Dolomites Research Notes on Approximation

Special Issue SA2022 – *Software for Approximation 2022*, Volume 15 · 2022 · Pages 25–34

GBFPUM - A MATLAB Package for Partition of Unity Based Signal Interpolation and Approximation on Graphs*

Roberto Cavoretto[a,c] · Alessandra De Rossi[a,c] · Wolfgang Erb[b,c]

Abstract

This is a manual for the software package GBFPUM, a MATLAB toolbox for the generation of a partition of unity on graphs and their application as an interpolation and approximation tool for graph signals. GBFPUM combines local kernel approximation based on graph basis functions (GBFs) with a partition of unity method (PUM) in order to obtain a low-cost global interpolation or classification scheme for large graphs. We give a detailed description of all the routines implemented in the MATLAB package and show how the code can be used to interpolate graph signals in concrete examples.

1 Introduction

Partition of unity methods (PUMs) on graphs are simple and highly adaptive auxiliary tools to decompose signals on large graphs into smaller pieces on overlapping subdomains. In this way, PUMs allow to perform signal processing tasks as interpolation, approximation or spectral filtering locally on smaller subdomains and then to merge the single pieces to a global result (see Figure 1 for a simple illustration of this procedure). This split and merge procedure leads generally to a considerably lower computational cost than applying a signal processing scheme on a global scale.

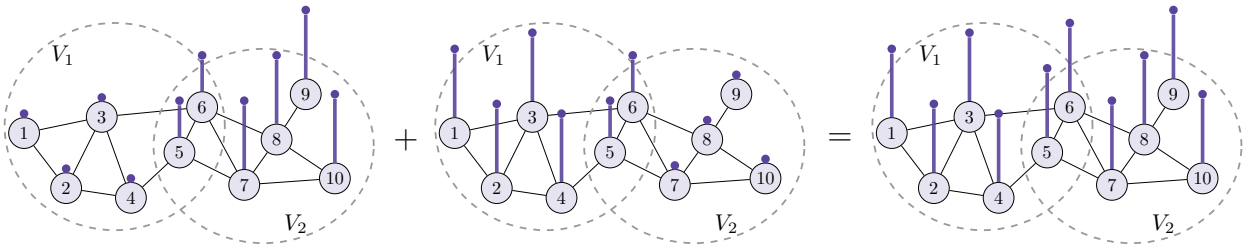

Figure 1: Schematic illustration of a partition of unity on a graph.

In a previous paper [4], we investigated how a partition of unity can be generated efficiently on graphs and how a PUM can be combined with a local graph basis function (GBF) approximation in order to obtain a low-cost global interpolation or approximation scheme. In this manual, we present the MATLAB package GBFPUM that was developed and implemented to test this new scheme. We give a detailed description of this software package and show how the single routines can be called inside MATLAB to perform signal processing steps on graphs. In particular, we will describe how the functions of the package can be used to generate a partition of unity on a graph and how signal approximation and interpolation with GBFs are implemented and combined with PUMs. This software is free and can be downloaded from the GitHub repository

https://github.com/WolfgangErb/GBFPUM.

The outline of this manual is as follows. In Section 2, we first show how the package is working based on a simple hands-on interpolation example on the Minnesota graph. We then give a brief overview on the theoretic background of the package, i.e., we recapitulate the basic notations (Section 3.1), we give an introduction to kernel-based interpolation (Section 3.2), and we show how PUMs can be generated on graphs (Section 3.3). The principal algorithm of the combined GBFPUM interpolation scheme is then summarized in Section 4. Finally, in Section 5, all the routines of the software are presented and explained.

2 How to use the toolbox: a simple example on the Minnesota graph

To see how this MATLAB package works, we apply the toolbox to a simple interpolation problem on the Minnesota graph. In this example, a graph signal f is interpolated based on a few samples of f on the graph.

*The preface of this special issue to which the article belongs is given in [3].

[a]Dipartimento di Matematica "Giuseppe Peano", Università di Torino, via Carlo Alberto 10, 10123 Torino, Italia
[b]Dipartimento di Matematica "Tullio Levi-Civita", Università di Padova, Via Trieste 63, 35121 Padova, Italia
[c]Member of the INdAM Research group GNCS

2.1 Loading the graph

The Minnesota graph [20] is a road network of the state of Minnesota consisting of $n = 2642$ vertices and 3304 edges. In the toolbox, this graph G is stored in the file `data_graphminnesota.m`. The graph signal f as well as a sequence of sampling nodes is stored in `data_f.m`. To load the data we can simply digit the following commands in MATLAB:

```
% Load Minnesota graph, test function and sampling data
load data_graphminnesota.mat
load data_f.mat
```

The loaded variable `G` contains all relevant information of the Minnesota graph and is a structure array with the following fields:

```
>> G =
  struct with fields:
     type: 'minnesota'
    nodes: [2642x2 double]
    edges: [3304x2 double]
        A: [2642x2642 double]
        N: 2642
      deg: [1x2642 double]
        L: [2642x2642 double]
```

Herein, `G.nodes` contains the real-world coordinates of the graph vertices, `G.edges` is a list of all the edges of G (stored as an ordered list of node indices) and `G.A` is the unweighted adjacency matrix of G. Further `G.deg` contains the degree of the single vertices of G and `G.L` the normalized graph Laplacian of the Minnesota graph.

The test signal f is stored in the variable `f`. It consists of the sum of the first 10 eigenvectors of the normalized graph Laplacian of G. The variable `idxW` in the file `data_f.mat` contains a preselected list of sampling nodes. We choose the first $N = 200$ nodes on this list as sampling nodes and extract the respective interpolation values from the graph signal `f`.

```
% Choose number of interpolation nodes and sampling data
N = 200;
idxW = idxW(1:N); y = f(idxW);
```

2.2 Generating the partition of unity

We next generate a partition of unity (PU) of the Minnesota graph. In our example we split the vertex set of G in $J = 6$ (`JJ=6` in the script) overlapping subdomains. The size of the overlapping is characterized by an augmentation parameter $r = 8$ (`RR=8` in the code).

```
% PU parameter selection
JJ = 6;               % Number of subdomains
RR = 8;               % Augmentation parameter for subdomains
```

The generation of the partition of unity itself is done in three steps. In the first step, the vertices of G are split in J parts by a J-center clustering procedure:

```
% 1a) Calculate J-center clustering of graph in JJ components
[idxcluster,idxQ] = GBF_Jcenters_greedy(G.A,JJ,idxW,idxW(1));
```

As an output we get the center nodes `idxQ` of the $J = 6$ clusters as well as the respective clustering `idxcluster` of the nodes. We then perform an augmentation procedure

```
% 1b) Augment the clusters to subdomains for PU
[idxdomain] = GBF_domainaugment(G.edges,idxcluster,RR);
```

by augmenting the single clusters with an augmentation radius $r = 8$. Finally, we get the partition of unity by the command

```
% 1c) Generate PU out of the single domains
[phi,idxWdomain,ydomain] = GBF_genPUM(G.edges,idxcluster,idxdomain,idxW,y);
```

The variable `phi` contains the calculated PU, while `idxWdomain` and `ydomain` encode the interpolation data on the single subdomains. All three routines `GBF_Jcenters_greedy.m`, `GBF_domainaugment.m` and `GBF_genPUM.m` for the generation of the PU will be described in more detail in Section 5.

2.3 Calculating the GBFPUM interpolant

For the calculation of the GBFPUM interpolant based on the partition of unity method, we require local interpolants on the J subdomains. The local interpolation is achieved via a kernel method based on so-called GBF kernels. As a GBF kernel on the subgraphs, we use in this example a variational spline kernel with the parameters $s = 2$ and $\epsilon = 0.001$ (for the details on this kernel see Section 3.2 below). In our MATLAB example these values are collected in the vector `alpha=[-2,0.001]`, with a change of sign in the first component. Setting the regularization parameter to `lambda=0` guarantees that we will obtain a GBFPUM interpolant of the data.

```
% Kernel parameters for local interpolation (variational splines)
kertype = 'varspline';
alpha = [—2,0.001];
lambda = 0;
```

The final GBFPUM interpolant is then calculated with the command

```
% 2)3)4) Use PU to calculate global GBFPUM interpolant
s = GBF_RLSGBFPUM(G.L,idxdomain,phi,idxWdomain,ydomain,kertype,alpha,lambda);
```

2.4 Evaluation of the GBFPUM interpolation

Figure 2: PUM interpolation based on local interpolants with the variational spline kernel ($\epsilon = 0.001$, $s = 2$) on $J = 6$ subdomains and $N = 200$ samples (black ringed nodes). The red ringed nodes denote the centers of the subdomains.

In Figure 2, the resulting PU on the Minnesota graph and the local kernel interpolants (the 6 images on the right), as well as the global GBFPUM interpolant (the image on the left) are graphically illustrated. We additionally evaluate the accuracy of the global GBFPUM scheme by computing the relative maximum absolute error (`rmaerr`) and the relative root mean square error (`rrmserr`).

```
% Compute errors
rmaerr = norm(s(:)—f,inf)/norm(f,inf);            % relative max absolute error
rrmserr = norm((s(:)—f))/max(f)/sqrt(length(f));  % relative rms error
% Print errors
fprintf('no. points\t rmaerr \t rrmserr \n')
fprintf('\t%4d\t %.3e\t %.3e\n ',N,rmaerr,rrmserr);
```

Using these commands, we get

```
no. points    rmaerr      rrmserr
     200      3.704e-01   3.912e-02
```

as a final printed result on the accuracy of the GBFPUM interpolation. The entire example presented in this section is provided in the MATLAB script `example.m` of the GBFPUM package.

3 Theoretic background of the package

3.1 Notation

In the next sections, we will recapitulate the theoretical background required to understand the GBFPUM package in a very schematic form. A complete introduction to graph signal interpolation with GBF kernels is given in [8, 9], while the entire description for the generation of a partition of unity on graphs is provided in [4]. A general overview on current aspects of the research field related to graph signal processing can be found in [17, 18, 21, 22]

In this work, we will consider simple and undirected graphs G as underlying domains, formally given as

$$G = (V, E, \mathbf{A}, \mathrm{d}),$$

where $V = \{\mathrm{v}_1, \ldots, \mathrm{v}_n\}$ denotes the n vertices of the graph, $E \subset V \times V$ the set of undirected edges, and $\mathbf{A}$ the symmetric adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ with non-negative entries

$$\begin{cases} \mathbf{A}_{i,j} > 0 & \text{if } (\mathrm{v}_i, \mathrm{v}_j) \in E, \\ \mathbf{A}_{i,j} = 0 & \text{otherwise.} \end{cases}$$

Further, we need a metric distance d between the graph nodes. A standard choice for the distance d is the length of the shortest path connecting two nodes. As further auxiliary matrices, we use the diagonal degree matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$ given by

$$\mathbf{D}_{i,j} = \begin{cases} \sum_{j=1}^{n} \mathbf{A}_{i,j} & \text{if } i = j, \\ 0 & \text{if } i \neq j, \end{cases}$$

and the normalized graph Laplacian $\mathbf{L} \in \mathbb{R}^{n\times n}$ defined as

$$\mathbf{L} = \mathbf{I}_n - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}, \quad \mathbf{L}_{i,j} := \begin{cases} 1 & \text{if } i = j, \\ -(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})_{i,j} & \text{if } i \neq j. \end{cases}$$

The matrix $\mathbf{L}$ is symmetric and positive semi-definite and encodes the principal topological information of the graph. The graph Laplacian $\mathbf{L}$ allows to define a Fourier transform on G in terms of its orthonormal eigenvectors $\{u_1, \ldots, u_n\}$.

In graph signal processing, the main research objects are graph signals. Graph signals are mappings $x : V \to \mathbb{R}$ (or $x : V \to \mathbb{C}$) on the vertex set V of the graph. As the vertices in V are ordered, we can represent every x as a vector

$$x = (x(\mathrm{v}_1), \ldots, x(\mathrm{v}_n))^\intercal \in \mathbb{R}^n \quad (\in \mathbb{C}^n).$$

The linear n-dimensional space of all graph signals is denoted by $\mathcal{L}(G)$. An example of a graph signal is plotted on the left hand side of Figure 3.

3.2 Interpolation on graphs

A general interpolation problem on a graph G can be formulated as follows: from the knowledge of the signal x on a subset

$$W = \{\mathrm{w}_1, \ldots, \mathrm{w}_N\} \subset V,$$

the goal is to reconstruct the signal x on the entire vertex set V based on particular assumptions on the underlying interpolation space. An example of such a problem is illustrated in Figure 3.

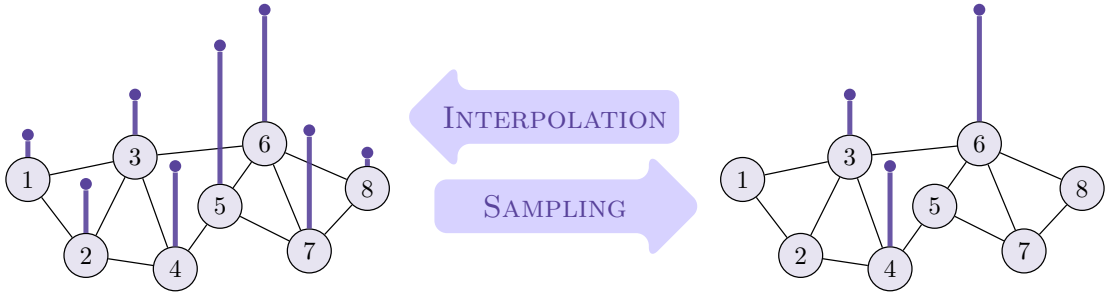

Figure 3: Interpolation and sampling of a graph signal on a graph with 8 vertices.

Particularly simple interpolation spaces can be generated by kernel functions $K : V \times V \to \mathbb{R}$ on the graph G. The kernel K allows to introduce a linear operator $\mathbf{K} : \mathcal{L}(G) \to \mathcal{L}(G)$ as

$$\mathbf{K}x(\mathrm{v}_i) = \sum_{j=1}^{n} K(\mathrm{v}_i, \mathrm{v}_j) x(\mathrm{v}_j).$$

This operator can also be expressed as a matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$ given by

$$\mathbf{K} = \begin{pmatrix} K(\mathrm{v}_1, \mathrm{v}_1) & K(\mathrm{v}_1, \mathrm{v}_2) & \ldots & K(\mathrm{v}_1, \mathrm{v}_n) \\ K(\mathrm{v}_2, \mathrm{v}_1) & K(\mathrm{v}_2, \mathrm{v}_2) & \ldots & K(\mathrm{v}_2, \mathrm{v}_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathrm{v}_n, \mathrm{v}_1) & K(\mathrm{v}_n, \mathrm{v}_2) & \ldots & K(\mathrm{v}_n, \mathrm{v}_n) \end{pmatrix}.$$

The kernel K is called positive definite, if the respective matrix $\mathbf{K}$ is symmetric and positive definite. The positive definiteness of the kernel K guarantees that the interpolation space

$$\mathcal{N}_{K,W} = \left\{ x \in \mathcal{L}(G) \mid x(\mathrm{v}) = \sum_{k=1}^{N} c_k K(\mathrm{v}, \mathrm{w}_k) \right\}$$

contains a unique interpolant $\mathrm{I}_W x$ such that

$$\mathrm{I}_W x(\mathrm{w}) = x(\mathrm{w}), \quad \text{for all } \mathrm{w} \in W.$$

Further, the coefficients $c = (c_1, \ldots, c_N)^\intercal$ of the interpolant can be calculated as

$$\underbrace{\begin{pmatrix} K(\mathrm{w}_1,\mathrm{w}_1) & K(\mathrm{w}_1,\mathrm{w}_2) & \ldots & K(\mathrm{w}_1,\mathrm{w}_N) \\ K(\mathrm{w}_2,\mathrm{w}_1) & K(\mathrm{w}_2,\mathrm{w}_2) & \ldots & K(\mathrm{w}_2,\mathrm{w}_N) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathrm{w}_N,\mathrm{w}_1) & K(\mathrm{w}_N,\mathrm{w}_2) & \ldots & K(\mathrm{w}_N,\mathrm{w}_N) \end{pmatrix}}_{\mathbf{K}_W} \underbrace{\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{pmatrix}}_{c} = \underbrace{\begin{pmatrix} x(\mathrm{w}_1) \\ x(\mathrm{w}_2) \\ \vdots \\ x(\mathrm{w}_N) \end{pmatrix}}_{x(W)}. \tag{1}$$

Concrete examples of positive definite kernels can be obtained by calculating matrix functions of the graph Laplacian $\mathbf{L}$ as, for instance, the matrix exponential of $\mathbf{L}$. Such kernels can also be interpreted as generalized translates of a positive definite GBF. GBFs and the respective kernels were thoroughly studied in [7, 8, 9]. Main examples of GBF kernels are:

- the diffusion kernel [14] based on the eigendecomposition $\mathbf{L} = \sum_{k=1}^{n} \lambda_k u_k u_k^\intercal$ of the graph Laplacian and given as

$$\mathbf{K} = e^{-t\mathbf{L}} = \sum_{k=1}^{n} e^{-t\lambda_k} u_k u_k^\intercal, \quad t \in \mathbb{R}.$$

- the variational splines [19, 23] based on the decomposition

$$\mathbf{K} = (\epsilon \mathbf{I}_n + \mathbf{L})^{-s} = \sum_{k=1}^{n} \frac{1}{(\epsilon + \lambda_k)^s} u_k u_k^\intercal, \quad \epsilon > 0,\ s > 0.$$

In Algorithm 1, we summarize the procedure to obtain a kernel-based interpolant on a graph. In our GBFPUM package, the generation of the interpolation matrix and the resolution of the linear system are implemented through the routines `GBF_genGBF.m` and `GBF_RLSGBF.m`.

Algorithm 1 Kernel-based interpolation on graphs

Input: Samples $x(\mathrm{w}_1), \ldots, x(\mathrm{w}_N)$ of a graph signal s, and a positive definite kernel K.

Calculate the N kernel columns $K(\cdot, \mathrm{w}_1), \ldots, K(\cdot, \mathrm{w}_N)$.

Solve the linear system of equations

$$\begin{pmatrix} K(\mathrm{w}_1,\mathrm{w}_1) & K(\mathrm{w}_1,\mathrm{w}_2) & \ldots & K(\mathrm{w}_1,\mathrm{w}_N) \\ K(\mathrm{w}_2,\mathrm{w}_1) & K(\mathrm{w}_2,\mathrm{w}_2) & \ldots & K(\mathrm{w}_2,\mathrm{w}_N) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathrm{w}_N,\mathrm{w}_1) & K(\mathrm{w}_N,\mathrm{w}_2) & \ldots & K(\mathrm{w}_N,\mathrm{w}_N) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} x(\mathrm{w}_1) \\ x(\mathrm{w}_2) \\ \vdots \\ x(\mathrm{w}_N) \end{pmatrix}.$$

Calculate and return the kernel interpolant

$$\mathrm{I}_W x(\mathrm{v}) = \sum_{k=1}^{N} c_k K(\mathrm{v}, \mathrm{w}_k).$$

We highlight that there are some time-critical parts in the calculation of the interpolant $\mathrm{I}_W x$:

1. For graphs (with an irregular structure) the N kernel columns $K(\cdot, \mathrm{w}_k)$, $k \in \{1, \ldots, N\}$, have to be calculated in a preliminary step. This might be cost-intensive for large graphs.
2. The resolution of the interpolation problem $\mathbf{K}_W c = x(W)$ in (1) might get computationally expensive if the number N of samples is large.

Regarding item 1, we remark that in general iterative methods can reduce the computational cost for the calculation of the vectors $K(\cdot, \mathrm{w}_k)$, in particular if the kernel is a matrix function of the graph Laplacian.

3.3 Partition of unity on graphs

To reduce the computational complexity of signal interpolation when the sampling sets are large the GBFPUM package uses a PUM to generate a global interpolant out of local ones. This idea is based on the following observations and experiences:

- On smaller parts of the graph, the computational cost to solve a local kernel-based interpolation problem is much smaller.
- For interpolation and approximation problems in the Euclidean space, the combination of radial basis function (RBF) interpolation with PUMs yields significantly sparser interpolation matrices, and, therefore, a considerable speed-up of calculations [5, 2, 1, 6], [10, Chapter 29], [15, 24], [25, Section 15.4]. Similar properties have been observed for meshfree Galerkin solvers of differential equations [12, 16]. Similar effects can therefore be expected for GBF kernels on graphs.

For this reason, we investigated in the previous paper [4] the idea to calculate local GBF interpolants on small subgraphs of G and to merge the local interpolants via a partition of unity. As a large number of segmentation algorithms is available on graphs, one inherent advantage of graph domains is that a partition of unity can be generated in a relatively simple way.

Definition 3.1. Let $\{V_j\}_{j=1}^J$ be a cover of the vertex set V. Then, a partition of unity $\{\varphi^{(j)}\}_{j=1}^J$ subordinate to this cover is a set of functions $\varphi^{(j)} \in \mathcal{L}(G)$, $j \in \{1,\dots,J\}$, such that

$$\operatorname{supp}(\varphi^{(j)}) \subseteq V_j, \quad \varphi^{(j)} \geq 0, \quad \text{and} \quad \sum_{j=1}^{J} \varphi^{(j)}(\mathrm{v}) = 1 \quad \text{for all } \mathrm{v} \in V.$$

A cover $\{V_j\}_{j=1}^J$ of the node set V and a partition of unity $\{\varphi^{(j)}\}_{j=1}^J$ subordinate to $\{V_j\}_{j=1}^J$ can be generated on a graph G by the following three operations:

1a) We use a (modified) J-center clustering algorithm to decompose V into J disjoint clusters C_j. For this, we use the graph geodesic as a distance between two nodes.

1b) We augment the clusters C_j with neighboring nodes and obtain a cover of overlapping subdomains V_j, $j \in \{1,\dots,J\}$.

1c) We use Shepard weight functions on the cover $\{V_j\}_{j=1}^J$ to obtain a desired partition of unity $\{\varphi^{(j)}\}_{j=1}^J$.

The procedure to obtain item 1a) is summarized in Algorithm 2. It corresponds to a modified version of the Gonzalez algorithm for J-center clustering [11, 13]. In the package GBFPUM it is implemented as MATLAB function `GBF_Jcenters_greedy.m`. Regarding the operation in item 1b), i.e., the augmentation procedure, we enlarge the clusters C_j to obtain overlapping subdomains V_j of the node set V. The augmentation procedure consists in adding all vertices to C_j that have a graph distance to C_j less than or equal to a distance $r \geq 0$. The details of this procedure are described in Algorithm 3 and implemented in the MATLAB routine `GBF_domainaugment.m`. We note that, although possible, it is not recommendable to use the disjoint clusters C_j, $j \in \{1,\dots,J\}$, directly as subdomains for the PU, as the topological information of the graph G and therefore also possible smoothness properties of signals get lost.

Algorithm 2 Greedy J-center clustering based on interpolation nodes

Input: The interpolation nodes $W = \{\mathrm{w}_1,\dots,\mathrm{w}_N\}$, a starting center $\mathrm{q}_1 \in W$ and the number J of partitions.

For $j = 2$ to J select a center $\mathrm{q}_j = \operatorname*{argmax}_{\mathrm{w}\in W} \min_{\mathrm{q}\in\{\mathrm{q}_1,\dots,\mathrm{q}_{j-1}\}} \mathrm{d}(\mathrm{q},\mathrm{w})$ farthest away from $Q_{j-1} = \{\mathrm{q}_1,\dots,\mathrm{q}_{j-1}\}$.

Calculate the J clusters

$$\mathrm{C}_j = \left\{ \mathrm{v} \in V \,:\, \mathrm{d}(\mathrm{q}_j,\mathrm{v}) = \min_{\mathrm{q}\in\{\mathrm{q}_1,\dots,\mathrm{q}_J\}} \mathrm{d}(\mathrm{q},\mathrm{v}) \right\}, \quad j \in \{1,\dots,J\}.$$

Return centers $\mathrm{Q}_J = \{\mathrm{q}_1,\dots,\mathrm{q}_J\}$ and clusters $\{\mathrm{C}_1,\dots,\mathrm{C}_J\}$.

Algorithm 3 Augmentation of clusters with neighbors of distance r

Input: A cluster C_j and an enlargement distance $r \geq 0$.

Add all nodes $\mathrm{v} \in V$ with a graph distance less than r to C_j, i.e.,

$$V_j = \mathrm{C}_j \cup \{\mathrm{v} \in V \mid \mathrm{d}(\mathrm{v},\mathrm{w}) \leq r,\ \mathrm{w} \in \mathrm{C}_j\}.$$

Return subdomain V_j.

Finally, in item 1c), the generation of the partition of unity with Shepard weights is performed. A simple construction principle to generate a partition of unity $\{\varphi^{(j)}\}_{j=1}^J$ is based on Shepard weights. If $\psi^{(j)}$, $j \in \{1,\dots,J\}$, denote nonnegative weight functions on V with $\operatorname{supp}(\psi^{(j)}) \subseteq V_j$ and $\sum_{j=1}^J \psi^{(j)} > 0$, then the functions

$$\varphi^{(j)}(\mathrm{v}) = \frac{\psi^{(j)}(\mathrm{v})}{\sum_{j=1}^J \psi^{(j)}(\mathrm{v})}, \quad \mathrm{v} \in V,$$

form a partition of unity subordinate to $\{V_j\}_{j=1}^J$. As an example, we consider the disjoint clusters C_j as subsets of the augmented domains V_j. Then, choosing $\psi^{(j)}(\mathrm{v}) = \mathbf{1}_{\mathrm{C}_j}(\mathrm{v})$, the corresponding partition of unity is given as

$$\varphi^{(j)}(\mathrm{v}) = \psi^{(j)}(\mathrm{v}) = \mathbf{1}_{\mathrm{C}_j}(\mathrm{v}).$$

In our MATLAB package GBFPUM the generation of the partition of unity is implemented in the routine `GBF_genPUM.m`.

4 GBFPUM interpolation on graphs

Combining GBF interpolation on the local subdomains V_j with a partition of unity on the graph, we obtain a global interpolation method for given sampling information on G. This global GBFPUM interpolation scheme is sketched in Algorithm 4. In our package, Algorithm 4 is implemented in terms of four routines. The MATLAB functions `GBF_Jcenters_greedy.m`, `GBF_domainaugment.m` and `GBF_genPUM.m` resolve the parts 1a)b)c) of Algorithm 4, respectively. The fourth routine `GBF_RLSGBFPUM.m` performs the final steps 2) 3) and 4) of the interpolation algorithm. Inside `GBF_RLSGBFPUM.m`, the subroutines `GBF_genGBF.m` and `GBF_RLSGBF.m` are used to solve the local interpolation problems in step 3).

Algorithm 4 GBFPUM interpolation on graphs

Input: (i) Samples $x(\mathrm{w}_1), \ldots, x(\mathrm{w}_N) \in \mathbb{R}$ at the interpolation nodes $W = \{\mathrm{w}_1, \ldots, \mathrm{w}_N\}$,
(ii) Number J of subdomains.

1.a) Use J-center clustering to decompose V into J disjoint clusters C_j.

1.b) Create J subdomains V_j by augmenting the clusters C_j into J overlapping subdomains $V_j = \{\mathrm{v}_{j_1}, \ldots, \mathrm{v}_{j_{n_j}}\}$ with n_j elements.

1.c) Generate partition of unity $\{\varphi^{(j)}\}_{j=1}^J$ subordinate to the cover $\{V_j\}_{j=1}^J$ such that

$$\mathrm{supp}(\varphi^{(j)}) \subseteq V_j, \ \varphi^{(j)} \geq 0, \text{ and } \sum_{j=1}^{J} \varphi^{(j)}(\mathrm{v}) = 1 \text{ for all } \mathrm{v} \in V.$$

2. Extract local graphs and local Laplacians $\mathbf{L}^{(j)}$ For all subdomains V_j, $j \in \{1, \ldots, J\}$, generate the subgraphs $G_j = (V_j, E_j)$, with the node sets V_j, the edges $E_j = \{(\mathrm{v}, \mathrm{w}) \in E \mid \mathrm{v}, \mathrm{w} \in V_j\}$ and the local Laplacian

$$\mathbf{L}^{(j)} = \begin{pmatrix} \mathbf{L}_{j_1,j_1} & \cdots & \mathbf{L}_{j_1,j_{n_j}} \\ \vdots & \ddots & \vdots \\ \mathbf{L}_{j_{n_j},j_1} & \cdots & \mathbf{L}_{j_{n_j},j_{n_j}} \end{pmatrix}.$$

If required, calculate the spectral decomposition $\mathbf{L}^{(j)} = \mathbf{U}^{(j)} \mathbf{M}_{\lambda^{(j)}} \mathbf{U}^{(j)\intercal}$ to define a local graph Fourier transform on the subgraphs G_j.

3.a) Construct a local GBF kernel $K^{(j)}(\mathrm{v}, \mathrm{w})$ on the subgraph G_j. One possibility is to use the variational spline kernel

$$\mathbf{K}^{(j)} = (\epsilon \mathbf{I}_{n_j} + \mathbf{L}^{(j)})^{-s} = \sum_{k=1}^{n_j} \frac{1}{(\epsilon + \lambda_k^{(j)})^s} u_k^{(j)} u_k^{(j)\intercal}, \quad \epsilon > -\lambda_1^{(j)}, \ s > 0.$$

3.b) For the sampling sets $W_j = W \cap V_j$ with $N_j \geq 1$ elements **solve**

$$\mathbf{K}_{W_j}^{(j)} c^{(j)} = x(W_j).$$

3.c) Calculate the local GBF interpolants $\mathrm{I}_{W_j} x$ on G_j:

$$\mathrm{I}_{W_j} x(\mathrm{v}) = \sum_{i=1}^{N_j} c_i^{(j)} K^{(j)}(\mathrm{v}, \mathrm{w}_{j_i}).$$

4. Calculate and return the **global kernel-PUM interpolant** on G given as

$$\mathrm{I}_W x(\mathrm{v}) = \sum_{j=1}^{J} \varphi^{(j)}(\mathrm{v}) \mathrm{I}_{W_j} x(\mathrm{v}).$$

5 The routines of the MATLAB package GBFPUM

In this final section, we present the single routines of the MATLAB package GBFPUM developed and implemented for partition of unity based kernel interpolation. The principle task of these routines is to perform the GBFPUM interpolation method presented in Algorithm 4. Overall, the package consists of six MATLAB functions, which can be downloaded freely from the GitHub repository

https://github.com/WolfgangErb/GBFPUM.

In this package, the three routines `GBF_Jcenters_greedy.m`, `GBF_domainaugment.m` and `GBF_genPUM.m` are used to generate the partition of unity on the graph. The routines `GBF_genGBF.m` and `GBF_RLSGBF.m` contain the code for the solution of the local interpolation problems. Finally, `GBF_RLSGBFPUM.m` calculates the global GBFPUM interpolant.

For the sake of brevity, we report in the following only the introductory part of the MATLAB functions. These (commented) explanatory lines contain information regarding the name of the file, the goal of the function, how it is used, and what the input and the output are.

Listing 1 (`GBF_Jcenters_greedy.m`) performs a reduced J-center clustering based on a greedy search technique as described in Algorithm 2. Determining the cluster centers and the cluster assignments is both carried out in an efficient way. This code recalls internally the function `distcenter.m` that calculates the graph geodesic distance of the graph nodes to the centers.

```
% File:    GBF_Jcenters_greedy.m
%
% Goal:    Performs a reduced J-center clustering based on greedy search:
%          Finds clusters based on a reduced greedy procedure
%          on the subset W (given by idxW) of the nodes
%
% Use:     [idxcluster,idxQ] = GBF_Jcenters_greedy(A,J,idxW,idxQin)
%
% Input:
%           A          = adjacency matrix
%           J          = number of clusters
%           idxW       = indices of subset on which the greedy search is performed
%           idxQin     = predefined input centers (optional)
%
% Output:
%           idxcluster = Jx1 cell vector with cluster assignments
%           idxQ       = Jx1 vector of cluster center indices
```

Listing 1: Implementation of a J-center clustering procedure based on a greedy search.

Listing 2 (`GBF_domainaugment.m`) is a MATLAB function that allows us to enlarge the graph clusters to obtain overlapping subdomains of the graph. It is an implementation of Algorithm 3.

```
% File:     GBF_domainaugment.m
%
% Goal:     Augments the graph cluster domains
%
% Use:      [idxsub] = GBF_domainaugment(edges,idxsub,RR)
%
% Input:
%           edges  = edges of the graph
%           idxsub = JJ subdomains of the graph
%           RR     = extension radius for augmentation
%
% Output:
%           idxsub = augmentend subdomains of the graph
```

Listing 2: Routine that augments the graph clusters to get overlapping subdomains.

Listing 3 (`GBF_genPUM.m`) generates a partition of unity subordinate to the cover of the graph provided by the previous routine `GBF_domainaugment.m`.

```
% File:    GBF_genPUM.m
%
% Goal:    Computes PU based on given graph clusters and domains
%
% Use:     [phi,idxWsub,ysub] = GBF_genPUM(edges,cluster,domain,idxW,y)
%
% Input:
%           edges       = edges of the graph
%           cluster     = JJ clusters of the graph
%           domain      = JJ augmented domains of the graph
%           idxW        = indices of the interpolation nodes
%           y           = interpolation values
%
% Output:
%           phi         = PU for subdomains
%           idxWsub     = indices of interpolation nodes in subdomains
%           ysub        = interpolation values in the subdomains
```

Listing 3: Creation of partition of unity for kernel-based interpolation on graphs.

Listing 4 (`GBF_genGBF.m`) generates the columns of a GBF kernel required for the GBF interpolation on the subgraphs. The GBF kernel implemented in this code for interpolation is the variational spline kernel.

```
% File:    GBF_genGBF.m
%
% Goal:    Computes K generalized translates of the GBF for the nodes in idxW
%
% Use:     bf = GBF_genGBF(L,idxW,type,alpha)
%          (reduced implementation: contains only variational splines)
%
% Input:
%          L     = NxN matrix - the sparse graph Laplacian
%          idxW  = K vector - the indices of the K interpolation nodes
%          type  = type of graph basis function
%          alpha = additional shape parameter
% Output:
%          bf    = NxK matrix with the K basis function vectors
```

Listing 4: Construction of a local GBF kernel on the subgraph of the partition of unity.

Listing 5 (`GBF_RLSGBF.m`) calculates a GBF regularized least squares solution for each subdomain of the partition of unity. By setting the parameter `lambda=0`, it computes local GBF interpolants on the subdomains.

```
% File:    GBF_RLSGBF.m
%
% Goal:    Computes the GBF regularized least squares solution s,
%          minimizing the functional
%                  1/K * sum_i (s(i) - y(i)) + lambda * |s|_K,
%          based on K sampling nodes in idxW.
%
% Use:     [s,c,Kf] = GBF_RLSGBF(bf,idxW,y,lambda)
%
% Input:
%          bf     = NxK matrix - the K graph basis vectors
%          idxW   = K vector - the indices of the K sampling nodes
%          y      = K vector - the sampling values at the K nodes
%          lambda = regularization parameter for RLS
% Output:
%          s     = N-vector - the RLS-GBF solution
%          c     = K-vector - the coefficients in the basis expansion
%          Kf    = interpolation matrix of the kernel
```

Listing 5: Solution of the local system and computation of the related GBF approximant on the subgraph.

Listing 6 (`GBF_RLSGBFPUM.m`) calculates a global GBFPUM approximation of a graph signal using local GBF least squares approximations and a partition of unity to merge the local approximants. By setting `lambda=0`, it computes a global GBFPUM interpolation of a graph signal.

```
% File:    GBF_RLSGBFPUM.m
%
% Goal:    Computes a global approximant of the graph signal based on local regularized least squares
%          GBF approximants and a partition of unity on the graph domain
%
% Use:     s = GBF_RLSGBFPUM(LL,idxdomain,phi,idxWdomain,ydomain,type,alpha,lambda)
%
% Input:
%          LL         = NxN matrix - graph Laplacian
%          idxdomain  = JJ subdomains of the PU
%          phi        = Partition of Unity
%          idxWdomain = indices of sampling nodes on subdomains
%          ydomain    = sampling values on subdomains
%          type       = type of kernel for local RLS
%          alpha      = kernel parameters
%          lambda     = regularization parameter for local RLS
% Output:
%          s         = N-vector - The RLS-GBFPUM solution
```

Listing 6: Calculation of the global GBFPUM interpolant based on a local GBF kernel and a partition of unity on the graph.

Acknowledgements

This work was supported by the INdAM – GNCS Project, code CUP_E55F22000270001, and by the 2020 projects "Models and numerical methods in approximation, in applied sciences and in life sciences" and "Mathematical methods in computational sciences" funded by the Dipartimento di Matematica "Giuseppe Peano" of the Università di Torino. This research has been accomplished within the RITA "Research ITalian network on Approximation" and the UMI Group TAA "Approximation Theory and Applications". The authors sincerely thank the anonymous referees for valuable comments and suggestions, which enabled to significantly improve the quality of this paper.

References

[1] R. Cavoretto. Adaptive radial basis function partition of unity interpolation: A bivariate algorithm for unstructured data. *J. Sci. Comput.*, 87:41, 2021.

[2] R. Cavoretto, A. De Rossi. Error indicators and refinement strategies for solving Poisson problems through a RBF partition of unity collocation scheme. *Appl. Math. Comput.*, 369:124824, 2020.

[3] R. Cavoretto, A. De Rossi. Software for Approximation 2022 (SA2022). *Dolomites Res. Notes Approx.*, Special Issue SA2022, 15:i–ii, 2022.

[4] R. Cavoretto, A. De Rossi, W. Erb. Partition of unity methods for signal processing on graphs. *J. Fourier Anal. Appl.*, 27:66, 2021.

[5] R. Cavoretto, S. De Marchi, A. De Rossi, E. Perracchione, G. Santin. Partition of unity interpolation using stable kernel-based techniques, *Appl. Numer. Math.*, 116:95–107, 2017.

[6] S. Cuomo, A. De Rossi, L. Rizzo, F. Sica. Reconstruction of volatility surfaces: A first computational study. Accepted (2022).

[7] S. Cuomo, W. Erb, G. Santin. Kernel-based models for influence maximization on graphs based on Gaussian process variance minimization. *arXiv:2103.01575* (2021)

[8] W. Erb. Graph signal interpolation with positive definite graph basis functions. *Appl. Comput. Harmon. Anal.*, 60:368–395, 2022.

[9] W. Erb. Semi-supervised learning on graphs with feature-augmented graph basis functions. *arXiv:2003.07646* (2020).

[10] G.E. Fasshauer. Meshfree Approximation Methods with MATLAB. World Scientific, Singapore, 2007.

[11] T.F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[12] M. Griebel, M.A. Schweitzer. A particle-partition of unity method for the solution of elliptic, parabolic and hyperbolic PDE. *SIAM J. Sci. Comput.*, 22(3):853–890, 2000.

[13] D.S. Hochbaum, D.B. Shmoys. A best possible heuristic for the k-center problem. *Math. Oper. Res.*, 10(2):180–184, 1985.

[14] R.I. Kondor, J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In Proc. of the 19th. Intern. Conf. on Machine Learning ICML02, 315–322, 2002.

[15] E. Larsson, V. Shcherbakov, A. Heryudono. A least squares radial basis function partition of unity method for solving PDEs. *SIAM J. Sci. Comput.*, 39(6):A2538–A2563, 2017.

[16] J.M. Melenk, I. Babuška. The partition of unity finite element method: Basic theory and applications. *Comput. Methods Appl. Mech. Engrg.*, 139(1-4):289–314, 1996.

[17] A. Ortega, P. Frossard, J. Kovacevic, J.M.F. Moura, P. Vandergheynst. Graph signal processing: Overview, challenges, and applications. In Proceedings of the IEEE, 106(5):808–828, 2018.

[18] I.Z. Pesenson. Sampling in Paley-Wiener spaces on combinatorial graphs. *Trans. Amer. Math. Soc.*, 360(10):5603–5627, 2008.

[19] I.Z. Pesenson. Variational splines and Paley-Wiener spaces on combinatorial graphs. *Constr. Approx.*, 29(1):1–21, 2009.

[20] R.A. Rossi, N.K. Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, *http://networkrepository.com*.

[21] D.I. Shuman, B. Ricaud, P. Vandergheynst. Vertex-frequency analysis on graphs. *Appl. Comput. Harm. Anal.*, 40(2):260–291, 2016.

[22] L. Stanković, L. Daković, E. Sejdić. Introduction to Graph Signal Processing. In Vertex-Frequency Analysis of Graph Signals, Springer, 3–108, 2019.

[23] J.P. Ward, F.J. Narcowich, J.D. Ward. Interpolating splines on graphs for data science applications. *Appl. Comput. Harm. Anal.*, 49:540–557, 2020.

[24] H. Wendland. Fast evaluation of radial basis functions: Methods based on partition of unity. In: C.K. Chui et al. (Eds.), Approximation Theory X: Wavelets, Splines, and Applications, 473–483, 2002.

[25] H. Wendland. Scattered Data Approximation. Cambridge University Press, Cambridge, 2005.