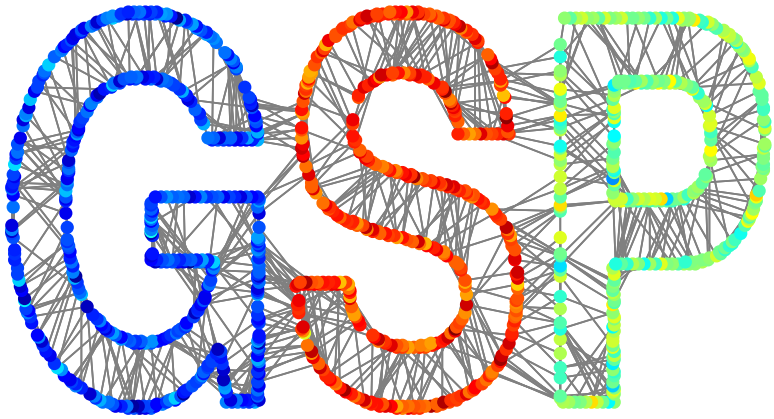


GSPBOX: A toolbox for signal processing on graphs

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

Contents

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 [6]. With paper, Hammond et al. provide another Maltab toolbox called sgwt. The GSPBox generalizes the sgwt as it contains all its the features.

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.
References: [9]

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();
```

Input parameters

install_unlcobox bool (1 to install the UNLocBoX)

Description

This function installs third party software. It require an internet connection.
You should install the UNLocBoX only if you are not using the developement version.
It will install the gaimc toolbox and compile some functions.

1.1.4 GSP_INSTALL_UNLOCBOX - Install the UNLocBoX from the web

Usage

```
gsp_install_unlocbox();
```

Description

This function installs the UNLocBoX. It require an internet connection.

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 lauch the toolbox, just go into the repository where the GSPBox was extracted and type:

```
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 adjacendy or the weight matrix W. Once you have it, you can construct a graph using:

```
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:

```
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 newtwork 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_random_ring             -  Create a random ring graph
     gsp_full_connected          -  Create a fully connected graph
     gsp_nn_graph                -  Create a nearest neighbors graph
     gsp_rmse_mv_graph           -  Create a nearest neighbors graph with missing values
```

```
  gsp_sphere                  -  Create a spherical-shaped graph
  gsp_cube                    -  Create a cubical-shaped graph
  gsp_2dgrid                  -  Create a 2d-grid graph
  gsp_torus                   -  Create a torus graph
  gsp_logo                    -  Create a GSP logo graph
  gsp_community               -  Create a community graph
  gsp_bunny                   -  Create a bunny graph
  gsp_spiral                  -  Create a spiral graph
  gsp_stochastic_block_graph  -  Create a graph with the stochastic block model

Hypergraphs
  gsp_nn_hypergraph           -  Create an hyper nearest neighbor graph
  gsp_hypergraph              -  Create an hypergraph

Utils
  gsp_graph_default_parameters-  Initialise all parameters for a graph
  gsp_graph_default_plotting_parameters-  Initialise all plotting parameters for a graph
  gsp_graph                   -  Create a graph from a weight matrix
  gsp_update_weights          -  Update the weights of a graph
  gsp_update_coordinates      -  Update the coordinate of a graph
  gsp_components              -  Cuts non connected graph into several connected ones
  gsp_subgraph                -  Create a subgraph


For help, bug reports, suggestions etc. please send email to
gspbox 'dash' support 'at' groupes 'dot' epfl 'dot' ch
```

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

```
G = gsp_logo
```

This code produces the following output:

```
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
  hypergraph: 0
     lap_type: 'combinatorial'
            L: [1130x1130 double]
            d: [1130x1 double]
           Ne: 3131
```

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

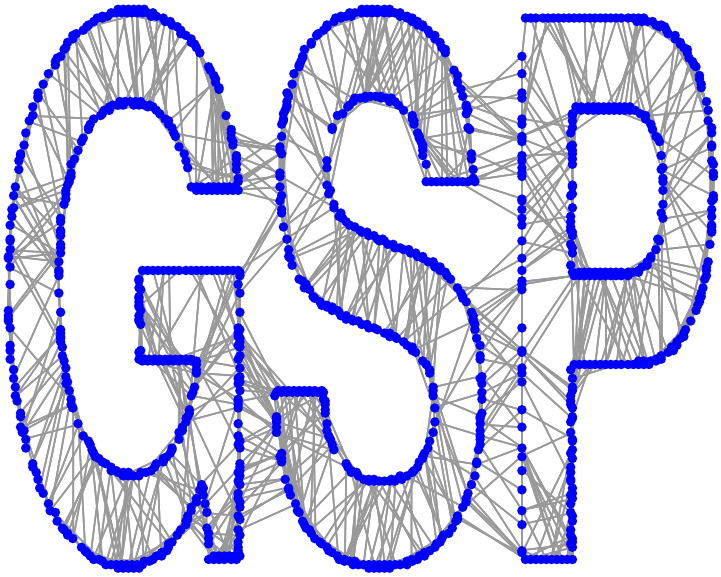


Figure 2.1: GSP graph
This figure shows the result of the command 'gsp_plot_graph(G)'

- *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:

```
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 by:

```
G = gsp_compute_fourier_basis(G);
```

The function gsp_compute_fourier_basis add 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 sinusoide on the graph. Let's plot the second and the third ones. (The first one is constant!):

```
gsp_plot_signal(G,G.U(:,2));
title('Second eigenvector')
subplot(212)
gsp_plot_signal(G,G.U(:,3));
title('Third eigenvector')
```

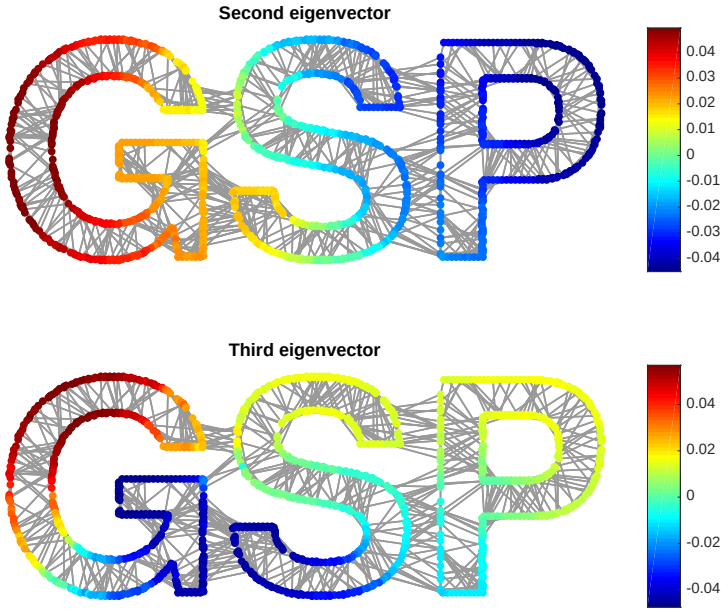


Figure 2.2: Eigenvectors

Now, we are going to show a basic filtering operation. Filters are usually defined in the spectral domain. To define the following filter

$$h(x) = \frac{1}{1+\tau x},$$

just write in Matlab:

```
tau = 1;
h = @(x) 1./(1+tau*x);
```

Hint: You can define filterbank using cell array!
Let's display this filter:

```
gsp_plot_filter(G,h);
```

To apply the filter to a given signal, you only need to run a single function:

```
f2 = gsp_filter(G,h,f);
```

`gsp_filter` is actually a shortcut to gsp_filter_analysis. `gsp_filter_analysis` performs the analysis operator associated to a filterbank. See the gsp_demo_wavelet for more information.
Finnaly, 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:

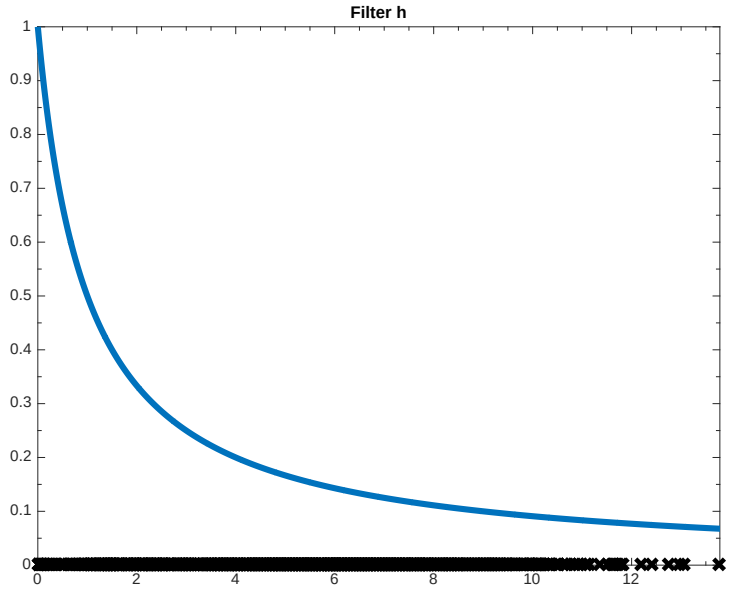


Figure 2.3: Low pass filter h

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.

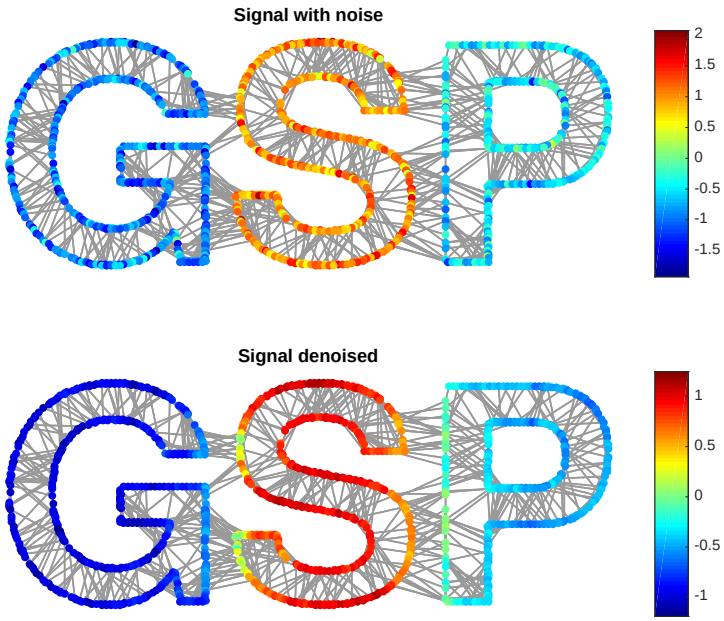


Figure 2.4: Result of filtering

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 = 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 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

```
taus = [1, 10, 25, 50];
Hk = gsp_design_heat(G, taus);
```

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

```
S = zeros(G.N, 1);
vertex_delta = 83;
S(vertex_delta) = 1;

Sf_vec = gsp_filter_analysis(G, Hk, S);
Sf = gsp_vec2mat(Sf_vec, length(taus));
```

and plot the filtered signal

```
param_plot.cp = [0.1223, -0.3828, 12.3666];

figure;
subplot(221)
gsp_plot_signal(G,Sf(:,1), param_plot);
axis square
title(sprintf('Heat diffusion tau = %d', taus(1)));
subplot(222)
gsp_plot_signal(G,Sf(:,2), param_plot);
axis square
title(sprintf('Heat diffusion tau = %d', taus(2)));
subplot(223)
gsp_plot_signal(G,Sf(:,3), param_plot);
axis square
title(sprintf('Heat diffusion tau = %d', taus(3)));
subplot(224)
gsp_plot_signal(G,Sf(:,4), param_plot);
axis square
title(sprintf('Heat diffusion tau = %d', taus(4)));
```

Visualizing wavelets atoms

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

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

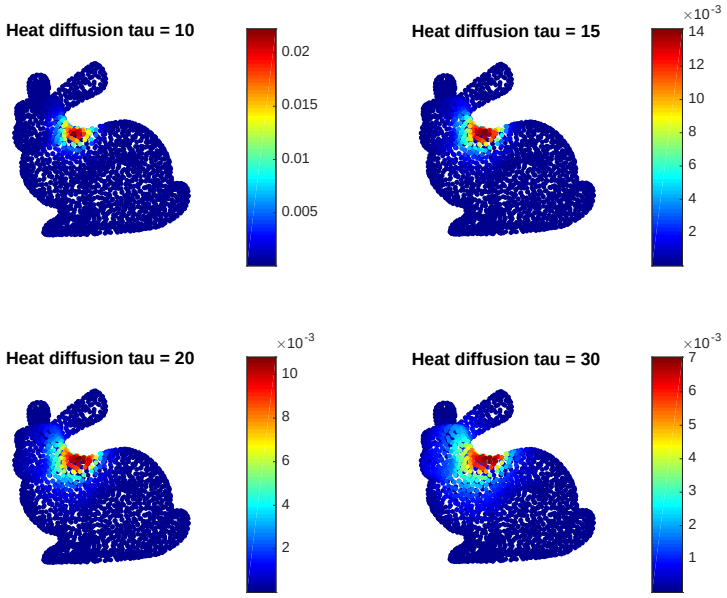


Figure 2.5: Heat diffusion at different scales

We can plot the filter bank spectrum

```
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

```
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;
```

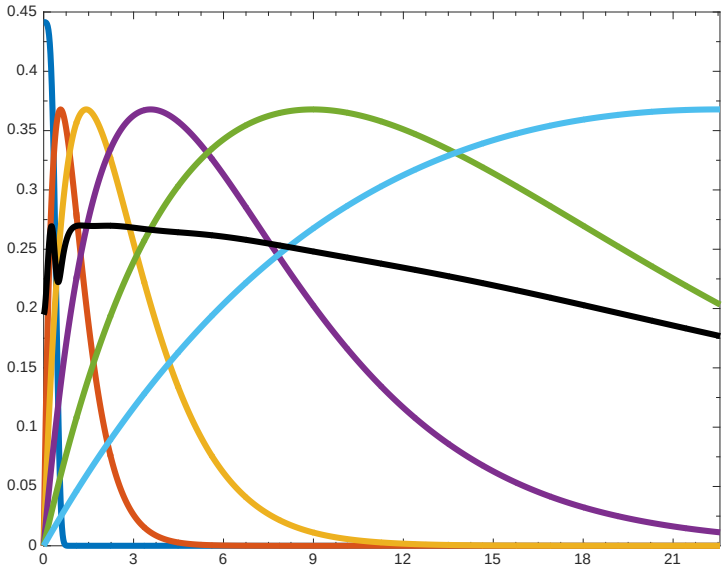


Figure 2.6: Wavelets filterbank (Original)

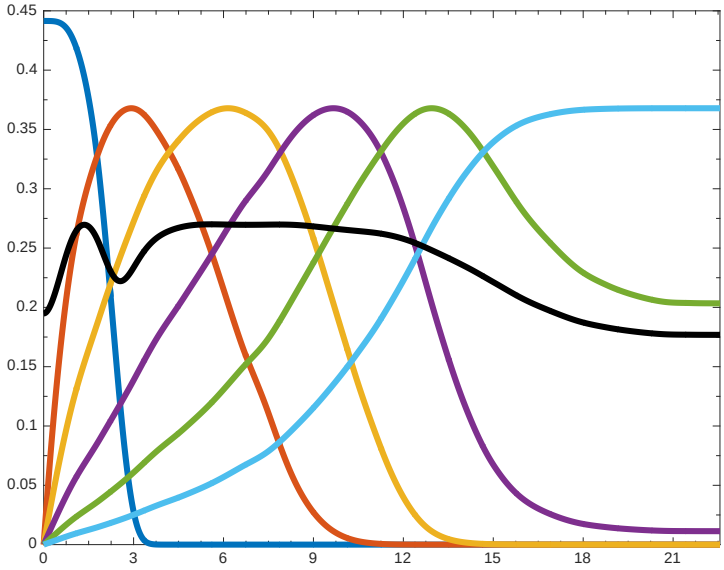


Figure 2.7: Wavelet filterbank (spectrum adaptated)

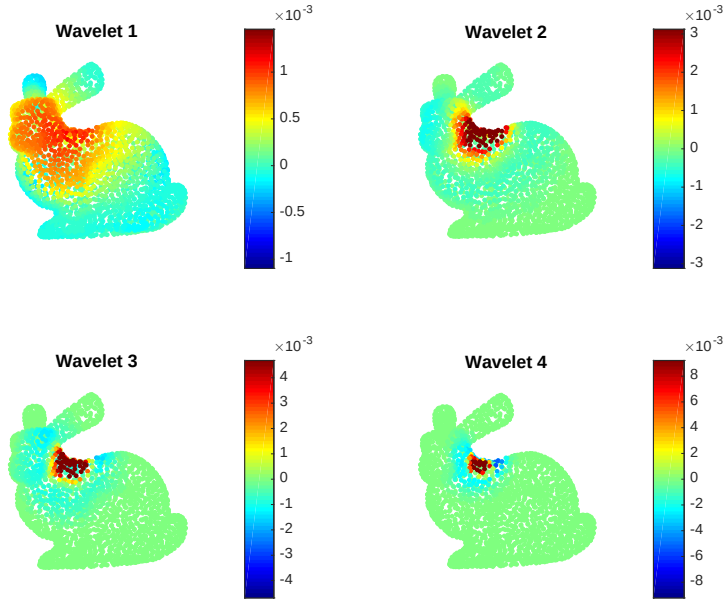


Figure 2.8: A few wavelets atoms

```
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

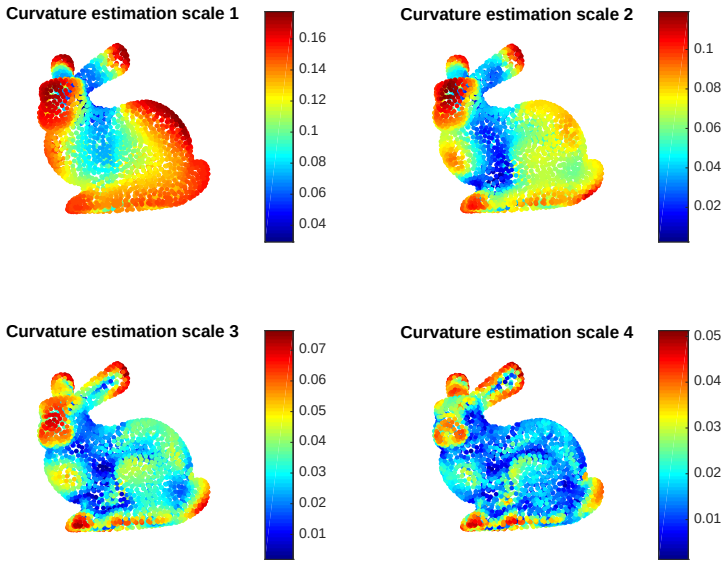


Figure 2.9: Curvature estimation using wavelet feature

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

```
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 number of connection = 3.211187e+01
The matrix W is symmetric
GSP_DESIGN_WARPED_TRANSLATES: has to compute the spectrum continuous density function approximation
```

2.1.3 GSP_DEMO_GRAPH_EMBEDDING - Demonstration file for graph embeddings

Description

In this demo we perform a low-dimensional embedding on a specific graph using three different algorithms and we plot the resulting embeddings on a single plot. At first we create a 2-D sensor graph embedded in a 3-D space. We then compute 3 different embeddings for this graph: Laplacian eigenmaps, LLE, and Isomap.

Laplacian eigenmaps

This function uses the Laplacian of the graph and the diagonal degree matrix to compute the eigenvalues and eigenvectors of generalized eigenvector problem, $LU = DU\Lambda$. The coordinates of the embedding are the eigenvectors that correspond to the bottom *dim* eigenvalues (ignoring always the zero eigenvalue).

Locally Linear Embedding (LLE)

This method first converts the weighed adjacency matrix to a distance matrix. Then it computes another weight matrix A by minimizing each reconstruction error $\varepsilon = \|x - \sum_j a_j x_j\|^2$ where the x_j are the neighboors of x. To do so we first compute the following local covariance matrix:

$$C_{jk} = \frac{1}{2}(D_{\cdot j} + D_{i,\cdot} - D_{ij} - D_{\cdot\cdot})$$

where D_{ij} is the squared distance matrix,

$$D_{\cdot,j} = \frac{1}{n}\sum_i D_{ij},$$

$$D_{i,\cdot} = \frac{1}{n}\sum_j D_{ij},$$

$$D_{\cdot,\cdot} = \frac{1}{n^2}\sum_i\sum_j D_{ij},$$

and where D is the n by n squared distance matrix ($D = d^2$). One can observe that the local covariance matrix is simply a Multi-Dimensional Scaling (MDS) of the squared distance matrix D.

We calculate the weight matrix A by solving the system of linear equations $\sum_k C_{jk} a_k = 1$ and rescale the weights so that any column of A sums up to 1. Finally the coordinates of the embedding are the eigenvectors of the matrix $M = (I - A)^T(I - A)$. More precisely the coordinates of the embedding are the eigenvectors that correspond to the dim+1 bottom eigenvalues of M (we always ignore the first eigenvector since the first eigenvalue is always zero) leaving us with a dim dimensional embedding.

Isomap

This algorithm computes the embedding using the distance matrix d. Firstly we compute the dijkstra distance of all possible nodes of the graph. We store these values squared in a matrix D where D_{ij} is the squared dijkstra distance from node i to node j. Continuing we performe a MDS on the squared dijkstra matrix D. This way we compute Matrix B as

$$B_{jk} = \frac{1}{2}\left(D_{\cdot j} + D_{i\cdot} - D_{ij} - D_{\cdot\cdot}\right).$$

Finally the coordinates of the embedding are the eigenvectors that corespond to the top dim eigenvalues. Finaly one can scale the coordinates as $X = \Gamma\Lambda^{1/2}$ where Λ is the diagonal matrix of dim top eigenvalues of B and Γ is the matrix of the corresponding eigenvectors.

The signal on the graph is related to the coordinate information of the original graph and therefore allows us to evaluate the resulting embedding by looking at the smoothness of this signal on the graph.

References: [13], [18], [1]

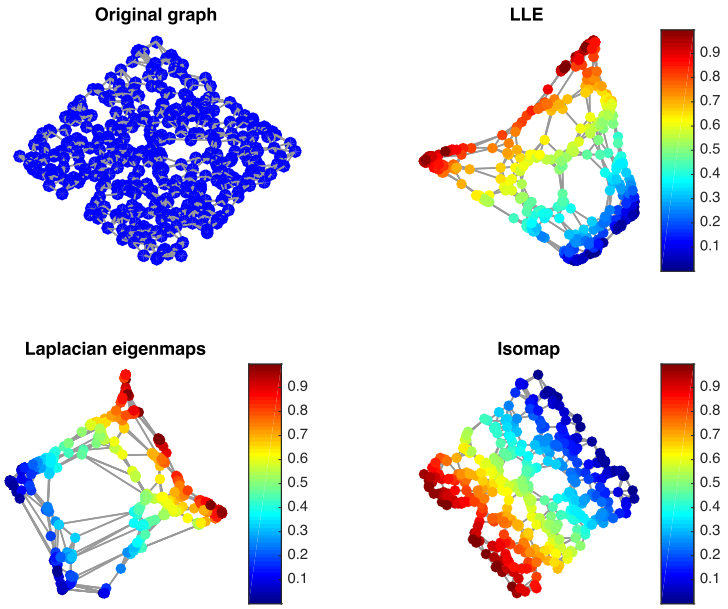


Figure 2.10: Resulting embeddings

Output

```
GSPBox version 0.5.1. Copyright 2013-2015 LTS2-EPFL,
by Nathanael Perraudin, Johan Paratte, David Shuman and Vassilis Kalofolias
```

2.2 Convex optimization on graphs

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-known TV norm defined on the graph.

For this example, you need the unlocbox. You can download it here: `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 represents the known measurements, M is an operator representing the mask and ε is the radius of the l2 ball.

We set

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

$$prox_{f1,\gamma}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + \gamma\|\nabla z\|_1$$

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

$$prox_{f2,\gamma}(z) = arg\min_x \frac{1}{2}\|x-z\|_2^2 + i_S(x),$$

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

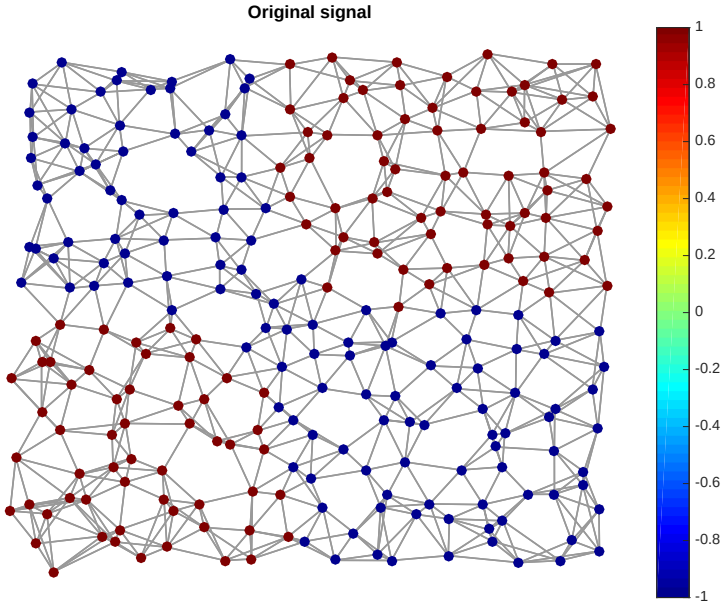


Figure 2.11: Original signal on graph

This figure shows the original signal on graph.

$$arg\min_{z} \|x-z\|_2^2 \qquad such \quad that \qquad \|Mz-b\|_2 \leq \varepsilon$$

It is simply a projection on the B2-ball.

Results

Comparison with Tikhonov regularization

We can also use the Tikhonov 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:

Output

```
UnLocBoX version 1.7.3. Copyright 2012-2015 LTS2-EPFL, by Nathanael Perraudin
DOUGLAS_RACHFORD  f(x^*) = 4.671607e+01, rel_eval = 2.926214e-04, it = 50, MAX_IT
DOUGLAS_RACHFORD  f(x^*) = 2.074368e+01, rel_eval = 2.820457e-06, it = 50, MAX_IT
```

2.2.2 GSP_DEMO_WAVELET_DN - Demonstratration of the use of wavelet for denoising

In this small example, we show how to perform wavelet denoising using the GSPBox and particularly the function gsp_wavelet_dn .

The function gsp_wavelet_dn removes the low frequency part of the signal and solves a l1 minimization problem to remove the noise.

Output

```
A function has both the prox and a grad fields. The gradient is used
The time step is set manually to : 0.5
Algorithm selected: FORWARD_BACKWARD
```

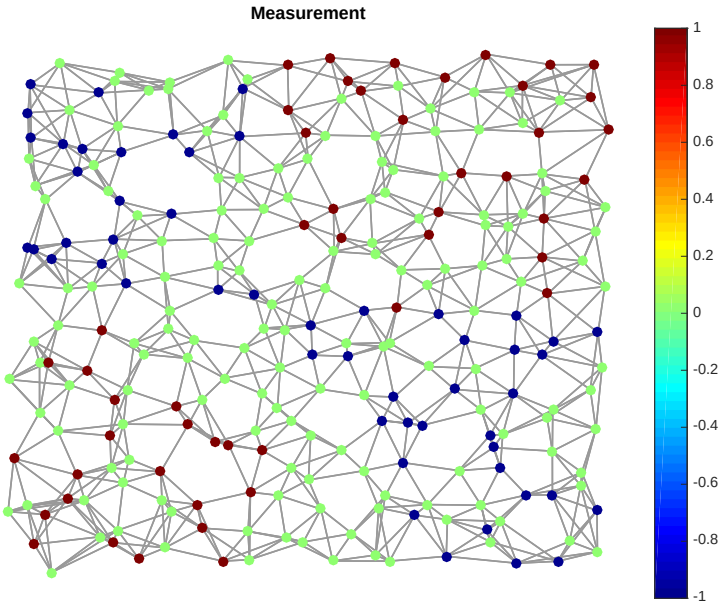


Figure 2.12: Depleted signal on graph

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.

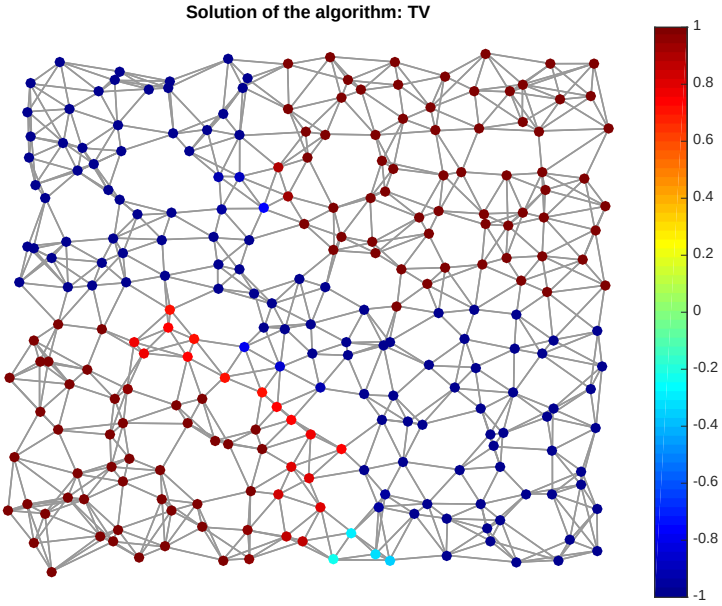


Figure 2.13: Reconstructed signal on graph usign TV

This figure shows the reconstructed signal thanks to the algorithm.

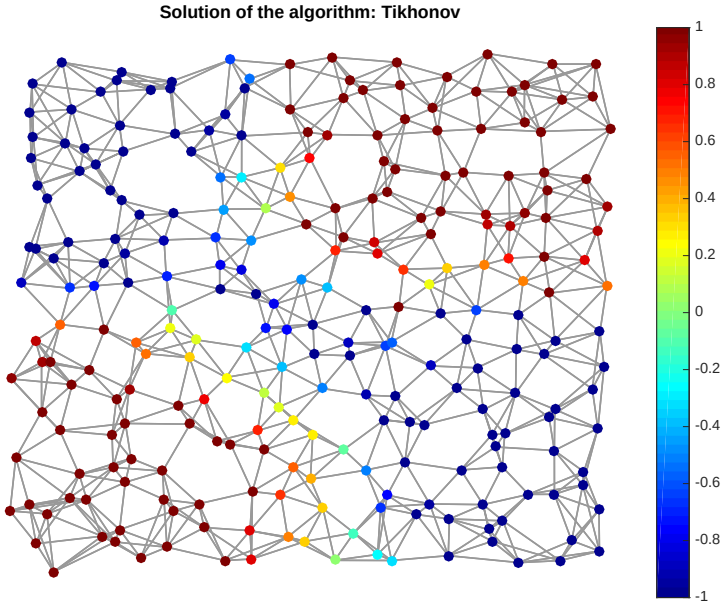


Figure 2.14: Reconstructed signal on graph using Tikhonov

This figure shows the reconstructed signal thanks to the algorithm.

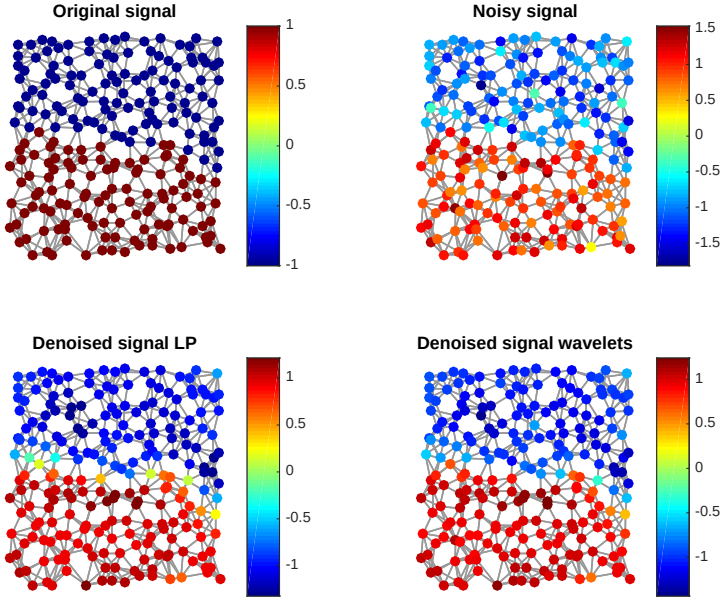


Figure 2.15: Result of filtering

We observe that the wavelet denoising allows rapid change on the graph signal. This is not possible with a simple low pass filtering.

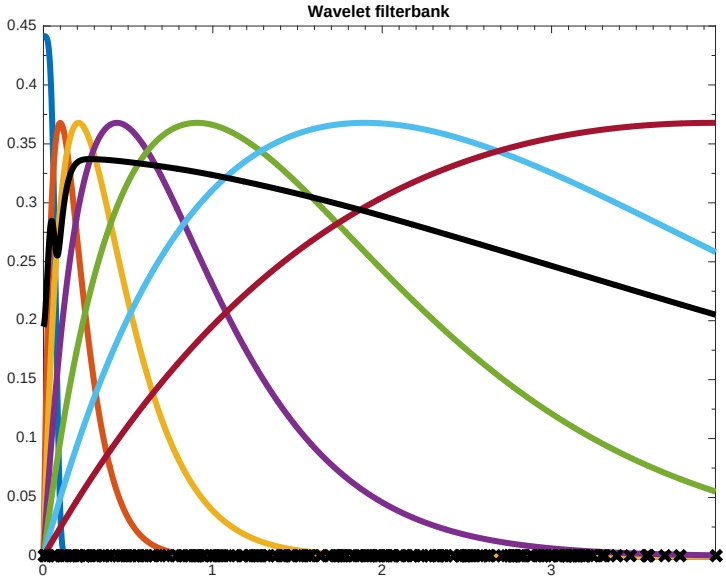


Figure 2.16: Choosen wavelet filterbank

Here we use a mexican hat frame for the wavelet construction.

```
Iter 001:     prox_L1: ||A x-y||_1 = 0.000000e+00, REL_OBJ, iter = 1
  f(x^*) = 1.635623e+01, rel_eval = 1.918014e+02
Iter 002:     prox_L1: ||A x-y||_1 = 0.000000e+00, REL_OBJ, iter = 1
  f(x^*) = 1.635623e+01, rel_eval = 0.000000e+00

 FORWARD_BACKWARD:
  f(x^*) = 1.635623e+01, rel_eval = 0.000000e+00
 2 iterations
 Stopping criterion: REL_NORM_OBJ

A function has both the prox and a grad fields. The gradient is used
The time step is set manually to : 0.5
Algorithm selected: FORWARD_BACKWARD
Iter 001:     prox_L1: ||A x-y||_1 = 3.950718e+00, REL_OBJ, iter = 1
  f(x^*) = 1.842782e+01, rel_eval = 1.307758e+00
Iter 002:     prox_L1: ||A x-y||_1 = 1.905458e+00, REL_OBJ, iter = 1
  f(x^*) = 1.490933e+01, rel_eval = 2.359922e-01
Iter 003:     prox_L1: ||A x-y||_1 = 1.791944e+00, REL_OBJ, iter = 1
  f(x^*) = 1.468075e+01, rel_eval = 1.556998e-02
Iter 004:     prox_L1: ||A x-y||_1 = 1.829279e+00, REL_OBJ, iter = 1
  f(x^*) = 1.457597e+01, rel_eval = 7.188845e-03
Iter 005:     prox_L1: ||A x-y||_1 = 1.894864e+00, REL_OBJ, iter = 1
  f(x^*) = 1.451468e+01, rel_eval = 4.222307e-03
Iter 006:     prox_L1: ||A x-y||_1 = 1.952902e+00, REL_OBJ, iter = 1
  f(x^*) = 1.446909e+01, rel_eval = 3.150956e-03
Iter 007:     prox_L1: ||A x-y||_1 = 1.999188e+00, REL_OBJ, iter = 1
  f(x^*) = 1.443249e+01, rel_eval = 2.536113e-03
Iter 008:     prox_L1: ||A x-y||_1 = 2.032134e+00, REL_OBJ, iter = 1
  f(x^*) = 1.440235e+01, rel_eval = 2.092857e-03
Iter 009:     prox_L1: ||A x-y||_1 = 2.048022e+00, REL_OBJ, iter = 1
  f(x^*) = 1.437391e+01, rel_eval = 1.978575e-03
Iter 010:     prox_L1: ||A x-y||_1 = 2.053312e+00, REL_OBJ, iter = 1
  f(x^*) = 1.434766e+01, rel_eval = 1.829546e-03
```

```
Iter 011:     prox_L1: ||A x-y||_1 = 2.064799e+00, REL_OBJ, iter = 1
  f(x^*) = 1.432871e+01, rel_eval = 1.322199e-03
Iter 012:     prox_L1: ||A x-y||_1 = 2.076503e+00, REL_OBJ, iter = 1
  f(x^*) = 1.431229e+01, rel_eval = 1.147399e-03
Iter 013:     prox_L1: ||A x-y||_1 = 2.090651e+00, REL_OBJ, iter = 1
  f(x^*) = 1.429848e+01, rel_eval = 9.659487e-04

 FORWARD_BACKWARD:
  f(x^*) = 1.429848e+01, rel_eval = 9.659487e-04
 13 iterations
 Stopping criterion: REL_NORM_OBJ
```

2.2.3 GSP_DEMO_LEARN_GRAPH - Demonstration of learning a graph from data

In this demo, we show how the graph learning can be used to learn a graph from smoothly changing signals. The theory behind the algorithm can be found in

[1] V. Kalofolias, How to learn a graph from smooth signals, AISTATS 2016.

Suppose that we have some 2 dimensional smooth functions:

```
f1 = @(x,y) 20 * (-sin((2-x-y).^2)/2 + cos(y*3));
f2 = @(x,y) 30 * cos((x+y).^2);
f3 = @(x,y) 30 * ((x-.5).^2 + (y-.5).^3 + x - y);
f4 = @(x,y) 50 * sin(3*((x-.5).^2+(y-.5).^2));
```

and we have uniform samples as features, displayed below:

```
figure;
subplot(2,2,1); scatter(xc, yc, 700, X(:,1), '.');
title('1st smooth signal'); axis off; colorbar;
subplot(2,2,2); scatter(xc, yc, 700, X(:,2), '.');
title('2nd smooth signal'); axis off; colorbar;
subplot(2,2,3); scatter(xc, yc, 700, X(:,3), '.');
title('3rd smooth signal'); axis off; colorbar;
subplot(2,2,4); scatter(xc, yc, 700, X(:,4), '.');
title('4th smooth signal'); axis off; colorbar;
```

We can compute the pairwise distances of the features and learn a graph using them:

```
Z1 = gsp_distanz(X(:, 1)').^2;
W1 = gsp_learn_graph_log_degrees(Z1, 1.5, 1, params);
```

The second parameter penalizes the formation of un-connected nodes, and the third penalizes the formation of too strong weights. We then clean any tiny edges (due to numerical error), to obtain a sparse weighted adjacency matrix. We feed this to gsp_graph to create a graph with the given coordinates and weights:

```
W1(W1<1e-5) = 0;
G1 = gsp_graph(W1, [xc, yc]);
```

We can also update the weights of an already existing graph using gsp_update_weights. If we learn the graphs of all four above functions, we get quite different results:

```
figure;
subplot(2,2,1); gsp_plot_edges(G1, params_plot);
title('graph learned from 1st smooth signal');
subplot(2,2,2); gsp_plot_edges(G2, params_plot);
title('graph learned from 2nd smooth signal');
```

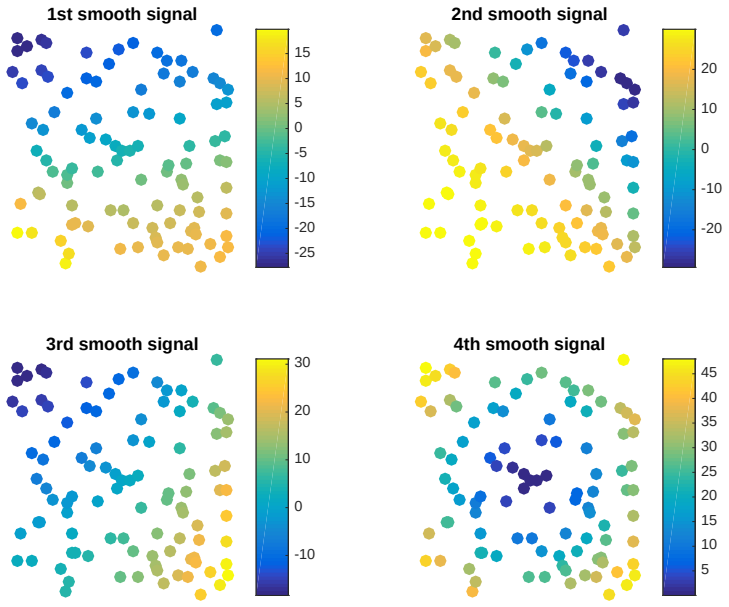


Figure 2.17: Different signals

```
subplot(2,2,3); gsp_plot_edges(G3, params_plot);
title('graph learned from 3rd smooth signal');
subplot(2,2,4); gsp_plot_edges(G4, params_plot);
title('graph learned from 4th smooth signal');
```

Note that the edges follow the level curves of the above functions.
If we use all four above smooth functions as features to learn the graph:

```
Z = gsp_distanz(X').^2;
W = gsp_learn_graph_log_degrees(Z/500, 2, 1, params);
```

we get a result that has more local edges:

```
params_plot.show_edges = 1;
G.plotting.vertex_size = 5;
figure; gsp_plot_graph(G, params_plot);
title('Graph with edges learned from above 4 signals');
```

This is close to the graph that we would learn using the acutal coordinates as features. So why does it work so well? We can see that the pattern of the pairwise distances using these features is similar to the one of the pairwise geometric distances between nodes:

```
figure;
subplot(1, 2, 1);
imagesc(gsp_distanz(X'));
title('Geometric pairwise distances between nodes');
subplot(1, 2, 2);
imagesc(gsp_distanz([xc, yc]'));
title('Pairwise distances computed from features');
```

The functions available for learning a graph are **??** and **??**.
References: [7]

Output

```
# iters: 2077. Rel primal: 2.9934e-06 Rel dual: 9.9886e-06  OBJ -4.902e+01
Time needed is 0.607323 seconds
```

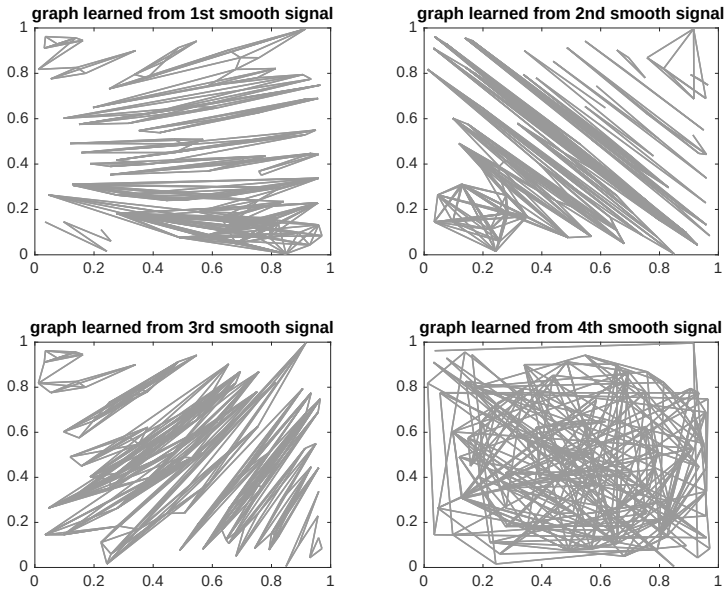


Figure 2.18: Different graphs learned

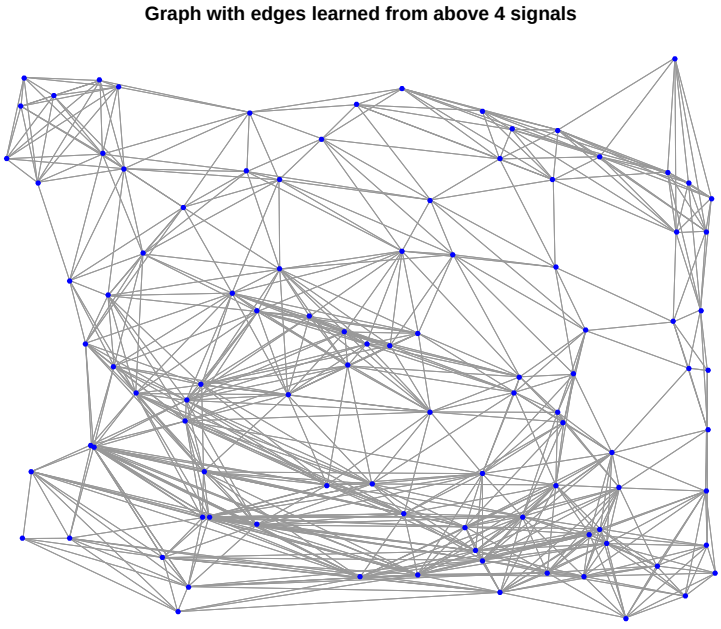


Figure 2.19: Graph with edges learned from above 4 signals

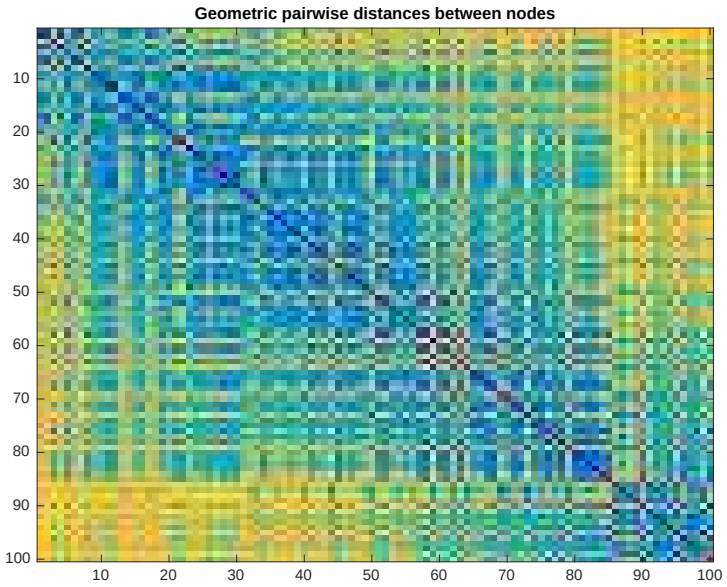


Figure 2.20: Geometric pairwise distances between nodes

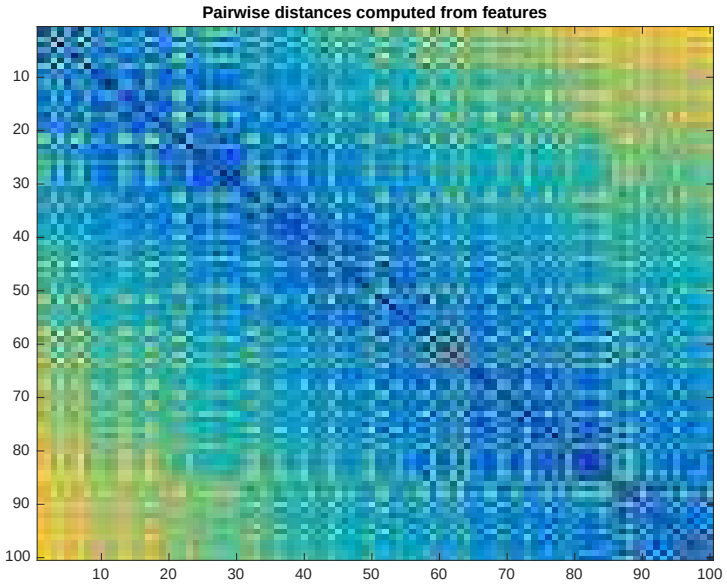


Figure 2.21: Pairwise distances computed from features

```
# iters: 5992. Rel primal: 5.8960e-06 Rel dual: 9.9954e-06  OBJ -3.598e+02
Time needed is 1.665346 seconds
# iters: 1110. Rel primal: 4.9372e-06 Rel dual: 9.9719e-06  OBJ -1.141e+03
Time needed is 0.363148 seconds
# iters: 1063. Rel primal: 4.5236e-06 Rel dual: 9.9927e-06  OBJ -1.149e+02
Time needed is 0.339802 seconds
# iters:  406. Rel primal: 9.7565e-06 Rel dual: 7.1212e-06  OBJ -2.846e+02
Time needed is 0.122376 seconds
Graph of signal 1: 288  edges
Graph of signal 2: 325  edges
Graph of signal 3: 311  edges
Graph of signal 4: 245  edges
Graph of all signals: 680  edges
Plotting edges with different sizes. It may take some time.
```

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 a simple signal.

The function gsp_graph_multiresolution computes the graph pyramid for you:

```
param.sparsify = 1;
Gs = gsp_graph_multiresolution(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).
- *sparsify_epsilon*: Parameter epsilon used in the 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's display the results:

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

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:

```
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)]);
```

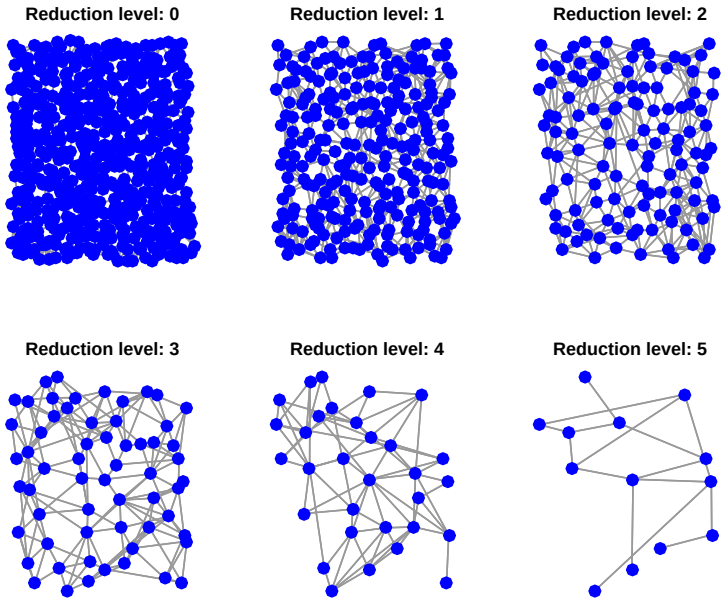


Figure 2.22: Reduction of the graph

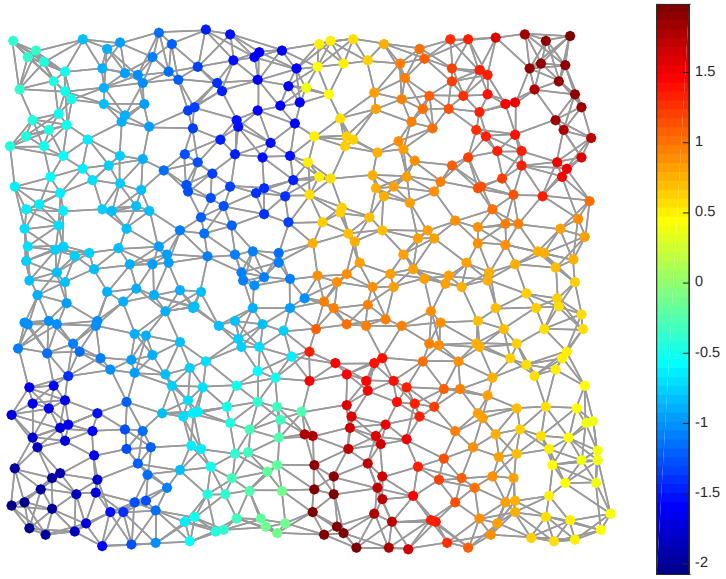


Figure 2.23: Signal to be analysed

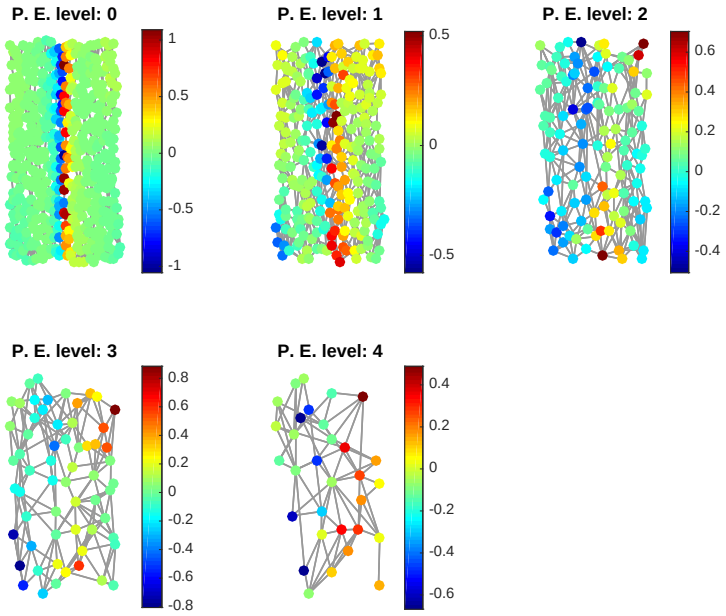


Figure 2.24: Prediction errors

```
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

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

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

Enjoy!

Output

```
The relative reconstruction error is : 1.98862e-15
```

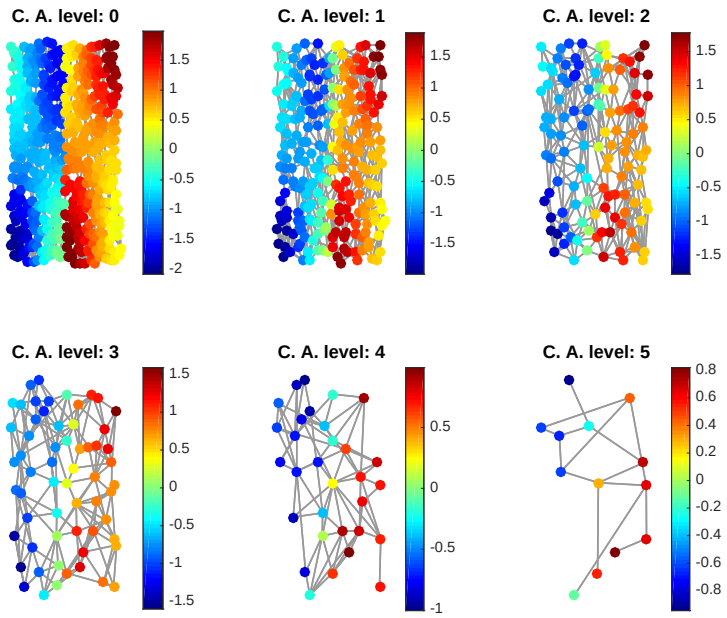


Figure 2.25: Coarse approximations

Chapter 3

GSPBOX - Graphs

3.1 Specific graphs

3.1.1 GSP_SWISS_ROLL - Initialize a swiss roll graph

Usage

```
G = gsp_swiss_roll(N,rand_state,param);
```

Input parameters

N	Number of vertices.
s	sigma (default: sqrt(2/N))
thresh	threshold (default: 1e-6)
rand_state	rand seed (default: 0)

Output parameters

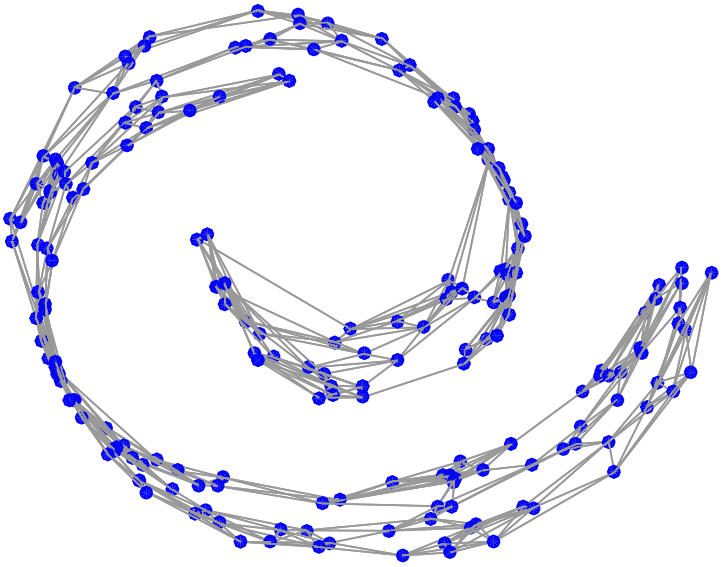
G	Graph structure.

Description

'gsp_create_swiss_roll(N,s,thresh,rand_state)' initializes a graph structure containing the swiss roll graph

Example:

```
G = gsp_swiss_roll(200);
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.

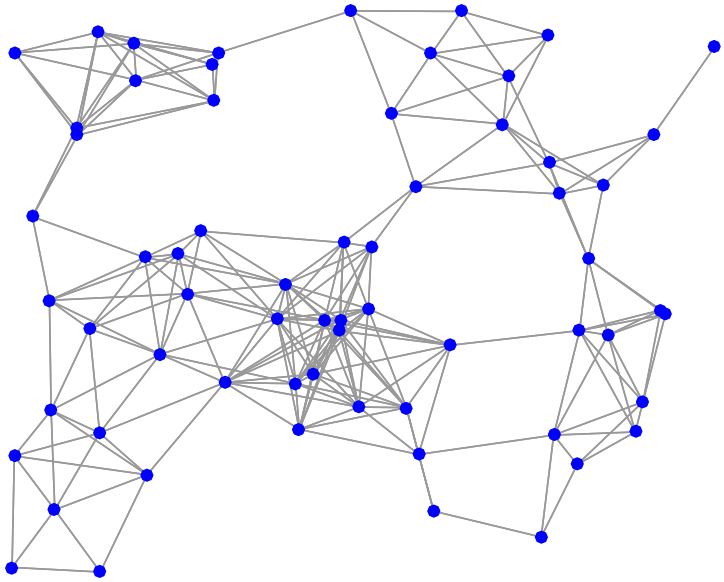
Description

'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 necessarly 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)

Output parameters

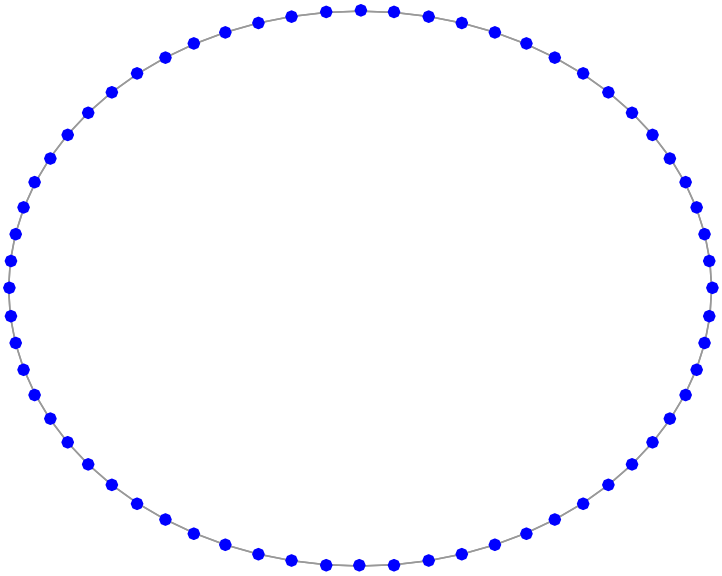
G	Graph structure.

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:

```
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);
G = gsp_path();
```

Input parameters

N	Number of vertices (default 32).

Output parameters

G	Graph structure.

Des

'gs

References: [17]

3.1.5 GSP_AIRFOIL - Initialize the airfoil graph

Usage

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

Input parameters

non none

Output parameters

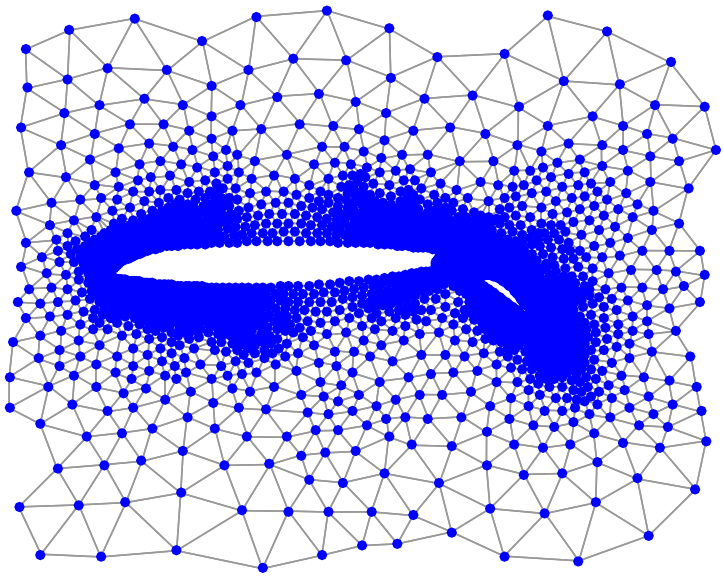
G	Graph structure.

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:

```
G = gsp_airfoil();
paramplot.show_edges = 1;
gs
```



3.1.6 GSP_COMET - Initialize a comet graph

Usage

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

Input parameters

N	Number of vertices. (default 32)
k	Degree of center vertex. (default 12)

Output parameters

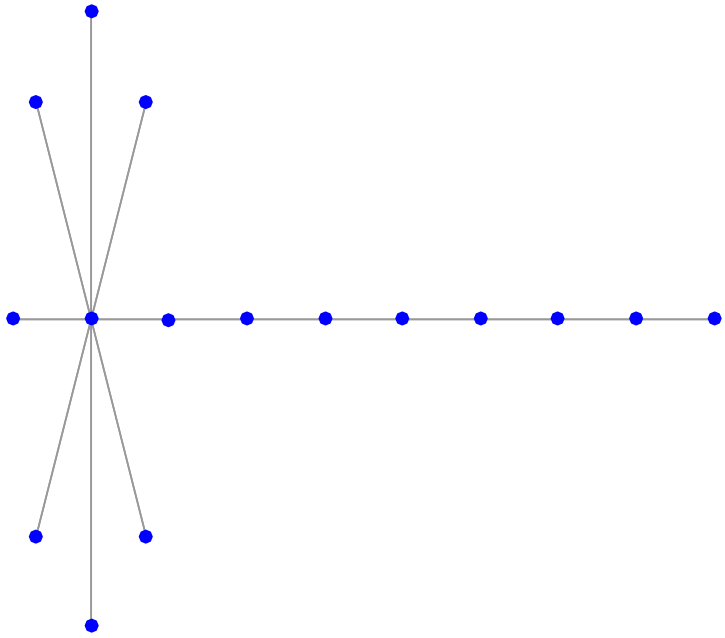
G	Graph structure.

Description

'gsp_comet(N,k)' initializes the commet graph. The commet graph is a simple path graph with a star od 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 create 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 0.
- *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 = gsp_erdos_renyi(100,0.05)
```

3.1.8 GSP_MINNESOTA - Initialize the Minnesota road network

Usage

```
G = gsp_minnesota();
G = gsp_minnesota(connect);
```

Input parameters

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

Output parameters

G	Graph structure.

Description

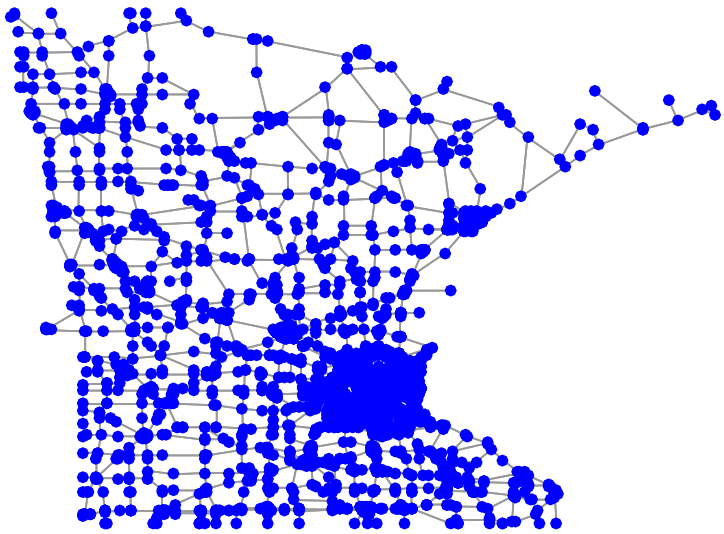
'gsp_minnesota()' initializes a graph structure containing the weighted adjacency matrix (G.W), the number of vertices (G.N), the plotting coordinates (G.coords), and the plotting coordinate limits (G.limits) of the Minnesota road network from the MatlabBGL library.

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 orinial disconnected graph, use:

```
G = gsp_minnesota(connect);
```

Example:



References: [5]

3.1.9 GSP_LOW_STRETCH_TREE - Initialize a low stretch tree

Usage

```
G = gsp_low_stretch_tree(k);
G = gsp_low_stretch_tree();
```

Input parameters

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

Output parameters

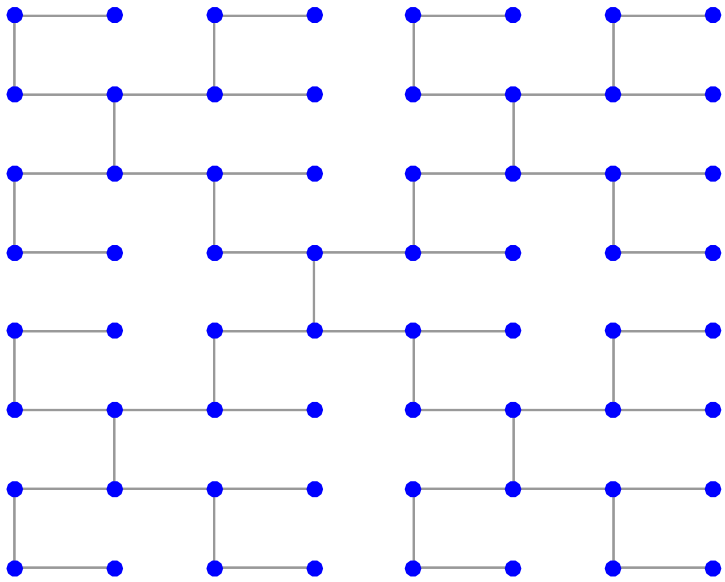
G Graph structure.

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:

```
G = gsp_low_stretch_tree(3);
```



3.1.10 GSP_SENSOR - Create a random sensor graph

Usage

```
G = gsp_sensor(N);
G = gsp_sensor( );
G = gsp_sensor(N, param);
```

Input parameters

- N Number of nodes (default 128)

- 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.

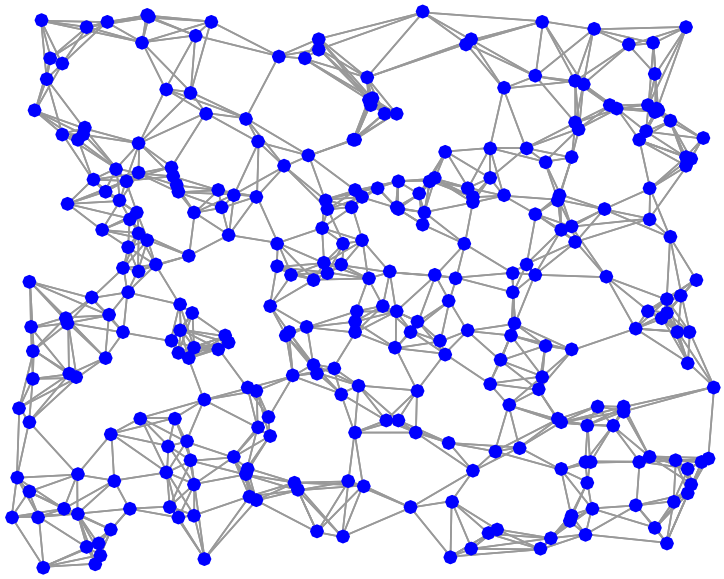
param is an optional structure with the following field

- *param.verbose*: display parameter - 0 no log - 1 display the errors (default 1)

- *param.N_try*: Number of attempts to create the graph (default 10)
- *param.distribute*: To distribute the points more evenly (default 0)
- *param.connected*: To force the graph to be connected (default 1)
- *param.nnparam*: optional parameter for the gsp_nn_graph

Example:

```
G = gsp_sensor(300);
```



3.1.11 GSP_RANDOM_REGULAR - Create a random regualar graph

Usage

```
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)

Output parameters

G	Graph structure.

Description

'gsp_random_regular(N,k)' initializes create 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 nodes is connected to 'k' other nodes.

Example:

```
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
    hypergraph: 0
      lap_type: 'combinatorial'
             L: [100x100 double]
             d: [100x1 double]
            Ne: 150
        coords: []
      plotting: [1x1 struct]
```

3.1.12 GSP_RANDOM_RING - Initialize a random ring graph

Usage

```
G = gsp_random_ring(N);
G = gsp_random_ring();
```

Input parameters

N	Number of vertices. (default 64)

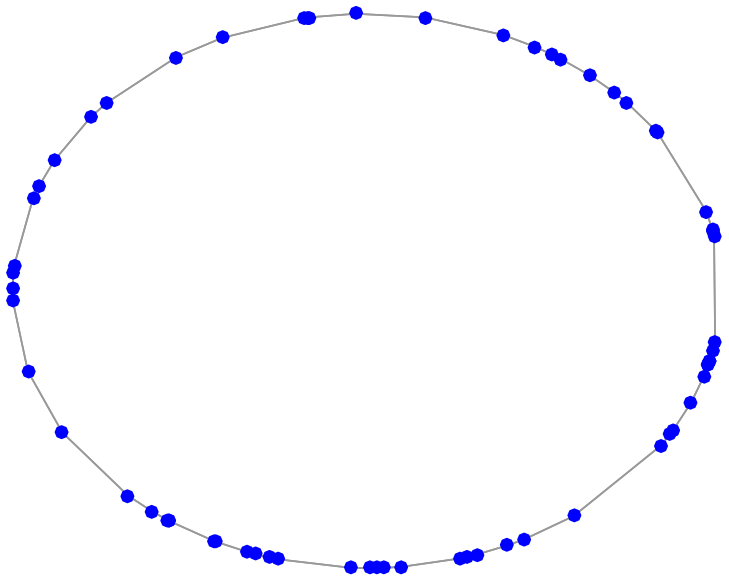
Output parameters

G	Graph structure.

Description

'gsp_ring(N)' weights are all equal to 1.
Example:

```
G = gsp_random_ring(64);
```



3.1.13 GSP_FULL_CONNECTED - Create a fully connected graph

Usage

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

Input parameters

N	Number of vertices (default 10)

Output parameters

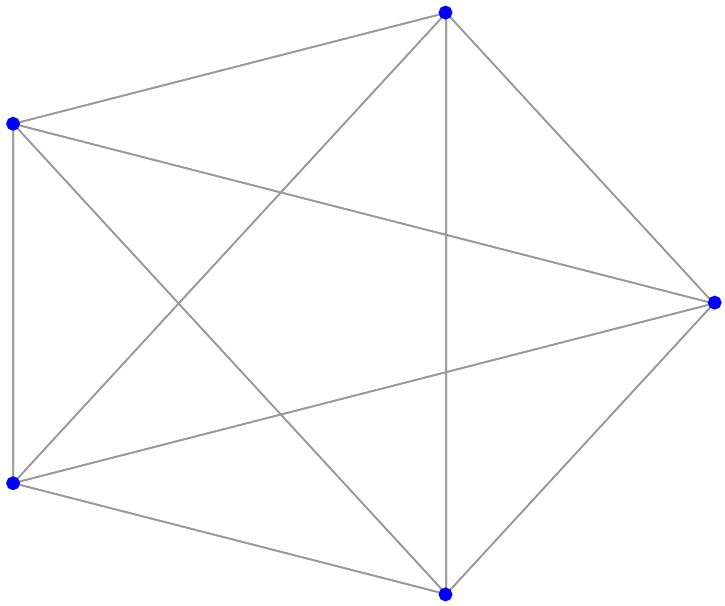
G	Graph structure.

Description

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

Example:

```
G = gsp_full_connected(5);
param.show_edges = 1;
```



3.1.14 GSP_NN_GRAPH - Create a nearest neighbors graph from a point cloud

Usage

```
:  G = gsp_nn_graph( Xin );
G = gsp_nn_graph( Xin, param );
```

Input parameters

Xin	Input points
param	Structure of optional parameters

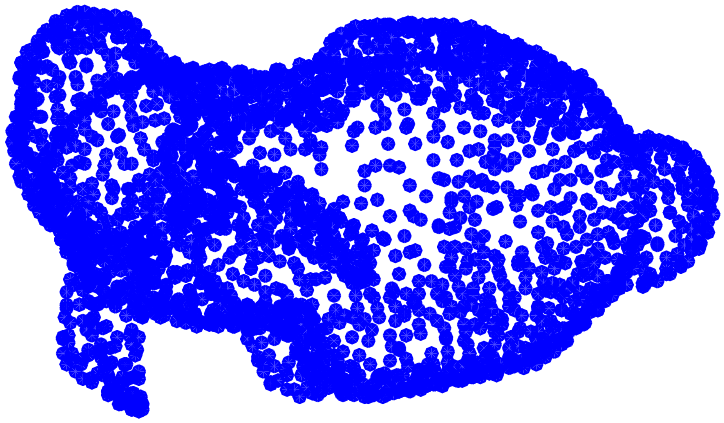
Output parameters

G	Resulting graph

Description

'gsp_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:



Additional parameters

- *param.type* : ['knn', 'radius'] the type of graph (default 'knn')
- *param.use_flann* : [0, 1] use the FLANN library
- *param.use_full* : [0, 1] - Compute the full distance matrix and then sparsify it (default 0)
- *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
- *param.use_l1* : [0, 1] use the l1 distance
- *param.symmetrize_type*: ['average','full'] symmetrization type (default 'full')

3.1.15 GSP_RMSE_MV_GRAPH - Root meean square error missing value graph

Usage

```
:  G = gsp_rmse_mv_graph( Xin );
G = gsp_rmse_mv_graph( Xin, param );
```

Input parameters

Xin	Input points (with missing value (nan))
param	Structure of optional parameters

Output parameters

G	Resulting graph

Description

'gsp_rmse_mv_graph(X,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. This function ignore all nan value. But it is much slower than gsp_nn_graph.

Additional parameters

- *param.type* : ['knn', 'radius'] the type of graph (default 'knn')
- *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
- *param.symmetrize_type*: ['average','full'] symmetrization type (default 'full')
- *param.center* : [0, 1] center the data
- *param.rescale* : [0, 1] rescale the data (in a 1-ball)

3.1.16 GSP_SPHERE - Create a spherical-shaped graph

Usage

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

Input parameters

param	Structure of optional parameters

Output parameters

G	Resulting graph

Description

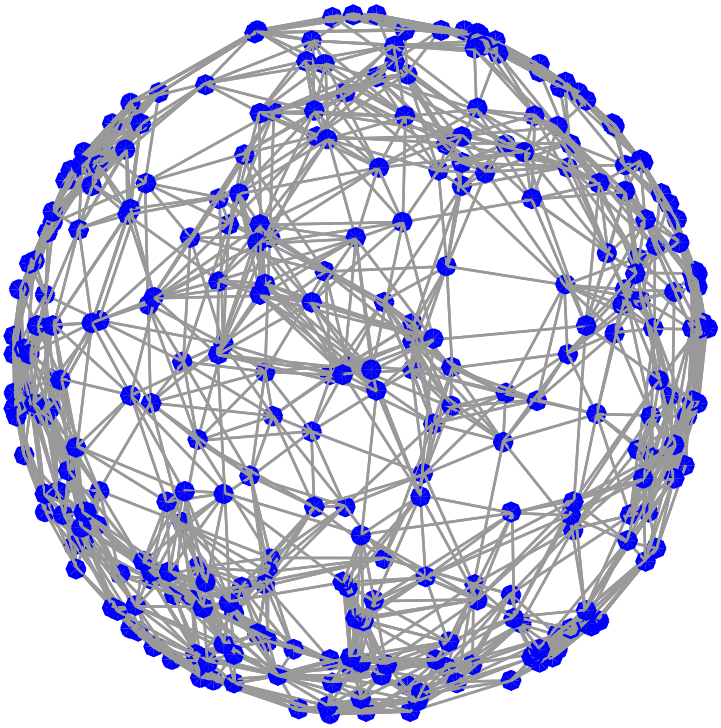
'gsp_sphere(param)' creates a graph from points sampled on a hyper-sphere. The dimension of the sphere can be passed as a parameter. It can be sampled in a uniform voxel grid or randomly.

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
```



3.1.17 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

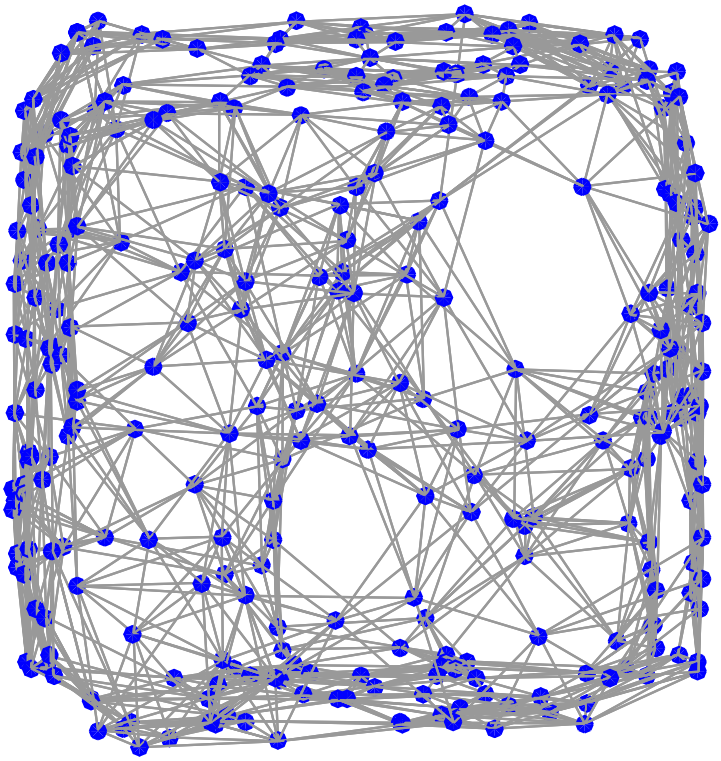
'gsp_cube(param)' creates a graph from points sampled on a hyper-cube. The dimension of the cube can be passed as a parameter. It can be sampled in a uniform voxel grid or randomly.

Additional parameters

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

Example:

```
G = gsp_cube();
gsp_plot_graph(G);
axis square
```



3.1.18 GSP_2dgrid - Initialize a 2 dimentional grid graph

Usage

```
G=gsp_path(N);
```

Input parameters

N	Number of vertices along the first dimention (default 16)
M	Number of vertices along the second dimention (default N)

Output parameters

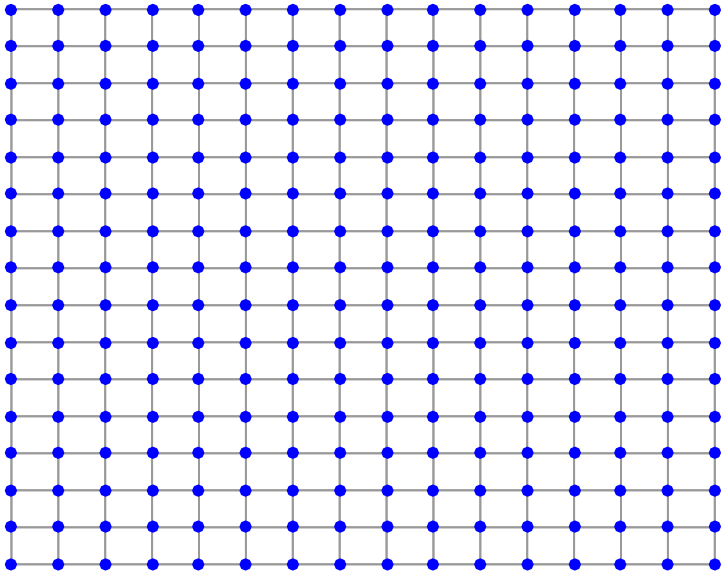
G	Graph structure.

Description

The 2d grid graph correspond the graph used for the DCT2????

Example:

```
G = gsp_2dgrid(16);
param.show_edges = 1;
gsp_plot_graph(G,param);
```



References: [17]

3.1.19 GSP_TORUS - Initialize a 2 dimentional grid graph

Usage

```
G=gsp_path(N);
```

Input parameters

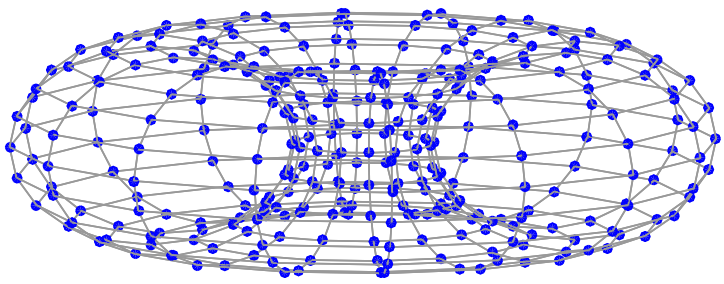
N	Number of vertices along the first dimention (default 16)
M	Number of vertices along the second dimention (default N)

Output parameters

G	Graph structure.

Description

The 2
E



References: [17]

3.1.20 GSP_LOGO - Initialize a graph with the GSP logo

Usage

```
G = gsp_logo();
```

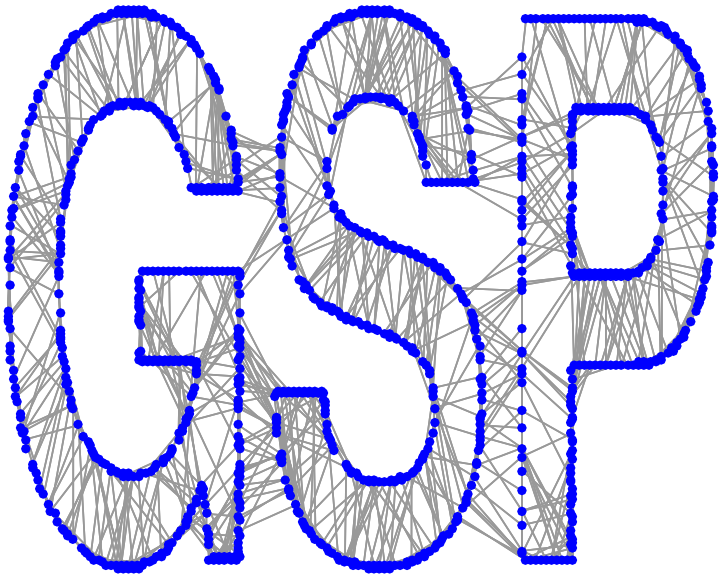
Output parameters

G	Graph structure.

Description

'gsp_logo()' initializes a graph structure containing the GSP logo
Example:

```
G = gsp_logo();
```



3.1.21 GSP_COMMUNITY - Create a community graph

Usage

```
G = gsp_community(N);
G = gsp_community();
G = gsp_community(N, param );
```

Input parameters

N	Number of nodes (default 256)
param	Structure of optional parameters

Output parameters

G	Graph

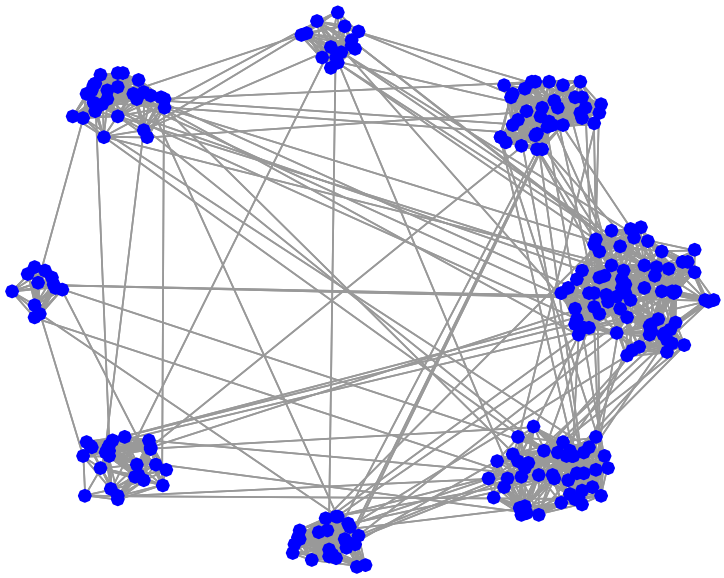
Description

This function create a 2 dimentional random sensor graph. All the coordonates are between 0 and 1.
param is an optional structure with the following fields

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

```
G = gsp_community();
paramplot.show_edges = 1;
```



3.1.22 GSP_BUNNY - Create a graph of the stanford bunny

Usage

```
:  G = gsp_bunny();
```

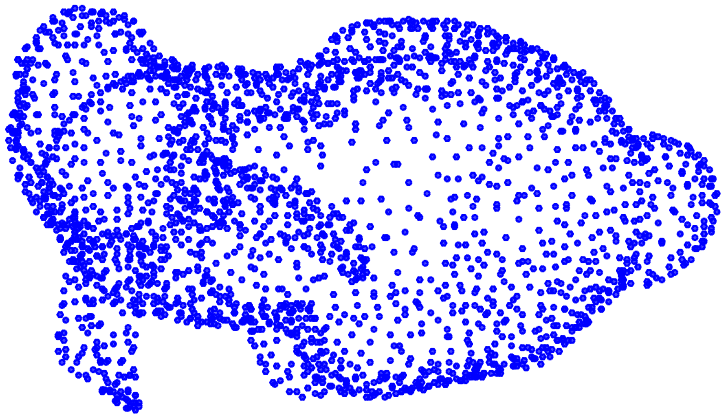
Output parameters

G	Resulting graph

Description

'gsp_bunny()' creates a graph from the pointcloud of the Stanford Bunny model.
Example:

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



References: [19]

3.1.23 GSP_SPIRAL - Initialize a spriral graph

Usage

```
G = gsp_spiral();
G = gsp_spiral(N);
G = gsp_spiral(N,k);
G = gsp_spiral(N, k, param);
```

Input parameters

N	Number of vertices. (default 200)
k	Number of turns (3)
param	Structure of optional parameters

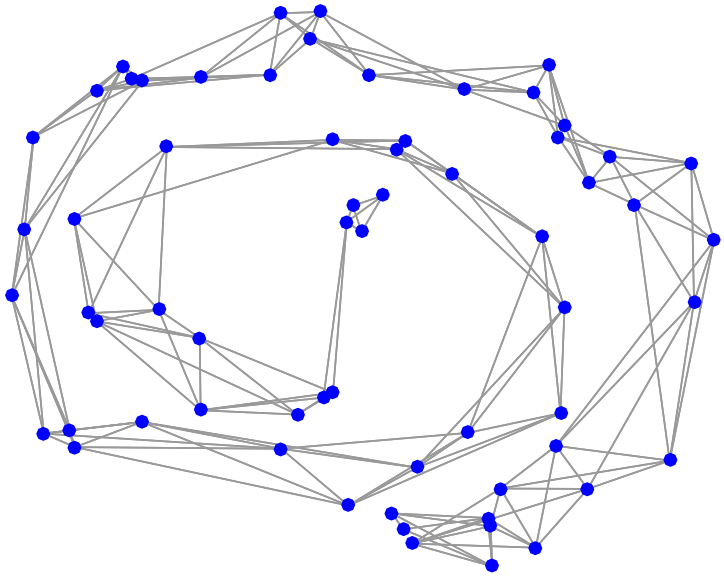
Output parameters

G	Graph structure.

Description

'gsp_spiral(N)' initializes a graph structure for the sprial graph.
The optional parameters are: * *param.noise* : Noise level (Default 5) * *param.start* : Starting angle in degree (Default 90) * *param.k* : Approximate number of connections per nodes (Default 4)
Example:

```
G = gsp_spiral(64);
param.show_edges = 1;
gsp_plot_graph(G,param);
```



3.1.24 GSP_STOCHASTIC_BLOCK_GRAPH - Create a stochastic block graph

Usage

```
G = gsp_stochastic_block_graph( N );
G = gsp_stochastic_block_graph(N , k);
```

Input parameters

N	Number of nodes (default 5)
k	Number of clusters (default 1024)

Output parameters

G	Graph structure.

Description

Use the stochastic block model to create a graph.

3.2 Hypergraphs

3.2.1 GSP_NN_HYPERGRAPH - Create a nearest neighbors hypergraph from a point cloud

Usage

```
:  G = gsp_nn_hypergraph( Xin );
G = gsp_nn_hypergraph( Xin, param );
```

Input parameters

Xin	Input points
param	Structure of optional parameters

Output parameters

G	Resulting graph

Example:

```
P = rand(100,2);
G = gsp_nn_hypergraph(P)
```

This code produces the following output:

```
G =

             N: 100
            Ne: 500
             W: [100x100 double]
             E: {100x1 cell}
    hypergraph: 1
      directed: 0
        coords: [100x2 double]
          type: 'Nearest neighboors hypergraph'
      lap_type: 'normalized'
         sigma: 0.0584
             A: [100x100 double]
            de: [100x1 double]
            dv: [100x1 double]
             L: [100x100 double]
             d: [100x1 double]
      plotting: [1x1 struct]
```

Description

Additional parameters

- *param.use_flann* : [0, 1] use the FLANN library
- *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

3.2.2 GSP_HYPERGRAPH - Initialize a hypergraph from a set of edges and weights

Usage

```
G = gsp_hypergraph(N,E);
G = gsp_hypergraph(N,E, w);
G = gsp_hypergraph(N,E, w, coords);
G = gsp_hypergraph(N, E, w, coords, limits);
```

Input parameters

N	Number of nodes
E	Set of edges (cell array)
w	weights of the edges (default all ones)
coords	Coordonates of the points (optional)
limits	limits for the coordonates (optional)

Output parameters

G Graph structure.

Example:

```
N = 100;
Nf = 2;
k = 4;
x = rand(N,Nf);
paramnn.k = k;
[indx, indy, d] = gsp_nn_distanz(x',x',paramnn);
sigma = mean(d)^2;
wt = exp(-d.^2/sigma);
E = cell(N,1);
w = zeros(N,1);
for ii = 1:N
    edge = indx((1:k)+(ii-1)*k);
    E{ii} = edge;
    w(ii) = sum(wt(edge));
end

G = gsp_hypergraph(N,E,w)
```

This code produces the following output:

```
G =

             N: 100
            Ne: 200
             W: [100x100 double]
             E: {100x1 cell}
          type: 'hypergraph from edges'
      directed: 0
    hypergraph: 1
             A: [100x100 double]
      lap_type: 'normalized'
            de: [100x1 double]
            dv: [100x1 double]
             L: [100x100 double]
             d: [100x1 double]
        coords: []
      plotting: [1x1 struct]
```

3.3 Utils

3.3.1 GSP_GRAPH_DEFAULT_PARAMETERS - load default parameters for graphs

Usage

```
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 compabatible 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 coordonate, you can also set *G.coords*. *G.coords* is a *N* x 2 or a *N* x 3 matrix with each columns beeing the coordonates in each dimention. Finally, we recommend to set the fiel *G.type* with a name that suits your graph.

Example:

```
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:

```
G =

             W: [30x30 double]
             A: [30x30 logical]
             N: 30
          type: 'unknown'
      directed: 0
    hypergraph: 0
      lap_type: 'combinatorial'
             L: [30x30 double]
             d: [30x1 double]
            Ne: 435
        coords: []
      plotting: [1x1 struct]
```

This function can be used to update the weights of your graph. It will recompute the Laplacian operator. Warning this function does not perform any change to the Fourier basis:

```
G.W = Wnew;
G = gsp_graph_default_parameters(G);
```

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 (`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 redudancy 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.

3.3.2 GSP_GRAPH_DEFAULT_PLOTTING_PARAMETERS - Default plotting parameters

Usage

```
G = gsp_graph_default_plotting_parameters( G );
```

Input parameters

G	Graph

Output parameters

G	Graph

Description

This function complete all plotting parameter for a graph.

List of plotting parameters

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')

3.3.3 GSP_GRAPH - Create a graph given weighted adjacency matrix

Usage

```
G = gsp_graph(W);
G = gsp_graph(W, coords);
G = gsp_graph(W, coords, limits);
```

Input parameters

W	(n by n) Weighted adjacency matrix
coords	(n by 2) or (n by 3) Coordinates of the points (optional)
limits	limits for the coordinates (optional)

Output parameters

G	Graph structure.

Description

'gsp_graph(W, coords, limits)' initializes a graph structure with W as weight matrix.
Example:

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

3.3.4 GSP_UPDATE_WEIGHTS - update weight matrix of the graph G

Usage

```
G = gsp_update_weight( G, W );
```

Input parameters

G	Graph
W	W new weight matrix

Output parameters

G	Graph

Description

This function will update the weight of graph and recompute the Laplacian the new Laplacian. It will also recompute the value *G.lmax* if it is present in the graph.
For now this function does not recompute the Fourier basis!
Example:

```
G.W = Wnew;
G = gsp_graph_default_parameters( G );
```

3.3.5 GSP_UPDATE_COORDINATES - Updates the coordinates of a graph structure

Usage

```
G = gsp_update_coordinates(G, coords);
```

Input parameters

G	Graph
coords	New coordinates

Output parameters

G	Output graph

Description

Update the coordinates of a graph structure

3.3.6 GSP_COMPONENTS - cuts non connected graph into several connected ones

Usage

```
Idx             = gsp_components(G);
[Idx, Gz]       = gsp_components(G);
[Idx, Gz, bl] = gsp_components(G);
```

Input parameters

G	Graph

Output parameters

Idx	Index vector
Gz	Cell of sub-graphs
bl	0 or 1 (Connected or Disconnected graph)

Description

This function cuts the non connected graph G into smaller connected sub-graphs that are elements of the cell Gz. In order to access the ith sub-graph one must use Gz{i}. The vector Idx contains elements from 1 to the number of subgraphs. The indices of the elements of a specific subgraph correspond to the indices of the elements of Idx that contain the same value. In order to obtain the coordenets of the first subgraph one can use G.coords(Idx==1, :);

3.3.7 GSP_SUBGRAPH - Create a subgraph from G

Usage

```
[ subG ] = gsp_subgraph( G,c )
```

Input parameters

G	Original graph
c	Node to keep

Output parameters

subG	Subgraph

Description

This function create a subgraph from G taking only the node in c.

Chapter 4

GSPBOX - Filters

4.1 Design - N filter

4.1.1 GSP_DESIGN_MEXICAN_HAT - Design the mexican hat filter bank

Usage

```
g =  gsp_design_mexican_hat(G, Nf, param);
gsp_design_mexican_hat(G ,Nf);
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 return a array of filters designed to be mexican hat wavelet. The mexican hat wavelet is the second oder derivative of a Gaussian. Since we express the filter in the Fourier domain, we find:

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

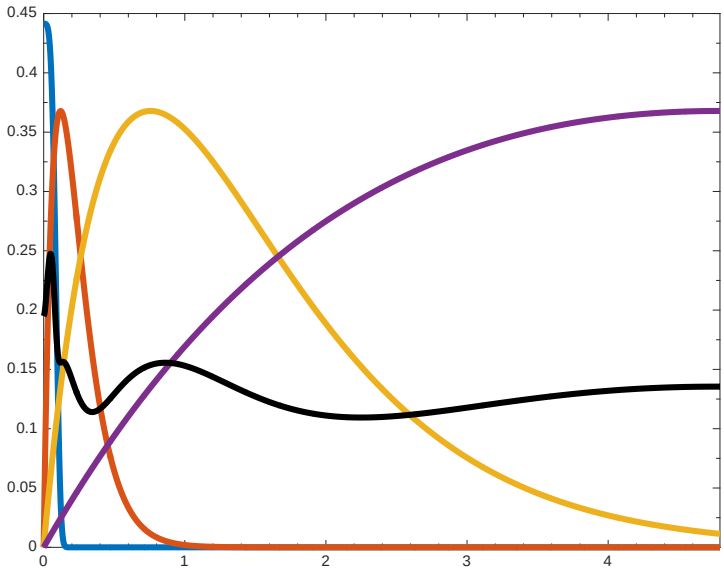
- *param.t*: vector of scale to be used (default: log scale)
- *param.lpfactor*: *lmin*=*lmax/lpfactor* will be used to determine scales, then scaling function kernel will be created to fill the lowpass gap. (default 20)
- *param.verbose*: verbosity level. 0 no log - 1 display warnings. (default 1)
-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

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

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

Example:

```
Nf = 4;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_mexican_hat(G, Nf);
gsp_p
```



This function is inspired by the sgwt_toolbox.

4.1.2 GSP_DESIGN_ABSPLINE - Design the mexican hat filter bank

Usage

```
g =  gsp_design_abspline(G, Nf, param);
gsp_design_abspline(G ,Nf);
gsp_design_abspline(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 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

param is an optional structure containing the following fields

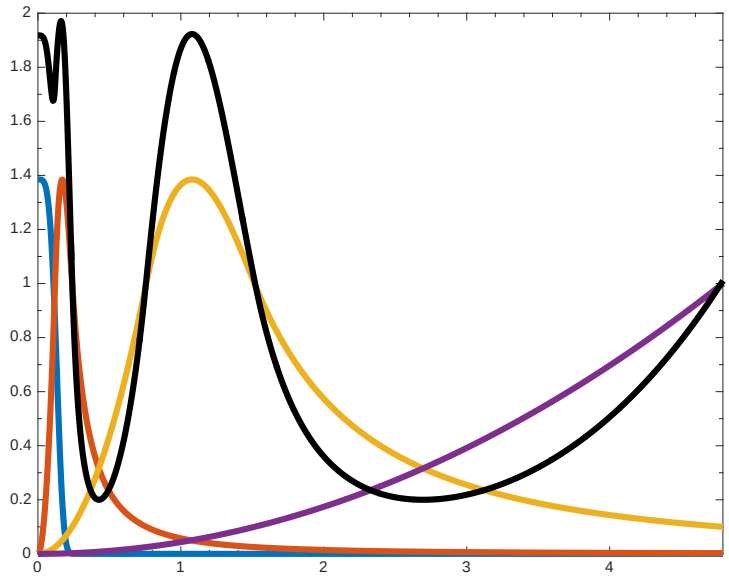
- *param.t*: vector of scale to be used (default: log scale)
- *param.lpfactor*: *lmin*=*lmax/lpfactor* will be used to determine scales, then scaling function kernel will be created to fill the lowpass gap. (default 20)
- *param.verbose*: verbosity level. 0 no log - 1 display warnings. (default 1)

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

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

Example:

```
Nf = 4;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_abspline(G, Nf);
gsp_
```



This function is inspired by the sgwt_toolbox.

4.1.3 GSP_DESIGN_MEYER - Design the meyer filterbank

Usage

```
g =  gsp_design_meyer(G, Nf, param);
gsp_design_meyer(G ,Nf);
gsp_design_meyer(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 return a array of filters designed to be meyer wavelet.
param is an optional structure containing the following fields

- *param.t*: vector of scale to be used (default: log scale)

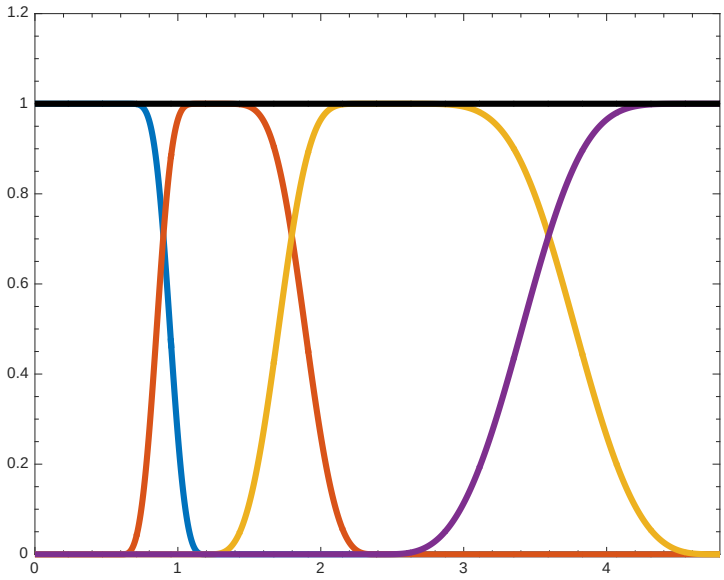
- *param.verbose*: verbosity level. 0 no log - 1 display warnings. (default 1)

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

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

Example:

```
Nf = 4;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_meyer(G, Nf);
gsp_
```



This function is inspired by the sgwt_toolbox.

4.1.4 GSP_DESIGN_SIMPLE_TF - Design a simple tight frame filterbank

Usage

```
g =  gsp_design_simple_tf(G, Nf, param);
gsp_design_simple_tf(G ,Nf);
gsp_design_simple_tf(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 return a array of filters designed to be simple tight frame wavelet filterbank.
param is an optional structure containing the following fields

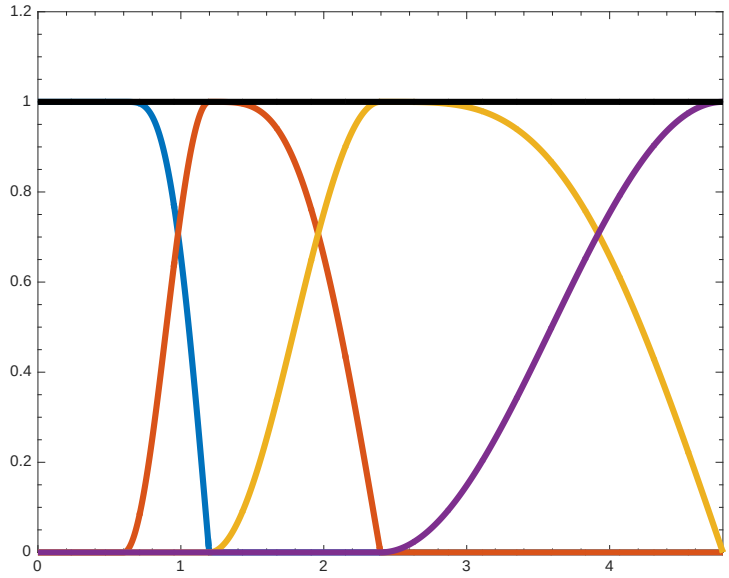
- *param.t*: vector of scale to be used (default: log scale)
- *param.verbose*: verbosity level. 0 no log - 1 display warnings. (default 1)

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

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

Example:

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



This function is inspired by the sgwt_toolbox.

4.1.5 GSP_DESIGN_ITERSINE - Create a itersine filterbanks

Usage

```
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 mu : centers of the filters

This function create a itersine half overlap filterbank of Nf filters Going from 0 to λ_{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 precompute it using:

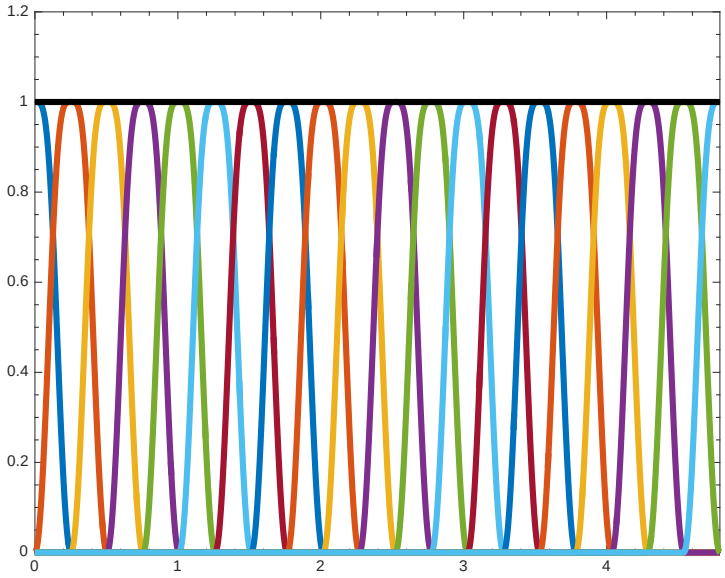
```
G = gsp_estimate_lmax(G);
```

Example:

```
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

```
filters = gsp_design_half_cosine( Nf, UBT );
```

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

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

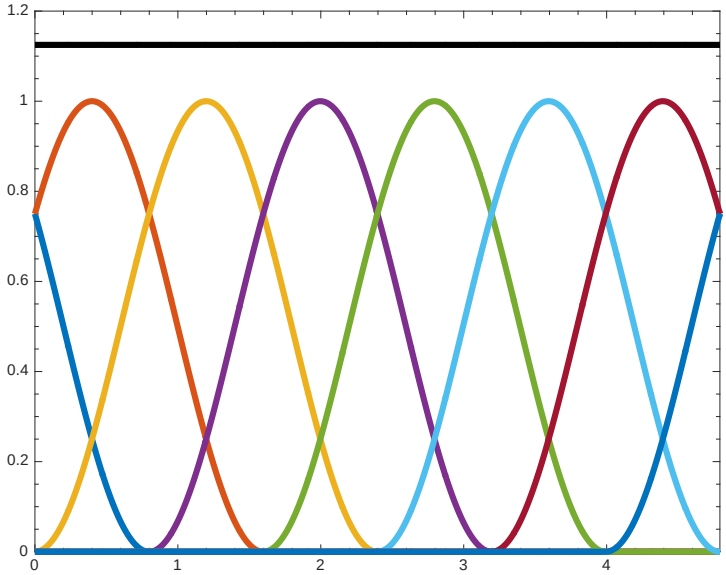
is translated uniformly to create the filterbank.

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

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

Example:

```
Nf = 8;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_half_cosine(G, Nf);
gsp_
```



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

```
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 param : Structure of optional parameters

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:

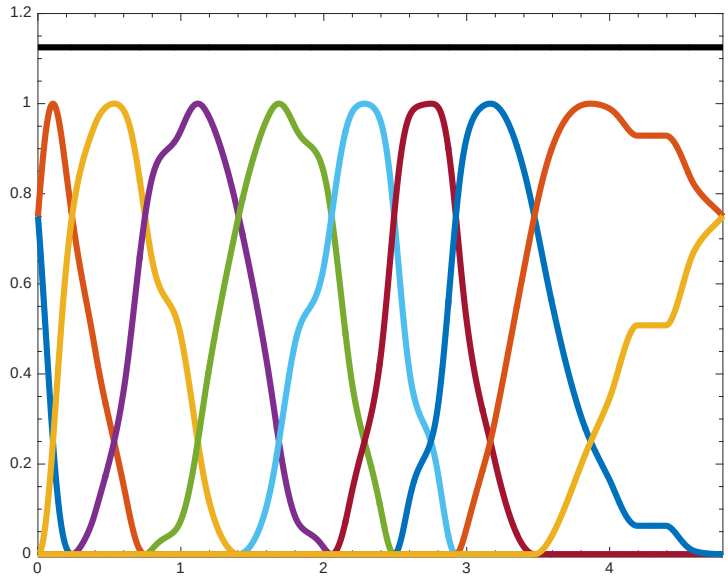
```
G = gsp_estimate_lmax(G);
```

Example:

```
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 =
```



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 alterntive 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:

  ```
  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 availlable 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 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:

  ```
  G = gsp_spectrum_cdf_approx(G);
  ```
- 'custom': The user provide the warping function in the parameter: *param.warp_function*.

References: [15]

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

4.2.1 GSP_DESIGN_REGULAR - Create a Simoncelli filterbank

Usage

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

Description

Inputs parameters: G : Graph structure or lmax 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 $f_l(x)$ 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 adaptated to obtain a tight frame.

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

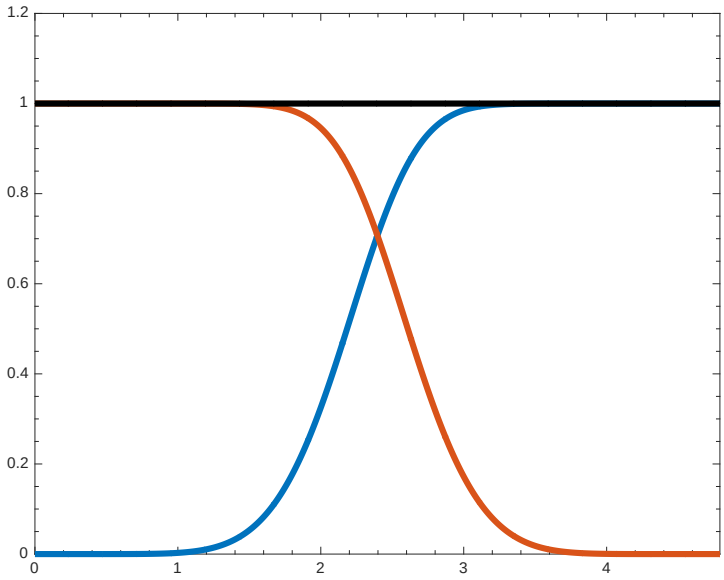
```
G = gsp_estimate_lmax(G);
```

Example:

```
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
```



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 mor informations. (default 3)

4.2.2 GSP_DESIGN_HELD - Create a Held filterbank

Usage

```
g = gsp_design_held( G );
g = gsp_design_held( G, param );
```

Description

Inputs parameters: G : Graph structure or lmax 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 $f_l(x)$:

$$f_l = \begin{cases} 1 & \text{if } x \leq a \\ \sin\left(2\pi\mu\left(\frac{x}{8a}\right)\right) & \text{if } a < x \leq 2a \\ 0 & \text{if } x > 2a \end{cases}$$

with

$$\mu(x) = -1 + 24x - 144 * x^2 + 256 * x^3$$

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

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

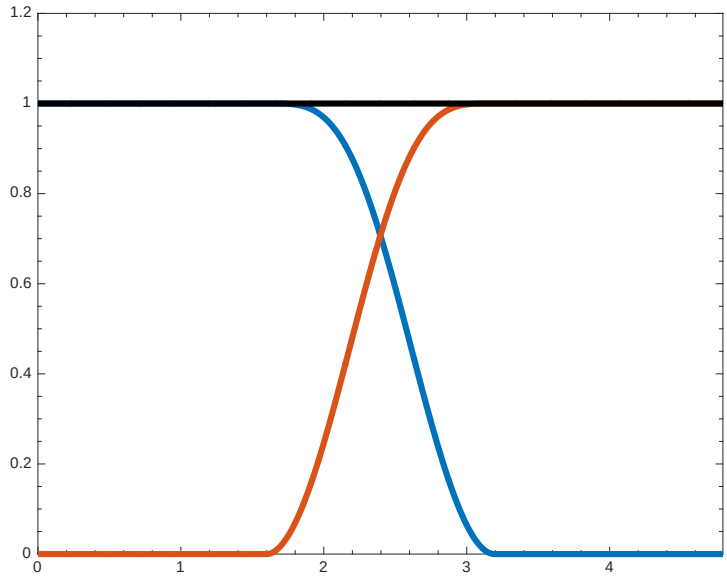
```
G = gsp_estimate_lmax(G);
```

Example:

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

This code produces the following output:

```
A =

    1.0000


B =
```



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

```
g = gsp_design_simoncelli( G );
g = gsp_design_simoncelli( G, param );
```

Description

Inputs parameters: G : Graph structure or lmax 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 $f_l(x)$:

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

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

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

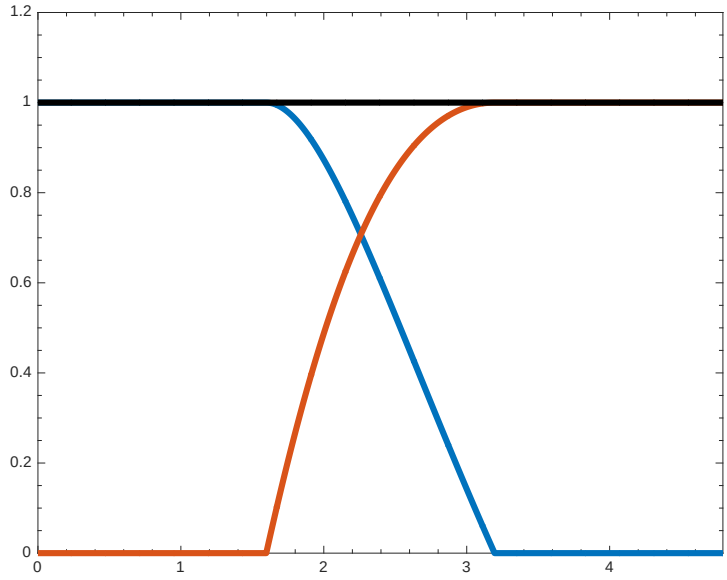
```
G = gsp_estimate_lmax(G);
```

Example:

```
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
```



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).

4.2.4 GSP_DESIGN_PAPADAKIS - Create a Simoncelli filterbank

Usage

```
g = gsp_design_papadakis( G );
g = gsp_design_papadakis( G, param );
```

Description

Inputs parameters: G : Graph structure or lmax 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 $f_l(x)$:

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

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

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

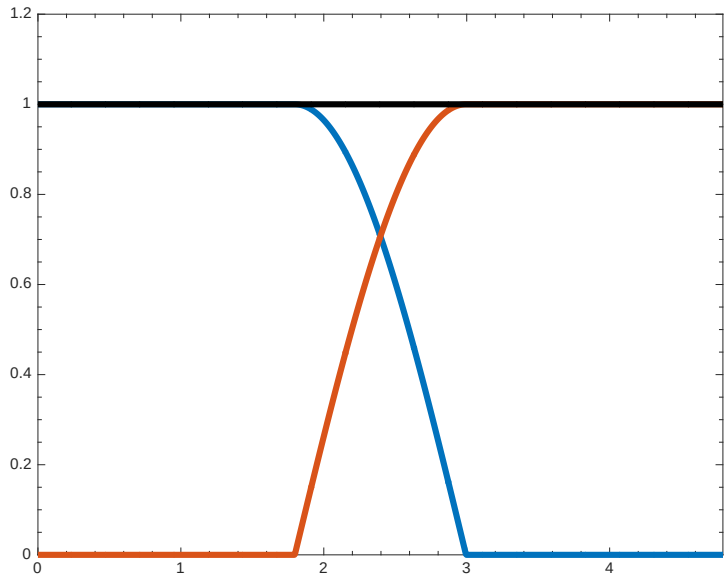
```
G = gsp_estimate_lmax(G);
```

Example:

```
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:

```
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 3/4).

4.3 Dual filterbank

4.3.1 GSP_DESIGN_CAN_DUAL - This function return the canonical dual filters of g

Usage

```
gd = gsp_design_can_dual( g );
```

Description

Inputs parameters: g : cell array of filters tol : tolerance for the pseudo-inverse

Ouputs parameters: g : cell array of filters

This function returns the canonical dual filterbank g. Note that it might not be the be the optimal solution in term of computation.

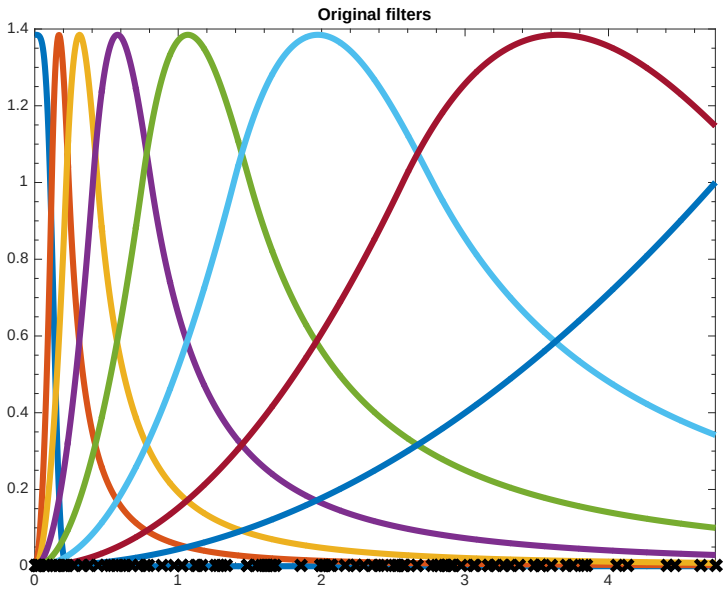
Example:

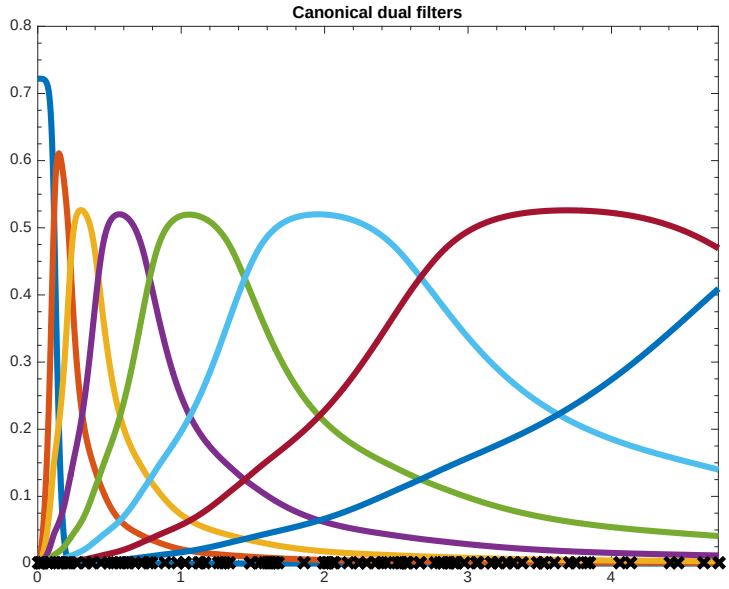
```
N = 100;
G = gsp_sensor(N);
G = gsp_compute_fourier_basis(G);
```

```
g = gsp_design_abspline(G,8);
gd = gsp_design_can_dual(g);
paramplot.show_sum = 0;
figure(1)
gsp_plot_filter(G,g,paramplot);
title('Original filters')
figure(2)
gsp_plot_filter(G,gd,paramplot);
title('Canonical dual filters');

x = rand(N,1);
param.method = 'exact';
coeff = gsp_filter_analysis(G,g,x,param);
xs = gsp_filter_synthesis(G,gd,coeff,param);
norm(xs-x)
```

This code produces the following output:

```
ans =

   1.1039e-14
```





4.3.2 GSP_EVALUATE_CAN_DUAL - Evaluate the canonical dual filterbank

Usage

```
hcoeff = gsp_evaluate_can_dual( g,val )
```

Description

Inputs parameters: g : cell array of filters val : column vectors of values tol : tolerance

Ouputs parameters: s : Matrix of value

This function compute the value of the canonical dual of a filterbank *g* at the point specified in *val*. The function return a matrix. Each column is the output of one dual filter.

4.3.3 GSP_TEST_DUALITY - Test if two filterbanks are dual

Usage

```
bool = gsp_test_duality(G, g,h )
bool = gsp_test_duality(G, g,h,tol )
```

Input parameters

G	Graph or arange (min and max value)
g	filter 1 (or filterbank)
h	filter 2 (or filterbank)
tol	tolerance for the test (default 1e-8)

Description

Ouput paramters: bool : boolean

This function test if two filterbanks are dual.
Example:

```
N = 100;
G = gsp_sensor(N);
G = gsp_estimate_lmax(G);
g = gsp_design_abspline(G,8);
gd = gsp_design_can_dual(g);
gsp_test_duality(G, g,gd )
```

This code produces the following output:

```
ans =

     1
```

4.3.4 GSP_TEST_DUALITY_COEFFICIENT - Test if the coefficient are from dual filters

Usage

```
bool = gsp_test_duality_coefficient( gcoeff,hcoeff );
bool = gsp_test_duality_coefficient( gcoeff,hcoeff,tol );
```

Input parameters

gcoeff	coefficient of the filter 1 (matrix N x M)
hcoeff	coefficinet of the filter 2 (matrix N x M)
tol	tolerance for the test (default 1e-5)

Description

Ouput paramters: bool : boolean

This function test if two discrete filterbanks are dual. Each filter is a culumn in the matrix *gcoeff* or *hcoeff*. M is the number of filters and N the number of coefficients (size of the graph signal).

4.4 Low pass filters

4.4.1 GSP_DESIGN_HEAT - Design a simple heat kernel

Usage

```
gsp_design_heat(G);
gsp_design_heat(G,tau);
gsp_design_heat(G,tau,param);
```

Input parameters

G	Graph structure
tau	scaling parameter (default 10)

Output parameters

g	filter

Description

This function design the following filter:

$$g(x) = e^{-\tau \frac{x}{\lambda_{\max}}}$$

If *tau* is a vector, the function returns a cell array of filters.

param is an optional structure containing the following fields

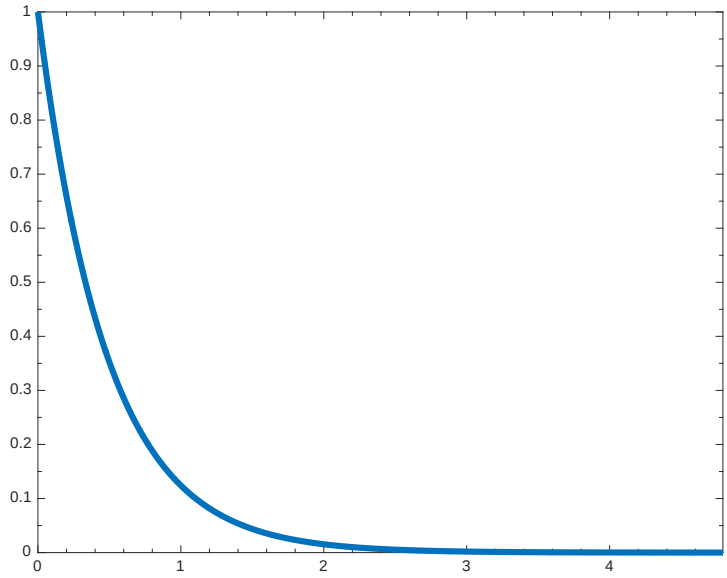
- *param.verbose*: verbosity level. 0 no log - 1 display warnings. (default 1)
- *param.normalize*: Normalize the kernel (works only if the eigenvalues are present in the graph. Use gsp_compute_fourier_basis for this.) (default 0)

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

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

Example:

```
Nf = 4;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_heat(G);
gsp_
```



4.4.2 GSP_DESIGN_EXPWIN - create an expwin window of lenth N with parameter a

Usage

```
g = gsp_design_expwin(G);
g = gsp_design_expwin(G, bmax);
g = gsp_design_expwin(G, bmax, a);
```

Input parameters

G	Graph structure
bmax	Maximum relative band (default 0.2)
a	Slope parameter (default 1).

Output parameters

g	filter

Description

This function design the following filter:

$$g(x) =$$

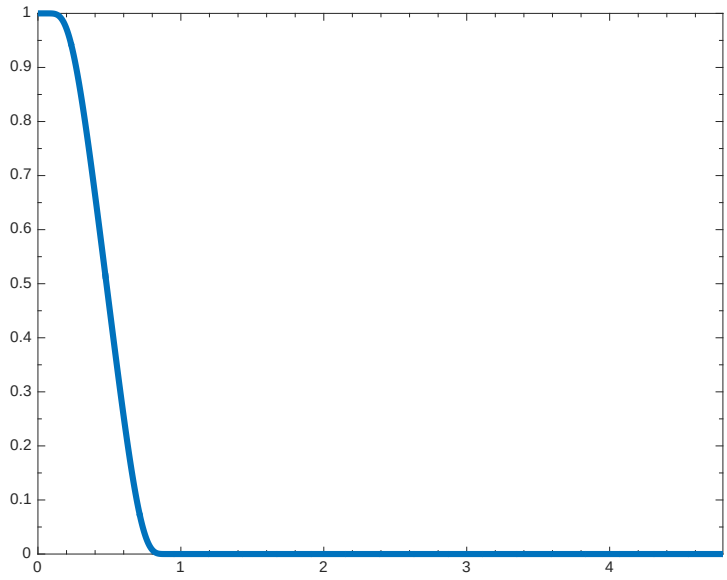
It use a clever exponential construction to obtain a infinitely differentiable function that is band limited!

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

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

Example:

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



4.4.3 GSP_DESIGN_SMOOTH_INDICATOR - create a smooth indicator function

Usage

```
g = gsp_design_smooth_indicator(G, a1);
g = gsp_design_smooth_indicator(G, a1, a2);
```

Input parameters

G	Graph structure
a1	Start of band cuttoff
a2	End of band cuttoff (default 2*a1)

Output parameters

g	filter

Description

This function design the following filter:

$$g(x) = TBD$$

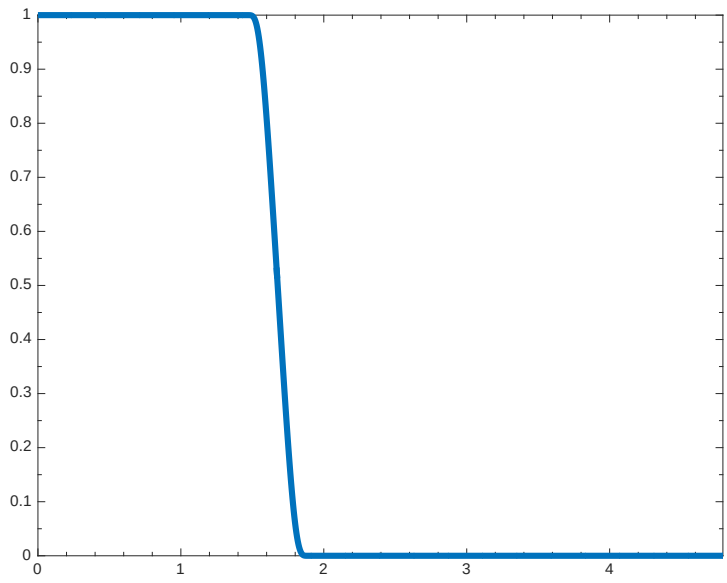
It use a clever exponential construction to obtain a infinitely differentiable function that is band limited!

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

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

Example:

```
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_smooth_indicator(G,0.3,0.4);
gsp_
```



4.5 Application

4.5.1 GSP_FILTER - Filter function

Usage

```
coeffs = gsp_filter(G, fi, signal);
coeffs = gsp_filter(G, fi, signal, param);
```

Input parameters

G	Graph structure.
fi	Spectral filter.
s	Graph signal to filters
param	Optional parameters

Output parameters

c	Filtered signal

Description

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

4.5.2 GSP_FILTER_ANALYSIS - Analysis operator of a gsp filterbank

Usage

```
coeffs = gsp_filter_analysis(G, fi, signal);
coeffs = gsp_filter_analysis(G, fi, signal, param);
```

Input parameters

G	Graph structure.
fi	Set of spectral graph filters.
s	graph signal to analyze.
param	Optional parameter

Output parameters

c	Transform coefficients

Description

'gsp_filter_analysis(G,fi,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.

$$c = D^* f$$

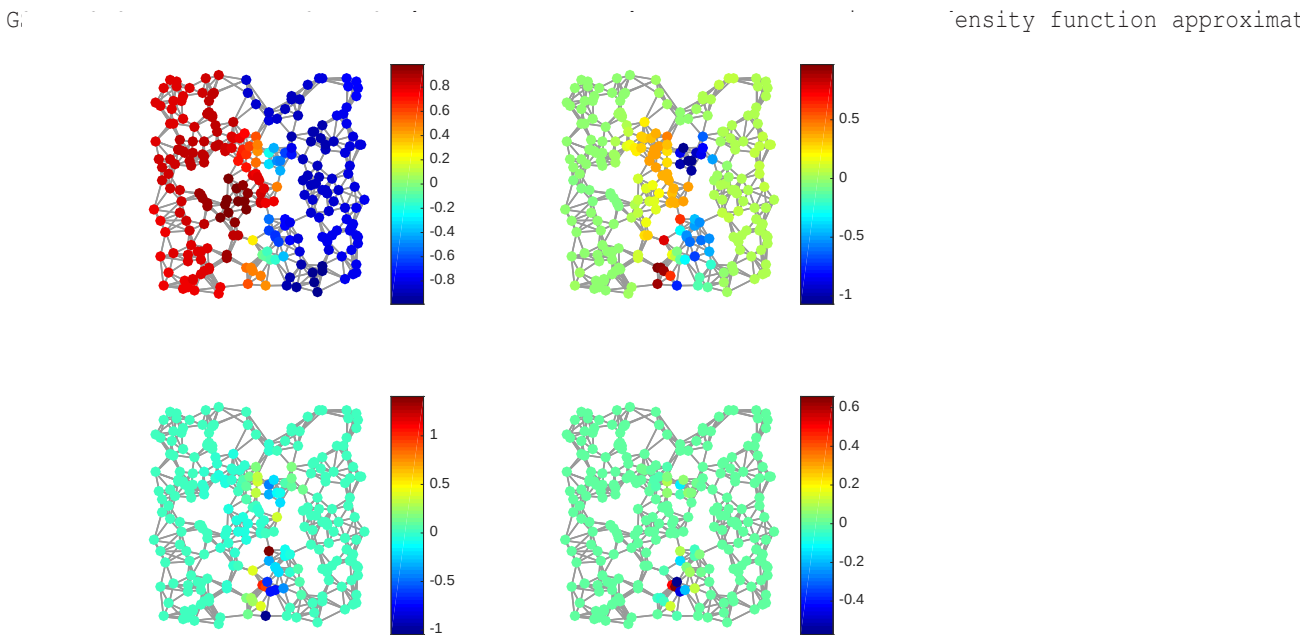
where the columns of D are $g_{i,m} = T_i g_m$, and T_i is a generalized translation operator applied to each filter $\hat{g}_m(\cdot)$.

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

Example:

```
Nf = 5;
param.distribute = 1;
G = gsp_sensor(256);
G = gsp_compute_fourier_basis(G);
paramf.log = 1;
g = gsp_design_warped_translates(G, Nf,paramf);
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:



Additional parameters

- *param.method* : Select the method ot be used for the computation. * 'exact' : Exact method using the graph Fourier matrix * 'cheby' : Chebyshev polynomial approximation * 'lanczos' : Lanczos approximation Default: if the Fourier matrix is present: 'exact' otherwise 'cheby'
- *param.order* : Degree of the Chebyshev approximation (default=30).
- *param.verbose* : Verbosity level (0 no log - 1 display warnings) (default 1).

References: [6]

4.5.3 GSP_FILTER_SYNTHESIS - Synthesis operator of a gsp filterbank

Usage

```
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	sythesis signal

Description

'gsp_filter_synthesis(G,filters,c)' computes the sythesis 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.

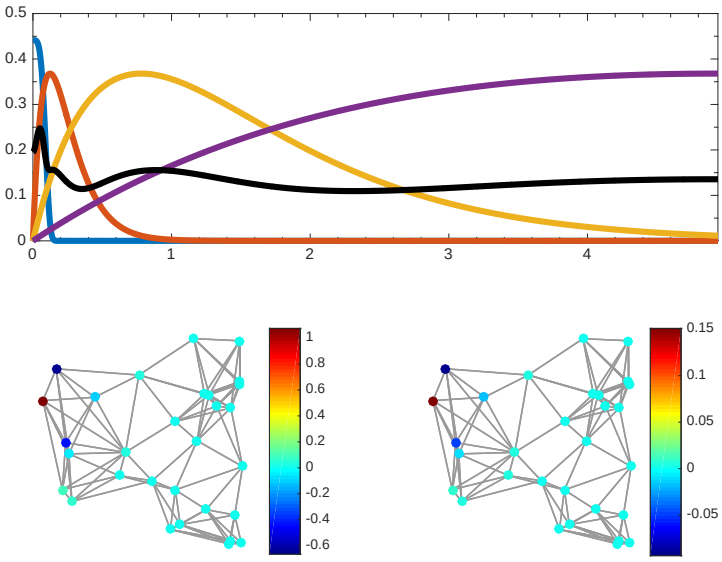
$$f = Dc$$

where the columns of D are $g_{i,m} = T_i g_m$, 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:

```
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_
```



Additional parameters

- *param.method* : Select the method ot be used for the computation. * 'exact' : Exact method using the graph Fourier matrix * 'cheby' : Chebyshev polynomial approximation * 'lanczos' : Lanczos approximation Default: if the Fourier matrix is present: 'exact' otherwise 'cheby'
- *param.order* : Degree of the Chebyshev approximation (default=30).
- *param.verbose* : Verbosity level (0 no log - 1 display warnings) (default 1).

References: [6]

4.5.4 GSP_FILTER_INVERSE - Inverse operator of a gsp filterbank

Usage

```
s = gsp_filter_inverse(G, filter, c);
s = gsp_filter_inverse(G, filter, c, param);
```

Input parameters

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

Output parameters

signal	sythesis signal

Description

'gsp_filter_inverse(G,filters,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.

$$f = (D^* D)^{-1} D c$$

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:

```
Nf = 4;
G = gsp_sensor(30);
G = gsp_estimate_lmax(G);
G = gsp_estimate_lmax(G);
g = gsp_design_mexican_hat(G, Nf);
f = rand(G.N,1);
f = f/norm(f);
ff = gsp_filter_analysis(G,g,f);
f2 = gsp_filter_inverse(G,g,ff);
norm(f-f2)
```

This code produces the following output:

```
ans =

    0.0716
```

For additional parameters, please see gsp_filter_synthesis

References: [6]

4.6 Size Handling

4.6.1 GSP_MAT2VEC - vector to matrix transform

Usage

```
d  = gsp_mat2vec( d );
[ d ,Nf] = gsp_mat2vec( d );
```

Input parameters

d	Data

Description

Ouput parameter d : Data Nf : Number of filter

Reshape the data from the matrix form to the vector form

4.6.2 GSP_VEC2MAT - vector to matrix transform

Usage

```
d  = gsp_vec2mat( d,Nf );
```

Input parameters

d	Data
Nf	Number of filter

Description

Ouput parameter d : Data

Reshape the data from the vector form to the matrix form

4.7 Utils

4.7.1 GSP_APPROX_FILTER - : Create the approximation filter for a filterbank

Usage

```
c = gsp_approx_filter(G, filter, m, N);
c = gsp_approx_filter(G, filter, m);
c = gsp_approx_filter(G, filter);
c = gsp_approx_filter(G, filter, m, N,param);
```

Input parameters

G	graph structure or range of application
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)
param	optional parameter

Output parameters

c	matrix of Chebyshev coefficients

Description

This function create the approximate filters of g with a truncated Chebyshev polynomial.

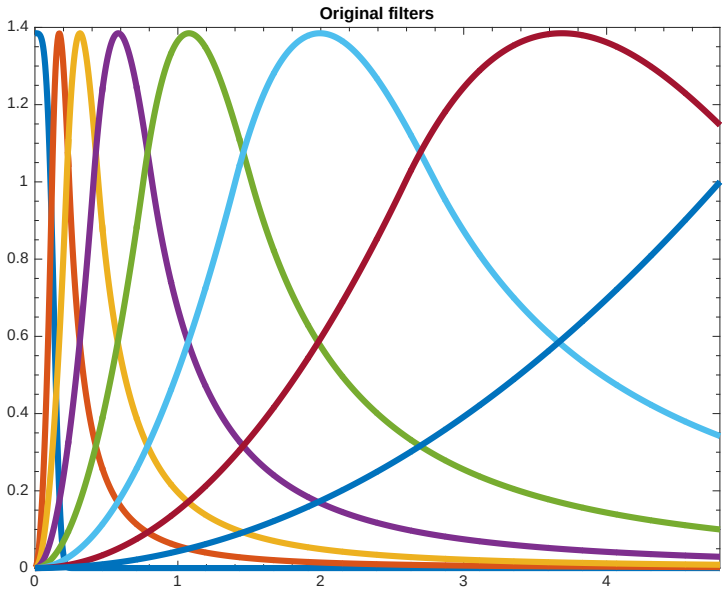
param contain only one field param.verbose to controle the verbosity.

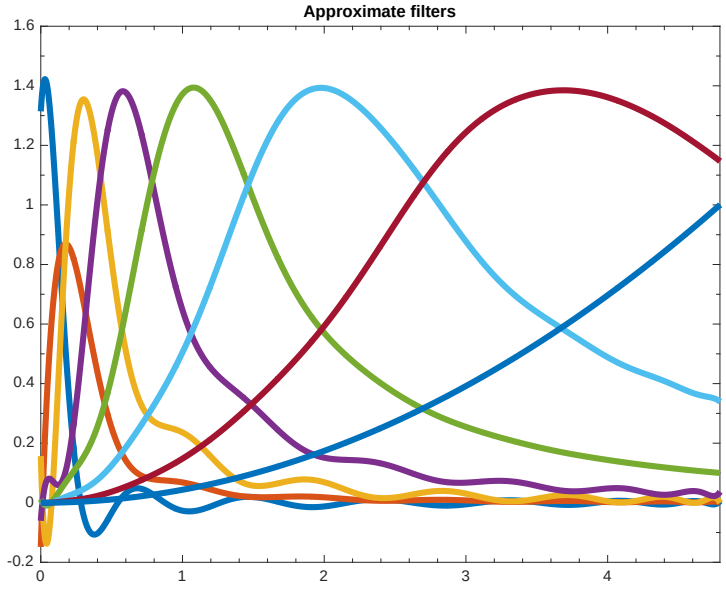
Example:

```
N = 100;
order = 15;
G = gsp_sensor(N);
G = gsp_estimate_lmax(G);
g = gsp_design_abspline(G,8);
ga = gsp_approx_filter(G,g,order);
paramplot.show_sum = 0;
figure(1)
gsp_plot_filter(G,g,paramplot);
title('Original filters')
figure(2)
gsp_plot_filter(G,ga,paramplot);
title('Approximate filters');
x = rand(N,1);
param.order = order;
c1 = gsp_filter_analysis(G,g,x,param);
c2 = gsp_filter_analysis(G,ga,x,param);
norm(c1-c2)/norm(c1)
```

This code produces the following output:

```
ans =

   4.5004e-15
```





4.7.2 GSP_WLOG_SCALES - compute logarithm scales for wavelet

Usage

```
s = gsp_wlog_scales(lmin, lmax, Nscales);
```

Input parameters

lmin	Minimum non zero eigenvalue
lmax	Maximum eigenvalue
Nscales	Number of scale

Output parameters

s	scale

Description

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

returns scales logarithmicaly spaced between minimum and maximum "effective" scales : i.e. scales below minumum or above maximum will yield the same shape wavelet (due to homogoneity of kernel : currently assuming sgwt kernel g given as abspline with t1=1, t2=2)

Note that in design of transform with scaling function, lmin may be taken just as a fixed fraction of lmax, and may not actually be the smallest nonzero eigenvalue

This function is inspired by the sgwt_toolbox.

4.7.3 GSP_FILTER_EVALUATE - Evaluate the filterbank

Usage

```
fd = gsp_filter_evaluate(filter, x)
```

Input parameters

filter	cell array of filter
x	data

Output parameters

fd	response of the filters

Description

This function apply all the filters in *filter* to the data *x*. Every filter correspond to one column of the matrix *fd*.

4.7.4 GSP_FILTERBANK_BOUNDS - Compute approximate frame bounds for a filterbank

Usage

```
[A, B] = gsp_filterbank_bounds(G,W);
[A, B] = gsp_filterbank_bounds(G,W,param);
[A, B] = gsp_filterbank_bounds([xmin, xmax],W);
[A, B] = gsp_filterbank_bounds([xmin, xmax],W,param);
```

Input parameters

G	Graph structure or interval to compute the bound
W	Filterbank (cell array of inline function)
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 999)
- *param.use_eigenvalues* : Use eigenvalues if possible (default 1). To be used, the eigenvalues have to be computed first using gsp_compute_fourier_basis.

4.7.5 GSP_TIGHTEN_FILTER - Create a function that tighten a filterbank

Usage

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

Input parameters

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

Description

Ouput parameters: ftighten : Inline function

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

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

4.7.6 GSP_WARP_FILTER - Warp the filterbank g with the filter w

Usage

```
gw = gsp_warp_filter(g,w);
```

Input parameters

g	filterbank
w	warping filter

Output parameters

gw	warped filterbank

Description

The resulting filter gw is $gw(x) = w(g(x))$.

4.7.7 GSP_MULTIPLY_FILTERS - Mutliply to filters

Usage

```
gm = gsp_multiply_filters(g1,g2);
```

Input parameters

g1	filterbank
g2	filterbank

Output parameters

gm	multiplied filterbank

Description

The resulting filter is $gm(x) = g1(x)g2(x)$.

4.7.8 GSP_FILTERBANK_MATRIX - Create the matrix of the filterbank frame

Usage

```
F = gsp_filterbank_matrix(G, g param );
F = gsp_filterbank_matrix(G,g);
```

Input parameters

G	Graph
g	Filters
param	Structure of optional parameter

Output parameters

F	Frame

Description

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

param a Matlab structure containing the following fields:

- *param.verbose* : 0 no log, 1 print main steps, 2 print all steps. By default, it is 1.

Chapter 5

GSPBOX - Operators

5.1 Localisation

5.1.1 GSP_LOCALIZE - Localize a kernel g to the node i

Usage

```
gt = gsp_localize(G, g, i);
```

Input parameters

G	Graph
g	kernel (or filterbank)
i	Indices of vertex (int)
param	Optional parameters

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.

5.1.2 GSP_MODULATE - Generalized modulation of the signal f to the frequency k

Usage

```
fm = 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 applicated to each column.

5.1.3 GSP_TRANSLATE - Generalized translation of the signal f to the node i

Usage

```
ft = 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 translate the column vector f onto the node i. If f is a matrix, the translation will be done to each column.

5.2 Differential

5.2.1 GSP_GRAD_MAT - Gradient sparse matrix of the graph G

Usage

```
D = 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 effiecient, call the function:

```
G = gsp_adj2vec(G)
```

before this function.
Example:

```
N = 40;
G = gsp_sensor(N);
G = gsp_adj2vec(G);
D = gsp_grad_mat(G);
```

5.2.2 GSP_GRAD - Graph gradient

Usage

```
gr = gsp_grad(G,s)
```

Input parameters

G	Graph structure
s	Signal living on the nodes

Output parameters

gr	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 = gsp_adj2vec(G)
```

5.2.3 GSP_DIV - Graph divergence

Usage

```
di = gsp_div(G,s)
```

Input parameters

G	Graph structure
s	Signal living on the edges

Output parameters

di	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:

```
gsp_div(G,gsp_grad(G,s)) = G.L * s,
```

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

```
G = gsp_adj2vec(G)
```

5.3 Transforms

5.3.1 GSP_GFT - Graph Fourier transform

Usage

```
f_hat=gsp_gft(G,f);
```

Input parameters

G	Graph or Fourier basis
f	f (signal)

Output parameters

f_hat	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.

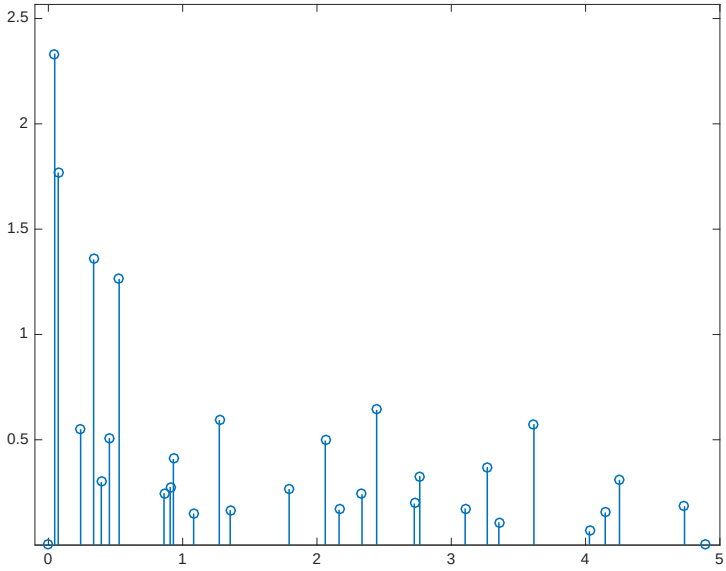
$$\hat{f}(\lambda_\ell) = \langle f, u_\ell \rangle$$

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

```
G = gsp_compute_fourier_basis(G);
```

Example:

```
N = 30;
G = gsp_sensor(N);
G = gsp_compute_fourier_basis(G);
f = sin((1:N)'*2*pi/N);
fhat = gsp_gft(G,f);
gsp_
```



5.3.2 GSP_IGFT - Inverse graph Fourier transform

Usage

```
f = 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: G.U. Alternatively, one can provide directly the Fourier basis instead of the graph G.

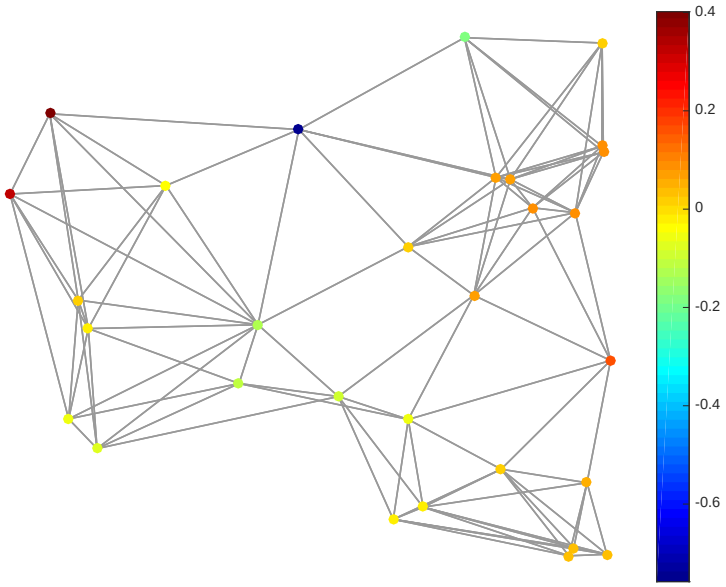
$$\hat{f}(\lambda_\ell) = \langle f, u_\ell \rangle$$

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

```
G = gsp_compute_fourier_basis(G);
```

Example:

```
N = 30;
G = gsp_sensor(N);
G = gsp_compute_fourier_basis(G);
f_hat = zeros(N,1);
f_hat(5) = 1;
f = gsp_igft(G,f_hat);
```



5.3.3 GSP_GWFT - Generalized windowed Fourier transform

Usage

```
C = gsp_gwft(G,g,f, param );
C = gsp_gwft(G,g,f);
```

Input parameters

G	Graph
g	Window (graph signal or kernel)
f	Graph signal (column vector)
param	Structure of optional parameter

Output parameters

C	Coefficient.

Description

This function compute the graph windowed Fourier transform of a signal f with the window g. The function returns a matrix of size N^2*N.

param a Matlab structure containing the following fields:

- *param.verbose* : 0 no log, 1 print main steps, 2 print all steps. By default, it is 1.
- *param.lowmemory* : use less memory. By default, it is 1.

Reference: shuman2013windowed

5.3.4 GSP_NGWFT - Normalized graph windowed Fourier transform

Usage

```
G = gsp_ngwft(G,f,g, param);
G = gsp_ngwft(G,f,g);
```

Input parameters

G	Graph
f	Graph signal
g	window
param	Structure of optional parameter

Output parameters

C	Coefficient

Description

This function compute the normalized graph windowed Fourier transform of a signal f with the window g. The function returns a matrix of size N^2*N.

param a Matlab structure containing the following fields:

- *param.verbose* : 0 no log, 1 print main steps, 2 print all steps. By default, it is 1.
- *param.lowmemory* : use less memory. By default, it is 1.

5.4 Pyramid - Reduction

5.4.1 GSP_KRON_REDUCE - Performs Kron reduction

Usage

```
Greduced=gsp_kron_reduce(G,keep_inds);
Lreduced=gsp_kron_reduce(L,keep_inds);
```

Input parameters

G	Graph structure or graph Laplacian matrix.
keep_inds	The set of indices to keep in the reduced graph.

Output parameters

Greduced	The Kron-reduced graph structure or Laplacian.

Description

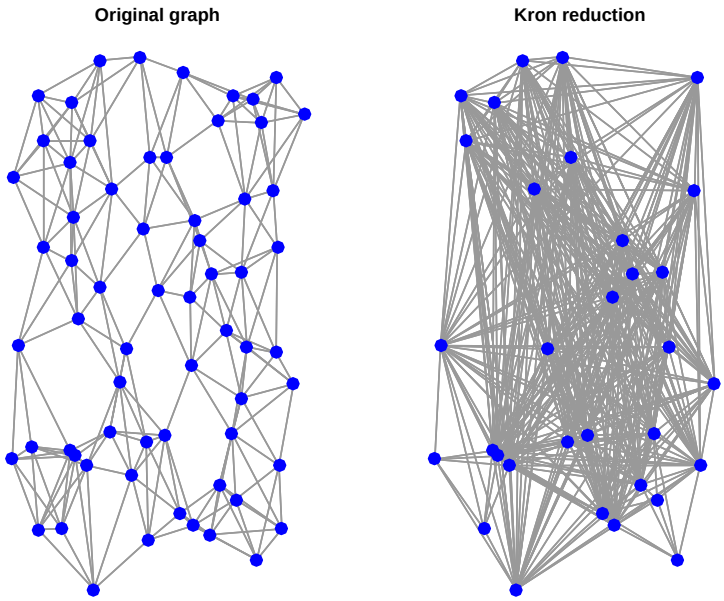
'gsp_kron_reduce(G,keep_inds)' performs Kron reduction:

$$\mathcal{L}_{reduced} = \mathcal{L}_{\mathcal{V}_1,\mathcal{V}_1} - \mathcal{L}_{\mathcal{V}_1,\mathcal{V}_2}\left[\mathcal{L}_{\mathcal{V}_2,\mathcal{V}_2}\right]^{-1}\mathcal{L}_{\mathcal{V}_2,\mathcal{V}_1}$$

If a matrix is given, then a matrix is returned

Example:

```
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)
gsp_plot_graph(G);
title('Original graph');
subplot(122)
gsp_plot_graph(Gnew);
title('Kron reduction');
```



Notes: may be able to speed this up with LAMG toolbox

References: [4]

5.4.2 GSP_GRAPH_MULTIRESOLUTION - Compute a multiresolution of graphs

Usage

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

Input parameters

G	Graph structure
num_levels	Number of times to downsample and coarsen the graph
param	Optional structure of parameters

Output parameters

Gs	Cell array of graphs

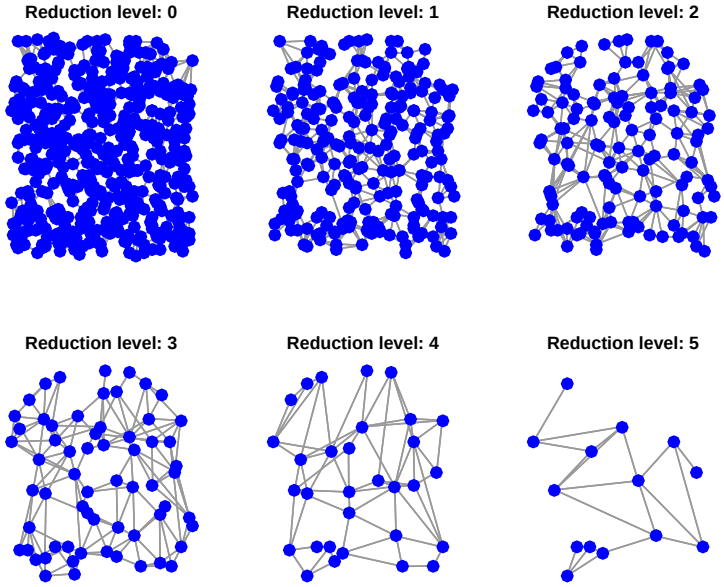
Description

'gsp_graph_multiresolution(G,num_levels)' computes a multiresolution of graph by repeatedly downsampling and performing graph reduction. The default downsampling method is the largest eigenvector method based on the polarity of the components of the eigenvector associated with the largest graph Laplacian eigenvalue. The default graph reduction method is Kron reduction followed by a graph sparsification step. *param* is a structure of optional parameters containing the following fields:

- *sparsify*: To perform a spectral sparsification step immediately after the graph reduction (default=1)
- *sparsify_epsilon*: Parameter epsilon used in the spectral sparsification (default=min(10/sqrt(G.N),.3))
- *downsampling_method*: The graph downsampling method (default='largest_eigenvector')
- *reduction_method*: The graph reduction method (default='Kron')
- *compute_full_eigen*: To also compute the graph Laplacian eigenvalues and eigenvectors for every graph in the multiresolution sequence (default=0)

Example:

```
N = 500;
G = gsp_sensor(N);
Nlevel = 5;

Gs = gsp_graph_multiresolution(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

Authors : David I Shuman, Elle Weeks, Andre Archer, Stefan Faridani, Yan Jin, Nathanael Perraudin. Date: 26 November 2015 Testing: test_operators

References: [14]

5.4.3 GSP_PYRAMID_ANALYSIS - Compute the graph pyramid transform coefficients

Usage

```
[coarse_approximations,prediction_errors]=gsp_pyramid_analysis(Gs,signal,num_levels);
[coarse_approximations,prediction_errors]=gsp_pyramid_analysis(Gs,signal,num_levels,param);
```

Input parameters

Gs	A multiresolution sequence of graph structures, including the idx parameters tracking the subsampling pattern.
signal	Graph signal to analyze.
num_levels	Number of levels in the pyramid transform.

Output parameters

coarse_approximations	Cell array with the coarse approximations at each level.
prediction_errors	Cell array with the prediction errors at each level.

Description

'gsp_pyramid_analysis(Gs,signal,num_levels)' computes the graph pyramid transform coefficients of a signal f.

param is a structure containing optional arguments including

- *param.regularize_epsilon* : Interpolation parameter.
- *param.h_filters* : A cell array of graph spectral filters. If just one filter is included, it is used at every level of the pyramid. Default

$$h(x) = \frac{0.5}{0.5+x}$$

Please read the documentation of gsp_filter_analysis for other optional arguments.
Demo: gsp_demo_pyramid
References: [14]

5.4.4 GSP_PYRAMID_ANALYSIS_SINGLE - Compute a single level of the graph pyramid transform coefficients

Usage

```
[coarse_approximation,prediction_error]=gsp_pyramid_analysis_single(G,signal,keep_inds,h_filter,
```

Input parameters

signal	Graph signal to analyze.
G	Graph structure on which the signal resides.
keep_inds	The indices of the vertices to keep when downsampling the graph and signal.
h_filter	The H filter in the pyramid transform.

Output parameters

coarse_approximation	The next coarse approximation.
prediction_error	The next prediction error.

Description

'gsp_pyramid_analysis_single(G,signal,keep_inds,h_filter,param)' computes a single level of the graph pyramid transform coefficients.

param is a structure containing optional arguments including * *param.regularize_epsilon* : Interpolation parameter.

Please read the documentation of gsp_filter_analysis for other optional arguments.
Demo: gsp_demo_pyramid
References: [14]

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

Usage

```
reconstruction=gsp_pyramid_synthesis(Gs,coarsest_approximation,prediction_errors);
reconstruction=gsp_pyramid_synthesis(Gs,coarsest_approximation,prediction_errors,param);
```

Input parameters

Gs	A multiresolution sequence of graph structures, including the idx parameters tracking the subsampling pattern.
coarsest_approximation	The coarsest approximation of the original signal.
prediction_errors	Cell array with the prediction errors at each level.

Output parameters

reconstruction	The synthesized signal.
coarse_approximations	Sequence of coarse approximations

Description

'gsp_pyramid_synthesis(Gs,coarsest_approximation,prediction_errors)' synthesizes a signal from its graph pyramid transform coefficients.

param is a structure containing optional arguments including

- *param.regularize_epsilon* : Interpolation parameter.
- *param.least_squares* : Set to 1 to use the least squares synthesis (default=0)
- *param.use_landweber* : Set to 1 to use the Landweber iteration approximation in the least squares synthesis.
- *param.landweber_its* : Number of iterations in the Landweber approximation for least squares synthesis.
- *param.landweber_tau* : Parameter for the Landweber iteration.
- *param.h_filters* : A cell array of graph spectral filters. If just one filter is included, it is used at every level of the pyramid. These filters are required for least squares synthesis, but not for the direct synthesis method Default

$$h(x) = \frac{0.5}{0.5+x}$$

Please read the documentation of gsp_filter_analysis for other optional arguments.

Demo: gsp_demo_pyramid

References: [14]

5.4.6 GSP_PYRAMID_SYNTHESIS_SINGLE - Synthesize a single level of the graph pyramid transform

Usage

```
[finer_approximation]=gsp_pyramid_synthesis_single(G,coarse_approximation,prediction_error,keep_
```

Input parameters

G	Graph structure on which the signal resides.
coarse_approximation	Coarse approximation on a reduced graph.
prediction_error	Prediction error that was made when forming the coarse approximation.
keep_inds	Indices of the vertices to keep when downsampling the graph and signal.
param	Additional parameters.

Output parameters

finer_approximation	Coarse approximation of the signal on a higher resolution graph.

Description

'gsp_pyramid_synthesis_single(G,coarse_approximation,prediction_error,keep_inds,param)' synthesizes a single level of the graph pyramid transform.

param is a structure containing optional arguments including

- *param.regularize_epsilon* : Interpolation parameter.
- *param.least_squares* : Set to 1 to use the least squares synthesis (default=0)
- *param.use_landweber* : Set to 1 to use the Landweber iteration approximation in the least squares synthesis.
- *param.landweber_its* : Number of iterations in the Landweber approximation for least squares synthesis.
- *param.landweber_tau* : Parameter for the Landweber iteration.
- *param.h_filter* : A graph spectral filter. This filter is required for least squares synthesis, but not for the direct synthesis method

Please read the documentation of gsp_filter_analysis for other optional arguments.
Demo: gsp_demo_pyramid
References: [14]

5.4.7 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

5.4.8 GSP_INTERPOLATE - Interpolation of a graph signal

Usage

```
f_interpolated=gsp_interpolate(f_subsampled, G, keep_inds);
f_interpolated=gsp_interpolate(f_subsampled, G, keep_inds, param);
```

Input parameters

f_subsampled	A signal on the subset of the vertices of G indexed by keep_inds
G	Graph structure.
keep_inds	The vertex indices of V_1

Output parameters

f_interpolated	Interpolated graph signal on G.

Description

'gsp_interpolate(f_subsampled,G,keep_inds)' performs the interpolation:

$$f_{interpolated} = \Phi_{\mathcal{V}_1} \Phi^{-1}_{\mathcal{V}_1,\mathcal{V}_1} f_{\mathcal{V}_1} = \Phi_{\mathcal{V}_1} \left[\bar{\mathcal{L}}_{\mathcal{V}_1,\mathcal{V}_1} - \bar{\mathcal{L}}_{\mathcal{V}_1,\mathcal{V}_1^c} (\bar{\mathcal{L}}_{\mathcal{V}_1^c,\mathcal{V}_1^c})^{-1} \bar{\mathcal{L}}_{\mathcal{V}_1^c,\mathcal{V}_1} \right] f_{\mathcal{V}_1}$$

Note that *keep_inds* are the vertex indices of V_1; i.e., those where the signal values are available to use in the interpolation.

param contains the following fields

- *param.order* : Degree of the Chebyshev approximation (default=100 to yield a good approximation of the default Green's kernel). This parameter is passed to gsp_filter_analysis.
- *param.regularize_epsilon* : The regularized graph Laplacian is $\bar{L} = L + \varepsilon I$ (default = 0.005). A smaller epsilon may lead to better regularization, but will also require a higher order Chebyshev approximation.

References: [10]

Chapter 6

GSPBOX - Pointclouds

6.1 Generate pointclouds

6.1.1 GSP_TWOSPIRALS - Generate "two spirals" dataset with N instances

Usage

```
gsp_twospirals(N, degrees, start, noise);
gsp_twospirals(N, degrees, start);
gsp_twospirals(N, degrees);
gsp_twospirals(N);
gsp_twospirals();
```

Description

Input arguments: N : Number of points (default 2000) degrees : The length of the spirals in degree (default 570) start : how far from the origin the spirals start, in degrees (default 90) noise : Noise level (default 0.2)

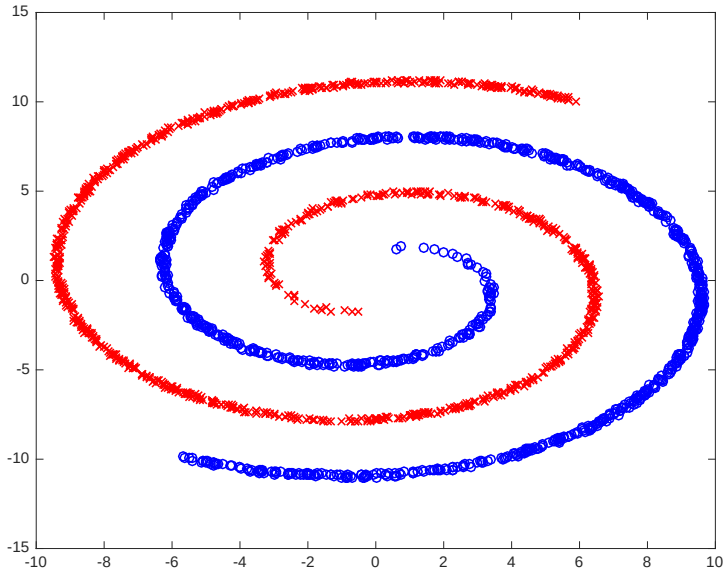
Output arguments: x : Position of the points y : label

Note that for *noise=0*, there is no noise and at *noise=1* the spirals will start overlapping

This function is adaptated from: http://stackoverflow.com/questions/16146599/create-artificial-data-in-ma and http://stackoverflow.com/questions/5837572/generate-a-random-point-within-a-circle-uniformly

Example:

```
[x,y] = gsp_twospirals();
figure();
plot(x(y>0,1),x(y>0,2),'xr');
hold on
plot(x(y<=0,1),x(y<=0,2),'ob');
hold off
```



6.2 Utils

6.2.1 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: [19]

Chapter 7

GSPBOX - Proximal operators and other solvers

7.1 Gradient based proximal operators

7.1.1 GSP_PROX_TV - Proximal TV operator for graphs signal

Usage

```
sol = gsp_prox_tv(x, gamma, G, param)
sol = gsp_prox_tv(x, gamma, G)
[sol, info] = gsp_prox_tv(...)
```

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

Description

This function computes 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, param)` 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

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

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

- *param.maxit* : max. nb. of iterations (default: 200).
- *param.A* : Forward operator (default: Id). This parameter allows to solve the following problem

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

- *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_martrix* : 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 exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

Demo: gsp_demo_graph_tv

7.1.2 GSP_PROX_TIK - Proximal tikhonov operator for graphs

Usage

```
sol = gsp_prox_tik(x, gamma, G, param)
sol = gsp_prox_tik(x, gamma, G)
[sol, info] = 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

Description

This function require the UNLocBoX to be executed.
`gsp_prox_tik(y, gamma, param)` solves:

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

Note the nice following relationship

$$x^T L x = ||\nabla x||_2^2$$

param is a Matlab structure containing the following fields:

- *param.tol* : 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=10e-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

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|\nabla A x\|_2^2$$

- *param.At* : Adjoint operator (default: Id).
- *param.pcg* : Use the fast PCG algorithm (default 1).
- *param.use_martrix* : 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.

$$\|Ax\|^2 \leq \nu\|x\|^2$$

info is a Matlab structure containing the following fields:

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

7.2 Filterbank based proximal operators

7.2.1 GSP_PROJ_B2_FILTERBANK - Projection on the B2 ball for a filterbank

Usage

```
sol = gsp_proj_b2_filterbank(x, T, G, W, param);
sol = gsp_proj_b2_filterbank(x, T, G, W);
[sol, info] = gsp_proj_b2_filterbank(...)
```

Input parameters

x	Input signal.
gamma	Compatibility parameter
G	Graph structure
W	Filterbank (cell array of functions)
y	Measurements
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, gamma, G, W, param)` 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.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=10e-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').
- *param.normalize*: normalize the frame (for testing purpose only) this is not efficient

info is a Matlab structure containing the following fields:

- *info.algo* : Algorithm used
- *info.iter* : Number of iteration
- *info.time* : Time of exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion 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 informations at convergence

Description

This function require the UNLocBoX to be executed.
`gsp_prox_l2_filterbank(x, gamma, G, W, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma\|AW^*x - y\|_2^2$$

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 $n(t) = f(x) + 0.5\|x-z\|_2^2$ is the objective function at iteration t by default, `tol=10e-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 exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

7.2.3 GSP_PROX_L1_FILTERBANK - Proximal L1 operator for a filterbank

Usage

```
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 informations at convergence

Description

This function require the UNLocBoX to be executed.
`gsp_prox_l1_filterbank(x, gamma, G, W, param)` solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 + \gamma \|Wx\|_1$$

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* : 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=10e-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 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 exectution of the function in sec.
- *info.final_eval* : Final evaluation of the function
- *info.crit* : Stopping critterion used

7.2.4 GSP_PROJ_FILTERBANK - Projection onto the synthesis coefficients

Usage

```
sol = gsp_proj_filterbank(x, 0, G, W, y, param);
sol = gsp_proj_filterbank(x, 0, G, W, y);
[sol, info] = gsp_proj_filterbank(...)
```

Input parameters

x	Input signal
G	Graph structure
W	Filterbank (cell array of functions)
y	Measurements
param	Structure of optional parameters

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

`gsp_proj_filterbank(x, gamma, G, W, param)` can solves:

$$sol = \min_z \frac{1}{2}\|x-z\|_2^2 \text{ s. t. } W^*x = y$$

Where W is the linear analysis operator associated with the filterbank.
param is a Matlab structure containing the following fields:

- *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)

info is a Matlab structure containing the following fields:

- *infos.algo* : Algorithm used
- *infos.iter* : Number of iteration
- *infos.time* : Time of exectution of the function in sec.
- *infos.final_eval* : Final evaluation of the function
- *infos.crit* : Stopping critterion used

7.3 Wavelet based operator

7.3.1 GSP_WAVELET_DN - Wavelet denoising

Usage

```
sol = gsp_wavelet_dn(G, w, x, lambda, param)
sol = gsp_wavelet_dn(G, w, x, lambda)
[sol, info] = gsp_wavelet_dn(...)
```

Input parameters

G	Graph structure
w	Wavelet filterbank
x	Signal to be denoised
lambda	Regularization parameter
param	Structure of optional parameters.

Output parameters

sol	Solution.
info	Structure summarizing informations at convergence

Description

This function will denoise a signal by solving the following convex promblem:

$$sol = \min_z \|W^* z - x'\|_2^2 + \gamma \|z\|_{TV}$$

Where W is the frame associated to the filterbank w, x' a part of the signal to be denoised and z the wavelet coefficient.

x' consists of the high frequency part of x. It is obtained by setting down to zero the low pass filter of the filerbank w.

This function require the UNLocBoX to be executed. You can download it at `http://unlcobox.sourceforge.net`

param is a Matlab structure containing the following fields:

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

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

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

- *param.maxit* : max. nb. of iterations (default: 200).

- *param.tight* : 1 W^* are both tight frame or 0 if not (default = 0)
- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.method* : Solver to be used ('FISTA', 'ISTA', 'DG') By default it is 'FISTA'. ('DG' is Douglas Rachford)
- *param.gamma* : stepsize for the 'DG' algorithm

info is a Matlab structure containing the following fields:

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

Demo: gsp_demo_wavelet_dn

7.3.2 GSP_SOLVE_L1 - Solve a BP problem using l1

Usage

```
alpha = gsp_solve_l1(G, W, s, lambda );
alpha = gsp_solve_l1(G, W, s, lambda, param );
```

Input parameters

G	Graph
W	Filterbank
s	Signal
lambda	Regularization parameter
param	Optional structure of parameters

Description

Ouptut parameters: alpha : Filterbank coefficients

This function solve the following minimization problem
In *param*, you can optionaly set:

- *param.verbose* : 0 no log, 1 a summary at convergence, 2 print main steps (default: 1)
- *param.normalize*: Use weight on the l1 norm to remove the effect of the size of the atoms of W. Default 0
- *param.guess*: Initial guess for the starting point. (Default zeros)

info is a Matlab structure containing optimization information from the UNLocBoX

7.3.3 GSP_SOLVE_L0 - Solve a BP problem using l0

Usage

```
alpha = gsp_solve_l0(G, W, s, lambda );
alpha = gsp_solve_l0(G, W, s, lambda, param );
```

Input parameters

G	Graph
W	Filterbank
s	Signal
lambda	regularization parameter
param	Optional structure of parameters

Description

Ouptut parameters: alpha : Filterbank coefficients

This function solve the following minimization problem
In *param*, you can optionaly set:

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

info is a Matlab structure containing optimization information from the UNLocBoX

Chapter 8

GSPBOX - Embeddings

8.1 Availlable embeddings

8.1.1 GSP_LLE - Local Linear Embedding

Usage

```
coords = gsp_lle(G, dim);
coords = gsp_lle(G, dim, param);
```

Input parameters

G	Graph
dim	Dimensionality of the embedding
param	Structure of optional parameters

Output parameters

coords	Coordinates of the embedding

Description

This function uses the weight matrix of a graph G, in order to compute a *dim* -dimensional embedding (output coordinates). The algorithm used is Locally Linear Embedding (LLE). Warning, this function might not work if the graph is not connected.

param is a structure with optional parameters:

- *param.tol* : Tolerance for the spectral gap (default 1e-6).
- *param.kernel* : The kernel used to create the graph weight matrix: + 'exp' : exponential kernel ($e^{\langle} frac - d_{ij} sigma^2$)) + '1/x' : inverse of x kernel ($frac 1 sigma + d_{ij}$) + '1/x^2' : inverse of x^2 kernel ($frac 1 (sigma + d_{ij})^2$) + 'resistance' : Resistance distance.
- *param.k* : Max number of nearest neighbors. If not defined, the

number of neighbors varies from node to node since the algorithm considers all the columns j of the weight matrix that have non zero values on the line i as the neighbors of i.

Demo: gsp_demo_graph_embedding

References: [13]

8.1.2 GSP_LAPLACIAN_EIGENMAPS - Laplacian eigenmaps embedding

Usage

```
coords = gsp_laplacian_eigenmaps(G, dim);
coords = gsp_laplacian_eigenmaps(G, dim, param);
```

Input parameters

G	Graph
dim	Dimension of the embedding
param	Structure of optional paramters

Output parameters

coords	Coordinates of the embedding

Description

This function uses the weight matrix of a graph G, in order to compute a *dim* -dimensional embedding (output coordinates). The algorithm used is Laplacian eigenmaps. Warning, this function might not work if the graph is not connected.

param is a structure optional parameters:

- *param.tol* : Tolerance for the spectral gap (default 1e-6).

Demo: gsp_demo_graph_embedding
References: [1]

8.1.3 GSP_ISOMAP - isomap

Usage

```
coords = gsp_isomap(G, dim, param);
coords = gsp_isomap(G, dim);
```

Input parameters

G	Graph
dim	Dimensionality of the embedding
param	Structure of optional parameters

Output parameters

coords	Coordinates of the embedding

Description

This function uses the weight matrix of a graph G, in order to compute a *dim* -dimensional embedding (output coordinates). The algorithm used is Isomap. Warning, this function might not work if the graph is not connected.

param is a structure with optional parameters:

- *param.tol* : Tolerance for the spectral gap (default 1e-6).
- *param.kernel* : The kernel used to create the graph weight matrix: + 'exp' : exponential kernel ($e^{\backslash} frac - d_{ij} sigma^2$)) + '1/x' : inverse of x kernel ($frac 1 sigma + d_{ij}$) + '1/x^2' : inverse of x^2 kernel ($frac 1 (sigma + d_{ij})^2$) + 'resistance' : Resistance distance.

- *param.k* : Max number of nearest neighbors. If not defined, the

Demo: gsp_demo_graph_embedding
References: [18]

8.2 Utils

8.2.1 GSP_WEIGHT2DISTANCE - Distance matrix from weight matrix

Usage

```
D = gsp_weight2distance(G);
D = gsp_weight2distance(G, method);
D = gsp_weight2distance(G, method, nnn);
```

Input parameters

G	Graph or weight matrix
method	Type of kernel used to compute the weight matrix
sigma	kernel paramter

Output parameters

D	Distance matrix (sparse)

Description

This function uses the weight matrix of a graph in order to compute the distance matrix *D*. If G is a structure it is considered as a graph otherwise it is considered as a weight matrix. The resulting matrix is in sparse form. The indices idx are of the nearest neighbors or (if k is not used) the indices of all non zero elements of the weight matrix. In both cases indices are saved in a cell where idx{ii} contains a vector jj of indices where D(ii, jj) are the Distances of the nearest neighbours of element ii.

The type of *method* one can use are the following: * 'exp' : exponential kernel ($e^{(\frac{-d_{ij}}{\sigma^2})}$) * '1/x' : inverse of x kernel ($\frac{1}{\sigma+d_{ij}}$) * '1/x^2' : inverse of x^2 kernel ($\frac{1}{(\sigma+d_{ij})^2}$) * 'resistance' : Resistance distance.

8.2.2 GSP_COMPUTE_COORDINATES - calculate coordinates

Usage

```
coords = gsp_compute_coordinates( G , dim );
coords = gsp_compute_coordinates( G , dim, method );
coords = gsp_compute_coordinates( G , dim, method, param );
```

Input parameters

G	Graph
dim	Number of lower dimentions
method	Method to be used
param	Structure of optional paramter

Output parameters

coords	New coordinates of graph that lives in a lower dimention

Description

This function computes coordinates from the weighted adjacency matrix. The available methods are:

- 'laplacian_eigenmaps' : Laplacian Eigenmaps. For further information, see the help of gsp_laplacian_eigenmaps (default)
- 'lle' : LLE (Loccaly Linear Embedding). For further information, see the help of gsp_lle
- 'isomap' : Isomap. For further information, see the help of gsp_isomap

The coordinates can be insterted inside the graph structure with the function gsp_update_coordinates.
param is a structure forwarding optiona paramter to the different functions.

- *param.tol* : Tolerance for the spectral gap (default 1e-6). For further parameters, check the selected method function's help.

Demo: gsp_demo_graph_embedding

Chapter 9

GSPBOX - Utils

9.1 Tests graphs

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

Usage

```
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 martrix
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

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

Usage

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

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

9.1.3 GSP_ISDIRECTED - Check is the graph is directed

Usage

```
bool = gsp_isdirected(G);
bool = gsp_isdirected(W);
```

Input parameters

G	Graph structure or square matrix

Output parameters

bool	Boolean

Description

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

9.1.4 GSP_CHECK_WEIGHTS - Check a weight matrix

Usage

```
a = gsp_check_weights( W );
```

Input parameters

W	Weight matrix

Output parameters

a	Warning code

Description

This function performs various test on the weight matrix W. It returns:

- 0 : Everything is ok
- 1 : The martrix contains inf values
- 2 : The diagonal is not 0
- 3 : The matrix is not square
- 4 : The matrix contains nan values

9.2 Norms

9.2.1 GSP_NORM_TV - TV norm on graph

Usage

```
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

9.2.2 GSP_NORM_TIK - Squared L2 norm of the gradient on graph

Usage

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

Input parameters

G	Graph structure (or symetric positive matrix)
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.
This function can also be used for general symetric positive matrices

9.2.3 GSP_NORM_L1_FILTERBANK - Compute the l2 norm of the analysis coefficients

9.2.4 GSP_NORM_L2_FILTERBANK - Compute the l2 norm of the analysis coefficients

Usage

```
n = gsp_norm_l2_filterbank(G, W, x);
```

Input parameters

G	Graph structure
W	Filterbank (cell array of functions)
x	coefficients
param	structure of optional paramter

Output parameters

n	L2 norm

Description

`gsp_norm_l2_filterbank(G, W, x, param)` computes:

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

param is a Matlab structure containing the following fields:

- *param.A* : Forward operator (default: Id).
- *param.y* : measurements (default: 0).

9.2.5 GSP_NORM_TIG - Compute the norm of the tig of a frame

Usage

```
gsp_norm_tig( G,g );
gsp_norm_tig( G,g, exact );
```

Input parameters

G	Graph
g	filterbank
exact	Exact method (default 1)
M	Order for the approximation (default 50)

Output parameters

ntig	norm of tig

Description

This function compute the norm of all atoms of the filterbank g.

If *exact* is set to one, you can compute the norm chunk by chunk by setting *M* to the size of the desired chunk. If *M* is -1, then parpool is used.

9.3 Distance

9.3.1 GSP_RESISTANCE_DISTANCES - : Compute the resitance distances of a graph

Usage

```
rd = gsp_resistance_distances(G);
rd = gsp_resistance_distances(L);
```

Input parameters

G	Graph structure or Laplacian matrix (L)
param	optional parameters

Output parameters

rd	distance matrix

Description

This function compute the resistance distances between all vertices in a graph. The distance between two nodes is defined as the inverse of the weight matrix. For example the distance matrix:

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

corresponds to the weight matrix:

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

The function will compute the resistance distance following Kirshoff's law. In the our example it is:

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

In matlab, you can reproduce this example using:

```
% 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)
% Resitance 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:

```
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: [8]

9.3.2 GSP_DISTANZ - calculates the distances between all vectors in X and Y

Usage

```
D = gsp_distanz(X, Y);
```

Input parameters

X	matrix with col vectors
Y	matrix with col vectors (default == x)
P	distance matrix (default Identity)

Output parameters

D	distance matrix, not squared

Description

This code computes the following

$$D = \sqrt{(X-Y)^T P (X-Y)}$$

for all vectors in X an Y!

This code is not optimized for memory, but for speed because it uses no loops.

9.3.3 GSP_NN_DISTANZ - Compute the nearest neighboor distances

Usage

```
: [indx, indy, dist] = gsp_nn_distanz( X1 );
[indx, indy, dist] = gsp_nn_distanz( X1, X2 );
[indx, indy, dist] = gsp_nn_distanz( X1, X2, param );
[indx, indy, dist, Xo1, Xo2, epsilon] = gsp_nn_distanz(...)
```

Input parameters

X1	Input points 1
X2	Input points 2
param	Structure of optional parameters

Output parameters

indx	Indices over x
indy	Indices over y
dist	Distances
Xo1	Points 1 after rescaling
Xo2	Points 2 after rescaling
epsilon	Radius of the ball (if the ball is used!)

Description

This function computes the nearest neighboors of Xin.

Additional parameters

- *param.type* : ['knn', 'radius'] - the type of graph (default 'knn')
- *param.use_flann* : [0, 1] - use the FLANN library (default 0)
- *param.use_full* : [0, 1] - Compute the full distance matrix and then sparsify it (default 0)
- *param.flan_checks*: int - Number of checks for FLANN (default 256) the higher the more precise, but the slower. Please consider the following values: a) 32 not precise and fast, b) precise enought and still fast, c) 4096 precise and may be slow.
- *param.nb_cores* : int - Number of cores for FLANN (default 1)
- *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
- *param.use_l1* : [0, 1] - use the l1 distance

9.3.4 GSP_RMSE_MV - Compute columnwise RMSE ignoring missing values (NaN's)

Usage

```
C = gsp_rmse_mv(X);
C = gsp_rmse_mv(X, param);
[C, N_obs] = gsp_rmse_mv(...);
```

Input parameters

X	Data matrix
param	Parameter

Output parameters

C	RMSE
N_obs	number of commonly observed values

Description

The RMSE between two different columns will only take into account the common observed values.

C(i,j) = ||xi-xj||/sqrt(N), where xi and xj only contain the elements that are commonly observed in both and N is the number of these elements.

If an element is equal to NaN, it is not taken into account.

N_obs gives the number of commonly observed values for all pairs of columns.

param is an optional structure of argument containing the following fields: * *param.verbose*: Verbosity level of the function (default 0)

TODO: write fast implementation for case with no missing values!! like corr of matlab does

9.3.5 GSP_HOP_DISTANZ - Compute the hop distance between two node

Usage

```
d = gsp_hop_distanz(G,i,j);
```

Input parameters

G	Graph
i	node
j	node

Output parameters

d	hop distanz

Description

This code computes the hop distance between node i and node j. It uses a naive greedy algorithm and has to be improved.

9.4 Approximation

9.4.1 GSP_CHEBY_COEFF - : Compute Chebyshev coefficients for a filterbank

Usage

```
c = gsp_cheby_coeff(G, filter, m, N);
c = gsp_cheby_coeff(G, filter, m);
c = gsp_cheby_coeff(G, filter);
```

Input parameters

G	graph structure or range of application
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)
param	optional parameter

Output parameters

c	matrix of Chebyshev coefficients

Description

This function compute the Chebyshef coefficients for all the filter contained in the cell array filter. The coefficient are returned in a matrix. Every collumn correspond to a filter. The coefficients are ordered such that c(j+1) is j'th Chebyshev coefficient

param contain only one field param.verbose to controle the verbosity.

Example:

```
Nf = 4;
G = gsp_sensor(100);
G = gsp_estimate_lmax(G);
g = gsp_design_meyer(G, Nf);
c = gsp_cheby_coeff(G, g);
```

This function is inspired by the sgwt_toolbox

9.4.2 GSP_CHEBY_OP - : Chebyshev polynomial of graph Laplacian applied to vector

Usage

```
r = gsp_cheby_op(G, c, signal)
```

Input parameters

G	Graph structure
c	Chebyshef 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:

```
r(1) = gsp_cheby_op(G, c(:,1), signal);
r(2) = gsp_cheby_op(G, c(:,2), signal);
...
```

but is more efficient as the Chebyshev polynomials of G.L applied to *signal* can be computed once and shared.

The output *r* is a matrix with each column corresponding to a filter.

param contain only one field param.verbose to controle the verbosity.

Example:

```
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

9.4.3 GSP_CHEBY_EVAL - Evaluate chebyshev polynomial

Usage

```
r = gsp_cheby_eval(x,c,arange)
```

Input parameters

x	Points to evaluate the polynomial
c	Chebyshef coefficients
arrange	arange (range to evaluate the polynomial)

Output parameters

r	Result

Description

In this function, *arrange* has to be [0, lmax]!

9.5 Graph operations

9.5.1 GSP_ADJ2VEC - Prepare the graph for the gradient computation

Usage

```
[G] = gsp_adj2vec(G)
```

Input parameters

G	Graph structure

Output parameters

G	Graph structure

Description

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

9.5.2 GSP_COMPUTE_FOURIER_BASIS - Compute the fourier basis of the graph G

Usage

```
G = gsp_compute_fourier_basis(G);
G = gsp_compute_fourier_basis(G,param);
```

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

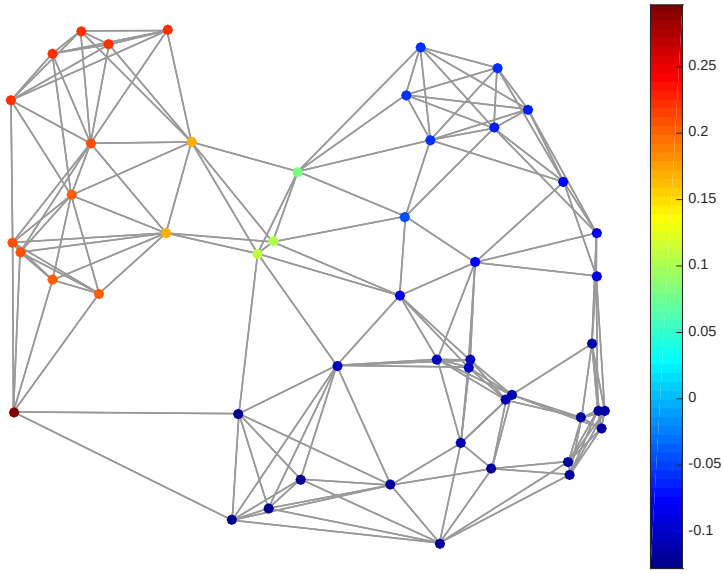
'gsp_compute_fourier_basis(G)' computes a full eigendecomposition of the graph Laplacian G.L:

$$\mathcal{L} = U \Lambda U^*$$

where Λ 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 *lmax* and *G.mu* the coherence of the Fourier basis into the structure *G*.

Example:

```
N = 50;
G = gsp_sensor(N);
G = gsp_compute_fourier_basis(G);
c
```



References: [3]

9.5.3 GSP_CREATE_LAPLACIAN - create the graph laplacian of the graph G

Usage

```
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. For undirected graph, the following type are availlable:

- *combinatorial*: Non normalized laplacian. This is the default.

$$L = D - W$$

- *normalized*: Normalized laplacian .. L_n = I - D^-0.5 W D^-0.5

$$L_n = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

- *none*: No laplacian

And for directed graph, the following types are availlable.

- *combinatorial* : Non normalized laplacian. This is the default

$$L = \frac{1}{2}(D^+ + D^- - W - W^*)$$

- *chung*: Normalized laplacian with the Perron eigenvector

$$L_{cn} = I - \frac{\Pi^{\frac{1}{2}} P \Pi^{-\frac{1}{2}} + \Pi^{-\frac{1}{2}} P^* \Pi^{\frac{1}{2}}}{2}$$

- *chung-non-normalized*: Experimental version of non normalized Chung laplacian .. L_c = Pi^0.5 L_cn Pi^0.5

$$L_c = \Pi^{\frac{1}{2}} L_{cn} \Pi^{\frac{1}{2}}$$

References: [2]

9.5.4 GSP_ESTIMATE_LMAX - estimates the maximum laplacian eigenvalue

Usage

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

Description

Inputs parameters: G : Graph structure (or cell array of graph structure)

Outputs parameters: G : Graph structure (or cell array of graph structure)

This function will compute an approximation of the maximum laplacian eigenvalue and fill it in the field *G.lmax*

Runs Arnoldi algorithm with a large tolerance, then increases calculated maximum eigenvalue by 1 percent. For much of the gspbox machinery, we need to approximate the wavelet kernels on an interval that contains the spectrum of L. The only cost of using a larger interval is that the polynomial approximation over the larger interval may be a slightly worse approxmation on the actual spectrum. As this is a very mild effect, it is not likely necessary to obtain very tight bonds on the spectrum of L

This function is inspired by the sgwt_toolbox

9.5.5 GSP_GRAPH_SPARSIFY - sparsify a graph using Spielman-Srivastava algorithm

Usage

```
Gnew = gsp_graph_sparsify(G,epsilon);
```

Input parameters

G	Graph structure or Laplacian matrix
epsilon	Sparsification parameter

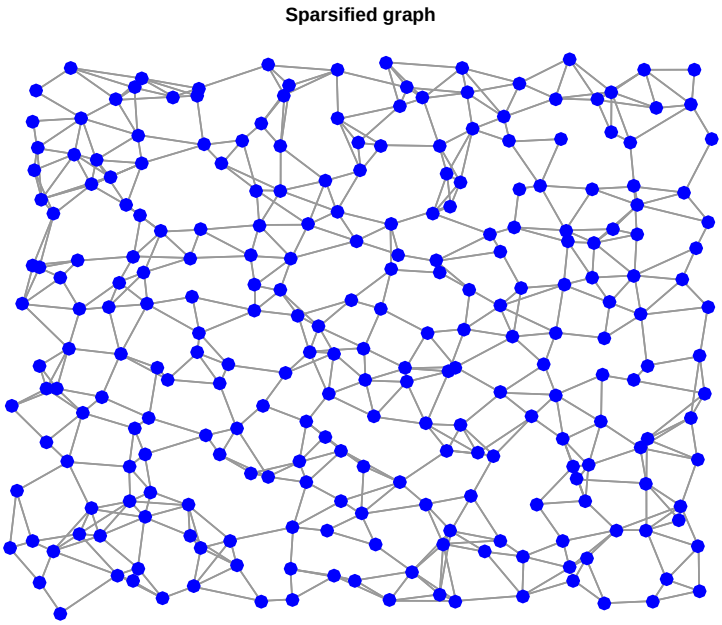
Description

Ouput 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:

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



References: [16], [11], [12]

9.5.6 GSP_SYMMETRIZE - symmetrize a matrix

Usage

```
W = gsp_symmetrize(W)
W = gsp_symmetrize(W, type)
```

Input parameters

W	square matrix
type	type of symmetrization (default 'full')

Output parameters

W	symmetrized matrix

Description

The available symmetrization types are: * 'average' : average of W and W^T (default) * 'full' : copy the missing entries * 'none' : nothing is done (the matrix might stay unsymmetric!)

9.6 Others

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

Usage

```
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 matix 1 2 3 4

becomes 1 1 2 2 3 3 4 4

9.6.2 GSP_CLASSIC2GRAPH_EIG_ORDER - Compute the graph eigen value ordering

Usage

```
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.

9.6.3 GSP_RESET_SEED - Reset the seed of the random number generator

Usage

```
gsp_reset_seed(n);
```

Input parameters

n	seed

Description

Ouptut parameters: none

This function resets the seed

9.6.4 GSP_PLOTFIG - Ploting figures with optimal size for paper

Usage

```
gsp_plotfig(save_name);
gsp_plotfig(save_name,param);
```

Input parameters

save_name	name to save the figure
param	optional parameters

Description

param a Matlab structure containing the following fields:

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

9.6.5 GSP_POINT2DCDF - points to discrete continuous density function

Usage

```
: [x,y] = gsp_point2dcdf(v);
```

Input parameters

v	vector of sample (1 dimention)

Output parameters

x	coordinate along x
y	coordinate along y

Description

This function compute a discrete continuous density function from a sample list.
Example:

```
gsp_point2dcdf([0 3 4 2 2 0 1 0 1 1 2])
```

This code produces the following output:

```
ans =

     0
     1
     2
     3
     4
```

9.6.6 GSP_DDF2DCDF - Discrete density function to dicrete cumulative density function

Usage

```
: [y] = gsp_ddf2dcdf(v);
[y,x] = gsp_ddf2dcdf(v,x);
```

Input parameters

v	Discrete density function
x	Axis value (Optional)

Output parameters

y	Discrete cumulative ensity function
x	Axis value (Same as input)

Description

This function goes from discrete density function to discrete cumulative density function.

Bibliography

[1] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, pages 585–591, 2001.

[2] F. Chung. Laplacians and the cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 2005.

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

[4] 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.

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

[6] 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.

[7] V. Kalofolias. How to learn a graph from smooth signals. Technical report, AISTATS 2016: proceedings at Journal of Machine Learning Research (JMLR)., 2016.

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

[9] N. Perraudin, J. Paratte, D. Shuman, V. Kalofolias, P. Vandergheynst, and D. K. Hammond. GSPBOX: A toolbox for signal processing on graphs. *ArXiv e-prints*, Aug. 2014.

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

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

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

[13] L. K. Saul and S. T. Roweis. An introduction to locally linear embedding. *unpublished. Available at: http://www. cs. toronto. edu/~ roweis/lle/publications. html*, 2000.

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

[15] 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.

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

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

[18] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[19] 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.