# Image Processing using Graph Laplacian Operator

David Wobrock
david.wobrock@gmail.com

March 22, 2018

Under the responsability of:

Supervisor: Frédéric Nataf
ALPINES Team - INRIA Paris
Laboratoire Jacques-Louis Lions - Sorbonne Université

INSA Supervisor: Christine Solnon
Département Informatique
INSA de Lyon

**Abstract**

The latest image processing methods are based on global data-dependent filters. These methods involve huge affinity matrices which cannot fit in memory and need to be approximated using spectral decomposition. The inferred eigenvalue problem concerns the smallest eigenvalues of the Laplacian operator which can be solved using the inverse iteration method which, in turn, involves solving linear systems. In this master thesis, we detail the functioning of the spectral algorithm for image editing and explore the behaviour of solving large and dense systems of linear equations in parallel, in the context of image processing. We use Krylov type solvers, such as GMRES, and precondition them using domain decomposition methods to solve the systems on high-performance clusters. The experiments show that Schwarz methods as preconditioner scale well as we increase the number of processors. However, we observe that the limiting factor is the Gram-Schmidt orthogonalisation procedure. We also compare the performances to the state-of-the-art Krylov-Schur algorithm.

**Keywords** image processing, graph Laplacian operator, eigenvalue problem, high performance computing

**Résumé**

Les méthodes récentes de traitement d'images sont basées sur des filtres globaux dépendant des données. Ces méthodes impliquent le calcul

de matrices de similarités de grandes tailles qui ne peuvent pas être contenues en mémoire et nécessitent ainsi d'être approchées par une décomposition spectrale. Le problème aux valeurs propres qui en découle concerne les plus petites valeurs propres du Laplacien qui peuvent être calculées grâce à la méthode de la puissance inverse qui implique la résolution de systèmes linéaires. Dans cette thèse de master, nous détaillons le fonctionnement de l'algorithme spectral pour l'édition d'images et étudions le comportement de la résolution de systèmes d'équations linéaires sur de grandes matrices pleines en parallèle, dans le cadre du traitement d'images. Nous utilisons des solveurs de type Krylov, tel que GMRES, et les préconditionnons avec des méthodes de décomposition de domaines pour résoudre les systèmes sur des clusters haute performance. Les expériences montrent que les méthodes de Schwarz comme préconditionneur passent bien à l'échelle quand on augmente le nombre de processeurs. Toutefois, nous observons que le facteur limitant est la procédure d'orthogonalisation de Gram-Schmidt. Nous comparons également les performances avec l'algorithme de l'état de l'art Krylov-Schur.

**Mots-clés** traitement d'images, Laplacien graphe, problème aux valeurs propres, calcul haute performance

# 1 Introduction

## 1.1 Background

The talk [1] and articles [2] [3] by Milanfar, working at Google Research, about using the graph Laplacian operator for nonconformist image processing purposes awakes curiosity.

Indeed, Milanfar reports that these techniques to build image filters are used on smartphones, which implies a reasonable execution time with limited computational resources. Over 2 billion photos are shared daily on social media [1], with very high resolutions and most of the time some processing or filter is applied to them. The algorithm must be efficient to be deployed at such scale.

## 1.2 Objective

The aim of this degree project is not to explore and improve the state of image processing. Instead, the spectral methods used in the algorithm will be our focus point. Those will inevitably expose eigenvalue problems, which may involve solving systems of linear equations.

Concerning the challenges about solving linear systems, on one hand, the size of the systems can be large considering high-resolution images with millions of pixels, or even considering 3D images. We process huge matrices of size $N^2$, with $N$ the number of pixels of the input image. On the other hand, those huge affinity matrices are dense, thus the linear systems are dense. Often, linear systems result from discretising partial differential equations (PDEs)

yielding sparse matrices, and therefore most linear solvers are specialised in sparse systems.

We want to explore the performance of linear solvers on dense problems, their stability and convergence. This will include preconditioning the linear systems, especially using domain decomposition methods, and analyse their behaviour on dense systems.

# 2 Image processing using graph Laplacian operator

A global image filter consists of a function which outputs one pixel, taking all pixels as input and applying weights to them.

As a practical notation, we say that, with $W$ the matrix of weights and $y$ and $z$ respectively the input and output images as vectors,

$$z = Wy.$$

The filter matrix $W$ considered here is data-dependent and built upon the input image $y$. A more mathematical notation would consider $W$ as a nonlinear function of the input image such as $z = W(y) \cdot y$.

**Image as graph** Let's think of an image as a graph. Each pixel is a node and has edges to other nodes. The simplest way to connect pixels to other pixels is their direct neighbours, in which case each node has four edges. To avoid losing any information, we will instead consider the case of a complete graph; each node connects to all other nodes.

To preserve the image information in the graph, the graph edges will be assigned a weight, measuring the similarity[1] between the two nodes, thus between two pixels.

There are multiple ways the similarity can be defined. The most intuitive definition considers spatial proximity. This means that similar pixels are spatially close, which, translated to a basic filter, is the same as a Gaussian filter which computes a weighted average of the pixel's neighbourhood and produces what is known as Gaussian blur. Another similarity definition is to consider the pixel's colour. A good compromise is to consider an average of both, spatial and colour closeness, with a certain weighting.

Once the similarity is defined, we can compute the adjacency matrix of the graph including the edge weights. We will call this matrix the affinity matrix[2] $K$ which represents the similarity of each pixel to every other pixel in the image. Consequently, this matrix is symmetric and of size $N \times N$ with $N$ the number of pixels in the image. Also, most similarity functions define bounds on the values of $K$ such as $0 \leq K_{ij} \leq 1$.

[1] Also called affinity.

[2] Or similarity matrix, or kernel matrix

Using this affinity matrix, we obtain the graph Laplacian $\mathcal{L}$, used to build the filter matrix $W$.

**Building the filter** Multiple graph Laplacian definitions, more or less equivalent, exist and can have slightly different properties. In the case of image smoothing, the filter $W$ is roughly defined such as $\mathcal{L} = I - W$ [1] and so $W = I - \mathcal{L}$. To get various filters using one Laplacian, we can apply some function $f$ to $\mathcal{L}$ and obtain $W = I - f(\mathcal{L})$ which gives us more possibilities on the filter computation.

However, all these matrices $K$, $\mathcal{L}$ and $W$ represent huge computational costs. Only storing one of these matrices is already a challenge since they have a size of $N^2$.

For example, a tiny test image of size $256 \times 256$ has 65 536 pixels, so one of these matrices has approximately $4.29 \times 10^9$ elements. Considering storing those with a 64 bits type, one matrix takes more than 34 GB of memory. Scaling to a modern smartphone picture, taken with a 10 megapixel camera, a matrix contains $10^{14}$ elements, meaning 800 TB of memory for each matrix.

**Approximation by sampling and Nyström extension** To avoid storing any of those huge matrices, approximation will be necessary. Following [4], we define the Nyström extension. It starts by sampling the image and only select a subset of $p$ pixels, with $p \ll N$. Numerically, $p$ should represent around 1% or less of the image pixels. We only need to compute of subset of the affinity matrix $K$ and then compute the eigendecomposition of a submatrix. The eigenvectors of the submatrix are extended to the entire matrix and used to reconstruct it.

Therefore, we will only need to compute two submatrices. For our previous examples, if we sample 1% of the pixels, we need to store 0.34 GB of data for each matrix, instead of 34 GB for the $256 \times 256$ image. For a 10 megapixel image, each matrix needs 8 TB of memory, which is still a lot of memory. However, as [4] and [2] study, the sampling rate can be lower than 1% and still contain most of the relevant image information.

**Eigendecomposition** We need to compute the largest eigenvalues of the filter $W$. The reason can be found by formulating the filter by its diagonalisation $W = \sum_i^N \lambda_i \phi_i \phi_i^T$. We know that the eigenvalues of $W$ are non-negative. When the $i$th eigenvalue $\lambda_i$ is small and tends to 0, the eigenvector product will be negligible, and therefore the largest eigenvalues of the filter $W$ are the most relevant.

It can easily be shown that the largest eigenvalues of the submatrix filter is equivalent to computing the smallest eigenvalues of the corresponding Laplacian operator.

The goal of this observation is the way of computing the eigenvalues. For the largest eigenvalues, the most famous algorithm is the power method. For the smallest eigenvalues, the inverse power method is a usual choice. Both methods converge faster when two successive eigenvalues are far from each other. In broad

strokes, the power method requires many cheap iterations, whereas the inverse iteration needs few expensive iterations. In our case, the largest eigenvalues are close to each other; hence the inverse of these eigenvalues will be far from each other. We will therefore prefer the inverse power method.

The algorithm will, in an iterative manner, compute the associated eigenvector of an eigenvalue. This requires either to invert a matrix such as $x_{k+1} = A^{-1}x_k$, or to solve the linear system $Ax_{k+1} = x_k$. We will solve systems of linear equations to compute the first eigenvalues of the Laplacian in order to observe the behaviour of solvers on these dense matrices.

The main drawback of the Nyström method for us, is that it approximates the leading eigenvalues but we compute the trailing ones of the Laplacian $\mathcal{L}$ [5]. It is possible to obtain the eigendecomposition of the filter $W$, even when $W$ is indefinite, through a method proposed by [4]. It consists of computing the inverse square root matrix $W_A^{-1/2}$, which could be done either by the complete eigendecomposition or by using Cauchy's integral formula. After this step, two more diagonalisation of matrices are required, demanding an important computation time.

Nevertheless, as stated in the objectives of the project, our main goal is not the image processing aspect, but the behaviour of linear solvers of these dense matrices using domain decomposition methods. We will therefore stick to computing the smallest eigenvalues of the Laplacian operator $\mathcal{L}$ and avoid spending too much time on the end of the algorithm implementation.

Below a summary of the complete algorithm using spectral decomposition of the matrix to approximate it:

**Algorithm 1** Image processing using approximated graph Laplacian operator

**Input:** $y$ an image of size $N$, $f$ the function applied to $\mathcal{L}$
**Output:** $\tilde{z}$ the output image by the approximated filter
{Sampling}
Sample $p$ pixels, $p \ll N$
{Kernel matrix approximation}
Compute $K_A$ (size $p \times p$) and $K_B$ (size $p \times (N-p)$)
Compute the Laplacian submatrices $\mathcal{L}_A$ and $\mathcal{L}_B$
{Eigendecomposition}
Compute the $m$ smallest eigenvalues $\Pi_A$ and the associated eigenvectors $\Phi_A$ of $\mathcal{L}_A$
{Nyström extension and compute the filter}
See methods of solution proposed by [4]
$\tilde{z} \leftarrow \tilde{W}y$

# 3 Implementation

## 3.1 Parallel implementation

To scale our algorithm to use usual camera pictures, but also much larger inputs, we implemented it in a parallel manner using the C language and the Portable, Extensible Toolkit for Scientific Computation (PETSc) [6]. This library is built upon MPI and contains distributed data structures and parallel scientific computation routines. In a nutshell, PETSc provides an impressive parallel linear algebra toolkit which is very useful to shorten the development time.

In order to verify the correctness of our implementation, we used the Scalable Library for Eigenvalue Problem Computation (SLEPc) [7], which is based on PETSc and provides parallel eigenvalue problem solvers. Furthermore, we need the library Elemental [8] in order to do dense matrix operations in PETSc.

The implementation associated to this project is open source and can be found on GitHub[3].

## 3.2 Inverse subspace method

The used algorithm to compute the smallest eigenvalues is the inverse subspace iteration inspired by [9].

Solving the systems of linear equations is done using the Krylov type solvers and the preconditioners included in PETSc. As a standard approach, we use GMRES as our solver and the RAS method as preconditioner, without overlap and 2 domains per process. Each subdomain is solved using the GMRES method also.

On each outer iteration, we must compute the residuals to see if we converged. This requires multiple matrix-matrix products and computing a norm, so communication cannot be avoided here.

## 3.3 Entire matrix computation

We start by showing the result of the computation using the full matrices in order to see the image processing result. We limit ourselves to grayscale images for the beginning. As stated before, computing the entire matrices can only be done on small images. The image below contains 135 000 pixels, so each matrix needs around 145 GB.

[3] `https://github.com/David-Wobrock/image-processing-graph-laplacian/`

Figure 1 – Left: input image. Right: sharpened image.

The cat's fur on the left-hand side, the person's hand and the cushion on the right-hand side appear to be more detailed. We observe that the already sharp part of the image, such as the cat's head, stays nice and is not over-sharpened. We obtained this filter by defining $f(\mathcal{L}) = -3\,\mathcal{L}$ in the output image $z = (I - f(\mathcal{L}))y$.

This corresponds to the adaptive sharpening operator defined in [1] as $(I + \beta\,\mathcal{L})$ with $\beta > 0$. This approach remains a simple application of a scalar and doesn't require any eigenvalue computation. A more complete approach is called multiscale decomposition [3] and consists of applying a polynomial function to the Laplacian $\mathcal{L}$. It applies various coefficients to different eigenvalues of $\mathcal{L}$ because each eigenpair captures particular features of the image.

### 3.4 Approximate matrix computation

As a reminder, we used GMRES to solve the linear systems with RAS preconditioning and using GMRES on the subdomains. We sample 1% of the pixels of an image with $4 \cdot 10^5$ pixels using spatially uniform sampling. The performances of the inverse subspace iteration for 50 and 500 eigenvalues:

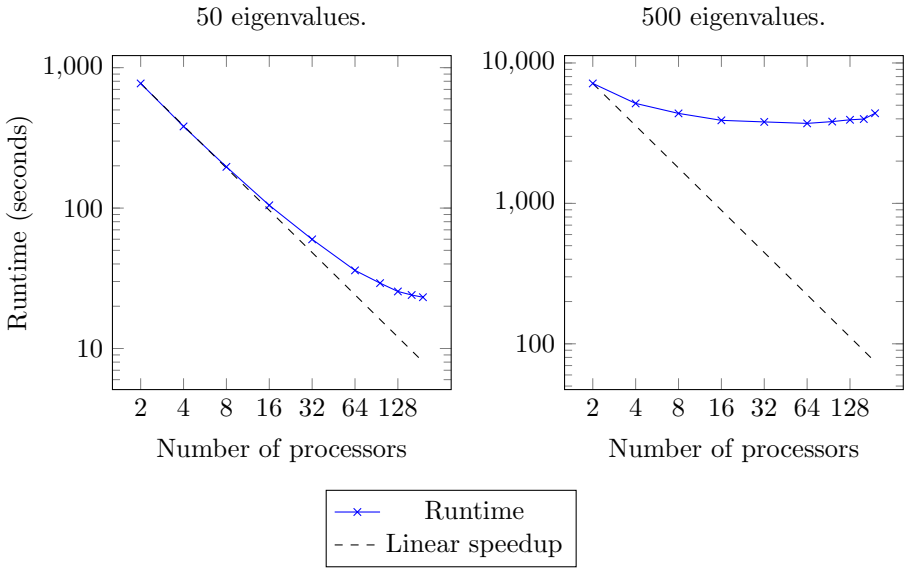


Figure 2 – Runtime of the inverse subspace iteration part of the algorithm (log scale).

When increasing a low number of processors, we see an improvement of the performances in both cases. But the runtime stagnates slowly for 50 eigenvalues and quickly for 500 eigenvalues. We even observe a raise of the runtime for 500 eigenvalues. The algorithm reaches its parallelisation limit and the communication overhead takes over. For 500 eigenvalues, the runtime for 2 processors is over 7000 seconds, and the fastest runtime is reached for 64 processors and is of 3700 seconds.

We know that the inverse iteration part of the algorithm is not scaling correctly compared to the other parts. For any amount of computed eigenvalues, when we increase the number of processes, the proportion of time spent computing the eigenvalues increases. For 500 eigenvalues and 128 processors, we spend more than 99% of the time computing the eigenvalues. This confirms that the algorithm does not quite scale yet.

We look at the internal steps of the inverse power method to see where lies the problem. The algorithm consists of iteratively solving $m$ linear systems, orthonormalising the vectors and computing the residual norm. Here is the proportion of each step of the inverse subspace interation for the computation of 50 and 500 eigenvalues:

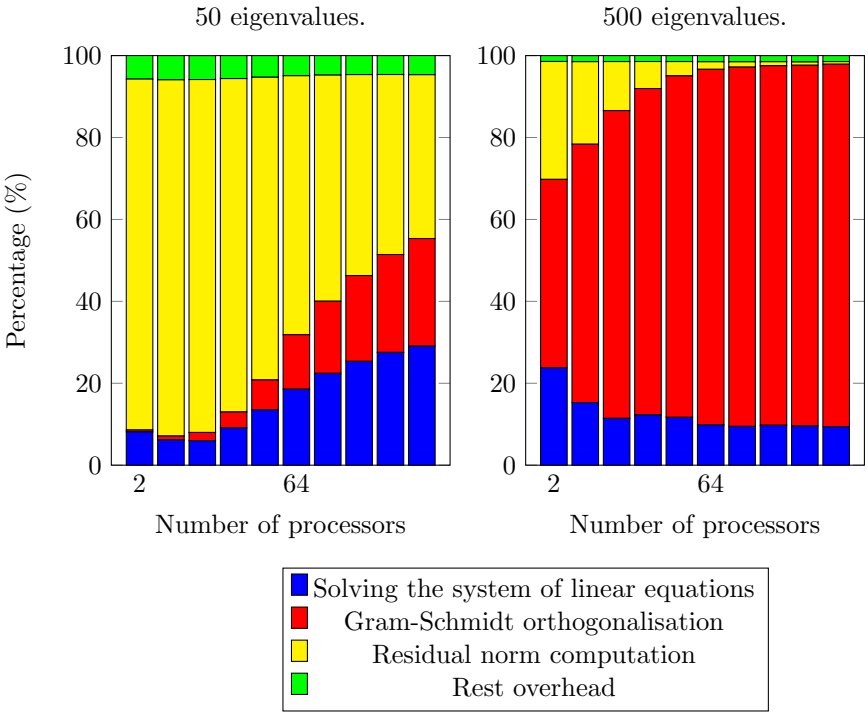


Figure 3 – Proportion of each step in the inverse subspace iteration.

We observe that the Gram-Schmidt orthogonalisation is the limiting factor and is the most time-consuming step of the inverse iteration as the number of processors grows. It is a well-known problem that the simple Gram-Schmidt process is actually difficult to parallelise efficiently. Small optimisations for a parallel Gram-Schmidt orthogonalisation exist [10] but they do not properly solve the problem. This issue will be difficult to overcome completely.

### 3.5 Skipping some orthogonalisations

Fundamentally, the orthogonalisation is used to stabilise the algorithm. To accelerate our algorithm further, we try to orthogonalise the vectors $X_k$ less often and skip some Gram-Schmidt procedures.

Below the runtime for 2 and 64 processors of the inverse subspace algorithm depending on the frequency of orthogonalisation:

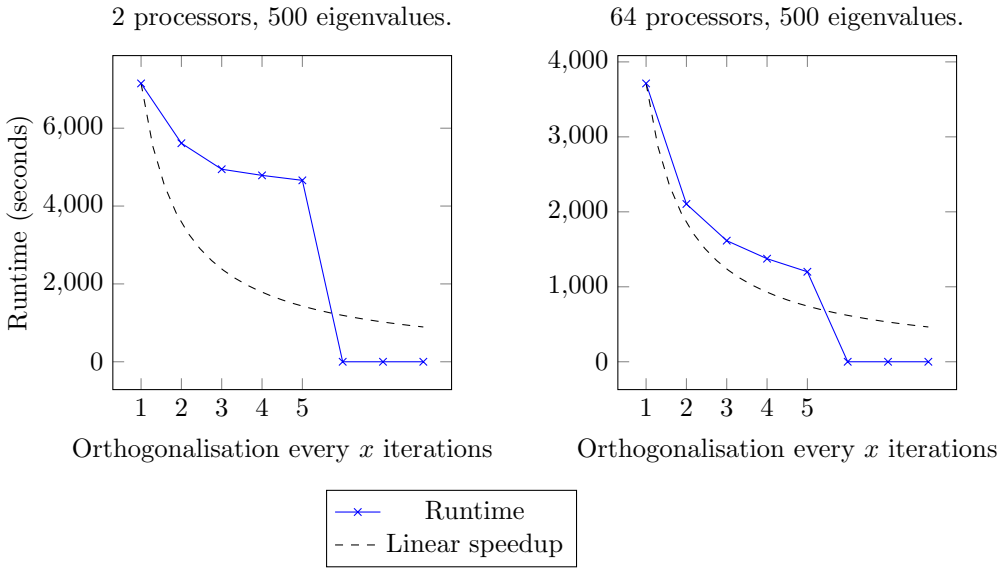


Figure 4 – Runtime of the inverse subspace iteration depending on the amount of Gram-Schmidt procedures.

For 2 processors, we see that the speedup stagnates quickly when skipping the Gram-Schmidt orthogonalisation more and more often. Indeed, the orthogonalisation represents a decent proportion of the execution time for 2 processors whereas it represents over 85% for 64 processors. Therefore we observe a speedup of the inverse subspace iteration that is nearly linear for 64 processors. The number of outer iterations between not skipping the Gram-Schmidt algorithm and applying it every 5 iterations only varies from 38 to 40 which explains the resulting runtime.

# 4 Conclusion

## 4.1 Discussions

**Linear solvers & domain decomposition methods on dense matrices** As we seen through the experiments, the resolution of linear systems scales slightly with respect to the number of processors. Domain decomposition methods improve the performances of solving dense systems of linear equations, even without overlap. In our small case, a direct solver showed the best results for preconditioning. This might not be the case for larger systems.

**Gram-Schmidt process** We saw that the orthogonalisation process is difficult to parallelise efficiently. Skipping the Gram-Schmidt procedure every other

iteration, to stabilise the algorithm less often, gave an improvement, but we cannot totally avoid the cost of it when increasing the number of processors. This problem is well-known and one of the biggest limitations for scaling diverse algorithms to a large number of processors.

The Gram-Schmidt procedure orthogonalises a set of vectors by sequentially substracting from a vector the projections on the previously orthogonalised vectors. The inner product of two vectors is computed frequently, because of the projection, and since each vector is shared over all processors, a lot of communication is involved in this operation. Attempts for parallel implementation are numerous, like [10], but they either still have many communications or they suggest a different memory distribution schema.

## 4.2 Perspectives

**Image processing** An improvement of our algorithm would be to finish the part computing the output image. This requires to compute the inverse of the square root of the matrix.

A way to improve the filtering is multiscale decomposition. As explained in [3], instead of applying a linear function to all eigenvalues such as $f(W) = \phi f(\Pi)\phi^T$, we can actually use a polynomial function $f$. This is interesting because each eigenpair captured various features of the image and one can apply different coefficients on different aspects of the image.

For the state-of-the-art, the article [11] proposes an enhancement of global filtering. It argues that the eigendecomposition remains computationally too expensive and shows results of an improvement. The presented results and performances are astonishing; however, the method is hardly described and replicating it would be difficult. This is understandable since this algorithm seems to be in the latest Pixel 2 smartphone by Google and they want to preserve their market advantage in the field of image processing.

A real improvement would be to formulate a method for extending the trailing eigenvectors of the sampled Laplacian $\mathcal{L}_A$. This way, it would be possible to apply the spectral decomposition of the Laplacian, and thus apply a filter to the input image.

**Linear solver** A way to highly parallelise the matrix computations could be using graphical processing units (GPUs). Especially the matrix-matrix and matrix-vector products could be nicely improved with GPUs. However, solving systems of linear equations is a task that GPUs are not designed for.

It would be interesting to explore more the impact of the number of sampled pixels, which corresponds to the input matrix of the linear system. The articles [4] and [2] started a study on the size of the samples, but only for small images. This work could be extended.

# References


[1] Peyman Milanfar. *Non-conformist Image Processing with Graph Laplacian Operator*. 2016. URL: https://www.pathlms.com/siam/courses/2426/sections/3234.

[2] Hossein Talebi and Peyman Milanfar. "Global Image Denoising". In: *IEEE Transactions on Image Processing* 23.2 (Feb. 2014), pp. 755–768. ISSN: 1057-7149, 1941-0042. DOI: 10.1109/TIP.2013.2293425. URL: http://ieeexplore.ieee.org/document/6678291/.

[3] H. Talebi and P. Milanfar. "Nonlocal Image Editing". In: *IEEE Transactions on Image Processing* 23.10 (2014), pp. 4460–4473. ISSN: 1057-7149. DOI: 10.1109/TIP.2014.2348870.

[4] Charless Fowlkes et al. "Spectral grouping using the Nystrom method". In: *IEEE transactions on pattern analysis and machine intelligence* 26.2 (2004), pp. 214–225. URL: http://ieeexplore.ieee.org/abstract/document/1262185/.

[5] Serge Belongie et al. "Spectral partitioning with indefinite kernels using the Nyström extension". In: *European conference on computer vision*. Springer, 2002, pp. 531–542.

[6] Satish Balay et al. *PETSc Web page*. http://www.mcs.anl.gov/petsc. 2017. URL: http://www.mcs.anl.gov/petsc.

[7] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems". In: *ACM Trans. Math. Software* 31.3 (2005), pp. 351–362.

[8] Jack Poulson et al. "Elemental: A New Framework for Distributed Memory Dense Matrix Computations". In: *ACM Trans. Math. Softw.* 39.2 (Feb. 2013), 13:1–13:24. ISSN: 0098-3500. DOI: 10.1145/2427023.2427030. URL: http://doi.acm.org/10.1145/2427023.2427030.

[9] G. El Khoury, Yu. M. Nechepurenko, and M. Sadkane. "Acceleration of inverse subspace iteration with Newton's method". In: *Journal of Computational and Applied Mathematics*. Proceedings of the Sixteenth International Congress on Computational and Applied Mathematics (ICCAM-2012), Ghent, Belgium, 9-13 July, 2012 259 (Mar. 2014), pp. 205–215. ISSN: 0377-0427. DOI: 10.1016/j.cam.2013.06.046. URL: http://www.sciencedirect.com/science/article/pii/S0377042713003440.

[10] Takahiro Katagiri. "Performance Evaluation of Parallel Gram-Schmidt Re-orthogonalization Methods". In: *High Performance Computing for Computational Science — VECPAR 2002*. Ed. by José M. L. M. Palma et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 302–314. ISBN: 978-3-540-36569-3.

[11] Hossein Talebi and Peyman Milanfar. "Fast Multilayer Laplacian Enhancement". In: *IEEE Transactions on Computational Imaging* 2.4 (Dec. 2016), pp. 496–509. ISSN: 2333-9403, 2334-0118. DOI: 10.1109/TCI.2016.2607142. URL: http://ieeexplore.ieee.org/document/7563313/.