NFFT meets Krylov methods: Fast matrix-vector products for the graph Laplacian of fully connected networks

Dominik Alfke*, Daniel Potts†, Martin Stoll‡, and Toni Volkmer§

Abstract. The graph Laplacian is a standard tool in data science, machine learning, and image processing. The corresponding matrix inherits the complex structure of the underlying network and is in certain applications densely populated. This makes computations, in particular matrix-vector products, with the graph Laplacian a hard task. A typical application is the computation of a number of its eigenvalues and eigenvectors. Standard methods become infeasible as the number of nodes in the graph is too large. We propose the use of the fast summation based on the nonequispaced fast Fourier transform (NFFT) to perform the dense matrix-vector product with the graph Laplacian fast without ever forming the whole matrix. The enormous flexibility of the NFFT algorithm allows us to embed the accelerated multiplication into Lanczos-based eigenvalues routines or iterative linear system solvers and even consider other than the standard Gaussian kernels. We illustrate the feasibility of our approach on a number of test problems from image segmentation to semi-supervised learning based on graph-based PDEs. In particular, we compare our approach with the Nyström method. Moreover, we present and test an enhanced, hybrid version of the Nyström method, which internally uses the NFFT.

Key words. Graph Laplacian, Lanczos Method, Eigenvalues, Nonequispaced Fast Fourier Transform, Machine Learning

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

1. Introduction.

Graphs are a fundamental tool in the modeling of imaging and data science applications [44, 37, 2, 3, 15]. To apply graph-based techniques, individual data points in a data set or pixels of an image represent the vertex set or nodes V of the graph, and the edges indicate the relationship between the vertices. In a number of real-world examples, the graph is sparse in the sense that each vertex is only connected to a small number of other vertices, i.e., the graph affinity matrix is sparsely populated. In other applications, such as the mentioned data points or image pixels, the natural choice for the graph would be a fully connected graph, which is then reflected in dense matrices that represent the graph information. Naturally, if there is no underlying graph the most natural choice is the fully connected graph. As the eigenvectors of the corresponding graph Laplacian are crucial in reducing the complexity of the underlying problem or for the extraction of quantities of interest [4, 5, 37], it is important to compute them accurately and fast. If this matrix is sparse, numerical analysis has provided efficient tools based on the Lanczos process with sparse matrix-

*Technische Universität Chemnitz, Faculty of Mathematics, Chair of Scientific Computing, 09107 Chemnitz, Germany, (dominik.alfke@mathematik.tu-chemnitz.de)

†Technische Universität Chemnitz, Faculty of Mathematics, Chair of Applied Functional Analysis, 09107 Chemnitz, Germany, (daniel.potts@mathematik.tu-chemnitz.de)

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

§Technische Universität Chemnitz, Faculty of Mathematics, Chair of Applied Analysis, 09107 Chemnitz, Germany, (toni.volkmer@mathematik.tu-chemnitz.de)

vector products that can compute the eigeninformation efficiently. For complex interactions leading to dense matrices, these methods suffer from the high cost of the matrix-vector product.

Our goal is hence to obtain the eigeninformation despite the fact that the graph is fully connected and without any a priori reduction of the graph information. For this we rely on a Lanczos procedure based on [1]. This method needs to perform the matrix-vector product in a fast way and thus, evaluating all information, without ever fully assembling the graph matrices. In a similar fashion the authors in [5] utilize the well-known Nyström method to only work with partial information from the graph and only approximately represent the remaining parts. Such methods are well-known within the fast solution of integral equations and have found applicability within the data science community [9, 22]. The technique we present here is known as a fast summation method [31, 32] and is based on the nonequispaced fast Fourier transform (NFFT), see [18] and the references therein. We apply this method in the setting where the weights of the edges between the vertices are modelled by a Gaussian kernel function of medium to large scaling parameter, such that the Gaussian is not well-localized and most vertices interact with each other. For the case of a smaller scaling parameter and consequently a more localized Gaussian, we refer to [26], which is partially based on a technique presented [46] for Gaussian kernels. Moreover, we remark that the NFFT-based fast summation method considered in this paper does not only support Gaussians but can handle various other rotational invariant functions.

The remaining parts of this paper are structured as follows. In Section 2, we first introduce the graph Laplacian and discuss the matrix structure. In Section 3, we introduce the NFFT-based fast summation, which allows for computing fast matrix-vector products with the graph Laplacian. In Section 4, we then recall Krylov subspace methods and in particular the Lanczos method, which sits at the engine room of the numerical computations to obtain a small number of eigenvectors. We then show that the graph Laplacian provides the ideal environment to be used together with the NFFT-based fast summation, and we obtain the NFFT-based Lanczos method. In Section 5 we briefly discuss the Nyström method as a direct competitor to our approach. We improve and accelerate this method, creating a new hybrid Nyström-Gaussian-NFFT version, which incorporates the NFFT-based fast summation. In Section 6, we present comparisons between the NFFT-based Lanczos method, the Nyström method and the hybrid Nyström-Gaussian-NFFT method with the direct application of the Lanczos method for a dense, large-scale problem. Additionally, we illustrate on a number of exemplary applications, such as spectral clustering and semi-supervised learning, that our approach provides a convenient infrastructure to be used within many different schemes.

2. The graph Laplacian and fully connected graphs.

We consider an undirected graph $G = (V, E)$ with the vertex set $V = \{v_j\}_{j=1}^n$ and the edge set E, cf. [8] for more information. An edge $e \in E$ is a pair of nodes (v_j, v_i) with $v_j \neq v_i$ and $v_j, v_i \in V$. For weighted undirected graphs, such as the ones considered in this paper, we also have a weight function $w : V \times V \to \mathbb{R}$ with $w(v_j, v_i) = w(v_i, v_j)$ for all j, i. We assume further that the function is positive for existing edges and zero otherwise. The degree of the vertex $v_j \in V$ is defined as

$$d(v_j) = \sum_{v_i \in V} w(v_j, v_i).$$

Let $\mathbf{W}$, $\mathbf{D} \in \mathbb{R}^{n\times n}$ be the weight matrix and the diagonal degree matrix with entries $W_{ji} = w(v_j, v_i)$ and $D_{jj} = d(v_j)$. Since we do not permit graphs with loops, $\mathbf{W}$ is zero on the diagonal. Now the crucial tool for further investigations is the graph Laplacian $\mathbf{L}$ defined via

$$\mathbf{L}(v_j, v_i) = \begin{cases} d(v_j) & \text{if } v_j = v_i \\ -w(v_j, v_i) & \text{otherwise,} \end{cases}$$

i.e. $\mathbf{L} = \mathbf{D} - \mathbf{W}$. The matrix $\mathbf{L}$ is typically known as the *combinatorial graph Laplacian* and we refer to [44] for an excellent discussion of its properties. Typically its normalized form is employed for segmentation purposes and we obtain the normalized Laplacian as

$$\mathbf{L}_s = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}, \tag{2.1}$$

obviously a symmetric matrix. Another normalized Laplacian of nonsymmetric form is given by

$$\mathbf{L}_w = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W}.$$

For the purpose of this paper we focus on the symmetric normalized Laplacian $\mathbf{L}_s$ but everything we derive here can equally be applied to the nonsymmetric version, where we would then have to resort to nonsymmetric Krylov methods such as GMRES [36]. It is well known in the area of data science, data mining, image processing and so on that the smallest eigenvalues and its associated eigenvectors possess crucial information about the structure of the data and/or image [44, 47, 5]. For this we state an amazing property of the graph Laplacian $\mathbf{L}$ for a general vector $\mathbf{u} \in \mathbb{R}^n$ with n the dimension of $\mathbf{L}$

$$\mathbf{u}^T\mathbf{L}\mathbf{u} = \frac{1}{2}\sum_{j,i=1}^{n} W_{ji}(\mathbf{u}_j - \mathbf{u}_i)^2,$$

which, as was illustrated in [44], is equivalent to the objective function of the graph *RatioCut* problem. Intuitively, assuming the vector $\mathbf{u}$ to be equal to a constant on one part of the graph A and a different constant on the remaining vertices $\bar{A}$. In this case $\mathbf{u}^T\mathbf{L}\mathbf{u}$ only contains terms from the edges with vertices in both A and $\bar{A}$. Thus a minimization of $\mathbf{u}^T\mathbf{L}\mathbf{u}$ results in a minimal cut with respect to the edge weights across A and $\bar{A}$. Obviously, 0 is an eigenvalue of $\mathbf{L}$ and its normalized variants as $\mathbf{L1} = \mathbf{D1} - \mathbf{W1} = \mathbf{0}$ by the definitions of $\mathbf{D}$ and $\mathbf{W}$ with $\mathbf{1}$ being the vector of all ones. Additionally, spectral clustering techniques heavily rely on the computation of the smallest k eigenvectors [44] and recently semi-supervised learning based on PDEs on graphs introduced by Bertozzi and Flenner [5] utilizes a small number of such eigenvectors for a complexity reduction. It is therefore imperative to obtain efficient techniques to compute the eigenvalues and eigenvectors fast and accurately. Since we are interested in the k smallest eigenvalues of the matrix $\mathbf{L}_s = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ it is clear that we can compute the k largest postive eigenvalues of the matrix $\mathbf{A} := \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$. In case that the graph $G = (V, E)$ is sparse in the sense that every vertex is only connected to a small number of other vertices and thus the matrix $\mathbf{W}$ is sparse, we can utilize the whole arsenal of numerical algorithms for the computations of a small number of eigenvalues, namely the Lanczos process

[13], the Krylov-Schur method [41], or the Jacobi-Davidson algorithm [39]. In particular, the ARPACK library [21] in MATLAB via the `eigs` function is a recommended choice. So frankly speaking, in the case of a sparse and symmetric matrix $\mathbf{W}$ the eigenvalue problem is fast and the algorithms are very mature. Hence, we focus on the case of fully connected graphs meaning that the matrix $\mathbf{W}$ is considered dense.

The standard scenario for this case is that each node $v_j \in V$ corresponds to a data vector $\mathbf{v}_j \in \mathbb{R}^d$ and the weight matrix is constructed as

$$W_{ji} = w(v_j, v_i) = \begin{cases} \exp(-\|\mathbf{v}_j - \mathbf{v}_i\|^2 / \sigma^2) & \text{if } j \neq i, \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

with a scaling parameter σ. For example, approaches with this kind of graph Laplacian have become increasingly popular in image processing [34], where the data vectors $\mathbf{v}_j$ encode color information of image pixels via their color channels. The data point dimension may then be $d = 1$ for grayscale images and $d = 3$ for RGB images. Other applications may involve simple Cartesian coordinates for $\mathbf{v}_j$. While Equation (2.2) is derived from a Gaussian kernel function $K(\mathbf{y}) := \exp(-\|\mathbf{y}\|^2 / \sigma^2)$, other applications might call for different kernel functions like the "Laplacian RBF kernel" $K(\mathbf{y}) := \exp(-\|\mathbf{y}\| / \sigma)$, the multiquadric kernel $K(\mathbf{y}) := (\|\mathbf{y}\|^2 + c^2)^{1/2}$, or inverse multiquadric kernel $K(\mathbf{y}) := (\|\mathbf{y}\|^2 + c^2)^{-1/2}$ for a parameter $c > 0$, e.g. cf. Section 6.3. This means, the weight matrix may be of the form

$$W_{ji} = \begin{cases} K(\mathbf{v}_j - \mathbf{v}_i) & \text{if } j \neq i, \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

Often certain techniques are used to sparsify the Laplacian or otherwise reduce its complexity in order to apply the methods named above. In particular, sparsification has been proposed for the construction of preconditioners [40] for iterative solvers, which still require the efficient implementation of the matrix vector products. In image processing, this can be achieved by considering only patches or other reduced representations of the image [47]. However, this might drop crucial nonlocal information encoded in the full graph Laplacian [34, 12], which is why we want to avoid it here and focus on fully connected graphs with dense Laplacians.

3. NFFT-based fast summation. For eigenvalue computation as well as various other applications with the graph Laplacian, one needs to perform matrix-vector multiplications with the matrix $\mathbf{W}$ or the matrix $\mathbf{A} := \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$. In general, this requires $\mathcal{O}(n^2)$ arithmetic operations. When the matrix $\mathbf{W}$ has entries (2.2), this arithmetic complexity can be reduced to $\mathcal{O}(n)$ using the NFFT-based fast summation [31, 32]. In general, this method may be applied when the entries of the matrix $\mathbf{W}$ can be written in the form $W_{ji} = K(\mathbf{v}_j - \mathbf{v}_i)$, where $K\colon \mathbb{R}^d \to \mathbb{C}$ is a rotational invariant and smooth kernel function. For applying the NFFT-based fast summation for (2.3), it would be more convenient to consider the matrix $\mathbf{W}$ to have entries equal to $K(\mathbf{0})$ on the diagonal and we refer to this matrix as $\tilde{\mathbf{W}}$. Note that it can be written as $\tilde{\mathbf{W}} = \mathbf{W} + K(\mathbf{0})\,\mathbf{I}$ and thus $\mathbf{W} = \tilde{\mathbf{W}} - K(\mathbf{0})\,\mathbf{I}$. In order to efficiently compute the row sums of $\mathbf{W}$, which appear on the diagonal of $\mathbf{D}$, we use

$$\mathbf{W}\mathbf{1} = \tilde{\mathbf{W}}\mathbf{1} - K(\mathbf{0})\,\mathbf{I}\mathbf{1} = \tilde{\mathbf{W}}\mathbf{1} - K(\mathbf{0})\,\mathbf{1}.$$

We now illustrate how to efficiently compute the matrix-vector product with the matrix $\tilde{\mathbf{W}}$ using the NFFT-based fast summation. For instance, for the Gaussian kernel function, we have

$$\left(\tilde{\mathbf{W}}\mathbf{x}\right)_j = \sum_{i=1}^{n} x_i \exp\left(-\frac{\|\mathbf{v}_j - \mathbf{v}_i\|^2}{\sigma^2}\right) \quad \forall j = 1, \ldots, n \tag{3.1}$$

with $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ and we rewrite (3.1) by

$$\left(\tilde{\mathbf{W}}\mathbf{x}\right)_j = f(\mathbf{v}_j) := \sum_{i=1}^{n} x_i \, K(\mathbf{v}_j - \mathbf{v}_i) \tag{3.2}$$

with the kernel function $K(\mathbf{y}) := \exp(-\|\mathbf{y}\|^2/\sigma^2)$. The key idea of the efficient computation of (3.2) is approximating K by a trigonometric polynomial K_{RF} in order to separate the computations involving the vertices $\mathbf{v}_j$ and $\mathbf{v}_i$. Assuming we have such a d-variate trigonometric polynomial

$$K(\mathbf{y}) \approx K_{\mathrm{RF}}(\mathbf{y}) := \sum_{\mathbf{l} \in I_N} \hat{b}_{\mathbf{l}} \, \mathrm{e}^{2\pi \mathrm{i} \mathbf{l} \mathbf{y}}, \quad I_N := \{-N/2, -N/2+1, \ldots, N/2-1\}^d, \tag{3.3}$$

with bandwidth $N \in 2\mathbb{N}$ and Fourier coefficients $\hat{b}_{\mathbf{l}}$, we replace K by K_{RF} in (3.2) and we obtain

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

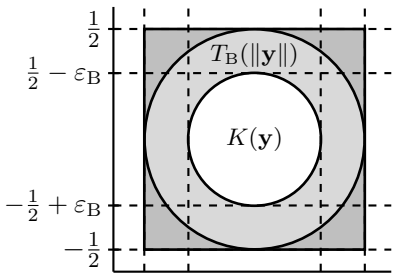
Using the NFFT [18], one computes the inner and outer sums for all $j = 1, \ldots, n$ totally in $\mathcal{O}(m^d \, n + N^d \log N)$ arithmetic operations, where $m \in \mathbb{N}$ is an internal window cut-off parameter which influences the accuracy of the NFFT. Please note that since K_{RF} and f_{RF} are 1-periodic functions but neither K nor f are, ones needs to shift and scale the nodes $\mathbf{v}_j$ such that they are contained in a subset of the cube $[-1/4, 1/4]^d$ ensuring $\mathbf{v}_j - \mathbf{v}_i \in [-1/2, 1/2]^d$. Depending on the Fourier coefficients $\hat{b}_{\mathbf{l}}$, $\mathbf{l} \in I_N$, of the trigonometric polynomial K_{RF}, where $\hat{b}_{\mathbf{l}}$ still have to be determined, we may need to scale the nodes $\mathbf{v}_j$ to a slightly smaller cube.

We emphasize that we are not restricted to the Gaussian weight function $w(v_j, v_i) = \exp(-\|\mathbf{v}_j - \mathbf{v}_i\|^2/\sigma^2)$ or a rotational invariant weight function. In fact, any kernel function K that can be well approximated by a trigonometric polynomial K_{RF} may be used.

Next, we describe an approach to obtain suitable Fourier coefficients $\hat{b}_{\mathbf{l}}$ of K_{RF} based on sampling values of K. Especially, we want to obtain a good approximation of K using a small number of Fourier coefficients $\hat{b}_{\mathbf{l}}$. Therefore, we regularize K to obtain a 1-periodic smooth kernel function K_{R}, which is $p-1$ times continuously differentiable (in the periodic setting), such that its Fourier coefficients decay in a fast way. Then, we approximate the Fourier coefficients of K_{R} using the trapezoidal rule and this yields the Fourier coefficients $\hat{b}_{\mathbf{l}}$ of K_{RF}.

For a rotational invariant kernel function $K(\mathbf{y})$, which is sufficiently smooth except at the "boundaries" of the cube $[-1/2, 1/2]^d$, e.g. $K(\mathbf{y}) = \exp(-\|\mathbf{y}\|^2/\sigma^2)$, we only need to regularize near $\|\mathbf{y}\| = 1/2$. We use the ansatz

$$K_{\mathrm{R}}(\mathbf{y}) := \begin{cases} K(\mathbf{y}) & \text{if } \|\mathbf{y}\| \le \frac{1}{2} - \varepsilon_{\mathrm{B}} \\ T_{\mathrm{B}}(\|\mathbf{y}\|) & \text{if } \frac{1}{2} - \varepsilon_{\mathrm{B}} < \|\mathbf{y}\| \le \frac{1}{2}, \\ T_{\mathrm{B}}\left(\frac{1}{2}\right) & \text{otherwise,} \end{cases}$$



where T_{B} is a suitably chosen univariate polynomial, e.g. computed by a two-point Taylor interpolation. The parameter $0 < \varepsilon_{\mathrm{B}} \ll 1/2$ determines the size of the regularization region, cf. [32, Sec. 2]. For the treatment of a rotational invariant kernel function which has a singularity at the origin, we also refer to [32, Sec. 2]. Now we approximate K_{R} by the d-variate trigonometric polynomial K_{RF} from (3.3), where we compute the Fourier coefficients

$$\hat{b}_{\mathbf{l}} := \frac{1}{N} \sum_{\mathbf{j} \in I_N} K_{\mathrm{R}}\left(\frac{\mathbf{j}}{N}\right) \mathrm{e}^{-2\pi \mathrm{i} \mathbf{j}\mathbf{l}/N} \quad \forall \mathbf{l} \in I_N. \tag{3.4}$$

Assuming one evaluation of K_{R} takes $\mathcal{O}(1)$ arithmetic operations, the computations in (3.4) require $\mathcal{O}(N^d \log N)$ arithmetic operations in total using the fast Fourier transform.

If all vertices v_j and their corresponding data vectors $\mathbf{v}_j \in \mathbb{R}^d$, $j = 1, \ldots, n$, fulfill the property $\|\mathbf{v}_j\| \le 1/4 - \varepsilon_{\mathrm{B}}/2$, we have $\|\mathbf{v}_j - \mathbf{v}_i\| \le 1/2 - \varepsilon_{\mathrm{B}}$ and we obtain an approximation of (3.2) by

$$\left(\tilde{\mathbf{W}}\mathbf{x}\right)_j = f(\mathbf{v}_j) = f_{\mathrm{R}}(\mathbf{v}_j) := \sum_{i=1}^{n} x_i K_{\mathrm{R}}(\mathbf{v}_j - \mathbf{v}_i) \approx f_{\mathrm{RF}}(\mathbf{v}_j) := \sum_{i=1}^{n} x_i K_{\mathrm{RF}}(\mathbf{v}_j - \mathbf{v}_i).$$

Otherwise, we compute a correction factor $\rho := (1/4 - \varepsilon_{\mathrm{B}}/2)/\max_{j=1,\ldots,n} \|\mathbf{v}_j\|$, using transformed vertices $\tilde{\mathbf{v}}_j := \mathbf{v}_j \rho$, and adjust parameters of the kernel function appropriately. For instance, in case of the Gaussian kernel function, we replace the scaling parameter σ by $\tilde{\sigma} := \sigma\rho$ for the regularized kernel function K_{R}.

The error of the approximation $f(\mathbf{v}_j) := (\tilde{\mathbf{W}}\mathbf{x})_j \approx f_{\mathrm{RF}}(\mathbf{v}_j)$ depends on the kernel function as well as on the choice of the regularization smoothness p, the size of the regularization region ε_{B}, the bandwidth N, and the window cut-off parameter m. For a fixed accuracy, we fix these parameters p, ε_{B}, N, and m. Hence, for small to medium dimensions d, we obtain a fast approximate algorithm for the matrix-vector multiplication $\tilde{\mathbf{W}}\mathbf{x}$ of complexity $\mathcal{O}(n)$, cf. Algorithm 3.1. This algorithm is implemented as `applications/fastsum` and `matlab/fastsum` in C and MATLAB within the NFFT3 software library[1], see also [18], and we use the default Kaiser-Bessel window function. In Figure 1, we list the relevant control parameters of Algorithm 3.1 and regularization approach (3.4).

[1] https://www.tu-chemnitz.de/~potts/nfft/

Algorithm 3.1 Fast approximate matrix-vector multiplication $\tilde{\mathbf{W}}\mathbf{x}$ using NFFT-based fast summation, $(\tilde{\mathbf{W}}\mathbf{x})_j = \sum_{i=1}^n x_i\, K(\mathbf{v}_j - \mathbf{v}_i)$, e.g. $(\tilde{\mathbf{W}}\mathbf{x})_j = \sum_{i=1}^n x_i \exp(-\|\mathbf{v}_j - \mathbf{v}_i\|^2/\sigma^2)$, $\forall j = 1,\dots,n$.

Input:

$\left(\hat{b}_{\mathbf{l}}\right)_{\mathbf{l}\in I_N}$	Fourier coefficients of trigonometric polynomial K_{RF} which approximates $K(\mathbf{y})$ for $\mathbf{y}\in\mathbb{R}^d$, $\|\mathbf{y}\| \le 1/2 - \varepsilon_{\mathrm{B}}$, e.g. obtained by (3.4),
$\{\mathbf{v}_j\}_{j=1}^n$	vertex set, $\mathbf{v}_j \in \mathbb{R}^d$, $\|\mathbf{v}_j\| \le 1/4 - \varepsilon_{\mathrm{B}}/2$,
$\mathbf{x} = [x_1, x_2, \dots, x_n]^T$	vector $\in \mathbb{R}^n$.

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

Output: $\left[\tilde{f}_{\mathrm{RF}}(\mathbf{v}_j)\right]_{j=1,\dots,n}$ $\quad \tilde{f}_{\mathrm{RF}}(\mathbf{v}_j) \approx (\tilde{\mathbf{W}}\mathbf{x})_j \quad \forall j = 1,\dots,n.$

Complexity: $\mathcal{O}(n)$ for fixed accuracy.

Parameter	Description
N	$\in 2\mathbb{N}$ bandwidth (in each dimension) of trigonometric polynomial, such that $K_{\mathrm{RF}} \approx K$
m	$\in \mathbb{N}$ window cut-off parameter of NFFT ($m = 8$ gives approximately IEEE double precision for default Kaiser-Bessel window)
p	$\in \mathbb{N}$ regularization smoothness for K_R (default choice $p = m$)
ε_{B}	size of the regularization region, $0 \le \varepsilon_{\mathrm{B}} \ll 1/2$ (default choice $\varepsilon_{\mathrm{B}} = p/N$)

Figure 1. *Control parameters for NFFT-based fast summation.*

Note that every part of Algorithm 3.1 is deterministic and linear in the input vector $\mathbf{x}$, i.e., the algorithm constitutes a linear operator that can be written as $\tilde{\mathbf{W}} + \mathbf{E}$ with an error matrix $\mathbf{E}$. For theoretical error estimates on $\|\mathbf{E}\mathbf{x}\|_\infty = \max_j |f(\mathbf{v}_j) - f_{\mathrm{RF}}(\mathbf{v}_j)|$, we refer to [31, 32, 19]. The basic idea is to start with the estimate

$$|f(\mathbf{v}_j) - f_{\mathrm{RF}}(\mathbf{v}_j)| \le \|\mathbf{x}\|_1\, \|K_{\mathrm{ERR}}\|_\infty, \quad \|K_{\mathrm{ERR}}\|_\infty := \max_{\mathbf{y}\in\mathbb{R}^d, \|\mathbf{y}\|\le 1/2-\varepsilon_{\mathrm{B}}} |K(\mathbf{y}) - K_{\mathrm{RF}}(\mathbf{y})|, \tag{3.5}$$

caused by the approximation of the kernel K by a trigonometric polynomial K_{RF}, and to

additionally take the errors caused by the NFFT into account. In practice, one may guess $\|K_{\mathrm{ERR}}\|_\infty$ based on sampling values of K and K_{RF}. For theoretical error estimates of the NFFT for various window functions, we refer to [18, 27]. In practice, choosing the window cut-off parameter of the NFFT $m = 8$ yields approximately IEEE double precision for the default Kaiser-Bessel window, see e.g. [18, Sec. 5.2].

We again emphasize that Algorithm 3.1 is not restricted to the Gaussian kernel function. Any kernel function K that can be well approximated by a trigonometric polynomial may be used and the corresponding Fourier coefficients $\hat{b}_{\mathbf{l}}$, $\mathbf{l} \in I_N$, are an input parameter of Algorithm 3.1.

Moreover, for the Gaussian kernel function, one could also use the analytic Fourier coefficients $\hat{b}_{\mathbf{l}}$ from [19] for small values of the scaling parameter σ instead of computing $\hat{b}_{\mathbf{l}}$ by interpolation in (3.4). In this case, explicit error bounds for $\|K_{\mathrm{ERR}}\|_\infty$ are available.

3.1. Error propagation for normalized matrices. As seen in Section 2, many applications involving the Graph Laplacian require matrix vector products with a matrix $\mathbf{A}$ that itself does not follow the form of (2.3), but results from normalization of such a matrix $\mathbf{W}$. This normalization can be written as $\mathbf{A} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$, where $\mathbf{D} = \mathrm{diag}(\mathbf{W1})$. Since our approach includes replacing all matrix-vector products $\mathbf{Wx}$ by the approximations $(\tilde{\mathbf{W}} + \mathbf{E})\mathbf{x} - K(\mathbf{0})\,\mathbf{x}$, this also includes the computation of the degree matrix $\mathbf{D}$. The error occurring from this approximation will then propagate to the evaluation error of $\mathbf{Ax}$.

Algorithm 3.2 summarizes the usage of Algorithm 3.1 for this case. Note that if multiple matrix-vector products are required, e.g. in an iterative scheme, steps 1–4 can be performed once in a setup phase. The following lemma gives an estimation of the error of Algorithm 3.2 depending on the relative error of Algorithm 3.1.

Lemma 3.1. *Let $\mathbf{W} \in \mathbb{R}^{n\times n}$ be a matrix with non-negative entries and at least one positive entry per row. Given an error matrix $\mathbf{E} \in \mathbb{R}^{n\times n}$, we define $\mathbf{W_E} = \mathbf{W} + \mathbf{E}$ and*

$$\begin{aligned} [d_1,\dots,d_n]^T &:= \mathbf{W1}, & \mathbf{D} &:= \mathrm{diag}(d_1,\dots,d_n), & \mathbf{A} &:= \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2},\\ [d_{\mathbf{E},1},\dots,d_{\mathbf{E},n}]^T &:= \mathbf{W_E1}, & \mathbf{D_E} &:= \mathrm{diag}(d_{\mathbf{E},1},\dots,d_{\mathbf{E},n}) & \mathbf{A_E} &:= \mathbf{D_E}^{-1/2}\mathbf{W_E}\mathbf{D_E}^{-1/2}. \end{aligned}$$

Let $d_{\min} > 0$ denote the minimum diagonal entry of $\mathbf{D}$ and furthermore set

$$\eta := \frac{d_{\min}}{\|\mathbf{W}\|_\infty} \qquad \text{and} \qquad \varepsilon := \frac{\|\mathbf{E}\|_\infty}{\|\mathbf{W}\|_\infty}.$$

Then, for $\varepsilon < \eta$, it holds

$$\|\mathbf{A} - \mathbf{A_E}\|_\infty \le \frac{\varepsilon(1+\eta)}{\eta(\eta-\varepsilon)}.$$

Proof. Due to

$$|d_i - d_{\mathbf{E},i}| \le \|\mathbf{W1} - \mathbf{W_E1}\|_\infty = \|\mathbf{E1}\|_\infty \le \|\mathbf{E}\|_\infty\|\mathbf{1}\|_\infty = \|\mathbf{E}\|_\infty = \varepsilon\|\mathbf{W}\|_\infty$$

Algorithm 3.2 Fast approximate matrix-vector multiplication $\mathbf{Ax}$ using NFFT-based fast summation, with $\mathbf{A} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$, $\mathbf{W}$ as in (2.3) and $\mathbf{D} = \operatorname{diag}(\mathbf{W1})$

Input: σ or c, $\{\mathbf{v}_j\}_{j=1}^n$ Scaling parameter and vertex set, $\mathbf{v}_j \in \mathbb{R}^d$, specifying $\mathbf{W}$,
$\mathbf{x} = [x_1, \ldots, x_n]^T$ vector $\in \mathbb{R}^n$.

1. Choose correction factor ρ such that $\|\rho\mathbf{v}_j\|_2 \leq 1/4 - \varepsilon_B/2$ for all $j = 1, \ldots, n$.
   Set $\mathbf{v}_j := \rho\mathbf{v}_j$ for all $j = 1, \ldots, n$.
2. For Gaussian and Laplacian RBF kernel, adjust scaling parameter $\sigma := \rho\sigma$.
   For multiquadric and inverse multiquadric kernel, adjust parameter $c := c/\rho$.
3. For the computation of matrix-vector products with the matrix

$$\tilde{\mathbf{W}}_{\mathbf{E}} = \tilde{\mathbf{W}} + \mathbf{E} = K(\mathbf{0})\,\mathbf{I} + \mathbf{W} + \mathbf{E}$$

   by Algorithm 3.1, determine appropriate control parameters for the NFFT-based fast summation, see Figure 1, and obtain Fourier coefficients $\hat{b}_{\mathbf{l}}$, $\mathbf{l} \in I_N$, e.g. by (3.4) or [19].
4. Compute $\mathbf{D}_{\mathbf{E}} = \operatorname{diag}(\tilde{\mathbf{W}}_{\mathbf{E}}\mathbf{1} - K(\mathbf{0})\,\mathbf{1}) \approx \operatorname{diag}(\mathbf{W1}) = \mathbf{D}$ via Algorithm 3.1
   (scale output of Algorithm 3.1 by ρ for multiquadric kernel and $1/\rho$ for inverse multiquadric kernel).
5. Compute $\mathbf{y} = \mathbf{D}_{\mathbf{E}}^{-1/2}\left(\tilde{\mathbf{W}}_{\mathbf{E}}(\mathbf{D}_{\mathbf{E}}^{-1/2}\mathbf{x}) - K(\mathbf{0})\,\mathbf{D}_{\mathbf{E}}^{-1/2}\mathbf{x}\right) \approx \mathbf{Ax}$ via Algorithm 3.1
   (scale output of Algorithm 3.1 by ρ for multiquadric kernel and $1/\rho$ for inverse multiquadric kernel).

Output: $\mathbf{y}$ Approximate result of $\mathbf{Ax}$.

Complexity: $\mathcal{O}(n)$ for fixed accuracy.

and the fact that $x \mapsto x^{-1/2}$ and its first derivative are monotoneously decreasing, we obtain

$$\begin{aligned}
\|\mathbf{D}^{-1/2} - \mathbf{D}_{\mathbf{E}}^{-1/2}\|_\infty &= \max_i \left|d_i^{-1/2} - d_{\mathbf{E},i}^{-1/2}\right| \\
&\leq \max_i \max_{-\|\mathbf{E}\|_\infty \leq \delta \leq \|\mathbf{E}\|_\infty} \left|d_i^{-1/2} - (d_i + \delta)^{-1/2}\right| \\
&= \max_i \left|d_i^{-1/2} - (d_i - \|\mathbf{E}\|_\infty)^{-1/2}\right| \\
&= \left|d_{\min}^{-1/2} - (d_{\min} - \|\mathbf{E}\|_\infty)^{-1/2}\right| \\
&= \left|\eta^{-1/2} - (\eta - \varepsilon)^{-1/2}\right| \, \|\mathbf{W}\|_\infty^{-1/2} \\
&= \left((\eta - \varepsilon)^{-1/2} - \eta^{-1/2}\right)\|\mathbf{W}\|_\infty^{-1/2}.
\end{aligned}$$

Analogously we obtain

$$\|\mathbf{D}_{\mathbf{E}}^{-1/2}\|_\infty \leq (\eta - \varepsilon)^{-1/2}\|\mathbf{W}\|_\infty^{-1/2}.$$

Together with $\|\mathbf{D}^{-1/2}\|_\infty = \eta^{-1/2}\|\mathbf{W}\|_\infty^{-1/2}$ and $\|\mathbf{W_E}\|_\infty \le (1+\varepsilon)\|\mathbf{W}\|_\infty$, this yields

$$\begin{aligned}
\|\mathbf{A}-\mathbf{A_E}\|_\infty &= \Big\| (\mathbf{D}^{-1/2}-\mathbf{D_E}^{-1/2})\mathbf{W}\mathbf{D}^{-1/2} \\
&\qquad + \mathbf{D_E}^{-1/2}(\mathbf{W}-\mathbf{W_E})\mathbf{D}^{-1/2} \\
&\qquad\quad + \mathbf{D_E}^{-1/2}\mathbf{W_E}(\mathbf{D}^{-1/2}-\mathbf{D_E}^{-1/2})\Big\|_\infty \\
&\le \Big((\eta-\varepsilon)^{-1/2}-\eta^{-1/2}\Big)\|\mathbf{W}\|_\infty^{-1/2}\,\|\mathbf{W}\|_\infty\,\eta^{-1/2}\|\mathbf{W}\|_\infty^{-1/2} \\
&\qquad + (\eta-\varepsilon)^{-1/2}\|\mathbf{W}\|_\infty^{-1/2}\,\varepsilon\|\mathbf{W}\|_\infty\,\eta^{-1/2}\|\mathbf{W}\|_\infty^{-1/2} \\
&\qquad\quad + (\eta-\varepsilon)^{-1/2}\|\mathbf{W}\|_\infty^{-1/2}\,(1+\varepsilon)\|\mathbf{W}\|_\infty\,\Big((\eta-\varepsilon)^{-1/2}-\eta^{-1/2}\Big)\|\mathbf{W}\|_\infty^{-1/2} \\
&= \Big((\eta-\varepsilon)^{-1/2}-\eta^{-1/2}\Big)\Big(\eta^{-1/2}+(1+\underline{\varepsilon)(\eta-\varepsilon)^{-1/2}}\Big) + \underline{(\eta-\varepsilon)^{-1/2}\varepsilon}\eta^{-1/2}.
\end{aligned}$$

Now detach the left underlined part from its paranthesed expression and combine it with the right underlined part:

$$\begin{aligned}
&= \Big((\eta-\varepsilon)^{-1/2}-\eta^{-1/2}\Big)\Big((\eta-\varepsilon)^{-1/2}+\eta^{-1/2}\Big) \\
&\qquad + \underline{\varepsilon(\eta-\varepsilon)^{-1/2}}\Big((\eta-\varepsilon)^{-1/2}-\eta^{-1/2}+\eta^{-1/2}\Big)
\end{aligned}$$

Resolve the binomial expression in the first line and simplify the second line:

$$= (\eta-\varepsilon)^{-1}-\eta^{-1}+\varepsilon(\eta-\varepsilon)^{-1} = \frac{(1+\varepsilon)\eta-(\eta-\varepsilon)}{\eta(\eta-\varepsilon)} = \frac{\varepsilon(1+\eta)}{\eta(\eta-\varepsilon)}.$$

This concludes the proof for the desired inequality. ∎

The requirement $\varepsilon < \eta$ means that $\|\mathbf{E}\|_\infty$ must be smaller than the smallest diagonal entry in $\mathbf{D}$. This condition cannot be avoided since otherwise, negative entries in $\mathbf{D_E}$ could not be ruled out, leading to imaginary entries in $\mathbf{D_E}^{-1/2}$ and thus in $\mathbf{A_E}$. On the other hand, if ε is well below η, Lemma 3.1 yields that the absolute error in $\mathbf{A}$ is linear in ε, which is the relative error of Algorithm 3.1.

Alternatively, by ignoring the error caused by the NFFT, we obtain error estimations of the form

$$\|\mathbf{E}\mathbf{x}\|_\infty \lesssim \|K_{\mathrm{ERR}}\|_\infty\|\mathbf{x}\|_1 \le n\,\|K_{\mathrm{ERR}}\|_\infty\|\mathbf{x}\|_\infty \qquad \Rightarrow \qquad \varepsilon = \frac{\|\mathbf{E}\|_\infty}{\|\mathbf{W}\|_\infty} \lesssim n\,\frac{\|K_{\mathrm{ERR}}\|_\infty}{\|\mathbf{W}\|_\infty}. \tag{3.6}$$

In other words, the perturbation grows linearly in the size of the dataset. If either $\|\mathbf{W}\|_\infty$ or $d_{\min}$ grew less fast, then Lemma 3.1 would not be applicable for large n because ε would eventually supersede η. However, if we assume that increasing n means adding more similarly-distributed data points to the dataset, the *average* entry in $\mathbf{W}$ does not change and thus all row sums of $\mathbf{W}$ also grow linearly in n, including $d_{\min}$ and the maximum row sum $\|\mathbf{W}\|_\infty$. A mathematical quantification of this observation is beyond the scope of this article, but in practice, the values for η and ε can be approximated and monitored to give a-posteriori error

bounds. One way to do this is by using (3.6) and approximating $\|K_{\mathrm{ERR}}\|_\infty$ via (3.5), where the maximum can be discretized in a large number of randomly drawn sample points. The accuracy of this approximation can be validated by explicitly computing the exact absolute row sum $\|\mathbf{E}\|_\infty$ via

$$\|\mathbf{E}\|_\infty = \left\| \sum_{i=1}^{n} |\mathbf{E}\mathbf{e}_i| \right\|_\infty = \left\| \sum_{i=1}^{n} \left| \tilde{\mathbf{W}}_{\mathbf{E}}\mathbf{e}_i - \mathbf{W}\mathbf{e}_i - K(\mathbf{0})\mathbf{e}_i \right| \right\|_\infty , \tag{3.7}$$

where $|\cdot|$ is applied elementwise, $\mathbf{e}_i$ denotes the i-th unit vector, and matrix-vector products with $\tilde{\mathbf{W}}_{\mathbf{E}} = \mathbf{W} + K(\mathbf{0})\mathbf{I} + \mathbf{E}$ are evaluated using Algorithm 3.1. The effort of computing (3.7) is $\mathcal{O}(n^2)$. Equivalently, the true value for $\|\mathbf{A} - \mathbf{A}_{\mathbf{E}}\|_\infty$ can be computed via

$$\|\mathbf{A} - \mathbf{A}_{\mathbf{E}}\|_\infty = \left\| \sum_{i=1}^{n} |\mathbf{A}\mathbf{e}_i - \mathbf{A}_{\mathbf{E}}\mathbf{e}_i| \right\|_\infty .$$

4. Krylov subspace methods and NFFT. The main contribution of this paper is the usage of NFFT-based fast summation for accelerating Krylov subspace methods, which are the state-of-the-art schemes for the solution of linear equation systems, eigenvalue problems, and more [35]. In the case of large dense matrices, the computational bottleneck is the setup of and multiplication with the system matrix itself. We will here exemplarily illustrate this for the Lanczos algorithm [20], which is the standard method for computation of a few dominating, i.e. largest, eigenvalues of a symmetric matrix $\mathbf{A}$ [30, 13]. It is based on looking for an $\mathbf{A}$-invariant subspace in the Krylov space

$$\mathcal{K}_k(\mathbf{A}, \mathbf{r}) = \operatorname{span}\left\{ \mathbf{r}, \mathbf{A}\mathbf{r}, \mathbf{A}^2\mathbf{r}, \mathbf{A}^3\mathbf{r}, \ldots, \mathbf{A}^{k-1}\mathbf{r} \right\}.$$

This is achieved by iteratively constructing an orthonormal basis $\mathbf{q}_1, \ldots, \mathbf{q}_k$ of this space in such a way that the matrix $\mathbf{Q}_k = [\mathbf{q}_1, \ldots, \mathbf{q}_k] \in \mathbb{R}^{n \times k}$ yields a tridiagonalization of $\mathbf{A}$, i.e.

$$\mathbf{Q}_k^T \mathbf{A} \mathbf{Q}_k = \mathbf{T}_k = \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_k \\ & & \beta_k & \alpha_k \end{bmatrix}.$$

Such a matrix $\mathbf{Q}_k$ as well as the entries of $\mathbf{T}_k$ can be computed by the iteration

$$\mathbf{q}_1 = \frac{\mathbf{r}}{\|\mathbf{r}\|}, \qquad \mathbf{q}_{k+1} = \frac{1}{\beta_{k+1}} \left(\mathbf{A}\mathbf{q}_k - \alpha_k \mathbf{q}_k - \beta_k \mathbf{q}_{k-1} \right) \quad \forall k = 1, 2, \ldots$$

where $\alpha_k = \mathbf{q}_k^T \mathbf{A} \mathbf{q}_k$ and $\beta_{k+1} = \|\mathbf{A}\mathbf{q}_k - \alpha_k \mathbf{q}_k - \beta_k \mathbf{q}_{k-1}\|$. The remarkable fact that this iteration produces orthonormal vectors is a consequence of the symmetry of $\mathbf{A}$. We now summarize the first k steps of the Lanczos process in the relation

$$\mathbf{A}\mathbf{Q}_k = \mathbf{Q}_k \mathbf{T}_k + \beta_{k+1} \mathbf{q}_{k+1} \mathbf{e}_k^T , \tag{4.1}$$

where $\mathbf{e}_j$ denotes the j-th standard basis vector of the appropriate dimension. The eigenvalues and eigenvectors of the small matrix $\mathbf{T}_k$ are called the Ritz values and vectors, respectively, and can be computed efficiently. From $\mathbf{T}_k\mathbf{w} = \lambda\mathbf{w}$ we then obtain

$$\mathbf{A}\mathbf{Q}_k\mathbf{w} = \mathbf{Q}_k\mathbf{T}_k\mathbf{w} + \beta_{k+1}\mathbf{q}_{k+1}\mathbf{e}_k^T\mathbf{w} = \lambda\mathbf{Q}_k\mathbf{w} + \beta_{k+1}w_k\mathbf{q}_{k+1},$$

where w_k is the k-th component of the Ritz vector $\mathbf{w}$. We finally see via

$$\|\mathbf{A}\mathbf{Q}_k\mathbf{w} - \lambda\mathbf{Q}_k\mathbf{w}\| = |\beta_{k+1}\mathbf{w}_k| \leq |\beta_{k+1}|$$

that a small value $|\beta_{k+1}|$ indicates that $(\lambda, \mathbf{Q}_k\mathbf{w})$ is a good approximation to an eigenpair of $\mathbf{A}$ and that the Krylov space is close to containing an $\mathbf{A}$-invariant subspace. There are many more practical issues that make the implementation of the Lanczos process more efficient and robust. We do not discuss these points in detail but refer to [30, 21] for the details.

Additionally, we want to point out that the above procedure can also be used for the solution of linear systems of equations. Standard methods based on the Lanczos method are the conjugate gradients method [16] and the minimal residual method [29], which are tailored for the solution of linear systems of the form $\mathbf{A}\mathbf{x} = \mathbf{b}$. Note that such applications involving the graph Laplacian are commonly found in kernel based methods [6]. In the nonsymmetric case that comes up e.g. when considering $\mathbf{L}_w$, we can employ the Arnoldi method [35], which relies on a similar iteration where $\mathbf{T}_k$ is replaced by an upper Hessenberg matrix.

One main contribution of this paper is the fact that by evaluating matrix-vector products via the NFFT-based Algorithms 3.1 or 3.2, Krylov subspace methods are still applicable for dense matrices that are too large to store, let alone apply, as long as they stem from the kernel structure of (2.3) or normalization of such a matrix. In our experiments, this method will be denoted as NFFT-based Lanczos method.

A detailed discussion of the effect of inexact matrix-vector products on Krylov-based approximations can be found in [38].

5. Alternative eigenvalue algorithm: The Nyström method.

5.1. The traditional Nyström extension. The Nyström extension is currently used as a method of choice to compute eigenvalue approximations of kernel-based matrices that are too large to allow for direct eigenvalue computation. See e.g. [11] and [25] for its applications in different settings. Originally introduced to the matrix computations context in [45], further improvements have been suggested in [10] and [9] and its usage for classification problems has been proposed in [5]. It is based on dividing the data points into a sample set X of L nodes and its complement Y. After permutation, the adjacency matrix $\mathbf{W}$ can be split into blocks

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_{XX} & \mathbf{W}_{XY} \\ \mathbf{W}_{XY}^T & \mathbf{W}_{YY} \end{bmatrix},$$

where the blocks $\mathbf{W}_{XX} \in \mathbb{R}^{L\times L}$ and $\mathbf{W}_{YY} \in \mathbb{R}^{(n-L)\times(n-L)}$ are the adjacency matrices of the canonical subgraphs with node sets X and Y, respectively, and the block $\mathbf{W}_{XY} \in \mathbb{R}^{L\times(n-L)}$ contains the similarities between all combinations of nodes from X and Y.

The basic idea of the Nyström method is to compute only $\mathbf{W}_{XX}$ and $\mathbf{W}_{XY}$ explicitly, but not the remaining block $\mathbf{W}_{YY}$. If $L \ll n$, the approach significantly decreases the required

number of data point comparisons. Assuming that $\mathbf{W}_{XX}$ is regular, the method approximates $\mathbf{W}$ by

$$\mathbf{W} \approx \mathbf{W}_{\mathbf{E}} = \begin{bmatrix} \mathbf{W}_{XX} \\ \mathbf{W}_{XY}^T \end{bmatrix} \mathbf{W}_{XX}^{-1} \begin{bmatrix} \mathbf{W}_{XX} & \mathbf{W}_{XY} \end{bmatrix} = \begin{bmatrix} \mathbf{W}_{XX} & \mathbf{W}_{XY} \\ \mathbf{W}_{XY}^T & \mathbf{W}_{XY}^T \mathbf{W}_{XX}^{-1} \mathbf{W}_{XY} \end{bmatrix}, \tag{5.1}$$

which constitutes a rank-L approximation due to the size and regularity of $\mathbf{W}_{XX}$. This formula is used once in approximating the degree matrix $\mathbf{D}$ by $\mathbf{D}_{\mathbf{E}} = \operatorname{diag}(\mathbf{W}_{\mathbf{E}}\mathbf{1})$ and once in approximating the eigenvalues of $\mathbf{A}$ via the rank-L eigenvalue decomposition

$$\mathbf{A}_{\mathbf{E}} := \mathbf{D}_{\mathbf{E}}^{-1/2} \mathbf{W}_{\mathbf{E}} \mathbf{D}_{\mathbf{E}}^{-1/2} = \mathbf{V}_L \mathbf{\Lambda}_L \mathbf{V}_L^*.$$

This can be computed without having to set up the full matrix, e.g. by the technique described in [10] made up mainly of two singular value decompositions of $(L \times L)$-sized matrices, which is technically only applicable if $\mathbf{W}$ is positive definite. Alternatively, we have achieved better results by computing the QR factorization $\hat{\mathbf{Q}}\hat{\mathbf{R}} := \mathbf{D}_{\mathbf{E}}^{-1/2}[\mathbf{W}_{XX}\ \mathbf{W}_{XY}]^T$ and the eigenvalue decomposition $\mathbf{U}_L \mathbf{\Lambda}_L \mathbf{U}_L^T := \hat{\mathbf{R}} \mathbf{W}_{XX}^{-1} \hat{\mathbf{R}}^T$, leading to the eigenvector matrix $\mathbf{V}_L = \hat{\mathbf{Q}}\mathbf{U}_L$. The arithmetic complexity of this algorithm can be easily confirmed to be $\mathcal{O}\big(n\,L^2\big)$.

The eigenvalue accuracy depends strongly on the quality of the approximation

$$\mathbf{W}_{YY} \approx \mathbf{W}_{XY}^T \mathbf{W}_{XX}^{-1} \mathbf{W}_{XY}.$$

Since the sample set X is a randomly chosen subset of the indices from $1, \ldots, n$, its size L is the decisive method parameter and its choice is a nontrivial task. On the one hand, L needs to be small for the method to be efficient. On the other hand, a too small choice of L may cause extreme errors, especially because the approximation error in $\mathbf{D}_{\mathbf{E}}$ propagates to the eigenvalue computation. In spite of the positivity of the diagonal of $\mathbf{D}$, negative entries in $\mathbf{D}_{\mathbf{E}}$ cannot be ruled out and are observed in practice. Hence imaginary entries may occur in $\mathbf{D}_{\mathbf{E}}^{-1/2}$ and thus $\mathbf{A}_{\mathbf{E}}$, making the results extremely unreliable. This behaviour follows the same structure as Lemma 3.1, however, we do not have a meaningful bound on $\|\mathbf{W}_{YY} - \mathbf{W}_{XY}^T \mathbf{W}_{XX}^{-1} \mathbf{W}_{XY}\|_\infty$ that would guarantee favorable error behaviour.

5.2. A NFFT-based accelerated Nyström-Gaussian method. Another important contribution of this paper is the development of an improved Nyström method, which utilizes the NFFT-based fast summation from Section 3. It is based on a slightly different algorithm that has been recently introduced as a Nyström method, cf. [24] and the references therein. Their basic idea is rewriting the traditional Nyström approximation as

$$\mathbf{A} \approx (\mathbf{AQ})(\mathbf{Q}^T\mathbf{AQ})^{-1}(\mathbf{AQ})^T$$

where $\mathbf{Q} \in \mathbb{R}^{n \times L}$ is a matrix with orthogonal columns. If $\mathbf{Q}$ holds the first L columns of a permutation matrix, one obtains the traditional Nyström method from Section 5.1. Inspired by similar randomized linear algebra algorithm such as randomized singular value decomposition, this choice of $\mathbf{Q}$ is replaced in [24] by $\mathbf{Q} = \operatorname{orth}(\mathbf{AG})$, where $\mathbf{G} \in \mathbb{R}^{n \times L}$ is a Gaussian matrix with normally distributed random entries and orth denotes column-wise orthonormalization. Unfortunately, this setup requires $2L$ matrix-vector products with the full matrix $\mathbf{A}$.

We now propose accelerating these matrix-vector products by computing $\mathbf{AQ}$ column-wise via the NFFT-based fast summation Algorithm 3.1 in order to avoid full matrix setup or slow direct matrix-vector products. In addition, we propose replacing the inverse $(\mathbf{Q}^T\mathbf{AQ})^{-1}$ by a low-rank approximation based only on the $M \in \mathbb{N}$ largest eigenvalues of $\mathbf{Q}^T\mathbf{AQ}$. This way, a rank-M approximation of $\mathbf{A}$ is produced, where M may be the actual number of required eigenvalues or larger. The resulting method "Nyström-Gaussian-NFFT" is presented in Algorithm 5.1. Its arithmetic complexity is $\mathcal{O}\big(n\,L^2\big)$. On the first glance, this arithmetic complexity seems to be identical to the one of the traditional Nyström method from Section 5.1. However, as we observe in the numerical tests in Section 6.1, we may choose the parameter L distinctly smaller for Algorithm 5.1, i.e., $L \sim k$, where k is the number of eigenvalues and eigenvectors. Then, the resulting arithmetic complexity is $\mathcal{O}\big(n\,k^2\big)$.

Algorithm 5.1 NFFT-based accelerated Nyström-Gaussian method ("Nyström-Gaussian-NFFT") for eigenvalue approximation $\mathbf{V}_k\mathbf{\Lambda}_k\mathbf{V}_k^* \approx \mathbf{A} := \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$.

Input:	σ, $\{\mathbf{v}_j\}_{j=1}^n$	scaling parameter and vertex set, $\mathbf{v}_j \in \mathbb{R}^d$, specifying $\mathbf{W}$,
	k	number of desired eigenvalues $\in \mathbb{N}$,
	L	number of random Gaussian columns $\geq M \geq k$,
	M	rank of inversion $\geq k$.

1. Setup the NFFT-based fast summation parameters for computing matrix-vector products with $\mathbf{W}$ using Algorithm 3.1, cf. Section 3.
2. Compute the degree matrix $\mathbf{D} = \mathrm{diag}(\mathbf{W1})$ using Algorithm 3.1.
3. Setup a random Gaussian matrix $\mathbf{G} \in \mathbb{R}^{n\times L}$, compute $\mathbf{Y} = \mathbf{AG}$ column-wise via Algorithm 3.1, and $\mathbf{Q} = \mathrm{orth}(\mathbf{Y}) \in \mathbb{R}^{n\times L}$ by QR-factorization.
4. Compute $\mathbf{B}_1 = \mathbf{AQ} \in \mathbb{R}^{n\times L}$ column-wise via Algorithm 3.1 and $\mathbf{B}_2 = \mathbf{Q}^T\mathbf{B}_1 \in \mathbb{R}^{L\times L}$.
5. Compute the diagonal matrix $\mathbf{\Sigma}_M$ of the M largest positive eigenvalues of $\mathbf{B_2}$ and the matrix $\mathbf{U}_M \in \mathbb{R}^{L\times M}$ holding the corresponding orthonormal eigenvectors as columns.
6. Compute the QR-factorization $\hat{\mathbf{Q}}\hat{\mathbf{R}} = \mathbf{B}_1\mathbf{U}_M$, $\hat{\mathbf{Q}} \in \mathbb{R}^{n\times M}$, $\hat{\mathbf{R}} \in \mathbb{R}^{M\times M}$.
7. Compute the eigenvalue decomposition $\hat{\mathbf{U}}_M\mathbf{\Lambda}_M\hat{\mathbf{U}}_M^T = \hat{\mathbf{R}}\mathbf{\Sigma}_M^{-1}\hat{\mathbf{R}}^T$ and set $\mathbf{V}_M = \hat{\mathbf{Q}}\hat{\mathbf{U}}_M$.
8. Put the k largest eigenvalues from $\mathbf{\Lambda}_M$ into the diagonal matrix $\mathbf{\Lambda}_k$ and corresponding eigenvectors from $\mathbf{V}_M$ into $\mathbf{V}_k$.

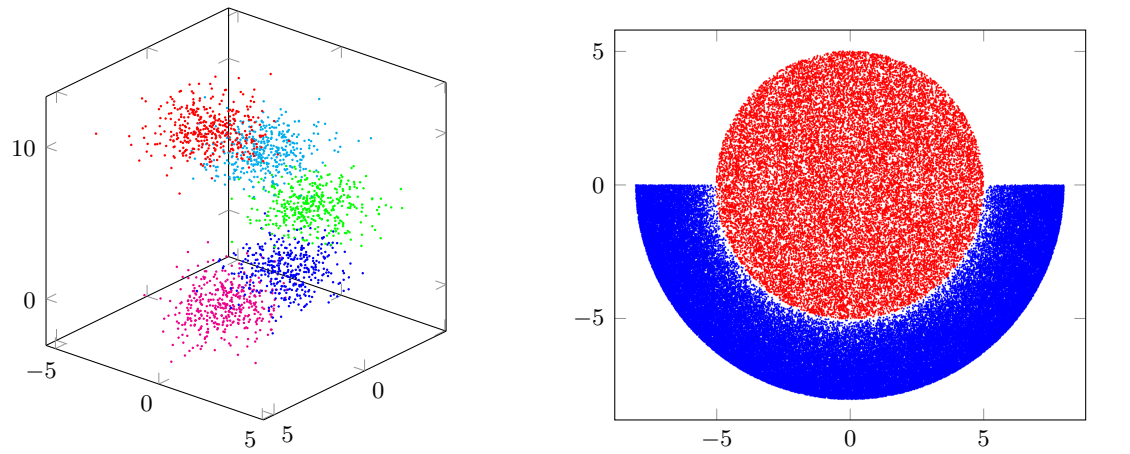
Output:	$\mathbf{\Lambda}_k \in \mathbb{R}^{k\times k}$	diagonal matrix of approximated largest eigenvalues of $\mathbf{A}$,
	$\mathbf{V}_k \in \mathbb{R}^{n\times k}$	corresponding orthonormal eigenvector matrix.
Complexity:	$\mathcal{O}\big(nL^2\big)$	

6. Numerical results.

All our experiments are performed using MATLAB implementations based on the NFFT3 library and MATLAB's `eigs` function. A short example code can be found on the homepage of the authors.[2]

[2] https://www.tu-chemnitz.de/mathematik/wire/people/files_alfke/NFFT-Lanczos-Example-v1.tar.gz

6.1. Accuracy and runtime of eigenvalue computations. We use the function `generateSpiralDataWithLabels.m`[3] to generate varying sets of three-dimensional data. The data points are in the form of a spiral and we can specify the number of classes as well as the number of points per class. We generate data sets with 5 classes and equal numbers of points per class, which have a total number of data points $n \in \{2\,000, 5\,000, 10\,000, 20\,000, 50\,000, 100\,000\}$. For the generation, we use the default parameters $h = 10$ and $r = 2$ in `generateSpiralDataWithLabels.m`. For each n, we generate 5 random spiral data sets. In Figure 2a, we visualize an example data set with $n = 2\,000$ total points. For the adjacency matrix $\mathbf{W}$, we set the scaling parameter $\sigma = 3.5$. Using the NFFT-based Lanczos method from Section 4, we compute the 10 largest eigenvalues and the corresponding eigenvectors of the matrix $\mathbf{A} := \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ for each data set. We consider three different parameter setups for the NFFT in Algorithm 3.1, achieving different accuracies. We set the bandwidth $N = 16$ and the window cut-off parameter $m = 2$ in setup #1, $N = 32$ and $m = 4$ in setup #2, as well as $N = 64$ and $m = 7$ in setup #3. For all three setups, we use $\varepsilon_{\mathrm{B}} = 0$. For comparison, we also apply the Nyström method from Section 5.1, where we perform 10 repetitions for each data set, since the method uses random sub-sampling in order to obtain a rank-L approximation of the adjacency matrix $\mathbf{W}$. We consider two different Nyström setups with rank $L \in \{n/10, n/4\}$. Moreover, we use the hybrid Nyström-Gaussian-NFFT method from Algorithm 5.1 in Section 5.2 with $L \in \{20, 50\}$ Gaussian columns, parameter $M = 10$ as well as fast summation parameters corresponding to setup #2, where we perform 10 repetitions for each data set. Additionally, we compute the eigenvalues and eigenvectors by a direct method, which applies the Lanczos method using full matrix-vector products with the adjacency matrix $\mathbf{W}$. For the Nyström method from Section 5.1 and the direct computation method, we only run tests for a total number of data points $n \in \{2\,000, 5\,000, 10\,000, 20\,000\}$ due to long runtimes.

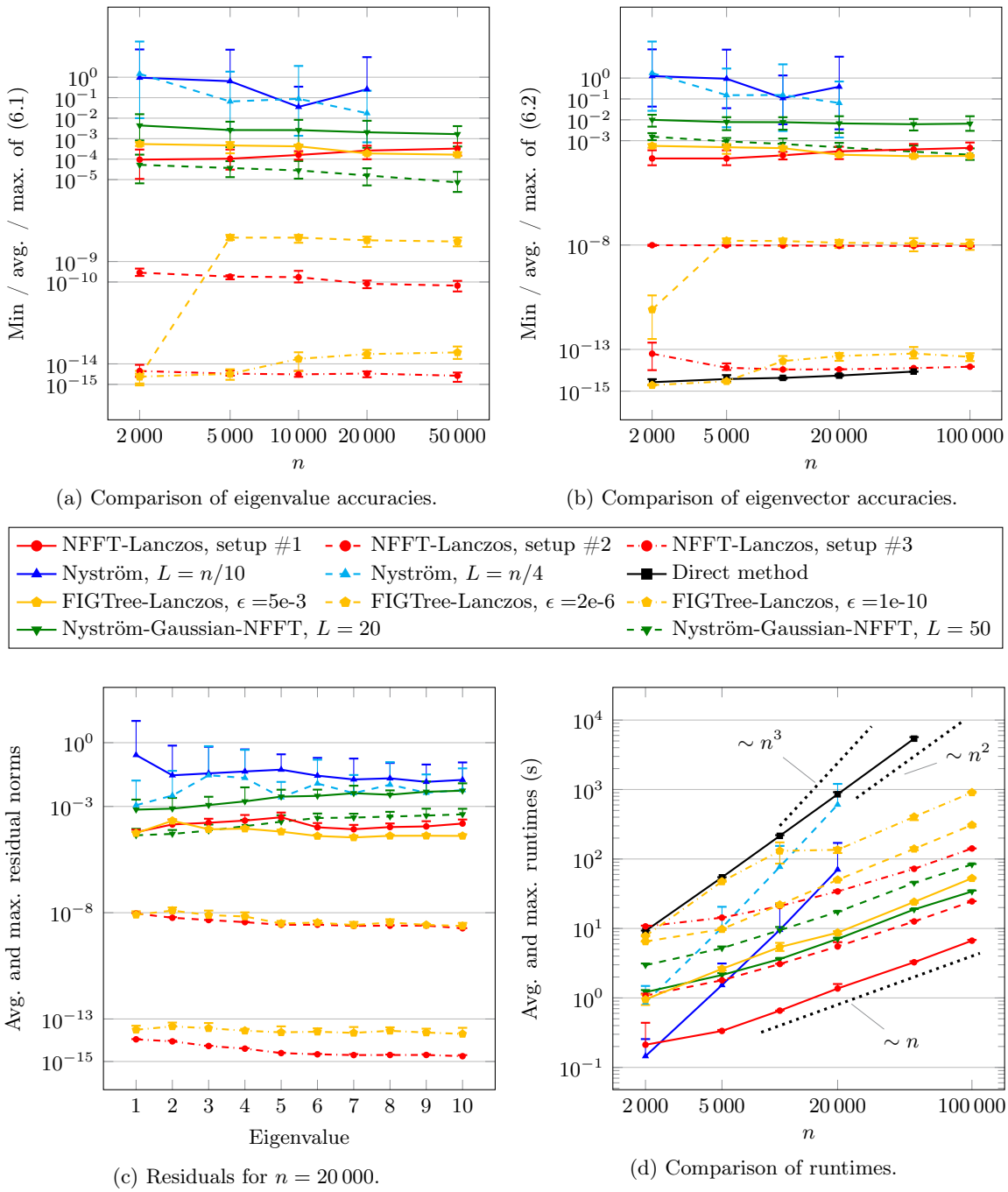


(a) Spiral example with $n = 2\,000$ points.

(b) Crescent-fullmoon example with $n = 100\,000$ points.

Figure 2. *Illustration of spiral and crescent-fullmoon data sets.*

[3] https://sites.google.com/site/kittipat/matlabtechniques



(a) Comparison of eigenvalue accuracies.

(b) Comparison of eigenvector accuracies.

(c) Residuals for $n = 20\,000$.

(d) Comparison of runtimes.

Figure 3. *Comparison of accuracies and runtimes for spiral data sets.*

In Figure 3, we visualize the results of the test runs. We show the minimum, average and maximum of the maximum eigenvalue errors in Figure 3a. For this, we first determine the

maximum eigenvalue errors

$$\max_{j=1,\dots,10} \left| \lambda_j - \lambda_j^{(\text{direct})} \right| \tag{6.1}$$

for each test run, where λ_j denotes the j-th eigenvalue computed by the method under consideration and $\lambda_j^{(\text{direct})}$ the one computed by a direct method using full matrix-vector products with the matrix $\mathbf{A}$. Then, for fixed total number of data points n and fixed parameter setup, we compute the minimum, average and maximum of (6.1), where the minimum, average and maximum are computed using 5 instances of (6.1) for the NFFT-based Lanczos method and $5 \cdot 10$ instances of (6.1) for the Nyström-based methods. We observe that the averages of the maximum eigenvalue errors (6.1) are above 10^{-2} for the two considered parameter choices of the Nyström method from Section 5.1, even when the rank L is chosen as a quarter of the matrix size n. Moreover, the minima and maxima of (6.1) differ distinctly from the averages. In particular, the accuracies may vary strongly across different Nyström runs on an identical data set. For the NFFT-based Lanczos method, each minimum, average and maximum of the maximum eigenvalue errors (6.1) only differs slightly from one another. The maximum eigenvalue errors (6.1) are around 10^{-4} to 10^{-3} for parameter setup #1, around 10^{-10} to 10^{-9} for setup #2, and below 10^{-14} for setup #3. For the hybrid Nyström-Gaussian-NFFT method, which internally uses $2L$ many NFFT-based fast summations with parameter setup #2, the maximum eigenvalue errors (6.1) are around 10^{-3} to 10^{-2} for parameter $L = 20$ and around 10^{-5} to 10^{-4} for $L = 50$. This means that the observed maximum eigenvalue errors (6.1) are distinctly smaller compared to the ones of the traditional Nyström method, and the errors for parameter $L = 50$ are slightly smaller than the ones of the NFFT-based Lanczos method with parameter setup #1.

In Figure 3b, we depict the minimum, average and maximum of the maximum residual norms (6.2) for each total number of data points n. We compute these numbers by first determining the maximum residual norms

$$\max_{j=1,\dots,10} \|\mathbf{A}\mathbf{v}_j - \lambda_j \mathbf{v}_j\|_2 \tag{6.2}$$

for each test run, where λ_j denotes the j-th eigenvalue of $\mathbf{A}$ and $\mathbf{v}_j$ the corresponding eigenvector. Then, for fixed n and fixed parameter setup, we compute the minimum, average and maximum of (6.2). We observe that the averages of the maximum residual norms (6.2) are above 10^{-1} for the considered parameter choices of the Nyström method, even when the rank L is chosen as a quarter of the matrix size n. Moreover, the minima and maxima of the maximum residual norms (6.2) differ distinctly from the averages. Especially, the accuracies may vary strongly across different Nyström runs on an identical data set. For the NFFT-based Lanczos method, each minimum, average and maximum of (6.2) only differs slightly from one another. The maximum residual norms (6.2) are around 10^{-4} to 10^{-3} for parameter setup #1, around 10^{-8} for setup #2, and around 10^{-15} to 10^{-13} for setup #3. For the hybrid Nyström-Gaussian-NFFT method, maximum residual norms (6.2) are around 10^{-2} for parameter $L = 20$ and around 10^{-4} to 10^{-3} for $L = 50$. In the latter case, the errors are slightly larger than the ones of the NFFT-based Lanczos method with parameter setup #1 for $n \in \{2\,000, 5\,000, 10\,000, 20\,000\}$ data points and slightly smaller for $n \in \{50\,000, 100\,000\}$.

Additionally, in Figure 3c, we investigate the average and maximum of the maximum residual norms (6.2) for each fixed eigenvalue λ_j for $n = 20\,000$ data points. For Nyström $L = n/10$, we observe that the residual norms belonging to the first eigenvalue are distinctly larger than for the remaining eigenvalues. In general, the observed maximal residual norms (6.2) vary similarly for each eigenvalue. For the NFFT-based Lanczos method with parameter setup #2 and #3, the maximum residual norms (6.2) of the tail eigenvalues are slightly smaller than of the leading eigenvalues, which is not the case for the parameter setup #1 as well as for the results of the hybrid Nyström-Gaussian-NFFT method.

In Figure 3d, we show the average and maximum runtimes of the different methods and parameter choices in dependence of the total number of data points n. The runtimes were determined on a computer with Intel Core i7 CPU 970 (3.20 GHz) using one thread. We remark that the NFFT supports OpenMP, cf. [43], but we restricted all time measurements to 1 thread for better comparison. We observe that the runtimes of the traditional Nyström method grow approximately like $\sim n^3$, and the runtimes of the direct computation method for the eigenvalues grow approximately like $\sim n^2$. Moreover, the slopes of the runtime graphs of the NFFT-based Lanczos method are distinctly smaller and the runtimes grow approximately like $\sim n$. Depending on the parameter choices, the NFFT-based Lanczos method is faster than the Nyström method once the total number of data points n is above $2\,000$ – $10\,000$. The hybrid Nyström-Gaussian-NFFT method with parameter $L = 20$ is slightly slower than the NFFT-based Lanczos method with setup #2. For the parameter $L = 50$ the method is slower by a factor of approximately 2.5. In both cases, the runtimes grow approximately like $\sim n$. The runtimes of the direct method were the highest ones in most cases. For the tests, we precomputed the diagonal entries of the matrix $\mathbf{D}^{-1/2}$ but we computed the entries of the weight matrix $\mathbf{W}$ again for each matrix-vector multiplication with the matrix $\mathbf{A}$. Alternatively, one could store the whole matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ for small problem sizes n and this would have reduced the runtimes of the direct method to $1/20$. However, then we would have to store at least $n(n-1)/2$ values, which would already require about 10 GB RAM for $n = 50\,000$ and double precision.

For comparison, we also applied the FIGTree method from [26] to our testcases, and we denote the obtained results by "FIGTree-Lanczos" in Figure 3. The FIGTree accuracy parameter ϵ was chosen $\in \{5{\cdot}10^{-3}, 2{\cdot}10^{-6}, 10^{-10}\}$ such that the resulting residual norms (6.2) in Figure 3b approximately match those of the NFFT-based Lanczos method for setup #1,#2,#3. We observe that the obtained eigenvalue accuracies in Figure 3a are similar for $\epsilon = 5\cdot 10^{-3}$ and 10^{-10} to the ones of the NFFT-based Lanczos method for setup #1 and #3, respectively. For $n \geq 5\,000$ data points and FIGTree accuracy parameter $\epsilon = 2\cdot 10^{-6}$, we observe for our testcase that the obtained eigenvalue accuracies are lower by about two order of magnitudes compared to the NFFT-based Lanczos method with setup #2. When looking at the obtained runtimes, we observe that "FIGTree-Lanczos" requires approximately 4 times to 7 times the runtime of the corresponding NFFT-based Lanczos method with comparable eigenvector accuracy in most cases.

6.2. Applications. In the following, we will showcase the effect of the improved accuracy on popular data science methods that utilize the graph Laplacian matrix. We will compare how the methods perform if the eigenvectors are computed with the NFFT-based Lanczos

method or the traditional Nyström extension.

6.2.1. Spectral clustering. Spectral clustering is an increasingly popular technique [44] and we briefly illustrate the method proposed in [28]. The basis of their algorithm is a truncated eigenapproximation $\mathbf{V}_k\mathbf{D}_k\mathbf{V}_k^T$ with $\mathbf{V}_k \in \mathbb{R}^{n\times k}$, which is an approximation based on the smallest eigenvalues and eigenvectors of the graph Laplacian. Now the rows of $\mathbf{V}_k$ are normalized to obtain a matrix $\mathbf{Y}_k$. The normalized rows are then divided into a fixed number of disjoint clusters by a standard k-means algorithm.

Here, we apply spectral clustering to an image segmentation problem. The original image of size 533×800 is depicted in Figure 5a. We construct a graph Laplacian where each pixel corresponds to a node in the graph and the distance measure is the distance between the values in all three color channels, such that each vertex $\mathbf{v}_j \in \{0, 1, \ldots, 255\}^3$. Correspondingly, the graph Laplacian would be a dense matrix of size $426\,400 \times 426\,400$. We set the scaling parameter $\sigma = 90$. Figure 4 shows the first ten eigenvalues of the matrix $\mathbf{A}$.

For obtaining reference results, we use the Matlab function `eigs` on the full matrix $\mathbf{A}$ computing 4 eigenvectors and this required more than 31 hours using up to 32 threads on a computer with Intel Xeon E7-4880 CPUs (2.50 GHz), using more than 500 CPU hours in total. Next, we applied the NFFT-based Lanczos method from Section 4 with parameters $N = 16$, $m = 2$, $p = 2$ and $\varepsilon_{\mathrm{B}} = 1/8$ for the eigenvector computations. We show the results in Figure 5b and 5c for $k = 2$ and $k = 4$ classes, respectively. The segmented images look satisfactory. The main features of the image are preserved and large areas of similar color are correctly assigned to the same cluster, while there are only small "noisy" areas. Compared to the segmented image from the direct computations, we have approximately 0.1 % differences (467 out of 426,400) in the class assigments in the case of $k = 4$ classes. For the runtimes, we measure approximately 25 seconds for the NFFT-based Lanczos method and 18 seconds for the k-means algorithm on a computer with Intel Core i7 CPU 970 (3.20 GHz) using one thread.

Additionally, we ran the Nyström method 100 times with parameter $L = 250$. Here the runtimes were approximately 60 seconds on average without the runtime for the clustering. We applied the k-means algorithm for $k = 4$ classes, which required approximately 22 seconds on average. We observed that in 79 of the 100 test runs of Nyström followed by k-means, the images appear to be very close to the ones obtained when applying `eigs` on the full matrix $\mathbf{A}$, i.e., the differences are less than 2 %. In Figure 5d, we visualize the results of a corresponding test run. However, in 13 of the 100 test runs, the Nyström method returned eigenvectors which caused segmentation differences of more than 20 % with such "noisy" images that we consider these as "failed" runs. See Figure 5e for one example with approximately 25 % differences. The differences between Figure 5c and 5e are shown as a black and white picture in Figure 5f.

Moreover, we tested increasing the parameter L to 500. Then, the run times increased to approximately 152 seconds on average. When applying the k-means algorithm to the obtained eigenvectors, the results improved. The differences compared to the reference image segmentation are less than 2 % in 85 of the 100 test runs and larger than 20 % in 9 test runs.

6.2.2. Semi-supervised learning by a phase field method. We here want to state an exemplary method that relies heavily on a number of eigenvectors of the graph Laplacian. It was proposed by Bertozzi and Flenner [5] and corresponds to a semi-supervised learning (SSL)

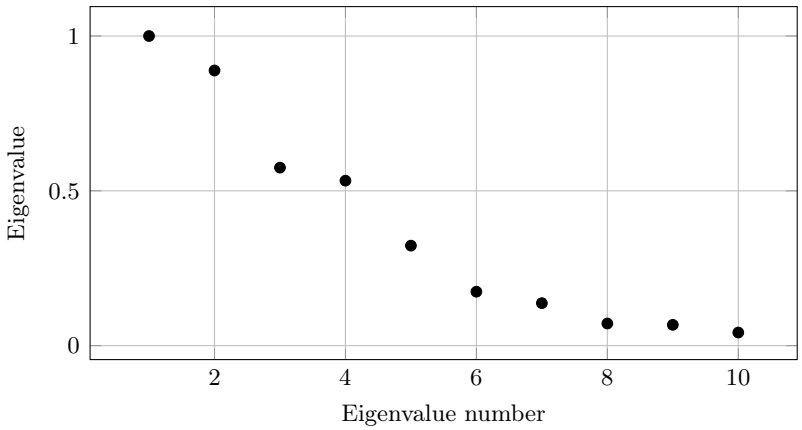


Figure 4. *First ten eigenvalues of* $\mathbf{A}$ *using Gaussian weights and scaling parameter* $\sigma = 90$ *for Figure 5a.*

problem. Suppose we have a graph-based dataset as before where each vertex is assigned to one of C classes. A training set of s random sample vertices from each class is set up. For the case of $C = 2$ classes, a training vector $\mathbf{f} \in \mathbb{R}^n$ is set up with entries -1 for training nodes from one class, 1 for training nodes from the other class, and 0 for nodes that do not belong to the training data. The task of SSL is to use $\mathbf{f}$ to find a classification vector $\mathbf{u} \in \mathbb{R}^n$. The sign of its entries is then used to predict each node's assigned class.

One successful approach computes $\mathbf{u}$ as the end point of the trajectory described by the Allen–Cahn equation

$$\mathbf{u} : [0, \infty) \to \mathbb{R}^n, \qquad \mathbf{u}_t = -\varepsilon \mathbf{L}_s \mathbf{u} - \frac{1}{\varepsilon}\psi'(\mathbf{u}) + \mathbf{\Omega}(\mathbf{f} - \mathbf{u}), \qquad \mathbf{u}(0) = \mathbf{f}$$

(see [42, 23] for details). Here $\psi(u) = (u^2 - 1)^2$ is the double-well potential, which we understand to be applied component-wise, and $\mathbf{\Omega}$ denotes a diagonal matrix with entries $\Omega_{ii} = \omega_0 > 0$ if vertex i belongs to the training data and $\Omega_{ii} = 0$ otherwise. To discretize this ODE we will not introduce an index for the temporal discretization but rather assume that all values $\mathbf{u}$ are evaluated at the new time-point whereas $\bar{\mathbf{u}}$ indicates the previous time-point. We then obtain

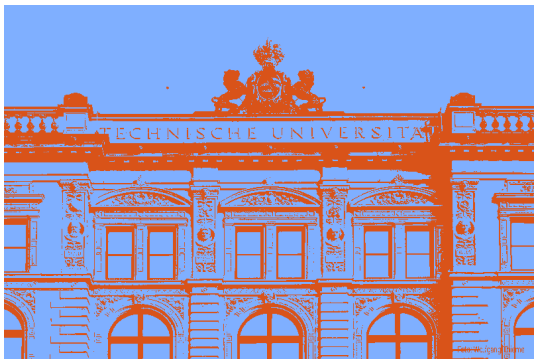
$$\frac{\mathbf{u} - \bar{\mathbf{u}}}{\tau} + \varepsilon \mathbf{L}_s \mathbf{u} + c\mathbf{u} = -\frac{1}{\varepsilon}\psi'(\bar{\mathbf{u}}) + c\bar{\mathbf{u}} + \mathbf{\Omega}(\mathbf{f} - \bar{\mathbf{u}}),$$

where $\mathbf{u}$ is a vector defined on the graph on which we base the final classification decision. Here, $c > 0$ is a positive parameter for the convexity splitting technique [5]. For a more detailed discussion of how to set these parameters we refer to [5, 7]. We now use the k computed eigenvalues and eigenvectors $(\lambda_j, \mathbf{v}_j)$ of $\mathbf{L}_s$ such that we can write $\mathbf{u} = \sum_{j=1}^{k} u_j \mathbf{v}_j$ and from this we get

$$\frac{u_j - \bar{u}_j}{\tau} + \varepsilon \lambda_j u_j + c u_j = -\frac{1}{\varepsilon}\mathbf{v}_j^T \psi'(\bar{\mathbf{u}}) + c\bar{u}_j + \mathbf{v}_j^T \mathbf{\Omega}(\mathbf{f} - \bar{\mathbf{u}}).$$

This equation can be solved to obtain the new coefficients u_j from the old coefficients $\bar{u}_j$. After a sufficient number of time steps, $\mathbf{u}$ will converge against a stable solution.

(a) Original image[4]

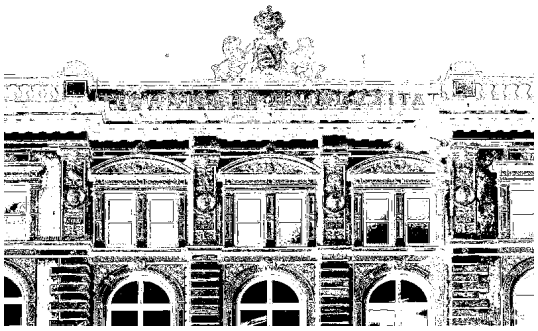


(b) $k = 2$ classes, NFFT-Lanczos

(c) $k = 4$ classes, NFFT-Lanczos

(d) $k = 4$ classes, Nyström

(e) $k = 4$ classes, Nyström ("failed" run)



(f) differences between (c) and (e)

Figure 5. *Results of image segmentation* ($533 \times 800 = 426\,400$ *pixels) via spectral clustering and k-means using the NFFT-based Lanczos method from Section 4 and the Nyström method from Section 5.1. "Failed run" in Subfigure (e) means segmentation differences of more than 20 % compared to the results obtained when applying* `eigs` *on the full matrix* $\mathbf{A}$.

We apply this method to the same spiral data set as seen in Section 6.1, again with $\sigma = 3.5$ but this time only with $n = 100\,000$. The data points have been generated by a multivariate

[4]Image source: TU Chemnitz/Wolfgang Thieme

normal distribution around five center points, and the true label of each vertex has been set to the center point that is closest to it. We computed the eigenvectors to the $k = 5$ smallest eigenvalues of the Laplacian; once by the NFFT-based Lanczos method with $n = 32$, $m = 4$, and $\varepsilon_B = 0$, and once with the traditional Nyström method with $L = 1\,000$ where only 5 columns of $\mathbf{V}_L$ are used. We then applied the described method with $\tau = 0.1$, $\varepsilon = 10$, $\omega_0 = 10\,000$, and $c = \frac{2}{\varepsilon} + \omega_0$. The iteration terminated if the squared relative change in $\mathbf{u}$ was less than 1e-10. We repeat this process for 50 instances of the spiral dataset and sample sizes $s \in \{1, 2, 3, 4, 5, 7, 10\}$.

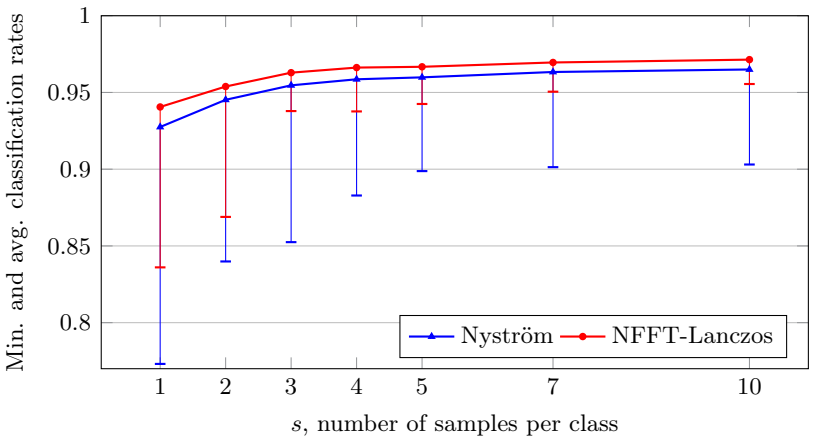


Figure 6. *Comparison of average classification rates with the phase field method on relabeled spiral data sets.*

Figure 6 depicts the average accuracy results. We conclude that in this example, the increased eigenvector quality achieved by the NFFT-based method yields an average accuracy boost of approximately 0.5 to 1.5 percentage points, as well as the worst result being significantly less bad. On a computer with Intel Core i7 CPU 4770 (3.40 GHz), the runtimes were approximately 8 seconds for the NFFT-based Lanczos method, 27 seconds for the Nyström method, and less than a second for the solution of the Allen–Cahn equation, which almost always converged after only three time steps.

6.2.3. Semi-supervised learning by a kernel method. In addition to the phase field method, we employ a second semi-supervised learning technique used in [48, 14] for SSL problems with only two classes. Based on a training vector $\mathbf{f}$ holding 1, -1, or 0 just as in the previous section, a similar $\mathbf{u}$ is obtained by minimizing the function

$$\arg\min_{\mathbf{u}\in\mathbb{R}^n} \frac{1}{2}\|\mathbf{u}-\mathbf{f}\|_2^2 + \frac{\beta}{2}\mathbf{u}^T\mathbf{L}_s\mathbf{u}, \tag{6.3}$$

where β can be understood as a regularization parameter. For the solution of this minimization problem, we only have to solve the equation

$$(\mathbf{I} + \beta\mathbf{L}_s)\,\mathbf{u} = \mathbf{f}, \tag{6.4}$$

where $\mathbf{I}$ is the identity matrix. Similar systems arise naturally in scattered data interpolation [17]. We run numerical tests using the `crescentfullmoon.m`[5] data set with $n = 100\,000$ data points and parameters `r1`=5, `r2`=5, `r3`=8. As illustrated in Figure 2b, the set is divided into two classes of points in the full moon and the crescent, distributed in a 1-to-3 ratio. We generate 5 random instances of the data set, and for each instance we run 10 repetitions with randomly chosen training data, where we consider $s \in \{1, 2, 5, 10, 25\}$ known samples per class. For the adjacency matrix $\mathbf{W}$, we set the scaling parameter $\sigma = 0.1$. The tests are run with regularization parameter $\beta \in \{10^3, 3 \cdot 10^3, 10^4, 3 \cdot 10^4, 10^5\}$. We solve each system (6.4) using the CG algorithm with tolerance parameter 10^{-4} and a maximum number of 1 000 iterations. For the fast matrix-vector multiplications with the matrix $\mathbf{L}_s$, we use the NFFT-based fast summation in Algorithm 3.1 with parameters $N = 512$, $m = 3$, $\varepsilon_{\mathrm{B}} = 0$.

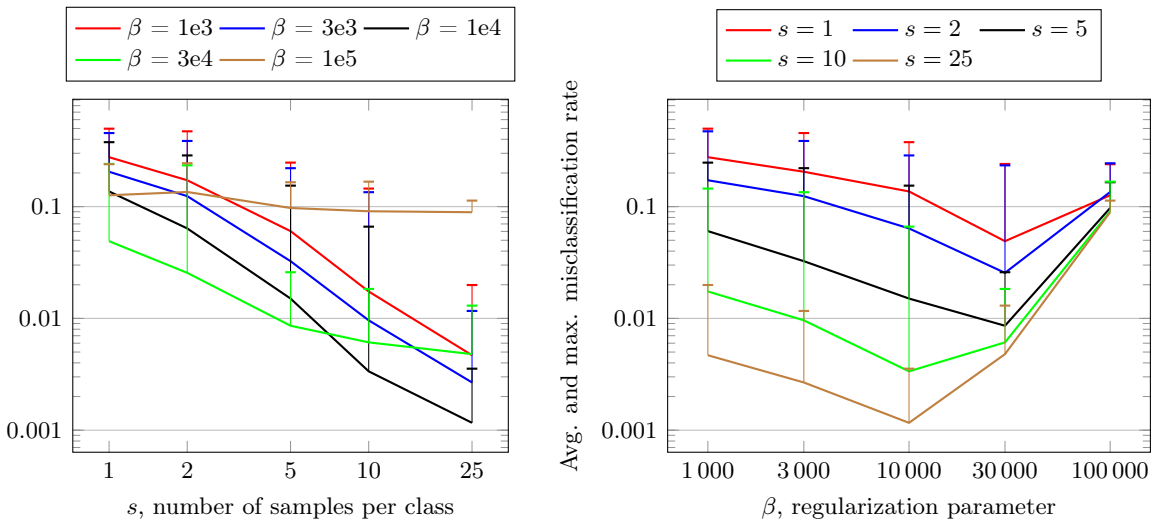


Figure 7. *Misclassification rate solving (6.4) using the CG algorithm and Algorithm 3.1 for the `crescentfullmoon.m` data set with $n = 100\,000$ data points.*

In Figure 7, we visualize the average and maximum misclassification rate of the $5 \cdot 10$ test runs for each fixed s and β. In the left plot, we show the misclassification rate in dependence of the number of samples s per class for the different regularization parameters β. We observe in general that the misclassification rates decrease for increasing s. The lowest rate is achieved for $s = 25$ samples per class and $\beta = 10^4$, where the average and maximum misclassification rate are 0.0012 and 0.0036, respectively. In the right plot, we depict the misclassification rate in dependence of the regularization parameter β for fixed number of samples s per class. For $s \in \{1, 2, 5\}$, the average misclassification rates decline for increasing β until $\beta = 3 \cdot 10^4$ and grow again for $\beta = 10^5$. For $s \in \{10, 25\}$, the average misclassification rates decline for increasing β until $\beta = 10^4$ and grow again afterwards. We remark that in all test runs, the maximum number of CG iterations was 536 and the maximum runtime for solving (6.4) was approximately 151 seconds on a computer with Intel Core i7 CPU 970 (3.20 GHz) using one

[5] https://www.mathworks.com/matlabcentral/fileexchange/41459-6-functions-for-generating-artificial-datasets

thread.

Additionally, we used the NFFT-based Lanczos method from Section 4 in order to approximate the matrix $\mathbf{A} := \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ by a truncated eigenapproximation $\mathbf{V}_k\mathbf{D}_k\mathbf{V}_k^T$ with $\mathbf{V}_k \in \mathbb{R}^{n,k}$ and this allows for computing the matrix-vector products in (6.4) in a fast way for fixed small k. Using $k = 10$ eigenvalues and eigenvectors, we achieve similar results as those shown in Figure 7. The computation of the eigenapproximation required up to 6 minutes on a computer with Intel Core i7 CPU 970 (3.20 GHz) using one thread. The maximum runtime for solving (6.4) was approximately 0.15 seconds.

Alternatively, we applied the Nyström method from Section 5.1 with parameter $L = 5\,000$ to obtain a truncated eigenapproximation, where the corresponding computation required more than 3 hours for each eigenapproximation. However, the eigenvalues were not computed correctly in our tests. This was due to the matrix block $\mathbf{W}_{XX}$ in Equation (5.1) being ill-conditioned. Consequently the CG method aborted in the first iteration and the output could not be used for classification.

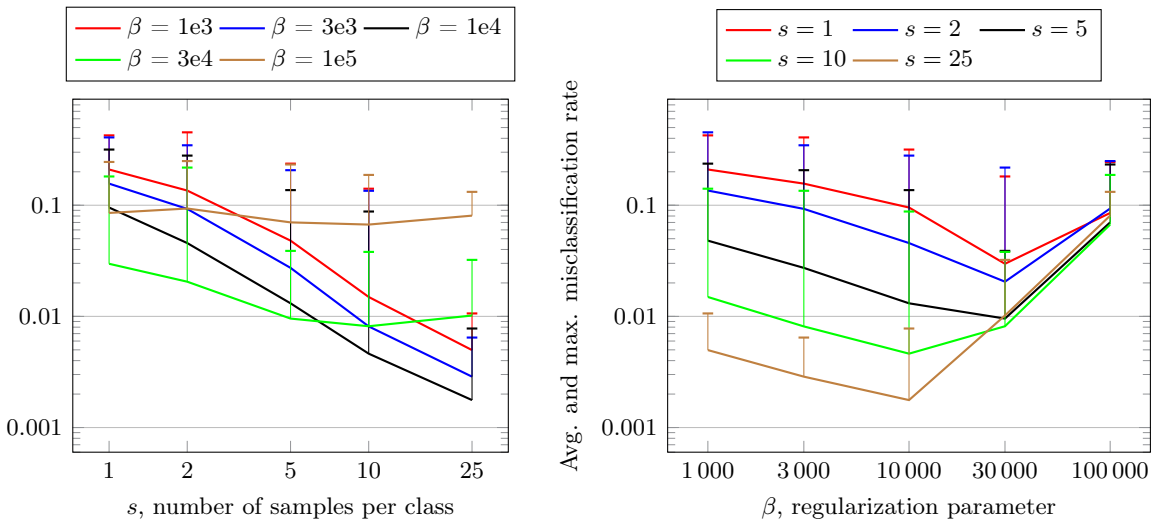


Figure 8. *Misclassification rate solving (6.4) using the CG algorithm and Algorithm 3.1 for the* `crescentfullmoon.m` *data set with $n = 100\,000$ data points and Laplacian RBF kernel (6.5).*

In order to illustrate the flexibility of the NFFT-based fast summation, we also apply Algorithm 3.1 to a non-Gaussian weight function w in (2.2). Here, we consider the "Laplacian RBF kernel" $K(\mathbf{y}) := \exp(-\|\mathbf{y}\|/\sigma)$, such that the weight matrix is constructed as

$$W_{ji} = w(v_j, v_i) = \begin{cases} \exp(-\|\mathbf{v}_j - \mathbf{v}_i\|/\sigma) & \text{if } j \neq i, \\ 0 & \text{otherwise.} \end{cases} \tag{6.5}$$

In our numerical tests, we set the shape parameter $\sigma = 0.05$ and we visualize the test results in Figure 8. We observe that the obtained misclassification rates are similar to the ones in Figure 7, where the Gaussian kernel was used. For some parameter settings, the misclassification rates are slightly better, for other ones slightly worse.

6.3. Kernel ridge regression. In this section we show that our approach can be applied to the problem of kernel ridge regression, which has a similar flavour to the problem from the previous section. We here illustrate that our method is very flexible since other than just Gaussian kernels can be used for the fast evaluation of matrix-vector products. The starting point is a simple linear regression problem via the minimization of

$$\arg\min_{\mathbf{u}\in\mathbb{R}^d} \frac{1}{2}\left\|\mathbf{f}-\mathbf{X}\mathbf{u}\right\|_2^2+\frac{\beta}{2}\left\|\mathbf{u}\right\|_2^2, \tag{6.6}$$

where $\mathbf{X}\in\mathbb{R}^{n\times d}$ is a design matrix holding training feature vectors $\mathbf{x}_j\in\mathbb{R}^d$ in its rows, i.e. $\mathbf{X}^T=[\mathbf{x}_1,\ldots,\mathbf{x}_n]$, and $\mathbf{f}\in\mathbb{R}^n$ is a given response vector. The solution $\mathbf{u}$ to this problem can then be used in a linear model to predict a response for any new point $\mathbf{x}\in\mathbb{R}^d$ as $F(\mathbf{x})=\mathbf{u}^T\mathbf{x}$.

The well-known solution formula can be rearranged using the Sherman–Morrison–Woodbury formula to obtain

$$\begin{aligned}
\mathbf{u} &= \left(\mathbf{X}^T\mathbf{X}+\beta\mathbf{I}_d\right)^{-1}\mathbf{X}^T\mathbf{f}\\
&= \left(\beta^{-1}\mathbf{I}_d-\beta^{-2}\mathbf{X}^T\left(\mathbf{I}_n+\beta^{-1}\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}\right)\mathbf{X}^T\mathbf{f}\\
&= \mathbf{X}^T\left(\beta^{-1}\mathbf{I}_n-\beta^{-1}\left(\beta\mathbf{I}_n+\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}\mathbf{X}^T\right)\mathbf{f}\\
&= \mathbf{X}^T\left(\beta\mathbf{I}_n+\mathbf{X}\mathbf{X}^T\right)^{-1}\left(\beta^{-1}\left(\beta\mathbf{I}_n+\mathbf{X}\mathbf{X}^T\right)-\beta^{-1}\mathbf{X}\mathbf{X}^T\right)\mathbf{f}\\
&= \mathbf{X}^T\left(\mathbf{X}\mathbf{X}^T+\beta\mathbf{I}_n\right)^{-1}\mathbf{f}.
\end{aligned}$$

Using this formula, we can introduce the dual variable $\boldsymbol{\alpha}=\left(\mathbf{X}\mathbf{X}^T+\beta\mathbf{I}\right)^{-1}\mathbf{f}$ and rewrite the predicted response of a new point $\mathbf{x}$ as

$$F(\mathbf{x})=\mathbf{u}^T\mathbf{x}=\left(\mathbf{X}^T\boldsymbol{\alpha}\right)^T\mathbf{x}=\sum_{i=1}^n\boldsymbol{\alpha}_i\mathbf{x}_i^T\mathbf{x}.$$

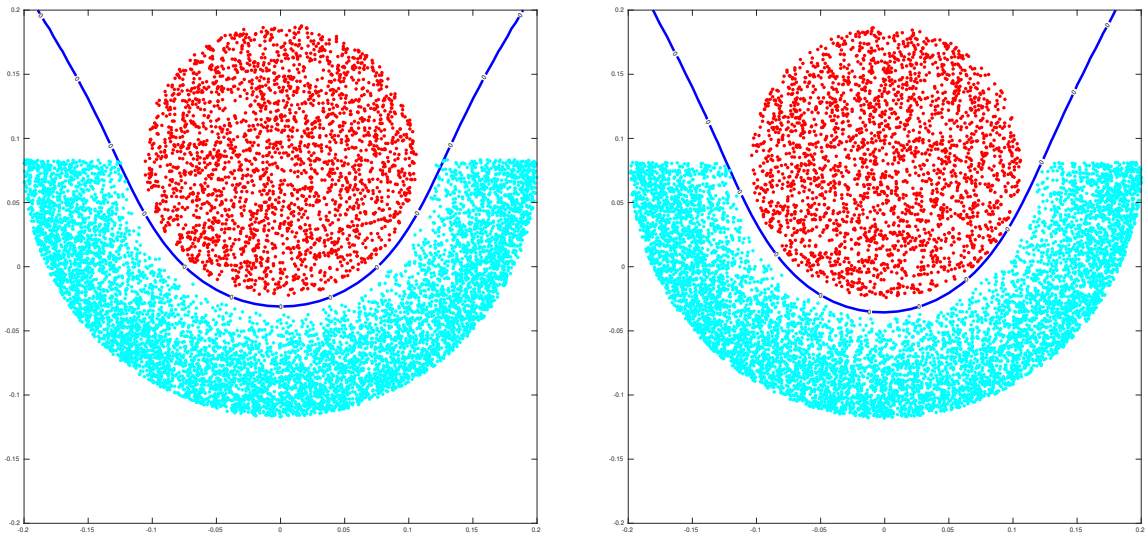
An idea for increasing the flexibility of this method is replacing expressions $\mathbf{x}_i^T\mathbf{x}_j$ with $K(\mathbf{x}_i,\mathbf{x}_j)$ where $K:\mathbb{R}^d\times\mathbb{R}^d\to\mathbb{R}$ is an arbitrary kernel function [33]. This leads to replacing $\mathbf{X}\mathbf{X}^T$ with the Gram matrix $\mathbf{K}$ with entries

$$\mathbf{K}_{ij}=K(\mathbf{x}_i,\mathbf{x}_j)\quad\forall\, i,j=1,\ldots,n.$$

Consequently, the dual variable becomes $\boldsymbol{\alpha}=\left(\mathbf{K}+\beta\mathbf{I}_n\right)^{-1}\mathbf{f}$ and we obtain the kernel-based prediction function

$$F(\mathbf{x})=\sum_{i=1}^n\boldsymbol{\alpha}_iK(\mathbf{x}_i,\mathbf{x}).$$

For more details we refer to [33]. It is easily seen that the main effort of this algorithm goes into the computation of the coefficient vector $\boldsymbol{\alpha}=\left(\mathbf{K}+\beta\mathbf{I}_n\right)^{-1}\mathbf{f}$. Note that this is were we again use the NFFT-based matrix vector products in combination with the preconditioned CG method as the matrix $\mathbf{K}+\beta\mathbf{I}_n$ is positive definite and amenable to being treated using the NFFT for a variety of different kernel functions. In Figure 9 we illustrate the results when kernel ridge regression is used with two different kernels, namely the Gaussian and the inverse multiquadric kernel.



(a) Inverse multiquadric (b) Gaussian

Figure 9. *Results of kernel ridge regression applied using an inverse multiquadric kernel (left) and a Gaussian kernel (right). The blue line indicates the decision boundary for the classification of new points.*

7. Conclusion. In this work, we have successfully applied the computational power of NFFT-based fast summation to core tools of data science. This was possible due to the nature of the fully connected graph Laplacian and the fact that many algorithms – most notably the Lanczos method for eigenvalue computation – only require matrix-vector products with the Laplacian matrix. By using Fourier coefficients to approximate the Gaussian kernel, we use Algorithm 3.1 to compute strong approximations of the matrix-vector product in $\mathcal{O}(n)$ complexity without storing or setting up the full matrix, as opposed to the full matrix's $\mathcal{O}(n^2)$ storage, setup, and application complexity.

For eigenvalue and eigenvector computations, we have discussed the current alternative method of choice in the Nyström extension and developed a hybrid method that allows the basic Nyström idea to benefit from NFFT-based fast matrix-vector products. In our numerical experiments, we found that the Nyström-Gaussian-NFFT method achieved much better eigenvalue accuracy than the traditional Nyström extension even for a significantly smaller parameter L, but was in turn outperformed by the NFFT-based Lanczos method.

In strongly eigenvector-dependent applications like in Section 6.2.2, the higher accuracy of the NFFT-based Lanczos method directly leads to better classification results. In some other applications, however, it is hard to predict if better eigenvector accuracy distinctly improves the results. For instance in Section 6.2.1, the traditional Nyström extension still achieved good image clusterings on average with small parameter L despite its rather inaccurate eigenvectors. Here, the NFFT-based Lanczos method still has very good selling points in its greatly improved runtime as well as its consistency, while the traditional Nyström tends to "fail" in some test runs.

REFERENCES

[1] J. Baglama and L. Reichel, *Augmented implicitly restarted Lanczos bidiagonalization methods*, SIAM J. Sci. Comput., 27 (2005), pp. 19–42.

[2] M. Belkin and P. Niyogi, *Laplacian eigenmaps and spectral techniques for embedding and clustering*, in Advances in Neural Information Processing Systems 14, 2002, pp. 585–591.

[3] ———, *Laplacian eigenmaps for dimensionality reduction and data representation*, Neural Comput., 15 (2003), pp. 1373–1396.

[4] A. Bertozzi, S. Esedoglu, and A. Gillette, *Inpainting of binary images using the Cahn–Hilliard equation*, IEEE Trans. Image Process., 16 (2007), pp. 285–291.

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

[6] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[7] J. Bosch, S. Klamt, and M. Stoll, *Generalizing diffuse interface methods on graphs: non-smooth potentials and hypergraphs*, SIAM J. Appl. Math., 78 (2018), pp. 1350–1377.

[8] F. R. K. Chung, *Spectral graph theory*, vol. 92 of CBMS Regional Conference Series in Mathematics, Amer. Math. Soc., Providence, RI, 1997.

[9] P. Drineas and M. W. Mahoney, *On the Nyström method for approximating a Gram matrix for improved kernel-based learning*, J. Mach. Learn. Res., 6 (2005), pp. 2153–2175.

[10] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, *Spectral grouping using the Nyström method*, IEEE Trans. Pattern Anal. Mach. Intell., 26 (2004), pp. 214–225.

[11] C. Garcia-Cardona, E. Merkurjev, A. L. Bertozzi, A. Flenner, and A. G. Percus, *Multiclass data segmentation using diffuse interface methods on graphs*, IEEE Trans. Pattern Anal. Mach. Intell., 36 (2014).

[12] G. Gilboa and S. Osher, *Nonlocal operators with applications to image processing*, Multiscale Model. Simul., 7 (2008), pp. 1005–1028.

[13] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, third ed., 1996.

[14] M. Hein, S. Setzer, L. Jost, and S. S. Rangapuram, *The total variation on hypergraphs – learning on hypergraphs revisited*, in Advances in Neural Information Processing Systems 26, 2013, pp. 2427–2435.

[15] M. Henaff, J. Bruna, and Y. LeCun, *Deep convolutional networks on graph-structured data*, arXiv preprint, (2015). http://arxiv.org/abs/1506.05163v1.

[16] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Stand, 49 (1952), pp. 409–436.

[17] A. Iske, S. L. Borne, and M. Wende, *Hierarchical matrix approximation for kernel-based scattered data interpolation*, SIAM Journal on Scientific Computing, 39 (2017), pp. A2287–A2316.

[18] J. Keiner, S. Kunis, and D. Potts, *Using NFFT3 - a software library for various nonequispaced fast Fourier transforms*, ACM Trans. Math. Software, 36 (2009), pp. 19:1–19:30.

[19] S. Kunis, D. Potts, and G. Steidl, *Fast Gauss transform with complex parameters using NFFTs*, J. Numer. Math., 14 (2006), pp. 295–303.

[20] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Stand, 45 (1950), pp. 255–282.

[21] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, 1998.

[22] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci. USA, 104 (2007), pp. 20167–20172.

[23] X. Luo and A. L. Bertozzi, *Convergence of the Graph Allen–Cahn Scheme*, J. Stat. Phys., 167 (2017), pp. 934–958.

[24] P.-G. Martinsson, *Randomized methods for matrix computations*, arXiv preprint, (2018). http://arxiv.org/abs/1607.01649v2.

[25] E. Merkurjev, T. Kostic, and A. L. Bertozzi, *An MBO scheme on graphs for classification and image processing*, SIAM J. Imaging Sci., 6 (2013).

[26] 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 21, Curran Associates, Inc., 2009, pp. 1113–1120.

[27] F. Nestler, *Automated parameter tuning based on RMS errors for nonequispaced FFTs*, Adv. Comput. Math., 42 (2016), pp. 889–919.

[28] A. Y. Ng, M. I. Jordan, and Y. Weiss, *On spectral clustering: Analysis and an algorithm*, in Advances in Neural Information Processing Systems 14, 2002, pp. 849–856.

[29] C. C. Paige and M. A. Saunders, *Solutions of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal, 12 (1975), pp. 617–629.

[30] B. N. Parlett, *The symmetric eigenvalue problem*, vol. 20 of Classics in Applied Mathematics, SIAM, Philadelphia, PA, 1998. Corrected reprint of the 1980 original.

[31] D. Potts and G. Steidl, *Fast summation at nonequispaced knots by NFFTs*, SIAM J. Sci. Comput., 24 (2003), pp. 2013–2037.

[32] D. Potts, G. Steidl, and A. Nieslony, *Fast convolution with radial kernels at nonequispaced knots*, Numer. Math., 98 (2004), pp. 329–351.

[33] C. Robert, *Machine learning, a probabilistic perspective*, Taylor & Francis, 2014.

[34] Y. Romano, M. Elad, and P. Milanfar, *The little engine that could: Regularization by denoising (RED)*, SIAM J. Imaging Sci., 10 (2017), pp. 1804–1844.

[35] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, Philadelphia, PA, 2003.

[36] Y. Saad and M. H. Schultz, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput, 7 (1986), pp. 856–869.

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

[38] V. Simoncini and D. B. Szyld, *Theory of inexact Krylov subspace methods and applications to scientific computing*, SIAM J. Sci. Comput., 25 (2003), pp. 454–477.

[39] G. L. Sleijpen and H. A. Van der Vorst, *A Jacobi–Davidson iteration method for linear eigenvalue problems*, SIAM review, 42 (2000), pp. 267–293.

[40] D. A. Spielman and S.-H. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, ACM, 2004, pp. 81–90.

[41] G. Stewart, *A Krylov-Schur algorithm for large eigenproblems*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 601–614.

[42] Y. van Gennip, N. Guillen, B. Osting, and A. L. Bertozzi, *Mean curvature, threshold dynamics, and phase field theory on finite graphs*, Milan J. Math., 82 (2014), pp. 3–65.

[43] T. Volkmer, *OpenMP parallelization in the NFFT software library*, tech. rep., Preprint 2012-07, Faculty of Mathematics, Technische Universität Chemnitz, 2012.

[44] U. von Luxburg, *A tutorial on spectral clustering*, Stat. Comput., 17 (2007), pp. 395–416.

[45] C. Williams and M. Seeger, *Using the Nyström Method to Speed Up Kernel Machines*, in Advances in Neural Information Processing Systems 13, MIT Press, 2001, pp. 682–688.

[46] C. Yang, R. Duraiswami, N. A. Gumerov, and L. Davis, *Improved fast gauss transform and efficient kernel density estimation*, in null, IEEE, 2003, p. 464.

[47] L. Zelnik-Manor and P. Perona, *Self-tuning spectral clustering*, in Advances in Neural Information Processing Systems 17, MIT Press, 2004, pp. 1601–1608.

[48] 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 16, 2003.