph structure (or cell array of graph structure)
Description

'gsp_full_eigen(G)' computes a full eigendecomposition of the graph Laplacian $G.L$:

$$L = U \Lambda U^*$$

where $\Lambda$ is a diagonal matrix of the Laplacian eigenvalues. $G.e$ is a column vector of length $G.N$ containing the Laplacian eigenvalues. The function will store the basis $U$, the eigenvalues $e$, the maximum eigenvalue $l_{max}$ and $G.mu$ the coherence of the Fourier basis into the structure $G$.

Example:

```matlab
N = 50;
G = gsp_sensor(N);
G = gsp_compute_fourier_basis(G);
gsp_plot_signal(G,G.U(:,2));
```

References: [1]
Chapter 6

GSPBOX - Pointclouds

6.1  Utils

6.1.1  GSP_POINTCLOUD - Load models and return the points

Usage

P = gsp_pointcloud(name)
P = gsp_pointcloud(name, max_dim)

Input parameters

name  the name of the point cloud to load (‘airfoil’, ‘two_moons’, ‘bunny’)
max_dim  the maximum dimensionality of the points (only valid for two_moons)

Output parameters

P  set of points in a NxD with N the number of points and D the dimensionality of the pointcloud
info  optional additional information

Description

‘gsp_pointcloud(name, max_dim)’ load pointcloud data and format it in a unified way as a set of points with each dimension in a different column

Note that the bunny is the model from the Stanford Computer Graphics Laboratory see references.

References: [13]
Chapter 7

GSPBOX - Proximal operators

7.1 Gradient based proximal operators

7.1.1 GSP_PROX_TV - Proximal TV operator for graphs signal

Usage

\[
\text{sol} = \text{gsp\_prox\_tv}(x, \gamma, G, \text{param})
\]

Input parameters

- **x** \(\text{Input signal.}\)
- **gamma** \(\text{Regularization parameter.}\)
- **G** \(\text{Graph structure}\)
- **param** \(\text{Structure of optional parameters.}\)

Output parameters

- **sol** \(\text{Solution.}\)
- **info** \(\text{Structure summarizing informations at convergence}\)

Description

This function compute the TV proximal operator for graphs. The TV norm is the one norm of the gradient. The gradient is defined in the function [gsp_grad].

This function require the UNLocBoX to be executed.

\[
gsp\_prox\_tv(y, \gamma, \text{param}) \quad \text{solves:}
\]

\[
sol = \min_z \frac{1}{2} \| x - z \|_2^2 + \gamma \| x \|_{TV}
\]

param is a Matlab structure containing the following fields:

- **param.tol** : is stop criterion for the loop. The algorithm stops if

\[
\frac{n(t) - n(t-1)}{n(t)} < \text{tol},
\]

where \(n(t) = f(x) + 0.5 \| x - z \|_2^2\) is the objective function at iteration \(t\) by default, \(\text{tol} = 10^{-4}\).

- **param.maxit** : max. nb. of iterations (default: 200).
CHAPTER 7. GSPBOX - PROXIMAL OPERATORS

- **param.A**: Forward operator (default: Id). This parameter allows to solve the following problem

\[ \text{sol} = \min_{z} \frac{1}{2} \|x - z\|^2 + \gamma \|Ax\|_{TV} \]

- **param.At**: Adjoint operator (default: Id).

- **param.nu**: bound on the norm of the operator A (default: 1), i.e.

\[ \|Ax\|^2 \leq \nu \|x\|^2 \]

- **param.verbose**: 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)

- **param.use_matrix**: 1 use the matrix operation for the gradient. This faster but requires more memory (default: 1).

info is a Matlab structure containing the following fields:

- **info.algo**: Algorithm used

- **info.iter**: Number of iteration

- **info.time**: Time of execution of the function in sec.

- **info.final_eval**: Final evaluation of the function

- **info.crit**: Stopping criterion used

Demo: gsp_demo_graph_tv

### 7.1.2 GSP_PROX_TIK - Proximal tikhonov operator for graphs

**Usage**

\[
\text{sol} = \text{gsp_prox_tik}(x, \gamma, G, \text{param}) \\
\text{sol} = \text{gsp_prox_tik}(x, \gamma, G) \\
[\text{sol, info}] = \text{gsp_prox_tik}(...) \\
\]

**Input parameters**

- **x**: Input signal.

- **gamma**: Regularization parameter.

- **G**: Graph structure

- **param**: Structure of optional parameters.

**Output parameters**

- **sol**: Solution.

- **info**: Structure summarizing informations at convergence
7.2. FILTERBANK BASED PROXIMAL OPERATORS

Description
This function require the UNLocBoX to be executed.

\[ gsp\_prox\_tik(y, \gamma, \text{param}) \]

solves:

\[ \text{sol} = \min_z \frac{1}{2} \|x - z\|_2^2 + \gamma \|\nabla x\|_2^2 \]

Note the nice following relationship

\[ x^T L x = \|\nabla x\|_2^2 \]

param is a Matlab structure containing the following fields:

- **param.tol** : stop criterion for the loop. The algorithm stops if
  \[ \frac{n(t) - n(t-1)}{n(t)} < \text{tol}, \]
  where \( n(t) = f(x) + 0.5\|x - z\|_2^2 \) is the objective function at iteration \( t \) by default, \( \text{tol} = 10^{-4} \).
- **param.maxit** : max. nb. of iterations (default: 200).
- **param.verbose** : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- **param.A** : Forward operator (default: Id). This parameter allows to solve the following problem
  \[ \text{sol} = \min_z \frac{1}{2} \|x - z\|_2^2 + \gamma \|\nabla Ax\|_2^2 \]
- **param.At** : Adjoint operator (default: Id).
- **param.pcg** : Use the fast PCG algorithm (default 1).
- **param.use_matrix** : 1 use the matrix operation for the gradient. This faster but requires more memory (default 1).
- **param.nu** : bound on the norm of the operator \( A \) (default: 1), i.e.
  \[ \|Ax\|_2^2 \leq \nu \|x\|_2^2 \]

info is a Matlab structure containing the following fields:

- **info.algo** : Algorithm used
- **info.iter** : Number of iteration
- **info.time** : Time of execution of the function in sec.
- **info.final_eval** : Final evaluation of the function
- **info.crit** : Stopping criterion used

7.2 Filterbank based proximal operators

7.2.1 GSP_PROJ_B2_FILTERBANK - Projection on the B2 ball for a filterbank

Usage

\[ \text{sol} = \text{gsp\_proj\_b2\_filterbank}(x, T, G, W, \text{param}); \]
\[ \text{sol} = \text{gsp\_proj\_b2\_filterbank}(x, T, G, W); \]
\[ [\text{sol}, \text{info}] = \text{gsp\_proj\_b2\_filterbank}(\ldots) \]
Input parameters

- **x**: Input signal.
- **gamma**: Compatibility parameter
- **G**: Graph structure
- **W**: Filterbank (cell array of functions)
- **param**: Structure of optional parameters.

Output parameters

- **sol**: Solution.
- **info**: Structure summarizing informations at convergence

Description

This function require the UNLocBoX to be executed.

\[
gsp_{proj\_b2\_filterbank}(x, \text{gamma, G, W, param})\text{ can solves:}
\]

\[
sol = \min_z \frac{1}{2} \|x - z\|_2^2 \text{ s. t. } \|WAx - y\|_2 < \varepsilon
\]

Where \(W\) is the linear analysis operator associated with the filterbank. In this case, we solve the problem on the signal side. Alternatively, we can also solve it on the coefficient side.

\[
sol = \min_z \frac{1}{2} \|x - z\|_2^2 \text{ s. t. } \|AW^*x - y\|_2 < \varepsilon
\]

You can select the problem you want to solve by setting `param.type` to ’coefficient’ or ’signal’.

`param` is a Matlab structure containing the following fields:

- **param.tight**: 1 if \(AW^*\) is a tight frame or 0 if not (default = 0)
- **param.y**: measurements (default: 0).
- **param.tol**: is stop criterion for the loop. The algorithm stops if

\[
\frac{n(t) - n(t-1)}{n(t)} < tol,
\]

where \(n(t) = f(x) + 0.5\|x - z\|_2^2\) is the objective function at iteration \(t\) by default, \(tol=10^{-4}\).

- **param.maxit**: max. nb. of iterations (default: 200).
- **param.verbose**: 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- **param.weights**: weights for a weighted L2-norm (default = 1)
- **param.A**: Forward operator (default: Id).
- **param.At**: Adjoint operator (default: A).
- **param.nu**: bound on the norm of the operator A (default: 1), i.e.

\[
\|Ax\|^2 \leq \nu\|x\|^2
\]

- **param.epsilon**: Radius of the L2 ball (default = 1e-3).
- **param.type**: ’coefficient’ or signal ’signal’ select the problem type. (default ’signal’).

`info` is a Matlab structure containing the following fields:
7.2. FILTERBANK BASED PROXIMAL OPERATORS

- info.algo : Algorithm used
- info.iter : Number of iteration
- info.time : Time of execution of the function in sec.
- info.final_eval : Final evaluation of the function
- info.crit : Stopping criterion used

7.2.2 GSP_PROX_L2_FILTERBANK - Proximal L2 operator for a filterbank

Usage

sol = gsp_prox_l2_filterbank(x, T, G, W, param);
sol = gsp_prox_l2_filterbank(x, T, G, W);
[sol, info] = gsp_prox_l2_filterbank(...)

Input parameters

- x : Input signal.
- gamma : Regularization parameter
- G : Graph structure
- W : Filterbank (cell array of functions)
- param : Structure of optional parameters.

Output parameters

- sol : Solution.
- info : Structure summarizing information at convergence

Description

This function requires the UNLocBoX to be executed.

gsp_prox_l2_filterbank(x, gamma, G, W, param) solves:

sol = \min_{z} \frac{1}{2} \|x - z\|^2 + \gamma \|AW^*x - y\|^2

Where W is the linear analysis operator associated with the filterbank.

param is a Matlab structure containing the following fields:

- param.tight : 1 if A is a tight frame or 0 if not (default = 0)
- param.y : measurements (default: 0).
- param.tol : is stop criterion for the loop. The algorithm stops if

\[ \frac{n(t) - n(t-1)}{n(t)} < tol, \]

where \( n(t) = f(x) + 0.5 \|x - z\|^2 \) is the objective function at iteration \( t \) by default, \( tol = 10^{-4} \).

- param.maxit : max. nb. of iterations (default: 200).
- param.verbose : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- param.weights : weights for a weighted L2-norm (default = 1)
• param.A: Forward operator (default: Id).
• param.At: Adjoint operator (default: A).
• param.nu: bound on the norm of the operator A (default: 1), i.e.

\[ \|Ax\|^2 \leq \nu\|x\|^2 \]

info is a Matlab structure containing the following fields:
• info.algo: Algorithm used
• info.iter: Number of iteration
• info.time: Time of execution of the function in sec.
• info.final_eval: Final evaluation of the function
• info.crit: Stopping criterion used

7.2.3 GSP_PROX_L1_FILTERBANK - Proximal L1 operator for a filterbank

Usage

```matlab
sol = gsp_prox_l1_filterbank(x, T, G, W, param);
sol = gsp_prox_l1_filterbank(x, T, G, W);
[sol, info] = gsp_prox_l1_filterbank(...)
```

Input parameters

- x: Input signal.
- gamma: Regularization parameter.
- G: Graph structure
- W: Filterbank (cell array of functions)
- param: Structure of optional parameters.

Output parameters

- sol: Solution.
- info: Structure summarizing information at convergence

Description

This function requires the UNLocBoX to be executed.

\[ \text{gsp_prox_l1_filterbank}(x, \gamma, G, W, \text{param}) \text{ solves:} \]

\[ \text{sol} = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|Wz\|_1 \]

Where \( W \) is the linear analysis operator associated with the filterbank.

param is a Matlab structure containing the following fields:
• param.tight: 1 if A is a tight frame or 0 if not (default = 0)
• param.tol: stop criterion for the loop. The algorithm stops if

\[ \frac{n(t) - n(t-1)}{n(t)} < \text{tol}, \]

where \( n(t) = f(x) + 0.5\|x-z\|_2^2 \) is the objective function at iteration \( t \) by default, \( \text{tol}=10^{-4} \).
7.2. FILTERBANK BASED PROXIMAL OPERATORS

- **param.maxit**: max. nb. of iterations (default: 200).
- **param.verbose**: 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- **param.weights**: weights for a weighted L1-norm (default = 1)

info is a Matlab structure containing the following fields:

- **info.algo**: Algorithm used
- **info.iter**: Number of iteration
- **info.time**: Time of execution of the function in sec.
- **info.final_eval**: Final evaluation of the function
- **info.crit**: Stopping criterion used
Chapter 8

GSPBOX - Utils

8.1 Connectivity

8.1.1 GSP_CHECK_CONNECTIVITY - Check if the graph G is aperiodic strongly connected

Usage

```matlab
bool=gsp_check_connectivity( G );
bool=gsp_check_connectivity( L );
bool=gsp_check_connectivity( W );
[bool,in,out]=gsp_check_connectivity( ... );
```

Input parameters

- **G, W, L**: Graph, Laplacian matrix or Weight matrix
- **param**: Optional parameters

Output parameters

- **bool**: Boolean
- **in**: Nodes without any in connections
- **out**: Nodes without any out connections

Description

Test if each node have at least one in connection and one out connection. If this simple test give good results, the function compute the perron vector of G and test it. It might take some time.

`param` is an optional structure that contains the following field

- `param.verbose`: display parameter - 0 no log - 1 display the errors

8.1.2 GSP_CHECK_CONNECTIVITY_UNDIRECTED - Check if the graph G is aperiodic strongly connected

Usage

```matlab
bool = gsp_check_connectivity_undirected( G );
bool = gsp_check_connectivity_undirected( L );
bool = gsp_check_connectivity_undirected( W );
[bool,in]=gsp_check_connectivity_undirected( ... );
```
Input parameters

- **G, W, L**: Graph, Laplacian matrix or Weight matrix
- **param**: Optional parameters

Output parameters

- **bool**: Boolean
- **in**: Nodes without any in connections

Description

Test if each node have at least one in connection and one out connection. If this simple test give good results, the function compute the perron vector of G and test it. It might take some time.

- **param**: an optional structure that contains the following field
  - **param.verbose**: display parameter - 0 no log - 1 display the errors

### 8.2 Norms

#### 8.2.1 GSP_NORM_TV - TV norm on graph

**Usage**

```matlab
y = gsp_norm_tv(G,x);
```

**Input parameters**

- **G**: Graph structure
- **x**: Signal on graph

**Output parameters**

- **y**: Norm

**Description**

Compute the TV norm of a signal on a graph.

#### 8.2.2 GSP_NORM_TIK - Squared L2 norm of the gradient on graph

**Usage**

```matlab
y = gsp_norm_tv(G,x);
```

**Input parameters**

- **G**: Graph structure
- **x**: Signal on graph

**Output parameters**

- **y**: Norm

**Description**

Compute the squared L2 norm of the gradient on graph. If x is a matrix a vector of norm is returned.
8.3. DISTANCE

8.2.3 GSP_NORM_L1_FILTERBANK - Compute the l2 norm of the analysis coefficients

Usage

\[ n = \text{gsp\_norm\_l2\_filterbank}(G, W, x); \]

Input parameters

- \( G \): Graph structure
- \( W \): Filterbank (cell array of functions)
- \( x \): coefficients
- \( \text{param} \): structure of optional parameter

Output parameters

- \( n \): L2 norm

Description

\[ n = \| AW^* x - y \|_2^2 \]

\( \text{param} \) is a Matlab structure containing the following fields:

- \( \text{param}.A \): Forward operator (default: Id).
- \( \text{param}.y \): measurements (default: 0).

8.3 Distance

8.3.1 GSP_RESISTANCE_DISTANCE - : Compute the resistance distances of a graph

Usage

\[ \text{rd} = \text{gsp\_resistance\_distance}(G); \]
\[ \text{rd} = \text{gsp\_resistance\_distance}(L); \]

Input parameters

- \( G \): Graph structure or Laplacian matrix (L)
- \( \text{param} \): optional parameters

Output parameters

- \( \text{rd} \): distance matrix
Description

This function computes the resistance distance of a graph. The distance between two nodes is defined as the inverse of the weight matrix. For example, the distance matrix:

```plaintext
dist = [0, 3, 1;...
      3, 0, 2;...
      1, 2, 0];
```

Corresponds to the weight matrix:

```plaintext
W = [0, 1/3, 1/1;...
     1/3, 0, 1/2;...
     1/1, 1/2, 0];
```

The function will compute the resistance distance following Kirchoff’s law. In our example, it is:

```plaintext
rd2 = [0, 3/2, 5/6;...
      3/2, 0, 4/3;...
      5/6, 4/3, 0]
```

In MATLAB, you can reproduce this example using:

```matlab
% The weight
dist = [0, 3, 1;...
       3, 0, 2;...
       1, 2, 0];
% The weight is the inverse of the distance...
W = dist.^(-1);
% Fix the diagonal
W([1,5,9])=0;
G = gsp_graph(W);
rd = gsp_resistance_distance(G)
% Resistance computed by hand
rd2 = [0, 3/2, 5/6;...
      3/2, 0, 4/3;...
      5/6, 4/3, 0]
```

This code produces the following output:

```plaintext
rd =

0   1.5000   0.8333
1.5000   0   1.3333
0.8333   1.3333   0

rd2 =

0   1.5000   0.8333
1.5000   0   1.3333
0.8333   1.3333   0
```

`param` is an optional structure that contains the following field:

- `param.verbose`: display parameter - 0 no log - 1 display warnings (default 1)

References: [5]

8.3.2 GSP_DISTANZ - calculates the distances between all vectors in x and y

Usage

```plaintext
d = gsp_distanz(x,y);
```
8.4. CHEBYSHEFF

Input parameters
- \( x \) matrix with col vectors
- \( y \) matrix with col vectors (default == \( x \))

Output parameters
- \( d \) distance matrix, not squared

Description
This code uses the Euclidean distance!
This code is not optimized for memory, but for speed because it uses no loops.

8.4 Chebysheff

8.4.1 GSP_CHEBY_COEFF - : Compute Chebyshev coefficients for a filterbank

Usage
\[
c = \text{gsp\_cheby\_coeff}(G, \text{filter}, m, N);
c = \text{gsp\_cheby\_coeff}(G, \text{filter}, m);
c = \text{gsp\_cheby\_coeff}(G, \text{filter});
\]

Input parameters
- \( G \) graph structure or range of application
- \( \text{filter} \) filter or cell array of filters
- \( m \) maximum order Chebyshev coefficient to compute (default 30)
- \( N \) grid order used to compute quadrature (default is \( m + 1 \))
- \( \text{param} \) optional parameter

Output parameters
- \( c \) matrix of Chebyshev coefficients

Description
This function computes the Chebyshef coefficients for all the filters contained in the cell array \( \text{filter} \). The coefficients are returned in a matrix. Every column corresponds to a filter. The coefficients are ordered such that \( c(j+1) \) is \( j \)th Chebyshev coefficient.

\( \text{param} \) contains only one field \( \text{param}.\text{verbose} \) to control the verbosity.

Example:
\[
\begin{align*}
N_f &= 4; \\
G &= \text{gsp\_sensor(100)}; \\
G &= \text{gsp\_estimate\_lmax}(G); \\
g &= \text{gsp\_design\_meyer}(G, N_f); \\
c &= \text{gsp\_cheby\_coeff}(G, g);
\end{align*}
\]

This function is inspired by the sgtw\_toolbox
8.4.2 **GSP_CHEBY_OP** - Chebyshev polynomial of graph Laplacian applied to vector

**Usage**

\[ r = \text{gsp_cheby_op}(G, c, \text{signal}) \]

**Input parameters**

- **G**: Graph structure
- **c**: Chebyshev coefficients
- **signal**: Signal to filter

**Output parameters**

- **r**: Result of the filtering

**Description**

Compute (possibly multiple) polynomials of graph laplacian (in Chebyshev basis) applied to input. Coefficients for multiple polynomials may be passed as a matrix. This is equivalent to setting:

\[
\begin{align*}
  r(1) &= \text{sgwt_cheby_op}(G, c(:,1), \text{signal}); \\
  r(2) &= \text{sgwt_cheby_op}(G, c(:,2), \text{signal}); \\
  \ldots
\end{align*}
\]

but is more efficient as the Chebyshev polynomials of \( G.L \) applied to \( \text{signal} \) can be computed once and shared.

The output \( r \) is a matrix with each column corresponding to a filter. 

**Example:**

```plaintext
Nf = 4;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_meyer(G, Nf);
c = gsp_cheby_coeff(G, g);
f = rand(G.N,1);
r = gsp_cheby_op(G, c, f);
```

This function is inspired by the sgwt_toolbox

8.5 **Others**

8.5.1 **GSP_GRAPH_SPARSIFY** - sparsify a graph using Spielman-Srivastava algorithm

**Usage**

\[ \text{Gnew} = \text{gsp_graph_sparsify}(G, \text{epsilon}) \]

**Input parameters**

- **G**: Graph structure or Laplacian matrix
- **epsilon**: Sparsification parameter
8.5. OTHERS

Description

Output parameters: Gnew : New sparsified graph or new laplacian

This function sparsifies a graph using Spielman-Srivastava algorithm. Note that epsilon should be between $1/\sqrt{N}$ and 1.

Example:

```matlab
epsilon = 0.4;
param.distribute = 1;
param.Nc = 20;
G = gsp_sensor(256,param);
G2 = gsp_graph_sparsify(G,epsilon);
figure(100);
gsp_plot_graph(G);
title('Original graph')
figure(101);
gsp_plot_graph(G2);
title('Sparsified graph')
close(100);
close(101);
```

References: [11], [7], [8]

8.5.2 GSP_REPMATLINE - This function repeat the matrix A in a specific manner

Usage

```matlab
Ar = gsp_repmatline( A, ncol, nrow );
```

Description

Inputs parameters A : Matrix ncol: Integer nrow: Integer

Outputs parameters Ar : Matrix

This function repeat a matrix line by line and column by column

For ncol=1 and nrow=2, the matrix 1 2 3 4
becomes 1 1 2 2 3 3 4 4

8.5.3 GSP_CLASSIC2GRAPH_EIG_ORDER - Compute the graph eigen value ordering

Usage

```matlab
v = gsp_classic2graph_eig_order(N)
```

Input parameters

N size of the graph

Output parameters

v vector of indexes

Description

This function make the link between the DFT and the ring graph. It returns the graph eigenvector ordering with respect of the DFT ordering.
8.5.4 GSP_RESET_SEED - Reset the seed

Usage

\[ \text{gsp\_reset\_seed}(\ n\ ); \]

Input parameters

\[ n \text{ seed} \]

Description

Output parameters: none

This function resets the seed.

8.5.5 GSP_PLOTFIG - Plotting figures with optimal size for paper

Usage

\[ \text{gsp\_plotfig}(\text{save\_name}); \]
\[ \text{gsp\_plotfig}(\text{save\_name},\text{param}); \]

Input parameters

\[ \text{save\_name} \text{ name to save the figure} \]
\[ \text{param} \text{ optional parameters} \]

Description

\[ \text{param} \text{ a Matlab structure containing the following fields:} \]

- \[ \text{param.pathfigure} \text{ path to the folder to save the figures} \]
- \[ \text{param.legendlocation} \text{ location of the figure (default 'Best');} \]
- \[ \text{param.position} \text{ position and size of the figure (default [100 100 600 400])} \]
- \[ \text{param.labelsize} \text{ Size of the label (default 12)} \]
- \[ \text{param.titlesize} \text{ Size of the title (default 16)} \]
- \[ \text{param.titleweight} \text{ Weight of the title (default 'normal')} \]
- \[ \text{param.save} \text{ Save the figure (default 1)} \]
- \[ \text{param.eps} \text{ Save the figure in eps instead of png (default 0)} \]

8.5.6 GSP_ISDIRECTED - Check is the graph is directed

Usage

\[ \text{bool} = \text{gsp\_isdirected}(\text{G}); \]
\[ \text{bool} = \text{gsp\_isdirected}(\text{W}); \]

Input parameters

\[ \text{G} \text{ Graph structure or square matrix} \]

Output parameters

\[ \text{bool} \text{ Boolean} \]
Description

This function test if the graph is directed. Alternatively, you can give a square matrix and it tests if it is symmetric. The function returns 0 if the matrix is symmetric and 1 otherwise!
Bibliography

[1] F. R. K. Chung. Spectral Graph Theory. Vol. 92 of the CBMS Regional Conference Series in Mathematics, American Mathematical Society, 1997.

[2] F. Dorfler and F. Bullo. Kron reduction of graphs with applications to electrical networks. Circuits and Systems I: Regular Papers, IEEE Transactions on, 60(1):150–163, 2013.

[3] D. Gleich. The MatlabBGL Matlab library. http://www.cs.purdue.edu/homes/dgleich/packages/matlab_bgl/index.html.

[4] D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. Appl. Comput. Harmon. Anal., 30(2):129–150, Mar. 2011.

[5] D. I. Klein and M. Randić. Resistance distance. Journal of Mathematical Chemistry, 12(1):81–95, 1993.

[6] I. Pesenson. Variational splines and paley–wiener spaces on combinatorial graphs. Constructive Approximation, 29(1):1–21, 2009.

[7] M. Rudelson. Random vectors in the isotropic position. Journal of Functional Analysis, 164(1):60–72, 1999.

[8] M. Rudelson and R. Vershynin. Sampling from large matrices: An approach through geometric functional analysis. Journal of the ACM (JACM), 54(4):21, 2007.

[9] D. I. Shuman, M. J. Faraji, and P. Vandergheynst. A framework for multiscale transforms on graphs. arXiv preprint arXiv:1308.4942, 2013.

[10] D. I. Shuman, C. Wiesmeyr, N. Holighaus, and P. Vandergheynst. Spectrum-adapted tight graph wavelet and vertex-frequency frames. arXiv preprint arXiv:1311.0897, 2013.

[11] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. SIAM Journal on Computing, 40(6):1913–1926, 2011.

[12] G. Strang. The discrete cosine transform. SIAM review, 41(1):135–147, 1999.

[13] G. Turk and M. Levoy. Zippered polygon meshes from range images. In Proceedings of the 21st annual conference on Computer graphics and interactive techniques, pages 311–318. ACM, 1994.