Semi-supervised Learning for Multilayer Graphs Using Diffuse Interface Methods and Fast Matrix Vector Products

Kai Bergermann*, Martin Stoll†, and Toni Volkmer‡

Abstract. We generalize a graph-based multiclass semi-supervised classification technique based on diffuse interface methods to multilayer graphs allowing for a very high number of layers. Besides the treatment of various applications with an inherent multilayer structure, we present a very flexible approach that interprets high-dimensional data in a low-dimensional multilayer graph representation. Highly efficient numerical methods involving the spectral decomposition of the corresponding differential graph operators as well as fast matrix-vector products based on the nonequispaced fast Fourier transform (NFFT) enable the rapid treatment of very large data sets and make the algorithm independent of specialized hardware as well as scalable to even larger problems. We test the performance of our method on a variety of large and high-dimensional data sets.

Key words. power mean Laplacian, multiclass semi-supervised learning, graph Laplacian, fast eigenpair computation, nonequispaced fast Fourier transform

AMS subject classifications. 68R10, 05C50, 65F15, 65T50, 68T05, 62H30

1. Introduction.

Complex networks have become an indispensable tool in the modeling of phenomena ranging from neurobiology to statistical physics [49]. As most sets of items interact in a variety of relationships, multilayer graphs have emerged as a flexible tool to reflect these complex interactions, see also [26, 6]. The power of these networks to model phenomena from social interactions to energy networks has greatly fueled research for a better understanding of the network properties and also to tailor numerical methods to incorporate their mathematical structures.

In this paper, we propose a technique for semi-supervised learning [58, 59] on multilayer graphs where only a small portion of the data is pre-labeled. In order to classify the remaining unlabeled nodes, we rely on a diffuse interface approach which was first introduced in [5]. This method by Bertozzi and Flenner borrows from well-known results that have mainly been studied in the context of phase separation phenomena in materials science [42, 56, 57, 4]. The crucial formulation on a graph then requires the use of a discrete differential operator, namely the graph Laplacian [53, 11]. Based on its properties and additional terms in the loss function, this method has shown great potential for different applications. This technique has recently been extended to various different scenarios including multiclass segmentation [18], the use of an MBO scheme [31, 52] and of non-smooth potentials [7], application to signed networks [36] and many more. Its extension and efficient implementation for the multilayer case is at the heart of this paper.

*Technische Universität Chemnitz, Department of Mathematics, Chair of Numerics of Partial Differential Equations, 09107 Chemnitz, Germany, (kai.bergermann@mathematik.tu-chemnitz.de)

†Technische Universität Chemnitz, Department of Mathematics, Chair of Scientific Computing, 09107 Chemnitz, Germany, (martin.stoll@mathematik.tu-chemnitz.de)

‡Technische Universität Chemnitz, Department of Mathematics, Chair of Scientific Computing, 09107 Chemnitz, Germany, (toni.volkmer@mathematik.tu-chemnitz.de)

Of course, we will require a corresponding differential operator, and we rely on the formulation of graph Laplacians for multilayer graphs. In particular, the power mean Laplacian proposed in [37, 38], which effectively combines the crucial information of the different graph layers, will provide an essential ingredient to our algorithm. The use of this operator for semi-supervised learning was proposed in [38], and we compare the numerical results of our scheme to the method introduced in [37].

The complexity of the network is reflected in the connectivity of the nodes. But since in graph-based image processing the resulting graph is often fully connected, the sparsity of the network cannot be exploited. Additionally, the dimensionality of the network is often vast, making it a crucial task to be able to work with the resulting matrices in an efficient manner [48]. Many techniques in machine learning rely on the spectral information of the graph Laplacian in question, see also [9, 27, 34, 39, 40] in addition to the ones mentioned before. The diffuse interface method considered in this paper combines the favorable properties of the eigeninformation of the graph Laplacian with a nonlinear function pushing the graph nodes into their corresponding classes. The computation of both eigenvalues and eigenvectors heavily relies on the efficiency of the matrix-vector products with the graph Laplacian. For moderate dimensions of the feature space, fast summation techniques [25, 45, 46, 1, 41] show great potential to implicitly realize the matrix-vector multiplication in $\mathcal{O}(n)$ where n is the number of nodes in the graph. These techniques are often based on arguments from Fourier analysis.

In order to also take advantage of fast summation techniques for high-dimensional data, we present a feature grouping approach that splits the feature space into several low-dimensional subspaces. We interpret each feature subspace as a layer in a multilayer graph giving rise to a novel class of multilayer graphs. Each subspace can then take advantage of the fast matrix-vector products described before. This highly scalable technique not only enables us to efficiently classify large data sets like huge images that would normally produce enormous graph Laplacian matrices but also allows for a high feature space dimensionality which can be found in various applications, including for example hyperspectral imaging.

We achieve the outlined tasks in this paper as follows. First, we give the necessary definitions for both graphs and multilayer graphs including the discrete differential operators in Section 2, and we comment on eigenpairs computation approaches in Section 3. The fast summation technique based on the nonequispaced fast Fourier transform (NFFT), which is utilized in this paper, is introduced in Section 4. We review the graph Allen-Cahn equation for (single layer) graphs in Section 5 focusing on the multiclass case. Its extension to the multilayer case is given in Section 6. In Section 7 we propose the reformulation of a graph-based problem with a high-dimensional feature space as a multilayer graph via a feature grouping technique, allowing for the application of the fastsum summation introduced in Section 4. Finally, Section 8 applies the derived methods to the classification of various data sets. Subsection 8.1 starts with illustrating some persuasive features of the power mean Laplacian by classifying data sets generated by the stochastic block model [23]. Next, several very high-dimensional real world data sets are considered with an inherent multilayer structure in Subsection 8.2, and large parts of the results obtained in [38] are further improved. Subsection 8.3 then presents the segmentation of a 10 megapixel image, taking full advantage of fast matrix-vector products, and shows that our method generalizes very well to similar unseen images. Finally, in Subsection 8.4 all methods presented in this paper join forces in order to efficiently treat the

both large and high-dimensional hyperspectral Pavia center data set [43] achieving excellent classification accuracies while working directly on the unfiltered raw data without requiring problem-tailored hard- or software architectures. In particular, all numerical experiments can be run on a laptop computer.

2. Graphs and multilayer graphs. First, we briefly introduce the notation of graphs and graph Laplacians. For more details and properties, we refer to [39, 40, 11].

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of vertices $x_i \in \mathcal{V}$, $|\mathcal{V}| = n$, and edges $e \in \mathcal{E} \subset \mathcal{V} \times \mathcal{V}$, where an edge e connects any pair of vertices $x_i, x_j \in \mathcal{V}$. In this paper, we do not allow self-edges, i.e., we require $(x_i, x_i) \notin \mathcal{E}\ \forall i$. In particular, we use weighted graphs $\mathcal{G}$ with a weight function $w\colon \mathcal{V} \times \mathcal{V} \to \mathbb{R}_{\geq 0}$. A value $w(x_i, x_j) > 0$ indicates that two vertices $x_i, x_j \in \mathcal{V}$ are connected by an edge and $w(x_i, x_j) = 0$ means $(x_i, x_j) \notin \mathcal{E}$. The weight matrix $\boldsymbol{W} := (w(x_i, x_j))_{i,j=1}^n \in \mathbb{R}_{\geq 0}^{n \times n}$ collects the weight information. Here, we only consider undirected graphs, which yields $w(x_i, x_j) = w(x_j, x_i)$, leading to a symmetric weight matrix $\boldsymbol{W}$.

Based on the weight function w, the degree $\deg(x_i)$ of a node x_i can be defined as

$$\deg(x_i) := \sum_{x_j \in \mathcal{V}} w(x_i, x_j)$$

and the degree matrix $\boldsymbol{D} \in \mathbb{R}^{n \times n}$ as the diagonal matrix

$$\boldsymbol{D} := \operatorname{diag}\left(\deg(x_1), \ldots, \deg(x_n)\right) = \operatorname{diag}\left(\boldsymbol{W} \cdot \boldsymbol{1}\right), \qquad \boldsymbol{1} := (1, 1, \ldots, 1)^\top \in \mathbb{R}^n.$$

Then, the (unnormalized symmetric) graph Laplacian $\boldsymbol{L} \in \mathbb{R}^{n \times n}$ is defined as $\boldsymbol{L} := \boldsymbol{D} - \boldsymbol{W}$ and the symmetric normalized graph Laplacian as

$$\boldsymbol{L}_{\text{sym}} := \boldsymbol{D}^{-1/2} \boldsymbol{L} \boldsymbol{D}^{-1/2} = \boldsymbol{I} - \boldsymbol{D}^{-1/2} \boldsymbol{W} \boldsymbol{D}^{-1/2}.$$

These graph Laplacians play a key role in many graph-based learning techniques, especially in classification tasks. For various supervised, semi-supervised and unsupervised learning tasks, repeated matrix-vector multiplications with the graph Laplacians or the weight matrix $\boldsymbol{W}$, respectively, are required, e.g., when computing eigenpairs of the graph Laplacian via a conjugate gradient type method.

In order to able to perform these tasks within a reasonable time frame for a huge number n of nodes, the computation time for such matrix-vector products has to be reasonable. This can be achieved if the weight matrix $\boldsymbol{W}$ fulfills one of the following properties:

(i) The weight matrix $\boldsymbol{W} \in \mathbb{R}_{\geq 0}^{n \times n}$ is of general structure, leading to $\mathcal{O}(n^2)$ computation time, but the number n of nodes is not too large.

(ii) The weight matrix $\boldsymbol{W}$ is sparse, e.g., containing only $\mathcal{O}(n)$ non-zero entries and resulting in $\mathcal{O}(n)$ computation time for a matrix-vector multiplication.

(iii) A feature vector $\boldsymbol{x}_i \in \mathbb{R}^d$ is associated with each node $x_i \in \mathcal{V}$ of the graph $\mathcal{G}$, and the weight function w is given by a suitable kernel function, $w(x_i, x_j) = K(\boldsymbol{x}_i - \boldsymbol{x}_j)$, such that the matrix-vector multiplication with $\boldsymbol{W}$ can still be realized in $\mathcal{O}(n)$ computation time via a highly efficient algorithm although $\boldsymbol{W}$ may be densely populated. This case will be discussed in more detail in Section 4.

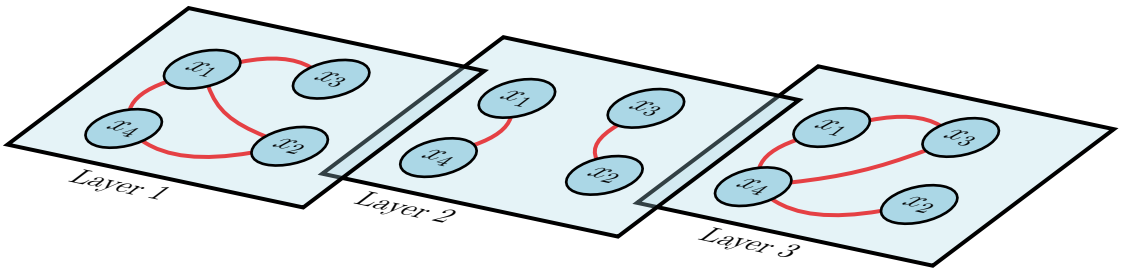

Figure 1. *Example of a multilayer network with* 3 *layers and* 4 *nodes*

As many applications produce data graphs with an inherent multilayer structure, describing e.g. different types of interactions between nodes, time series data or data sets combining data from independent sources [26], we next consider multilayer graphs, which consist of $T \in \mathbb{N}$ graph layers, see Figure 1 for an example. Now, each layer $\mathcal{G}^{(t)}, t = 1, \dots, T$, is a graph based on the same vertex set $\mathcal{V}$, $|\mathcal{V}| = n$. The edge sets $\mathcal{E}^{(t)} \subset \mathcal{V} \times \mathcal{V}$, however, are typically different across the layers and correspondingly also the weight matrices $\boldsymbol{W}^{(t)} \in \mathbb{R}^{n \times n}_{\geq 0}$.

Following [37], the symmetric normalized graph Laplacian $\boldsymbol{L}_{\text{sym}}$ of a graph $\mathcal{G}$ can then be generalized to multilayer graphs. $\boldsymbol{L}_{\text{sym}}$ is now defined for each layer t separately as $\boldsymbol{L}^{(t)}_{\text{sym}}$. To merge the information of all graph layers into one Laplacian, the power mean Laplacian M_p, $p \in \mathbb{R}$, was introduced in [37]. The definition is based on the matrix power mean of symmetric positive definite matrices $\boldsymbol{A}_1, \dots, \boldsymbol{A}_T$, which is given by

$$M_p(\boldsymbol{A}^{(1)}, \dots, \boldsymbol{A}^{(T)}) = \left(\frac{1}{T} \sum_{t=1}^{T} (\boldsymbol{A}^{(t)})^p \right)^{1/p}$$

for an exponent $p \in \mathbb{R}$, where $\boldsymbol{A}^{1/p}$ is the unique positive definite solution of the matrix equation $\boldsymbol{X}^p = \boldsymbol{A}$. This definition can be extended to positive semi-definite matrices $\boldsymbol{A}_1, \dots, \boldsymbol{A}_T$ for exponent $p > 0$. Correspondingly, for $p > 0$, this matrix power mean can be directly applied to symmetric graph Laplacians $\boldsymbol{L}^{(t)}_{\text{sym}}$, where all eigenvalues are ≥ 0 and at least one eigenvalue is zero. This yields the power mean Laplacian

$$\boldsymbol{L}_p := M_p(\boldsymbol{L}^{(1)}_{\text{sym}}, \dots, \boldsymbol{L}^{(T)}_{\text{sym}}) = \left(\frac{1}{T} \sum_{t=1}^{T} (\boldsymbol{L}^{(t)}_{\text{sym}})^p \right)^{1/p}. \tag{2.1}$$

Note that the case $p = 1$ already occurred in [27]. For $p < 0$, [37, Section 2.2] proposes to apply a diagonal shift of $\delta > 0$ to each symmetric graph Laplacian to obtain a (strictly) positive definite version

$$\boldsymbol{L}^{(t)}_{\text{sym},\delta} = \boldsymbol{L}^{(t)}_{\text{sym}} + \delta \boldsymbol{I},$$

where the choice $\delta = \log(1 + |p|)$ was suggested in [37]. Combining the $\boldsymbol{L}^{(t)}_{\text{sym},\delta}$ for the different

layers yields the shifted power mean Laplacian

$$\boldsymbol{L}_{p,\delta} := M_p(\boldsymbol{L}^{(1)}_{\mathrm{sym},\delta}, \dots, \boldsymbol{L}^{(T)}_{\mathrm{sym},\delta}) = \left(\frac{1}{T} \sum_{t=1}^{T} (\boldsymbol{L}^{(t)}_{\mathrm{sym},\delta})^p \right)^{1/p}. \tag{2.2}$$

Note, that for real symmetric matrices $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ with eigendecomposition $\boldsymbol{A} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^\top$, the result of a general matrix function $f(\boldsymbol{A})$ can be obtained via $f(\boldsymbol{A}) = \boldsymbol{\Phi} f(\boldsymbol{\Lambda})\boldsymbol{\Phi}^\top$, where $f(\boldsymbol{\Lambda}) = \mathrm{diag}(f(\lambda_i)_{i=1}^n)$. This means f only acts on the eigenvalues. In particular, λ^p is an eigenvalue of the matrix power $\boldsymbol{A}^p$ if λ is an eigenvalue of $\boldsymbol{A}$, and $1-\lambda$ is an eigenvalue of $\boldsymbol{I}-\boldsymbol{A}$, cf. e.g. [22].

3. Computation of eigenpairs and Polynomial Krylov Subspace Method. In several learning tasks when using the power mean Laplacian $\boldsymbol{L}_1$ or the shifted version $\boldsymbol{L}_{p,\delta}$, $p<0$, one needs to perform matrix-vector products with this matrix or the pth power

$$\boldsymbol{L}^p_{p,\delta} = \frac{1}{T} \sum_{t=1}^{T} (\boldsymbol{L}^{(t)}_{\mathrm{sym},\delta})^p. \tag{3.1}$$

For instance, for classification based on semi-supervised or unsupervised learning, the eigenpairs belonging to the k smallest eigenvalues are of particular interest, as they contain important information. cf. e.g. [53].

In the case $p=1$, one has $\boldsymbol{L}_1 = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{L}^{(t)}_{\mathrm{sym}}$, and the relevant eigeninformation of $\boldsymbol{L}_1$ can be easily obtained by computing the eigenpairs for the k largest eigenvalues of

$$\boldsymbol{I} - \boldsymbol{L}_1 = \frac{1}{T} \sum_{t=1}^{T} (\boldsymbol{D}^{(t)})^{-1/2}\, \boldsymbol{W}^{(t)}\, (\boldsymbol{D}^{(t)})^{-1/2} \tag{3.2}$$

via the Lanczos method [19]. As mentioned in Section 2, the resulting eigenvectors $\boldsymbol{\phi}$ are identical with the ones of $\boldsymbol{L}_1$ and the eigenvalues λ of (3.2) correspond to $1-\lambda$ of $\boldsymbol{L}_1$.

Likewise, in the case of general p for a given eigenvalue λ and eigenvector $\boldsymbol{\phi}$ of the real symmetric matrix $\boldsymbol{L}_{p,\delta}$, the corresponding eigenvalue of $\boldsymbol{L}^p_{p,\delta}$ is λ^p and the eigenvector remains $\boldsymbol{\phi}$, cf. Section 2. Since the function $f(\lambda) = \lambda^p$ is order reversing for $p<0$, it holds for $\lambda_1 \le \lambda_2 \le \dots \le \lambda_n$ that $\lambda_1^p \ge \lambda_2^p \ge \dots \ge \lambda_n^p$. In order to obtain the first k smallest eigenvalues $\lambda_1, \dots, \lambda_k$ and corresponding eigenvectors $\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_k$ in the case $p<0$, it is sufficient to compute the k largest eigenvalues $\lambda_1^p, \dots, \lambda_k^p$ with its eigenvectors of $\boldsymbol{L}^p_{p,\delta}$. For this, we propose to utilize the Lanczos method, which requires matrix-vector multiplications of the graph Laplacian matrices $(\boldsymbol{L}^{(t)}_{\mathrm{sym},\delta})^p$, cf. (3.1). The latter can be realized by using the Polynomial Krylov Subspace Method (PKSM) [37], which is again based on a Lanczos scheme for the real symmetric matrices $\boldsymbol{L}^{(t)}_{\mathrm{sym},\delta}$, cf. [22, Chap. 13].

When the weight functions $w^{(t)}$ and, correspondingly, the weight matrices $\boldsymbol{W}^{(t)}$ of the layers $t \in \{1, \dots, T\}$ have a special structure, the matrix-vector multiplications and eigenpair computations can be distinctly accelerated, as we discuss in the next section.

4. NFFT-based fast summation for fast matrix-vector multiplications with the weight matrix. For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, when the nodes $x_i \in \mathcal{V}$ are identified with feature vectors $\boldsymbol{x}_i \in \mathbb{R}^d$ and the weight function $w\colon \mathcal{V} \times \mathcal{V} \to \mathbb{R}_{\geq 0}$ is associated with a kernel function $K\colon \mathbb{R}^d \to \mathbb{R}_{\geq 0}$, $w(x_i, x_j) = K(\boldsymbol{x}_i - \boldsymbol{x}_j)$, matrix-vector multiplications with the weight matrix $\boldsymbol{W} \in \mathbb{R}_{\geq 0}^{n \times n}$ can be sped up dramatically even if the matrix is non-sparse. One very efficient method is the NFFT-based fast summation [45, 46] which achieves a runtime complexity of $\mathcal{O}(n)$. Subsequently, we briefly describe the general ideas and give a fast algorithm. For more details, we refer to [1, Sec. 3].

For technical reasons, we consider the modified weight matrix

$$\tilde{\boldsymbol{W}} := \boldsymbol{W} + K(\boldsymbol{0})\, \boldsymbol{I},$$

which corresponds to the original $\boldsymbol{W}$ except that the diagonal now contains the value $K(\boldsymbol{0})$ instead of 0, and we have $\boldsymbol{W} = \tilde{\boldsymbol{W}} - K(\boldsymbol{0})\, \boldsymbol{I}$. Then, the matrix-vector multiplication of $\boldsymbol{W}$ with an arbitrary vector $\boldsymbol{v} \in \mathbb{C}^n$ can be written as $\boldsymbol{W}\boldsymbol{v} = \tilde{\boldsymbol{W}}\boldsymbol{v} - K(\boldsymbol{0})\, \boldsymbol{v}$. The last part $K(\boldsymbol{0})\, \boldsymbol{v}$ is simply a multiplication of a scalar value with a vector, and we will compute the first part $\tilde{\boldsymbol{W}}\boldsymbol{v}$ in a fast way using the NFFT-based fast summation.

Each entry of the result $\tilde{\boldsymbol{W}}\boldsymbol{v}$ reads as

$$\left(\tilde{\boldsymbol{W}}\boldsymbol{v}\right)_i = f(\boldsymbol{x}_i) := \sum_{j=1}^{n} v_j\, K(\boldsymbol{x}_i - \boldsymbol{x}_j). \tag{4.1}$$

The key idea for the efficient computation of (4.1) uses methods from Fourier analysis [44]. In particular, the kernel function K is approximated by a d-variate trigonometric polynomial K_{RF}, which allows to separate the computations involving the nodes x_i and x_j. This will be one main ingredient for reducing the computational complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

Assuming we have a suitable approximation of K given by

$$K(\boldsymbol{y}) \approx K_{\mathrm{RF}}(\boldsymbol{y}) := \sum_{\boldsymbol{l} \in I_N} \hat{b}_{\boldsymbol{l}}\, \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{y}}, \quad I_N := \{-N/2, -N/2+1, \ldots, N/2-1\}^d, \tag{4.2}$$

with bandwidth $N \in 2\mathbb{N}$ and Fourier coefficients $\hat{b}_{\boldsymbol{l}}$, we replace K by K_{RF} in (4.1) and we obtain

$$\begin{aligned} \left(\tilde{\boldsymbol{W}}\boldsymbol{x}\right)_i = f(\boldsymbol{x}_i) \approx f_{\mathrm{RF}}(\boldsymbol{x}_i) &:= \sum_{j=1}^{n} v_j\, K_{\mathrm{RF}}(\boldsymbol{x}_i - \boldsymbol{x}_j) = \sum_{j=1}^{n} v_j \sum_{\boldsymbol{l} \in I_N} \hat{b}_{\boldsymbol{l}}\, \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} (\boldsymbol{x}_i - \boldsymbol{x}_j)} \\ &= \sum_{\boldsymbol{l} \in I_N} \hat{b}_{\boldsymbol{l}} \underbrace{\left(\sum_{j=1}^{n} v_j\, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{x}_j} \right)}_{=: \hat{f}_{\boldsymbol{l}}} \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{x}_i}, \quad \forall i = 1, \ldots, n. \end{aligned} \tag{4.3}$$

Comparing (4.3) with the initial problem of evaluating (4.1), the Fourier approximation K_{RF} of the kernel function K has so far only introduced an additional sum over I_N. In situations of large n however, the NFFT [24] manages to substantially speed up the evaluation of the

inner sums $\hat{f}_{\boldsymbol{l}} := \left(\sum_{j=1}^{n} v_j e^{-2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{x}_j}\right)$, $\boldsymbol{l} \in I_N$, as well as the computation of the outer sums $f_{\mathrm{RF}}(\boldsymbol{x}_i) := \sum_{\boldsymbol{l} \in I_N} \hat{b}_{\boldsymbol{l}}\, \hat{f}_{\boldsymbol{l}}$, $i = 1, \ldots, n$. Therefore, the NFFT is the second main ingredient for the reduction in the computational complexity from $\mathcal{O}(n^2)$, when using the direct summation (4.1), to $\mathcal{O}\big((\mathtt{m}_{\mathrm{NFFT}})^d\, n + N^d \log N\big)$ for computing (4.3) via the NFFT-based fast summation, where $\mathtt{m}_{\mathrm{NFFT}}$ represents an internal window cut-off parameter of the NFFT controlling the desired precision. In situations where the number n of nodes of the graph is large and the feature space dimension d is not too big, this represents a crucial gain in computational complexity.

The accuracy of the Fourier approximation K_{RF} of the kernel function K depends on the decay of the Fourier coefficients of K, which are influenced by smoothness properties of the kernel function. For instance, smooth rotational invariant kernel functions are particularly suited, e.g. the Gaussian RBF kernel $K(\boldsymbol{y}) = \exp(-\|\boldsymbol{y}\|^2/\sigma^2)$, Laplacian RBF kernel $K(\boldsymbol{y}) = \exp(-\|\boldsymbol{y}\|/\sigma)$ and many others. The Fourier coefficients $\hat{b}_{\boldsymbol{l}}$ of the Fourier approximation K_{RF} can be computed easily by sampling the kernel function K or a regularized version of K on an equispaced grid and applying a FFT which takes $\mathcal{O}(N^d \log N)$ arithmetic operations, see also [1, Sec. 3] for more details. Since the parameter $\mathtt{m}_{\mathrm{NFFT}}$ and the bandwidth N only depend on the desired accuracy, we have a computational complexity of $\mathcal{O}(n)$.

Note that since the Fourier approximation K_{RF} is 1-periodic by construction, but the original kernel K function may be not, the feature vectors $\boldsymbol{x}_i$ have to be scaled and shifted into a suitable subinterval of the cube $[-1/4, 1/4]^d$, cf. [1, Sec. 3] for further details. After this transformation, each difference $\boldsymbol{x}_i - \boldsymbol{x}_j$ lies in a subinterval of $[-1/2, 1/2]^d$, corresponding to the 1-periodicity of K_{RF}. Since the scaling may influence the values of the kernel function K or of the Fourier approximation K_{RF}, possible control parameters may have to be adapted, e.g., the scaling parameter σ of the Gaussian or Laplacian RBF kernel.

In total, we have a fast approximate algorithm for the matrix-vector multiplication $\tilde{\boldsymbol{W}}\boldsymbol{v}$ of complexity $\mathcal{O}(n)$ available, cf. Algorithm 4.1. This algorithm is implemented as `applications/fastsum` and `matlab/fastsum` in C and MATLAB within the NFFT3 software library and freely available, see [24]. Then, one easily computes $\boldsymbol{W}\boldsymbol{v}$ from $\tilde{\boldsymbol{W}}\boldsymbol{v}$ by subtracting the vector $K(\boldsymbol{0})\,\boldsymbol{v}$.

Note that Algorithm 4.1 can be used for accelerating the eigenpair computation of the power mean Laplacian $\boldsymbol{L}_1$ and of the shifted power mean Laplacian $\boldsymbol{L}_{p,\delta}^{p}$, $p < 0$. In the latter case, this means applying Algorithm 4.1 inside the Polynomial Krylov Subspace Method, see also Section 3.

In the next section, the eigeninformation of $\boldsymbol{L}_1$ and $\boldsymbol{L}_{p,\delta}^{p}$, $p < 0$, is used in order to perform semi-supervised learning based on multilayer graphs.

5. Graph Allen–Cahn for multiclass problems.

Diffuse interface methods are heavily used in materials science and beyond, see e.g. [42, 56, 57, 4] and the references therein. They offer an efficient and flexible way to model phase separation with one of the most prominent models being the Allen–Cahn equation [2] that originally describes the evolution of a binary solution over time. Here, a physical system consisting of two liquid components is described, which exhibits a phase separation behavior. The

Algorithm 4.1 ([1, Alg. 1]). Fast approximate matrix-vector multiplication $\tilde{\boldsymbol{W}}\boldsymbol{x}$ using NFFT-based fast summation, $(\tilde{\boldsymbol{W}}\boldsymbol{v})_i = \sum_{j=1}^{n} v_j\, K(\boldsymbol{x}_i - \boldsymbol{x}_j)\ \forall i = 1, \dots, n$,
e.g. $K(\boldsymbol{x}_i - \boldsymbol{x}_j) = \exp(-\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|^2 / \sigma^2)$.

Input:	$\left(\hat{b}_{\boldsymbol{l}}\right)_{\boldsymbol{l}\in I_N}$	Fourier coefficients of trigonometric polynomial K_{RF} which approximates K,
	$\{\boldsymbol{x}_i\}_{i=1}^{n}$	nodes, $\boldsymbol{x}_i \in \mathbb{R}^d$, $\|\boldsymbol{x}_i\| \in [-1/4, 1/4]^d$,
	$\boldsymbol{v} = [v_1, v_2, \dots, v_n]^T$	vector $\in \mathbb{R}^n$.

1. Apply d-dimensional adjoint NFFT on $\boldsymbol{v}$ and obtain $\hat{v}_{\boldsymbol{l}} \approx \sum_{j=1}^{n} v_j\, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l}\boldsymbol{x}_j}\ \forall \boldsymbol{l} \in I_N$.
2. Multiply result by Fourier coefficients $\left(\hat{b}_{\boldsymbol{l}}\right)_{\boldsymbol{l}\in I_N}$ of K_{RF} and obtain $\hat{f}_{\boldsymbol{l}} := \hat{b}_{\boldsymbol{l}}\, \hat{v}_{\boldsymbol{l}}\ \forall \boldsymbol{l} \in I_N$.
3. Apply d-dimensional NFFT on $\left(\hat{f}_{\boldsymbol{l}}\right)_{\boldsymbol{l}\in I_N}$ and obtain output $\tilde{f}_{\mathrm{RF}}(\boldsymbol{x}_i) \approx \sum_{\boldsymbol{l}\in I_N} \hat{f}_{\boldsymbol{l}}\, \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{l}\boldsymbol{x}_i}\ \forall i = 1, \dots, n$.

Output:	$\left[\tilde{f}_{\mathrm{RF}}(\boldsymbol{x}_i)\right]_{i=1,\dots,n}$	$\tilde{f}_{\mathrm{RF}}(\boldsymbol{x}_i) \approx (\tilde{\boldsymbol{W}}\boldsymbol{v})_i\ \forall i = 1, \dots, n$.
Complexity:	$\mathcal{O}(n)$ for fixed accuracy.	

Allen–Cahn equation is derived as the gradient flow of the Ginzburg-Landau energy functional

$$E(u) = \int \frac{\epsilon}{2} |\nabla u|^2 \mathrm{d}\boldsymbol{x} + \int \frac{1}{\epsilon} \psi(u) \mathrm{d}\boldsymbol{x}, \tag{5.1}$$

where we obtain

$$\frac{\partial u}{\partial t} = -\nabla E(u) = \epsilon \Delta u - \frac{1}{\epsilon} \psi'(u). \tag{5.2}$$

Here u describes the concentration of the (liquid) components depending on a physical domain Ω and time t, and $\epsilon > 0$ is a (typically small) parameter which influences the width of the interface regions where the transition from one pure phase to the other happens. The gradient term represents the Dirichlet energy that penalizes the length of the interface and ψ is a suitable potential function describing the inner energy content of the system having minima at the pure phases. For more details, we refer to Appendix A. Standard solution methods for partial differential equations, such as the finite element method or the finite difference method, can be used to solve (5.2), given suitable initial conditions as well as boundary conditions.

The Allen–Cahn equation has been adapted to binary semi-supervised classification on graphs, see [5, 9] and Appendix A. Most importantly, the spatial domain Ω is replaced by the graph domain, and in particular, this means that Ω is identified with the set of vertices $x_i \in \mathcal{V}$ of a graph $\mathcal{G}$. Moreover, a data fidelity term is added to the Ginzburg-Landau energy functional (5.1) to include the correct classification of the a priori labeled data as an additional objective.

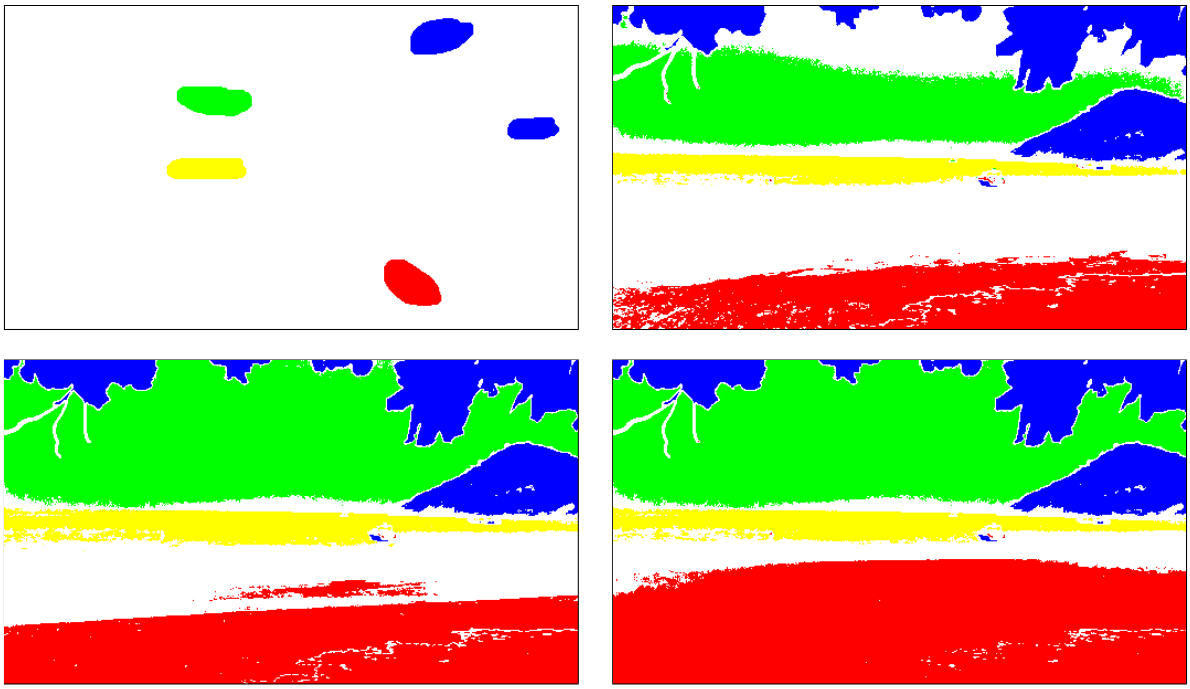

Figure 2. *Example of the evolution of a phase-field simulation for image segmentation based on Graph Allen–Cahn with four classes, where the four colors red, green, blue, and yellow indicate different class affiliations with score values higher than 0.66.*

Furthermore, the method has been extended to the multiclass case, see e.g. [18, 7, 36] as well as Figure 2 for an image segmentation example. For the general case of $m \geq 2$ classes, the concentration u of the phase-field description on the finite set of vertices $\mathcal{V} = \{x_i\}_{i=1}^n$ is identified with a matrix $\boldsymbol{U} \in \mathbb{R}^{n \times m}$ containing one row of m entries per vertex x_i. The goal is to identify vertices x_i with the j-th class whenever the j-th entry in the corresponding row $\boldsymbol{u}_i^\top$ of $\boldsymbol{U}$ is largest for $j \in \{1, \dots, m\}$. In order to incorporate the known class information of a priori labeled data, the data fidelity term

$$\frac{1}{2} \sum_{i=1}^{n} \omega(x_i) \|\boldsymbol{f}_i - \boldsymbol{u}_i\|_{\ell^2}^2$$

is added to the Ginzburg-Landau energy functional (5.1) in the multiclass case, where $\omega(x_i)$ is a penalty parameter that is equal to the constant $\omega_0 \gg 0$ for labeled vertices x_i and 0 for unlabeled vertices. For the class information of the known labels, one-hot encoding is used, i.e., the vectors $\boldsymbol{f}_i$ are set to the j-th unit vector $\boldsymbol{e}_j \in \mathbb{R}^m$ for labeled data,

$$\boldsymbol{f}_i := \begin{cases} \boldsymbol{e}_j & \text{if node } x_i \text{ is of class } j, \\ \boldsymbol{0} & \text{if class of node } x_i \text{ is unknown.} \end{cases} \tag{5.3}$$

The choice of ω_0 controls the trade-off between the classical Ginzburg-Landau energy and the least squares data fidelity term. Choosing ω_0 too small bares the risk of underfitting while

choosing it too large may cause overfitting to the pre-classified data.

In addition, the Dirichlet energy term $\int \frac{\epsilon}{2}|\nabla u|^2 \mathrm{d}\boldsymbol{x}$ in the Ginzburg-Landau energy functional (5.1) is replaced by $\frac{\epsilon}{2}\mathrm{trace}(\boldsymbol{U}^\top \boldsymbol{L}_{\mathrm{sym}}\boldsymbol{U})$, which is motivated by [53, Section 5.2]. Moreover, the multi-well potential

$$\psi(\boldsymbol{u}_i) := \prod_{j=1}^{m} \frac{1}{4}\|\boldsymbol{u}_i - \boldsymbol{e}_j\|_{\ell^1}^2 = \prod_{j=1}^{m} \frac{1}{4}\left(\sum_{l=1}^{n} |\boldsymbol{u}_{il} - \delta_{jl}|\right)^2 \tag{5.4}$$

is used with minima of 0 in the corners of the Gibbs simplex

$$\Sigma^m := \left\{ (s_1, \ldots, s_m) \in [0,1]^m \colon \sum_{j=1}^{m} s_j = 1 \right\}, \tag{5.5}$$

cf. [18], and the integral is replaced by a finite sum over the vertices x_i, $i = 1, \ldots, n$. The locations of these minima correspond to the one-hot encoding (5.3) of the known class labels $\boldsymbol{f}_i$ and model the goal that each row $\boldsymbol{u}_i^\top$ of the solution $\boldsymbol{U}$ should be close to one of the unit vectors $\boldsymbol{e}_j^\top$, $j \in \{1, \ldots, m\}$. Note that for the binary case $m = 2$, the multi-well potential (5.4) corresponds to the double-well potential (A.1).

With the modifications discussed above, the discretized Ginzburg-Landau functional for the multiclass case becomes

$$\tilde{E}(\boldsymbol{U}) = \frac{\epsilon}{2}\mathrm{trace}(\boldsymbol{U}^\top \boldsymbol{L}_{\mathrm{sym}}\boldsymbol{U}) + \frac{1}{2\epsilon}\sum_{i=1}^{n}\left(\prod_{l=1}^{m}\frac{1}{4}\|\boldsymbol{u}_i - \boldsymbol{e}_l\|_{\ell^1}^2\right) + \frac{1}{2}\sum_{i=1}^{n}\omega(x_i)\|\boldsymbol{f}_i - \boldsymbol{u}_i\|_{\ell^2}^2 \tag{5.6}$$

and forms the basis for the semi-supervised classification technique considered in this work, as it can be viewed as its loss function. The potential term in this case slighty varies in contrast to the binary case as the scaling changes from $\frac{1}{\epsilon}$ to $\frac{1}{2\epsilon}$ and we stick to this formulation as this is used throughout the literature for the multi-class case.

For the solution of the Allen–Cahn equation (5.2), a numerical scheme called convexity splitting [16, 15] is commonly applied, see e.g. [36, 7, 5, 18], where $E(u)$ is split up into a convex part E_1 and a concave part $-E_2$ so that $E(u) = E_1(u) - E_2(u)$. The convex part E_1 is then treated implicitly to allow for numerical stability while the concave part $-E_2$ is treated explicitly. The derivation of the numerical scheme for the binary classification of the data into $m = 2$ classes using the double-well potential (A.1) is considered in Appendix A. Analogously, for the general multiclass case $m \geq 2$, one possible splitting

$$\tilde{E}(\boldsymbol{U}) = \tilde{E}_1(\boldsymbol{U}) - \tilde{E}_2(\boldsymbol{U})$$

reads

$$\tilde{E}_1(\boldsymbol{U}) = \frac{\epsilon}{2}\mathrm{trace}(\boldsymbol{U}^\top \boldsymbol{L}_{\mathrm{sym}}\boldsymbol{U}) + \frac{c}{2}\mathrm{trace}(\boldsymbol{U}^\top \boldsymbol{U}), \tag{5.7}$$

$$\begin{aligned}\tilde{E}_2(\boldsymbol{U}) &= \frac{c}{2}\mathrm{trace}(\boldsymbol{U}^\top \boldsymbol{U}) - \frac{1}{2\epsilon}\sum_{i=1}^{n}\left(\frac{1}{4}\prod_{l=1}^{m}\|\boldsymbol{u}_i - \boldsymbol{e}_l\|_{\ell^1}^2\right) - \frac{1}{2}\sum_{i=1}^{n}\omega(x_i)\|\boldsymbol{f}_i - \boldsymbol{u}_i\|_{\ell^2}^2, \\ &= \sum_{i=1}^{n}\frac{c}{2}(\boldsymbol{u}_i^\top \boldsymbol{u}_i) - \frac{1}{2\epsilon}\left(\frac{1}{4}\prod_{l=1}^{m}\|\boldsymbol{u}_i - \boldsymbol{e}_l\|_{\ell^1}^2\right) - \frac{1}{2}\omega(x_i)\|\boldsymbol{f}_i - \boldsymbol{u}_i\|_{\ell^2}^2,\end{aligned} \tag{5.8}$$

where a productive zero is inserted into $\tilde{E}(\boldsymbol{U})$ by adding and subtracting the convex function $\frac{c}{2}\mathrm{trace}(\boldsymbol{U}^\top \boldsymbol{U})$ with a suitable constant $c > 0$ ensuring the convexity of $\tilde{E}_2$, cf. [18]. The resulting scheme is then given by

$$\frac{\boldsymbol{U}^{l+1} - \boldsymbol{U}^l}{\Delta t} = -\epsilon \boldsymbol{L}_{\mathrm{sym}} \boldsymbol{U}^{l+1} - c\boldsymbol{U}^{l+1} + c\boldsymbol{U}^l - \frac{1}{2\epsilon}\boldsymbol{\mathcal{T}}^l - \boldsymbol{\omega}\,(\boldsymbol{U}^l - \boldsymbol{F}), \tag{5.9}$$

analog to the scheme (A.3) in the binary case. Here the derivative of the term $\frac{1}{2\epsilon}\sum_{i=1}^n (\prod_{l=1}^m \frac{1}{4}\|\boldsymbol{u}_i - \boldsymbol{e}_l\|_{\ell^1}^2)$ is given by the matrix $\boldsymbol{\mathcal{T}}(\boldsymbol{U}) \in \mathbb{R}^{n\times m}$ with the entries

$$\mathcal{T}_{ij}(\boldsymbol{U}) := \sum_{q=1}^m \frac{1}{2}(1 - 2\delta_{jq})\|\boldsymbol{u}_i - \boldsymbol{e}_q\|_{\ell^1} \prod_{\substack{p=1\\ p\neq q}}^m \frac{1}{4}\|\boldsymbol{u}_i - \boldsymbol{e}_p\|_{\ell^1}^2, \tag{5.10}$$

$\boldsymbol{\mathcal{T}}^l := \boldsymbol{\mathcal{T}}(\boldsymbol{U}^l)$, $\boldsymbol{\omega} := \mathrm{diag}\,(\omega(x_i)_{i=1}^n)$, and $\boldsymbol{F} := (\boldsymbol{f}_i^\top)_{i=1}^n \in \mathbb{R}^{n\times m}$ contains the known label information, cf. (5.3). Solving (5.9) for $\boldsymbol{U}^{l+1}$ yields

$$\boldsymbol{U}^{l+1} = \big[\underbrace{(1 + c(\Delta t))\boldsymbol{I} + \epsilon(\Delta t)\boldsymbol{L}_{\mathrm{sym}}}_{:=\boldsymbol{B}}\big]^{-1} \left((1 + c(\Delta t))\boldsymbol{U}^l - \frac{\Delta t}{2\epsilon}\boldsymbol{\mathcal{T}}^l - (\Delta t)\,\boldsymbol{\omega}\,(\boldsymbol{U}^l - \boldsymbol{F})\right), \tag{5.11}$$

and using the eigendecomposition $\boldsymbol{L}_{\mathrm{sym}} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^\top$ gives

$$\boldsymbol{B}^{-1} = \big[(1 + c(\Delta t))\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \epsilon(\Delta t)\boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^\top\big]^{-1} = \boldsymbol{\Phi}\big[(1 + c(\Delta t))\boldsymbol{I} + \epsilon(\Delta t)\boldsymbol{\Lambda}\big]^{-1}\boldsymbol{\Phi}^\top. \tag{5.12}$$

Then, writing $\boldsymbol{U}^{l+1} \in \mathbb{R}^{n\times m}$ in (5.11) with respect to the basis $\boldsymbol{\Phi} \in \mathbb{R}^{n\times n}$, $\boldsymbol{U}^{l+1} = \boldsymbol{\Phi}\tilde{\boldsymbol{V}}^{l+1}$ with the coefficient matrix $\tilde{\boldsymbol{V}}^{l+1} \in \mathbb{R}^{n\times m}$, results in

$$\tilde{\boldsymbol{V}}^{l+1} = \big[(1 + c(\Delta t))\boldsymbol{I} + \epsilon(\Delta t)\boldsymbol{\Lambda}\big]^{-1}\boldsymbol{\Phi}^\top \left((1 + c(\Delta t))\boldsymbol{U}^l - \frac{\Delta t}{2\epsilon}\boldsymbol{\mathcal{T}}^l - (\Delta t)\,\boldsymbol{\omega}\,(\boldsymbol{U}^l - \boldsymbol{F})\right).$$

Next, the eigendecomposition $\boldsymbol{\Lambda} \in \mathbb{R}^{n\times n}$, $\boldsymbol{\Phi} \in \mathbb{R}^{n\times n}$ of $\boldsymbol{L}_{\mathrm{sym}}$ is approximated by a truncated version $\boldsymbol{\Lambda}_k \in \mathbb{R}^{k\times k}$, $\boldsymbol{\Phi}_k \in \mathbb{R}^{n\times k}$ using only the k smallest eigenvalues of $\boldsymbol{L}_{\mathrm{sym}}$ (cf. Appendix A for the binary case), and $\boldsymbol{B}^{-1}$ is projected into the corresponding eigenspace. This yields the modified iteration scheme

$$\boldsymbol{V}^{l+1} = \underbrace{\big[(1 + c(\Delta t))\boldsymbol{I} + \epsilon(\Delta t)\boldsymbol{\Lambda}_k\big]^{-1}\boldsymbol{\Phi}_k^\top}_{=:\boldsymbol{Z}} \left((1 + c(\Delta t))\boldsymbol{U}^l - \frac{\Delta t}{2\epsilon}\boldsymbol{\mathcal{T}}^l - (\Delta t)\,\boldsymbol{\omega}\,(\boldsymbol{U}^l - \boldsymbol{F})\right). \tag{5.13}$$

with the coefficient matrix $\boldsymbol{V}^{l+1} \in \mathbb{R}^{k\times m}$ and the new iterate $\tilde{\boldsymbol{U}}^{l+1} := \boldsymbol{\Phi}_k \boldsymbol{V}^{l+1}$. As the matrix $\big[(1 + c(\Delta t))\boldsymbol{I} + \epsilon(\Delta t)\boldsymbol{\Lambda}_k\big] \in \mathbb{R}^{k\times k}$ is diagonal, its inverse and consequently the matrix $\boldsymbol{Z} \in \mathbb{R}^{k\times n}$ are easy to compute. Note that $\boldsymbol{Z}$ does not depend on the matrices $\boldsymbol{U}^l$ and $\boldsymbol{\mathcal{T}}^l$ from the previous iterations, which means that it can be precomputed before starting the first iteration $l = 0$. Similar to the binary case, however, the computational costs for matrix-matrix multiplications like $\boldsymbol{Z}\boldsymbol{\mathcal{T}}^l$ remain dominant, for instance involving the matrices $\boldsymbol{\Phi}_k^\top \in \mathbb{R}^{k\times n}$ and $\boldsymbol{\mathcal{T}}^l \in \mathbb{R}^{n\times m}$.

Since we cannot expect that after an iteration step a row of $\tilde{\boldsymbol{U}}^{l+1} := \boldsymbol{\Phi}_k \boldsymbol{V}^{l+1} \in \mathbb{R}^{n\times m}$ is still an element of the Gibbs simplex Σ^m, as defined in (5.5), we need to project the result of each iteration back to Σ^m, as otherwise we encountered a blow-up of the solution. To this end, the technique proposed in [10] is employed. Here, for a given $\tilde{\boldsymbol{u}}_i^{\ell+1} \in \mathbb{R}^m$, we search for the element $\boldsymbol{\sigma}_i \in \Sigma^m$ with the minimal distance, i.e.

$$\boldsymbol{\sigma}_i = \arg\min_{\boldsymbol{\sigma}\in\Sigma^m} \|\boldsymbol{\sigma} - \tilde{\boldsymbol{u}}_i^{\ell+1}\|,$$

and set $\boldsymbol{u}_i^{\ell+1}$ to $\boldsymbol{\sigma}_i$ afterwards. Due to this projection to the Gibbs simplex Σ^m, we can interpret the components of $\boldsymbol{u}_i$ as the empirical probabilities that the node $x_i \in \mathcal{V}$ belongs to each of the m classes, or in other words, scores for class affiliation.

For the initialization of $\boldsymbol{U}^0$, we set $\boldsymbol{u}_i := \boldsymbol{e}_j \in \mathbb{R}^m$ for the pre-labeled vertices x_i where j is the corresponding class. For the vertices with unknown labels, [18, Algorithm 1] suggests to use randomized initial conditions $\boldsymbol{U}^0$, which are then scaled to the Gibbs simplex. As we have no a-priori information about the unlabeled nodes, we propose to set uniform empirical probabilities to belong to the respective class, i.e. $\boldsymbol{u}_i := \frac{1}{m}\mathbf{1} \in \mathbb{R}^m$. A random initialization bares the risk of initially assigning high probabilities to wrong classes, which potentially reduces the classification accuracy of the whole method.

6. Graph Allen–Cahn on multilayer graphs for multiclass problems. In this section, we extend the graph-based multiclass Allen–Cahn approach from Section 5 to multilayer graphs. For this, we employ the concept of the power mean Laplacian in order to combine the information of all graph layers, see Section 2. Formally, we replace the graph Laplacian matrix $\boldsymbol{L}_{\text{sym}}$ in (5.6) by the power mean Laplacian $\boldsymbol{L}_1$ from (2.1) for $p = 1$ and by the shifted version $\boldsymbol{L}_{p,\delta}$ from (2.2) with $\delta = \log(1 + |p|)$ for $p < 0$. This causes changes in the term $\tilde{E}_1(\boldsymbol{U})$ in (5.7), and correspondingly, $\boldsymbol{L}_{\text{sym}}$ is replaced in the iteration formula (5.11) yielding

$$\boldsymbol{B} = \begin{cases} (1 + c(\Delta t))\boldsymbol{I} + \epsilon(\Delta t)\boldsymbol{L}_1 & \text{for } p = 1, \\ (1 + c(\Delta t))\boldsymbol{I} + \epsilon(\Delta t)\boldsymbol{L}_{p,\delta} & \text{for } p < 0. \end{cases}$$

Analogously to the single layer case, we utilize an eigendecomposition $\boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^\top$ of $\boldsymbol{L}_1$ and $\boldsymbol{L}_{p,\delta}$ for $p = 1$ and $p < 0$, respectively, where $\boldsymbol{\Lambda} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{\Phi} \in \mathbb{R}^{n\times n}$. This results in $\boldsymbol{B}^{-1}$ as in (5.12). The eigendecomposition will then be approximated by a truncated version $\boldsymbol{\Lambda}_k \in \mathbb{R}^{k\times k}$, $\boldsymbol{\Phi}_k \in \mathbb{R}^{n\times k}$ using the k smallest eigenvalues. In order to compute this eigeninformation, we use the Lanczos method, which relies on matrix-vector products with $\boldsymbol{L}_1$ and $\boldsymbol{L}_{p,\delta}^p$. For the case $p = 1$, we compute the eigenpairs belonging to the k largest eigenvalues of (3.2) as described in detail in Section 3. Otherwise, for the case $p < 0$, we use the relation

$$\boldsymbol{L}_{p,\delta}^p = \frac{1}{T}\sum_{t=1}^{T} (\boldsymbol{L}_{\text{sym},\delta}^{(t)})^p$$

to compute the eigenpairs belonging to the k largest eigenvalues $\lambda_i^p, i \in \{1, \dots, k\}$ of the pth power of the shifted power mean Laplacian $\boldsymbol{L}_{p,\delta}^p$ again using the Lanczos method. Within the Lanczos process, we employ the Polynomial Krylov Subspace Method [37] in order to

approximate the matrix powers of the single layer graph Laplacians $(\boldsymbol{L}^{(t)}_{\mathrm{sym},\delta})^p$ as described in Section 3.

Afterwards, we can use the iteration (5.13), since it only depends on the eigendecomposition of the graph Laplacian, the known label information $\boldsymbol{F}$, and the previous iterate $\boldsymbol{U}^l$. As before, we project each row of the result $\tilde{\boldsymbol{U}}^{l+1} := \boldsymbol{\Phi}_k \boldsymbol{V}^{l+1} \in \mathbb{R}^{n\times m}$ of each iteration l back to the Gibbs simplex Σ^m by the method [10] and use these projected values as the input $\boldsymbol{U}^{l+1}$ for the next iteration $l+1$.

Finally, we obtain the method summarized in Algorithm 6.1. The algorithm takes the multilayer graph Laplacian and the known label information as inputs. Algorithm 6.1 outputs the scores for the class affiliations of each node $x_i \in \mathcal{V}$ of the multilayer graph. Based on these scores, we predict the class by a majority vote, i.e., we take the row-wise maximum of the output matrix $\boldsymbol{U}^l$, where ties are broken by class ID in ascending order.

Algorithm 6.1 Computation of the multiclass scores for the class affiliations of each node of the multilayer graph using a graph Allen–Cahn type method.

Input: $\boldsymbol{L}^{(t)}_{\mathrm{sym}} \in \mathbb{R}^{n\times n}$, $t = 1,\dots,T$ — graph Laplacian matrix for each layer or function handle for matrix-vector multiplication,

$\boldsymbol{F} := (\boldsymbol{f}_i^\top)_{i=1}^n \in \{0,1\}^{n\times m}$ — known label information as per (5.3).

Parameters: p, k, ϵ, ω_0, c, Δt, `tolerance`, `max_iter`

1. If $p = 1$, then compute eigenpairs $\tilde{\boldsymbol{\Lambda}}_k, \boldsymbol{\Phi}_k$ of $\boldsymbol{I} - \boldsymbol{L}_1$ belonging to the k largest eigenvalues, using the Lanczos method. Set $\boldsymbol{\Lambda}_k := \boldsymbol{I} - \tilde{\boldsymbol{\Lambda}}_k$.
   Otherwise, for $p < 0$, using the Lanczos method in combination with the PKSM [37], compute the k-largest eigenvalues $\lambda_1^p,\dots,\lambda_k^p$ of $\boldsymbol{L}^p_{p,\delta}$ with its eigenvectors, $\delta := \log(1+|p|)$. Collect the eigenvectors in $\boldsymbol{\Phi}_k$. Obtain $\boldsymbol{\Lambda}_k := \mathrm{diag}(\lambda_1,\dots,\lambda_k)$.
2. Compute $\boldsymbol{Z} := \big[(1+c(\Delta t))\boldsymbol{I} + \epsilon(\Delta t)\boldsymbol{\Lambda}_k\big]^{-1}\boldsymbol{\Phi}_k^\top$.
3. Initialize $\boldsymbol{U}_0 := \big((\boldsymbol{u}_i^0)^\top\big)_{i=1}^n \in \mathbb{R}^{n\times m}$ with $\boldsymbol{u}_i^0 := \boldsymbol{e}_j \in \mathbb{R}^m$ for the pre-labeled vertices x_i where j is the corresponding class. Otherwise, use $\boldsymbol{u}_i^0 := \frac{1}{m}\mathbf{1} \in \mathbb{R}^m$ for unlabeled x_i. Initialize $l := 0$.
4. **do**
   (a) Compute the matrix $\boldsymbol{\mathcal{T}}^l = \boldsymbol{\mathcal{T}}(\boldsymbol{U}^l)$ with entries (5.10).
   (b) Compute $\boldsymbol{V}^{l+1} := \boldsymbol{Z}\big((1+c(\Delta t))\boldsymbol{U}^l - \frac{\Delta t}{2\epsilon}\boldsymbol{\mathcal{T}}^l - (\Delta t)\,\boldsymbol{\omega}\,(\boldsymbol{U}^l - \boldsymbol{F})\big)$.
   (c) Project rows of $\tilde{\boldsymbol{U}}^{l+1} := \boldsymbol{\Phi}_k\boldsymbol{V}^{l+1} \in \mathbb{R}^{n\times m}$ to Gibbs simplex Σ^m using [10] and obtain new iterate $\boldsymbol{U}^{l+1}$.
   (d) Calculate `relative_change` $:= \max_{i=1,\dots,n}\|\boldsymbol{u}_i^{l+1} - \boldsymbol{u}_i^l\|^2 / \max_{i=1,\dots,n}\|\boldsymbol{u}_i^{l+1}\|^2$.
   (e) $l \leftarrow l+1$.
   while `relative_change` $>$ `tolerance` and $l <$ `max_iter`

Output: $\boldsymbol{U}^l := \big((\boldsymbol{u}_i^l)^\top\big)_{i=1}^n \in [0,1]^{n\times m}$ — scores for class affiliation of each node x_i, $\boldsymbol{u}_i^l \in \Sigma^m$.

When the weight matrix $\boldsymbol{W}^{(t)}$ of each layer $t \in \{1, \dots, T\}$ is associated with a suitable kernel function $K^{(t)} \colon \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ as discussed in Section 4, we can substantially accelerate the computation of the required eigenpairs of the power mean Laplacian by using the NFFT-based fast summation. This is true even when each weight matrix $\boldsymbol{W}^{(t)}$ is densely populated, as the computational complexity of one matrix-vector multiplication with $\boldsymbol{W}^{(t)}$ is reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ with the help of Algorithm 4.1.

This approach combined with the method presented in Section 7 will allow us to consider huge image data in our numerical tests in Subsection 8.3, applying Algorithm 6.1 on a 10 megapixel image which corresponds to a two layer graph with 10 million nodes.

7. Higher dimensional data and feature grouping. Since the number of elements in the matrices $\boldsymbol{W}$ and $\boldsymbol{L}_{\text{sym}}$ grows quadratically with respect to n for a fully connected graph, the explicit storage of the matrices quickly becomes infeasible for large data sets. One matrix for a 10 megapixel image with $n = 10^7$, for example, would require 800 Terabytes of memory in double precision. A less memory intensive approach is to blockwise assemble $\boldsymbol{W}$ on-the-fly for each matrix-vector multiplication. This does not effect the computational complexity of $\mathcal{O}(n^2)$ for one matrix-vector multiplication but still increases the computation time due to the repeated calculation of all entries of $\boldsymbol{W}$ for each matrix-vector multiplication.

Incorporating the NFFT-based fast summation from Section 4 dramatically speeds up the computations as it adapts the computational complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}\big((\mathtt{m}_{\text{NFFT}})^d\, n + N^d \log N\big)$. While the linear dependence on n enables the treatment of large data sets, the polynomial dependence of this complexity on the spatial dimension d restricts us to low-dimensional feature spaces, ideally $d \leq 3$.

For feature dimensions $d \geq 4$, we propose a feature grouping approach. The idea is to decompose the d-dimensional feature space into T subspaces of (not necessarily equal) dimensions $d^{(t)} \leq 3$, so that

$$d = \sum_{t=1}^{T} d^{(t)}.$$

We interpret the data $\boldsymbol{X}^{(t)}$ in each subspace as one graph layer $\mathcal{G}^{(t)}$ in a multilayer graph as introduced in Section 2. Defining a weight function $w^{(t)}$ for each layer enables the formation of the weight matrices $\boldsymbol{W}^{(t)}$, the degree matrices $\boldsymbol{D}^{(t)}$ and finally the individual layer graph Laplacians $\boldsymbol{L}_{\text{sym}}^{(t)}$.

While there is no theory on an optimal feature grouping yet, numerical experiments show, that grouping 'similar' features together leads to much higher accuracies in classification tasks. Again considering the image example, a very reasonable grouping of the 5-dimensional feature space of RGB-values and the x- and y-pixel coordinates for each pixel into one layer containing the RGB color information and a second layer containing the spatial xy information of the pixel yields the best image segmentation results, cf. [50].

The feature grouping approach thus gives rise to a new class of multilayer graphs, where the information from the different graph layers $\mathcal{G}^{(t)}$ is merged together again using the individual layer graph Laplacians $\boldsymbol{L}_{\text{sym}}^{(t)}$ in the power mean Laplacian introduced in Section 2. The power mean Laplacian $\boldsymbol{L}_1$ or $\boldsymbol{L}_{p,\delta}$ is then used in the Allen–Cahn classification scheme as described

in Section 6.

8. Numerical experiments. In this section, we present the numerical results for the various methods and approaches discussed in the previous sections. The corresponding MATLAB code is available at https://www.tu-chemnitz.de/mathematik/wire/codes.php.

8.1. Data generated by the stochastic block model. We start by considering two example data sets generated by the multilayer stochastic block model (SBM) [23] which creates weight matrices with a prescribed clustering structure following a random distribution approach. The value 1 in the binary weight matrix represents the presence of an edge between two nodes while 0 means no edge. The parameters $p_{\text{in}}, p_{\text{out}} \in [0, 1]$ represent the probabilities for the generation of an edge between two nodes belonging to the same class and to different classes, respectively.

The first example demonstrates that the power mean Laplacian combining the information of all layers is required for good classification results when all layers are informative. Furthermore, it compares the classification performance for different powers p in the power mean Laplacian. The second example illustrates that negative powers p in the power mean Laplacian are robust against noisy data in some layers when other layers are informative while the power mean Laplacian with positive powers p is not.

In the first example we consider a multilayer graph with $T = 3$ layers and $m = 3$ classes each consisting of $n_{\text{cluster}} = 50$ nodes. We choose $p_{\text{in}} = 0.7$ and $p_{\text{out}} = 0.3$ such that each layer $i = 1, 2, 3$ separates the nodes belonging to cluster i from the remaining two classes which distributes the necessary information for perfect graph segmentation across all three layers. We apply Algorithm 6.1 and set the Allen–Cahn parameters $\epsilon = 0.005$, $\omega_0 = 1\,000$, $c = \omega_0 + 3/\epsilon$, $\Delta t = 0.01$, `max_iter` $= 300$ and `tolerance` $= 10^{-6}$. As mentioned in Section 6, we predict the class taking the row-wise maximum of the output matrix $\boldsymbol{U}^l$. We pre-label 4% of the nodes per class, average over 100 random graphs and compare the performance of the three single layer graph Laplacians $\boldsymbol{L}_{\text{sym}}^{(1)}$, $\boldsymbol{L}_{\text{sym}}^{(2)}$, $\boldsymbol{L}_{\text{sym}}^{(3)}$, the three combinations of power mean Laplacians using two out of the three layers $\boldsymbol{L}_{p,\delta}^{(12)}$, $\boldsymbol{L}_{p,\delta}^{(13)}$, $\boldsymbol{L}_{p,\delta}^{(23)}$, and the power mean Laplacian using all three layers $\boldsymbol{L}_{p,\delta}$ in the multiclass Allen–Cahn scheme. For each graph Laplacian, we choose $k = 3$ eigenpairs. We visualize the average clustering errors, i.e., the relative differences between our predicted classes and the groundtruth averaged over the 100 random graphs, for different p in Figure 3.

The results illustrate that the multiclass Allen–Cahn scheme obtains very good classification results only when the information of all three layers is combined in the power mean Laplacian $\boldsymbol{L}_{p,\delta}$. Removal of informative layers leads to a significant loss in classification accuracy. While the informativity of different layers will generally not be equally distributed across different layers in real world data sets this example still illustrates the advantage of the power mean Laplacian for multilayer graphs over classical single-layer graph tools like the single layer graph Laplacian. Figure 3 also shows a monotonous decrease in the classification error of all power mean Laplacians for a descreasing power p.

Another example for the superiority of the choice of negative parameters p in the power mean Laplacian is given in the second example where we generate a multilayer graph with $T = 2$ layers and $m = 2$ classes each consisting of $n_{\text{cluster}} = 50$ nodes with both layers

p	$\boldsymbol{L}_{\mathrm{sym}}^{(1)}$	$\boldsymbol{L}_{\mathrm{sym}}^{(2)}$	$\boldsymbol{L}_{\mathrm{sym}}^{(3)}$	$\boldsymbol{L}_{p,\delta}^{(12)}$	$\boldsymbol{L}_{p,\delta}^{(13)}$	$\boldsymbol{L}_{p,\delta}^{(23)}$	$\boldsymbol{L}_{p,\delta}$
10	33.9	34.3	34.0	32.2	32.7	32.9	3.50
5	33.9	34.3	34.0	31.2	31.1	31.7	1.59
2	33.9	34.3	34.0	28.8	29.7	30.0	0.84
1	33.9	34.3	34.0	27.2	28.7	28.9	0.53
−1	33.9	34.3	34.0	20.2	23.9	23.2	0.24
−2	33.9	34.3	34.0	19.0	22.4	22.0	0.23
−5	33.9	34.3	34.0	14.1	17.1	16.4	0.17
−10	33.9	34.3	34.0	9.05	8.99	10.1	0.07

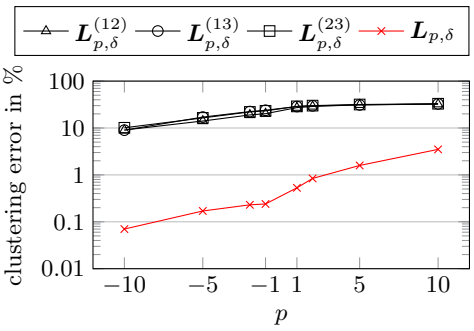

Figure 3. *Clustering errors in percent for different graph Laplacians on the 3 class SBM data set with* $p_{\mathrm{in}} = 0.7$ *and* $p_{\mathrm{out}} = 0.3$.

separating the two classes by the choice of the probabilities $p_{\mathrm{in}} = 0.7$ and $p_{\mathrm{out}} = 0.3$ in a first step. We employ the Allen–Cahn scheme using $k = 2$ eigenpairs and 4% pre-labeled points and compare its performance using $p = 10$ and $p = -10$ averaging over 100 random graphs. The classification errors of 0.05% for $p = 10$ and 0.03% for $p = -10$ are both very good. In a second step we keep the first layer informative while making the second layer noisy by assigning the equal probabilities $p_{\mathrm{in}} = p_{\mathrm{out}} = 0.5$. The same Allen–Cahn classifier now produces a classification error of 19.0% for $p = 10$ while the negative power $p = -10$ performs much better with an error of only 2.8%. For comparison, $p = 1$ yields an accuracy of 7.7% in this example. This result confirms the observations made for spectral clustering in [37] for the Allen–Cahn scheme, namely that negative powers in the power mean Laplacian tend to outperform positive powers and are robust against uninformative layers. In that light, the development of efficient numerical methods for the computation of the eigeninformation of the power mean Laplacian for the more difficult case $p > 0$ appears unattractive.

8.2. Small multilayer data sets. Next we classify some small real world example data sets with an inherent multilayer structure that still permit the explicit formation of the power mean Laplacian. Here we consider the data sets from [38, Section 6], where the three data sets Citeseer [30], Cora [33] and WebKB [12] have one layer, the three data sets 3sources [29], BBCS [21] and Wikipedia [47] have three layers, the data set BBC [20] has four layers, and the data set UCI [51] six layers.

We chose these data sets as they are readily available, and we can compare our new results with the multilayer SSL approach presented in [38, Section 3]. This approach uses the power mean Laplacian for a generalized Tikhonov type regularization, also called ridge regression, solving the optimization problem

$$(u_{i,j})_{i=1}^{n} = \arg\min_{\boldsymbol{y}\in\mathbb{R}^n} \|\boldsymbol{y} - (f_{i,j})_{i=1}^{n}\|^2 + \lambda \boldsymbol{y}^\top \boldsymbol{L}\boldsymbol{y} \tag{8.1}$$

for each class $j \in \{1, \dots, m\}$, where $\boldsymbol{L}$ is either $\boldsymbol{L}_1$ or $\boldsymbol{L}_{p,\delta}$ with $p < 0$, and the label of a node v_i is determined by $\arg\max\{u_{i,j}\}_{j=1}^{m}$. In particular, the test code for [38, Section 6] including the known label information for the data sets can be obtained from [35], and we were able to exactly reproduce the corresponding results in [38, Table 2] which present the mean misclassification rates of 10 test runs with different random known label information.

3sources [29] ($n = 169$, $T = 3$, $m = 6$; AC: $\sigma = 5$, $k = 16$)

known labels	1%	5%	10%	15%	20%	25%
SSL $\boldsymbol{L}_1$	33.5	23.9	23.4	20.1	15.6	14.6
SSL $\boldsymbol{L}_{-1,\delta}$	**28.4**	**20.0**	21.8	22.0	17.2	17.9
SSL $\boldsymbol{L}_{-10,\delta}$	40.9	29.1	21.9	19.3	14.8	14.7
AC $\boldsymbol{L}_1$	39.9	25.1	**17.3**	15.6	**10.6**	**10.2**
AC $\boldsymbol{L}_{-1,\delta}$	40.2	25.3	**17.3**	**15.3**	**10.6**	10.4
AC $\boldsymbol{L}_{-10,\delta}$	38.5	25.1	17.6	**15.4**	**10.7**	10.6

BBC [20] ($n = 685$, $T = 4$, $m = 5$; AC: $\sigma = 6$, $k = 31$)

known labels	1%	5%	10%	15%	20%	25%
SSL $\boldsymbol{L}_1$	31.3	22.8	17.4	13.5	10.2	8.9
SSL $\boldsymbol{L}_{-1,\delta}$	**31.0**	17.0	11.5	10.5	9.2	8.7
SSL $\boldsymbol{L}_{-10,\delta}$	51.6	26.9	16.6	12.8	10.3	9.5
AC $\boldsymbol{L}_1$	41.9	**12.9**	8.9	**7.6**	**7.0**	**6.1**
AC $\boldsymbol{L}_{-1,\delta}$	41.9	**13.0**	**8.8**	**7.5**	**6.9**	**6.1**
AC $\boldsymbol{L}_{-10,\delta}$	42.5	13.2	**8.7**	**7.5**	**6.9**	**6.2**

BBCS [21] ($n = 544$, $T = 2$, $m = 5$; AC: $\sigma = 2$, $k = 62$)

known labels	1%	5%	10%	15%	20%	25%
SSL $\boldsymbol{L}_1$	29.9	15.0	13.5	10.6	8.7	7.2
SSL $\boldsymbol{L}_{-1,\delta}$	**23.8**	**11.6**	8.7	**6.3**	**5.8**	**5.1**
SSL $\boldsymbol{L}_{-10,\delta}$	48.7	22.5	14.2	9.1	7.8	6.1
AC $\boldsymbol{L}_1$	52.4	14.5	**8.5**	6.5	**5.8**	**5.0**
AC $\boldsymbol{L}_{-1,\delta}$	52.3	14.7	**8.6**	6.5	**5.8**	**5.0**
AC $\boldsymbol{L}_{-10,\delta}$	52.1	14.2	**8.6**	**6.4**	6.0	**5.0**

Wikipedia [47] ($n = 693$, $T = 2$, $m = 10$; AC: $\sigma = 2$, $k = 74$)

known labels	1%	5%	10%	15%	20%	25%
SSL $\boldsymbol{L}_1$	68.2	61.1	53.6	48.3	44.1	42.3
SSL $\boldsymbol{L}_{-1,\delta}$	59.1	52.3	40.2	36.3	35.1	34.1
SSL $\boldsymbol{L}_{-10,\delta}$	66.9	57.2	43.2	38.7	36.3	34.9
AC $\boldsymbol{L}_1$	**48.4**	**39.4**	**31.4**	**30.1**	**28.7**	**27.9**
AC $\boldsymbol{L}_{-1,\delta}$	**48.4**	**39.4**	**31.4**	**30.0**	**28.7**	**27.9**
AC $\boldsymbol{L}_{-10,\delta}$	**48.4**	**39.4**	**31.4**	**30.1**	**28.8**	**27.9**

UCI [51] ($n = 2\,000$, $T = 6$, $m = 10$; AC: $\sigma = 10$, $k = 98$)

known labels	1%	5%	10%	15%	20%	25%
SSL $\boldsymbol{L}_1$	31.3	23.8	18.7	15.6	14.4	13.2
SSL $\boldsymbol{L}_{-1,\delta}$	**30.5**	17.1	13.8	12.6	12.3	11.9
SSL $\boldsymbol{L}_{-10,\delta}$	57.0	33.8	23.7	17.6	15.3	13.4
AC $\boldsymbol{L}_1$	39.6	**11.6**	**8.7**	**7.9**	7.1	**6.5**
AC $\boldsymbol{L}_{-1,\delta}$	44.5	**11.5**	**8.6**	**8.0**	**6.9**	**6.6**
AC $\boldsymbol{L}_{-10,\delta}$	47.4	12.3	9.1	8.1	7.2	6.9

Citeseer [30] ($n = 3\,312$, $T = 1$, $m = 6$; AC: $\sigma = 2$, $k = 130$)

known labels	1%	5%	10%	15%	20%	25%
SSL $\boldsymbol{L}_1$	56.3	44.1	41.2	38.5	36.1	34.7
SSL $\boldsymbol{L}_{-1,\delta}$	**52.4**	39.0	35.6	32.6	30.9	**29.5**
SSL $\boldsymbol{L}_{-10,\delta}$	68.6	54.6	48.5	43.0	39.7	37.2
AC $\boldsymbol{L}_1$	57.9	**34.3**	**32.1**	**31.3**	**30.0**	**29.4**
AC $\boldsymbol{L}_{-1,\delta}$	57.9	**34.3**	**32.1**	**31.3**	**30.0**	**29.4**
AC $\boldsymbol{L}_{-10,\delta}$	57.9	**34.3**	**32.1**	**31.3**	**30.0**	**29.4**

Cora [33] ($n = 2\,708$, $T = 1$, $m = 7$; AC: $\sigma = 2$, $k = 85$)

known labels	1%	5%	10%	15%	20%	25%
SSL $\boldsymbol{L}_1$	50.7	38.2	33.4	31.2	28.2	25.6
SSL $\boldsymbol{L}_{-1,\delta}$	**43.2**	**31.8**	**24.5**	**21.1**	**18.8**	**17.2**
SSL $\boldsymbol{L}_{-10,\delta}$	62.0	46.3	35.4	29.4	25.2	22.3
AC $\boldsymbol{L}_1$	62.3	43.1	34.6	32.1	30.3	29.3
AC $\boldsymbol{L}_{-1,\delta}$	62.3	43.1	34.6	32.1	30.3	29.3
AC $\boldsymbol{L}_{-10,\delta}$	62.3	43.1	34.6	32.1	30.3	29.3

WebKB [12] ($n = 187$, $T = 1$, $m = 5$; AC: $\sigma = 2$, $k = 15$)

known labels	1%	5%	10%	15%	20%	25%
SSL $\boldsymbol{L}_1$	58.5	49.0	44.8	44.3	44.5	44.4
SSL $\boldsymbol{L}_{-1,\delta}$	49.9	45.5	40.7	39.5	39.9	40.3
SSL $\boldsymbol{L}_{-10,\delta}$	52.3	41.9	38.0	38.1	36.8	39.5
AC $\boldsymbol{L}_1$	**43.7**	**33.2**	**23.3**	**19.6**	**15.5**	**14.8**
AC $\boldsymbol{L}_{-1,\delta}$	**43.7**	**33.2**	**23.3**	**19.6**	**15.5**	**14.8**
AC $\boldsymbol{L}_{-10,\delta}$	**43.7**	**33.2**	**23.3**	**19.6**	**15.5**	**14.8**

Table 1

Mean misclassification rate in percent of the multiclass Allen–Cahn scheme (Algorithm 6.1 denoted by "AC") using power mean Laplacians with different parameters p in comparison with results from [37] (denoted by "SSL") using the same power mean Laplacians. For Algorithm 6.1, the parameters $\epsilon = 5 \cdot 10^{-3}$, $\omega_0 = 1\,000$, $c = \omega_0 + 3/\epsilon$, $\Delta t = 0.01$, $\texttt{tolerance} = 10^{-6}$, and $\texttt{max\_iter} = 300$ are used.

We denote these results by the prefix "SSL" in Table 1.

We employ the multiclass multilayer Allen–Cahn classification scheme (Algorithm 6.1 from Section 6) on those data sets using the power mean Laplacians $\boldsymbol{L}_1$, $\boldsymbol{L}_{-1,\delta}$, and $\boldsymbol{L}_{-10,\delta}$, where we set the parameters $\epsilon = 5 \cdot 10^{-3}$, $\omega_0 = 1\,000$, $c = \omega_0 + 3/\epsilon$, $\Delta t = 0.01$, $\texttt{tolerance} = 10^{-6}$, and $\texttt{max\_iter} = 300$. The resulting mean misclassification rates are shown in Table 1 with the prefix "AC". We use the Gaussian kernel for the weight matricies $\boldsymbol{W}$ with scaling parameter σ as mentioned in the table for each dataset.

In general, we observe that Algorithm 6.1 has a lower misclassification rate for a ratio of known labels $\geq 5\%$ than the SSL classification scheme [37], while the latter often performs better for 1% known labels. Moreover, the value of p seems to have a higher influence on the misclassification rate for [37], whereas the influence of p is distinctly smaller in case of Algorithm 6.1. Interestingly, the misclassification rates when applying the SSL classification scheme from [37] also depend on p for the single layer data sets Citeseer, Cora, and WebKB, which is not the case for Algorithm 6.1. We suspect that this behavior is caused by the shift $\delta \boldsymbol{I}$ for $p < 0$ with $\delta = \log(1 + |p|)$ in (2.2), which may influence the regularization in (8.1). In addition, the results for the method [37] are better in case of the Cora data set, whereas the results for Algorithm 6.1 in case of the WebKB data set are distinctly improved.

Figure 4. *Left: Test image* 1 *for image segmentation. Right: Pre-labeled pixels.*

One likely explanation for the different behavior of the two considered methods is as follows. Since the modified Ginzburg-Landau energy functional $\tilde{E}$ in (5.6) can be viewed as the loss function of our approach, one main difference between method [37] and Algorithm 6.1 is the additional potential term $\frac{1}{2\epsilon}\sum_{i=1}^{n}\left(\prod_{l=1}^{m}\frac{1}{4}\|\boldsymbol{u}_i-\boldsymbol{e}_l\|_{\ell^1}^2\right)$ in (5.6), which promotes more distinct class affiliations.

8.3. Image data. Image segmentation tasks have been a key application for machine learning techniques for many years. While many methods rely on convolutions, image data can also be represented as a graph. A typical approach is to interpret each pixel of an image as a graph node, which is represented by its 3-dimensional 8-bit color channel (i.e. RGB) values, that range in the interval $[0,255]\cap\mathbb{N}_0$.

One possible way to form a dense weight matrix $\boldsymbol{W}$ which also takes the spatial relation between the pixels into account is adding each pixel's location in the horizontal direction x as well as in the vertical direction y to the feature space. This way, we can again operate on a fully connected graph with feature space dimension $d=5$.

We employ our feature grouping approach from Section 7 and divide the 5-dimensional feature space into two separate graph layers of a multilayer graph with $\mathcal{G}^{(1)}$ containing the 3-dimensional color information and $\mathcal{G}^{(2)}$ the 2-dimensional spatial information of the pixel locations. The information of the two layers is then recombined using the power mean Laplacian introduced in (2.1). This feature grouping enables the fast computation of the first eigenpairs of $\boldsymbol{L}_1$ belonging to the smallest eigenvalues by applying the NFFT-based fast summation to both low-dimensional graph layers. Furthermore, numerical experiments revealed, that this grouping of "similar" features achieves better image segmentation results than the single-layer graph Laplacian on the full 5-dimensional feature space.

We test that approach on the image of Figure 4, which has about 9.7 megapixels. We vectorize the image data which leads to two data matrices $\boldsymbol{X}^{(1)}\in\mathbb{R}^{9\,734\,400\times 3}$ and $\boldsymbol{X}^{(2)}\in\mathbb{R}^{9\,734\,400\times 2}$, which are centered and then scaled to the box $[-1,1]^3$ and $[-1,1]^2$, respectively. We apply a multiclass approach with $m=4$ classes in order to segment the image into the four regions 'tree', 'beach', 'sea' and 'sky'. The right image in Figure 4 marks the classified pixels ($\approx 4\,\%$), which are read out for the initialization of the Allen–Cahn multiclass method in order to act as the known label matrix $\boldsymbol{F}$. The rows of the initial solution matrix $\boldsymbol{U}_0$ are again initialized with the corresponding unit vector $\boldsymbol{e}_j^\top$ where available and $\frac{1}{m}\boldsymbol{1}^\top$ otherwise.

(a) tree (b) beach

(c) sea (d) sky

Figure 5. *4-class image segmentation result for test image from Figure 4 using the 2-layer power mean Laplacian* $\boldsymbol{L}_1$.

For this example we use the scaling parameters $\sigma^{(1)} = 1$ and $\sigma^{(2)} = 4$ in the Gaussian kernel as well as compute $k = 12$ eigenpairs. We set the Allen–Cahn parameters $\epsilon = 0.005$, $\omega_0 = 1\,000$, $c = \omega_0 + 3/\epsilon$, $\Delta t = 0.01$, `max_iter` $= 500$ and `tolerance` $= 10^{-6}$. Moreover, for the NFFT-based fast summation, we choose the NFFT parameters bandwidth $N = 64$, window cutoff parameter $\mathtt{m}_{\mathrm{NFFT}} = 5$, regularization length $\varepsilon_{\mathrm{B}} = 1/16$, and regularization degree $\mathtt{p}_{\mathrm{NFFT}} = 5$.

For $\boldsymbol{L}_1$, i.e., the case $p = 1$, the computation of the $k = 12$ eigenpairs using the Lanczos method and Algorithm 4.1 requires 1 075 seconds on a laptop computer with 16 GB RAM and an Intel Core i5-8265U CPU with 4 × 1.60–3.90 GHz cores. The Allen–Cahn scheme (Algorithm 6.1) reaches its tolerance and terminates after 260 iterations, which require 953 seconds. Note that all runtimes scale almost linearly with respect to the number of pixels n, which makes the segmentation of larger images possible.

We then take the row-wise maximum of the output matrix U in order to make our prediction, to which class each pixel most likely belongs. Figure 5 shows the original pixel color for pixels, that are identified as belonging to the respective class and white pixels otherwise. Apart from minor systematic misclassifications for objects on the sea which do not possess an own class, the method segments the image very well. When we consider $\boldsymbol{L}_{p,\delta}$ with $p = -10$ instead of $\boldsymbol{L}_1$, the classification results improve slightly.

In a second step, we consider a downscaled version of the same image as well as a similar

(a) tree (b) beach

(c) sea (d) sky

Figure 6. *4-class image segmentation result for label transfer to a second image using the 2-layer using the 2-layer power mean Laplacian* $\boldsymbol{L}_1$.

image of the same size and concatenate both images to one image with $n = 304\,720$ pixels. By keeping only the pre-labeled nodes from the first image we aim to transfer the known-label information not only to the image itself but to an unseen image consisting of the same classes. We use the same feature grouping and power mean Laplacian $\boldsymbol{L}_1$. In the second graph layer $\mathcal{G}^{(2)}$ containing the 2-dimensional spatial information of the pixel locations, we now use two connected components, one for each partial image, and the corresponding graph Laplacian $\boldsymbol{L}_{\text{sym}}^{(2)}$ is block diagonal with two large densely populated blocks belonging to each partial image. For the Gaussian kernel, the NFFT-based fast summation, and the Allen–Cahn scheme, we use the same parameters as for the single image. Moreover, we increase the number of eigenpairs to $k = 35$. Now, we only have around 2% of pre-labeled nodes and obtain the classification results displayed in Figure 6 showing obvious misclassifications in Figure 6c and 6d.

Similar to the observations made in Subsection 8.1, the result can be enhanced using a negative power $p < 0$ in the power mean Laplacian. Here in our case, using $p = -10$ leads to much better results as displayed in Figure 7. The correct detection of the different areas in the second image is again a strong result given the varying colors in the two images, especially in the sea and sky regions in Figure 7c and 7d, respectively. The computation of the $k = 35$ eigenpairs requires 602 seconds and the Allen–Cahn scheme 383 seconds.

8.4. Hyperspectral data. Finally, we tackle a problem that necessitates the full arsenal of methods derived in this paper. We consider the urban mapping problem posed by the Pavia center data set [43] as an example for the classification of the vast amounts of earth observation data gathered these days.

The original image size of 1096 by 1096 pixels contains valid ground truth labels for $n = 148\,152$ of those pixels. This together with the considerable number of classes $m = 9$ as well as the high feature space dimension arising from 102 atmospherically corrected hyperspectral frequency bands make this classification problem a demanding task. The frequently discussed combination of spectral and spatial information in the context of hyperspectral im-

(a) tree (b) beach

(c) sea (d) sky

Figure 7. *4-class image segmentation result for label transfer to a second image using the 2-layer using the 2-layer power mean Laplacian* $\boldsymbol{L}_{-10,\delta}$.

age classification [43, 55, 17] is here accomplished by adding the x- and y-pixel coordinates to the feature space, leading to a total of $d = 104$ feature variables.

As, depending on the assembling strategy of the weight matrix, the memory requirement and/or the runtime of forming the Laplacian matrices would become infeasible, we enable the efficient numerical treatment of the problem by Algorithm 6.1 using our feature grouping approach from Section 7 for the 104-dimensional feature space and the NFFT-based fast summation as discussed in Section 4 and given in Algorithm 4.1. Note, that we refrain from data preprocessing of any kind and directly work on the raw hyperspectral data. For the Gaussian kernel, we set the scaling parameter $\sigma := 8000$ for features involving hyperspectral frequency bands, which have values between 0 and 8000, and $\sigma := 2 \cdot 1095$ for coordinates layers, which have pixel coordinates between 1 and 1096. We set the Allen–Cahn parameters $\epsilon = 0.5$, $\omega_0 = 10\,000$, $c = \omega_0 + 3/\epsilon$, $\Delta t = 0.01$, $\texttt{max\_iter} = 300$, $\texttt{tolerance} = 10^{-6}$. For the NFFT-based fast summation, we choose the NFFT parameters bandwidth $N = 64$, window cutoff parameter $\mathtt{m}_{\mathrm{NFFT}} = 3$, regularization length $\varepsilon_{\mathrm{B}} = 1/16$, and regularization degree $\mathtt{p}_{\mathrm{NFFT}} = 3$. Moreover, we use 5 percent random known labels per class. The results are shown in Table 2.

These results illustrate the classification performance for different data modeling approaches. The lowest accuracies are obtained for the single layer case where we draw 100 random combinations of two and three bands, respectively, and average over the 100 test runs. Following the approach presented in Subsection 8.3, these results can be distinctly improved by adding the pixel coordinates as a second layer and using the power mean Laplacian with $p = -10$. Interestingly, a deterministic band selection improves the achieved accuracy further. Here, we compute the results averaged over the 51 combinations $(1, 52)$, $(2, 53)$, $\ldots$, $(51, 102)$ of two bands and the 34 combinations $(1, 35, 69)$, $(2, 36, 70)$, $\ldots$, $(34, 68, 102)$ of three bands respectively.

The best results, however, are obtained by utilizing all band information by employing our feature grouping approach from Section 7. Here, we use all bands with two bands per layer

layers	type	#tests	$k = 20$	$k = 40$	$k = 120$
1	2 bands (rand.)	100	0.760 ± 0.031	—	—
2	2 bands (rand.) + coord.	100	0.850 ± 0.043	0.859 ± 0.062	—
2	2 bands (det.) + coord.	51	0.864 ± 0.036	0.879 ± 0.034	—
52	51×2 bands (det.) + coord.	100	0.928 ± 0.001	0.942 ± 0.001	$\mathbf{0.957} \pm 0.001$
1	3 bands (rand.)	100	0.787 ± 0.028	—	—
2	3 bands (rand.) + coord.	100	0.885 ± 0.040	0.896 ± 0.035	—
2	3 bands (det.) + coord.	34	0.915 ± 0.020	0.920 ± 0.014	—
35	34×3 bands (det.) + coord.	100	0.930 ± 0.001	0.943 ± 0.001	$\mathbf{0.959} \pm 0.001$

Table 2

Average accuracies and standard deviations for Pavia center data set [43] using Algorithm 6.1 in combination with Algorithm 4.1 for 5% known labels per class. In the multilayer cases, $p = -10$ is used. "rand." means random and "det." deterministic frequency band selection, "coord." means coordinates layer. The Gaussian kernel with scaling parameter $\sigma = 8000$ is used for frequency bands and $\sigma = 2 \cdot 1095$ for the coordinates layer.

$((1, 52), (2, 53), \ldots, (51, 102))$, resulting in 51 frequency bands and 1 coordinate layer, as well as all bands with three bands per layer $((1, 35, 69), (2, 36, 70), \ldots, (34, 68, 102))$, resulting in 34+1 layers respectively.

Within the range of the number k of eigenpairs of the respective graph Laplacian we consider in our experiments, a higher number of eigenpairs tendencially improves the achieved accuracy at the cost of an increased runtime. Depending of the feature space dimension, a saturation of the accuracy typically sets in at some point. Choosing k too large, however, bares the risk of including noisy eigenvectors, resulting in worse accuracies, or potential convergence issues for the eigenvectors computations by the Lanczos algorithm.

Furthermore, the results can be distinctly improved in the 52 layers case of two bands per layer by modifying the scaling parameter $\sigma := 8000/2$ for the frequency bands layers and $\sigma := 1095/2$ for the coordinates layer. We achieve an average accuracy of 0.958 ± 0.001 when using $k = 40$ eigenvectors and of 0.975 ± 0.001 for $k = 120$. Using these scaling parameters σ in the case of all bands with three bands per frequency bands layer and the additional coordinates layer, we get almost the same numbers. Moreover, varying the percentage of known labels per class has only a relatively small influence on the accuracies, cf. Table 3. In particular, for only 0.5% known labels per class, we achieve average accuracies of 0.972 ± 0.003 and 0.974 ± 0.003 for two frequency bands per layer and three frequency bands per layer, respectively. Similarly, applying the same approach to other hyperspectral data sets like the Pavia university [43] or the Indian pines data set [3] also yields mean classification accuracies above 0.97 given 5% and 10% pre-known labels, respectively.

Note that there are many results for different classification approaches of the Pavia center data set available in the literature with varying foci and classification accuracies. While there are examples for the application of 'classical' methods [43, 54], some authors reduce the feature space dimension by feature selection techniques [13, 14], whereas the top classification results are achieved with highly specialized convolutional neural network (CNN) architectures [32, 28]. Our results can compete with most results presented in the literature except for problem-taylored CNN architectures. However, we emphasize again that our method operates directly on the raw data, without feature selection and data preprocessing of any kind and that it

type	known labels per class 0.25%	0.5%	1%	5%
51 × 2 bands (det.) + coord.	0.965 ± 0.005	$\mathbf{0.972} \pm 0.003$	0.968 ± 0.002	0.975 ± 0.001
34 × 3 bands (det.) + coord.	0.967 ± 0.005	$\mathbf{0.974} \pm 0.003$	0.972 ± 0.002	0.977 ± 0.001

Table 3

Average accuracies and standard deviations over 100 test runs for Pavia center data set [43] using Algorithm 6.1 with $p = -10$ and $k = 120$ in combination with Algorithm 4.1 for varying percentage of known labels per class. "det." means deterministic frequency band selection and "coord." means coordinates layer. The Gaussian kernel with scaling parameter $\sigma = 8000/2$ is used for frequency bands and $\sigma = 1095/2$ for the coordinates layer.

does not require excessive compute power or specialized hardware. In fact, all numerical experiments presented in this paper can be run on an average laptop computer.

Acknowledgement. T. Volkmer gratefully acknowledges partial funding by the Sächsische Aufbaubank – Förderbank – (SAB) 100378180.

REFERENCES

[1] D. Alfke, D. Potts, M. Stoll, and T. Volkmer, *NFFT meets Krylov methods: Fast matrix-vector products for the graph laplacian of fully connected networks*, Frontiers in Applied Mathematics and Statistics, 4 (2018).

[2] S. M. Allen and J. W. Cahn, *A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening*, Acta metallurgica, 27 (1979), pp. 1085–1095.

[3] M. F. Baumgardner, L. L. Biehl, and D. A. Landgrebe, *220 Band AVIRIS Hyperspectral Image Data Set: June 12, 1992 Indian Pine Test Site 3*, Sep 2015.

[4] K. Bergermann, *Modeling the morphology evolution of organic solar cells*, GAMM Archive for Students, 1 (2019), pp. 18–27.

[5] A. L. Bertozzi and A. Flenner, *Diffuse interface models on graphs for classification of high dimensional data*, Multiscale Modeling & Simulation, 10 (2012), pp. 1090–1118.

[6] S. Boccaletti, G. Bianconi, R. Criado, C. I. Del Genio, J. Gómez-Gardenes, M. Romance, I. Sendina-Nadal, Z. Wang, and M. Zanin, *The structure and dynamics of multilayer networks*, Physics Reports, 544 (2014), pp. 1–122.

[7] J. Bosch, S. Klamt, and M. Stoll, *Generalizing diffuse interface methods on graphs: Nonsmooth potentials and hypergraphs*, SIAM Journal on Applied Mathematics, 78 (2018), pp. 1350–1377.

[8] J. W. Cahn and J. E. Hilliard, *Free Energy of a Nonuniform System. I. Interfacial Free Energy*, The Journal of Chemical Physics, 28 (1958), pp. 258–267.

[9] L. Calatroni, Y. van Gennip, C.-B. Schönlieb, H. M. Rowland, and A. Flenner, *Graph clustering, variational image segmentation methods and Hough transform scale detection for object measurement in images*, Journal of Mathematical Imaging and Vision, 57 (2017), pp. 269–291.

[10] Y. Chen and X. Ye, *Projection onto a simplex*, ArXiv e-prints, (2011). arXiv:1101.6081.

[11] F. R. K. Chung, *Spectral Graph Theory*, vol. 92 of CBMS Regional Conference Series in Mathematics, University of Pennsylvania, Philadelphia, PA, 1997.

[12] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery, *Learning to Extract Symbolic Knowledge from the World Wide Web*, in Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, AAAI '98/IAAI '98, USA, 1998, American Association for Artificial Intelligence, pp. 509–516.

[13] M. Dalla Mura, J. A. Benediktsson, J. Chanussot, and L. Bruzzone, *The evolution of the morphological profile: From panchromatic to hyperspectral images*, in Optical Remote Sensing, Springer, 2011, pp. 123–146.

[14] A. Davari, V. Christlein, S. Vesal, A. Maier, and C. Riess, *GMM supervectors for limited training data in hyperspectral remote sensing image classification*, in International Conference on Computer Analysis of Images and Patterns, Springer, 2017, pp. 296–306.
[15] D. J. Eyre, *An unconditionally stable one-step scheme for gradient systems*, 1997.
[16] ———, *Unconditionally gradient stable time marching the Cahn-Hilliard equation*, MRS Proceedings, 529 (1998).
[17] B. Fang, Y. Li, H. Zhang, and J. C.-W. Chan, *Semi-supervised deep learning classification for hyperspectral image based on dual-strategy sample selection*, Remote Sensing, 10 (2018), p. 574.
[18] C. Garcia-Cardona, E. Merkurjev, A. L. Bertozzi, A. Flenner, and A. G. Percus, *Multiclass data segmentation using diffuse interface methods on graphs*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 36 (2014), pp. 1600–1613.
[19] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3, JHU press, 2012.
[20] D. Greene and P. Cunningham, *Producing accurate interpretable clusters from high-dimensional data*, in European Conference on Principles of Data Mining and Knowledge Discovery, Springer, 2005, pp. 486–494.
[21] ———, *A matrix factorization approach for integrating multiple data views*, in Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2009, pp. 423–438.
[22] N. J. Higham, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
[23] P. W. Holland, K. B. Laskey, and S. Leinhardt, *Stochastic blockmodels: First steps*, Social networks, 5 (1983), pp. 109–137.
[24] J. Keiner, S. Kunis, and D. Potts, *NFFT 3.5, C subroutine library.* http://www.tu-chemnitz.de/~potts/nfft and https://github.com/NFFT/nfft. Contributors: F. Bartel, M. Fenn, T. Görner, M. Kircheis, T. Knopp, M. Quellmalz, M. Schmischke, T. Volkmer, A. Vollrath.
[25] ———, *Using NFFT3 - a software library for various nonequispaced fast Fourier transforms*, ACM Transactions on Mathematical Software, 36 (2009), pp. Article 19, 1–30.
[26] M. KivelÃČÂď, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter, *Multilayer networks*, Journal of Complex Networks, 2 (2014), pp. 203–271.
[27] J. Kunegis, S. Schmidt, A. Lommatzsch, J. Lerner, E. W. D. Luca, and S. Albayrak, *Spectral Analysis of Signed Graphs for Clustering, Prediction and Visualization*, SIAM, 2010, pp. 559–570.
[28] L. Lin, C. Chen, and T. Xu, *Spatial-spectral hyperspectral image classification based on information measurement and cnn*, EURASIP Journal on Wireless Communications and Networking, 2020 (2020), pp. 1–16.
[29] J. Liu, C. Wang, J. Gao, and J. Han, *Multi-view clustering via joint nonnegative matrix factorization*, in Proceedings of the 2013 SIAM International Conference on Data Mining, SIAM, 2013, pp. 252–260.
[30] Q. Lu and L. Getoor, *Link-based classification*, in Proceedings of the 20th International Conference on Machine Learning (ICML-03), 2003, pp. 496–503.
[31] X. Luo and A. L. Bertozzi, *Convergence of the Graph AllenÃćâĆňâĂIJCahn Scheme*, Journal of Statistical Physics, 167 (2017), pp. 934–958.
[32] K. Makantasis, K. Karantzalos, A. Doulamis, and N. Doulamis, *Deep supervised learning for hyperspectral data classification through convolutional neural networks*, in 2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), IEEE, 2015, pp. 4959–4962.
[33] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, *Automating the construction of internet portals with machine learning*, Information Retrieval, 3 (2000), pp. 127–163.
[34] Z. Meng, E. Merkurjev, A. Koniges, and A. L. Bertozzi, *Hyperspectral image classification using graph clustering methods*, Image Processing On Line, 7 (2017), pp. 218–245.
[35] P. Mercado, *MATLAB implementation of the paper: Generalized matrix means for semi-supervised learning with multilayer graphs.* https://github.com/melopeo/PM_SSL.
[36] P. Mercado, J. Bosch, and M. Stoll, *Node classification for signed social networks using diffuse interface methods*, in Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2019, pp. 524–540.
[37] P. Mercado, A. Gautier, F. Tudisco, and M. Hein, *The power mean Laplacian for multilayer graph clustering*, in Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, A. Storkey and F. Perez-Cruz, eds., vol. 84 of Proceedings of Machine Learning Research,

Playa Blanca, Lanzarote, Canary Islands, 09–11 Apr 2018, PMLR, pp. 1828–1838.
[38] P. Mercado, F. Tudisco, and M. Hein, *Generalized matrix means for semi-supervised learning with multilayer graphs*, ArXiv e-prints, (2019). arXiv:1910.13951.
[39] B. Mohar, *The Laplacian spectrum of graphs*, in Graph Theory, Combinatorics, and Applications, Wiley, 1991, pp. 871–898.
[40] B. Mohar, *Some applications of Laplace eigenvalues of graphs*, Springer Netherlands, Dordrecht, 1997, pp. 225–275.
[41] V. I. Morariu, B. V. Srinivasan, V. C. Raykar, R. Duraiswami, and L. S. Davis, *Automatic online tuning for fast Gaussian summation*, in Advances in neural information processing systems, 2009, pp. 1113–1120.
[42] A. Onuki, *Phase transition dynamics*, Cambridge University Press, 2002.
[43] A. Plaza, J. A. Benediktsson, J. W. Boardman, J. Brazile, L. Bruzzone, G. Camps-Valls, J. Chanussot, M. Fauvel, P. Gamba, A. Gualtieri, et al., *Recent advances in techniques for hyperspectral image processing*, Remote sensing of environment, 113 (2009), pp. S110–S122.
[44] G. Plonka, D. Potts, G. Steidl, and M. Tasche, *Numerical Fourier Analysis*, Applied and Numerical Harmonic Analysis, Birkhäuser, 2018.
[45] D. Potts and G. Steidl, *Fast summation at nonequispaced knots by NFFTs*, SIAM Journal on Scientific Computing, 24 (2003), pp. 2013–2037.
[46] D. Potts, G. Steidl, and A. Nieslony, *Fast convolution with radial kernels at nonequispaced knots*, Numerische Mathematik, 98 (2004), pp. 329–351.
[47] N. Rasiwasia, J. Costa Pereira, E. Coviello, G. Doyle, G. R. Lanckriet, R. Levy, and N. Vasconcelos, *A new approach to cross-modal multimedia retrieval*, in Proceedings of the 18th ACM international conference on Multimedia, ACM, 2010, pp. 251–260.
[48] M. Stoll, *A literature survey of matrix methods for data science*, ArXiv e-prints, (2019). arXiv:1912.07896.
[49] S. H. Strogatz, *Exploring complex networks*, nature, 410 (2001), pp. 268–276.
[50] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images*, in Sixth international conference on computer vision (IEEE Cat. No. 98CH36271), IEEE, 1998, pp. 839–846.
[51] M. van Breukelen, R. P. W. Duin, D. M. J. Tax, and J. E. den Hartog, *Handwritten digit recognition by combined classifiers*, Kybernetika, 34 (1998), pp. 381–386.
[52] Y. Van Gennip, N. Guillen, B. Osting, and A. L. Bertozzi, *Mean curvature, threshold dynamics, and phase field theory on finite graphs*, Milan Journal of Mathematics, 82 (2014), pp. 3–65.
[53] U. von Luxburg, *A tutorial on spectral clustering*, Statistics and Computing, 17 (2007), pp. 395–416.
[54] Q. Wang and J. Zhang, *A data transfer fusion method for discriminating similar spectral classes*, Sensors, 16 (2016), p. 1895.
[55] Y. Wang, H. Song, and Y. Zhang, *Spectral-spatial classification of hyperspectral images using joint bilateral filter and graph cut based model*, Remote Sensing, 8 (2016), p. 748.
[56] A. A. Wheeler, W. J. Boettinger, and G. B. McFadden, *Phase-field model for isothermal phase transitions in binary alloys*, Physical Review A, 45 (1992), p. 7424.
[57] O. Wodo and B. Ganapathysubramanian, *Modeling morphology evolution during solvent-based fabrication of organic solar cells*, Computational Materials Science, 55 (2012), pp. 113–126.
[58] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, *Learning with local and global consistency*, in Advances in neural information processing systems, 2004, pp. 321–328.
[59] X. Zhu and A. B. Goldberg, *Introduction to semi-supervised learning*, Synthesis lectures on artificial intelligence and machine learning, 3 (2009), pp. 1–130.

Appendix A. Binary Allen–Cahn.

In 1958, John W. Cahn and John E. Hilliard introduced a phase-field approach to describe phase separation phenomena in general solutions [8]. The derivation of the partial differential equation, which is today called the Cahn–Hilliard equation, assumed mass conservation within the solution.

In 1979, after investigating metal alloys for several years, John W. Cahn and Sam Allen proposed a very similar approach. The today called Allen–Cahn equation [2], however, renounces the mass conservation condition of the Cahn–Hilliard equation. It describes the evolution of

a binary solution over time, i.e., a physical system consisting of two liquid components, which exhibits a phase separation behavior by a phase field model approach. Here, a scalar field $u\colon \Omega \to [-1,1] \subset \mathbb{R}$ describes one components concentration on a typically 3-dimensional domain Ω, where pure phases of the solution are represented by the values 1 and -1, respectively, and a value of 0 means a perfect mixture of both components.

In order to describe the evolution of the scalar field u as a function of time, the Ginzburg-Landau energy functional is defined as in (5.1). The parameter ϵ weights the gradient term proportionally, which represents the Dirichlet energy that penalizes strong concentration gradients. As there naturally exist large concentration gradients at the phase boundaries, this gradient term enforces a rounded shape of the regions, as the ratio between circumference and area of a region is minimal for a circle. The interface parameter ϵ also weights $\psi(u)$ inversely proportionally, which is a potential function that describes the chemical interaction energy at a single point $\boldsymbol{x} \in \Omega$ given its current concentration distribution $u(\boldsymbol{x})$. In order for the system to exhibit a phase separation behavior, the potential function $\psi(u)$ must have energetic minima at or close to the pure concentrations $u = 1$ and $u = -1$, respectively. While the original work [8] employs a logarithmic potential, in practice, often polynomial approximations such as

$$\psi(u) = \frac{1}{4}(u^2 - 1)^2, \tag{A.1}$$

are used. Standard solution methods for partial differential equations, such as the finite element method or the finite difference method, can be used to solve (5.2), given suitable initial conditions as well as boundary conditions. Here, the time derivative as well as the spatial derivative must be discretized. For the spatial discretization, the domain Ω must be triangularized, so that the equation is solved on a finite set of grid points instead of the continuous domain.

For binary classification tasks, the Allen–Cahn equation can be adapted by identifying the grid points from the spatial discretization of the physical domain Ω with vertices $x_i \in \mathcal{V}$ of a graph $\mathcal{G}$, cf. [5]. In doing so, one defines the concentration u of the phase-field description on the finite set of vertices by identifying u with a vector $\boldsymbol{u} \in \mathbb{R}^n$ with one entry per vertex x_i. The goal in the binary classification case is to identify vertices with a corresponding entry in $\boldsymbol{u}$ close to 1 with the first class and entries close to -1 with the second class, belonging to either pure component in the phase-field formulation.

In order to ensure the correct classification of the labeled data, an additional term is included in the Ginzburg-Landau energy functional (5.1). Inspired by applications of the Cahn–Hilliard equation in image inpainting, [5] suggests a penalty term of the form $\frac{1}{2}\omega(x_i)(f_i - u_i)^2$ for a least squares fit to the data with $x_i \in \mathcal{V}$, where $\omega(x_i)$ is some (usually large) constant ω_0 for pre-labeled vertices and 0 for unlabeled vertices. Furthermore, the entries f_i of the vector $\boldsymbol{f} \in \mathbb{R}^n$ are set to 1 and -1 for the first and second class, respectively, if the label is known, as well as to 0 for unknown labels. This way, a deviation in $\boldsymbol{u}$ from $\boldsymbol{f}$ gets penalized in the energy functional, that we seek to minimize. It can, however, still be energetically beneficial for $\boldsymbol{u}$ to deviate from $\boldsymbol{f}$ at some vertices for the sake of shorter interface lengths and thus a smaller Dirichlet energy. This way, the method even remains stable given few misclassified training samples. The choice of ω controls the trade-off between the classical Ginzburg-Landau

energy and the least squares term. Choosing ω too small bares the risk of underfitting while choosing it too large may cause overfitting to the pre-classified data. With this modification, the discretized Ginzburg-Landau functional becomes

$$\tilde{E}(\boldsymbol{u}) = \frac{\epsilon}{2}\boldsymbol{u}^\top \boldsymbol{L}_{\mathrm{sym}}\boldsymbol{u} + \frac{1}{\epsilon}\sum_{i=1}^{n}\frac{1}{4}(u_i^2-1)^2 + \frac{1}{2}\sum_{i=1}^{n}\omega(x_i)(f_i-u_i)^2, \tag{A.2}$$

where the continuous integrals become sums over the vertex set in the discrete graph setting and the potential ψ from (A.1) was inserted. This modified Ginzburg-Landau energy functional $\tilde{E}$ forms the basis for the semi-supervised classification technique considered in this work as it can be viewed as its loss function. Note, that this loss function contains the potential function term $\frac{1}{\epsilon}\sum_{i=1}^{n}\psi(u_i)$ in addition to a regularized least squares fit approach for semi-supervised learning on graphs that is frequently used in the literature [58, 1, 38].

In order to solve the Allen–Cahn equation (5.2) for the modified Ginzburg-Landau energy functional (A.2), this section presents a suitable convexity splitting approach for the binary classification setting. In this case, the numerical scheme treating the convex part of the Ginzburg-Landau functional implicitly and the concave part explicitly reads

$$\frac{\boldsymbol{u}^{l+1}-\boldsymbol{u}^l}{\Delta t} = -\frac{\partial E_1}{\partial u}(\boldsymbol{u}^{l+1}) + \frac{\partial E_2}{\partial u}(\boldsymbol{u}^l) \tag{A.3}$$

with iterates $\boldsymbol{u}^l \in \mathbb{R}^n$, time step size $\Delta t \in \mathbb{R}$ and index l for the time step.

A possible splitting for the modified Ginzburg-Landau functional $\tilde{E}(\boldsymbol{u})$ in (A.2) is presented in [5], where a productive 0 is inserted into $\tilde{E}(\boldsymbol{u})$ by adding and subtracting the convex function $\frac{c}{2}\boldsymbol{u}^\top\boldsymbol{u}$ with a suitable constant $c>0$. The resulting functional is then split such that

$$\tilde{E}_1(\boldsymbol{u}) = \frac{\epsilon}{2}\boldsymbol{u}^\top \boldsymbol{L}_{\mathrm{sym}}\boldsymbol{u} + \frac{c}{2}\boldsymbol{u}^\top\boldsymbol{u} \tag{A.4}$$

and

$$\tilde{E}_2(\boldsymbol{u}) = \frac{c}{2}\boldsymbol{u}^\top\boldsymbol{u} - \frac{1}{4\epsilon}\sum_{i=1}^{n}(u_i^2-1)^2 - \frac{1}{2}\sum_{i=1}^{n}\omega(x_i)(f_i-u_i)^2. \tag{A.5}$$

While $\tilde{E}_1$ is convex for all $\epsilon, c>0$, the parameters have to be chosen

$$c > \omega_0 + \frac{3\overline{u}^2-1}{\epsilon}, \quad \overline{u} = \max_{i=1,\dots,n}|u_i|,$$

to ensure the concavity of $-\tilde{E}_2$, cf. [5].

Inserting $\tilde{E}_1$ and $\tilde{E}_2$ into the iteration scheme (A.3) yields

$$\frac{\boldsymbol{u}^{l+1}-\boldsymbol{u}^l}{\Delta t} = -\epsilon\boldsymbol{L}_{\mathrm{sym}}\boldsymbol{u}^{l+1} - c\boldsymbol{u}^{l+1} + c\boldsymbol{u}^l - \frac{1}{\epsilon}\left((\boldsymbol{u}^l)^3 - \boldsymbol{u}^l\right) - \boldsymbol{\omega}\,(\boldsymbol{u}^l - \boldsymbol{f}),$$

where we set $\boldsymbol{\omega} := \mathrm{diag}\,(\omega(x_i)_{i=1}^n)$ and the power in $(\boldsymbol{u}^l)^3$ is to be understood elementwise. Now, the solution $\boldsymbol{u}$ is restricted to the ansatz $\boldsymbol{\Phi}_k\boldsymbol{v}$, where $\boldsymbol{\Phi}_k := (\boldsymbol{\phi}_1,\dots,\boldsymbol{\phi}_k) \in \mathbb{R}^{n\times k}$ is the

matrix of the eigenvectors $\boldsymbol{\phi}_r$ belonging to the k smallest eigenvalues $\lambda_1, \ldots, \lambda_k$ of $\boldsymbol{L}_{\text{sym}}$ and $\boldsymbol{v} \in \mathbb{R}^k$ is a coefficient vector. Since $\boldsymbol{L}_{\text{sym}}\boldsymbol{\Phi}_k = \boldsymbol{\Phi}_k\boldsymbol{\Lambda}_k$ with $\boldsymbol{\Lambda}_k = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_k) \in \mathbb{R}^{k\times k}$, we obtain

$$\frac{\boldsymbol{\Phi}_k\boldsymbol{v}^{l+1} - \boldsymbol{\Phi}_k\boldsymbol{v}^l}{\Delta t} = -\epsilon\boldsymbol{\Phi}_k\boldsymbol{\Lambda}_k\boldsymbol{v}^{l+1} - c\boldsymbol{\Phi}_k\boldsymbol{v}^{l+1} + c\boldsymbol{\Phi}_k\boldsymbol{v}^l - \frac{1}{\epsilon}((\boldsymbol{\Phi}_k\boldsymbol{v}^l)^3 - \boldsymbol{\Phi}_k\boldsymbol{v}^l) - \boldsymbol{\omega}\,(\boldsymbol{\Phi}_k\boldsymbol{v}^l - \boldsymbol{f}).$$

Multiplication from the left with $\boldsymbol{\Phi}_k^\top$, simplifications and rearranging finally yields the iteration rule

$$v_r^{l+1} = \frac{1}{1 + \epsilon\lambda_r\,\Delta t + c\Delta t}\left[\left(1 + c\,\Delta t + \frac{\Delta t}{\epsilon}\right) v_r^l - \frac{\Delta t}{\epsilon} b_r^l - \Delta t\, d_r^l\right],\ r = 1, \ldots, k, \tag{A.6}$$

with $\boldsymbol{b}^l = \epsilon^{-1}\boldsymbol{\Phi}_k^\top(\boldsymbol{\Phi}_k\boldsymbol{v}^l)^3 \in \mathbb{R}^k$ and $\boldsymbol{d}^l = \boldsymbol{\Phi}_k^\top\boldsymbol{\omega}(\boldsymbol{\Phi}_k\boldsymbol{v}^l - \boldsymbol{f}) \in \mathbb{R}^k$.