Aggregation-based Adaptive Algebraic Multigrid for Sparse Linear Systems

Eran Treister
Aggregation-based Adaptive Algebraic Multigrid for Sparse Linear Systems

Research Thesis

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Eran Treister

Submitted to the Senate of the Technion
Israel Institute of Technology

Elul 5774 Haifa September 2014
The Research Thesis Was Done Under the Supervision of Prof. Irad Yavneh in the Department of Computer Science.

Acknowledgements

I would like to thank Prof. Irad Yavneh for his invaluable guidance throughout this research, and for being a friend and teammate whose door is always open. His original ideas, experienced suggestions and great help were crucial to this research. I hope to keep working with you in the future; you made this experience truly meaningful and enjoyable.

I want to thank my mentors and friends in the Technion for teaching me and sharing this experience with me. You have made my academic journey very special, and sparked my interest in computational mathematics. I would also like to thank my colleagues and mentors in the Multigrid community for fascinating collaborations and discussions that have enriched my experience as a researcher. Thanks to you, attending the Copper mountain conferences and AMG summit truly feels like a visiting a family, and I hope to continue attending those and strengthen our friendship in the future. Lastly, I would like to thank the members of my thesis committee for their insightful remarks and valuable advice.

Most importantly, I would like to thank my dear wife Dorit for her support, love, patience, and encouragement. It was an absolute pleasure to share this experience with you, and to raise our two children Kfir and Eyal, along the way. I would like to thank my parents, Shmuel and Kochava, and Dorit’s parents, Lisa and Ilan, for their great help and love throughout this journey.

My work would never be the same without you all.

Funding Acknowledgements

- Eran Treister is grateful to the Azrieli Foundation for the award of an Azrieli Fellowship. Being part of this fellowship has been a great honor and a truly enriching experience.
- This research was supported by the Israeli Science Foundation, grant number 795/08.
- This research was funded (in part) by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI).
Journal papers included in this dissertation


*Shorter versions of these papers have also appeared as conference papers.

Conference and journal papers not included in this dissertation


## Contents

1 Abstract 1
2 Definitions, notations and acronyms. 3
3 Introduction 9

4 General Background 13
   4.1 Non-negative Matrices, Stochastic Matrices, M-matrices 13
   4.2 Algebraic Multigrid Methods 16
      4.2.1 Classical Algebraic Multigrid (AMG) 19
      4.2.2 Aggregation-Based AMG 23
      4.2.3 Pure Aggregation AMG 25
      4.2.4 Smoothed Aggregation AMG 27
   4.3 The Adaptive Multigrid Framework 29

5 Aggregation-based Adaptive AMG for Markov Chains 31
   5.1 Introduction: Markov Chains 31
   5.2 Adaptive AMG for Markov Chains: The Exact Interpolation Scheme 34
      5.2.1 The General Exact Interpolation Scheme 34
      5.2.2 Aggregation-based Exact Interpolation Scheme 36
      5.2.3 Aggregation—a bottom-up approach 37
      5.2.4 Smoothing the tentative operators 40
      5.2.5 Analysis of Aggregation vs. Smoothed Aggregation 42
   5.3 On-the-fly Adaptive Algebraic Multigrid for Markov Chains 45
      5.3.1 The relation between the classical and EIS algorithms 46
      5.3.2 Optimizing the use of EIS and classical cycles 47
      5.3.3 A qualitative analysis 54
   5.4 Numerical Results 56
      5.4.1 Serial environment experiments 58
## CONTENTS

### 5.4.2 Parallel environment simulation

5.5 Conclusions and Future Work

### 6 Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

6.1 Introduction

6.2 Motivation and Background

6.3 The Sparsified Smoothed Aggregation (SpSA) Algorithm
   6.3.1 Sparsity Patterns in the Aggregation Framework
   6.3.2 The Sparsening Procedure
   6.3.3 Theoretical results

6.4 Numerical Results
   6.4.1 Non-symmetric test cases: convection-diffusion
   6.4.2 Diffusion with discontinuous coefficients
   6.4.3 Random graph-Laplacian problems
   6.4.4 Communication complexity

6.5 Conclusions

### 7 A Multilevel Approach for $l_1$ Penalized Least Squares (LASSO)

7.1 Introduction

7.2 Iterated shrinkage methods and accelerations

7.3 A multilevel iterated shrinkage approach
   7.3.1 Definition of the low-level problem
   7.3.2 Choosing the low-level variables
   7.3.3 Definition of the multilevel V-cycle

7.4 Theoretical properties of the multilevel V-cycle
   7.4.1 Treatment of the lowest level
   7.4.2 A gradual initialization—“full multilevel cycle”
   7.4.3 A note on implementation

7.5 Numerical Results
   7.5.1 Experiment 1: Well-conditioned random $A$
   7.5.2 Experiment 2: Random $\pm 1$ entries $A$
   7.5.3 Experiment 3: ill-conditioned $A$
   7.5.4 Experiment 4: well-conditioned $A$ with similar columns
CONTENTS

7.5.5 Discussion ........................................... 125
7.6 Conclusions .............................................. 126

8 Conclusions, current and future work ........................................ 127
  8.1 Current and future work ........................................ 128

A Adaptive AMG for Markov Chains ........................................ 131
  A.1 Analysis of the Weighted Power Method for Complex Spectrum ............ 131
  A.2 Analysis of Smoothed Aggregation for Complex Spectrum .................. 132
  A.3 Acceleration of SA when only $P$ is smoothed ................................ 134

B Smoothed Aggregation—a modified version. .................................. 135
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Multilevel V-cycle and W-cycle. ‘▽’ refers restricting the residual, ‘□’ refers to performing a coarse-grid solve, ‘↗’ refers to prolonging the correction, and ‘•’ refers to applying relaxations.</td>
</tr>
<tr>
<td>2</td>
<td>Algebraically smooth error for a simple 2D problem, discretized by finite elements, taken from [55].</td>
</tr>
<tr>
<td>3</td>
<td>Interpolation error for the 1D discrete Laplacian operator: Upper drawing: approximation of a smooth mode (e) by a piece-wise constant interpolation from a coarser grid vector (e_c) ((e_c = \arg \min_{v_c \in \mathbb{R}^{nc}} |e - P_tv_c|_2)). Lower drawing: the approximation error is shown to contain high-frequency modes.</td>
</tr>
<tr>
<td>4</td>
<td>An example of a Markov chain with five states. The transition graph is on the left and the corresponding column-stochastic matrix is on the right.</td>
</tr>
<tr>
<td>5</td>
<td>BottomUp algorithm using (s = 4). Bold/dashed lines refer to connections within/between the aggregates. Left—2D lattice with uniform weights. Right—Unstructured planar graph.</td>
</tr>
<tr>
<td>6</td>
<td>Overlapping on-the-fly adaptive AMG: A scheme for combining setup improvements in parallel to classical solution cycles for the solution of Markov chains. Solution cycles are denoted by “(V_{SOL})”, and “OP” denotes the hierarchy of multigrid operators.</td>
</tr>
<tr>
<td>7</td>
<td>Left: Nonsymmetric random planar graph—directed edges are marked with dotted lines while undirected edges are marked with a solid line. Right—the spectrum of the Markov chain bounded by the unit circle.</td>
</tr>
<tr>
<td>8</td>
<td>Triangular lattice of 28 states.</td>
</tr>
<tr>
<td>9</td>
<td>Tandem Queue: Parallel environment results.</td>
</tr>
<tr>
<td>10</td>
<td>A random walk on a nonsymmetric random planar graph: Parallel environment results.</td>
</tr>
<tr>
<td>11</td>
<td>A random walk on a on a triangular lattice: parallel environment results.</td>
</tr>
</tbody>
</table>
12 A comparison between the sparsity patterns of AGG and SA on level 3, for the 3D Poisson equation on a $64^3$ grid, using a 7-point stencil. The left matrix corresponds to the AGG algorithm; It is of size $6538 \times 6538$, and contains 99,878 non-zeros, which is about 15 non-zeros per row on average. The right matrix corresponds to the SA algorithm, and is smaller because of the more aggressive coarsening. It is of size $1645 \times 1645$, and contains 120,299 non-zeros, which is about 73 non-zeros per row on average.

13 The sparsity pattern of the pure aggregation Galerkin operator. The left image shows the sparsity pattern (a graph) of a fine-grid matrix and circles that correspond to aggregates. On the right image the sparsity pattern of the aggregation Galerkin operator. The image shows that the edges between aggregates are preserved on the coarse grid, and no new connections are generated by the coarsening. The image is taken from the homepage of Prof. Yvan Notay.

14 A surrogate path of $(A_g)_{k,j}$. The dashed arrow represents the eliminated entry, while the surrogate path is comprised of solid arrows.

15 Two small graph Laplacians. The right graph is a sparsified version of the left graph with partitioning $(\delta, 1-\delta)$.

16 Optimal partitioning with respect to the two different measures. The left corresponds to minimizing $\| (I - A_c^{-1}A_g) \|_2$ and the right corresponds to minimizing $\| (I - A_c^{-1}A_g) (I - \frac{3}{4}D^{-1}A_g) \|_2$.

17 2D convection-diffusion problems: velocity fields.

18 2D diffusion problems with discontinuous coefficients: square, diamond, and L shapes.

19 Unstructured random planar graph.

20 Illustration of the multilevel V-cycle for $l_1$ penalized LS minimization.

21 Full multilevel cycle initialization. ‘↘’ refers to choosing $C$ and reducing the problem, ‘□’ refers to performing a lowest-level solve, ‘↗’ refers to prolonging the solution, and ‘•’ refers to applying $\nu$ relaxations.

22 The singular values of the matrix $A$ with $n = 512$. 

71

77

80

82

82

89

93

96

109

117

120
23 A comparison between the weighted power method effectiveness when using weights $\omega = 1$ and $\omega = 1/2$, for damping eigenmodes in the unit circle. Absolute value 0 appears in white while absolute value 1 appears in black, with linear gray-scale for values in-between. . . . . . . . . . . . . . . . . . . 132

24 A comparison between $|\lambda_k|$ (left) and $|\lambda_k||1 - \frac{1}{2}\lambda_k|$ (right), for $\lambda_k = (1 - \lambda_B)$ and $\lambda_B$ in the unit circle. Absolute value of 0 appears in white while absolute value of 2 appears in black, with linear gray-scale for values in-between. . . . 133
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Qualitative analysis notation</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>Qualitative analysis: solution costs</td>
<td>55</td>
</tr>
<tr>
<td>3</td>
<td>Qualitative analysis: solution costs in parallel environment using overlapping setup</td>
<td>56</td>
</tr>
<tr>
<td>4</td>
<td>Tandem Queue results</td>
<td>59</td>
</tr>
<tr>
<td>5</td>
<td>Result Summary: Random walk on a nonsymmetric random planar graph</td>
<td>61</td>
</tr>
<tr>
<td>6</td>
<td>Result Summary: Random walk on a Triangular Lattice</td>
<td>62</td>
</tr>
<tr>
<td>7</td>
<td>Spectral equivalence measures. $S_J$ denotes the damped Jacobi operator $(I - \frac{2}{3}D^{-1}A_g)$</td>
<td>88</td>
</tr>
<tr>
<td>8</td>
<td>2D convection-diffusion. ‘#it’: the number of V-cycles, $C_{op}$: the operator complexity, ‘st’: the maximal stencil size in the hierarchy, ‘WU’: the work units count.</td>
<td>90</td>
</tr>
<tr>
<td>9</td>
<td>Convection diffusion 3D. ‘#it’: the number of V cycles, $C_{op}$: the operator complexity, ‘st’: the maximal stencil size in the hierarchy, ‘WU’: the work units measures.</td>
<td>92</td>
</tr>
<tr>
<td>10</td>
<td>2D and 3D diffusion problems on structured meshes. ‘#it’ denotes the number of V cycles, $C_{op}$ is the operator complexity, ‘st’ is the maximal stencil size, and ‘WU’ is the work units measures.</td>
<td>94</td>
</tr>
<tr>
<td>11</td>
<td>Unstructured Graph-Laplacian</td>
<td>97</td>
</tr>
<tr>
<td>12</td>
<td>Communication complexity</td>
<td>99</td>
</tr>
<tr>
<td>13</td>
<td>Experiment 1: Well-conditioned $A$. Mean numbers of iterations (work-units in brackets). The average timings are for $n = 2048$.</td>
<td>122</td>
</tr>
<tr>
<td>14</td>
<td>Experiment 2: Random $\pm 1$ entries $A$. Mean numbers of iterations (work-units in brackets). The average timings are for $n = 2048$.</td>
<td>123</td>
</tr>
<tr>
<td>15</td>
<td>Experiment 3: ill-conditioned $A$. Mean numbers of iterations (work-units in brackets). The average timings are for $n = 2048$.</td>
<td>123</td>
</tr>
<tr>
<td>16</td>
<td>Experiment 4: well-conditioned $A$ with similar columns. Mean numbers of iterations (work-units in brackets). The average timings are for $n = 2048$.</td>
<td>124</td>
</tr>
</tbody>
</table>
## List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Two-level standard cycle</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>Classical Neighborhood-based Aggregation</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>Two-level EIS cycle</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>Bottom-up aggregation</td>
<td>39</td>
</tr>
<tr>
<td>5</td>
<td>AFTER—a simple adaptive AMG scheme</td>
<td>49</td>
</tr>
<tr>
<td>6</td>
<td>On-the-fly adaptive AMG</td>
<td>51</td>
</tr>
<tr>
<td>7</td>
<td>Overlapping on-the-fly adaptive AMG</td>
<td>53</td>
</tr>
<tr>
<td>8</td>
<td>Sparsified Smoothed Aggregation (SpSA) Setup</td>
<td>76</td>
</tr>
<tr>
<td>9</td>
<td>The Sparsening procedure</td>
<td>83</td>
</tr>
<tr>
<td>10</td>
<td>Diagonally Preconditioned Non-linear Conjugate Gradient for LASSO</td>
<td>105</td>
</tr>
<tr>
<td>11</td>
<td>V-cycle for $l_1$ penalized LS minimization</td>
<td>109</td>
</tr>
<tr>
<td>12</td>
<td>Full multilevel cycle initialization</td>
<td>117</td>
</tr>
<tr>
<td>13</td>
<td>Modified Neighborhood-based Aggregation</td>
<td>137</td>
</tr>
</tbody>
</table>
1 Abstract

Multigrid methods have long been recognized for their efficiency as solvers of sparse linear systems of equations, mainly such that arise from discretizations of Partial Differential Equations (PDE). The classical algebraic multigrid (AMG), which was introduced in the early 1980’s, is still commonly used today for solving such systems. However, when it came to solving more general systems, some drawbacks were evident in the classical AMG. Consequently, a great effort was invested in extending the applicability of algebraic multigrid methods. A recent effort, which started in the mid 2000’s, is the development of adaptive versions of these methods, that aim at the solution of new and more complicated problems. The adaptive AMG framework extends the classical AMG framework, and relies on fewer assumptions about the underlying problem.

Our first study is a part of this effort, and focuses on adaptive aggregation-based multigrid approaches for the solution of Markov-Chains. This problem has drawn recent attention, largely due to its relevance in web search applications, among many others. It involves finding the principal eigenvector of column stochastic matrices. This task is rather problematic for AMG methods, because they require a good approximation for this vector to define the multigrid operators in their setup. From that point their efficiency in improving this approximation depends heavily on its initial accuracy. To overcome this conflict, adaptive AMG methods were commonly used to gradually improve the multigrid operators, with the iterated solution. We developed several promising new ideas for improving the so called “Petrov-Galerkin Smoothed Aggregation” AMG approach, which is suitable for both symmetric and non-symmetric problems. We introduced a novel Bottom-Up aggregation technique that has been shown to significantly improve the performance of the relatively uncommon method of smoothed aggregation, where only the prolongation is smoothed. Also, we developed a dynamic over-correction technique for accelerating multigrid cycles for non-symmetric linear systems. Our most significant development in this context involves a new “on-the-fly” adaptive framework for simultaneously applying the adaptive multigrid components (also in parallel). Our framework yields a highly effective self-adapting mechanism that can incorporate and accelerate any existing AMG method for the solution of Markov-Chains.

Our second study involves the rather challenging approach of non-Galerkin AMG methods. Traditional AMG methods use a hierarchy of smaller and smaller problems that represent the original problem on coarser and coarser scales. To define these problems, such
methods use (Petrov-)Galerkin coarsening, where the coarse problems are defined by recursively projecting the original operator with the so-called restriction and prolongation operators. Thus, these operators affect the sparsity pattern and operator complexity of the multigrid hierarchy, and therefore, AMG algorithms often compromise between the quality of these operators and the aggressiveness of the coarsening. In many scenarios, the multigrid coarse operators tend to be much denser than the fine operator as the coarsening progresses. This is evident in most AMG algorithms in the literature, including both the classical AMG and Smoothed Aggregation algorithms. Such behavior is especially problematic in parallel AMG computations where it imposes an expensive communication overhead. In this work we present a new algebraic technique for controlling the sparsity pattern of the operators in the multigrid hierarchy, independently of the choice of the restriction and prolongation. Our method is based on the aggregation multigrid framework, and it “sparsifies” Smoothed Aggregation operators while preserving their right and left near null-spaces. Our method is suitable for the solution of non-symmetric linear systems, and also retains the symmetry of the AMG operators for symmetric systems. Numerical experiments for convection-diffusion problems and diffusion problems with discontinuous coefficients demonstrate the efficiency and potential of this multigrid approach.

Our third study introduced the multilevel approach to the area of sparse approximation of signals, which is drawing tremendous attention in recent years. Typically, sparse solutions of underdetermined linear systems of equations are required. Such solutions are often achieved by minimizing an $l_1$ penalized least squares functional. Various iterative shrinkage algorithms have recently been developed and are quite effective for handling these problems, often surpassing traditional optimization techniques. In this work, we suggest a new multilevel framework that reduces the computational cost of existing solvers for these inverse problems. Our framework takes advantage of the typically sparse representation of the signal, and at each iteration it adaptively creates and processes a hierarchy of lower-dimensional problems employing one-level iterated shrinkage methods of a common generic type. Analytical observations suggest, and numerical results confirm, that this new approach may significantly enhance the performance of existing iterative shrinkage algorithms in cases where the matrix is given explicitly.
2 Definitions, notations and acronyms.

We start with definitions and notations that are used throughout the report. Matrices are denoted by capital letters: $A, B$. Vectors are denoted by bold letters: $\mathbf{x}, \mathbf{v}$. Scalars are denoted by either Greek letters $\lambda, \mu$ etc., or by non-capital and non-bold English letters $a, b$. Indices are usually denoted by the letters $i, j, k, l$ and pairs or triplets of indices are denoted by parentheses: $(i, j), (i, j, k)$. Below are also sections with notation that is specific for the different sections of this work. Each of these is also defined when it is used in the text itself.

General notation - applies for the whole document.

- $\rho(B)$ denotes the spectral radius of a matrix $B$. That is, $\rho(B) = \max_i \{|\lambda_i(B)|\}$ where $\lambda_i(B)$ are the eigenvalues of $B$.

- A real matrix $A \in \mathbb{R}^{n \times n}$ is called positive (semi-)definite if
  $$\forall \mathbf{v} \in \mathbb{R}^n : \mathbf{v}^T A \mathbf{v} > 0 \quad (\mathbf{v}^T A \mathbf{v} \geq 0).$$

  This definition also applies for a non-symmetric $A$.

- A real matrix $A \in \mathbb{R}^{n \times n}$ is an M-matrix if $A = \mu I - B$, where $B$ is a non-negative matrix and $\mu$ is a positive scalar which satisfies $\mu \geq \rho(B)$. If $\mu = \rho(B)$ then $A$ is singular, and if $\mu > \rho(B)$ then $A$ is non-singular and therefore positive definite.

- A graph $G(A) = (V, E)$ is a graph of a matrix $A \in \mathbb{R}^{n \times n}$ if $V = \{1, \ldots, n\}$ is the set of vertices and $E = \{(i, j) : a_{ij} \neq 0\}$ is the set of edges that correspond to the non-zero entries of the matrix $A$.

- $I$: the identity matrix.

- $1$: the vector of all ones.

- Vector norms: $\| \mathbf{x} \|_1, \| \mathbf{x} \|_2, \| \mathbf{x} \|_\infty$ are the 1, 2 and maximum norms respectfully.

- Frobenius norm: $\| A \|_F = \sqrt{\sum_{i,j} a_{ij}^2}$.

- Vector inner product: $\langle \mathbf{x}, \mathbf{v} \rangle = \mathbf{x}^T \mathbf{v}$. 
2. Definitions, notations and acronyms.

- **A-norm:** $\|x\|_A = \sqrt{\langle x, Ax \rangle} = \sqrt{x^T Av}$, for any positive definite matrix $A$.

- **AMG:** Algebraic Multigrid. In the literature, this phrase has two meanings. One is a general name for all algebraic multilevel methods, i.e., there are multiple such methods: Smoothed Aggregation AMG, Bootstrap AMG, Markov-Chains AMG etc. The other meaning is a specific name of an algorithm which is often also referred to as “classical” AMG, since it was the first of its kind.

- **AGG:** Pure Aggregation Multigrid.

- **SA:** Smoothed Aggregation Multigrid.

- **Relaxation:** a generic name for a simple and usually cheap iterative method, that is used within the multilevel structure.

- **CGC:** Coarse Grid Correction.

- **diag($A$):** a diagonal matrix whose entries are given by the diagonal of the matrix $A$.

- **$x^k$:** an approximated solution after the $k$-th iteration.

- **$e^k$:** the error after the $k$-th iteration.

- **$P \in \mathbb{R}^{n \times n_c}$:** prolongation (or, interpolation) operator. ($n_c < n$).

- **$R \in \mathbb{R}^{n_c \times n}$:** restriction operator.

- **$\theta$:** strength of connection parameter in multigrid algorithms.

- **Galerkin coarsening:** suitable mostly for symmetric problems $A_c = P^T A P$.

- **Petrov-Galerkin coarsening:** suitable mostly for non-symmetric problems $A_c = RAP$.

- **Subscript $c$:** coarse-grid components.
  
  - **$A_c$:** coarse-grid operator.
  - **$n_c$:** the size of the coarse-grid.
  - **$e_c$:** an error vector on the coarse-grid.
  - **$x_c$:** coarse-grid approximated solution (like $x^k$ on the fine-level).
2. Definitions, notations and acronyms.

- $\mathcal{N} = \{1, ..., n\}$ is the set of all indices of unknown variables.
- $\mathcal{C}/\mathcal{F}$: the set of coarse-grid variables ($\mathcal{C}$), and the rest of the fine-grid variables ($\mathcal{F} = \mathcal{N} \setminus \mathcal{C}$). This notation is used for algorithms based on $\mathcal{C}/\mathcal{F}$ splitting.
- $\{\mathcal{C}_j\}_{j=1}^{n_c}$: the splitting of $\mathcal{N}$ into disjoints sets—aggregates. Each of these will be represented by a variable on the coarse-grid.
- $\hat{P} \in \mathbb{R}^{n \times n_c}$: piece-wise constant binary prolongation matrix.
- $P_t, R_t$: Aggregation-based tentative transfer operators (non-smoothed).
- $C_{op}$: operator complexity. Defined as the total number of non-zero elements in all the approximations to $A$ on all levels, divided by the number of nonzero elements in the fine-level $A$ itself.
- $P_s, R_s$: Smoothed Aggregation transfer operators.
- $S$: strength of connection matrix for AMG coarsening.

Specific notation for section 5: AMG for Markov chains.

- $V_{SOL}$: classical (linear) solution V-cycle.
- $V_{EIS}$: non-linear Exact Interpolation Scheme (EIS) V-cycle.
- $\varepsilon_\alpha$: Accuracy threshold for adaptive multigrid algorithms.
- $q(x)$: quality measure of the solution for the On-The-Fly approach.
- $\gamma_k$ (In the context of On-The-Fly multigrid): a letter for indicating the convergence factor of a solution cycle after $k$ EIS cycles.
- $u_{set}, u_{sol}$, the cost of the setup and solution phase in work units, respectively. Each work unit in this section equals to the cost of a $V(2, 1)$ cycle.
- $t_{set}, t_{sol}$ the cost of the setup and solution phase in seconds, respectively.
- $\gamma$: a letter that represents a convergence factor. For a convergent process, $0 < \gamma < 1$. 
2. Definitions, notations and acronyms.

- **$R$**: The ratio between the computational costs of EIS and a solution cycle. An EIS cycle costs the same as $R$ solution cycles.

- **$R_p$**: The ratio between the computational costs of EIS and a solution cycle in parallel environment.

Specific notation for section 6: Non-Galerkin multigrid (SpSA).

- **SpSA**: Sparsified Smoothed Aggregation.

- **$A_g$**: the Galerkin or Petrov-Galerkin coarse operators, $A_g = RAP$. In this section we mostly refer to a Smoothed Aggregation operator.

- **$A_t$**: the Galerkin product of the tentative operators (pure aggregation operators). This operator is used to define the sparsity pattern of the sparsified SA operator.

- **$S_p(A)$**: the set of non-zero entries in the matrix $A$.

- **$\theta_{(i,j,k,l)}$**: the portion of an entry that will be corrected via the path $i \rightarrow j \rightarrow k \rightarrow l$.

- **$v$**: velocity vector in the convection-diffusion problem.

- **$\kappa$**: conductivity coefficient for the Poisson equation.

- **$\Theta$**: coefficients for the inhomogenous graph-Laplacian problems.

- **(P)CG**: (Preconditioned) Conjugate Gradients.

- **GMRES**: Generalized Minimal Residuals—a Krylov subspace acceleration method.

- **‘st’**: maximal stencil size in the whole multigrid hierarchy.

- **MAT-VEC**: matrix-vector multiplication.

- **$WU$**: work-units, each equals to a sparse matrix-vector multiplication with the finest level operator $A$.

- **$p$**: number of processors.

- **$T$**: an operator for measuring the Latency in communication complexity.
2. Definitions, notations and acronyms.

- Latency: the number of non-zero off-diagonal blocks in the matrix with the blocks defined according to the cluster partitioning.
- Bandwidth: the total number of values of $x^k$ that are needed to be sent between the computing nodes.

Specific notation for section 7: A multilevel approach for LASSO.

- LASSO (least absolute shrinkage and selection operator): An $l_1$ regularized least squares minimization problem. It is a non-smooth convex optimization problem.
- The $l_0$ quasi-norm: $||x||_0 = \# \{i : x_i \neq 0\}$.
- $\mu$: regularization parameter.
- $\partial f(x)$: the subdifferential of $f$. It is a set of sub-gradients.
- $\text{sign}(\cdot)$: the sign function.
- $\text{sup}(x)$: the support, the set of non-zero entries in $x$.
- $\mathcal{C}$: The set of coarse variables.
- $S_q(x)$: the shrinkage function. See (90).
- CG-PCD: non linear conjugate gradients acceleration applied on PCD (Parallel Coordinate Descent).
- $F(x), F_c(x_c)$: fine and coarse objective functions, respectively.
- likely($k$): is the set of indices of the $k$ largest elements in the likelihood vector $|A^T(Ax - y)|$.
- MLMI: memory-less monotonic iteration.
- $A(x)$: see (98).
2. Definitions, notations and acronyms.

- $s_0$: true solution in the synthetic experiments.

- $n$: noise vector which is added to the right hand side to produce the denoising experiment.

- ISNR: improvement in signal to noise ratio: this value should be as high as possible.

- ML-[1L Method]: a prefix for the multilevel approach that is applied using the “1L Method” as a relaxation.
3 Introduction

Sparse linear systems and eigenproblems occur in numerous applications in science and engineering. Development of hardware and software over the years has allowed scientists and engineers to larger and larger linear systems. Such huge sparse linear systems require solvers with a computational complexity that is linear in the number of unknowns, $n$. Traditional algorithms of super-linear complexity (such as LU factorization) are not considered applicable for such problems since the number of unknowns is too high. Other simple iterative methods such as *Jacobi*, *Gauss-Seidel* and *SOR*, may be highly inefficient in cases where the condition number of the matrices is high. Often, Krylov subspace acceleration is used to enhance such simple methods, and indeed it is effective in many cases [107]. However, also for those accelerations, the large condition number of the matrices constitutes a problem, and Krylov methods often tend to be inefficient in such cases. The matrices considered in this research typically have very large condition numbers. For a large $n$, these condition numbers might challenge Krylov subspace methods, and therefore they can only be used together with an efficient preconditioner. This is where multigrid methods are required. These methods can be used as stand-alone solvers, but for most cases, using them as preconditioners for Krylov methods may be the optimal way.

For quite some time, multigrid methods have been found to be highly efficient in solving various problems, thanks to their attractive linear complexity. At first, multigrid methods were applied to the solution of linear systems arising from discretization of elliptic PDEs on structured grids. Later, the “classical” algebraic multigrid (AMG) was developed [21, 22, 106] as an extension of (geometric) multigrid in order to deal with harder problems such as PDEs with discontinuous coefficients, PDEs on unstructured grids and many other applications. The classical AMG remains one of the most efficient methods for solving such linear systems, which often involve M-matrices\(^1\). At first, AMG was considered as a black-box solver for a large variety of linear systems, however, AMG failed to solve problems where strength of connections was not easily measured [31]. Still, the need for a general multigrid solver remained, and the search for it brought a variant of AMG called smoothed aggregation (SA) [132, 130, 131]. A natural generalization of smoothed aggregation is the so-called Petrov-Galerkin smoothed aggregation [62, 133] that was shown to be useful mainly for

\(^1\)For a definition of M-matrices, please refer to section 2.
solving convection-diffusion problems [63], and other nonsymmetric problems [108]. Another related variant that recently received attention is the accelerated pure-aggregation method [92, 96, 95, 98]. In addition, some other algorithms were targeted to specific applications such as finite elements solution of PDEs [28, 39, 71, 78].

Most of these multigrid methods feature operators that are built based on the assumption that the constant vector locally approximates the low-energy modes (also known as the near null-space) of the matrix, or at least it is assumed that they are known. If this assumption is not met, the efficiency of the methods may deteriorate. Already in [22, 105, 106], an adaptive AMG approach was suggested and self-improving AMG algorithms were outlined. The idea was to construct a prolongation operator whose range matches approximately the near null-space of the matrix. More recently, the adaptive Exact Interpolation Scheme (EIS) multigrid approach was suggested [23] for solving linear eigenproblems, whereby the solution itself is approximated on the coarser levels, rather than the error as in the classical multigrid. This approach requires the prolongation operator to be consistently improved as the iterations progress until, ultimately, an accurate solution is obtained by prolongating a smaller vector. This approach is also known as the multiplicative correction scheme. Following that, the adaptive approach was further developed, and adaptive AMG ($\alpha$AMG) [31] and adaptive smoothed aggregation ($\alpha$SA) [29, 30, 32] were suggested as more general and robust solvers for symmetric linear systems. In these algorithms, the multigrid components are built using a separate multilevel setup-phase similar to EIS. The linear system is solved using the same multigrid components in a separate solution-phase. The adaptive multigrid methods found use mostly for the applications of solid mechanics, quantum chromodynamics, and Markov-chains.

The adaptive framework is the main example for the effort to extend the applicability of multigrid and multilevel methods for solving new types of computational problems. That is a current focus of the multigrid research community, and the aim of our research in particular. The main approach addressed in our research is the aggregation based multigrid methods, but it is mostly used as a platform to demonstrate approaches that may be incorporated on other multigrid methods.

In our first study we aim at developing adaptive multigrid solvers for computing the principal eigenvector of column-stochastic matrices (Markov chains). This problem is used for modeling of random walks, queuing networks, and web search. It has drawn significant
interest in the early 2000’s, partly due to its relevance in Google’s PageRank algorithm. Since multigrid methods require the same sought vector to define their hierarchy of matrices, the adaptive multigrid framework was found to be suitable for the task, and the first work on this problem involved algorithms that are based only on the EIS approach. Based on the classical work of [76], a line of recent Markov-chain solvers was developed in [65, 66, 67, 68, 69, 122]. Appealing as these methods may be, like EIS they all suffer from one fundamental common drawback—they require a lot more computations because they calculate the whole multigrid hierarchy of operators in every cycle. Our contribution in this scope include several promising new ideas for improving the so called “Petrov-Galerkin Smoothed Aggregation” AMG approach. We introduced a novel Bottom-Up aggregation technique [122], a dynamic over-correction technique for accelerating multigrid cycles for non-symmetric linear systems [114], and a new “on-the-fly” adaptive framework for simultaneously applying the adaptive multigrid components (also in parallel) [123]. The later work resulted is a powerful self-adapting mechanism that can incorporate and accelerate any existing AMG method for the solution of Markov-chains.

In our second study we target the rather challenging approach of non-Galerkin AMG methods. Traditional algebraic multigrid (AMG) methods use a hierarchy of smaller and smaller problems that represent the original problem on coarser and coarser scales. To define these problems, such methods use Galerkin coarsening, where the coarse problem is defined by projecting the error equation onto the subspace defined by some prolongation and restriction operators. One drawback of the Galerkin coarsening is that the control over the sparsity of the operators in the hierarchy is quite limited. This might lead to an increased density of the Galerkin coarse-grid operators, and high overall computational complexity of the multigrid algorithm. Thus, one must often compromise between the quality of these operators, and the aggressiveness of coarsening, which affect the rate of convergence and the number of non-zeros in the whole hierarchy of matrices, reflecting its operator complexity. Classical Galerkin algorithms such as “classical” AMG or Smoothed Aggregation, which include high quality transfer operators, may lead to a severe stencil growth on coarse grids, even when the operator complexity of the hierarchy remains moderate. In the case of large-scale parallel computing, this usually leads to high communication overhead on coarse grids [115, 110] and loss of scalability, especially in 3D. Recently, there has been an effort to develop efficient multigrid algorithms that explicitly control the sparsity pattern of the multigrid hierarchy.
3. Introduction

[137, 124], or sparsify the Galerkin AMG operators [110]. These ideas have yet to reach their
full potential and were only applied for symmetric problems. We follow a similar frame-
work in this work, and propose a new method that is applicable to both non-symmetric and
symmetric problems. Our algorithm controls the sparsity pattern of the multigrid hierarchy,
using ideas related to [124, 110]. It is developed for non-symmetric problems but it preserves
the symmetry property of the coarse-grid operators for symmetric fine-grid operators. Our
algorithm uses the aggregation framework as a basis platform, by applying smoothed aggrega-
tion to define the prolongation and restriction, and pure aggregation Galerkin coarse-grid
operators for defining a sparse non-zero pattern of coarse operators.

In our third study we aim at introducing the multilevel techniques to the area of sparse
approximation of signals. This is an emerging area of research that is drawing vast interest
and finding use in numerous applications. One popular application is sparse representation
of signals and images, where the key underlying observation is that natural signals, such as
images, admit sparse decompositions over specific spatial transforms [35, 52]. Another pop-
ular application is known as compressive sensing [49, 37, 36] where signals are reconstructed
from only a few linear measurements. Other applications include statistical analysis, ma-
chine learning, and coding theory [128]. Our research introduces a straightforward multilevel
method for sparse modelling problems in the form of $l_1$ regularized least squares minimiza-
tion (a LASSO problem), based on the main concept of classical multigrid methods. That is,
we accelerate the convergence of typical iterative methods using smaller (coarser) versions
of the LASSO problem. This idea of introducing multilevel methods to sparse modelling has
yet to be explored and it has tremendous potential. In this work we follow the general ideas
of adaptive algebraic multigrid methods, which exploit a hierarchy of coarse approximate
operators that evolve with the solution process, eventually becoming exact.
4 General Background

4.1 Non-negative Matrices, Stochastic Matrices, M-matrices

This research thesis is mostly concerned with the solution of sparse linear systems with multigrid methods. We focus on algebraic multigrid methods and derive our algorithms using purely algebraic considerations. However, algebraic multigrid methods cannot be used to solve any linear system, and indeed the scope of matrices for which our algorithms may be applied is limited to certain types—mostly for M-matrices. Therefore, we provide some background on these matrices in this section, as well as some background on the closely related non-negative, and stochastic matrices. For more information about these topics, see [75, 112, 8, 117]. We restrict our discussion to real matrices in this work. We begin with some definitions:

**Definition 1 (Directed graph of a matrix).** A graph \( G(A) = (V, E) \) is a graph of a matrix \( A \in \mathbb{R}^{n \times n} \) if \( V = \{1, ..., n\} \) is the set of vertices and \( E = \{(i, j) : a_{ij} \neq 0\} \) is the set of edges that correspond to the non-zero entries of the matrix \( A \).

**Definition 2 (Non-negative matrix).** A matrix \( B \in \mathbb{R}^{n \times n} \) is called non-negative, and denoted \( B \geq 0 \), if all its entries are non-negative, i.e., \( b_{ij} \geq 0 \) for all pairs \((i, j)\).

We further say that \( B \geq C \) if \( B - C \geq 0 \). These relations may also be applied with strict inequalities, and also hold for vectors.

**Definition 3 (Reducible matrix).** A matrix \( B \in \mathbb{R}^{n \times n} \) is called reducible if there exists a permutation matrix \( P \) such that

\[
P^T A P = \begin{bmatrix} B_{11} & A_{12} \\ 0 & B_{22} \end{bmatrix},
\]

where \( B_{11} \in \mathbb{R}^{k \times k} \) for some \( 1 \leq k \leq n - 1 \). We say that \( B \) is irreducible if it is not reducible.

The property of irreducibility can also be considered as a graph property: A matrix \( B \) is irreducible if and only if, in its directed graph \( G(B) \), there exists a path from each vertex \( i \) to each vertex \( j \) (i.e., the graph \( G(B) \) is strongly connected). We note that any matrix whose elements are all nonzero is necessarily irreducible. The most important result concerning irreducible nonnegative matrices is the Perron-Frobenius theorem [75]:
Theorem 1 (Perron-Frobenius). Let \( B \in \mathbb{R}^{n \times n} \) be an irreducible nonnegative matrix, then the following holds:

1. There exists a positive real eigenvalue of \( B \) that is equal to the spectral radius \( \rho(B) \).

2. The eigenvalue \( \rho(B) \) is simple, i.e., both right and left eigenspaces associated with \( \rho(B) \) are one-dimensional.

3. \( B \) has left and right eigenvectors \( v_l \) and \( v_r \), both with eigenvalue \( \rho(B) \). The components of both \( v_l \) and \( v_r \) are all positive.

4. The only eigenvectors whose components are all positive are the eigenvectors associated with the eigenvalue \( \rho(B) \).

5. If \( B \) has exactly \( p \) eigenvalues equal in modulus to \( \rho(B) \), then these eigenvalues are distinct and equal to \( \lambda = \rho(B) \exp(2\pi ik/p) \), \( k = 0, ..., p-1 \).

In our study of Markov chains, we will consider a subclass of non-negative matrices, called column-stochastic matrices:

Definition 4 (Column-stochastic matrix). A non-negative matrix \( B \in \mathbb{R}^{n \times n} \) is called column-stochastic if for every column \( j \), \( \sum_{i=1}^{n} B_{ij} = 1 \).

By definition, because the columns of \( B \) sum to one, we have that \( \|B\|_1 = 1 \), where \( \|B\|_1 = \max_j \sum_{i=1}^{n} B_{ij} \) is the induced \( l_1 \) matrix norm. Since any matrix norm bounds the spectral radius from above, we have \( \rho(B) \leq 1 \). Furthermore, one of the left eigenvectors of a column-stochastic matrix \( B \) is \( 1 \):

\[ 1^T B = 1^T, \]

and we get that \( \rho(B) = 1 \). As a result, by the Perron–Frobenius theorem above, there exists a unique vector \( x \) (up to scaling), with strictly positive entries, which is the right eigenvector that corresponds to the eigenvalue 1, and satisfies

\[ Bx = x. \]

We consider such matrices is this thesis because, in particular, the transition matrices of finite state Markov chains are column-stochastic matrices. These encode all the information of the chain.


**M-matrices.** Throughout our work, we will mostly consider the solution of linear systems which involve M-matrices. There are many definitions to such matrices, and we follow the definition that is most natural to our work:

**Definition 5** (M-matrix). A real matrix \( A \in \mathbb{R}^{n \times n} \) is an **M-matrix** if \( A = \mu I - B \), where \( B \) is a non-negative matrix and \( \mu \) is a positive scalar which satisfies \( \mu \geq \rho(B) \).

If \( \mu = \rho(B) \) then \( A \) is singular, and if \( \mu > \rho(B) \) then \( A \) is non-singular and therefore positive definite. Let \( A \in \mathbb{R}^{n \times n} \) be a singular M-matrix. We state the following properties of singular M-matrices:

1. If \( A \) is irreducible, then \( A \) has strictly positive left and right eigenvectors \( \mathbf{v}_l \) and \( \mathbf{v}_r \), both with eigenvalue 0.

2. If \( A \) is irreducible, then every principal sub-matrix of \( A \) (other than itself), is a non-singular M-matrix.

3. If \( A \) is irreducible, then its diagonal is strictly positive.

4. The real part of each nonzero eigenvalue of \( A \) is positive.

5. If \( A \) is symmetric, then we can write for any vector \( \mathbf{e} \):

\[
\mathbf{e}^T A \mathbf{e} = \sum_{i<j} (-a_{ij})(e_i - e_j)^2.
\]

The last equality follows from

\[
\mathbf{e}^T A \mathbf{e} = \sum_{i,j} a_{ij}e_i e_j = \sum_{i \neq j} a_{ij}(e_i e_j - e_i^2) = \sum_{i \neq j} \frac{1}{2} a_{ij}(2e_i e_j - e_j^2 - e_i^2) = \sum_{i<j} (-a_{ij})(e_i - e_j)^2
\]

(1)

by setting \( a_{ii} = -\sum_{j \neq i} a_{ij} \) for the second row, and using the symmetry in \( \sum_{j \neq i} a_{ij} e_i^2 = \sum_{j \neq i} a_{ij} e_j^2 \) for the third row.
4.2 Algebraic Multigrid Methods

Algebraic Multigrid linear-system solvers generally follow a common basic idea. Given the linear system

\[ \mathbf{A} \mathbf{x} = \mathbf{b}, \]  

(2)

where \( \mathbf{A} \in \mathbb{R}^{n \times n} \) is positive semi definite, they use cheap iterative methods, such as Richardson, damped-Jacobi, Gauss-Seidel or SOR. Such methods, which are called relaxations, tend to converge slowly for most problems of interest. These methods are typically of the form:

\[ \mathbf{x}^{k+1} = \mathbf{x}^k - \omega \mathbf{Q}^{-1} (\mathbf{A} \mathbf{x}^k - \mathbf{b}), \]  

(3)

where \( \mathbf{x}^k \) is the approximated solution after the \( k \)-th iteration, \( \omega > 0 \) is a scalar parameter, and the matrix \( \mathbf{Q} \) is an easily-invertible simple preconditioner of \( \mathbf{A} \). For example, the damped-Jacobi relaxation uses \( \mathbf{Q} = \text{diag} (\mathbf{A}) \), i.e, the diagonal of the matrix \( \mathbf{A} \), while the Gauss Seidel and SOR methods use the lower triangular part of \( \mathbf{A} \). It can easily be shown that if we define the error after the \( k \)-th iteration as: \( \epsilon^k = \mathbf{x}^k - \mathbf{x} \), where \( \mathbf{x} \) is the solution to the system (2), then the error propagation of (3), is given by

\[ \epsilon^{k+1} = \epsilon^k - \omega \mathbf{Q}^{-1} \mathbf{A} \epsilon^k = (I - \omega \mathbf{Q}^{-1} \mathbf{A}) \epsilon^k, \]  

(4)

which is equivalent to applying (3) to solve the homogenous system

\[ \mathbf{A} \epsilon = 0. \]  

(5)

For a non-singular \( \mathbf{A} \), \( \epsilon = 0 \) is the solution to (5), so if the relaxation converges, then \( \mathbf{x}^k \rightarrow \mathbf{x} \).

The typically slow convergence of the relaxation is usually due to certain components of the error, called algebraically smooth eigenmodes. From (4), the error propagation operator is

\[ I - \omega \mathbf{Q}^{-1} \mathbf{A}, \]  

(6)

and the slowly converging (algebraically smooth) error modes are the eigenvectors that correspond to the eigenvalues with the largest magnitude of the matrix in (6). For a simple preconditioner \( \mathbf{Q} \), in some cases these errors approximately satisfy the homogeneous system
Therefore, the *algebraically smooth* components are also dubbed *near null-space* of $A$. To eliminate these components of the error, multigrid methods use a *coarse grid correction* (CGC), which is applied by constructing and solving a smaller (coarse) system. Thus, the CGC and the relaxation fulfill complementary roles. We denote the components of the coarse system by a $c$ subscript.

The coarse-grid correction aims at eliminating the algebraically smooth components, which are not eliminated effectively by the relaxation. To apply the CGC, AMG methods use a full rank prolongation matrix $P \in \mathbb{R}^{n \times n_c}$, where $n_c < n$. In order for the CGC to complement the relaxation, it is important that this prolongation well approximates the algebraically smooth error components, i.e., if $e^k$ is a smooth error, then $e^k \approx Pe_c$ for some vector $e_c \in \mathbb{R}^{n_c}$. Given $x^k$ and $P$, in the CGC we aim to find such a vector $e_c$ and correct $x^k$ by

$$x^k \leftarrow x^k + Pe_c.$$  \hspace{1cm} (7)

To achieve that, a suitable full rank restriction matrix $R \in \mathbb{R}^{n_c \times n}$ is defined, and the coarse-grid correction is done by solving a smaller-size ($n_c \times n_c$) coarse-grid problem

$$A_c e_c = R r^k = r_c,$$  \hspace{1cm} (8)

where $r^k$ is the residual $b - Ax^k$, $A_c$ is a coarse-grid operator, and $n_c$ is the number of coarse-grid variables. The solution $e_c$ of the coarser system (8) is then used in (7) to eliminate the algebraically smooth error.

In most AMG methods, problem (8) is defined by the Galerkin or Petrov-Galerkin coarsening

$$A_c = RAP \in \mathbb{R}^{n_c \times n_c}.$$  \hspace{1cm} (9)

With this choice, (8) becomes a projection of the error equation, $Ae = b - Ax^k$, onto the subspace defined by the full-rank prolongation and restriction operators. A typical two-grid algebraic multigrid cycle, which is a combination of relaxations and the CGC, is given in Algorithm 1.

Starting from an initial guess, such a cycle is repeated iteratively, until some convergence criterion is satisfied. A *multi-level* V-cycle is obtained by recursively treating the coarse grid problem in Step 4 of the two-level cycle. This V-cycle is the most common type of recursive
4. General Background

**Input:** Initial vector: $x^k \in \mathbb{R}^n$, Right-hand side vector: $b \in \mathbb{R}^n$;
Operators: $A \in \mathbb{R}^{n \times n}$, $P \in \mathbb{R}^{n \times n_c}$, $R \in \mathbb{R}^{n_c \times n}$, $A_c \in \mathbb{R}^{n_c \times n_c}$.
**Output:** Approximation: $x_{k+1} \in \mathbb{R}^n$

**Algorithm:**

1. Apply pre-relaxations: $x^k \leftarrow Relax(A, x^k, b)$.
2. Define the residual $r = b - Ax^k$.
3. Restrict the residual: $r_c = Rr$.
4. Define $e_c$ as the solution of the coarse grid problem $A_c e_c = r_c$.
5. Prolong $e_c$ and apply CGC: $x^{k+1} \leftarrow x^k + Pe_c$.
6. Apply post-relaxations: $x_{k+1} \leftarrow Relax(A, x_{k+1}, b)$.

**Algorithm 1:** Two-level standard cycle

Scheme in the multigrid literature. In some cases, when the V-cycle does not provide the sufficient performance, the recursive structure can be elaborated by applying two consecutive recursive calls to treat the problem in Step 4 of Algorithm 1—resulting in a W-cycle. Figure 1 illustrates the recursive structures of the V and W cycles. The prolongation and restriction, and also the coarse grid operator, can be defined in various ways, which are the main subject of investigation in the AMG-related research.

By subtracting the exact solution $x$ from both sides in Step 5, we obtain the CGC error

---

**Figure 1:** Multilevel V-cycle and W-cycle. ‘\(\downarrow\)’ refers restricting the residual, ‘\(\square\)’ refers to performing a coarse-grid solve, ‘\(\uparrow\)’ refers to prolonging the correction, and ‘\(\bullet\)’ refers to applying relaxations.
propagation operator:
\[ e^{k+1} = [I - P(A_c)^{-1}RA]e^k. \] (10)

Additionally, if (9) is used, and the error before the CGC happens to be in the range of the prolongation, that is, if there exists a vector \( w_c \) such that \( e^k = Pw_c \), then the CGC operator eliminates it:
\[ e^{k+1} = [I - P(A_c)^{-1}RA]Pw_c = [P - P(RAP)^{-1}RAP]w_c = [P - P]w_c = 0. \] (11)

The variational property of the Galerkin operator in the symmetric case:

If the matrix \( A \) in (2) is symmetric and positive definite, then the natural choice for \( A_c \) in (8)-(9) is \( R = P^T \), which corresponds to the Galerkin operator. Then, (8) satisfies a variational property—it is equivalent to the solution of
\[ \arg \min_{e_c \in \mathbb{R}^{n_c}} \|Pe_c - e^k\|_A, \] (12)

where \( \| \cdot \|_A = \sqrt{\langle \cdot, A \cdot \rangle} \) is the \( A \)-norm (also called the “energy norm”). In this case, if the relaxation is convergent in the \( A \)-norm, then so is the two-level cycle. In the non-symmetric case, a Petrov-Galerkin operator is usually chosen (\( R \neq P^T \)), and (8)-(9) impose that the new residual \( r^k - APe_c \) is orthogonal to the rows of \( R \). In this case, the two-level cycle is not guaranteed to converge.

4.2.1 Classical Algebraic Multigrid (AMG)

The classical AMG algorithm [22, 19, 106, 33] has been the prototype algorithm for many other AMG versions that followed it, and it is still widely used today. We follow the description of the introductory paper [55].

For simplicity, assume that we solve the system (2) for a symmetric positive semi-definite matrix \( A \), which is also an M-matrix\(^2\) with row-sums of zero (and hence, \( A1 = 0 \), where 1 is the constant vector of ones in all entries). Also assume that \( A \) is scaled such that its largest eigenvalue is 1. Following (11)-(12), in order for the CGC and relaxation to fulfill their complementary roles, we must choose the prolongation, \( P \), to have the algebraically smooth

\(^2\)For the definition of an M-matrix, please refer to Section 2.
components in its range. Let $\mathbf{e}$ be a normalized error vector (different from the constant vector). The (squared) energy norm of $\mathbf{e}$ is then following Equation (1) given by

$$\lambda(\mathbf{e}) = \mathbf{e}^T A \mathbf{e} = \sum_{i<j} (-a_{ij})(e_i - e_j)^2. \quad (13)$$

$\mathbf{e}$ is algebraically smooth if $\lambda(\mathbf{e}) \ll 1$. For $\lambda(\mathbf{e})$ to be small, we need $e_i \approx e_j$ if $(-a_{ij}) > 0$ is relatively large. This motivates one of the main heuristics of AMG: smooth error varies slowly in the direction of relatively large (negative) entries of $A$. The next concept that complements this heuristic characterizes what is a relatively large entry. We say that the variable $e_i$ strongly depends on the variable $e_j$ if

$$-a_{ij} > \theta \max_{k \neq i} \{-a_{ik}\}, \quad (14)$$

where $0 < \theta < 1$ is a strength threshold parameter. Generally, (14) is not a symmetric relation, that is, it may happen that $i$ strongly depends on $j$, but not vice-versa. There are alternative (symmetric) formulations and extensions to (14), and here we will simply say that if $a_{ij}$ or $a_{ji}$ satisfy (14), then $i$ and $j$ are “strongly connected”. Such strong connections define our previous notion of “relatively large entries”, and hence, all the above discussion is commonly summarized by the phrase: the near null-space of $A$ can be approximated by the constant vector between strongly connected entries $(e_i \approx e_j)$. This is the case, for example, in operators that are discretizations of the Laplacian operator, and, more generally, graph-Laplacian matrices. Figure 2 shows a typical algebraically smooth error, for a simple 2D problem. On the left side of the domain, where the PDE coefficients are isotropic, the error is geometrically smooth. However, on the right side of the domain, where the coefficients are anisotropic, the geometric notion of smoothness is evident only in the direction of the strong coefficients.

We continue with the definition of the classical AMG prolongation operator, $P$, and the rest of the coarse grid construction follows, using Galerkin coarsening, i.e., (9) with $R = P^T$. Assume that we partition the variable indices $\mathcal{N} = \{1, \ldots, n\}$ into a coarse-grid variable set $\mathcal{C}$ (which we refer to as $\mathcal{C}$-points) and a fine-grid variable set $\mathcal{F}$ (referred to as $\mathcal{F}$-points), where $\mathcal{N} = \mathcal{C} \cup \mathcal{F}$ and $\mathcal{C} \cap \mathcal{F} = \emptyset$. 


4. General Background

Figure 2: Algebraically smooth error for a simple 2D problem, discretized by finite elements, taken from [55].

Since $A$ is sparse, we define the neighborhood of a variable $i$,

$$\mathcal{N}_i = \{j \neq i : a_{ij} \neq 0\}.$$  \hspace{1cm} (15)

which can be split into strong neighborhood $\mathcal{N}_i^s$ and weak neighborhood $\mathcal{N}_i^w$

$$\mathcal{N}_i^s = \{j \neq i : i, j \text{ are strongly connected}\}, \quad \mathcal{N}_i^w = \mathcal{N}_i \setminus \mathcal{N}_i^s.$$  \hspace{1cm} (16)

Accordingly, we define $\mathcal{C}_i^s = \mathcal{N}_i^s \cap \mathcal{C}$ and $\mathcal{F}_i^s = \mathcal{N}_i^s \cap \mathcal{F}$.

Next, let the vector $e$ be an algebraically smooth error vector. That is, $Ae \approx 0$, hence, for each $i \in \mathcal{N}$ we have the approximation

$$a_{ii}e_i \approx -\sum_{j \in \mathcal{C}_i^s} a_{ij}e_j - \sum_{k \in \mathcal{F}_i^s} a_{ik}e_k - \sum_{\ell \in \mathcal{N}_i^w} a_{i\ell}e_\ell,$$  \hspace{1cm} (17)

which is the basic equation used to define the interpolation. The AMG concept is to assume that by solving the coarse-grid problem we have a good approximation to the values of the $\mathcal{C}$-points. Then we interpolate each value of a strongly connected $\mathcal{F}$-point by its neighboring $\mathcal{C}$-points. That is, for each $i \in \mathcal{F}$ we write

$$e_i \approx \sum_{j \in \mathcal{C}_i^s} w_{ij}e_j,$$  \hspace{1cm} (18)

where $w_{ij}$ are the interpolation coefficients that we need to choose. To define these weights,
we first approximate $e_\ell$ for $\ell \in \mathcal{N}_i^w$, in (17) by $e_i$ and move the resulting term to the left-hand-side:

$$
\left( a_{ii} + \sum_{\ell \in \mathcal{N}_i^w} a_{i\ell} \right) e_i \approx - \sum_{j \in \mathcal{C}_i^s} a_{ij} e_j - \sum_{k \in \mathcal{F}_i^s} a_{ik} e_k. \tag{19}
$$

On the one hand, we apply this because all the entries in $\mathcal{N}_i^w$ are weak connections, and replacing $e_\ell$ with $e_i$ does not significantly break the approximate equality in (17). On the other hand, we wish that the coarse-grid matrix will also have zero row sums like the fine-grid matrix. That is, we have a strict equality in (17) for $e = 1$, and we wish to maintain it in (19). Furthermore, for each $k \in \mathcal{F}_i^s$ we write

$$
e_k \approx \frac{\sum_{j \in \mathcal{C}_i^s} a_{kj} e_j}{\sum_{j \in \mathcal{C}_i^s} a_{kj}}, \tag{20}
$$

which basically assumes that the error at point $k$ is a weighted average of its neighboring errors, with weights according to the matrix entries. This is, in fact, where the smoothness heuristic come into play. (20) can also be viewed as a local energy minimization, because setting

$$
e_k = \arg \min_e \sum_{j \in \mathcal{C}_i^s} (-a_{kj})(e - e_j)^2
$$

leads to (20). Overall, by plugging (20) into (19) we get

$$
\left( a_{ii} + \sum_{\ell \in \mathcal{N}_i^w} a_{i\ell} \right) e_i \approx - \sum_{j \in \mathcal{C}_i^s} a_{ij} e_j - \sum_{k \in \mathcal{F}_i^s} a_{ik} \frac{\sum_{j \in \mathcal{C}_i^s} a_{kj} e_j}{\sum_{j \in \mathcal{C}_i^s} a_{kj}}. \tag{21}
$$

We conclude that the interpolation weights for each $i \in \mathcal{F}$ in (18) are given by

$$
w_{ij} = - \frac{a_{ij} + \sum_{k \in \mathcal{F}_i^s} \frac{a_{jk} a_{ik}}{\sum_{j \in \mathcal{C}_i^s} a_{kj}}}{a_{ii} + \sum_{\ell \in \mathcal{N}_i^w} a_{i\ell}}. \tag{22}
$$

As one can see, there might be a division by zero in (22) if the denominator of (20) is zero. This necessitates a constraint on the $\mathcal{C}/\mathcal{F}$ splitting, that there will be no strong $\mathcal{F} - \mathcal{F}$ connection between two entries without a common $\mathcal{C}$ point neighbor that is strongly connected to both of them. The classical $\mathcal{C}/\mathcal{F}$ splitting algorithm enforces that by applying
4. General Background

two passes [55, 33]:

1. Choose an initial $C$ as an independent set of fine-grid points based on the graph induced by the strong connections in $A$.

2. Add points to $C$ if needed, to satisfy the interpolation requirements. That is, if there is an $F - F$ connection between two entries without a common strong $C$-point neighbor, add one such common neighbor to $C$.

While the second pass enforces the constraint mentioned earlier, the first pass aims to choose the set of $C$ points as small as possible, reducing the number of coarse grid variables. The first pass is usually performed by a greedy algorithm, choosing the $C$-points one by one, preferring points with the highest number of strong neighbors. For more information about the $C/F$ splitting algorithm, see [55, 33]. Another widely-used alternative for this classical splitting algorithm is known as “compatible relaxation”, which can be found in [56, 86, 27]. We next describe a class of AMG variants that are based on “aggregation”—these are the main subject of our research.

4.2.2 Aggregation-Based AMG

Assume that we have a partitioning of the variables in $N = \{1, ..., n\}$ into disjoint subsets $\{C_J\}_{J=1}^{nc}$ called aggregates. Each of these aggregates will correspond to a variable (or a point) in the coarse-grid problem (8). Using these aggregates we first define tentative prolongation and restriction $P_t$ and $R_t$ which are simple aggregation/disaggregation operators. That is, both $P_t$ and $R_t^T$ have the same non-zero structure as the following piece-wise constant matrix $\hat{P}$:

$$
\hat{P}_{i,J} = \begin{cases} 
1 & \text{if } i \in C_J, \\
0 & \text{otherwise}
\end{cases} \quad i = 1, ..., n; \quad J = 1, ..., nc,
$$

(23)

which visually has the form as in the following:

$$
\hat{P} = \begin{pmatrix} 
1 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}.
$$

(24)
Note that $\hat{P}$, $P_t$, and $R_t^T$ must have only one non-zero value in each row.

As in the classical AMG prolongation, we aim that the algebraically smooth error modes are represented by the range of the prolongation. That is, the near null-space of $A$ is approximately in the range of $P_t$. For symmetric M-matrices arising from discretization of PDEs,

$$P_t = R_t^T = \hat{P}$$

is normally imposed, and under these definitions, the tentative prolongation corresponds to an interpolation of low order. For non-symmetric problems, we may also define $R_t$ such that $R_t^T$ has the the near null-space of $A^T$ approximately in its range [32].

The aggregates $\{C_j\}_{j=1}^n$ may be constructed in various ways, usually taking into account the strength of connections in the matrix $A$. For a symmetric M-Matrix $A$, a strong coupling between two members $i, j$, is usually implied by a relatively high absolute value of the entry $a_{ij}$ in the matrix (the off-diagonal entries are negative in M-Matrices), similarly to (14). Many aggregation heuristics were developed in [67, 65, 96, 97, 122, 123, 131], some of which are in the context of adaptive AMG, but can be easily applied in a standard AMG context as well. All of them are based on the strong connections heuristics described in the previous section.

**Classical Neighborhood-based Aggregation.** In this section we show the classical aggregation method of [131], which we use later in our work, with a slight modification. We note that the original description used a different strength of connection scheme (not based on (14)), however, the aggregation algorithm that we describe below is identical to [131].

To build the aggregates $\{C_j\}_{j=1}^n$ used in (23), we first define a strength of connection matrix $S$, and then apply a greedy neighborhood aggregation algorithm. Suppose that $S$ is a symmetric binary strength of connection matrix. That is, if $s_{ij} = s_{ji} = 1$, then $i$ and $j$ are strongly connected. $S$ can be defined based on (14) or by some other formulas, like that of [131], for example. Given such a matrix, the aggregation algorithm first traverses the unknowns $\{1, ..., n\}$, and joins each unknown $i$ with its neighbors into an aggregate, only if none of its neighbors were previously aggregated. If one of $i$'s neighbors in $S$ is already aggregated, then $i$ is skipped. Then, in a second pass, the algorithm assigns all the skipped unknowns into the existing aggregates (defined in the first pass) based on the maximal number of strong connections. Algorithm 2 summarizes this classical aggregation scheme.
We note that the aggregation provided by this algorithm may not be optimal, however, its simplicity and the low complexity of its computation are very appealing (from our experience, the cost of more sophisticated aggregation algorithms may sometimes dominate the cost of the entire solution of the linear system).

\begin{algorithm}
\textbf{Algorithm:} \{\mathcal{C}_i\}^n_{j=1} \leftarrow \text{Neighborhood-Aggregation}(S)
\begin{align*}
\% \ S & \text{ - Strength of connection matrix.} \\
\text{Let } S_i = \{i\} \cup \{j : s_{ij} \neq 0\} & \text{ denote the strong neighborhood of each point } i. \\
\text{Set } \mathcal{N} & \leftarrow \{1, \ldots, n\}, \text{ and } J \leftarrow 0 \\
\% \ First \ Pass: \ Assign \ neighborhoods \ to \ aggregates. \\
\text{foreach } i \in \mathcal{N} \text{ do} \\
& \quad \text{if each } j \in S_i \text{ does not belong to an aggregate then} \\
& \quad \quad \text{Set } J \leftarrow J + 1, \mathcal{C}_J \leftarrow S_i, \text{ and } \hat{\mathcal{C}}_J \leftarrow \mathcal{C}_J. \\
& \quad \text{end} \\
\% \ Second \ Pass: \ Assign \ remaining \ points \ to \ the \ aggregates. \\
\text{n_c} \leftarrow J \\
\text{foreach } i \in \mathcal{N} \text{ do} \\
& \quad J \leftarrow \arg \max_{K=1,\ldots,n_c} \{ |\hat{\mathcal{C}}_K \cap S_i| \} \\
& \quad \text{Set } \mathcal{C}_J \leftarrow \mathcal{C}_J \cup \{i\} \\
\text{end}
\end{align*}
\end{algorithm}

\textbf{Algorithm 2:} Classical Neighborhood-based Aggregation

4.2.3 Pure Aggregation AMG

In the pure (or non-smoothed) aggregation approach, the tentative operators \(P_t\) and \(R_t\) defined above are employed without further modification, that is, \(R = R_t\) and \(P = P_t\) are used in in Algorithm 1. This choice often yields slow convergence rates of the multigrid process, and the reason for this is the generation of high-energy modes by such low-order transfer operators. Figure 3 shows an example of such behavior for a discretization of the 1D Laplacian operator, whose stencil is \([-1, 2, -1]\). It shows a smooth mode of the associated matrix and its closest piecewise constant vector, where the aggregates are chosen as pairs of neighboring points. The interpolation error is shown to be highly oscillatory geometrically, and has a relatively high energy in the algebraic sense of Equation (13).
Figure 3: Interpolation error for the 1D discrete Laplacian operator: Upper drawing: approximation of a smooth mode $e$ by a piece-wise constant interpolation from a coarser grid vector $e_c$ ($e_c = \arg\min_{v_c \in \mathbb{R}^n_c} \|e - P_tv_c\|_2$). Lower drawing: the approximation error is shown to contain high-frequency modes.
The main advantage of the pure aggregation approach is its low operator complexity. The latter is defined as the total number of non-zero elements in all the approximations to $A$ on all levels, divided by the number of nonzero elements in the fine-level $A$ itself. Bounded operator complexity, independently on the problem size, is essential for the effectiveness of multigrid methods, since it emphasizes the computational cost of a V-cycle, which should not grow with the problem size. The pure aggregation method has one of the lowest complexities among algebraic multigrid methods.

To overcome the slow convergence of this method, Krylov subspace accelerations are often used [69, 92, 96, 95, 98] on all levels of the multigrid hierarchy, often requiring a more elaborate recursive structure such as a W-cycle, as in Figure 1. Recall that W-cycles are obtained by treating the coarse grid problem in Algorithm 1 by two consecutive recursive calls. Because these two calls are made on the same level and solve the same problem, they can be further accelerated by Krylov subspace methods (like GMRES [107] for example). This approach is known as K-cycle, and it yields a moderately efficient but often robust method.

### 4.2.4 Smoothed Aggregation AMG

To improve the tentative operators $P_t$ and $R_t$, classical and adaptive Smoothed Aggregation (SA) techniques were introduced for symmetric problems [29, 132, 130, 131], and Petrov-Galerkin SA [32, 62, 63, 108, 133, 65] for nonsymmetric problems. The main concept of smoothed aggregation is that the transfer operators are multiplied with a simple smoothing operator, similar but not identical to the relaxation operator in Equation (6). This is motivated by the fact that high frequencies that are generated by the low-order interpolation (as in Figure 3) are damped efficiently by the smoother. The prolongation is thus given by:

$$P_s = (I - \omega Q^{-1} A) P_t. \quad (26)$$

Here, unlike the relaxations, only a diagonal preconditioner $Q$ is suitable, because it is important to keep the number of non-zeros in $P$ small. Usually, the diagonal of $A$ is chosen, and then $I - \omega Q^{-1} A$ is the error propagation matrix associated with the damped Jacobi relaxation. Other options for $Q$ include the SPAI diagonal preconditioner [34], the EMIN preconditioner [108] or the $\ell_1$ diagonal preconditioner [2]. Non-diagonal preconditioners, like
the lower triangular matrix used in Gauss-Seidel, may over-fill the entries of the matrix $P$ with non-zeros, making the resulting AMG method inefficient.

For symmetric problems, the standard SA algorithm uses $R_s^T = P_s$. Mostly, when non-symmetric problems are considered, the Petrov-Galerkin approach is applied, whereby different transfer operators for prolonging and restricting are employed ($R \neq P^T$). A good choice for smoothing the restriction may be [65]:

$$R_s = R_t(I - \omega AQ^{-1}). \quad (27)$$

The usual choice for the damping parameter $\omega$ in (26) and (27) is [132]

$$\omega = \frac{4}{3\rho(Q^{-1}A)} \quad (28)$$

which is suitable mostly in the context of symmetric problems (see [132] for a complete analysis). For non-symmetric problems lower values of $\omega$ may be more suitable, and furthermore, different parameters may be used for the smoothers of the operators [108].

The usual way to apply smoothed aggregation is to use $P_s$ and $R_s$ in the (Petrov-)Galerkin coarse grid (9). However, the smoothing of the operators makes the coarse-grid operator much more dense, and the operator complexity might be unbounded (that is, as we coarsen the operator $A$ at each level contains more non-zeros). For that reason, the aggregation technique that is used in standard SA is much more aggressive than the one required for pure aggregation. That is, the aggregates are chosen to be bigger, and the coarse grid operators are consequently smaller (have less unknowns). However, in the framework of the Petrov-Galerkin SA, a compromise can be made by using smoothed prolongation only ($P = P_s$) and keeping the restriction of low order ($R = R_t$). This method has the advantage of using smaller aggregates as well as smoothed prolongation. This rather uncommon approach was used in [63] for the solution of convection-diffusion problems (arising from the compressible Navier-Stokes equations for example).

**Filtering of the Transfer Operators.** As noted before, the smoothing of the transfer operators in (26) may significantly enlarge the number of non-zeros in $P$ and $R$, and consequently also in $A_c$. This may generate denser and denser operators on the coarse levels of the hierarchy, and eventually may result in a less efficient algorithm. However, this can be
4. General Background

partially overcome by filtering the small entries out of the transfer operators, which do not contribute a lot to the quality of $P$ and $R$. This is crucial in problems that involve many small entries in the matrix, as, for example, in the discretized anisotropic diffusion problem in the right part of the domain in Figure 2. In such cases, the aggregation algorithm, which aggregates entries based on strong connections, does not “see” the weak connections, and hence does not aggregate based on those entries. On the other hand, such entries are included in the matrices multiplication for computing the coarse-grid operator (9), which results in a severe stencil growth, similarly to the stencil growth that happens when taking powers of $A$.

Motivated by these arguments, the authors of [131] suggested to replace the matrix $A$ in (26) with a filtered version $A^F$, which has non-zeros only in the entries that correspond to strong connections (where $S$ is non-zero in Algorithm 2). Similarly to the definition given in [131], $A^F$ may be defined by

$$ a^F_{ij} = \begin{cases} a_{ij} & s_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases} , \quad a^F_{ii} = a_{ii} + \sum_{j:s_{ij}=0} a_{ij} . $$

Note that by this definition, $A\mathbf{1} = A^F\mathbf{1}$, and this may be seen as the aggregation-based counterpart of the lumping in (19) discussed earlier in the $C/F$-based AMG.

4.3 The Adaptive Multigrid Framework

As described in the previous sections, the classical multigrid methods feature operators that are built based on the assumption that the constant vector locally approximates the low-energy modes (the near null-space) of the matrix, or at least it is assumed that they are known. If this assumption is not satisfied, the efficiency of the methods may deteriorate.

As opposed to the classical approach, the adaptive AMG approach aims at solving linear systems for which the algebraically smooth components cannot be easily characterized. By Equations (4)-(6), the error that is not eliminated well by the relaxation and needs to be treated by the CGC, will also remain after applying relaxations on the corresponding homogenous system, starting from a non-trivial initial guess. Therefore, the first adaptive AMG scheme of [22, 105], which was developed in the early stages of AMG history, constructed the interpolation to match vectors obtained by relaxation on the homogeneous problem. In [106],
the idea of fitting eigenvectors corresponding to the smallest eigenvalues was advocated, and an AMG algorithm determining these eigenvectors through Rayleigh quotient minimization was outlined.

A specific form of the adaptive approach, dubbed “Exact Interpolation Scheme” (EIS), was recently suggested in [23]. In EIS, the solution itself is approximated on the coarse grid, rather than the error as in the classical multigrid (as in Algorithm 1). This requires the prolongation operator to be consistently improved as the iterations progress until, ultimately, an accurate solution is obtained by prolongating a coarse approximation. This adaptive approach is also known as the multiplicative correction scheme. In this work we adopt the name EIS but acknowledge that the differences between the various algorithms of this type are very small. We elaborate on EIS in the next sections.

These ideas were further evolved and led to the development of adaptive AMG ($\alpha$AMG) [31] and adaptive SA ($\alpha$SA) [29] which were introduced as more general and robust solvers for symmetric linear systems. There, an approximation of the near null-space of $A$, and the multigrid coarse-grid operators are calculated by applying a multi-level setup cycle for solving the homogenous system (5). The linear system is then solved using the same multigrid components in a separate solution phase. $\alpha$SA was further developed and adapted to nonsymmetric problems [32].

We have adopted the adaptive algebraic framework for our work on AMG methods for the solution of Markov-chains, and we elaborate on the details of adaptive multigrid in the context of our work for solving Markov chains. This framework, however, can be used for solving other problems as well.
5. Aggregation-based Adaptive AMG for Markov Chains

In this section we describe our first study on adaptive algebraic multigrid methods for Markov chains. We provide a summary of the following two papers:


More discussions and results can be found in


This paper is not a part of this dissertation. It is a part of our co-author Dr. Killian Miller’s PhD thesis.

5.1 Introduction: Markov Chains

In this section we give a brief introduction on discrete time Markov chains (DTMC). A DTMC is a random process \( \{X_t, t \in T\} \), where \( t \) is the time parameter which varies over some discrete set \( T \), and the random state variable \( X_t \) varies over a finite set of states \( \{s_i\}_{i=1}^n \). This random process must obey the Markovian property that the conditional probability density function of future states of the process depends only on the present state:

\[
P(X_{t+1} | X_{t}, ..., X_0) = P(X_{t+1} | X_t).
\]

In other words, it is a memory-less process. We note that if the probability density function of the transitions does not depend on time then the Markov process is called homogeneous. If it does depend on time, then it is called non-homogenous. In this work, we focus on the homogenous case.
The conditional probability density function that corresponds to a homogenous DTMC is characterized by a transition matrix $B$ for which

$$b_{ij} = P(X_{t_{k+1}} = s_i | X_{t_k} = s_j) \quad k = 0, 1, ....$$

That is, $b_{ij}$ denotes the conditional probability to make a transition from state $j$ to state $i$ (and it does not depend on the time $t_k$). Also, for $B$ to represent a probability function it must obey

$$b_{ij} \geq 0, \sum_{i=1}^{n} B_{ij} = 1,$$

hence, $B$ is a column-stochastic matrix (see Definition 4). Figure 4 shows an example of a Markov chain: the transition graph and its corresponding column-stochastic matrix.

![Transition Graph and Column-Stochastic Matrix](image)

Figure 4: An example of a Markov chain with five states. The transition graph is on the left and the corresponding column-stochastic matrix is on the right.

We next consider the probability to be at each of the states $\{s_i\}$ at time $t_k$, and we denote this probability by $x_k^i > 0$. The vector of probabilities for all states is then given by $x^k$, and we have $\|x^k\|_1 = 1$. The probability distribution of all states of the chain at time step $k + 1$ is obtained by multiplying the current probability distribution at time step $k$ by the transition probability matrix:

$$x^{k+1} = Bx^k.$$  \hspace{1cm} (30)

It follows that $x^{k+1} = B^k x^0$, where $x^0$ is some initial probability vector. It is easy to show that if $B$ is a column-stochastic matrix, then so are the powers of $B$, and hence, $B^k$ can be seen as the transition matrix of $k$ steps.

An interesting probability distribution vector is the stationary probability vector—the
probability distribution of being at each of the states after an infinite number of steps:

\[ x = \lim_{k \to \infty} B^k x^0. \]  

(31)

This vector is the solution of the equation

\[ Bx = x, \quad x_i > 0, \quad \|x\|_1 = 1, \]  

(32)

and by the Perron–Frobenius theorem, if \( B \) is irreducible, then the vector \( x \) exists and is unique.

In this work, we aim at calculating the vector \( x \) satisfying (32), for a very large and sparse matrix \( B \). This problem has drawn recent attention, largely due to its relevance in web search applications [80, 79, 113, 61], and computational biology [91, 139, 7], among many others.

Following (31), we may apply a large number of steps (30) to some initial distribution. This would correspond to the well-known power method, and its asymptotic convergence factor is proportional to the ratio \( \frac{|\lambda_2|}{|\lambda_1|} \), where \( \lambda_1 \) is \( \rho(B) \) and \( \lambda_2 \) is the eigenvalue of \( B \) that is second largest in magnitude. Since \( \rho(B) = 1 \), the asymptotic convergence factor of (30) is \( |\lambda_2| \). However, Theorem 1 does not contradict situations where \( |\lambda_2| = 1 \), which result in a convergence factor of 1, implying that the method (30) may not converge. In this case, the matrix \( B \) is also said to be periodic. A simple solution to this can be achieved by the weighted power method

\[ x^{k+1} = (1 - \omega)x^k + \omega Bx^k, \]  

(33)

where \( 0 < \omega < 1 \) is some weighting parameter. In particular, by choosing \( \omega = 0.5 \), for example, we can overcome the problem in the periodic case and change the convergence factor to \( \max_i \{ \frac{1}{2}|1 + \lambda_i| \} \), where \( \lambda_i \) are the eigenvalues of \( B \). This value is typically close to \( \frac{1}{2} + \frac{1}{2} \text{Re}(\lambda_r) \), where \( \lambda_r \) is the eigenvalue with second largest real part. This factor is also guaranteed to be smaller than 1 (for a detailed analysis of this result, please see Appendix A.1). Still, this value can be very close to 1, and hence also the weighted power method may require many steps to converge. In our research, we aim at solving (32) effectively by algebraic multigrid methods.
5.2 Adaptive AMG for Markov Chains: The Exact Interpolation Scheme

Let us first summarize the definition of the Markov-chain problem (32) and reformulate it as a solution of a homogenous linear system. Let \( B \in \mathbb{R}^{n \times n} \) be an irreducible sparse column-stochastic matrix, that is, for every column \( j \), \( \sum_{i=1}^{n} B_{ij} = 1 \), and all the elements of \( B \) are non-negative. By the Perron–Frobenius theorem [75], there exists a unique vector \( x \) with strictly positive entries that satisfies \( Bx = x \) and \( \|x\|_1 = 1 \). Furthermore, \( \rho(B) = 1 \), where \( \rho \) denotes the spectral radius. This problem is often formulated as finding the null-vector of the singular M-matrix

\[
A = I - B. \tag{34}
\]

Our objective is to compute the principal eigenvector of \( B \), i.e., the unique vector \( x \) that satisfies \( Bx = x \), so we seek the solution of

\[
Ax = 0, \quad \|x\|_1 = 1. \tag{35}
\]

Since \( B \) is column-stochastic, \( 1^T B = 1^T \), where 1 is a constant vector of size \( n \). By (34), we have \( 1^T A = 0 \), which means that in this Markov Chain problem the constant vector is the (only) left null-vector of the matrix \( A \).

As a relaxation for this problem, we may use any of the methods that were described earlier, together with a right hand side of 0—see the discussion after Equation (3). The algebraically smooth error modes in this context are comprised of eigenvectors that correspond to eigenvalues that are small in magnitude, but different from 0.

5.2.1 The General Exact Interpolation Scheme

Suppose that we could construct some prolongation operator \( P \in \mathbb{R}^{n \times n_c} \), such that the solution \( x \) of (35) were in its range, that is, \( x = Px_c \) for some vector \( x_c \) of smaller size. Defining a suitable restriction operator \( R \in \mathbb{R}^{n_c \times n} \), substituting \( Px_c \) for \( x \) in equation (35), and multiplying through by \( R \), we obtain,

\[
RAPx_c = A_c x_c = 0, \quad \|x_c\| = 1. \tag{36}
\]
where \( A_c \in \mathbb{R}^{n_c \times n_c} \) is the coarse-grid matrix. After solving (36), we obtain the sought solution by prolongation: \( x = P x_c \). This motivates the EIS approach, where the prolongation \( P \) is constructed such that the current approximation to the solution is approximately in its range. As the solution becomes more and more accurate, so does the coarse representation of the original problem. The EIS cycle for homogeneous systems is described in Algorithm 3. As before, a multigrid V-cycle is obtained by treating the coarse-grid problem in step 5 recursively.

### Algorithm: Two-level EIS cycle

**Input:** Initial vector: \( x \in \mathbb{R}^n \), fine-grid operator: \( A \in \mathbb{R}^{n \times n} \).

**Output:** Better approximation to the solution of \( A x = 0, \| x \| = 1 \).

Coarse-grid and transfer operators: \( P \in \mathbb{R}^{n \times n_c}, R \in \mathbb{R}^{n_c \times n}, A_c \in \mathbb{R}^{n_c \times n_c} \).

**Algorithm:**

1. Apply pre-relaxations: \( x \leftarrow Relax(A, x) \).
2. Construct the interpolation operator \( P \) with \( x \) approximately in its range.
3. Construct a restriction operator \( R \).
4. Calculate the coarse-grid operator: \( A_c = R A P \).
5. Define \( x_c \) as the solution to the coarse-grid problem: \( A_c x_c = 0, \| x_c \|_1 = 1 \).
6. Prolong solution: \( x \leftarrow P x_c \).
7. Apply post-relaxations: \( x \leftarrow Relax(A, x) \).

When \( A \) is nonsymmetric, the restriction needs to be chosen such that the near null-space of \( A^T \) is in the range of \( R^T \) [32]. In Markov chains, the left null-space is known to be the constant vector, and so we define \( R \) with the constant vector in the range of \( R^T \), i.e., \( 1^T_c R = 1 \). By \( 1^T A = 0 \) and (36), the constant vector is then a left null-vector of \( A_c \).

Problem (35) is essentially the same as the homogenous problem (5) discussed earlier. As noted before, the coarse grid operators are designed to capture the subspace of the algebraically smooth errors, which approximately satisfy (5) and (35). Hence, the same techniques that will be described here for solving (35) may also apply as a setup for adaptive
approaches for solving non-homogeneous linear systems, or for solving eigenproblems, i.e., finding the few eigenvectors and eigenvalues that correspond to the eigenvalues which are smallest in magnitude.

5.2.2 Aggregation-based Exact Interpolation Scheme

In this section we describe the adaptive counterpart of Section 4.2.2. Assume we have a partitioning of the variables in $\mathcal{N} = \{1, ..., n\}$ into disjoint subsets $\{C_J\}_{J=1}^{n_c}$ called aggregates. Using these aggregates we first define tentative prolongation and restriction $P_t$ and $R_t$ that are simple aggregation/disaggregation operators. The construction of such aggregation was discussed in Section 4.2.2 and will be further discussed later in Section 5.2.3. Also, let $x$ be a positive approximate solution to (35).

Given the aggregates $\{C_J\}_{J=1}^{n_c}$ and the approximation $x$, we define a tentative aggregation-based prolongation matrix $P_t$ with $x$ in its range. As noted earlier, the left null space of the matrix $A$ is the constant vector. Therefore, we choose all restriction matrices to be piece-wise constant, so that the matrices on all the levels have a constant left null-vector. Specifically, given $x$ and the aggregates $\{C_J\}_{J=1}^{n_c}$, we use the matrices originally suggested in [76]

$$
(R_t)_{i,j} = \begin{cases} 
1 & \text{if } i \in C_J \\
0 & \text{otherwise}
\end{cases} \\
(P_t)_{i,j} = \begin{cases} 
\frac{x_i}{(x_c)_J} & \text{if } i \in C_J \\
0 & \text{otherwise}
\end{cases},
$$

with $(x_c)_J = (R_t x)_J = \frac{1}{|C_J|} \sum_{r \in C_J} x_r$, where $|C_J|$ denotes the number of nodes in the aggregate $C_J$. Note that there is a single non-zero value in each row of $P_t$ and $R_t^T$, and their sparsity structure is identical. Note that the approximate solution $x$ is in the range of the prolongation, that is $x = P_t x_c$ for $x_c = R_t x$, as required by the EIS framework.

Note in (37) that the disaggregation operator $P_t$ distributes each aggregate’s value amongst the fine-grid elements belonging to the aggregate, with relative weights proportional to the corresponding elements in the current approximation $x^k$. The main heuristic observation regarding the smoothing effect of the relaxation (damped Jacobi, for example) is that, for a sparse matrix $A$, the ratio between “strongly connected neighbors” in the current approximation $x^k$ quickly tends to the corresponding ratio in the exact solution $x$. That is, if $-a_{i,j}$ is relatively large in comparison to other off-diagonal terms in the $i$th row of $A$ ($i$ and $j$ are strongly connected) then, after several relaxations, the ratio between the $i$th and $j$th
elements of the current approximation $x^k$ tends to be close to the corresponding ratio in the exact solution $x$. We strive to create aggregates comprised of strongly connected elements. Thus, by this smoothing effect, the relaxation and coarse-grid correction fulfill complementary roles: relaxation causes the values within each aggregate to tend to the correct relations, and coarse-grid correction corrects the value of each aggregate, i.e., their sum.

The coarse grid matrix is defined by (36). One of the advantages of using the operators (37) is that the coarse operator remains an M-matrix. Furthermore, if $A$ is irreducible, then so is $A_c$. This means that we can continue recursively and obtain a well-defined multi-level process.

### 5.2.3 Aggregation—a bottom-up approach

The aggregation procedure should aim to approximately optimize the tradeoff between convergence rate and operator complexity. The latter is defined as the total number of non-zero elements in all operators, $A$, on all levels, divided by the number of nonzero elements in the fine-level operator. In general, bigger aggregates (aggressive coarsening) lead to slower convergence, but also to smaller operator complexity. However, experiments show, as expected, that the convergence rates are determined by the size of the biggest aggregates, even if there are only a few such big ones. Therefore, it makes sense to strive to select homogeneously-sized aggregates, and this motivates our approach.

The bottom-up aggregation approach we next describe is an alternative to the (more typical) top-down approaches of Algorithm 2 from [131] and the approaches in [67, 65]. All of these are motivated by the same notion of strength of connection. We refer as “popular” to variables with many strongly connected neighbors regardless of the directions of the dependencies. In contrast to common practice, our approach is to first make sure that the “less popular” variables are aggregated properly, and only later tend to the “more popular” ones. The reason for this is that “popular” variables will naturally be selected to aggregates anyway, whereas “unpopular” ones will not, unless we explicitly require it. We therefore do this early on, to avoid the situation where many unattached variables need to be added to existing aggregates at the end of the process. These same arguments motivated the work of [95, 96] where different bottom-up aggregation algorithms were suggested and in which avoiding singletons was also considered as a priority. In those algorithms, pairs of the most strongly connected elements are aggregated, and when this approach is not aggressive
enough, it is repeated, collecting pairs of pairs as aggregates, etc.

We define a weight matrix $S$ based on the elements of the matrix $A$ and the current approximation $\mathbf{x}$ to its near null-space. Specifically, we define a connectivity matrix $\hat{S}$ with

$$
\hat{S}_{ij} = \begin{cases} 
-A_{ij}x_j & \text{if } i \neq j \text{ and } -A_{ij}x_j \geq \theta \max_{l \neq i} -A_{il}x_l, \\
0 & \text{otherwise},
\end{cases}
$$

(38)

where we use $\theta = 0.1$. Then, for reasons that are described later, we symmetrize our connectivity matrix as follows:

$$
S = \frac{1}{2} \left( \hat{S} + \hat{S}^T \right).
$$

(39)

Since $S$ is symmetric, each pair of neighbors is a cycle of length 2. Note that a binary version of this matrix is used in the coarsening procedure of [67, 65]. Also, (38) is closely related to the classical strength of connection (14), and reduces to it if $\mathbf{x} = \mathbf{1}$.

Our algorithm aims to find the most strongly connected aggregates of size approximately equal to a given integer parameter $s$ (which is a guideline to the algorithm regarding the aggregate sizes). To this end, it searches for groups of about $s$ points which are closest to forming a clique, and at least have a circular dependency of all the members of the aggregate. We define a circle of length $s$ as a set of $s$ members $\{i_1, \ldots, i_s\}$ where $S_{i_j, i_{j+1}} > 0$ for all $1 \leq j \leq s - 1$ and $S_{i_s, i_1} > 0$. Note by (39) that if $S_{ij} > 0$ then $i$ and $j$ form a cycle of size two, so such cycles are ubiquitous. We also define the weight of an aggregate as the sum of the elements of the connectivity submatrix corresponding to its members:

$$
w(\mathcal{C}_j) = \sum_{i,j \in \mathcal{C}_j} S_{ij}.
$$

(40)

Our second aim is to limit the operator complexity. The tradeoff of low complexity versus better convergence rate is controlled by the parameter $s$. Small $s$ means higher complexity and, in general, better convergence rates per cycle, and vice versa. Our approach in this work is to simply choose for each problem the smallest value of $s$ that keeps the operator complexity sufficiently low. While we found this approach to be effective in our experiments, it carries with it an additional computational cost. The relative advantage of our approach is its tendency to form compact aggregates (and, in particular, avoid “chains”).

Our Bottom-up($s$) algorithm appears in Algorithm 4. It is of linear complexity, and it may
### 5. Aggregation-based Adaptive AMG for Markov Chains

**Input:** $s$—typical aggregate size, $S \in \mathbb{R}^{n \times n}$—strong connections matrix  
**Output:** Aggregates: $\{C_j\}_{j=1}^n$.

**Algorithm:**

Let $\mathcal{U} = \{1, \ldots, n\}$ be a set of unassigned elements.

repeat

Among the elements in $\mathcal{U}$, find the element $i$ with the least remaining neighbors in $\mathcal{U}$.

Initialize a new aggregate $\mathcal{C} = \{i\}$.

if $i$ has more than one neighbor then

Define $\mathcal{U}_{\text{near}} \subseteq \mathcal{U}$ to contain all elements whose distance from $i$ is at most $\lfloor \frac{s}{2} \rfloor$.

Find all circles within $\mathcal{U}_{\text{near}}$ of length $s$ or less that contain $i$.

% At least one circle will always be found.

Choose the circles of maximal length, and amongst those choose the circle of maximal sum of inner connections in $S$.

Add the members of the chosen circle to $\mathcal{C}$.

else

Let $p$ be the only neighbor of $i$ in $\mathcal{U}$.

Add $p$ to $\mathcal{C}$.

if other neighbors of $p$ with only one neighbor besides $p$ exist then

Add up to $(s - 2)$ of them to $\mathcal{C}$.

end

end

Remove the chosen aggregate’s members from $\mathcal{U}$.

if After removing the chosen aggregate there are elements with no neighbors then

Add the elements with no neighbors to $\mathcal{C}$.

end

until all elements are assigned to aggregates

**Algorithm 4:** Bottom-up aggregation.
be implemented in various ways. We use a Depth-First-Search technique (starting from $i$) in order to recognize circles of length $s$, and $s$ buckets in order to sort the members according to the number of neighbors. This procedure might be relatively expensive, depending on the parameter $s$ and the mesh structure. After this, we may, following a suggestion in [67, 65], calculate aggregates only once throughout the whole iterative process and then freeze them. In order to do that, we first need to obtain a locally smooth approximation to $x$ in order to define the matrix $S$, and that can be achieved by applying a few relaxations to a random guess.

One nice property that this Bottom-Up algorithm has, is that it produces structured aggregation for structured problems. For the Laplacian operator on structured grids, it yields aggregates of width 2 in each spatial direction in many scenarios we have studied, in 1D, 2D and 3D. Investigating and improving this aggregation method remains the subject of ongoing research. Figure (5) shows two examples of aggregations with $s = 4$, where in the case of the structured graph we get the optimal $2 \times 2$ aggregation, and in the case of the unstructured graph we find that most of the aggregates are indeed of size 4.

### 5.2.4 Smoothing the tentative operators

As described earlier in Section 4.2.4, in order to improve the tentative operators $P_t$ and $R_t$ we may multiply them by a simple smoothing operator, resulting in the smoothed operators in (26) and (27). The difference here is that, for Markov chains, we will smooth the adap-
5. Aggregation-based Adaptive AMG for Markov Chains

tive tentative operators (37) and not the piecewise constant operators as in (23). This SA approach was adopted in [65].

In this work, we focus on the case where the prolongation is smoothed as in (26), but the restriction remains unsmoothed—\(R_t\). SA with no \(R\) smoothing is generally considered inferior to smoothing both \(R\) and \(P\) [108]. However, the comparisons that led to this conclusion employed the same aggregates for both methods. When smoothing both \(R\) and \(P\), the eventual coarse-grid operator in (36) has the form of a cubic polynomial in \(A\), coarsened by the tentative operators. Unless the coarsening is aggressive enough, a significant stencil growth might occur on coarser grids. On the other hand, smoothing only \(P\) results in a coarsened quadratic polynomial in \(A\), which allows less aggressive coarsening. For a discretization of a \(d\)-dimensional second-order PDE on a rectangular grid, for example, smoothing both operators requires aggregates of width 3 in each spatial direction (aggregates of size \(3^d\)), while smoothing only \(P\) requires aggregates of width 2 [63]. The use of smaller aggregates is the key to the potential advantage of smoothing only \(P\), since less aggressive coarsening (more coarse-grid DOFs) generally leads to a better coarse representation of the fine-grid problem, and therefore to a better CGC.

The classical neighborhood aggregation of [131] is appropriate in the case where both \(R\) and \(P\) are smoothed, but it is too aggressive when only \(P\) is smoothed [108]. Therefore, in this work we developed an aggregation algorithm that, among other properties, forms smaller aggregates—see Section 5.2.3. We note that the smoothing of \(P_t\) should be applied with a filtered version of \(A\) as in (29), only this time we require that \(A\mathbf{x} = A^F\mathbf{x}\), where \(\mathbf{x}\) is the approximate null-space vector. Assume that we have a strength of connection matrix \(S\) (as in (39)), the (adaptively) filtered matrix is given by:

\[
a^F_{ij} = \begin{cases} a_{ij} & s_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases}, \quad a^F_{ii} = a_{ii} + \frac{1}{x_i} \sum_{j:s_{ij}=0} a_{ij}x_j. \quad (41)
\]

To smooth \(P_t\), we use a damped Jacobi operator. In this case, the damping parameter \(\omega\) for the prolongation smoother in (26) is usually chosen to be [62, 63, 108, 133]

\[
\omega = \frac{1}{\rho(Q^{-1}A^F)}, \quad (42)
\]

since this is the only choice that guarantees that the coarse-grid matrix remain positive.
definite. The spectral radius is usually approximated using an Arnoldi-type method, and this works well for symmetric matrices. Since the eigenvectors of the matrix are orthogonal in the symmetric case, the Rayleigh Quotient,

\[ \frac{v^T A v}{v^T v} \]

is guaranteed to be bounded by the spectral radius. However, finding the spectral radius of a nonsymmetric matrix may be costly, because the simple Rayleigh Quotient may provide a value far larger than the spectral radius. Nevertheless, we do estimate the spectral radii, but only once in the first setup cycle and with low accuracy. Specifically, in this work we apply 25 power-method iterations and use the Rayleigh Quotient (43) as an approximation to the spectral radius. This calculation is relatively expensive, and alternatives are worthy of investigation. Simpler options such as using constant smoothing parameters or using matrix norms (upper bound to the radii) were found to be significantly less effective than using the true spectral radius.

5.2.5 Analysis of Aggregation vs. Smoothed Aggregation

In this section we provide some motivation for why low-order transfer operators such as simple aggregation lead to slow convergence, and show why smoothing the transfer operators improves the coarse-grid process.

Denote the eigenpairs of \( A \) by \( \{ \lambda_k, v_k \}_{k=1}^n \) where\(^3\) \( 0 = \lambda_1 < \lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_n \), with \( v_1 = x \). The coarse-grid problem, \( A_c x_c = 0 \) can be written as a minimization problem,

\[ x_c = \arg \min_{||x'||=1} ||A_c x'|| = \arg \min_{||x'||=1} || R_t A P_t x'||, \]

and since the coarse operator is an M-matrix with column-sums of zero, this minimization functional has a unique solution with a value of 0. Suppose next that we can write the prolongated coarse-grid vector as a linear combination of the eigenvectors of \( A \):

\[ P_t x_c' = \sum_{k=1}^n \alpha_k v_k. \]

\(^3\)Recall that we only consider real-valued eigenvalues in this discussion. For an analysis of the complex spectrum case, see Appendix A.2.
Then the minimization problem becomes

\[
\arg \min_{\{\alpha_k\} \in \mathcal{A}_{P_t}} \left\| \sum_{k=2}^{n} \alpha_k \lambda_k R_t v_k \right\| \quad (46)
\]

where

\[
\mathcal{A}_{P_t} := \{\{\alpha_k\} : \sum_k \alpha_k v_k \in \text{Range}(P_t) \text{ and } \left\| \sum_k \alpha_k R_t v_k \right\| = 1\}
\]

Recall that the purpose of the coarse-grid correction is to reduce efficiently smooth eigenvectors, \(v_k\), with \(\lambda_k \approx 0\), while the relaxation reduces the rough eigenvectors, with negative or small positive eigenvalues. It is well-known that a low-order prolongation, such as the pure aggregation operators, tends to generate parasitic high frequencies—see Figure 3 for an example. In addition, the low-order restriction fails to damp strongly high-frequencies, but instead transfers them to the coarse-grid, where they alias with low frequencies. (This motivates the approach of Smoothed Aggregation, where the parasitic high frequencies are smoothed.)

Assume that the coarse-grid problem is solved exactly, and therefore the objective of the equivalent minimization problem (46) is reduced to zero. We now focus on minimizing each term in the argument of (46) separately where each term is either damped by the restriction (\(\|R_t v_k\|\) is small), or attenuated through the nature of the range of the prolongation (\(|\alpha_k|\) is small), or else reduced by the factor \(|\lambda_k|\), which is small only for smooth modes. Consider a term in the argument of (46), corresponding to a high-frequency \(v_k\). Since \(v_k\) is rough, then \(|\lambda_k|\) is not small, and assuming that \(R\) does not strongly damp \(v_k\), the only way that this term can be small is if \(|\alpha_k|\) itself is small. But then (45) appears to contradict the fact that parasitic high frequencies are generated by \(P_t\), which would imply a significantly large \(|\alpha_k|\). To resolve this contradiction, \(x'_c\) must remain close to \(R_t x^k\) (which, by the definition of \(P_t\) in (37), is the unique vector that is prolonged exactly to \(x^k\) without generation of parasitic modes). In other words, the minimization is distracted and focuses on attenuating the rough modes rather than the smooth ones as required of the coarse-grid. This leads to slow convergence. This can be overcome by using a high-order prolongation which reduces the generation of parasitic high frequencies, and a high-order restriction that strongly damps them. This is precisely what smoothed aggregation achieves.

To summarize, the slow convergence brought about by low-order transfers is due to
the generation and relative amplification of rough eigenmodes by $P_t$, and lack of sufficient damping of such modes by $R_t$. Therefore, $|\alpha_k|$ corresponding to rough eigenmodes have to be allowed to grow somewhat in order for the smooth modes to be reduced by the coarse-grid correction. Since these rough modes can easily be removed by post-relaxations, their generation does not cause undue deterioration of convergence.

Now we explain how the smoothing of $P$ helps us to overcome this. Suppose that we use a Richardson smoother for the prolongation. That is, we use $Q = I$ in (26), and (42) as damping parameter:

$$P_s = (I - \frac{1}{\rho(A)}A)P_t. \quad (47)$$

Also, we keep the restriction as $R_t$. The resulting coarse grid operator would be a coarsening of $A - \frac{1}{\rho(A)}A^2$. Clearly, this matrix retains the same null-space vector as $A$, but the “roughest” eigenmodes of $A$ (with eigenvalues close to $\rho(A)$) are no longer very significant in equation (44) with $P_s$ instead of $P_t$—their eigenvalues are now close to 0 and their corresponding $|\alpha_k|$ is allowed to grow. This leads to much better two-level convergence rates, even though we continue to use the simple restriction operator.
5.3 On-the-fly Adaptive Algebraic Multigrid for Markov Chains

As mentioned earlier, the adaptive AMG approach aims at solving linear systems for which the algebraically smooth components cannot be easily characterized. For this reason, already in [22, 105, 106] an adaptive AMG approach was suggested and self-improving AMG algorithms were outlined. The idea was to construct a prolongation operator aimed at matching the near null-space of the matrix. These ideas led to the development of Adaptive AMG (αAMG) [31] and adaptive SA (αSA) [29, 32] which were introduced as more general and robust solvers. There, an approximation of the near null-space of $A$, and the multigrid coarse-grid operators are calculated by applying a multi-level setup cycle for solving the homogenous system (5). This setup cycle is essentially the EIS procedure described in the previous section (Algorithm 3). Once the hierarchy is set, the linear system is then solved using the same multigrid components in a separate solution phase using iterations of Algorithm 1.

As described in the previous section, the EIS approach has been used as iterative solvers for Markov chains. The first multilevel solver of [76] is an example of an EIS algorithm, even though it was developed independently. Based on this approach, a collection of Markov Chain solvers was suggested in [65, 66, 67, 68, 69, 122]. Appealing as these methods may be, they all suffer from one fundamental common drawback—they calculate the whole multigrid hierarchy of operators in every cycle, with a computational cost that is at best comparable to the cost of the classical AMG setup phase. Experience has shown that, in single processor environments, the setup costs of AMG (or an exact interpolation V-cycle) are typically comparable to those of several classical V-cycles [42, 118]. In [122] it is noted that, empirically, more than 50% of the computation time of each V-cycle is spent on coarse matrix construction. In parallel (distributed memory) environments, the setup phase may require significant communication between nodes and therefore a larger fraction of the total computing time [74].

The approach presented in this section can be applied with various adaptive multilevel Markov chain solvers. Here, we apply it to the Smoothed Aggregation method where only the prolongation is smoothed as in (26), but the restriction remains unsmoothed (see section 5.2.4). In [114] it is also applied to the AMG-type method of [66] and the pure aggregation method of [76]. The present contribution addresses the aforementioned drawback of adaptive Markov chain solvers—the use of the multiplicative correction approach, which requires us
to calculate the multigrid hierarchy in every cycle. We follow the adaptive framework of [29, 31, 32] but incorporate a new scheme that interleaves the classical and EIS algorithms. Like other adaptive approaches, the new scheme uses a solution phase and updates the hierarchy of operators in a setup phase. The new feature is that the update of the hierarchy is done on-the-fly, interlaced with the solution process. We also suggest a parallel variant of this approach. An alternative adaptive approach for Markov chains is presented in [16], where standard solution cycles are also used followed by a more sophisticated setup process.

5.3.1 The relation between the classical and EIS algorithms

Suppose that we solve the same homogenous problem (35) using the two approaches described above—the classical cycle in Algorithm 1, and the EIS cycle in Algorithm 3. It is interesting to study the relation between the two algorithms. We next show that a two-level version of the EIS approach is actually equivalent to the classical approach, supplemented by adaptive updates of the prolongation and restriction. In the classical approach of Algorithm 1, the coarse-grid problem for \( b = 0 \) is given by

\[
RAP e_c = R r = -RA x, \tag{48}
\]

and the CGC is \( x \leftarrow x + Pe_c \). Suppose that the approximation \( x \) before the CGC is in the range of \( P \), so there exists a vector \( x_c \) that satisfies \( x = Px_c \). Adding the term \( RAP x_c \) to both sides of (48), we get

\[
RAP(x_c + e_c) = RA(Px_c - x) = 0. \tag{49}
\]

Now, by plugging the vector \( w_c = x_c + e_c \) in the left-hand-side of (49), we get \( A_c w_c = 0 \), which is exactly the same coarse-grid problem as in (36). Moreover, by applying the multiplicative CGC (step 6 of Algorithm 3), we get

\[
x \leftarrow Pw_c = P(x_c + e_c) = x + Pe_c,
\]

which is exactly the same correction as in the classical approach. Thus, the classical two-level approach is equivalent to EIS, provided that we use the \( P \) and \( R \) of EIS for both algorithms. This analysis can be readily extended to the multi-level case, provided that \( P \) and \( R \) are
adapted at every level.

The EIS cycle is clearly much more expensive than the classical cycle with ‘frozen’ operators, because EIS updates $P$ and $A_c$ before each CGC, whereas they remain constant in the classical cycle. The equivalence seen here motivates our approach of interleaving EIS cycles and classical cycles.

**Exact solution on the coarsest grid.** The solution on the coarsest grid in the EIS cycles is obtained by calculating the null-vector by an eigensolver or by the shifted inverse power method. The situation in classical cycles, however, requires some care. The matrix $A_c$ is singular on the coarsest grid (see remarks after Equation (35)). When we solve (48), the solution $e_c$ may contain an arbitrary proportion of the null-space of $A_c$, which $P$ may translate to an approximate equivalent null space of the matrix in the upper level. Such a “correction” may be counter-productive, chiselling away at the very null-vector we are trying to expose. Hence, in order to achieve a “null-vector free” approximation on the coarsest grid, we use the singular value decomposition (SVD) of the coarse-grid matrix,

$$A_c = U \Sigma V^T,$$

where $U$ and $V$ are orthogonal and $\Sigma$ is a diagonal non-negative matrix that has the (sorted) singular values of $A_c$ on its diagonal. We define a diagonal matrix $\Sigma^+$ whose diagonal is given by

$$\Sigma^+_{ii} = \begin{cases} \Sigma^{-1}_{ii} & \Sigma_{ii} \geq \epsilon \\ 0 & \Sigma_{ii} < \epsilon \end{cases} \quad (51),$$

where $\epsilon$ is small and positive (we use $10^{-14} \cdot \|A_c\|$). Note that $\Sigma_{nn} = 0$ because $A_c$ is singular. The coarsest-grid solution is then $x_c = V \Sigma^+ U^T r_c$, and is a “null-vector free” solution of (48). This does not quite guarantee a null-vector free correction $P e_c$, but the proportion of the null-vector in the correction is relatively small; see also related discussion in [135].

### 5.3.2 Optimizing the use of EIS and classical cycles

Generally, a more expensive setup is expected to provide a faster solution process; however, it is hard to optimize the overall cost of the setup and solution processes together. In addition, the effectiveness of the solution cycles depends on accuracy of the prototype vectors that are
used to define the transfer operators. Hence, there might be problems where a fixed setup may not suffice for an effective solution cycle. In [29, 31], the initial setup cycle is followed by a rather expensive self-testing procedure. This involves applying several solution cycles to a random guess and monitoring the convergence rate of the solver. If it is too poor, then more setup cycles are performed to improve the multigrid operators. Setup cycles with more pre and post relaxations and with standard cycles as relaxations were also considered. In that work, however, the algorithm aims at solving multiple systems of the same matrix with different right hand sides. Therefore, this effort is clearly worthy, because a single setup is used for solving many linear systems. Also, [29] aims at developing solvers for systems with a rich near null-space so more effort is invested in the setup. Here, we aim only to find the unique null-vector of the matrix, so this procedure might be overly expensive in our case. We therefore exploit the fact that the EIS and classical algorithms target the same homogenous problem, so they can enhance each other.

We next describe three approaches for combining setup and solution cycles. All approaches have four main steps and differ in only one or two of those steps. These approaches will be examined and compared later.

A simple adaptive AMG scheme. The first approach is to initially perform several setup cycles until the operators reach a certain quality and then continue and apply classical solution cycles. We denote this approach by AFTER($\varepsilon_\alpha$), and we assume that the quality of the operators is mostly controlled by the smoothness of the prototype $\mathbf{x}$ which is used for defining the prolongation $P$. More specifically, we perform several relaxation sweeps on an initial guess, followed by an initial setup cycle. Then we perform EIS cycles until the residual norm of the approximation drops below $\varepsilon_\alpha$. Finally, we perform one additional EIS cycle so that the resulting operators will be based on the approximate solution satisfying this criterion. For the rest of the process we use standard solution cycles. The algorithm is described in Algorithm 5, where $V_{SOL}(\cdot)$ denotes a solution cycle (Algorithm 1 applied with $\mathbf{b} = 0$), and $V_{EIS}(\cdot)$ denotes a setup cycle which also creates/updates the MG operators (Algorithm 3). This notation is used for the rest of this section.

A rule of thumb for $\varepsilon_\alpha$ is that it should be smaller than the magnitude of the smallest non-zero eigenvalue in absolute value. For the test cases and solution method considered in this work, $\varepsilon_\alpha = 10^{-5}$ is sufficient.
Input: Threshold $\varepsilon_\alpha$, fine-grid operator: $A \in \mathbb{R}^{n \times n}$, initial guess $x_0$.
Output: Approximate solution to: $Ax = 0, ||x|| = 1$
Algorithm: Perform setup followed by solution cycles

1. Initial Setup:
   - Apply a few relaxations to smooth $x_0$.
   - Do an initial EIS cycle: $x \leftarrow V_{EIS}(x_0)$.
   - \% $x_0$ after pre-relaxations is in the range of $P$.
   - if $||Ax_0||_1 < \varepsilon_\alpha$ goto Step 4.

2. Improve solution approximation:
   - while $||Ax||_1 > \varepsilon_\alpha$ do $x \leftarrow V_{EIS}(x)$.

3. Finalize setup:
   - \% Final $P$ based on pre-relaxed $x$.
   - $x \leftarrow V_{EIS}(x)$.

4. Solution:
   - Apply $x \leftarrow V_{SOL}(x)$ until convergence.

Algorithm 5: AFTER—a simple adaptive AMG scheme

On-the-fly adaptive algebraic multigrid. In the “on-the-fly” approach we first apply a single initial setup cycle and start the solution process. Additional setup cycles may be applied as the solution phase progresses. The underlying assumption is that solution cycles are considerably cheaper, but they require the operators supplied by the EIS setup. Again, we assume that better approximations supplied to the EIS cycle yield better operators in return.

The goal of the on-the-fly algorithm is to reach the accuracy $||Ax||_1 < \varepsilon_\alpha$ as fast as possible, preferring solution cycles over EIS cycles, because they are so much cheaper. We try to save computations by using the operators from previous setup cycles. Even though these operators may not be very accurate, they may suffice for obtaining useful solution cycles. For example, assuming that an EIS cycle costs the same as five solution cycles and achieves a convergence factor of about 0.4 (which is quite reasonable for difficult problems), a solution cycle that achieves an uninspiring convergence factor of about 0.75 may be more efficient, because $0.75^5 < 0.4$. With this motivation, we introduce the following procedure where $q(\cdot)$ is a quality measure of the approximation, and $0 < \gamma \leq 1$ is a scalar threshold.
for an acceptable convergence factor of the solution cycles.

**Procedure try-SOL-else-EIS(γ):**

1. $y = V_{SOL}(x)$.
2. if $q(y) > q(x)$ do $x \leftarrow V_{EIS}(x)$ and finish.
3. if $q(y) < \gamma q(x)$ then $x = y$, else $x \leftarrow V_{EIS}(y)$.

The choice of $q(x)$ is not obvious. If $A$ is symmetric semi-definite then the Rayleigh quotient (43) is a useful measure. However, for nonsymmetric problems, which are the focus of this work, this measure is inappropriate and we use instead

$$q(x) = \frac{\|Ax\|_1}{\|x\|_1},$$  \hspace{1cm} (52)

which means that we measure the convergence factor by the $l_1$ residual norm. The residual, which emphasizes high-energy modes, may not generally be an optimal measure of performance. However, we find it quite reliable in practice when used, as here, for choosing between two rather similar multigrid processes. With this procedure, the (non-overlapping) on-the-fly algorithm proceeds as in Algorithm 6.

**Overlapping on-the-fly adaptive algebraic multigrid.** The last strategy we consider is mostly suitable for distributed-memory parallel computations. Publications on adaptive multigrid have not focused on parallel computation. However, since classical and adaptive AMG have so many common ingredients, the parallelization of adaptive AMG should probably be very similar to that of classical AMG. With this in mind, we immediately note a potential drawback. In serial code, the setup cycle, which is similar to EIS, typically costs the same as several (less than 10) solution cycles [42, 118, 74, 54]. In parallel, however, the setup requires much shorter and more frequent communication between computing nodes. Coarsening procedures, matrix multiplication, construction of $P$ and $R$, and other ingredients of the setup cycle, are typically much more complicated than the simple matrix-vector multiplications of the solution cycle. Numerical results in [141, 115] show a ratio of about 15-45, and in many cases the setup cost exceeds the total solution cost. In addition, the algorithms are shown to be scalable with the number of processors only when the problem
5. Aggregation-based Adaptive AMG for Markov Chains

**Input:** Threshold $\varepsilon_\alpha$, Convergence parameter $\gamma$, operator: $A \in \mathbb{R}^{n \times n}$, initial guess $x_0$.

**Output:** Approximate solution to: $Ax = 0$, $||x|| = 1$

**Algorithm:** Solution by non-overlapping setup and solution cycles

1. **Initial Setup:**
   - Apply a few relaxations to smooth $x_0$.
   - Do an initial EIS cycle: $x \leftarrow V_{EIS}(x_0)$.
   - \% $x_0$ after pre-relaxations is in the range of $P$.
   - if $||Ax_0||_1 < \varepsilon_\alpha$ goto Step 4.

2. **Improve Solution Approximation:**
   - while $||Ax||_1 > \varepsilon_\alpha$ do try-SOL-else-EIS($\gamma$).

3. **Finalize Setup:**
   - \% Final $P$ based on pre-relaxed $x$.
   - $x \leftarrow V_{EIS}(x)$.

4. **Solution:**
   - Apply $x \leftarrow V_{SOL}(x)$ until convergence.

**Algorithm 6:** On-the-fly adaptive AMG

(size is growing as well (weak scaling). That is, each processor handles a fixed amount of grid points, and as the problem grows more processors are used. If we increase the number of processors for a fixed problem size, and refine the grid-point partitioning between processors, then we impose more communication between processors and hence a higher cost. This means that for a given problem size there is an optimal amount of processors, and exceeding it will harm the performance. Also, in [115], one clearly sees that the ratio between the setup cost and iteration cost grows with the number of processors (or, equivalently, with the problem size). As a result, in parallel adaptive AMG computations we should limit the amount of setup cycles performed even more than in serial computations.

The above discussion highlights even more the advantage of Algorithm 6 over Algorithm 5 in parallel computations. However, the overlapping on-the-fly algorithm that we propose next aims at improving Algorithm 6 even further by using double computing resources. The new algorithm uses high-level parallelism between two computing clusters that does not require frequent communication between them. One cluster applies solution cycles while the other performs a setup update. It is important to note that if doubling the resources for
an internally parallelized Algorithm 6 significantly improves its runtime, then it might be better than the overlapping approach we propose (ideally, the runtime would drop by a factor of 2). However, if adding more computing resources does not improve the performance of Algorithm 6 for a given problem size, then the overlapping version is worthy of consideration.

The overlapping on-the-fly adaptive algorithm is exhibited in a block diagram in Figure 6. The first EIS cycle is an essential setup phase. Once an initial hierarchy of operators is set, the solution process is good to go. As in the non-overlapping version in Algorithm 6, we apply solution cycles so long as they are efficient (marked by the ‘try-sol’ block), until either their performance deteriorates or the condition $||Ax||_1 < \varepsilon_\alpha$ is satisfied. Then, the hierarchy update evolves via EIS in its slow pace while the solution cycles are applied in parallel. Once a new hierarchy is computed, a cross-over of information may occur, whereby the solution process provides the setup process with a more accurate approximation, while the setup process provides the solution process with a new hierarchy of operators. At each cross-over, the two computing units switch roles of applying solution and setup phases, so as not to require transferring of the operators (a large amount of data) between the computing nodes. Since the two processes join only at cross-over points, this scheme does not require recurrent and frequent communication between the setup and solution clusters.

The function $\text{better}$ that appears in the cross-over in Figure 6 is optional. Because there
5. Aggregation-based Adaptive AMG for Markov Chains

**Input:** Threshold $\varepsilon_\alpha$, Convergence parameter $\gamma$, operator: $A \in \mathbb{R}^{n \times n}$.

**Output:** Approximate solution to: $Ax = 0, ||x|| = 1$

**Algorithm:** Solve by overlapping setup and solution cycles

1. **Initial Setup:**
   - Apply a few relaxations to smooth $x_0$.
   - Do an initial EIS cycle: $x \leftarrow V_{EIS}(x_0)$.
   - if $||Ax_0||_1 < \varepsilon_\alpha$ goto Step 4.

2. **Improve Solution Approximation:**
   - while $||Ax||_1 > \varepsilon_\alpha$ do
     - $y \leftarrow V_{SOL}(x)$.
     - if $q(y) < \gamma q(x)$ then $x \leftarrow y$
     - else $x \leftarrow$ overlapping-setup(better($x, y$)).

3. **Finalize Setup:**
   - $x \leftarrow$ overlapping-setup($x$)

4. **Solution:**
   - Apply solution cycles until convergence.

---

**Algorithm 7:** Overlapping on-the-fly adaptive AMG
is no guarantee that a single setup cycle yields a convergent solution process, we may use this safety procedure that aims to limit unnecessary computations. In practice, one may use

\[ \text{better}(x_1, x_2) = x_1 \text{ if } q(x_1) < q(x_2); \text{ else } x_2, \]

where \( q(x) \) is the quality measure mentioned earlier in Equations (43)-(52). We note that at least in the experiments considered in this work, the vector \( x' \) from the setup cycle is almost never preferred by the function ‘better’.

The overlapping on-the-fly adaptive MG algorithm appears in Algorithm 7, where the following procedure is used for overlapping solution and setup cycles:

Procedure overlapping-setup(\(x\)):

- Do setup update on cluster A:
  \[ x' \leftarrow V_{EIS}(x). \]
- Do solution cycles on cluster B:
  \[ \text{repeat } x \leftarrow V_{SOL}(x) \text{ until cluster A finishes or iterations diverge.} \]
- \( x \leftarrow \text{better}(x, x') \)

5.3.3 A qualitative analysis

To provide some insight into the properties of the different strategies, we resort to a simplified qualitative analysis. Given current multigrid operators and an approximation \( x \), we consider three scenarios:

1. Applying solution cycles with the existing operators until convergence;
2. Applying an EIS cycle and then solution cycles until convergence;
3. Applying \( r \) solution cycles, followed by an EIS cycle with the new approximation, and then solution cycles until convergence;

We examine the total time of solution for a fixed problem size in the three scenarios. We measure the approximate time for convergence in work units, each equal to the cost of one solution cycle. The convergence criterion is an error reduction by a factor \( \epsilon \ll 1 \). For this
discussion we use the notation given in Table 1. We expect, and indeed typically observe, $0 < \gamma_E < \gamma_2 < \gamma_1 < \gamma_0 \leq 1$.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma_0$</td>
<td>The average convergence factor when applying solution cycles with the existing operators, as in scenario 1.</td>
</tr>
<tr>
<td>$\gamma_E$</td>
<td>The convergence factor of an EIS cycle. For simplicity, we assume that it does not depend on the input $x$.</td>
</tr>
<tr>
<td>$R$</td>
<td>The ratio between the computational costs of EIS and a solution cycle. An EIS cycle costs the same as $R$ solution cycles.</td>
</tr>
<tr>
<td>$\gamma_1$</td>
<td>The average convergence factor of the solution cycles using operators after an additional EIS cycle, as in scenario 2.</td>
</tr>
<tr>
<td>$\gamma_2$</td>
<td>The average conv. factor of solution cycles using operators produced by an EIS cycle that followed $r$ solution cycles, as in scenario 3.</td>
</tr>
</tbody>
</table>

Table 1: Qualitative analysis notation

**Serial environment.** Table 2 summarizes the costs for the total solution processes, where $n$ denotes the number of solution cycles until convergence after the operators are finalized. To select the preferred strategy we need to know all the quantities in the equations, and, since those are problem and solver dependent, it is hard to choose the best option in advance. For example, in the unlikely case that EIS does not improve the performance of the solver significantly, i.e., $\gamma_0 \approx \gamma_1 \approx \gamma_2$, then the simple first strategy is best. On the other hand, if the EIS cycle is equally effective regardless of the stage in which we carry it out, i.e., $\gamma_0 >> \gamma_1 \approx \gamma_2$, then, unless $R$ is extremely large, the second strategy will be best. In this case, the third strategy is second-best because the initial $r$ cycles of factor $\gamma_0$ are relatively less effective.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Derivation</th>
<th>Approximate Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\gamma_0^n = \epsilon$</td>
<td>$\frac{\log \epsilon}{\log \gamma_0}$</td>
</tr>
<tr>
<td>2</td>
<td>$\gamma_E \gamma_1^n = \epsilon$</td>
<td>$R + \frac{\log (\epsilon/\gamma_E)}{\log \gamma_1}$</td>
</tr>
<tr>
<td>3</td>
<td>$\gamma_0 \gamma_E \gamma_2^n = \epsilon$</td>
<td>$r + R + \frac{\log (\epsilon/\gamma_0 \gamma_E)}{\log \gamma_2}$</td>
</tr>
</tbody>
</table>

Table 2: Qualitative analysis: solution costs
In this work we focus on hard problems, that require high-quality MG operators. In such cases we typically find $\gamma_2 \ll \gamma_1 \ll \gamma_0$ or $\gamma_2 \ll \gamma_1 \approx \gamma_0$. Then, the third strategy, which corresponds to the on-the-fly approach, will have the best asymptotic convergence factor, and will likely have the best performance.

**Parallel environment.** In scenarios 2 and 3, we also consider the case where solution cycles are applied in parallel to the setup cycle. As mentioned before, we double the resources by engaging a second computing cluster but assume that doubling the resources in one cluster does not lead to significant speedup. Table 3 summarizes the solution process costs of both scenarios using overlapping setup and solution cycles.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Derivation</th>
<th>Approximate Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$\min{\gamma_E, \gamma_0^R} \gamma_1^n = \epsilon$</td>
<td>$R + \frac{\log(\epsilon/\min{\gamma_E, \gamma_0^R})}{\log \gamma_1}$</td>
</tr>
<tr>
<td>3</td>
<td>$\gamma_0^r \min{\gamma_E, \gamma_0^R} \gamma_2^n = \epsilon$</td>
<td>$r + R + \frac{\log(\epsilon/\gamma_0^r \min{\gamma_E, \gamma_0^R})}{\log \gamma_2}$</td>
</tr>
</tbody>
</table>

Table 3: Qualitative analysis: solution costs in parallel environment using overlapping setup

From this analysis one can see that the overlapping scheme best suits situations where the EIS cycles are very expensive compared to the solution cycles, i.e., $R$ is rather big as is indeed expected in parallel-AMG computations. If $\gamma_0$ is not very close to 1, then the solution cycles run in parallel to the EIS cycle are still effective. Then, the overlapping approach gains from the $R$ solution cycles, and in addition we obtain good asymptotic behavior. This approach is also good for $\gamma_0 \approx \gamma_1 \approx \gamma_2$, where the first strategy is best, though it requires more computing units. In the non-overlapping approaches (Table 2), an excessively large $R$ might render the second and third scenarios ineffective due to the large cost of EIS. However, if $R$ is moderately high, and $\gamma_0$ is not sufficiently small, the second and third strategies come out ahead. In any case, the overlapping strategy is optimal in both situations, and therefore it is very useful if $R$ is big and the additional hardware cost is acceptable.

### 5.4 Numerical Results

In this section, we demonstrate and compare between the following four algorithms: one is the EIS approach as a solver, and the other three are strategies for integrating solution
and setup cycles, discussed in Section 5.3.2. The algorithms are applied to adaptive Petrov-Galerkin SA, where only the prolongation is smoothed: see Section 5.2.4, and in addition refer to Section A.3 for some simple procedures that we apply to accelerate the convergence of this method. For constructing the aggregation, we use the bottom up approach in Algorithm 4. This, together with the results for the EIS approach demonstrate the contribution of our work in [122], while the rest of the algorithms demonstrate the contribution in [123].

We show results for three representative Markov chain problems that were found to be difficult enough to require more than one setup cycle. For the other benchmarks that exist in the literature, we generally get better results.

In all the results shown, we start with an initial random guess and reduce its $l_1$ residual norm $||Ax||_1$ by a factor of $10^{10}$. We employ V(2,1) solution cycles with Jacobi relaxations ($Q$ is the diagonal part of the matrix $A$ in (3)). Similarly to [29, 31], when we perform setup cycles followed by solution cycles in the AFTER and OnTheFly algorithms, we do more pre-relaxations in the EIS cycles than in the solution cycles: 4 pre-relaxations and 1 post-relaxation. We do that in all setup cycles of OTF and in the last EIS cycle of AFTER (Step 3). All other EIS cycles are V(2,1). Note, that a $V_{EIS}(4,1)$ cycle is not significantly more costly than $V_{EIS}(2,1)$, because most of the computation is spent on operator construction and matrix multiplication (RAP calculations) [29, 31, 122]. Exact solution is performed once the problem size is below 16. The Jacobi relaxations are damped with $\omega = \frac{4}{3\rho(D^{-1}A)}$. For smoothing the prolongation we use the Jacobi operator with damping parameter as in (42).

In the following tables and figures we show computation times as well as work units. The latter is defined as the cost of a single $V_{SOL}(2,1)$ solution cycle. The experiments were performed using MATLAB R2010b on a machine with an Intel core i7 CPU with 8 GB of RAM memory, running Linux Ubuntu.

For the first approximation to the null-space on which we base the initial operators, we do 20 relaxation sweeps on the initial random guess, counting these as 3 work units (which assumes that about 7 relaxations cost the same as a V(2,1) solution cycle). Then we perform the first setup cycle, which also includes the aggregation construction and estimation of the spectral radii. This cycle is identical in all settings and the aggregates, as well as the spectral radii, are calculated only once in that cycle and kept fixed for the rest of the solution process. $u_{set}$ and $t_{set}$ represent the work-units and time (in seconds) of the first setup cycle, while
5. Aggregation-based Adaptive AMG for Markov Chains

\( u_{sol} \) and \( t_{sol} \) are the work-units and timings of the solution without the initial setup cycle. \( V_{EIS}, V_{SOL} \) are the number of EIS cycles (excluding the initial one) and solution cycles, respectively. The operator complexity denoted by \( C_{op} \) is the total number of non-zeros in the matrix \( A \) and all its coarse-grid approximations, divided by the number of non-zeros of the fine-grid \( A \). The operator complexity and number of levels are similar in all settings since all use the same Bottom-Up algorithm with a preferred aggregate size of \( s = 4 \). The asymptotic convergence factor is denoted by \( \gamma \) and is defined as the geometric mean of the residual drop factor of the last 5 solution cycles.

For the smaller-sized problems with about 65,000 unknowns, we use \( \varepsilon_{\alpha} = 10^{-4} \) as a default safe accuracy threshold for the algorithms considered. For the larger problems of about 262,000 unknowns, we use \( \varepsilon_{\alpha} = 10^{-5} \).

### 5.4.1 Serial environment experiments

We compare the AFTER algorithm with various values of \( \varepsilon_{\alpha} \) to the on-the-fly algorithm with the safe pre-chosen \( \varepsilon_{\alpha} \), and a convergence parameter of \( \gamma = 0.75 \) (detailed in Algorithm 6).

**Tandem Queue Markov Chain.** The first problem considered is the tandem queueing network problem appearing in [16, 65, 66, 76, 122]. This problem is nonsymmetric and has a significantly complex spectrum, which lies in the triangle whose vertices are \(-0.5 \pm \frac{\sqrt{3}}{2} i \) and 1. The stencil of the matrix \( A \) is given by

\[
\begin{pmatrix}
-\mu & 1 & -\mu_1 \\
-\mu_1 & 1 & -\mu_2 \\
-\mu_2 & 1 & -\mu_1 \\
\end{pmatrix},
\]

where \( \mu = 10/31, \mu_1 = 11/31 \) and \( \mu_2 = 10/31 \).

Table 4 shows the results for the Tandem Queue problem using the EIS algorithm as a solver (Algorithm 3), AFTER algorithm (Algorithm 5) with various choices of \( \varepsilon_{\alpha} \), and results for the non-overlapping on-the-fly algorithm (Algorithm 6).

The results show that a high accuracy of the prototype is required for this test-case. More setup cycles in the AFTER algorithm, or equivalently using a smaller \( \varepsilon_{\alpha} \), produced better asymptotic convergence factors. The AFTER and OnTheFly algorithms, with sufficiently
5. Aggregation-based Adaptive AMG for Markov Chains

### Table 4: Tandem Queue results

<table>
<thead>
<tr>
<th>size</th>
<th>lev</th>
<th>$C_{op}$</th>
<th>$u_{set}(t_{set})$</th>
<th>algorithm</th>
<th>$V_{EIS}, V_{SOL}$</th>
<th>$\gamma$</th>
<th>$u_{sol}(t_{sol})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>65536</td>
<td>7</td>
<td>1.64</td>
<td>23 (0.35s)</td>
<td>EIS</td>
<td>15 , 0</td>
<td>0.34</td>
<td>63 (0.91s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER(10^{-2})</td>
<td>0 , 32</td>
<td>0.64</td>
<td>35 (0.44s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER(10^{-3})</td>
<td>2 , 16</td>
<td>0.45</td>
<td>27 (0.35s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER(10^{-4})</td>
<td>3 , 12</td>
<td>0.36</td>
<td>28 (0.37s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER(10^{-5})</td>
<td>5 , 10</td>
<td>0.34</td>
<td>33 (0.44s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OnTheFly(10^{-4})</td>
<td>2 , 13</td>
<td>0.34</td>
<td>25 (0.33s)</td>
</tr>
<tr>
<td>262144</td>
<td>8</td>
<td>1.65</td>
<td>22 (1.00s)</td>
<td>EIS</td>
<td>16 , 0</td>
<td>0.36</td>
<td>85 (3.97s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER(10^{-2})</td>
<td>0 , 61</td>
<td>0.77</td>
<td>64 (2.92s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER(10^{-3})</td>
<td>2 , 22</td>
<td>0.54</td>
<td>35 (1.60s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER(10^{-4})</td>
<td>3 , 15</td>
<td>0.43</td>
<td>33 (1.50s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER(10^{-5})</td>
<td>5 , 11</td>
<td>0.35</td>
<td>39 (1.80s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OnTheFly(10^{-5})</td>
<td>2 , 15</td>
<td>0.35</td>
<td>30 (1.44s)</td>
</tr>
</tbody>
</table>

Small $\varepsilon_\alpha$, achieved asymptotic convergence rates similar to EIS. The cost of the EIS iterations, however, is significantly higher. The timings of AFTER clearly show the tradeoff: more setup cycles gain better asymptotic convergence, but require more computational time. Cost-wise, $\varepsilon_\alpha = 10^{-3}$ or $10^{-4}$ is the best compromise for this test case, even though the resulting convergence factors are not optimal. The on-the-fly algorithm achieved the best performance, using several solution cycles to reach the required accuracy instead of some of the setup cycles in AFTER.

**Random walk on a nonsymmetric random planar graph.** The next test problem is a random walk on a nonsymmetric unstructured planar graph. A similar problem with a symmetric graph appears in [65, 66, 122], and a slightly different nonsymmetric version is considered in [68, 69]. The graph is generated by choosing $n$ random points in a unit square and applying Delaunay triangulation. Then, a spanning tree of the graph is generated by a DFS algorithm and its edges are kept undirected. For the remaining edges, a direction is chosen at random. The spanning tree edges are kept undirected in order to ensure that the graph is strongly connected and hence the associated matrix is irreducible. The weight of an edge $(i, j)$ is determined by the reciprocal of the outgoing degree of node $i$, that is, we normalize the columns of the binary matrix representing the graph to make it column-
stochastic. The problem is demonstrated in Figure 7.

Figure 7: Left: Nonsymmetric random planar graph—directed edges are marked with dotted lines while undirected edges are marked with a solid line. Right—the spectrum of the Markov chain bounded by the unit circle.

Table 5 shows the results for this problem. Here, the accuracy required for the prototype vector was lower than in the previous problem. In the case of AFTER, requiring excessive accuracy from the prototype increases the solution time due to the expensive setups. However, such a large default choice of $\varepsilon_\alpha = 10^{-2}$ is too risky in general, and may result in very poor behavior in many other problems. Also, this choice has the worst asymptotic convergence ($\gamma \approx 0.5$). The on-the-fly approach does not suffer from this drawback and performs essentially as well as the best choice of AFTER in terms of solution cost, and with a better asymptotic convergence rate ($\gamma \approx 0.36$). The pure EIS solver is again the most expensive method by far. Its asymptotic convergence factor is similar to the other methods when $\varepsilon_\alpha$ is small enough.

Random walk on a Triangular Lattice. The last test problem appears in [122], and is taken from [116]. It considers a random walk on an $(m + 1) \times (m + 1)$ triangular grid, as shown in Figure (8) for $m = 6$. The points of the grid are labeled $(j, i), (i = 0, ..., m; j = 0, ..., m - i)$. From the point $(j, i)$, a transition may take place to one of the four adjacent points $(j \pm 1, i \pm 1)$. The probability of jumping to either of the nodes $(j - 1, i)$ or $(j, i - 1)$ is $\left(\frac{j + i}{m}\right)$, with the probability split equally between the two nodes when both are on the grid. The probability of jumping to either of the nodes $(j + 1, i)$ or $(j, i + 1)$ is $\left(1 - \frac{j + i}{m}\right)$, with the probability again split equally when both nodes are on the grid. The spectrum of this
matrix is real except for a few complex eigenvalues. This is one of the most difficult cases we tested with our smoothed aggregation solver.

Table 6 shows the results for this problem. For the bigger problem, using the AFTER algorithm requires several setup cycles in order to reach the optimal asymptotic converge. Again, we see the tradeoff of the AFTER algorithm, as more setup cycles lead to a better convergence but require expensive computations. In the smaller problem, one additional setup is optimal for AFTER. As expected, using the EIS method by itself has the highest cost. The on-the-fly approach again achieves the best timings.

**Discussion.** We observe a high variability in the behavior of the SA solver with different strategies in different problems. In cases where the initial setup suffices for a reasonable convergence (even if not optimal), additional setup may not be cost-effective. This depends on the gain in performance that the additional setup provides compared to its cost. However, if additional setup cycles are needed, it is likely that the non-overlapping on-the-fly method is a better choice than AFTER.
Figure 8: Triangular lattice of 28 states

<table>
<thead>
<tr>
<th>size</th>
<th>lev</th>
<th>$C_{op}$</th>
<th>$u_{set}(t_{set})$</th>
<th>algorithm</th>
<th>$V_{EIS}, V_{SOL}$</th>
<th>$\gamma$</th>
<th>$u_{sol}(t_{sol})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>65536</td>
<td>7</td>
<td>1.95</td>
<td>23 (0.37s)</td>
<td>EIS</td>
<td>34, 0</td>
<td>0.60</td>
<td>173 (2.72s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER($10^{-2}$)</td>
<td>1, 38</td>
<td>0.63</td>
<td>46 (0.71s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER($10^{-3}$)</td>
<td>4, 27</td>
<td>0.58</td>
<td>50 (0.75s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER($10^{-4}$)</td>
<td>7, 23</td>
<td>0.59</td>
<td>63 (0.96s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER($10^{-5}$)</td>
<td>11, 21</td>
<td>0.58</td>
<td>81 (1.24s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OnTheFly($10^{-4}$)</td>
<td>2, 27</td>
<td>0.58</td>
<td>41 (0.63s)</td>
</tr>
<tr>
<td>262144</td>
<td>8</td>
<td>1.96</td>
<td>22 (1.21s)</td>
<td>EIS</td>
<td>39, 0</td>
<td>0.65</td>
<td>215 (12.3s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER($10^{-2}$)</td>
<td>1, 63</td>
<td>0.82</td>
<td>72 (3.90s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER($10^{-3}$)</td>
<td>4, 53</td>
<td>0.76</td>
<td>80 (4.23s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER($10^{-4}$)</td>
<td>7, 30</td>
<td>0.65</td>
<td>70 (4.01s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AFTER($10^{-5}$)</td>
<td>12, 25</td>
<td>0.64</td>
<td>91 (5.05s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OnTheFly($10^{-5}$)</td>
<td>2, 36</td>
<td>0.64</td>
<td>52 (2.82s)</td>
</tr>
</tbody>
</table>

Table 6: Result Summary: Random walk on a Triangular Lattice
5.4.2 Parallel environment simulation

In this section, we simulate a parallel environment using our serial implementation. We thus estimate the expected results of a comparison between the non-overlapping on-the-fly algorithm and the overlapping algorithm in a parallel environment when the computing resources are doubled. We again measure the expected performance in work-units, each equivalent to the cost of a single parallelized $V_{SOL}$ cycle. As mentioned before, the more extensive parallelization results in a higher cost ratio between setup cycles and solution cycles. Therefore, the relative efficiency of parallel adaptive AMG algorithms deteriorates as the repeated setup cycles become more expensive (in work-units). In this section, we assume that we are already in a regime where adding more computing resources does not improve the cost of the multigrid cycles for a problem of fixed size.

We denote the run times of EIS and solution cycles in a serial environment by $time(V_{EIS}^{\text{serial}})$ and $time(V_{SOL}^{\text{serial}})$. These timings are measured in our serial code. $time(V_{EIS}^{\text{parallel}})$ and $time(V_{SOL}^{\text{parallel}})$ denote the corresponding timings in a parallel environment. Next, we define

$$R_p = \frac{time(V_{EIS}^{\text{parallel}})/time(V_{SOL}^{\text{parallel}})}{time(V_{EIS}^{\text{serial}})/time(V_{SOL}^{\text{serial}})},$$

(53)

which is the increase in the cost ratio between a setup and solution cycle as a result of the parallelization. The ratio $R_p$ is expected to grow as we use more computing resources for a fixed problem size, because each computing node gets less grid points and hence communication is relatively more expensive.

We compare the non-overlapping and overlapping on-the-fly algorithms (Algorithms 6 and 7 respectively), considering again only the solution time beyond the first setup cycle, as this cycle is the same in both algorithms. The simulation of the non-overlapping on-the-fly algorithm is performed simply by multiplying the cost of $V_{EIS}$ by $R_p$ while keeping the cost of $V_{SOL}$ fixed. For Algorithm 7 we need to simulate the procedure overlapping-setup in Section 5.3.2. We do this by multiplying the time it takes to perform an EIS cycle by $R_p$. This time (measured in work units) gives us the number of solution cycles carried out in parallel with the EIS cycle. We further assume that the communication required at each crossover costs the same as one solution cycle, so we add one work-unit to the cost of overlapping-setup. In the non-overlapping on-the-fly algorithm we use $\gamma = 0.75$ as before. In overlapping OTF we use a lower value of $\gamma = 0.5$, allowing more setup updates. The reason for this is that
setup cycles constitute a smaller burden in this case because solution cycles are performed simultaneously.

In the following graphs we compare the estimated costs of the overlapping and non-overlapping OTF algorithms as a function of $R_p$ in a parallel environment. Since we double the resources in overlapping OTF, the performance gain factor that can be achieved is bounded by 2. This is true also if we use double computing resources for the non-overlapping OTF. Therefore, as a reference, we also plot half the time of non-overlapping OTF (denoted by ‘OTF/2’).

**Tandem Queue Markov Chain.** Figure 9 shows the amount of work-units required for the solution as a function of $R_p$ defined in Equation (53). The results indicate that the overlapping on-the-fly method has better performance as $R_p$ increases. Indeed, the asymptotic slope of the overlapping OTF curve seems similar to that of the ideal OTF/2.

**Random walk on a nonsymmetric random planar graph.** Figure 10 shows the relevant results in work-units. Here, as in the results for $AFTER(10^{-2})$ shown in Table 5, one can see that the initial setup cycle suffices for achieving an adequate convergence factor. The overlapping on-the-fly algorithm takes full advantage of that, because the overlapping solution cycles are quite effective even though the operators are not very accurate yet. In
both problem sizes we see that the solution cost increases very little with $R_p$.

Technically, the best option for this problem is to simply apply the initial setup and then continue with solution cycles with no additional EIS cycles. This approach is not influenced by the increasing $R_p$. However, as in the serial environment case, we cannot know this in advance so it is not a useful default choice for a general algorithm.

**Random walk on a Triangular Lattice.** The results are shown in Figure 11. Again, the overlapping on-the-fly algorithm achieves better performance than the non-overlapping version, although the performance gain is less significant than before. The asymptotic slope of the overlapping OTF is again similar to that of OTF/2.

**Discussion.** In a parallel computing environment the overlapping on-the-fly method has an advantage over its non-overlapping version, at the expense of the additional hardware that is required. We note again that if doubling the resources for an internally parallelized non-overlapping OTF significantly improves its performance, then that option would probably be a better choice than overlapping OTF. It would create curves that lie somewhere in between the OTF and OTF/2 curves, and that is also where the overlapping OTF curve lies. Once the resources are such that further internal parallelization is ineffective, the overlapping OTF approach can be employed to advantage.
5.5 Conclusions and Future Work

In this work, we introduced some novel adaptive AMG approaches for computing the principal eigenvector of column-stochastic matrices. We investigated a simpler version of the Smoothed Aggregation multigrid approach, where only the prolongation is smoothed. This method exhibits better convergence properties than the pure aggregation algorithm and is only mildly more complicated. In addition, a new variant of the bottom-up aggregation method of [95, 96] is introduced. In this approach, nodes are aggregated in the order of increasing dominance, without giving a special meaning to a seed of an aggregate. The combination of this algorithm and the simple SA approach, allows us to choose relatively small aggregates, which leads to nice convergence and operator complexity properties of the multigrid solver. The performance is quite promising, and it is competitive with respect to the smoothed aggregation and AMG approaches presented in [65, 66].

The second part of this work introduced an on-the-fly adaptive multigrid approach of interleaving adaptive EIS and classical multigrid cycles for Markov chain problems. The new idea is demonstrated using the adaptive SA method mentioned above. For the problems considered, using classical solution cycles instead of EIS cycles significantly reduced the costs of the solution process. The on-the-fly approach was generally found to be superior to applying solution cycles after a fixed number of EIS setup cycles. It was also successfully applied in [114] for other types of multigrid solvers.
A strategy more suitable to a parallel environment was also suggested, applying overlapping setup and solution cycles. This strategy seems to have potential in parallel computation, in regimes where the effectiveness of traditional parallelization has been exhausted, and yet additional resources are available.
6 Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

In this section we describe our second study on non-Galerkin multigrid coarsening. This section is based on the following paper, which is in the final stages of the review process:


More discussions and results can be found in


The latter short paper is not a part of this dissertation. It is a part of our co-author Ran Zemach’s M.Sc. thesis.

6.1 Introduction

Let us summarize the main concepts of AMG discussed, in Section 4.2. AMG methods are mainly used to solve the linear system (2) using two complementary components: relaxation, and coarse-grid correction (CGC). The relaxation (3) is usually inefficient in handling certain error modes, called “algebraically smooth”, which relate to the low energy eigenvectors of $A$. CGC aims at handling these modes, and is done by solving a smaller-size coarse-grid problem (8). The coarse grid matrix $A_c$ in (8) is mostly defined by the Galerkin or Petrov-Galerkin\(^4\) coarsening in (9), which correspond to a projection method.

Because the coarse-grid problem is also typically too large to solve directly, the coarsening process is applied recursively, resulting in a hierarchy of successively coarser problems and

\(^4\)Most AMG methods target symmetric linear systems and use the Galerkin operator, where $R = P^T$. For non-symmetric problems, a Petrov-Galerkin operator is often used, that is $R \neq P^T$. In this work we target both symmetric and non-symmetric problems, and use both Galerkin and Petrov-Galerkin operators, respectively. Throughout this section, we denote the restriction by $R$, and present the non-symmetric version of the multigrid algorithms. However, in many cases we use the notion of “Galerkin” coarsening to represent both options, versus the alternative option of “Non-Galerkin” coarsening which is the main topic of this work.
their associated matrices. One drawback of the coarsening is that the control over the sparsity of $RAP$ is quite limited and is dictated by $P$ and $R$. This often leads to an increased density of the Galerkin coarse-grid operators, and high overall computational complexity of the multigrid algorithm. Thus, one must often compromise between the quality of these operators, and the aggressiveness of coarsening, which affect the rate of convergence and the operator complexity of the algorithm. The latter is defined as the total number of non-zeros in the whole multigrid hierarchy divided by those of the fine-grid matrix alone.

On top of that, maintaining a low operator complexity may not be the only restriction on the multigrid hierarchy. Classical Galerkin algorithms such as AMG or SA, which include high quality transfer operators, typically lead to a severe stencil growth on coarse grids, even when the operator complexity of the hierarchy remains moderate. In the case of large-scale parallel computing, this usually leads to high communication overhead on coarse grids [115, 110] and loss of scalability, especially in 3D. In [119] it was suggested to apply aggressive coarsening on the finer levels which leads to reduced complexities. Such coarsening is defined by generalizing the strength of connection between variables to strengths of short distance paths, rather than strengths of neighbors alone. The generalization for distance-two paths can be implemented as applying standard coarsenings twice. Similar parallel coarsening schemes were introduced in [115]. This, however, requires modifications to the standard prolongation definitions. In particular, in the classical AMG interpolation, there cannot be strong connections between $F$-points without a mutual neighboring $C$-point. This rule breaks if aggressive coarsening is applied, and to overcome this, [143, 46] suggest some modifications to the standard prolongation definitions to support long-range interpolation. These modifications, however, harm the quality of the prolongation, and the resulting AMG algorithms typically require more iterations to converge than standard (serial) AMG variants. For more information about parallel AMG see [5, 41, 2, 3, 4] and references therein.

Recently, there has been an effort to develop efficient multigrid algorithms that explicitly control the sparsity pattern of the multigrid hierarchy [137, 124], or sparsify the Galerkin AMG operators [110]. These ideas have yet to reach their full potential and were only applied for symmetric problems. We follow a similar framework in this work, and propose a new method that is applicable to both non-symmetric and symmetric problems. Other related works include [120, 84, 83], which apply similar sparsification techniques for solving graph-Laplacian problems that arise from computer vision applications. In these methods, the
multilevel transfer operators are defined by a Schur complement, or “exact interpolation”, and the resulting coarse operators are sparsified to maintain reasonable coarsening rate and operator complexity.

To demonstrate our algorithm on non-symmetric problems, we focus on the convection-diffusion equation

$$-\epsilon \Delta u + \mathbf{v} \cdot \nabla u = f,$$

that appears in flow simulations, and is commonly studied in the context of geometric and algebraic multigrid [24, 25, 144, 145, 63, 32, 108, 97]. One popular AMG approach to treating this problem is by using aggregation-based AMG methods [131, 63, 29, 32, 108, 95, 96, 97], where the coarsening is done by clustering (aggregating) the grid unknowns. In the simple aggregation method (AGG), $P$ and $R$ are typically sparser than those obtained by most other AMG approaches, and the operator complexity of the multigrid hierarchy is usually well-bounded and attractive. However, it is difficult to obtain grid independent convergence using this approach, and therefore, the approach of Smoothed Aggregation (SA) [131, 29, 32, 108, 65] is often preferred over AGG. In SA we smooth the simple aggregation operators by a relaxation operator. This improves the convergence properties of the multigrid solver, but it also increases the operator complexity of the multigrid hierarchy. Therefore, when using SA we must make sure that our coarsening is aggressive enough to prevent exaggerated stencil growth on coarse grids. However, for a convection-dominated problem (54), such aggressive aggregation coarsening (say $3 \times 3$ for 2D problems on structured meshes) significantly harms the quality of the aggregation operators compared to moderate coarsening (say, $2 \times 2$) [63]. The latter, on the other hand, leads to an unbounded operator complexity using SA. This behavior is an example of the tradeoff between using high quality transfer operator vs. maintaining low operator complexity. We note that there are more sophisticated methods to improve SA by changing the values of the transfer operators $P$ and $R$. Such methods, called “energy-minimization” methods [90, 140, 26, 100, 111], optimize the non-zero entries of the transfer operators for some constrained energy measure given a predefined sparsity pattern. By that, these methods improve the quality of the transfer operators without increasing the overall operator complexity. Also related are the adaptive and bootstrap AMG methods [31, 29, 32, 20] that involve learning of the characteristic algebraically smooth errors instead of assuming them apriori.

As mentioned before, multigrid algorithms often exhibit a severe stencil growth on coarse
6. Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

Figure 12: A comparison between the sparsity patterns of AGG and SA on level 3, for the 3D Poisson equation on a $64^3$ grid, using a 7-point stencil. The left matrix corresponds to the AGG algorithm; it is of size $6538 \times 6538$, and contains 99,878 non-zeros, which is about 15 non-zeros per row on average. The right matrix corresponds to the SA algorithm, and is smaller because of the more aggressive coarsening. It is of size $1645 \times 1645$, and contains 120,299 non-zeros, which is about 73 non-zeros per row on average.

grids, even though their operator complexity main remain moderate. Figure 12 compares between the sparsity patterns of AGG and SA on the third level, for the solution of the 3D Poisson equation on a $64^3$ cubic grid, using the standard 7-point stencil. The aggregates in both cases are calculated using the same algorithm as in [131]. In this case, the coarsening of SA is much more aggressive, but the matrix of AGG is much sparser than that of SA, where the stencil growth is evident. The stencil growth of SA further increases on coarser grids, until the matrix becomes essentially dense on the coarsest grid. Nevertheless, both algorithms exhibit a well bounded operator complexity of their multigrid hierarchies (1.25 and 1.6 for AGG and SA respectively). In our numerical results we observe much worse scenarios, especially for a 3D convection-dominated problem (54). We note that the stencil growth of SA in Figure 12 can be prevented if $3 \times 3 \times 3$ aggregates are chosen for the finest grid instead of the neighborhood aggregation of [131]. Then, a 27-point stencil is maintained on all levels.

In the recent paper [97], problem (54) is solved using AGG with moderate coarsening of factor close to 4. To overcome the slow convergence caused by using simple aggregation, the multigrid process includes acceleration on all levels of the hierarchy, requiring a
more elaborate recursive structure (usually W-cycles with recursive Krylov accelerations, known as K-cycles). W-cycles are also used in [25]. However, such cycles may be costly, even though their storage complexity is well-bounded, especially when considering parallel multigrid computations [115, 41].

In this work, we present a new AMG algorithm that controls the sparsity pattern of the multigrid hierarchy, using ideas related to [124, 110]. The new algorithm is developed for non-symmetric problems but it preserves the symmetry property of the coarse-grid operators for symmetric fine-grid operators. Our algorithm uses the aggregation framework as a basis platform, by applying SA to define the prolongation and restriction, and AGG Galerkin coarse-grid operators for defining a sparse non-zero pattern of $A_c$. Once the transfer operators and the target sparsity pattern are set, our algorithm sparsifies the Galerkin SA coarse-grid operator to match the chosen sparsity pattern of $A_c$.

### 6.2 Motivation and Background

We follow the rationale of [137, 124], where the main idea is to separate the issue of the coarse-grid operator sparsity pattern from the transfer operators $R$ and $P$. We fix the sparsity pattern of $A_c$ and, independently, use high quality transfer operators $P$ and $R$. Denote the Galerkin operator by

$$A_g = RAP,$$

and assume that the current error, $e$, is in the range of $P$, i.e., $e = Pe_c$, where $e_c \in \mathbb{R}^{n_c}$ is some coarse vector. Also, assume that our coarse operator, $A_c$, is constructed such that $A_c e_c = A_g e_c$. Then, the two-level cycle (10) of such an algorithm eliminates the error $e$:

$$e_{\text{new}} = [I - P(A_c)^{-1}RA] e = [I - P(A_c)^{-1}RA] Pe_c$$
$$= P[I - (A_c)^{-1}A_g]e_c = P(A_c)^{-1}[A_c - A_g]e_c = 0. \quad (55)$$

Because $A_c$ has fewer non-zeros than $A_g$, the equality $A_c e_c = A_g e_c$ cannot hold for all $e_c$. However, because the CGC (coarse-grid correction) is applied after the relaxation, the error $e$ before the CGC is typically smooth, and so is its coarse version $e_c$. Therefore, motivated by this property and (55), [137, 124] suggest to define the operator $A_c$ such that $A_c e_c = A_g e_c$ holds with respect to algebraically smooth errors, which are not eliminated by the relaxation. If $A_c$ and $A_g$ yield a similar coarse grid correction for those errors, the efficiency of our solution
6. Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

The process will not degrade due to the use of $A_c$ instead of $A_g$. Additionally, note that multigrid algorithms rely on the property that the algebraically smooth errors are in the range of $P$, and therefore they influence the definition of $P$. Therefore, when applying the non-Galerkin approach, the characterization of the smooth error modes should already be available to us from the construction of $P$. In some cases, the algebraically smooth errors are known. In the context of discretizations of scalar PDEs, the algebraically smooth error modes correspond to the constant and linear functions.

In addition, several algorithms in the literature show that the large Petrov-Galerkin stencil is not necessary, at least for simple cases. The oldest of those is the geometric multigrid, where the PDE that is associated with the fine level matrix $A$ is re-discretized on a coarse grid, without any relation to the prolongation or restriction. This mechanism does not introduce stencil growth on coarse grids, however, it is limited to solutions of PDEs on structured grids.

Furthermore, papers such as [95, 96, 97, 101, 114, 10, 134] apply simple aggregation (AGG) to various symmetric and non-symmetric problems. To overcome the slow convergence, some of those use accelerated W-cycles, and some apply acceleration by overcorrection. All of them rely on the fact that the two-level AGG cycle has a mesh-independent convergence, and the convergence deteriorates only in multilevel cycles. This may suggest that the coarse grid sparsity patterns of the AGG algorithms may suffice for mesh-independent convergence—we only need to adapt the values of the entries in the matrices. These properties motivate our choice of using AGG for the sparsity pattern of $A_c$. The simplest example for that is AGG with a constant overcorrection, which may yield good convergence rates even for unstructured and non-symmetric problems [114]. Choosing $\alpha > 1$ as a constant over-correction parameter is equivalent to choosing $A_c/\alpha$ as a coarse-grid operator:

$$I - \alpha P(A_c)^{-1}RA = I - P(A_c/\alpha)^{-1}RA.$$  

$A_c/\alpha$ can be interpreted as a non-Galerkin operator, with the same sparsity pattern as AGG but with different values.

All the works in [137, 124, 110, 120, 84, 83, 17] apply a sparsification mechanism to some multilevel algebraic framework, aiming to reduce the number of non-zeros of the coarse-grid operators while maintaining “spectral equivalence” to the original coarse operator. The simplest of those is [17], which is limited to circulant matrices, but other than that is quite
general. It shows how to generate spectrally equivalent coarse operators using 5-pt stencils instead of 9-pt stencils in 2D, or 7-pt stencils instead of 27-pt stencils in 3D. The simplest example is as follows: let the fine level $A$ be the 5-pt Laplacian operator ($H_A$), then the classical AMG coarsening generates a 9-pt stencil on the coarse grid ($H_{A_g}$). This stencil can be replaced by a spectrally equivalent 5-pt stencil ($H_{A_c}$) as follows:

$$H_A = \begin{pmatrix} -1 & 4 & -1 \\ -1 & 4 & -1 \\ -1 & 4 & -1 \end{pmatrix}, \quad H_{A_g} = \frac{1}{64} \begin{pmatrix} -1 & -2 & -1 \\ -2 & 12 & -2 \\ -1 & -2 & -1 \end{pmatrix}, \quad H_{A_c} = \frac{1}{16} \begin{pmatrix} -1 & 4 & -1 \\ -1 & 4 & -1 \end{pmatrix}. \quad (56)$$

This equivalence can be also seen from the fact that both $H_{A_g}$ and $H_{A_c}$ are discretizations of the ‘$-\Delta$’ operator with an equal scaling of $\frac{1}{16}$, which comes from the multiplication of $A$ with $P$ and $P^T$. We note that our sparsification method in this work also sparsifies $H_{A_g}$ to $H_{A_c}$, as do the methods of [137, 83]. The work of [110] shows a condition for the non-Galerkin method to work, which is related to (55). It shows that spectral equivalence between the two SPD coarse operators can be measured by the value of

$$\|I - (A_c)^{-1} A_g\|_2. \quad (57)$$

If (57) is smaller than and bounded away from 1, then the two-level cycle with $A_c$ instead of $A_g$ converges, and the performance of the two cycles becomes more similar as (57) tends to 0.

In the next section we describe our Sparsified Smoothed Aggregation (SpSA) framework, which uses Smoothed Aggregation for the transfer operators, and AGG for the sparsity pattern. Our version of SA is similar to the classical smoothed aggregation [131] described earlier in Section 4.2.4, except a few small changes that are summarized in Appendix B. Even though these are small, in some cases they dramatically change the properties of the SA solver compared to the classical one. Generally, our version of SA has better convergence, but also has a higher operator complexity than the original SA method.

We define our aggregation by a modified neighborhood approach, similar to the classical one of [131]. Other aggregation-based coarsening methods as in [122, 123, 96, 97] may also be suitable for our algorithm. The filtering in (29) aims at removing small entries from $P$ and $R$, to prevent unnecessary stencil growth. In the context of convection-dominated
problems (54), the small diffusion entries may have little influence on the quality of $P$, but cause a severe stencil growth on the coarse-grid operators. This also happens in anisotropic diffusion. We note that such filtering is not essential in our algorithm because the coarse operators are sparsified and stencil growth is prevented. However, we still recommend using this filtering to reduce the computational cost of both the coarse operator sparsification in the setup phase, and the cost of applying the restriction and prolongation in the solution phase.

6.3 The Sparsified Smoothed Aggregation (SpSA) Algorithm
Our algorithm, similarly to all the other non-Galerkin algorithms mentioned above, consists of three separate tasks:

1. Choose the transfer operators $R$ and $P$.
2. Choose the sparsity pattern for $A_c$.
3. Calculate the entries in $A_c$.

The first step is essential to every algebraic multigrid method and the other two are needed only in non-Galerkin AMG.

As mentioned above, our algorithm uses the aggregation framework. For the first task, it uses transfer operators $R = R_s$ and $P = P_s$ based on Smoothed Aggregation in (26)-(27). For the second and third tasks, we use two Galerkin operators

$$A_t = R_t AP_t, \quad \text{and} \quad A_g = R_s AP_s,$$

(58)

that are based on the non-smoothed and smoothed aggregation operators in (25) and (26)-(27), respectively. Once the target pattern $A_t$, and the Galerkin SA operator $A_g$ are set, our algorithm sparsifies $A_g$ to match the chosen sparsity pattern of $A_t$; the result is the coarse operator $A_c$. This setup process is described in Algorithm 8.

6.3.1 Sparsity Patterns in the Aggregation Framework
The operators in (26)-(27) and (58) have some unique properties that are related to their sparsity patterns. These will be used in our sparsening algorithm, described later. We denote
Algorithm: SpSA-Setup

1. Define the tentative prolongation $P_t$ and restriction $R_t = P_t^T$.
2. Define SA operators $P_s, R_s$ (see (26)-(27)).
3. Apply Galerkin Coarsening: $A_t = R_t AP_t$, $A_g = R_s AP_s$.
4. Sparsify $A_g$ to the sparsity pattern of $A_t$: $A_c = \text{Sparsify}(A_g, A_t)$
5. Apply recursion on $A_c$ to generate the next levels.

Algorithm 8: Sparsified Smoothed Aggregation (SpSA) Setup

the sparsity pattern of any matrix $A$ as

$$\mathcal{S}_P(A) = \{(i, j) : A_{i,j} \neq 0\}.$$  \hspace{1cm} (59)

By (26)-(27), we have that

$$\mathcal{S}_P(A_g) \supseteq \mathcal{S}_P(A_t),$$  \hspace{1cm} (60)

up to chance cancellations of elements which we ignore in our description. In addition, since $\mathcal{S}_P(A) = \mathcal{S}_P(I - QA)$ for any diagonal matrix $Q$, we have that

$$\mathcal{S}_P(A_t) \supseteq \mathcal{S}_P(R_s P_t) \quad \text{and} \quad \mathcal{S}_P(A_t) \supseteq \mathcal{S}_P(R_t P_s),$$  \hspace{1cm} (61)

with equalities in the case of no filtering in (26)-(27).

As mentioned above, the tentative operators (25), and their associated Galerkin operator, may be efficient when used directly in a multigrid process with acceleration. This suggests that the sparsity pattern of $A_t$ will not miss important non-zero entries. For example, if the graph of $A_g$ is connected, i.e., there is a path from each node $i$ to node $j$ in the graph, then the graph of $A_t$ is connected as well. To illustrate this, consider the grid-aligned anisotropic Laplacian using 5-point finite-difference discretization as in [110]. The stencil of the fine-level
6. Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

Figure 13: The sparsity pattern of the pure aggregation Galerkin operator. The left image shows the sparsity pattern (a graph) of a fine-grid matrix and circles that correspond to aggregates. On the right image the sparsity pattern of the aggregation Galerkin operator. The image shows that the edges between aggregates are preserved on the coarse grid, and no new connections are generated by the coarsening. The image is taken from the homepage of Prof. Yvan Notay.

A, and stencil $A_g$ using classical AMG intepolation with semi-coarsening are given by:

$$H_A = \begin{pmatrix} -1 & -\epsilon & \epsilon \\ 2 + 2\epsilon & -1 \\ -\epsilon \end{pmatrix}, \quad H_{A_g} = \frac{1}{4} \begin{pmatrix} -\epsilon & -6\epsilon & -\epsilon \\ -2 + 6\epsilon & 4 + 12\epsilon & -2 + 6\epsilon \\ -\epsilon & -6\epsilon & -\epsilon \end{pmatrix}.$$

Now, it is important that any non-Galerkin operator will have some of the weak $\epsilon$ entries in $A_g$. A stencil of $H_{A_c} = \begin{bmatrix} -1/2 & 1 & -1/2 \end{bmatrix}$, for example, would split the graph of $A_g$ into disconnected strong diffusion lines. As shown in [110], this would not be a good choice of a coarse-grid stencil, even though the norm of their difference is small: $\|A_c - A_g\|_F = O(\epsilon)$, where $\| \cdot \|_F$ is the Frobenius norm. In our case, the stencil of $A_t$ will always contain some of the non-zeros in the weak directions. More importantly, recall that $(a_t)_{ij} = \sum_{k \in C_i, \ell \in C_j} a_{k\ell}$, so if the graph of the fine-grid matrix $A$ is connected, then so is the graph of $A_t$, given that no chance cancellations of elements occur, as in M-matrices. This property is illustrated in Figure 13.
6.3.2 The Sparsening Procedure

We next describe our sparsening procedure used to approximate $A_g$ using the sparsity pattern of $A_t$. Our process aims to generate a coarse matrix $A_c$ that, on top of its sparsity constraints, has the same product as $A_g$ and $(A_g)^T$ with typical algebraically smooth error modes. As mentioned before, if $A_c e_c = A_g e_c$ for any smooth error $e_c$, the efficiency of our solution process will not degrade due to the sparsification of $A_g$. The typical smooth error modes are also necessary in the construction of the prolongation and restriction, and in the context of elliptic PDEs, they are often chosen to be represented by the constant vector. Thus, motivated by (55), we impose

$$A_g 1 = A_c 1 \text{ and } (A_c)^T 1 = (A_g)^T 1,$$

(62)

where $1$ is the vector of ones.

We start our process by copying the values in the entries of $A_g$ that belong to $S_P(A_t)$,

$$\text{if } (A_t)_{k,i} \neq 0 \text{ then } (A_c)_{k,i} \leftarrow (A_g)_{k,i}. \quad (63)$$

Otherwise, we have an entry satisfying $(A_t)_{k,i} = 0$ and $(A_g)_{k,i} \neq 0$; this is an entry that we wish to eliminate (set $(A_c)_{k,i} = 0$), while maintaining (62). Now, by setting $(A_c)_{k,i} = 0$, we break the equalities in (62), and thus we need to correct them by changing other entries as well. Our correction will be done by collapsing $(A_g)_{k,i}$ on other entries that correspond to variables that are “strongly connected” to $k$ and $i$. In this context, the main heuristic that is used in multigrid is that if $k$ and some variable $m$ are strongly connected, then a typical algebraically smooth error $e$ will satisfy $e_k \approx e_m$. We will use this heuristic in our algorithm. We elaborate on the choice of these entries in the next section.

Obtaining a surrogate path for eliminating the $(k, i)$ entry. First, we remark that we must avoid the simplest choice of entries to make up for an elimination of a $(k, i)$ entry—the diagonal and “mirror” entries, $(k, k)$, $(i, i)$ and $(i, k)$. Considering that we might need to eliminate both $(A_g)_{k,i}$ and $(A_g)_{i,k}$, we have to satisfy four equations: the equalities in (62) for both $i$ and $k$ row and column. Since we have only the two diagonal entries $(k, k)$, $(i, i)$ at our disposal (the off-diagonals are zeroed), this task is impossible because we have four equations and only two variables to satisfy them. Thus, we conclude that, generally,
additional or other entries need to be changed. We note, however, that this is not true in the
symmetric case where \((A_g)_{k,i} = (A_g)_{i,k}\). Then, (62) can be satisfied by adding the following
2 \times 2 block

\[
\begin{pmatrix}
i & k \\
k & i
\end{pmatrix}
\begin{pmatrix}
+(A_g)_{k,i} & -(A_g)_{k,i} \\
-(A_g)_{k,i} & +(A_g)_{k,i}
\end{pmatrix}
\]

(64)

which changes only the diagonal entries, and has zero row and column sums. This property
has led to various sparsification schemes in [110, 120, 84, 83]. Because we consider both
symmetric and non-symmetric problems in this work, we assume \((A_g)_{k,i} \neq (A_g)_{i,k}\) and do
not use (64) in our sparsening scheme. In fact, we show that other corrections must be
applied in this case.

We next describe a set of entries in \(A_t\) that are non-zeros and available to use for elimi-
nating \((A_g)_{k,i}\). In Galerkin coarsening, every non-zero entry satisfies

\[
(A_g)_{k,i} = \sum_j \sum_\ell (R)_{k,j} A_{j,\ell}(P)_{\ell,i},
\]

(65)

which means that every entry in \(A_g\) is generated by a sum of three-term multiplications.
Now, if \((A_t)_{k,i} = 0\) and \((A_g)_{k,i} \neq 0\), then we have at least one non-zero term \((R_s)_{k,j} A_{j,\ell}(P)_{\ell,i}\)
in the right hand side of (65), in which \(\ell\) belongs to the aggregate \(C_{m_1}\) and \(j\) belongs to
the aggregate \(C_{m_2}\), and at least one of them is different from \(k\) and \(i\) (note that \(j\) and \(\ell\) are
fine-grid variables while the rest are coarse-grid variables). Also,

\[
(R_s P_t)_{k,m_2} \neq 0, \quad (R_t P_s)_{m_1,i} \neq 0, \quad \text{and} \quad (A_t)_{m_2,m_1} \neq 0.
\]

(66)

Furthermore, by (61), all these entries also belong to \(S_p(A_t)\), and hence can be used together
with their associated diagonal entries to eliminate \((A_g)_{k,i}\). Overall, in order to eliminate
\((A_g)_{k,i}\) we add to \(A_g\) a submatrix of the form

\[
\begin{pmatrix}
i & m_1 & m_2 & k \\
i & 0 & 0 & 0 \\
m_1 & \times & \times & 0 \\
m_2 & 0 & \times & \times \\
k & -(A_g)_{k,i} & 0 & \times
\end{pmatrix}
\]

(67)
Figure 14: A surrogate path of \((A_g)_{k,j}\). The dashed arrow represents the eliminated entry, while the surrogate path is comprised of solid arrows.

where \(\times\) denotes a non-zero entry, and require that it has zero row and column sums. We excluded \((i,i)\) and \((k,k)\) because they are singles in their row and column, respectively, and hence cannot be changed. Because the marked non-zero entries constitute a distance-three path \(i \rightarrow m_1 \rightarrow m_2 \rightarrow k\) in \(A_t\), we will denote by \((i,m_1,m_2,k)\) the “surrogate path” for eliminating the \((k,i)\) entry. Figure 14 demonstrates this path.

Setting values in the surrogate path for eliminating the \((k,i)\) entry. We now describe how to set the values in the submatrix (67) so that (62) is satisfied, or equivalently, that (67) has zero row and column sums.

By imposing \((A_c)_{k,i} = 0\), or by setting \(-(A_g)_{k,i}\) in the \((k,i)\) entry of (67), we break the zero sum of the \(k\)-th row and \(i\)-th column. To satisfy them we must set:

\[
\begin{align*}
(A_c)_{m_1,i} &\leftarrow (A_c)_{m_1,i} + (A_g)_{k,i} \\
(A_c)_{k,m_2} &\leftarrow (A_c)_{k,m_2} + (A_g)_{k,i}. 
\end{align*}
\]

(68)

Now, the corresponding zeros sum for the \(m_1\)-th row and the \(m_2\)-th column are broken, so we must satisfy them as well by applying

\[
\begin{align*}
(A_c)_{m_1,m_1} &\leftarrow (A_c)_{m_1,m_1} - (A_g)_{k,i} \\
(A_c)_{m_2,m_2} &\leftarrow (A_c)_{m_2,m_2} - (A_g)_{k,i}.
\end{align*}
\]

(69)

This again breaks the zero sum for the \(m_2\)-th row and \(m_1\)-th column, and to finally satisfy them both we must apply

\[
(A_c)_{m_2,m_1} \leftarrow (A_c)_{m_2,m_1} + (A_g)_{k,i}.
\]

(70)
Overall, following (68)-(70), the submatrix (67) is given by

\[
\begin{pmatrix}
0 & 0 & 0 & 0 \\
(A_g)_{k,i} & -(A_g)_{k,i} & 0 & 0 \\
0 & (A_g)_{k,i} & -(A_g)_{k,i} & 0 \\
-(A_g)_{k,i} & 0 & (A_g)_{k,i} & 0
\end{pmatrix},
\]

(71)

and it has a zero row and column sum.

**Partitioning the corrections onto several surrogate paths.** In practice, there may be several paths from \(i\) to \(k\), so we eliminate \((A_g)_{k,i}\) using all paths simultaneously, each eliminating a portion \(0 < \theta_{(i,m_1,m_2,k)} < 1\) of \((A_g)_{k,i}\), where the portions sum to one. The weights \(\theta_{(i,m_1,m_2,k)}\) are chosen proportionally to the strength of the connection in the associated path. More specifically, we use

\[
\theta_{(i,m_1,m_2,k)} = |(R_t P_s)_{m_1,i}(A_t)_{m_2,m_1}(R_s P_t)_{k,m_2}|
\]

(72)
as the strength of the path \((i,m_1,m_2,k)\). However, there are situations where there is a distance-two path between \(i\) and \(k\), featuring only one connector, i.e., \(m_1 = m_2\). In such cases, which are also computationally easier to find, we choose the weight associated with the path \((i,m,m,k)\) as \(|(R_t P_s)_{m_1,i}(R_s P_t)_{k,m}|\) and set the weight of the distance-three paths to 0. We note that if we consider only symmetric problems, then our distance-two correction is similar to that of [83].

In our experience, partitioning the sparsified entry between several paths is important. We demonstrate this, and also motivate our choice of the portions in (72), by looking at a simple case of a graph Laplacian sparsification. Consider the graphs in Figure 15, and their associated graph-Laplacian matrices (the off-diagonal entries in the matrices are minus the weights in the graphs, and the diagonal entries are their sum). The middle edge of the graph \(A_g\) is collapsed onto the other two paths with portion \(\delta\) and \(1 - \delta\).

We now examine the *optimal* partitioning value \(\delta\) for each value of \(a\), with respect to two different equivalence measures. The first measure is the spectral equivalence measure in (57). The second measure is \(||(I - A_c^{-1}A_g)(I - \frac{3}{4}D^{-1}A_g)||_2\)\), which roughly aims to restrict (57)
6. Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

Figure 15: Two small graph Laplacians. The right graph is a sparsified version of the left graph with partitioning $(\delta, 1 - \delta)$.

Figure 16: Optimal partitioning with respect to the two different measures. The left corresponds to minimizing $\| (I - A_c^{-1}A_g) \|_2$ and the right corresponds to minimizing $\| (I - A_c^{-1}A_g)(I - \frac{3}{4}D^{-1}A_g) \|_2$.

to smooth modes (we show additional observations regarding this later). In both of these measures, we ignore the constant vector which is the null-space of $A_g$ and $A_c$. We compare all the optimal values of $\delta$ for each $a$ with the function $\frac{a^2}{1+a^2}$. This function corresponds to the partitioning ratio determined by the product of the entries in each of the two paths, similarly to our partitioning approach of (72). Figure 16 shows the two graphs. The left graph shows that if all the eigen-modes are considered equally, then the partitioning should be different from our choice in (72), although somewhat similar. In particular the optimal $\delta$ is monotonically increasing with $a$. The right graph, which is more focused on the smooth eigen-modes almost perfectly agrees with our choice of strengths of paths. This explains our choice of sparsification partitioning. Nevertheless, this partitioning is still a subject of research.
6. Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

Algorithm: $A_c \leftarrow \text{Sparsify}(A_g, A_t, R_P, R_t P)
\%
A_g \text{ - a Galerkin smoothed aggregation operator that needs to be sparsified}
A_t \text{ - a Galerkin aggregation operator that defines the desired sparsity pattern}
R_P, R_t P \text{ - Aggregated transfer operators. These help establishing surrogate paths.}
A_c \text{ - SpSA operator - has the sparsity pattern of } A_t \text{ and quality comparable to } A_g.$
foreach $(k, i) \in S_P(A_t)$ do
  $(A_c)_{k,i} \leftarrow (A_g)_{k,i}$
end
foreach $(k, i) \in S_P(A_g) \setminus S_P(A_t)$ do
  % Seek a set of distance-two surrogate paths:
  $S_{k\rightarrow i} = \{(i, m, m, k) : (R_t P_s)_{m,i} \neq 0 \land (R_s P_t)_{k,m} \neq 0\}.$
  if $S_{k\rightarrow i} \neq \emptyset$ then
    foreach $(i, m, m, k) \in S_{k\rightarrow i}$, associate a weight
    $\theta_{(i,m,m,k)} = |(R_t P_s)_{m,i}(R_s P_t)_{k,m}|$
    Normalize the weights: $\theta_{(i,m,m,k)} \leftarrow \left(\sum_p \theta_{(i,p,p,k)}\right)^{-1} \theta_{(i,m,m,k)}$
  end
else
  % Define a set of distance-three surrogate paths:
  $S_{k\rightarrow i} = \{(i, m_1, m_2, k) : (R_t P_s)_{m_1,i} \neq 0 \land (A_t)_{m_2,m_1} \neq 0 \land (R_s P_t)_{k,m_2} \neq 0\}.$
  foreach $(i, m_1, m_2, k) \in S_{k\rightarrow i}$, associate a weight
  $\theta_{(i,m_1,m_2,k)} = |(R_t P_s)_{m,i}(A_t)_{m_2,m_1}(R_s P_t)_{k,m}|$
  Normalize the weights: $\theta_{(i,m_1,m_2,k)} \leftarrow \left(\sum_p \theta_{(i,p,q,k)}\right)^{-1} \theta_{(i,m_1,m_2,k)}$
  foreach $(i, m_1, m_2, k) \in S_{k\rightarrow i}$ do
    Define a portion of $(A_g)_{k,i}$: $\delta = (A_g)_{k,i} \cdot \theta_{(i,m_1,m_2,k)}$
    Collapse $\delta$ onto the path $(i, m_1, m_2, k)$:
    $(A_c)_{m_1,i} \leftarrow (A_c)_{m_1,i} + \delta,$
    $(A_c)_{k,m_2} \leftarrow (A_c)_{k,m_2} + \delta,$
    $(A_c)_{m_1,m_1} \leftarrow (A_c)_{m_1,m_1} - \delta,$
    $(A_c)_{m_2,m_2} \leftarrow (A_c)_{m_2,m_2} - \delta,$
    $(A_c)_{m_2,m_1} \leftarrow (A_c)_{m_2,m_1} + \delta.$
  end
end
end

Algorithm 9: The Sparsening procedure
Remark 1 (Treatment of non-constant near null-space). The way Algorithm 9 is written in this work, it is suitable only for matrices that have the constant vector as a single near null-space prototype. The same approach can be generalized for matrices with a single non-constant near null-space by applying a diagonal scaling, similarly to the way it is applied for the restriction and prolongation in standard adaptive AMG [32, 123]. That is, let $x$ and $y$ be the (positive) right and left near null-space prototypes, respectively, and let $X = \text{diag}(x)$ and $Y = \text{diag}(y)$ be diagonal matrices whose diagonal elements are the entries in $x$ and $y$ respectively. Then the re-scaled matrix $YAX$ has a constant right and left near null-space prototype and can be treated by Algorithm 9. The case of multiple near null-space prototypes, as in the elasticity problem, requires a different treatment—see [110] for a discussion about this matter in the context of non-Galerkin coarsening.

Remark 2 (Computational cost). The computational cost of Algorithm 9 is comparable to the cost of the Galerkin product $R_sA_P$, because finding a distance 2 path between two variables is similar to applying an inner product between two sparse vectors. For example, the non-zeros of the product $R_sP_s$ correspond to all distance 2 paths in the first branch of Algorithm 9. In a way, the sparsening procedure tracks the three-term products of the Galerkin operator for sparsified entries and updates the entries of $A_c$ accordingly. Our computational savings come from the third level on, since then all matrices are much sparser than in the non-sparsified SA. We note that most of the sparsified non-zeros are treated as distance 2 paths which are cheaper to process.

The additional matrices $R_tP_s$, $R_sP_t$ and $A_t$ are calculated very fast since $R_t$ and $P_t$ are very simple—their calculation is much cheaper than calculating $R_s$, $P_s$, $R_sAP_s$, and applying Algorithm 9.

6.3.3 Theoretical results

In this section we state some of the theoretical properties of the sparsening procedure.

Proposition 1. If the fine matrix $A$ is symmetric, $R_t = P^T_t$, and $R_s = P^T_s$, then $A_c$ is symmetric as well.

Proof. By the given symmetries and (58), we get that $A_g$ and $A_t$ are symmetric, and so are their sparsity patterns. Now, if the entry $(k,i)$ needs to be eliminated, i.e., $(k,i) \in$
S\(p(A_g)\backslash S\(p(A_t)\), then the entry \((i, k)\) needs to be eliminated as well. Furthermore, if there is a path from \(i\) to \(k\) such that
\[
(P_s^T P_t)_{k,m_2} \neq 0, \quad (P_t^T P)_{m_1,i} \neq 0, \quad \text{and} \quad (A_t)_{m_2,m_1} \neq 0,
\] (73)
then the same path exists from \(k\) to \(i\) in the opposite direction (i.e., via \(m_2\) and \(m_1\)), because \((P_s^T P_t)^T = (P_t^T P_s)\) and \((A_t)\) is symmetric. Using this path, the submatrix (71) that corresponds to the elimination of \((A_g)_{i,k}\) is given by:
\[
\begin{pmatrix}
i & m_1 & m_2 & k \\
i & 0 & (A_g)_{i,k} & 0 & -(A_g)_{i,k} \\
0 & -(A_g)_{i,k} & (A_g)_{i,k} & 0 \\
0 & 0 & -(A_g)_{i,k} & (A_g)_{i,k} \\
k & 0 & 0 & 0 & 0
\end{pmatrix}.
\] (74)

If we set \((A_g)_{i,k} = (A_g)_{k,i}\) in (74) and in (71), we can see that they are the transpose of each other. This means, that the sum of (71) and (74) is a symmetric submatrix under these conditions. Furthermore, under the above symmetries, also each portion \(\theta_{(i,m_1,m_2,k)}\) equals \(\theta_{(k,m_2,m_1,i)}\) according to (72). Thus, when eliminating each portion of a pair \((i,k)\) and \((k,i)\), we add a symmetric submatrix to \(A_g\), and therefore \(A_c\) remains symmetric. 

In the next proposition, we consider diagonally dominant M-matrices. Recall that a matrix \(A\) is called an M-matrix if it is of the form of \(A = sI - B\), where \(B \geq 0\) and \(s \geq \rho(B)\) \((\rho(B) = \max_i \{|\lambda_i|\} \) is the spectral radius of \(B\). Furthermore, a matrix \(A\) is called diagonally dominant if every row \(i\) satisfies \(A_{i,i} \geq \sum_{j \neq i} |A_{i,j}|\).

**Proposition 2.** If \(A_g\) is a diagonally dominant M-matrix, then \(A_c\) is a diagonally dominant M-matrix as well.

**Proof.** Since \(A_g\) is a diagonally dominant M-matrix, then for every row \(k\):
\[
(A_g)_{k,k} \geq -\sum_{j \neq k} (A_g)_{k,j},
\]
or in matrix form: \((A_g)1 \geq 0\), where \(1\) is the vector of ones. By the sparsening construction in Algorithm 9, \((A_c)1 = (A_g)1 \geq 0\). Furthermore, any eliminated off-diagonal entry \((A_g)_{k,i}\)
is non-positive, so the submatrix (71) for replacing it has only non-positive off-diagonal entries, except the \((k,i)\) entry which cancels. Also, its diagonal entries are non-negative. This means that the sign-structure of \(A_c\) still corresponds to an M-matrix. This, together with \((A_c)1 \geq 0\) means that \((A_c)\) is diagonally dominant. Finally, by the Gerschgorin theorem, \(A_c\) is also positive definite and hence an M-matrix.

Our final observation deals with the energy preservation of our sparsification mechanism, and the spectral equivalence of the two operators: \(A_g\) and \(A_c\). It shows that the properties that were discussed in [17, 83, 137] are achieved also in our algorithm. Consider the following 5-point and 9-point 2D stencils that represent circulant matrices:

\[
H_1 = \begin{pmatrix} c & b & c \\ a & -2(a+b) - 4c & a \\ c & b & c \end{pmatrix}, \quad H_2 = \begin{pmatrix} b + 2c \\ a + 2c & -2(a+b) - 8c & a + 2c \\ b + 2c \end{pmatrix}. \tag{75}
\]

These two stencils were shown to be spectrally equivalent in [17]. The algorithm of [137] would find the entries of \(H_2\) given those of \(H_1\) by imposing that the response of both of them to the monomials \(\{1, x, y, x^2, y^2\}\) given by

\[
\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & 1 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 0 & 1 \\ -1 & -1 & -1 \\ 1 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \quad \tag{76}
\]

respectively, is equal. That is, the values of \(H_1\) and \(H_2\) times the stencils of each of those basis functions is equal.

Algorithm 9, similarly to the work of [83], would find the entries of \(H_2\) given those of \(H_1\) by imposing an equal response to the constant vector from right and from left. Consider the sparsification of the northwest (NW) entry of \(H_1\), which equals \(c\). Since \(H_2\) is a 5-pt stencil, the surrogate paths for elimination the NW entry are through the north entry (N), which equals \(b\) in \(H_1\), and the west entry (W) which equals \(a\) in \(H_1\). Each of these paths is a distance-two path. Furthermore, assume that the portions of both paths, \(\theta_{NW,N,C}\) and \(\theta_{NW,W,C}\), are both equal to 0.5 (\(C\) denotes the center). This would be the case if \(H_1\) is a Galerkin operator, and the coefficients in the prolongation \(P\) also correspond to a similar stencil coming from the fine-level matrix. In this situation, when eliminating the NW edge,
we add $\frac{1}{2}c$ to the N entry and to the W entry for each direction of this edge. Since the matrix that corresponds to $H_1$ is symmetric, this will add a full value of $c$ to both N and W entries. An additional $c$ will be added in a similar way when eliminating the other entries that relate to those paths (the SW and NE edges). Overall, for this correction, we add the following sub matrix for each path

$$
\begin{pmatrix}
0 & \delta & -\delta \\
\delta & -2\delta & \delta \\
-\delta & \delta & 0
\end{pmatrix}.
$$

This sub-matrix corresponds to the $[NW, N, C]$ entries or to the $[NW, W, C]$ entries, and in both options, $\delta = \frac{1}{2}c$. This matrix has a zero energy not only for the constant vector, but also for vectors in one of the two following subspaces:

$$
\begin{pmatrix}
NW \\
N/W \\
C
\end{pmatrix} : \text{Span} \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \right\} \text{ and Span } \left\{ \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \right\}.
$$

These subspaces can span all the functions in Equation (76) for the $[NW, N/W, C]$ entries, which implies the equivalence between the above algorithms. Similar equivalence is also evident in 3D for spectral equivalence between 7-pt and 27-pt stencils.

Let us now show some measures that demonstrate the spectral equivalence achieved by our algorithm (and all the algorithms that produce (75)). We refer again to the example in (56), and test two measures for this example on a $32 \times 32$ grid: one is (57), and the other is the operator in (57) multiplied by a smoothing operator. The multiplication by a smoothing operator aims (roughly speaking) to restrict the spectral equivalence measure to smooth modes. In Table 7 we show the measures for $A_g$ and $A_c$ in (56), and also for the pure aggregation operator $A_t$, which is equal to $2A_c$ in this case. The table shows that all measures are below 1, as is expected, given that the two-level cycles with either operator converge. However, we get better equivalence using $A_c$. Moreover, introducing the relaxation operators $S_J$ improves the equivalence measure of $A_c$ up to a value close to 0, whereas the measure for $A_t$ stagnates at 0.5. We note that 0.5 is indeed the convergence factor of the aggregation two level cycle for this example, and it deteriorates as more levels are used in a
V-cycle. Further investigation of the required characteristics for spectral equivalence in the context of non-Galerkin multigrid is a subject of our future research.

\[
\begin{array}{c|cc}
\|I - B^{-1}A_g\|_2 & 0.49887 & 0.74943 \\
\|(I - B^{-1}A_g)S_f\|_2 & 0.15815 & 0.49907 \\
\|(I - B^{-1}A_g)S_f^2\|_2 & 0.05611 & 0.49758 \\
\end{array}
\]

Table 7: Spectral equivalence measures. \(S_f\) denotes the damped Jacobi operator \((I - \frac{2}{3}D^{-1}A_g)\).

### 6.4 Numerical Results

In this section, the Sparsified Smoothed Aggregation (SpSA) method is compared to both Smoothed Aggregation (SA) and simple aggregation (AGG). We consider three groups of problems: non-symmetric 2D and 3D convection-diffusion problems, symmetric 2D and 3D diffusion problems with discontinuous coefficients, and symmetric 2D and 3D unstructured homogenous and non-homogenous graph-Laplacian problems on random graphs. For all problems, we compare the performance of the above algorithms as preconditioners to GMRES or PCG, and compare both iteration count and running time of our code. We consider only V-cycles in the work as it is the most suitable for parallel computations.

In the tables below, we present four measures for each run: ‘it’ denotes the number of V-cycles needed to reduce the initial residual by a factor of \(10^8\), starting with a zero initial guess. \(C_{op}\) denotes the operator complexity, which is the total number of non-zero elements in the operators \(A\) on all the grids, divided by that of the fine-level operator. Our coarsening is performed until \(n < 100\). ‘st’ denotes the maximal stencil size, that is, the maximal number of non-zeros in any row in the whole hierarchy of operators. The maximal (or average) stencil size reflects the amount of communication required in parallel computations [142, 110]. Lastly, ‘WU’ denotes the number of work-units required for the solution—for the setup and solution phases combined. Each work-unit is a matrix-vector multiplication and the values in the table are calculated by measuring the time needed for the solution and dividing it by the time of one fine-level matrix-vector multiplication. We note that such timings are highly implementation and machine dependent, but still, they provide some indication for comparing between the algorithms. Our code is MATLAB based with several procedures written in C using the \texttt{mex} environment. The experiments were performed using...
6. Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

Figure 17: 2D convection-diffusion problems: velocity fields.

MATLAB R2013b on a machine with an Intel core i7 quad-core CPU with 8 GB of RAM memory, running Windows 7.

6.4.1 Non-symmetric test cases: convection-diffusion

We consider the 2D and 3D convection-diffusion equation (54) on the unit square/cube with Dirichlet boundary conditions. The problem is discretized using the first order upwind finite differences method, leading to a five-point stencil on a discrete domain. We use several of the harder problems in [64, 97, 108]:

recirc:  \( \mathbf{v}^T = (x(1-x)(2y-1), -(2x-1)y(1-y)), \)

bent-pipe:  \( \mathbf{v}^T = (x(x-2)(1-2y), -4y(y-1)(1-x)), \)

2D-3:  \[
\mathbf{v}^T = \begin{cases} 
  \cos(2\pi x)\sin(2\pi y), & \text{if } x, y < 0.5, \\
  -\sin(2\pi x)\cos(2\pi y), & \text{otherwise} 
\end{cases}
\]

3D-1:  \( \mathbf{v}^T = (2x(1-x)(2y-1)z, (2x-1)y(y-1), (2x-1)(2y-1)(z-1)) \),

3D-2:  \( \mathbf{v}^T = (x(1-2y)(1-z), y(1-2z)(1-x), z(1-2x)(1-y)) \),

3D-3:  \( \mathbf{v}^T = (x(1-y)(2-z), y(1-z)(2-x), z(1-x)(2-y)) \),

and following [108] generate \( f \) in (54) so that the solution is given by \( u = \sin(\pi x)^2 + \sin(\pi y)^2 \) for 2D or \( u = \sin(\pi x)^2 + \sin(\pi y)^2 + \sin(\pi z)^2 \) for 3D. Figure 17 shows the 2D velocity fields \( \mathbf{v} \).

We use GMRES(10) acceleration, preconditioned with V-cycles. On the top level, we apply one pre-smoothing of forward Gauss-Seidel and one post-smoothing of backward Gauss-Seidel, and on coarser levels we apply one pre and post symmetric Gauss-Seidel smoothing.
Table 8: 2D convection-diffusion. ‘#it’: the number of V-cycles, \(C_{op}\): the operator complexity, ‘st’: the maximal stencil size in the hierarchy, ‘\(WU\)’: the work units count.
6. Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

Table 8 compares the three aggregation methods AGG, SA and SpSA for the 2D convection-diffusion (54) with the first three velocity fields in (79). Fields with ‘—’ show that convergence was not achieved in 100 iterations. It is clear that the AGG method is not mesh independent and not efficient, especially for the recirc problem. It also struggles for the diffusion-dominated problems \((\epsilon = 10^{-2} \text{ and } 2\text{D-3 that has a large diffusive domain for all } \epsilon)\). The SA method shows good scalability for all combinations. It has a higher operator complexity than the other two methods, especially in the more convective problems, where the coarsening becomes less aggressive. The SpSA obviously has the low and attractive operator complexity of AGG, but also enjoys the scalable convergence behavior of SA for both the diffusive and convective problems. The maximal stencil sizes, ‘st’, are similar between AGG and SpSA, and are higher in SA—as are the operator complexities. Since these are only 2D problems, the stencil growth of SA is moderate. In terms of the computing time of the algorithms, which is measured in work-units, ‘WU’, both SpSA and SA beat AGG even though the setup phase of AGG is about 2-3 times faster (not shown in the tables). SA and SpSA have surprisingly similar timings in our implementation, and in both cases the timing of the setup and solution phases are quite similar, each about half of the WUs that are shown in the table.

Table 9 shows the results for the 3D problems. Here, we see even larger operator complexities for SA than in 2D, especially for the convection-dominated problems \((\epsilon = 10^{-4}, 10^{-6})\). In most of these cases, we also see a severe stencil growth in SA, which reaches thousands of non-zeros in some cases. By changing the strength of connection parameters in (124) we may control the stencil growth, but at the same time impair the convergence rate of SA, which is rather good and scalable, as in the 2D case. Unlike in 2D, the AGG method shows moderate convergence, albeit not mesh-independent in several cases. Its convergence is expected to further deteriorate as the problem gets bigger. Again, ‘—’ denotes the cases where AGG failed to converge in less than 100 iterations, or it shows that the timings in WUs of the run are not relevant. This happens in the 192³ 3D problems, where the memory in our machine ran out in the setup phase and the execution was dominated by memory swapping. As in 2D, SpSA has the low operator complexity and maximal stencil sizes of AGG, and convergence similar to SA. In terms of work-units, all algorithms are competitive with some advantage to SpSA in the majority of the cases. In 3D, the setup of AGG is about 3-5 times faster than the setup of SA and SpSA, because of the stencil growth, and the need to multiply
<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$n$</th>
<th>$10^{-2}$</th>
<th>$10^{-4}$</th>
<th>$10^{-6}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{AGG}$</td>
<td>$\text{SpSA}$</td>
<td>$\text{SA}$</td>
<td>$\text{AGG}$</td>
<td>$\text{SpSA}$</td>
</tr>
<tr>
<td>$\text{it}$</td>
<td>$C_{\text{op}}$</td>
<td>$\text{st}$</td>
<td>$\text{WU}$</td>
<td>$\text{it}$</td>
</tr>
<tr>
<td>$64^3$</td>
<td>39</td>
<td>1.3</td>
<td>28</td>
<td>456</td>
</tr>
<tr>
<td>$96^3$</td>
<td>49</td>
<td>1.3</td>
<td>35</td>
<td>561</td>
</tr>
<tr>
<td>$128^3$</td>
<td>59</td>
<td>1.3</td>
<td>30</td>
<td>626</td>
</tr>
<tr>
<td>$192^3$</td>
<td>75</td>
<td>1.3</td>
<td>32</td>
<td>—</td>
</tr>
<tr>
<td>$64^4$</td>
<td>20</td>
<td>1.5</td>
<td>51</td>
<td>316</td>
</tr>
<tr>
<td>$96^4$</td>
<td>28</td>
<td>1.5</td>
<td>64</td>
<td>409</td>
</tr>
<tr>
<td>$128^4$</td>
<td>35</td>
<td>1.5</td>
<td>49</td>
<td>465</td>
</tr>
<tr>
<td>$192^4$</td>
<td>48</td>
<td>1.5</td>
<td>53</td>
<td>—</td>
</tr>
<tr>
<td>$64^6$</td>
<td>19</td>
<td>1.5</td>
<td>67</td>
<td>313</td>
</tr>
<tr>
<td>$96^6$</td>
<td>21</td>
<td>1.5</td>
<td>83</td>
<td>338</td>
</tr>
<tr>
<td>$128^6$</td>
<td>23</td>
<td>1.5</td>
<td>81</td>
<td>327</td>
</tr>
<tr>
<td>$192^6$</td>
<td>27</td>
<td>1.5</td>
<td>106</td>
<td>—</td>
</tr>
<tr>
<td>$64^4$</td>
<td>32</td>
<td>1.5</td>
<td>55</td>
<td>475</td>
</tr>
<tr>
<td>$96^4$</td>
<td>40</td>
<td>1.5</td>
<td>61</td>
<td>552</td>
</tr>
<tr>
<td>$128^4$</td>
<td>48</td>
<td>1.5</td>
<td>67</td>
<td>616</td>
</tr>
<tr>
<td>$192^4$</td>
<td>70</td>
<td>1.5</td>
<td>65</td>
<td>—</td>
</tr>
<tr>
<td>$64^6$</td>
<td>37</td>
<td>1.5</td>
<td>29</td>
<td>515</td>
</tr>
<tr>
<td>$96^6$</td>
<td>37</td>
<td>1.5</td>
<td>29</td>
<td>515</td>
</tr>
<tr>
<td>$128^6$</td>
<td>77</td>
<td>1.5</td>
<td>101</td>
<td>938</td>
</tr>
<tr>
<td>$192^6$</td>
<td>91</td>
<td>1.5</td>
<td>103</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 9: Convection diffusion 3D. ‘$\#it$’ the number of V cycles, $C_{\text{op}}$: the operator complexity, ‘$\text{st}$’: the maximal stencil size in the hierarchy, ‘$\text{WU}$’: the work units measures.
6. Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

Figure 18: 2D diffusion problems with discontinuous coefficients: square, diamond, and L shapes.

much denser matrices, in the case of SA. In SpSA, more non-zeros need to be eliminated using more surrogate paths. In most cases where the operator complexity is high, the setup cost of SA and SpSA dominated their execution time. Other than that, the SpSA method again seems to be the most efficient. Although the timings of SA and SpSA are similar in our serial code, we expect to see advantage for SpSA in parallel settings, especially in the solution phase.

6.4.2 Diffusion with discontinuous coefficients

In the next group of test cases we consider the two- and three-dimensional diffusion equation on the unit square/cube, Ω, with Dirichlet boundary conditions

\[ \nabla \cdot (\kappa \nabla u) = f, \quad \text{in } \Omega \]
\[ u = g, \quad \text{on } \partial \Omega. \]  

The diffusion problem is discretized by the vertex-centered finite differences method, leading to a five-point stencil on a discrete domain Ω_\text{h} with regular mesh size.

We consider three classical test cases of coefficient inhomogeneities, including jumps that are aligned with the grid lines and jumps that are not aligned (all the problems appear in
<table>
<thead>
<tr>
<th>2D-tests:</th>
<th>AGG+</th>
<th>SpSA</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>κ</td>
<td>n</td>
<td>it</td>
<td>C_{op}</td>
</tr>
<tr>
<td>256²</td>
<td>37</td>
<td>1.3</td>
<td>10</td>
</tr>
<tr>
<td>512²</td>
<td>60</td>
<td>1.3</td>
<td>10</td>
</tr>
<tr>
<td>1024²</td>
<td>72</td>
<td>1.3</td>
<td>11</td>
</tr>
<tr>
<td>2048²</td>
<td>13</td>
<td>1.4</td>
<td>31</td>
</tr>
<tr>
<td>512²</td>
<td>34</td>
<td>1.3</td>
<td>9</td>
</tr>
<tr>
<td>1024²</td>
<td>57</td>
<td>1.3</td>
<td>9</td>
</tr>
<tr>
<td>2048²</td>
<td>86</td>
<td>1.3</td>
<td>10</td>
</tr>
<tr>
<td>1024²</td>
<td>13</td>
<td>1.4</td>
<td>37</td>
</tr>
<tr>
<td>2048²</td>
<td>35</td>
<td>1.3</td>
<td>10</td>
</tr>
<tr>
<td>512²</td>
<td>55</td>
<td>1.3</td>
<td>10</td>
</tr>
<tr>
<td>1024²</td>
<td>49</td>
<td>1.3</td>
<td>10</td>
</tr>
<tr>
<td>2048²</td>
<td>72</td>
<td>1.3</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3D-tests:</th>
<th>AGG+</th>
<th>SpSA</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>κ</td>
<td>n</td>
<td>it</td>
<td>C_{op}</td>
</tr>
<tr>
<td>64³</td>
<td>33</td>
<td>1.3</td>
<td>31</td>
</tr>
<tr>
<td>96³</td>
<td>35</td>
<td>1.3</td>
<td>32</td>
</tr>
<tr>
<td>128³</td>
<td>46</td>
<td>1.3</td>
<td>31</td>
</tr>
<tr>
<td>192³</td>
<td>59</td>
<td>1.3</td>
<td>34</td>
</tr>
<tr>
<td>64³</td>
<td>30</td>
<td>1.3</td>
<td>29</td>
</tr>
<tr>
<td>96³</td>
<td>55</td>
<td>1.3</td>
<td>33</td>
</tr>
<tr>
<td>128³</td>
<td>49</td>
<td>1.3</td>
<td>32</td>
</tr>
<tr>
<td>192³</td>
<td>92</td>
<td>1.3</td>
<td>32</td>
</tr>
<tr>
<td>64³</td>
<td>23</td>
<td>1.3</td>
<td>29</td>
</tr>
<tr>
<td>96³</td>
<td>29</td>
<td>1.3</td>
<td>32</td>
</tr>
<tr>
<td>128³</td>
<td>38</td>
<td>1.3</td>
<td>34</td>
</tr>
<tr>
<td>192³</td>
<td>45</td>
<td>1.3</td>
<td>34</td>
</tr>
</tbody>
</table>

Table 10: 2D and 3D diffusion problems on structured meshes. ‘#it’ denotes the number of V cycles, C_{op} is the operator complexity, ‘st’ is the maximal stencil size, and ‘WU’ is the work units measures.
Figure 18 shows the three choices of coefficients which are explicitly given by

\begin{align}
\begin{cases}
\kappa(x) = \frac{1}{4} \quad &\text{if } \|x - \frac{1}{2}\|_\infty < \frac{1}{4} \\
1 &\text{otherwise}
\end{cases} &\quad \blacklozenge
\quad \begin{cases}
\kappa(x) = \frac{1}{\sqrt{8}} \quad &\text{if } \|x - \frac{1}{2}\|_1 < \frac{1}{\sqrt{8}} \\
1 &\text{otherwise}
\end{cases} &\quad \diamondsuit
\quad \begin{cases}
L \kappa(x) = \frac{1}{4} \quad &\text{if } \|x\|_\infty < \frac{1}{2} \\
1 &\text{otherwise}
\end{cases}
\end{align}

The resulting linear systems are symmetric.

We apply PCG, preconditioned using V-cycles with one pre- and post- Symmetric Gauss Seidel relaxations for all methods on all levels. Because the plain aggregation method, AGG, is known to struggle with such problems, we accelerate it using a multilevel overcorrection acceleration. That is, in Algorithm 1 we interpolate \( e = P_s e_c \), apply post relaxations for \( Ae = r \), and then calculate \[ x \leftarrow x + \alpha e, \quad \text{such that } \alpha = \arg \min_\alpha \|x + \alpha e\|_A = \frac{r^T e}{e^T A e}. \]

Using this procedure we would typically get \( \alpha > 1 \). This overcorrection technique may significantly accelerate the convergence of the multilevel AGG. We note that although this acceleration may be very efficient in serial code, it imposes an expensive communication overhead in parallel settings due to the inner products that are performed.

Table 10 summarizes the results for this group of problems. Because of the multilevel accelerations, the AGG+ method converged moderately fast, but again, its convergence is not mesh-independent. Again, ‘—’ denotes failure to converge, or irrelevant timings in WUs. Like before, we see the best iteration counts in SA, with SpSA needing a bit more iterations. In terms of timings in WUs, SpSA and SA are comparable and generally better than AGG+, although the setup of AGG+ is much cheaper, like in the previous examples.

For the problems considered, SA has a rather low operator complexity, and it does not introduce a severe stencil growth. Its maximal stencil size is only in the low hundreds, also in 3D. As before, AGG+ and SpSA have similar low complexity, and maximal stencil size.

### 6.4.3 Random graph-Laplacian problems

In the next group of test cases we consider a graph-Laplacian problem on a 2D/3D random graph \( G(V, E) \). We generate our graph by first generating \( n \) random points on the unit
square/cube as the nodes $V$, and applying a Delaunay triangulation to generate edges in $E$. Such graphs were first suggested in [65]. A 2D example of such graph is shown in figure 19. Given the edges $(i, j) \in E$ we create a matrix $A$ which is defined by

$$\forall (i, j) \in E : A_{i,j} = -\Theta_{ij} \quad \text{and} \quad A_{ii} = -\sum_{i \neq j} A_{ij},$$

and if we set $\Theta_{ij} = 1$, it represents the homogenous graph-Laplacian operator on $G$. In the inhomogenous graph-Laplacian case, we set a weight $\Theta_{ij} > 0$ as the coefficient for each edge $(i, j)$. The weights of the graph are assumed to be symmetric, i.e $\Theta_{ij} = \Theta_{ji}$, and so is the associated matrix. We use random weights $\Theta_{i,j} = 10^3 u$ where $u \in (0, 1)$ is a random uniformly distributed positive number, so that $\Theta_{i,j} \in (1, 10^3)$, but obey a logarithmic distribution. This is a much harder problem than choosing the $\Theta$'s uniformly, because there are many weak connections in the matrix. We note that these problems have significantly larger stencils than the structured problems in the previous sections, and also, their stencil sizes are not uniform. That is, there may be points with significantly more neighbors than other. Therefore, we needed to change the aggregation procedure, as explained in the Appendix B. In these examples, we could not generate the 3D $192^3$ problem because it exceeded the memory of our machine. We also note that this problem is singular, with the constant vector as a true null-space. To handle that, we apply a pseudo-inverse on the coarsest grid, and filter out the constant from the solution. See [123] for more discussion on that regard.

Table 11 summarizes the results for this group of problems, where $\text{rand}$ denotes the
6. Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

### 2D-tests:

<table>
<thead>
<tr>
<th>Θ</th>
<th>n</th>
<th>AGG+</th>
<th>SpSA</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2562</td>
<td>24</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>5122</td>
<td>29</td>
<td>1.2</td>
<td>15</td>
<td>1.4</td>
</tr>
<tr>
<td>10242</td>
<td>29</td>
<td>1.2</td>
<td>17</td>
<td>1.1</td>
</tr>
<tr>
<td>20482</td>
<td>33</td>
<td>1.2</td>
<td>20</td>
<td>1.1</td>
</tr>
</tbody>
</table>

### rand

<table>
<thead>
<tr>
<th>Θ</th>
<th>n</th>
<th>AGG+</th>
<th>SpSA</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2562</td>
<td>25</td>
<td>1.4</td>
<td>17</td>
<td>1.4</td>
</tr>
<tr>
<td>5122</td>
<td>33</td>
<td>1.4</td>
<td>18</td>
<td>1.4</td>
</tr>
<tr>
<td>10242</td>
<td>51</td>
<td>1.4</td>
<td>21</td>
<td>1.4</td>
</tr>
<tr>
<td>20482</td>
<td>37</td>
<td>1.4</td>
<td>22</td>
<td>1.4</td>
</tr>
</tbody>
</table>

### 3D-tests:

<table>
<thead>
<tr>
<th>Θ</th>
<th>n</th>
<th>AGG+</th>
<th>SpSA</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>643</td>
<td>18</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>963</td>
<td>19</td>
<td>1.1</td>
<td>11</td>
<td>1.1</td>
</tr>
<tr>
<td>1283</td>
<td>20</td>
<td>1.1</td>
<td>11</td>
<td>1.1</td>
</tr>
</tbody>
</table>

### rand

<table>
<thead>
<tr>
<th>Θ</th>
<th>n</th>
<th>AGG+</th>
<th>SpSA</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>643</td>
<td>15</td>
<td>1.4</td>
<td>10</td>
<td>1.3</td>
</tr>
<tr>
<td>963</td>
<td>16</td>
<td>1.4</td>
<td>10</td>
<td>1.3</td>
</tr>
<tr>
<td>1283</td>
<td>17</td>
<td>1.4</td>
<td>10</td>
<td>1.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Θ</th>
<th>n</th>
<th>AGG+</th>
<th>SpSA</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2562</td>
<td>24</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>5122</td>
<td>29</td>
<td>1.2</td>
<td>15</td>
<td>1.4</td>
</tr>
<tr>
<td>10242</td>
<td>29</td>
<td>1.2</td>
<td>17</td>
<td>1.1</td>
</tr>
<tr>
<td>20482</td>
<td>33</td>
<td>1.2</td>
<td>20</td>
<td>1.1</td>
</tr>
</tbody>
</table>

### rand

<table>
<thead>
<tr>
<th>Θ</th>
<th>n</th>
<th>AGG+</th>
<th>SpSA</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2562</td>
<td>25</td>
<td>1.4</td>
<td>17</td>
<td>1.4</td>
</tr>
<tr>
<td>5122</td>
<td>33</td>
<td>1.4</td>
<td>18</td>
<td>1.4</td>
</tr>
<tr>
<td>10242</td>
<td>51</td>
<td>1.4</td>
<td>21</td>
<td>1.4</td>
</tr>
<tr>
<td>20482</td>
<td>37</td>
<td>1.4</td>
<td>22</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Table 11: Unstructured Graph-Laplacian.

In this section we provide measures for the communication cost in applying a V-cycle with SA and SpSA. We use a simplified model for measuring communication, taken from [3]. In this model, the communication cost is broken down into the start-up time $\alpha$ (latency) and the per-element send time $\beta$ (inverse bandwidth). If a message has $m$ elements,
then the send cost is

\[ T_{\text{send}} = \alpha + \beta m. \]

The constants \( \alpha \) and \( \beta \) are machine dependent, and so we will focus on the factors that multiply them. Therefore, following the descriptions of the parallel AMG implementation in [72], we define two measures of “communication complexity” as follows. Let \( p \) be the number of available computing nodes. Define a partitioning of each matrix \( A \) in the multigrid hierarchy into \( p \) clusters of variables, with each cluster designated to a computing node. Each such node holds the corresponding rows in the matrices and the corresponding values of the iterate \( \mathbf{x}^k \). We then define the communication required for applying a matrix-vector multiplication (MAT-VEC) on each level. The latency measure is the number of non-zero off-diagonal blocks in the matrix with the blocks defined according to the cluster partitioning. It does not matter how many non-zeros there are in an off-diagonal block—it is the existence of at least one non-zero in \( A \) in that off-diagonal block that forces establishing communication between the two associated nodes when applying a MAT-VEC (the corresponding entries of \( \mathbf{x}^k \) need to be transferred). More precisely, assume that \( T \) is a matrix that corresponds to the \( p \) clusters just as the tentative matrix (24) corresponds to the \( n_c \) aggregates; for a given multigrid hierarchy we define:

\[
\text{Latency} = \sum_{\ell=0}^{L} \frac{(\text{nnz}(T_\ell^T A_\ell T_\ell) - p)}{(\text{nnz}(T_0^T A_0 T_0) - p)},
\]

where \( \ell \) denotes the level, the matrices \( T_\ell \) and \( A_\ell \) are the clustering matrix and operator on level \( \ell \), respectively, \( p \) is the number of clusters and \( \text{nnz}(\cdot) \) is the number of non-zeros. Next, we define the bandwidth that is needed for applying a MAT-VEC on each level—that is, the total number of values of \( \mathbf{x}^k \) that are needed to be sent between nodes. This corresponds to the number of non-zero columns in the off-diagonal blocks according to the cluster partitioning. As in (84), we divide this number by the required bandwidth on the finest level. We preform the partitioning for each level using a graph partitioning software METIS [81] which aims at forming clusters such that the number of neighbors between the clusters is minimal. We use \( p = 100 \) clusters, and we measure the communication complexity only for levels with \( n \geq 500 \).

Table 12 summarizes the communication measures for eight sample hierarchies from the previous sections. It shows that in 2D, the latency measures of both SA and SpSA are quite
6. Non-Galerkin Multigrid: Sparsified Smoothed Aggregation (SpSA)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Name</th>
<th>n</th>
<th>Parameters</th>
<th>SA</th>
<th>SpSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td></td>
<td></td>
<td></td>
<td>Latency</td>
<td>Bandwidth</td>
</tr>
<tr>
<td>2D Conv-Diff</td>
<td>1024²</td>
<td>recirc, $\epsilon = 10^{-6}$</td>
<td>5.98</td>
<td>3.02</td>
<td>5.26</td>
</tr>
<tr>
<td>3D Conv-Diff</td>
<td>96³</td>
<td>3D-k, $\epsilon = 10^{-6}$</td>
<td>16.00</td>
<td>3.48</td>
<td>6.19</td>
</tr>
<tr>
<td>2D Diffusion</td>
<td>1024²</td>
<td>$\kappa = \square$</td>
<td>4.19</td>
<td>1.91</td>
<td>3.99</td>
</tr>
<tr>
<td>3D Diffusion</td>
<td>96³</td>
<td>$\kappa = \square$</td>
<td>10.08</td>
<td>1.82</td>
<td>4.39</td>
</tr>
<tr>
<td>2D RandGraph</td>
<td>1024²</td>
<td>$\Theta = 1$</td>
<td>5.22</td>
<td>2.69</td>
<td>4.00</td>
</tr>
<tr>
<td>3D RandGraph</td>
<td>96³</td>
<td>$\Theta = 1$</td>
<td>6.42</td>
<td>1.81</td>
<td>3.00</td>
</tr>
<tr>
<td>2D RandGraph</td>
<td>1024²</td>
<td>$\Theta = \text{rand}$</td>
<td>7.05</td>
<td>4.79</td>
<td>5.00</td>
</tr>
<tr>
<td>3D RandGraph</td>
<td>96³</td>
<td>$\Theta = \text{rand}$</td>
<td>12.93</td>
<td>4.70</td>
<td>4.15</td>
</tr>
</tbody>
</table>

Table 12: Communication complexity

comparable, and are mostly dictated by the number of levels, which is equal for both algorithms in our tests. In 3D, on the other hand, there is a huge increase in the latency of SA, which is mostly due to the coarser levels. For SpSA, the latency measure is mostly equal to the number of levels, which is what we expect to achieve in this algorithm. In the bandwidth measures we see comparable complexities for diffusion and homogenous RandomGraph problems. This is also correlated with the similar operator complexity measures in the previous sections. A bigger difference is evident in the other problems, where we also saw an increase in the operator complexity of SA in the previous sections. In the bandwidth measures, we do not see a big difference between 2D and 3D problems (remember that the measures are relative to bandwidth on the finest grid). Obviously, SpSA required less communication in all the cases.

6.5 Conclusions

In this work we have presented a new algebraic multigrid algorithm where the choice of the sparsity pattern of the coarse operators is independent of the choice of the high-quality transfer operators. This property makes the algorithm particularly worthwhile for parallel settings.

The new algorithm uses the well-known aggregation framework, adopting simple non-smoothed aggregation for determining the sparsity pattern of the coarse operators, and smoothed aggregation for high-quality transfer operators. It sparsifies the smoothed aggregation coarse operators onto the simple aggregation sparsity patterns. Numerical experiments show that the algorithm has promising capabilities for 2D and 3D convection-diffusion problems, diffusion problems with varying coefficients and unstructured graph-Laplacian prob-
lems. It seems scalable and robust and may be advantageous in cases where strict sparsity constraints prevent us from using high-quality Galerkin operators, as in parallel settings.
7 A Multilevel Approach for $l_1$ Penalized Least Squares (LASSO)

In this section we describe our second study on multilevel approaches for sparsity regularized optimization. Here we focus on the quadratic case: $l_1$ penalized least squares (LASSO). This section is based on the following paper:


The LASSO problem is quite different from the classical problems of linear systems or eigenproblems, which are the usual focus of multigrid methods. Hence, in this research we follow different principles from those of traditional AMG solvers, requiring a great deal of investigation and analysis.

7.1 Introduction

Sparse approximation of signals is an emerging area of research that is drawing vast interest and finding use in numerous applications. One popular application is sparse representation of signals and images, where the key underlying observation is that natural signals, such as images, admit sparse decompositions over specific spatial transforms [35, 52]. Another popular application is known as compressive sensing [49, 37, 36] where signals are reconstructed from only a few linear measurements. Other applications include statistical analysis, machine learning, and coding theory [128].

There has been an enormous effort in recent years to develop mathematical formulations and computational methods for applying such reconstructions. The simplest way to mathematically formulate this idea is to assume that the sought signal $\mathbf{y} \in \mathbb{R}^n$ can be approximately represented by only a few columns of a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$. That is, $\mathbf{y} = \mathbf{A}\mathbf{x}$, where the representation vector $\mathbf{x} \in \mathbb{R}^m$ is sparse, containing few non-zero elements. The matrix $\mathbf{A}$, often called the dictionary, is usually over-complete, having more columns than rows, $m > n$. This means that the underdetermined system $\mathbf{A}\mathbf{x} = \mathbf{y}$ has infinitely many
solutions, and we seek the sparsest one by solving the problem

$$\min_{x \in \mathbb{R}^m} \|x\|_0 \quad \text{subject to} \quad Ax = y,$$

(85)

where the sparseness measure \(\|x\|_0 = |\{i : x_i \neq 0\}|\) is called the \(l_0\) quasi-norm, defined as the number of non-zero elements in the vector \(x\). There are alternative formulations based on the \(l_0\) quasi-norm. However, these optimization problems are non-convex and generally very hard to solve, as their solution usually requires an intractable combinatorial search [45]. Nevertheless, the solution for such problems can be approximated using so-called “greedy algorithms” such as (Orthogonal) Matching Pursuit (OMP/MP) [89, 102, 125, 127, 12], Stagewise OMP (StOMP) [48], CoSAMP [93], Subspace Pursuit (SP) [43], iterative hard thresholding [13, 14, 15, 11, 73, 38], and others.

A common alternative approach is to relax (85) by replacing the \(l_0\) quasi-norm with the well-known \(l_1\) norm, which has somewhat similar “sparsity properties” [49, 36, 126, 51]. The new problem,

$$\min_{x \in \mathbb{R}^m} \|x\|_1 \quad \text{subject to} \quad Ax = y,$$

(86)

called Basis Pursuit [40], is convex. Its solution may not be unique, but if more than one solution exists then all the solutions belong to a convex and compact set; that is, any convex combination of the basic solutions is itself a solution [51]. A typical solution vector \(x^*\) of (86) is relatively sparse, and under certain conditions it is in fact equal to a global minimizer of (85). Problem (86) can be formulated and solved as a linear programming problem [40].

Because observed signals typically contain some noise, which has no sparse representation, the constraint \(Ax = y\) is usually relaxed in both (85) and (86), with approximate equality measured using the quadratic penalty function \(\|Ax - y\|_2^2\), where \(y\) henceforth denotes the observed noisy signal. Two other approaches to treat this case are LASSO [121], and the common Basis Pursuit denoising (BPDN) [40]. The latter features an \(l_1\) penalized least-squares functional minimization:

$$\min_{x \in \mathbb{R}^m} F(x) = \min_{x \in \mathbb{R}^m} \frac{1}{2}\|Ax - y\|_2^2 + \mu\|x\|_1,$$

(87)

with \(\mu\) a scalar parameter that balances between sparsity and adherence to the data. Generally, a larger parameter \(\mu\) yields a sparser minimizer \(x^*\), but also a greater discrepancy
\[ ||Ax^* - y||^2_2. \] Although this problem is an unconstrained convex optimization problem, traditional optimization methods, such as gradient descent or quasi-Newton methods, tend to be slow due to the discontinuity of the gradient, which arises from using the \( l_1 \) norm. Therefore, various computational optimization methods were developed for the task. The most common methods are the so-called “iterative shrinkage” or “iterative soft thresholding (IST)” methods that are often used together with some accelerations [57, 77, 44, 58, 50, 53, 9, 70, 136, 147]. Other related approaches for solving (87) include [59, 146, 138, 18, 60, 129, 109]. Similarly to (86), problem (87) may also have more than one globally optimal solution. In this work we adopt the common practice of seeking any one of those solutions and refer to it as “the minimizer” of (87), denoted by \( x^* \).

This work introduces a straightforward multilevel method for \( l_1 \) penalized least-squares problems like (87), based on the main concept of classical algebraic multigrid methods [33]; that is, we accelerate the convergence of simple iterative methods for (87) using a nested hierarchy of smaller versions of the problem. Multigrid methods are commonly applied to linear systems arising from discretization of partial differential equations as well as other ill-conditioned systems. In many cases algebraic multigrid methods enable us to treat such problems effectively regardless of their condition number. This is done by projecting the original problem to a lower-dimensional subspace that contains the error components that are not treated effectively by standard iterative methods, such as Jacobi and Gauss-Seidel. Then, these error components are corrected by solving a lower-dimensional problem. This correction together with the standard iterative methods serve as two complementary processes which combine to yield an effective solver. The idea of introducing multigrid-like methods for (87) has yet to be explored and it has great potential. In this work we follow the idea of “multiplicative correction” multigrid methods, which exploit a hierarchy of approximate operators that evolve with the solution process, eventually becoming exact—see [23, 122] and references therein.

In classical multigrid methods, the aim is to define a multilevel solver with optimal asymptotic convergence behavior, because the asymptotic convergence rates of simpler iterative solvers tend to be slow for problems of interest. However, the present problem is different as the main challenge is finding the non-zero elements of the minimizer \( x^* \) and its sign-pattern. Therefore, our algorithm is different from the classical multigrid approach. At each iteration (called a “multilevel V-cycle”) it reduces the dimension of the problem and
creates a multilevel hierarchy of smaller and smaller problems, involving a lower dimensional dictionary at each “level”. We take advantage of the typical sparsity of $x$ and reduce the dimension of the problem (87) by ignoring ostensibly irrelevant columns from $A$. That is, each low-level problem is defined by (87), restricted to a specially chosen subset of the columns of $A$, resulting in a nested hierarchy of sub-dictionaries. It then performs sub-space correcting shrinkage sweeps over each of the low dimensional problems in turn, that aim to activate the atoms that comprise the support of a true minimizer. Under suitable conditions, our algorithm converges to the global minimizer of (87)—we do not compromise solution quality in return for improved performance.

### 7.2 Iterated shrinkage methods and accelerations

Many of the simple iterated shrinkage methods for solving (87) are of the form

$$z = S_{\mu/c} \left( \frac{1}{c} A^T (y - Ax^k) + x^k \right),$$  \hspace{1cm} (88)

$$x^{k+1} = x^k + \alpha (z - x^k),$$  \hspace{1cm} (89)

where $x^k$ is the approximate solution at the $k$-th iteration, $\alpha > 0$ is a line-search scalar, $c > 0$ is method dependent, and

$$S_q(t) = \text{sign}(t) \cdot \max(0, |t| - q)$$  \hspace{1cm} (90)

is the “soft shrinkage” function, so dubbed because the size of the argument $t$ is reduced by $q$ (or set to zero if $q > |t|$). Some methods, such as [44, 138], simply choose $\alpha = 1$ in (89), i.e., $x^{k+1} = z$. Others, such as [53, 136], apply a line-search by choosing $\alpha$ that minimizes the functional (87) of $x^{k+1}$ according to the search direction (89).

The constant $c > 0$ in (88) can be chosen in different ways and is the main difference between the various shrinkage methods. For example, in SSF [44], $c$ is chosen such that $c \geq \rho(A^T A)$; in SpaRSA [138] $c$ is recalculated at each iteration via a linesearch over $F(z)$ defined in (87); in [136] $c$ is adapted throughout the iterations, and in PCD [53] $1/c$ is replaced by the inverse of a diagonal matrix $D$ which is equal to the diagonal of $A^T A$.

The iterative shrinkage methods mentioned above are much faster than standard first-
order minimization methods such as classical Steepest Descent, however, for many problems they are still slow, requiring acceleration. In [53, 147] an optimization method called “sequential subspace optimization” (SESOP) was considered, together with the PCD and SSF iterations. Other acceleration methods include FISTA [6], TwIST [9], and a version of non-linear Conjugate Gradients (CG) [147]. In this work, we use a non-linear CG approach that is slightly different from that of [147], and is also based on the Polak-Ribiere CG method [94]. Given a search direction \( r_k = z - x_k \) obtained in (89) at the \( k \)-th iteration, we compute \( x_{k+1} \) via a line-search along the direction \( r_k = r_k + \beta r_k \), where

\[
\beta_{PR} = \max \left\{ \frac{(r^k)^T (r^k - r_{k-1})}{\|r_{k-1}\|^2}, 0 \right\}.
\]

We have found the Polak-Ribiere non-linear CG method to be the most efficient method which is based on matrix-vector multiplication (with the diagonal of \( A^T A \) as preconditioner, i.e., CG-PCD). A precise description of this method appears in Algorithm 10.

**Algorithm 10:** Diagonally Preconditioned Non-linear Conjugate Gradient for LASSO

Another type of acceleration technique that can be used with the above methods is the “continuation” strategy, also known as “warm-start”, that is applied in [70, 136, 59, 138]. In this approach, a large parameter \( \mu \) is first chosen for the problem (87) (lower than but proportional to \( \|A^T y\|_{\infty} \)). Once (87) is solved approximately for this value of \( \mu \), it is gradually decreased, and at each stage the new initial guess is given by the approximate solution to
(87) obtained with the previous, bigger, parameter $\mu$. This way, a sequence of problems is solved corresponding to the sequence of the decreasing $\mu$’s, until some convergence criterion is satisfied.

Asymptotically, once the sign-pattern of the minimizer $x^*$ is recovered and fixed by the iterations, the functional in (87) becomes quadratic and can normally be solved efficiently either by the CG method or directly by solving the system that corresponds only to the non-zeros of the vector $x^*$. Furthermore, in most applications the $l_1$ norm is only used as a regularizer for promoting sparsity of $x$, and so, once we determine the support of $x^*$, we may ignore the regularization term in (87) and only consider the quadratic term [59, 138] (an approach known as “debiasing”). Overall, the main effort in solving (87) is invested in determining the non-zero elements of the minimizer $x^*$ and their signs.

### 7.3 A multilevel iterated shrinkage approach

We next describe our new multilevel approach for solving (87). At each iteration, called a “V-cycle”, we define a hierarchy of reduced problems, referred to as low-level problems. Each low-level problem is defined by (87), restricted to a specially chosen subset of the columns of $A$, and in each V-cycle we traverse the entire hierarchy of levels. We iteratively repeat these V-cycles, reducing the functional of (87) at each one, until some convergence criterion is satisfied. A precise description is given in the following sections in a two-level framework, with the extension to the multi-level framework obtained by recursion. In this description, all elements that are related to the low-level problem are denoted by a subscript $c$. Keeping with common practice, we use the terms “matrix” and “dictionary” interchangeably, and similarly the terms “column” and “atom”.

#### 7.3.1 Definition of the low-level problem

In this subsection we define the reduced problem given its designated subset of atoms, $C \subset \{1, ..., m\}$, while the choice of $C$ will be discussed later. Given $C$, we define a so-called prolongation matrix $P \in \mathbb{R}^{m \times |C|}$, that transfers a low-level vector $x_c \in \mathbb{R}^{|C|}$ into an upper-level vector $x \in \mathbb{R}^m$ by the relation $x = P x_c$. We choose $P$ to be a zero-filling operator, which zeros the elements of $x$ that do not belong to $C$, while retaining the values of $x_c$ in the elements that do belong to $C$. We have found this simple approach to be effective, but
more sophisticated choices of \( P \) may be worthy of investigation for more general problems.

Next, we restrict (87) onto the atoms in \( C \), or more generally, onto the range of \( P \). That is, we substitute \( Px_c \) for \( x \) in the objective (87), and get the new problem:

\[
\min_{x_c \in \mathbb{R}^{|C|}} F_c(x_c) \equiv \min_{x_c \in \mathbb{R}^{|C|}} F(Px_c) = \\
\min_{x_c \in \mathbb{R}^{|C|}} \frac{1}{2} \|APx_c - y\|_2^2 + \mu \|Px_c\|_1,
\]

which has only \(|C|\) degrees of freedom. Since our \( P \) is zero-filling, we have that \( \|Px_c\|_1 = \|x_c\|_1 \) holds for all \( x_c \), and therefore we can write

\[
\min_{x \in \mathbb{R}^{|C|}} F_c(x_c) = \min_{x_c \in \mathbb{R}^{|C|}} \frac{1}{2} \|A_c x_c - y\|_2^2 + \mu \|x_c\|_1,
\]

where \( A_c = AP \) is the reduced sub-dictionary of the upper-level dictionary \( A \), with columns given by the columns of \( A \) corresponding to the indices in \( C \). Note that if \( C \) contain the support of the true minimizer of (87), and (92) is solved exactly, then \( Px_c \) is in fact a solution of (87). Furthermore, because this problem is similar to (87), we can recursively extend this two-level framework to multi levels.

### 7.3.2 Choosing the low-level variables

Our definition of the low-level problem (92) suggests that we need to select a subset of low-level variables, \( C \), that is as likely as possible to contain the support of the true minimizer. Therefore, for choosing \( C \) we use the approximate solution at the \( k \)-th iteration, \( x^k \), which is the best one currently available. Let

\[
supp(x) = \{i : x_i \neq 0\},
\]

denote the support of any vector \( x \). Then evidently, if \( supp(x^k) \subseteq C \), then \( x^k \) is in the range of \( P \). Indeed, \( x^k = Px_c \) where \( x_c \) is the vector \( x^k \) restricted to the indices in \( C \). Therefore, we start by requiring \( supp(x^k) \subseteq C \), so that by (91)-(92) we satisfy \( F(x^k) = F_c(x_c) \). This implies that the prolongation matrix \( P \) changes during the iterations, depending on \( x^k \).

Next, we decide on the additional atoms in \( C \), besides those in \( supp(x^k) \), aiming to limit its size to \(|C| = \lceil m/2 \rceil \) (this choice will be discussed later). If \(|supp(x^k)| \geq \lceil m/2 \rceil \), then
we choose $\mathcal{C} = \text{supp}(x^k)$. Otherwise (the common case) we add $\lceil m/2 \rceil - |\text{supp}(x^k)|$ atoms that are currently not in $\text{supp}(x^k)$, and yet have a relatively good chance of being in the support of the true solution $x^*$. These correspond to atoms $i$ with a relatively large value of $|a^T_i (Ax^k - y)|$, since including them in the support reduces the first term in the functional of \eqref{eq:87} more significantly per given increase in the second term (see also Proposition 7 below). This rationale is commonly employed in existing “greedy algorithms” mentioned earlier (OMP, MP etc.). This leads to the following definition of $\mathcal{C}$ at the $k$-th iteration:

$$
\mathcal{C} = \text{supp}(x^k) \cup \text{likely}(\kappa),
$$

where $\kappa = \max\{\lceil m/2 \rceil - |\text{supp}(x^k)|, 0\}$, likely(0) = $\emptyset$, and likely($\kappa$) is the set of indices of the $\kappa$ largest elements in the vector $|A^T (Ax^k - y)|$.

### 7.3.3 Definition of the multilevel V-cycle

For solving \eqref{eq:87}, we repeat

$$
x^{k+1} = \text{V-cycle}(A, x^k, y, \nu)
$$

iteratively, until some convergence criterion is satisfied. The multilevel V-cycle() procedure, along with its parameters, is defined in Algorithm 11. The algorithm creates a reduced version of the problem \eqref{eq:87} as described above, and then treats it recursively, yielding a hierarchy of smaller and smaller problems. The recursion is terminated (Step 3a) when one of the following happens. The common base case is when $|\text{supp}(x)| \geq \lceil m/2 \rceil$—then we cannot reduce the problem further. In this case we choose $\mathcal{C} = \text{supp}(x)$, and solve the problem \eqref{eq:92} directly. The second base case is where the problem becomes sufficiently small and can be solved easily. In practice, we choose a minimal number of allowable columns $m_{\text{min}}$, and if $|\mathcal{C}| < 2m_{\text{min}}$ then we process \eqref{eq:92} directly rather than continuing recursively. (In our tests we use $m_{\text{min}} = 10$.)

The algorithm uses iterated shrinkage methods as so-called “relaxations”—the usual name for the iterations employed within multilevel algorithms—carrying out $\nu$ such relaxations at each level ($\nu = 1$ in our tests). All shrinkage methods of the form \eqref{eq:88}, as well as most other shrinkage methods, can be incorporated into this multilevel approach. Figure 20 illustrates Algorithm 11.

In terms of cost, if we reduce the number of unknowns by a factor of two at each level
Algorithm: \( \mathbf{x} \leftarrow \text{V-cycle}(A, \mathbf{x}, y, \nu) \)

\%Iterative Shrinkage method: \( \text{Relax}(A, \mathbf{x}, y) \).
\%Number of relaxations at each level: \( \nu \).
\%Minimal number of columns allowed: \( m_{\text{min}} \).

1. Choose the low-level variables \( \mathcal{C} \) and define the prolongation \( P \).
2. Define the low-level dictionary \( A_c \) and approximation \( x_c = P^T x \).
3. If \( \mathcal{C} = \text{supp}(x) \) or \( |\mathcal{C}| < 2m_{\text{min}} \),
   (a) Solve the lowest-level problem (92).
Else \( x_c \leftarrow \text{V-cycle}(A_c, x_c, y, \nu) \) % Recursive call
4. Prolong solution: \( \mathbf{x} \leftarrow Px_c \) % Solution update.
5. Apply \( \nu \) relaxations: \( \mathbf{x} \leftarrow \text{Relax}(A, \mathbf{x}, y) \).

Algorithm 11: V-cycle for \( l_1 \) penalized LS minimization

Figure 20: Illustration of the multilevel V-cycle for \( l_1 \) penalized LS minimization.
(|\mathcal{C}| = \lceil m/2 \rceil), then the total cost of a V-cycle (excluding the treatment of the lowest level) is only about twice the cost of \( \nu \) iterated shrinkage relaxations on the highest level. This means that, although we include a relatively large fraction of the atoms in the next-lower level, the cost of the entire V-cycle remains relatively small, possibly excluding the lowest level solution. The latter is a key component of our algorithm, and it will be discussed later.

**Remark 3.** As one can see, our algorithm can be readily applied when the columns of \( A \) can be easily extracted. This requires \( A \) to be given explicitly as a matrix, rather than a fast transform operator as in the Wavelet or DCT transforms. Such explicit dictionaries are mostly used in the context of “trained dictionaries”, where they may be either sparse or dense; see [99, 85, 82, 1, 103, 104] and references therein. Furthermore, in many applications the dictionary \( A \) takes the form

\[
A = H \cdot B,
\]

where \( H \) is an operator that acts on the columns of the matrix \( B \) [147]. For example, in the image deblurring problem the operator \( H \) is a low-pass filter while \( B \) is the underlying dictionary. In this scenario, our algorithm is applicable if either \( H \) or \( B \) are given as explicit matrices, and the other is cheap to apply. In most cases, it is the matrix \( B \) which is explicit.

**Remark 4.** Algorithm 11 is a multilevel framework wrapped around a chosen iterative shrinkage (soft thresholding) method. In spirit, it bears some relation to accelerated iterative hard thresholding [11, 38] and subspace pursuit [93, 43] methods. However, the objectives, the techniques, and the available theoretical observations, are quite different. At the \( k \)-th iteration, these algorithms update the solution by minimizing \( || A x^k - y ||_2 \) with respect to a small subspace restricting the support of \( x^k \). This is similar to the lowest-level solution in Step 3a of our V-cycle, but with a different objective. In [93, 43], a sparse approximation with a small, fixed, support size is sought, and the selection of the atoms is done according to the largest elements in \( A^T (A x^k - y) \). In [11, 38], the selection of atoms is done by iterative hard thresholding.

In our approach we target \( l_1 \) penalized problems, as opposed to \( l_0 \) penalized problems in [11, 38] and support-size constrained problems in [93, 43]. We use the likelihood criterion \( A^T (A x^k - y) \) only to create a hierarchy of subspaces (low-level dictionaries), while the iterated shrinkage (relaxation) method is responsible for selecting the atoms assumed to belong to the support. We elaborate on this mechanism in the next section. Unlike these methods, the
subspaces in our hierarchy are very large (until we get to the lowest level) and the problem we target at each level is non-quadratic, and is analogous to the fine-level problem.

7.4 Theoretical properties of the multilevel V-cycle

This section is devoted to theoretical performance observations regarding Algorithm 11. In some cases we refer to its two-level version, which differs from the original algorithm only in Step 3. There, the recursive call is replaced with an exact solution of the problem (92) on the second level, that is, the condition in Step 3 is removed and Step 3a is always performed.

In the following discussions we use the iteration specific notation $x^k$ and the generic notation $x$, depending on the context. Both relate to an approximate solution to (87). Also, as before, we denote a solution to the problem (87) by $x^*$, and assume that it is a stationary point of any relaxation method.

By (91), together with $\text{supp}(x^k) \subseteq \mathcal{C}$, we can write

\[ F_c(x_c) = F(Px_c) = F_c(P^T x^k) = F(x^k), \tag{95} \]

from which the next two observations follow immediately.

**Proposition 3.** (Inter-level Correspondence.) Let $x$ be an approximation to the minimizer of $F$ in (87). Let $z_c$ be a better approximation than $x_c$ to the minimizer of the low-level problem (92), such that $F_c(z_c) < F_c(x_c)$. Then $F(Pz_c) < F(x)$.

**Proposition 4.** (Direct Solution.) If $\mathcal{C} \supseteq \text{supp}(x^*)$, then $x_c^* = P^T x^*$ is a solution of the low-level problem (92), and the two-level Algorithm 11 solves problem (87) in one V-cycle.

Equation (95) relates the original problem (87) to any of the problems (92) on any level of the V-cycle. Let the original problem (87) define level 0, and let $P_i$ be a prolongation from level $i$ to level $i - 1$. Then (95) holds for $F_c(x_c)$ on any level $i$ together with the so-called “composite prolongation”,

\[ P_i^c = P_1 \cdot P_2 \cdots P_i, \tag{96} \]
that transfers from level $i$ to level 0. Using $P^c_i$, Proposition 3 holds for any level $i$ of the V-cycle. The next corollary is an extension of Proposition 4 to the multilevel case, and its proof follows immediately from Proposition 4 using the composite prolongation (96).

**Corollary 1.** If $\text{supp}(x) \supseteq \text{supp}(x^*)$, then Algorithm 11 solves problem (87) in one V-cycle. Furthermore, $x^*$ is a stationary point of Algorithm 11.

For the following propositions we use the notion of sub-gradients [51]. $\partial F(x)$, the sub-differential of $F$, is a non-empty set of sub-gradients,

$$\partial F(x) = \{ A^T(Ax - y) + \mu z : z \in A(x) \},$$

where $A(x)$ is a set of all vectors $z \in \mathbb{R}^m$ whose elements satisfy

$$z_i = \text{sign}(x_i) \quad \text{if } x_i \neq 0,$$

$$z_i \in [-1, 1] \quad \text{if } x_i = 0.$$  

A vector $x^*$ is a minimizer of (87), if and only if $0 \in \partial F(x^*)$. That is, for all elements of $x^*$, if $x_i^* \neq 0$ then $a^T_j(Ax^* - y) + \mu \text{sign}(x^*_j) = 0$, and otherwise $|a^T_j(Ax^* - y)| \leq \mu$. Using this, we next show that the two-level algorithm does not stagnate.

**Proposition 5.** (No Stagnation.) If $C \nsubseteq \text{supp}(x^*)$, and $x = Px_c$ is the updated upper-level solution after Step 4 of the two-level Algorithm 11, then a single Iterated Shrinkage relaxation as in (88) on $x$ must cause at least one atom to be added to $\text{supp}(x)$.

**Proof.** Since $x_c$ is a minimizer of the low-level functional, then $0 \in \partial F_c(x_c)$, and since $A_c$ is comprised of a subset of the columns of $A$, and $Ax = APx_c = A_cx_c$, then $0 \in \partial F_c$ means that for all $j \in C$

$$a^T_j(Ax - y) + \mu \text{sign}(x_j) = 0 \quad \text{if } x_j \neq 0,$$

$$|a^T_j(Ax - y)| \leq \mu \quad \text{if } x_j = 0.$$  

Now, since $C \nsubseteq \text{supp}(x^*)$, then $x$ is not a minimizer of the upper-level functional, so $0 \notin \partial F(x)$. Therefore, there exists at least one variable $\ell \notin C$ for which $|a^T_j(Ax - y)| > \mu$. Since $x_\ell = 0$, then, according to (88), after one Iterated Shrinkage relaxation, index $\ell$ will enter the support of $x$. 

From the last two propositions we can see the complementary roles of the relaxation and solution update in Algorithm 1. The relaxation is largely responsible for inserting the correct atoms into the support, while the solution update is mainly responsible for finding the optimal values of the variables that are in the support.

For the next proposition we define the term “memory-less monotonic iteration” (MLMI). An iterated shrinkage relaxation $T(x)$ is called MLMI if

$$F(x) - F(T(x)) \geq K \cdot \min \limits_{z \in A(x)} (\|\partial F(x)\|_2^2) \quad \forall x \in \mathbb{R}^m,$$

where $K$ is a positive constant and $\partial F(x)$ is the sub-differential of $F$ defined in (97) and (98). This definition implies that every iteration of $T(x)$ reduces the functional $F(x)$ in (87) at worst proportionally to the size of $\min_{z \in A(x)}(\|\partial F(x)\|_2^2)$. For example, one such method is SSF [44], which was shown to be both MLMI and convergent under some conditions (see Appendix B in [53]). We next prove an auxiliary Lemma that will help us to show that Algorithm 11 is convergent under suitable conditions.

**Lemma 1.** (Monotonicity.) Assume that Algorithm 11 is used together with $\nu > 0$ MLMI relaxations $T$, and is applied on $x$. Let $Q(x)$ be the solution update in Step 4 of Algorithm 11 on level 0. Then $F(x) \geq F(Q(x))$.

**Proof.** The proof follows from Equation (95), which holds for any level $i$ via the corresponding composite prolongation (96). Since an exact minimization is performed on the lowest level, and since the relaxations that are used on each level are MLMI, the values of consecutive low-level functionals $F_c$ cannot increase as we traverse the V-cycle hierarchy. \hfill \Box

We now show that Algorithm 11 is convergent under suitable conditions, following the idea of the convergence proof of SSF in [53].

**Proposition 6.** (Convergence.) Assume that the level set $\mathcal{R} = \{x : F(x) \leq F(x^0)\}$ is compact. Also, assume that Algorithm 11 is applied with $\nu > 0$ MLMI relaxations $T$. Let $\{x^k\}$ be a series of points produced by Algorithm 1, i.e., $x^{k+1} = V$-cycle$(A, x^k, y, \nu)$, starting from an initial guess $x^0$. Then any limit point $x^*$ of the sequence $\{x^k\}$ is a stationary point of $F$ in (87), i.e., $0 \in \partial F(x^*)$, and $F(x^k)$ converges to $F(x^*)$. 
Proof. We start by showing that the series \( \{ F(x^k) \} \) is monotonically decreasing. By Lemma 1 we have that \( F(x^k) \geq F(Q(x^k)) \) where \( Q(x^k) \) is the solution update on level 0. Then, following the algorithm, we apply \( \nu \) relaxations on \( Q(x^k) \). By (100), and \( \nu > 0 \), we can bound

\[
F(x^k) - F(x^{k+1}) \geq F(x^k) - F(T(Q(x^k))) \\
\geq F(Q(x^k)) - F(T(Q(x^k))) \\
\geq K \min_{z \in A(Q(x^k))} \| \partial F(Q(x^k)) \|^2,
\]

which implies that \( \{ F(x^k) \} \) is monotonically decreasing.

Since the functional \( F \) in (87) is non-negative, then it is bounded from below, and hence the series \( \{ F(x^k) \} \) converges to a limit. Because the level set \( \mathcal{R} \) is compact by assumption, we have that \( \{ x^k \} \) is bounded in \( \mathcal{R} \), and therefore there exists a sub-series \( \{ x^{k_n} \} \) converging to a limit point \( x^* \).

Assume to the contrary that \( x^* \) is not stationary, i.e., \( 0 \notin \partial F(x^*) \). It is known that \( \partial F(x) \) is always a non-empty convex compact set, which means that \( \min_{x \in A(x^*)} \| \partial F(x) \| \) always exists. That, together with \( 0 \notin \partial F(x^*) \) ensures that \( \min_{x \in A(x^*)} \| \partial F(x^*) \| > 0 \). Denote \( \min_{x \in A(x^*)} \| \partial F(x^*) \| = \epsilon \). Since \( x^{k_n} \to x^* \), then there are infinitely many \( k_n \)'s satisfying \( \min_{x \in A(x^{k_n})} \| \partial F(x^{k_n}) \| = \epsilon \). By (101) we have infinitely many \( k_n \)'s satisfying

\[
F(x^{k_n}) - F(x^{k_{n+1}}) \geq K \cdot \epsilon^2,
\]

which contradicts the fact that \( F \) is bounded from below. This shows that the point \( x^* \) is stationary. Since \( F \) is continuous, \( x^{k_n} \to x^* \) yields \( F(x^{k_n}) \to F(x^*) \). The limit of \( \{ F(x^k) \} \) is equal to that of any of its sub-series, specifically \( \{ F(x^{k_n}) \} \), and thus \( F(x^k) \to F(x^*) \).

The next proposition justifies our criterion for the choice of the set \( C \) of atoms comprising the low level. This result is in the spirit of thresholding algorithms, as well as other greedy algorithms, which apply the same approach for selecting the support [51].

**Proposition 7.** (C-Selection Guarantee.) Without loss of generality, assume that the columns of \( A \) are normalized such that \( \text{diag}(A^T A) = 1 \), and then let \( \delta \) be the so-called mutual coherence of the dictionary \( A \), defined by

\[
\delta = \max_{i \neq j} \{|a_i^T a_j|\}.
\]
Let $x^*$ and $x$ be the solution and the approximate solution at the time $C$ is chosen, respectively, and let $e = x^* - x$ be the current error. Let $i$ be an index satisfying $i \in \text{supp}(x^*)$ and $i \notin \text{supp}(x)$. Then, so long as $|C| > |\text{supp}(x)|$, index $i$ is guaranteed to be included in the set $C$ prior to any index $\ell \notin \text{supp}(x) \cup \text{supp}(x^*)$ if

$$|x_i^*| \geq \frac{2\delta}{1 + \delta} \|e\|_1. \quad (103)$$

Proof. The choice of $C$ is based on the likelihood measure. We derive a condition that guarantees that index $i$ be included in $C$ prior to any index $\ell$ not belonging to $\text{supp}(x^*)$, i.e.,

$$|a_i^T(Ax - y)| > |a_\ell^T(Ax - y)| \quad (104)$$

for any $\ell \notin \text{supp}(x) \cup \text{supp}(x^*)$. To this end, we bound the left-hand side of (104) from below and the right-hand side from above, obtaining the condition (103).

Since $i \in \text{supp}(x^*)$ and $0 \in \partial F(x^*)$, then $a_i^T(Ax^* - y) + \mu \text{sign}(x_i^*)$ is equal to zero and may be subtracted from or added to the left-hand side of (104) yielding

$$|a_i^T(Ax - y)| = |a_i^T A e + \mu \text{sign}(x_i^*)| = \left| \sum_{j \neq i} (a_i^T a_j) e_j + e_i + \mu \text{sign}(x_i^*) \right|,$$  

where we have used the fact that the dictionary columns are normalized, $a_i^T a_i = 1$. Observe that $e_i = x_i^*$ because $i \notin \text{supp}(x)$. Using this, the triangle inequality $|a + b| \geq |a| - |b|$, the fact that $x_i^*$ and $\mu \text{sign}(x_i^*)$ have the same sign, and the definition of mutual coherence, we obtain

$$|a_i^T(Ax - y)| \geq |x_i^*| + \delta \sum_{j \neq i} |e_j| = |x_i^*(1 + \delta) + \mu - \delta \|e\|_1. \quad (106)$$

Next, we bound from above the right hand side of (104). Since $\ell \notin \text{supp}(x^*)$, there exists a scalar $z \in [-1, 1]$ for which $a_\ell^T(Ax^* - y) + \mu z$ is equal to zero and may be subtracted from or added to the right-hand side. Therefore,

$$|a_\ell^T(Ax - y)| = |a_\ell^T A e + \mu z| \leq \delta \|e\|_1 + \mu, \quad (107)$$

where the last inequality uses $e_\ell = 0$. Comparing the bounds (106) and (107), we obtain
that \( \mu \) drops out and the condition (104) is guaranteed if (103) is satisfied.

This proposition implies that if the dictionary \( A \) is far from being degenerate, i.e., \( \delta \ll 1 \), then, as our approximate solution gets better, any atom that contributes significantly to the solution is guaranteed to be chosen to the low-level set \( C \).

### 7.4.1 Treatment of the lowest level

One key component of our algorithm is the treatment of the lowest-level problem, whose dimension is typically the size of the current support. Assuming that in the lowest level of a V-cycle, \( \mathbf{x}_c \) is dense and of length \( |C| \), the cost of the shrinkage iterations (relaxations) discussed above is \( O(n|C|) \) operations each. Therefore, applying many relaxations for solving the lowest-level problem may be costly. Moreover, although the solution \( \mathbf{x}^* \) is normally sparse, there is no such guarantee for \( \mathbf{x} \) throughout the entire solution process. An exact solution might be wasteful if \( \mathbf{x} \) is too dense or if \( C \nsubseteq \text{supp}(\mathbf{x}^*) \). We note that generally our algorithm aims to retain sparse vectors \( \mathbf{x} \) throughout the solution process, but this property is not guaranteed.

Following the above reasoning, we limit the number of relaxations done on the lowest level to balance between cost and efficiency of the V-cycle. We aim to apply relaxations only until we increase the accuracy of \( \mathbf{x}_c \) by some order of magnitude compared to the initial \( \mathbf{x}_c \) on that lowest level. More specifically, we apply relaxations until the value of the expression \( \|\mathbf{x}_c - S_{\mu}(\mathbf{x}_c + A^T(\mathbf{y} - A\mathbf{x}_c))\| \) becomes 10 times smaller than it is initially on the lowest level. We found that this ratio balances well between the cost and efficiency of the V-cycles.

In addition, we explicitly limit the number of iterations proportionally to the cost ratio between a highest level relaxation and the lowest level relaxation, i.e., \( \frac{m}{|C|} \). Thus, the cost of the lowest-level solution will not exceed the cost of several high-level relaxations. In our tests, we apply at most \( 5\lceil \frac{m}{|C|} \rceil \) relaxations which cost roughly the same as 10 high level relaxations.

### 7.4.2 A gradual initialization—“full multilevel cycle”

Our strategy aims at limiting the support of \( \mathbf{x} \) throughout the solution process, thus saving unnecessary computations. The key problem is that if we initialize the solution process with \( \mathbf{x} = 0 \), and do a shrinkage iteration of the form (88), then many atoms will enter the support because of the rather large residual. This might happen with most shrinkage methods. Our
way to prevent this from happening is to start our solution process from the bottom, instead of the top. That is, initialize \( x = 0 \), choose a small set of atoms that are most likely to be in \( \text{supp}(x^*) \) (as in section 7.3.2), and solve the reduced problem (92). Then, gradually enlarge the relevant set of columns, and apply a V-cycle for each level. This strategy, together with the exact solution of the lowest-level problem, is expected to maintain a sparse \( x \) throughout the solution process. Figure 21 and Algorithm 12 describe this approach. A similar strategy is used in multigrid methods and is called a “full multigrid algorithm” (FMG) [33]. The input parameters for Algorithm 12 are identical to those of Algorithm 11.

**Algorithm:** F-cycle\((A, x, y, \nu)\)

1. Choose \( C = \text{likely}([m/2]) \) and define \( P \).
2. Define \( A_c = AP \) and restrict \( x_c = P^T x \).
3. If \(|C| < 2m_{\min}\), solve the problem (92).
   Else \( x_c \leftarrow \text{F-cycle}\(A_c, x_c, y, \nu\) % Recursive call \)
   \( x_c \leftarrow \text{V-cycle}\(A_c, x_c, y, \nu\) % Algorithm 11 \)
4. Prolong solution: \( x \leftarrow Px_c \).
5. Apply \( \nu \) relaxations: \( x \leftarrow \text{Relax}(A, x, y) \).

**Algorithm 12:** Full multilevel cycle initialization.

Figure 21: Full multilevel cycle initialization.

\( \downarrow \) refers to choosing \( C \) and reducing the problem, \( \Box \) refers to performing a lowest-level solve, \( \rightarrow \) refers to prolonging the solution, and \( \bullet \) refers to applying \( \nu \) relaxations.
7.4.3 A note on implementation

Algorithm 11 is presented in a rather symbolic way. In practice, we do not explicitly construct either the prolongations $P$ or the low-level operators $A_c$. Instead, we use a single vector $x$ and at low levels we address only the indices of $x$ belonging to that level. This saves the need to store and extract the dynamic hierarchy of low-level matrices which usually changes from one V-cycle to the next. This also lets us apply multiple-point methods (such as SESOP, CG and TwIST) on consecutive approximations from different levels (i.e., while traversing up the V-cycle). This approach, however, requires the ability to efficiently multiply a vector by $A_T^c$ using $A$ and the subset $C$ (in $n|C|$ operations). Similarly, since $x$ is sparse, for all the methods we compute $Ax$ while ignoring the zero entries of $x$ (i.e., in $n\|x\|_0$ operations). In MATLAB, for example, the latter can be achieved by storing $x$ in a sparse data structure, but the former needs to be programmed manually.

In our multilevel algorithm, we do not perform any significant computations besides the relaxations and lowest-level solution. Most importantly, we do not calculate the likelihood vector $A^T(Ax - y)$ for choosing $C$. Instead, we use the likelihood vector that corresponds to $x$ before the last relaxation, which is calculated inside that relaxation as a by-product. We assume that the slight change in the likelihood vector as a result of the relaxations does not justify the extra computational cost.

7.5 Numerical Results

In this section we compare the performance of several known shrinkage methods to our multilevel framework. We run the shrinkage iterations until the method-independent condition,

$$\|x - S_\mu(A^T(y - Ax))\|/\|x\| < 10^{-5},$$

(108)

is satisfied. This condition is taken from [87] and is related to the size of $\min_{x \in A(x)} \{\|\partial F(x)\|\}$. We do not require extreme accuracy because our aim is only to discover the support of the minimizer for the reason described in Section 7.2. In practice, all solutions achieved by all algorithms in our tests correspond to functional values which are identical up to several significant digits and have an essentially identical support size, which justifies this convergence criterion. Once this convergence criterion is satisfied, a debiasing phase is performed, where the norm $\|Ax - y\|_2$ is minimized with respect only to the atoms in the support of the
minimizer of (87).

The “one-level” methods that we show in our comparison include Coordinate Descent (CD) [18, 60]; Parallel Coordinate Descent (PCD) [50, 53, 147]; acceleration of PCD by SESOP [53, 147] and PCD-CG as described above (denoted by SESOP1 and CG-PCD, respectively); we also test GPSR-Basic (GPSR) and GPSR-BB (non-monotone) [59], TwIST [9] and SpaRSA (monotone, BB variant) [138].

For PCD, CG-PCD, and SESOP, we used the exact linesearch procedure of [136]. We also accelerated CD by a linesearch, seeking a parameter $\alpha \geq 1$ as in (89). We denote this method by CD$^+$. Within our multilevel framework we use CD$^+$ and CG-PCD. These are used both as relaxations and lowest-level solvers, and are denoted by ML-CD and ML-CG respectively. In some cases, we show our multilevel framework with CD$^+$ as a relaxation and CG-PCD as a low-level solver, and denote this option by ML-CD/CG.

For the methods of GPSR, SpaRSA, and TwIST, we used the default parameters suggested in the original papers and adapted the authors’ MATLAB software for our tests. All the rest of the methods are implemented in MATLAB with certain procedures in the mex framework, including the linesearch, CD iteration, $Ax$ and $A^Tc$ multiplications. The matrix-vector multiplications in mex were also parallelized using the OpenMP library, which enhanced their calculation timings so that they are comparable to MATLAB’s internal parallelization. Our parallel implementation of CD was not as fruitful because CD is sequential.

We perform four synthetic experiments using dense explicit dictionaries. In each of these we first generate a random, normally distributed dictionary,

$$A \in \mathbb{R}^{n \times m} \text{ s.t } a_{ij} \sim N(0,1),$$

and normalize its columns. Following this, we uniformly generate a support $S_0$ from all possible supports of size $[0.1n]$. Then, a vector $s_0$ is generated with normally distributed values in the indices corresponding to $S_0$ and zeros elsewhere and a clean signal is generated by $As_0$, normalized such that $\|As_0\|_\infty = 1$. Next, we add random Gaussian noise obtaining the noisy signal,

$$y = As_0 + n,$$

with $n \sim N(0,\sigma^2I)$. In all our experiments we set the noise level to $\sigma = 0.02$. Because
our experiments are randomly generated, each one is repeated 15 times, and the results are averaged and rounded. As in [87], our aim is only to examine the cost of computing the solution of (87), and not to measure the relation of its minimizer to the original sparse vector $s_0$ used to generate $y$.

The description above relates to our first experiment. A similar matrix is used in [87, 147], denoted by $K^{(2)}$. In the second experiment, taken from [88], the elements of $A$ in (109) are replaced by $\pm 1$ according to their signs, and the columns are normalized.

In the third experiment, we manipulate the singular values of $A$ in (109) to make it highly ill-conditioned, with a condition-number of about $10^{10}$. As before, we finalize the generation of the dictionary by normalizing its columns. A similar example is used in [87, 147], denoted by $K^{(4)}$. Figure 22 shows an example of the singular values of a dictionary $A$.

![Figure 22: The singular values of the matrix $A$ with $n = 512$](image)

In the fourth experiment we use a matrix that is similar to the $K^{(3)}$ matrix in [87]. The aim is to construct a matrix that is problematic for our solvers, even though it is well-conditioned. The construction is as follows: Let $U_1 \Sigma_1 V_1^T$ be the SVD decomposition of $A$ in (109) (after normalizing its columns). Let $A_1$ be the sub-matrix of $A$ comprised of its first $\lceil 0.55m \rceil$ columns. Let $v$ be the last column of $A_1$. Define: $A_2 = [A_1 | v \mathbf{1}^T + 0.05B]$, where $\mathbf{1}$ is a vector of all ones, and $B$ is a random matrix of size $n \times (m - \lceil 0.55m \rceil)$ like (109), with normalized columns. Let $U_2 \Sigma_2 V_2^T$ be the SVD decomposition of $A_2$, and then define the final matrix $A$ for this experiment as $U_2 \Sigma_1 V_2^T$. This matrix has the favorable spectrum of (109) and the problematic singular vectors of $A_2$.

In the tables below, each test is described by two numbers: the mean number of iterations—relaxations or V-cycles—and (in brackets) the number of work-units. Each work-unit stands for $mn$ floating point multiplications. We show these work-units in order to compare the multilevel method, that requires only a few albeit expensive cycles, with the shrinkage meth-
ods, that require many cheap iterations. We count all the operations performed in all the algorithms. We note that the two most significant operations are

1. Multiplying $A^T y$—costs $mn$ operations, (a single work-unit). Multiplying $A_c^T y$ costs $n|C|$ operations.

2. Multiplying $Ax$—costs $n\|x\|_0$ with $\|x\|_0$ as in (85). Similarly, multiplying $A_c x_c$ costs $n\|x_c\|_0$ operations.

For some of the runs (with the largest $n = 2048$) we also show the average solution time measured by MATLAB’s tic and toc mechanism. We note, however, that such timings depend on many factors that are not algorithmic (MATLAB compilation and parallel procedures, cache usage, memory usage, indexing etc.). The experiments were performed using MATLAB R2011b on a machine with an Intel core i7 quad-core CPU with 8 GB of RAM memory, running Windows 7.

The tables contain three parts. The upper section shows the influence of $n$, the middle part shows the influence of $m$, and the lower part shows the influence of $\mu$. We use multilevel V-cycles with only one relaxation applied in each level ($\nu = 1$). The multilevel process is initialized with a single full multilevel cycle (Algorithm 12) starting from $x = 0$. The one-level methods (excluding CD+) are initially accelerated by the warm-start (or continuation) strategy of [138] with parameter $\xi = 0.3$. The tables also show the support size of the minimizer, $\|x^*\|_0$, and improvement in signal-to-noise ratio defined as

$$\text{ISNR} = 10 \log_{10} \left( \frac{\|A s_0 - y\|_2^2}{\|A s_0 - A x^*\|_2^2} \right),$$

that was achieved after the debiasing phase. Here, $A s_0$ is the clean signal used in (110) to generate $y$. The higher this value is, the better is the denoised reconstruction. As noted above, the $\|x^*\|_0$ and ISNR measures are both almost identical for all the convergent algorithms.

### 7.5.1 Experiment 1: Well-conditioned random $A$

Table 13 summarizes the results for the first problem. Because $A$ is such that every one of its sub-matrices is well-conditioned, we find that all methods perform relatively well.
7. A Multilevel Approach for $l_1$ Penalized Least Squares (LASSO)

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>$\mu$</th>
<th>$|x|_0$</th>
<th>ISNR</th>
<th>CD$^+$</th>
<th>PCD</th>
<th>CG-PCD</th>
<th>ML-CG</th>
<th>SpaRSA</th>
<th>GPSR-BB</th>
<th>TwiST</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>4n</td>
<td>4\sigma</td>
<td>35.2</td>
<td>5.2</td>
<td>17.1(23)</td>
<td>2.5(17)</td>
<td>44.3(49)</td>
<td>36.8(43)</td>
<td>2.9(21)</td>
<td>36.8(43)</td>
<td>41.5(48)</td>
</tr>
<tr>
<td>512</td>
<td>4n</td>
<td>4\sigma</td>
<td>70.3</td>
<td>5.4</td>
<td>16.8(23)</td>
<td>2.6(17)</td>
<td>43.5(49)</td>
<td>34.6(42)</td>
<td>2.9(21)</td>
<td>37.3(43)</td>
<td>40.7(47)</td>
</tr>
<tr>
<td>1024</td>
<td>4n</td>
<td>4\sigma</td>
<td>139.8</td>
<td>5.4</td>
<td>16.8(25)</td>
<td>2.7(20)</td>
<td>43.5(51)</td>
<td>36.7(46)</td>
<td>3.0(24)</td>
<td>37.0(45)</td>
<td>40.2(49)</td>
</tr>
<tr>
<td>2048</td>
<td>4n</td>
<td>4\sigma</td>
<td>279.0</td>
<td>5.2</td>
<td>16.5(29)</td>
<td>2.9(24)</td>
<td>43.3(56)</td>
<td>35.7(49)</td>
<td>3.0(28)</td>
<td>37.4(50)</td>
<td>40.2(54)</td>
</tr>
<tr>
<td>1024</td>
<td>2n</td>
<td>4\sigma</td>
<td>117.9</td>
<td>6.3</td>
<td>13.6(24)</td>
<td>2.8(23)</td>
<td>36.9(47)</td>
<td>32.3(44)</td>
<td>3.0(26)</td>
<td>31.7(42)</td>
<td>32.9(43)</td>
</tr>
<tr>
<td>1024</td>
<td>6n</td>
<td>5\sigma</td>
<td>125.5</td>
<td>4.8</td>
<td>16.2(20)</td>
<td>2.3(14)</td>
<td>40.7(45)</td>
<td>33.8(40)</td>
<td>3.0(18)</td>
<td>35.7(41)</td>
<td>38.7(44)</td>
</tr>
<tr>
<td>1024</td>
<td>8n</td>
<td>5\sigma</td>
<td>144.9</td>
<td>4.6</td>
<td>18.4(22)</td>
<td>2.6(15)</td>
<td>46.9(51)</td>
<td>40.3(46)</td>
<td>3.0(19)</td>
<td>41.6(47)</td>
<td>46.9(54)</td>
</tr>
<tr>
<td>1024</td>
<td>4n</td>
<td>5\sigma</td>
<td>118.5</td>
<td>5.2</td>
<td>15.1(20)</td>
<td>2.6(17)</td>
<td>39.7(45)</td>
<td>32.3(39)</td>
<td>2.9(20)</td>
<td>34.3(41)</td>
<td>36.5(43)</td>
</tr>
<tr>
<td>1024</td>
<td>4n</td>
<td>3\sigma</td>
<td>182.7</td>
<td>4.0</td>
<td>19.7(34)</td>
<td>2.9(26)</td>
<td>51.6(64)</td>
<td>43.9(58)</td>
<td>3.2(32)</td>
<td>44.5(57)</td>
<td>47.7(61)</td>
</tr>
<tr>
<td>1024</td>
<td>4n</td>
<td>2\sigma</td>
<td>298.0</td>
<td>2.0</td>
<td>25.7(58)</td>
<td>3.1(44)</td>
<td>65.3(93)</td>
<td>55.2(84)</td>
<td>3.9(56)</td>
<td>52.3(81)</td>
<td>54.9(83)</td>
</tr>
</tbody>
</table>

Table 13: Experiment 1: Well-conditioned $A$. Mean numbers of iterations (work-units in brackets). The average timings are for $n = 2048$.

For all the one-level results, the cost in work-units is similar to the number of iterations, implying that most of the work in each iteration is spent on computing $A^T r$ (which costs 1 work-unit). The second significant operation in terms of cost is the multiplication $A x$, which is performed in $n \|x\|_0$ operations (approximately $\|x\|_0 / m$ work units). Most one-level methods are comparable in their performance (iterations and timings), with an edge to CD$^+$ (CD$^+$ linesearch). However, CD requires each column of $A$ separately, which is problematic in terms of parallelization and handling composite dictionaries as in (94). SESOP1, which does not appear in the table, performs similarly to CG-PCD.

The multilevel acceleration for CG-PCD reduces the required work-units by up to 50% or so compared to CG-PCD, while with CD$^+$ the multilevel speedup is only about 20%. In terms of timings, our implementation of ML has some overhead, therefore, the timings show a slightly smaller advantage than the work-units. In that respect, the cost per iteration of CD$^+$ is higher than that of the others due to the lack of efficient parallelization.

The values of the ISNR measures show that the noise level is indeed reduced in these experiments. The ISNR values in the lower section of the table show that if we choose $\mu$ too low, then the efficiency of the reconstruction decreases.

7.5.2 Experiment 2: Random $\pm 1$ entries $A$

Table 14 summarizes the results for the second problem, which involves a matrix with random $\pm 1$ entries. Although $A$ is random as in the previous problem, it is clear that this problem is more challenging, and all the methods require more iterations to converge.

Unlike the previous case, the CD$^+$ method does not have an advantage over the rest of
The table below presents the mean numbers of iterations (work-units in brackets) for Experiment 2, with random ±1 entries. The average timings are for $n = 2048$.

| $n$    | $m$ | $\mu$ | $||x||_0$ | ISNR | CD+ | ML-CD | CG-PCD | ML-CG | ML-CD/CG | SpaRSA | TwIST | GPSR |
|--------|-----|-------|-----------|------|-----|-------|--------|-------|----------|--------|-------|------|
| 256    | 4n  | 4$\sigma$ | 30.9 | 6.8 | 55.4(66) | 6.0(101) | 3.7(60) | 3.5(42) | 101(133) | 173(238) | 136(156) | |
| 512    | 4n  | 4$\sigma$ | 68.7 | 5.2 | 103(117) | 4.6(42) | 4.4(90) | 3.9(57) | 135(193) | 234(314) | 235(266) | |
| 1024   | 4n  | 4$\sigma$ | 132.3 | 5.3 | 238(258) | 116(132) | 4.8(103) | 3.8(57) | 201(304) | 326(421) | 438(493) | |
| 2048   | 4n  | 4$\sigma$ | 268.9 | 5.2 | 137.5 | 1.13(34) | 1.70sec | 1.66sec | 0.82sec | 5.88sec | 6.91sec | 12.9sec |
| 1024   | 4n  | 4$\sigma$ | 117.8 | 6.5 | 263(231) | 4.9(57) | 7.6(112) | 4.3(85) | 3.7(53) | 152(238) | 247(401) | 329(382) | |
| 1024   | 6n  | 5$\sigma$ | 126.3 | 5.0 | 214(226) | 4.9(37) | 88.2(131) | 4.4(88) | 3.8(47) | 207(304) | 377(482) | 427(473) | |
| 1024   | 8n  | 5$\sigma$ | 143.1 | 4.3 | 263(275) | 5.1(34) | 95.4(135) | 4.9(83) | 3.7(40) | 207(304) | 377(482) | 427(473) | |
| 1024   | 4n  | 5$\sigma$ | 113.8 | 4.7 | 187(200) | 4.9(39) | 73.9(110) | 4.5(90) | 3.7(45) | 172(252) | 326(440) | 374(417) | |
| 1024   | 4n  | 3$\sigma$ | 170.5 | 4.5 | 375(416) | 5.5(76) | 98.9(157) | 5.9(154) | 4.1(80) | 243(393) | 366(514) | 569(645) | |
| 1024   | 4n  | 2$\sigma$ | 268.3 | 4.7 | 684(773) | 6.0(133) | 123(197) | 8.1(204) | 4.7(106) | 308(513) | 448(732) | 983(1128) | |

Table 14: Experiment 2: Random ±1 entries $A$. Mean numbers of iterations (work-units in brackets). The average timings are for $n = 2048$.

The table below presents the mean numbers of iterations (work-units in brackets) for Experiment 3, with ill-conditioned $A$. The average timings are for $n = 2048$.

| $n$    | $m$ | $\mu$ | $||x||_0$ | ISNR | CD+ | ML-CD | CG-PCD | ML-CG | ML-CD/CG | SESOP1 | SpaRSA |
|--------|-----|-------|-----------|------|-----|-------|--------|-------|----------|--------|------|
| 256    | 4n  | 4$\sigma$ | 39.7 | 3.8 | 573(630) | 7.6(101) | 6.7(184) | 5.5(154) | 713(900) | 1941(2279) | |
| 512    | 4n  | 4$\sigma$ | 78.1 | 3.7 | 591(646) | 7.2(115) | 8.0(240) | 5.4(175) | 901(1096) | 2416(2813) | |
| 1024   | 4n  | 4$\sigma$ | 153.3 | 4.0 | 569(624) | 7.7(107) | 1049(1218) | 7.6(222) | 6.0(185) | 1012(1168) | 2542(2886) | |
| 2048   | 4n  | 4$\sigma$ | 293.3 | 4.4 | 635(701) | 5.1(34) | 95.4(135) | 4.9(83) | 3.7(40) | 713(900) | 1941(2279) | |
| 1024   | 2n  | 4$\sigma$ | 169.3 | 3.8 | 406(491) | 5.5(76) | 114(143) | 2.8(52) | 1.6(34) | 713(900) | 1941(2279) | |
| 1024   | 6n  | 5$\sigma$ | 126.3 | 5.0 | 214(226) | 7.7(107) | 1049(1218) | 7.6(222) | 6.0(185) | 1012(1168) | 2542(2886) | |
| 1024   | 8n  | 5$\sigma$ | 143.1 | 4.3 | 263(275) | 5.1(34) | 95.4(135) | 4.9(83) | 3.7(40) | 172(252) | 326(440) | 374(417) | |
| 1024   | 4n  | 1$\sigma$ | 139.1 | 4.5 | 363(395) | 6.2(63) | 992(143) | 5.6(145) | 5.3(134) | 713(900) | 1941(2279) | |

Table 15: Experiment 3: ill-conditioned $A$. Mean numbers of iterations (work-units in brackets). The average timings are for $n = 2048$.

The one-level methods in terms of iterations and work-units. The PCD and SESOP1 methods (omitted from the table) exhibit performance similar to SpaRSA and CG-PCD, respectively; GPSR-BB was outperformed by GPSR and TwIST in this test case.

The multilevel acceleration significantly improves the performance of CD+, while it only slightly accelerates CG-PCD. This is partly a result of an increased support of $x$ during the course of the iterations. Applying ML-CD/CG (which uses CD instead of CG-PCD as a relaxation) overcomes this and improves the performance of ML. In terms of timings ML-CD/CG is clearly the favorite. The ISNR measures for this test case are similar to those of the previous one.

### 7.5.3 Experiment 3: ill-conditioned $A$

In this experiment we use the matrix $A$ with the manipulated singular values. Most submatrices of $A$ are also ill-conditioned, but not necessarily with a similar condition number.
7. A Multilevel Approach for $l_1$ Penalized Least Squares (LASSO)

Table 16: Experiment 4: well-conditioned $A$ with similar columns. Mean numbers of iterations (work-units in brackets). The average timings are for $n = 2048$.

| $n$  | $m$  | $\mu$ | ||$\|x\|_0$| ISNR   | CD$^+$ | ML-CD    | CG-PCD  | ML-CG    | ML-CD/CG | SESOP1    | SpaRSA    |
|------|------|-------|------|---------|---------|--------|---------|---------|---------|---------|-----------|-----------|
| 256  | 4$\sigma$ | 0.0 | 26.8 | 2.5 | 1323 (1392) | 8.5 (123) | 462 (515) | 7.3 (117) | 4.1 (47) | 426 (556) | 1552 (2097) |
| 512  | 4$\sigma$ | 0.0 | 44.9 | 3.6 | 2411 (2497) | 10.9 (202) | 635 (768) | 9.7 (167) | 5.0 (50) | 586 (731) | 2382 (3144) |
| 1024 | 4$\sigma$ | 0.0 | 76.0 | 3.6 | 2460 (2526) | 12.0 (204) | 695 (839) | 9.1 (154) | 4.9 (48) | 719 (903) | 3026 (4057) |
| 2048 | 4$\sigma$ | 0.0 | 148  | 3.7 | 2738 (2800) | 13.9 (263) | 817 (969) | 11.6 (206) | 5.3 (49) | 944 (1177) | 3268 (4525) |

Regarding the one-level methods, again, Table 15 shows that the CD$^+$ method is more effective than the rest of the one-level methods. The PCD and GPSR methods (not shown) did not converge after 4000 iteration and were outperformed by the rest of the methods. The ‘—’ sign indicates that convergence was not reached after 3000 iterations in most of the tests.

Table 15 also shows that now the multilevel algorithm significantly reduces the cost of the solution, compared to the one-level methods. In most cases, the cost of the ML method is about 15%-25% of the cost of the 1L methods in terms of work-units and timings. The Results of ML-CD/CG are slightly better than those of ML-CG.

In the upper section (growing $n$), all methods show quite scalable performance in the work-unit measure. In the second section (growing $m$), it is seen that the problem becomes harder as $m$ grows, since there are more possible supports and more room for error. The ML versions, however, appear quite scalable in their work-unit cost with respect to the growing redundancy ($m$) of the dictionary. In the third section, where different values of $\mu$ are used, one can see the loss of scalability of the methods with respect to the support size. Asymptotically, bigger supports yield bigger effective matrices and also a bigger condition number. Furthermore, finding the true support becomes a much harder task.

The values of ISNR for this test case are lower than those in the previous two cases, indicating that the noise reduction using the $l_1$ regularization is somewhat less effective for this ill-conditioned dictionary.
7.5.4 Experiment 4: well-conditioned $A$ with similar columns

Our last experiment involves a matrix $A$ that consists of two types of columns: those that belong to the random part ($A_1$), and those that belong to the other, replicated part. If the support of the true solution has only columns from $A_1$, the effective matrix $A_S$ is well-conditioned and the solvers handle it well. However, if the support has columns from both sections, then $A_S$ is most likely ill-conditioned even though $A$ is not.

Table 16 shows the results for this problem. Again, as in the second test case, CD$^+$ has no advantage over the rest of the one-level methods. Because the support sub-matrix $A_S$ is ill-conditioned, the CG-PCD and SESOP methods outperform the rest of the one-level methods. This corresponds to CG-PCD outperforming all one-point iterations for quadratic problems. The results of PCD and GPSR (omitted) are similar to those of SpaRSA. GPSR-BB and TwIST were outperformed by all the other methods.

Here we have an interesting case for the multilevel approach. Because the CD$^+$ iteration is not very efficient, neither is the lowest-level solution of ML-CD. Because we limit the number of iterations on the lowest levels, the ML-CD method requires a few more cycles to converge. Compared to that, we see that ML-CD/CG outperforms the other ML options, showing that the CD iteration is better than CG-PCD as a relaxation even though it is far inferior as a solver.

Again, the values of ISNR for this test case are relatively low, but other choices of $\mu$ lead to lower values of ISNR. That is, the noise is reduced as effectively as possible via the solution of (87).

7.5.5 Discussion

The multilevel approach enhances the performance of one-level shrinkage iterations in almost all our tests, and especially in the harder problems. We note that any iterated shrinkage method can be incorporated in our ML framework. Generally, the best option that we found is ML-CD/CG (ML with CD$^+$ as relaxation and CG-PCD as lowest-level solver). That is because the CD iteration updates the entries $x_i$ one by one and updates the residual accordingly, so it does not have a tendency to fill $x$ with a large number of non-zeros which would harm the efficiency of ML. CG-PCD, on the other hand, is generally better at handling ill-conditioned matrices and therefore it is a more robust lowest-level solver.
The one-level methods that we tested behaved differently in the different problems. Overall, the best results were obtained by CD^+, CG-PCD, SESOP1 and SpaRSA. On top of the disadvantages of CD mentioned earlier (evident in its timings), it may struggle when the effective sub-matrix \( A_S \) is ill-conditioned.

### 7.6 Conclusions

A multilevel approach is introduced for the solution of (87) when the matrix \( A \) is given explicitly. The new method takes advantage of the typically sparse representation of the signal by gradually ignoring ostensibly irrelevant data from the over-complete dictionary. This approach significantly accelerates the performance of existing iterated shrinkage methods as well as their accelerated versions. The biggest advantage is gained when the solution is indeed sparse and the problem is effectively ill-conditioned (\( A_S \) ill-conditioned). Also, in most of our numerical tests, the multilevel approach reduced the required number of iterations dramatically. Therefore, we expect significant further gains as more efficient methods are developed for the dense lowest-level problem.
8 Conclusions, current and future work

In this dissertation we have focused on extending the applicability of multigrid and multilevel methods. Our research involves three main studies in which our contribution is as follows:

1. Adaptive AMG methods for Markov chains. We introduce several novel adaptive AMG approaches for computing the principal eigenvector of column-stochastic matrices. We investigate the Smoothed Aggregation method in the rather simple case where only the prolongation is smoothed, and introduce a new bottom-up aggregation approach. The combination of this algorithm and the simple SA approach allows us to choose relatively small aggregates, which leads to nice convergence and operator complexity properties of the multigrid solver. The performance of this method is competitive with the other approaches in literature, and surpassed them mainly for problems with highly complex spectrum. These are often challenging for AMG algorithms based on $C/F$ splitting.

We also introduce an on-the-fly adaptive multigrid framework of interleaving adaptive EIS and classical multigrid cycles for Markov chain problems. In addition to the results presented in this dissertation, this framework was applied with several other multigrid methods, and was found effective with all of them.

2. Non-Galerkin multigrid. In this work we present a new AMG algorithm where the choice of the sparsity pattern of the coarse operators is independent of the choice of the transfer operators. This property makes the algorithm particularly worthwhile for parallel settings.

The new algorithm adopts simple aggregation for determining the sparsity pattern of the coarse operators, and smoothed aggregation for high-quality transfer operators. It sparsifies the SA coarse operators onto the simple aggregation sparsity patterns. Numerical experiments over several benchmark test cases show that the algorithm has promising capabilities, and it seems scalable and robust, at least for linear systems involving M-matrices.

3. A multilevel approach to the LASSO problem. We introduce a multilevel approach for the solution of the LASSO problem. The new method takes advantage of the typically sparse representation of the signal by gradually ignoring ostensibly irrelevant data from
the over-complete dictionary. This approach significantly accelerates the performance of existing iterated shrinkage methods as well as their accelerated versions.

8. Conclusions, current and future work

While our research has contributed to extending the applicability of multigrid methods, there are numerous directions for future research. Below are a few such directions which can be viewed as a direct extension of this dissertation. We have already started tackling some of these ideas.

- **Effective multigrid coarsening of complex graphs.** Classical AMG coarsening, Smoothed Aggregation and other variants, have been found to be very efficient in solving linear systems where the underlying graph of the matrix is “local” or “mesh-like”. Then, the locality of the graph allows the use of sophisticated coarsening such as SA. However, some complex graphs are very hard to coarsen, and the Galerkin SA coarsening may result is a much denser coarse operator. In the extreme case, if the matrix has a dense row and column, the coarse matrix may be completely dense, resulting in $O(n^2)$ non-zeros. In such scenarios, only pure aggregation has been found useful in the literature among multigrid approaches, especially recursively accelerated cycles, which usually require W-cycles. However, some improvements are still necessary, especially for parallel settings where W-cycles are not efficient. Effective coarsening for such problems may eventually be achieved in the context of non-Galerkin multigrid. Although our SpSA algorithm results in an attractive pattern for the coarse matrix, it forms the Galerkin product $RAP$ (explicitly or implicitly) and shares some of the shortcoming of the other algorithms. A non-Galerkin algorithm which does not form the Galerkin product may be a good direction to solve this problem. Our non-Galerkin work in [124] has this property, but some of its ingredients are not suitable for complex graph problems (e.g. calculation of basis function by W-cycles).

- **Effective non-Galerkin multigrid coarsening of non M-matrices and matrices with multiple null-space modes.** In our work in Section 6, we consider mostly M-matrices. We assume that the off diagonal entries of the matrices are non-positive and that there is only one null-vector prototype (either the constant, or an adaptively calculated vector). Although M-matrices appear in many applications, there are others that include non
M-matrices, e.g., rotated anisotropic diffusion and systems of PDEs. This direction has been tackled in [110], but questions remain. In particular, the task of sparsifying an entry while preserving the response of the matrices for more than one vector is still considered open, although some ideas are given in [137, 110, 124].

- **Theory for non-Galerkin coarsening.** The works of [110] and in Section 6 contain some theoretical results and observations, but clearly more research is needed. We have been working with the authors of [110] on this direction in the context of M-matrices, and have achieved a condition for the convergence of the two-level non-Galerkin method. The investigation of the theoretical properties of these approaches has led to several algorithmic changes in [110], which improved the performance of the method. However, obviously, more investigation is still needed.

- **A multilevel approach for \(\ell_1\) regularized least squares with non-explicit matrices** Our work in Section 7 is suitable for cases where the matrix \(A\) is given explicitly. A challenging future research direction may target a multilevel approach for fast operator dictionaries, where the low-level dictionary is chosen as a smaller version of the upper-level fast operator. The key question in this research is how to define the transfer operators so that the solution updates significantly enhance the convergence of the relaxation methods.

Another example of non-explicit matrices is the case where the matrix \(A\) is an inverse of some other sparse matrix \(B\). Then, in the one-level methods we apply a linear system solve at each iteration (that is, we need the product of \(A\) with a vector \(x\), which can be achieved by solving \(Bv = x\)). In this context, in our multilevel approach we would require the result \(B^{-1}x\) only at certain indices in a cost-effective way. If this can be achieved, then our multilevel approach can be advantageous here as well.

- **A multilevel approach to \(\ell_1\) regularized convex optimization.** Consider a general convex sparsity regularized optimization:

\[
\min_{x \in \mathbb{R}^n} F(x) = \min_{x \in \mathbb{R}^n} f(x) + \lambda \|x\|_1,
\]

where \(f(x)\) is a strictly convex problem, and \(\lambda\) a scalar parameter that balances between sparsity and adherence to the minimization of \(f(x)\). Under some reasonable
assumptions on the problem, our multilevel approach of Section 7 is suitable for solving (111) as well. In particular, our current research involves the adaptation of this approach to the problem of sparse inverse covariance estimation, which arises in many statistical applications in Machine Learning and Signal Processing. In this problem, a sparse inverse covariance matrix is estimated from a few samples \( \{x_i\}_{i=1}^m \) of a multivariate normal distribution (each \( x_i \in \mathbb{R}^n \)). To calculate such matrices, an \( \ell_1 \) regularized log-determinant optimization problem is typically solved:

\[
\min_{A > 0} F(A) = \min_{A > 0} \{- \log \det(A) + \text{tr}(AS) + \lambda \|A\|_1\}, \tag{112}
\]

where \( A \in \mathbb{R}^{n \times n} \) is a symmetric matrix of variables, the term \( \|A\|_1 = \sum_{i,j} |A_{ij}| \) acts as a sparsity promoting regularization, and \( S = \frac{1}{k} \sum_{i=1}^m x_i x_i^T \in \mathbb{R}^{n \times n} \) is the empirical covariance estimate (the “data”). This problem corresponds to the formulation in (111) for \( f(A) = - \log \det(A) + \text{tr}(AS) \), which is strictly convex. However, its gradient is given by

\[
\nabla f(A) = -A^{-1} + S, \tag{113}
\]

and it contains the inverse of the sparse matrix \( A \). This inverse is one of the challenging ingredients of this problem, because 1) calculating an inverse of a matrix is expensive even though it is sparse, and 2) the inverse of a sparse matrix is typically dense, and does not fit in memory when the matrix is large.

We have approached the problem (112) in two aspects:

1. We developed a Block-Coordinate-Descent method that is suitable for large scale problems, and has many unique advantages.
2. We have adapted the multilevel method to accelerate several existing methods of generic type for the solution of (112).

Our results indicate that our multilevel approach is very useful in this context as well, especially for large-scale instances of this problem, which are more challenging.
A. Adaptive AMG for Markov Chains

A.1 Analysis of the Weighted Power Method for Complex Spectrum

Here, we present an analysis of the weighted power method with complex spectra. We show that significantly complex eigenmodes, i.e., those with a substantial imaginary part, are damped fairly efficiently by simple relaxation methods, which explains why we define smooth eigenmodes as modes with a corresponding eigenvalue with real part close to 1 (and not absolute value as traditionally considered). Also, we show that our choice of damping factor $\omega = 0.5$ is optimal in a certain worst-case sense.

Any eigenvalue of a stochastic matrix can be written as $\lambda = re^{i\theta}$, where the modulus satisfies $0 \leq r = |\lambda| \leq 1$ and $-\pi \leq \theta < \pi$. Throughout this analysis, we specifically focus on truly complex eigenvalues, with a non-vanishing imaginary part, hence $\theta \neq 0$. Consider applying a single iteration of the weighted power method (33) with (real) damping factor $\omega$.

The amplitude of the corresponding eigenvector is then multiplied by $\mu_w(r, \theta) = 1 - \omega + \omega re^{i\theta}$, which yields

$$|\mu_w(r, \theta)|^2 = 1 - 2\omega(1 - r \cos \theta) + \omega^2(1 - 2r \cos \theta + r^2).$$

The optimal $\omega$ which minimizes this factor for a given $\lambda$, can be obtained by differentiation, and it is a function of $r$ and $\theta$. As we only choose one value of $\omega$ to fit all values of $r$ and $\theta$, we first maximize (114) over $r$, and then optimize it over $\omega$. Since the second derivative of (114) with respect to $r$ is positive independently of $\theta$, the maximum is attained on the boundaries, $r = 0$ or $r = 1$, where

$$|\mu_w(0, \theta)|^2 = (1 - \omega)^2, \quad |\mu_w(1, \theta)|^2 = 1 - 2\omega(1 - \omega)(1 - \cos \theta).$$

Observe that, since $\cos \theta < 1$, the optimal $\omega$ for $r = 1$ is 0.5 independently of $\theta$, while the optimal $\omega$ for $r = 0$ is obviously 1. Furthermore, for $\omega = 0.5$, we obtain

$$|\mu_{1/2}(1, \theta)|^2 = \frac{1}{2}(1 + \cos \theta),$$

(115)
which is greater than $|\mu_{1/2}(0, \theta)|^2$ for any $-\frac{2}{3}\pi < \theta < \frac{2}{3}\pi$. Hence, for this range of $\theta$, (115) yields the worst-case error reduction. Moreover, for $\theta$’s outside this zone (that is, $|\theta| \in [\frac{2}{3}\pi, \pi]$, which correspond to very rough modes), $\omega = 0.5$ remains effective, and all these modes are damped efficiently, with $|\mu_{1/2}|^2 \leq 0.25$. Note by (115), that setting $\omega = 0.5$ yields damping factors that are bounded away from 1 for any eigenvalue that is significantly complex (that is, with $\theta$ bounded away from 0).

Figure 23 shows a comparison of the effectiveness of the weighted power method with the weights $\omega$ equal to 1 and 0.5, for damping eigenmodes in the unit circle. It shows that eigenmodes with highly complex eigenvalues are reduced fairly efficiently by the method (with $\omega = 0.5$), and therefore we consider only eigenmodes with eigenvalues close to 1 as smooth.

![Figure 23: A comparison between the weighted power method effectiveness when using weights $\omega = 1$ and $\omega = 1/2$, for damping eigenmodes in the unit circle. Absolute value 0 appears in white while absolute value 1 appears in black, with linear gray-scale for values in-between.](image)

A.2 Analysis of Smoothed Aggregation for Complex Spectrum

Suppose for simplicity that $\rho(A) = 2$, and recall that $A = I - B$. Then, the choice of damping parameter in (47) corresponds to the choice $\omega = 0.5$ described above:

$$I - \frac{1}{2}A = \frac{1}{2}I + \frac{1}{2}B.$$
Figure 24: A comparison between $|\lambda_k|$ (left) and $|\lambda_k||1 - \frac{1}{2}\lambda_k|$ (right), for $\lambda_k = (1 - \lambda_B)$ and $\lambda_B$ in the unit circle. Absolute value of 0 appears in white while absolute value of 2 appears in black, with linear gray-scale for values in-between.

We note, however, that the motivation for choosing (47) as a damping parameter for $P$ is different—it is ensuring the positive definiteness of the coarse-grid matrix. So far, only real eigenvalues were considered.

By applying Petrov-Galerkin coarsening in (36) we define the coarse-grid problem, which can be interpreted as the following minimization problem:

$$x_c = \arg \min_{\|x_c\|=1} \|R_t A P_s x'_c\| = \arg \min_{\|x_c\|=1} \|R_t A(I - \frac{1}{\rho(A)} A) P_t x'_c\|. \quad (116)$$

Note that this is equivalent to replacing $P_t$ by $P_s$ in the right-hand side of (44). Suppose again that we can write the prolongated coarse-grid vector as a linear combination of the eigenvectors of $A$: $P x'_c = \sum_{k=1}^{n} \alpha_k v_k$. Then the minimization problem becomes

$$\arg \min_{\{\alpha_k\} \in A_P} \left\| \sum_{k=2}^{n} \alpha_k \lambda_k (1 - \frac{1}{2}\lambda_k) R v_k \right\|, \quad (117)$$

where

$$A_P := \{\{\alpha_k\} : \sum_k \alpha_k v_k \in \text{Range}(P_t) \text{ and } \|\sum_k \alpha_k R_t v_k\| = 1 \}.$$ 

Following the arguments of section 5.2.5, we focus on minimizing each term in the argument of (117) separately. Every term is either damped by the restriction ($\|R v_k\|$ is small), or attenuated through the nature of the range of the prolongation ($|\alpha_k|$ is small), or else reduced
by the new factor $|\lambda_k| |1 - \frac{1}{2}\lambda_k|$. Now, in order to prevent the distraction of the minimization as discussed in section 5.2.5, if $v_k$ is rough we need $|\lambda_k| |1 - \frac{1}{2}\lambda_k|$ to be small.

Figure 24 shows a comparison between $|\lambda_k|$ and $|\lambda_k| |1 - \frac{1}{2}\lambda_k|$ in the unit circle. It shows that using (47) damps the highest values of $|\lambda_k|$, thus contributing to the efficiency of the coarse-grid correction. The eigenmodes that appear to have a relatively large factor in (117) are those with eigenvalues $\lambda \approx 1 \pm i$. However, as shown in the previous section, choosing a damping factor of 0.5 is optimal for those modes, so there is no room for improvement by choosing another value. (In addition, these modes are damped by the relaxation only moderately well, so attenuation of these modes by the coarse-grid correction is required to some extent.)

### A.3 Acceleration of SA when only $P$ is smoothed

In this section we describe some simple changes in the multigrid algorithm for accelerating the convergence of Smoothed Aggregation. In [63], an overcorrection technique is used to improve the performance of smoothed aggregation. Similarly, we apply over-correction,

$$x \leftarrow x + \alpha Pe_c,$$

instead of the regular CGC in Step 5 of Algorithm 1, where $\alpha > 1$ is the over-correction parameter. Correspondingly,

$$x \leftarrow (1 - \alpha)x + \alpha Px_c$$

is used in step 6 of Algorithm 3. In all our tests we fix $\alpha = 1.1$.

Finally, in [62, 133] it is advised to perform an additional pre-relaxation with the same smoothing operator used for smoothing $P$. Motivated by this, we modify $x$ after computing the residual that is transferred to the coarser level. That is, we add

$$x^k \leftarrow x^k + \omega Q^{-1}r$$

to step 2 of Algorithm 1, using the same $Q$ and $\omega$ as in Equation (42). Such use of the residual was suggested already in [47]. Note that this modification is very cheap as it does not require a matrix-vector multiplication. Using this procedure, the smoother of $P$ acts on both $x$ and the correction $Pte_c$, instead of $Pte_c$ alone. By linearity, the CGC operator of our
SA version of Algorithm 1 is then given by

\[(I + \omega Q^{-1}A)(I - \alpha P_t(A_c)^+ R_t A)\] (121)

instead of

\[(I - \alpha(I + \omega Q^{-1}A)P_t(A_c)^+ R_t A),\] (122)

as it would be without this modification. This small and cheap change has a significant effect, almost the same as an additional post-relaxation.

**B Smoothed Aggregation—a modified version.**

In this section we describe the details of the version of the SA method that we use in Section 6 as the “Galerkin” method. We note that for some of the non-symmetric problems that we tested, it performs significantly better than the original SA method [131], which was shown exhibit slow convergence for some convection-diffusion problems also in [108, 97, 32]. Our most significant change involves using two different parameters for strength of connections: one rather large in the aggregation procedure, and one rather small in the prolongation filtering procedure. In [131], these parameters are equal. Similarly, in the previous sections we described a binary strength-of-connection matrix \(S\) that is defined based on (14) for some parameter \(\theta\), and both the aggregation in Algorithm 2 and the filtering (29) is applied accordingly. Using two different values for these two procedures may result in rather smaller aggregates and rather large stencils on coarse grids, together with better convergence rates.

We note, however, that for isotropic problems we typically do not get a significantly different strength-of-connection matrix for the aggregation and also typically no filtering is performed. In particular, for a 5-point Laplacian operator, all the entries are considered strong and no filtering is applied. Therefore, for isotropic problems our version of SA behaves similarly to the original SA in terms of operator complexity and stencil sizes—this is evident in our numerical results. For other problems, e.g., convection dominated (54) and non-homogenous graph Laplacian, using the original version of SA (or, using just one parameter in our version) generates lower operator complexity. However, it also significantly degrades the convergence—especially for the 2D problems, which are harder because of their small mesh size. This is also evident in the literature mentioned earlier. Below, we choose the
parameters for our method such that SA has mesh independent convergence for all problems, and those parameters were mostly dictated by the 2D problems.

To apply the changes mentioned above, we first compute the following matrix:

\[
S_{ij} = \begin{cases} 
1 & i = j \\
\frac{-A_{i,j}}{\max_{k \neq i} \{ -A_{i,k} \}} & i \neq j \\
0 & \text{otherwise}
\end{cases} .
\]

(123)

Then, following (14) we treat \((i, j)\) as strongly connected if \(S_{ij} > \theta\), where \(0 < \theta < 1\) as a strength parameter.

For the aggregation procedure, we use a symmetric version of the strength matrix, \(S(\theta)\), such that:

\[
S_{ij}(\theta) = \frac{1}{2} (S_{ij} + S_{ji}) : S_{ij} > \theta,
\]

(124)

with a rather large \(\theta = 0.5\) (in the literature it is usually chosen in the range 0.15-0.35). Generally, the larger \(\theta\) we use, the smaller will be our aggregates, since less strong connections are considered.

Given the above strength of connection matrix, we found the original aggregation algorithm of [131] to be efficient, and most importantly—very fast to compute. However, for the unstructured problems that we test there are cases where some rows in the matrix have significantly more non-zeros than the rest of the rows. Such a row may result in a very large aggregate if it is chosen as a seed of an aggregate. It may also have the same effect even if it is not chosen as a seed. Furthermore, even if it is not a seed, all of its neighbors are then not aggregated in the first pass and less aggregates are generated. To illustrate this, consider the case of one dense column and row with strong connections corresponding to a point \(i\). Since all points are strongly connected to \(i\), only one aggregate will be chosen by the original algorithm, containing all points. To solve this, in the first pass we ignore points with a relatively large number of strong connections, and treat them separately in a second identical pass. Algorithm 13 summarizes our aggregation scheme. In the original aggregation algorithm in [131], only the first and third passes are applied for \(\mathcal{N}_1 = \{1, ..., n\}\) and \(\mathcal{N}_2 = \emptyset\). We use \(\tau = 3\) as the relative threshold for large neighborhoods, but note that any \(\tau \in [1.5, 4]\) works and seems reasonable for us.

As noted before, the next step in the SA algorithm includes smoothing the tentative
**Algorithm 13: Modified Neighborhood-based Aggregation**

\[ \{C_J\}_{j=1}^{n_c} \leftarrow \text{Neighborhood-Aggregation}(A, \theta, \tau) \]

- **% \( \theta \) - Strength of connection parameter.**
- **% \( \tau \) - Relative neighborhood size parameter.**

\( S(\theta) \): Strength of connection matrix based on (124).

Let \( S_i = \{j : S_{ij}(\theta) \neq 0\} \) denote the strong neighborhood of each point \( i \)

Compute average neighborhood size: \( \bar{s} = \frac{1}{n} \sum_{j=0}^{n} |S_i| \)

Set \( \mathcal{N}_1 = \{i : |S_i| \leq \tau \bar{s}\}, \mathcal{N}_2 = \{i : |S_i| > \tau \bar{s}\}, \) and \( J \leftarrow 0 \)

- **% First Pass: Assign neighborhoods to aggregates only for points with relatively small neighborhoods.**

  - foreach \( i \in \mathcal{N}_1 \) do
    - if each \( j \in S_i \) does not belong to an aggregate then
      - Set \( J \leftarrow J + 1, C_J \leftarrow S_i \setminus \mathcal{N}_2, \) and \( \hat{C}_J \leftarrow C_J. \)
    - end
  - end

- **% Second Pass: Assign neighborhoods to aggregates for the rest of the points.**

  - foreach \( i \in \mathcal{N}_2 \) do
    - if each \( j \in S_i \) does not belong to an aggregate then
      - Set \( J \leftarrow J + 1, C_J \leftarrow S_i, \) and \( \hat{C}_J \leftarrow C_J. \)
    - end
  - end

- **% Third Pass: Assign remaining points to the aggregates.**

  \( n_c \leftarrow J \)

  - foreach \( i \in \mathcal{N} \) do
    - \( J \leftarrow \arg \max_{K=1,\ldots,n_c} \left\{ \frac{1}{|\hat{C}_K|} \sum_{j \in \hat{C}_K} S_{ij} \right\} \)
    - Set \( C_J \leftarrow \hat{C}_J \cup \{i\} \)
  - end

---

B. Smoothed Aggregation—a modified version.
operators (Equations (26)-(27)), and for that, the filtered matrix \(A^F\) and the diagonal preconditioner \(Q\) need to be defined. Our filtering process to define \(A^F\) is similar to [131], only again we use a different strength of connection matrix, based on (123). Specifically, we define \(\hat{S}_i(\epsilon) = \{j : |S_{i,j}| < \epsilon\}\). Then, \(A^F\) is defined by

\[
A^F_{i,j} = \begin{cases} A_{i,j} & \text{if } j \in \hat{S}_i(\epsilon) \\ 0 & \text{otherwise} \end{cases}, \quad A^F_{i,i} = A_{i,i} + \sum_{j \notin \hat{S}_i(\epsilon)} A_{i,j}.
\]

(125)

In this work we use \(\epsilon = 0.02\). Note that \(A1 = A^F1\).

For the smoothing preconditioner \(Q\) in (26)-(27), the inverse of the diagonal of \(A\) is usually chosen, and then \(I - \omega QA\) is the error propagation matrix associated with the damped Jacobi relaxation. In this work, we use the SPAI diagonal preconditioner [34] for \(Q\), i.e., we choose \(Q\) as the diagonal matrix that minimizes \(\|I - QA^F\|_F\) which leads to \(Q_{ii} = \frac{A^F_{i,i}}{\sum_j (A^F_{i,j})^2}\) for \(i = 1, ..., n\).

This is a more sophisticated relaxation operator than Jacobi, and in the context of smoothing \(P_t\) it is somewhat related to the energy minimization diagonal preconditioner (EMIN) in [108]. As in [108], we also found that such a \(Q\) is much more efficient for solving (54) than the Jacobi operator. We also found that additional weighting is needed. While the minimization of the Frobenius norm \(\|I - QA^F\|_F\) is essentially minimizing the average of the singular values of the matrix \(I - QA^F\), prolongation smoothing is often related to a minimization of the maximal eigenvalue of the Galerkin coarse operator \(A_g\) (at least in the symmetric case). Therefore, for symmetric problems we follow the classical prolongation smoothing weight which is based on the Chebychev polynomials, \(\omega = \frac{4}{3\rho(QA^F)}\), and for non-symmetric problems we use a slightly lower value: \(\omega = \frac{5}{4\rho(QA^F)}\), following the observations in [122]. In both cases, \(\rho(QA^F)\) is approximated by \(\|QA^F\|_\infty\).
References


REFERENCES


REFERENCES


REFERENCES


שיטות רב-סריג אלגבריות אדווטיביות מובססות
ארגנטיה לפתרון מערכות ליניאיות דלילות

ערן טרייסנר

Technion - Computer Science Department - Ph.D. Thesis PHD-2014-12 - 2014
שיטות רב-סריג אלגבריות אדפטיביות מבוססות אגרגציה לフトורז מערכות ליניאריות דלילות

עור טרייסטר

 hồיש לסטט הטכניון - מכון טכנולוגי לישראל
 אליי טש"ד חיפה ספטמבר 2014
המחקר נערך בהנחיית פרופ’ יעייד יבנה בפקולטה למדעי המחשב.

תמיכת כספית:
אני מודה לקרן עזריאלי על מלגת עזריאלי הנדיבת.
אני מודה לפény הפעלתה על התמיכה הכספית הנדיבת בהשתלמותי.
הממתך מומן בע”מ קרן המדע הישראלית (ISF), הסכם מספר 795/08, הסכם מספר 795/08.
הממטה מומן בע”מ, מחקר של אינטלקטואל גנטיקה ו italia (ICRI-CL).
A\textsuperscript{x} = b,
1. \[ Bx = x, \quad \|x\|_1 = 1, \quad x > 0, \]

where \( B \) is a matrix that satisfies certain conditions, and \( x \) is a vector that satisfies the equation. This equation is crucial for solving Markov chains, as it helps in finding the initial vector and the hierarchy of the matrices, which is essential for the subsequent calculations.

The study of Markov chains is a vital area of research, especially due to its applications in network search, among others. The problem of finding the principal vector \( x \) of a stochastic matrix in the columns is relatively challenging for the existing iterative methods, especially since they require a good initial approximation of the vector we are searching for. This vector is needed to define the hierarchy of the matrices ("grids") that we work with at the computational stage.

At this point, the hierarchy is built using a certain vector as the initial vector, and the efficiency of these methods in solving the problem is very dependent on the accuracy of this initial vector. This vector is needed to improve the hierarchy of the matrices using an iterative solution of the problem. Within this framework, we developed new ideas for improving the "grids" method based on the "smoothed aggregation" (\textit{Smoothed Aggregation}), which is suitable for both symmetric and asymmetric problems.

In addition, we developed a method of increasing the \textit{V-cycle} for linear non-symmetric problems. The contribution focused on finding the correction parameter dynamically.

The most significant contribution of our research is the development of a "on-the-fly" algorithm. This algorithm is built in a way that it can construct and improve the operators of the hierarchy and also improve the approximation of the solution using linear procedures that are more computationally efficient.

In summary, our algorithm first tries to solve the problem using the hierarchy that was built using such approximation. This process is typically good for some iterations but its efficiency decreases. In this way, we improve the approximation, and when the iterations are no longer efficient, we produce a new hierarchy based on the improved approximation. This process, which improves itself, can be combined with any other "grids" method and is significant to accelerate it. In our experimental tests, the process was examined and accelerated several other "grids" methods that were examined in the literature.

Technion - Computer Science Department - Ph.D. Thesis  PHD-2014-12 - 2014
2. Show one method of solving the equation by the method of "Golarkin": Argeziya

Methodology.

Now the author of this book concludes with a detailed description of the method of solving the equations by the method of "Golarkin". The method of solving the equations is based on the fact that the equations are not linear. The method of solving the equations is based on the fact that the equations are not linear.

The second subject that we concentrated on is a special subject that is based on the idea of solving by the method of "Golarkin". Algebraic multigrid methods use a hierarchy of matrices that are used to represent the problem of the size of the matrix (1). The system is divided at the level of the matrix, the solution of which is the solution of the equation. The system is divided at the level of the matrix, the solution of which is the solution of the equation.

For the system of the equations, we use the method of "Golarkin". The system is divided at the level of the matrix, the solution of which is the solution of the equation.

In order to define these matrices, most of the available methods use the technique of "projection" by Golarkin. In this technique, the problems are defined by the projection of the original matrix on the spaces formed by the operators, which are used to transform the operators at the level of the matrix. The projection is done recursively of the original matrix on the spaces formed by the operators, which are used to transform the operators at the level of the matrix.

In a more formal way:

Let us denote by $c_{nn}R$ the projection operator to a smaller space of size $c_{nn}$, and let us denote by $c_{nn}P$ the interpolation operator to a larger space of size $c_{nn}$, then the matrix of the grid solution we define by the "Golarkin" method:

$$A_c = RAP.$$  

If the condition $P-I R$ is satisfied, the matrices $P-I R$ are of the form $P-I R = P-I R$, and the matrices of the matrix are defined by the projection of the original matrix on the spaces formed by the operators, which are used to transform the operators at the level of the matrix.

In this method, the operators are not very sparse, but the number of variables on the grid is relatively small. When using this idea, which is also used in classical algorithms such as aggregation splitting and algebraic multigrid, the matrices on the grids are less likely to be significantly less than the original, especially as the grid advances.

This phenomenon is mainly problematic in parallel and coupled multigrids, where a lot of communication between processors is needed to perform calculations with the matrices that are less sparse. This communication becomes more dominant than the time to perform the calculations themselves.

In this work, we present a new algebraic method that is not "Golarkin" to control the pattern of the matrices on the grids in the hierarchy, without regard to the choice of the operators $R$ and $P$. Our method is based on the aggregation, and it "projects" the operators of the aggregation splitting so that...

Technion - Computer Science Department - Ph.D. Thesis PHD-2014-12 - 2014
We're exploring various techniques for solving linear systems, particularly those that can be expressed in the form $Ax = y$, where $A$ is a matrix, $x$ is a vector, and $y$ is another vector. When $A$ is a large and sparse matrix, direct methods can be computationally expensive. Therefore, we propose an iterative method to solve such systems.

The core of our method is to iteratively construct a hierarchy of subsystems, starting from the original system. This hierarchy is dynamically updated in each iteration to minimize the norm of the residual vector, i.e.,

$$
\min_x \left\{ \|Ax - y\|_2^2 + \mu \|x\|_1 \right\},
$$

where $A \in \mathbb{R}^{m \times n}$, $m > n$, is the matrix, $y$ is the right-hand side vector, and $\mu > 0$ is a regularization parameter. The solution is sought in an iterative fashion, where at each iteration, a subsystem is solved, and the hierarchy is updated accordingly.

The method can be seen as a combination of shrinkage (4), which is a technique for reducing the norm of the solution vector, and gradient descent, which is a popular optimization method. The shrinkage technique helps in reducing the complexity of the system, while gradient descent helps in finding the optimal solution.

Our proposed method is particularly useful when dealing with large-scale problems, as it allows for efficient computation and can handle ill-conditioned matrices effectively. It is also adaptable to various types of problems, including those arising in physics and engineering.

In conclusion, our method provides a effective and efficient solution to the problem of solving large-scale systems of linear equations, outperforming traditional methods in terms of both speed and accuracy.
של בעיות ממימדちょizio מאלה השורות של הקווית את סדרת ההפרדה בין החוזיות, באפשרות.getHeader קלאסית, יש להוסטריסטים, ובשלול המתחדש של האטואים ניתן למגעים "עבורי מיצוב של אפסים" בפרטים. כל אחד של התפישור על פעולות הפרטים הסופיים. התואמות האטואים מרוצאות, והמתון מעמדות, מאשר⇏ה את החותים של הפרטים בקופסא משמעתיי" לתfefות, הקווית החופשת את המטריצה A נוגנת לפני. התשובה ביעילות וכפי קʇפיזת יש מספר מבר.ヴהנה

במסגרת זו אנו גורשים במחקרים המשקיר של הגילויים הרוב-שכבתיים של בעיות

אוספרימ نيوز שומטים לכל חומרים רף-גרליזיות לילופוט, אונח הפטולא את הגהה על ביעות משעון

מתווצצת של השומטמיה (covariance) שלילות. בעיה זו היא שמיושמת בתחרום גני טמירות שבשלבי

עיבוד וינואות, זכויות א-יוואות (uncertainty quantification). בעיה זו היא מיטור של משמעויות

דיליל מתווצצות ממקבזת יחסן קסטה של דימויי ה-ממדים (מספר הד.paging קסט משמעויות-מ). על מעמס של השעון אך המיטורixa לפקוט ביעה מיניימיציה שאיס גרסה לא ריבועי של הيفة (4)

סוכס של פונקציה תלקת קומורא עם גרליזיות ל-1 ל-1. כלל התפיפות שלילות, בעיה זו היא קושי משמוד

כאשר המשך 2 במדים. המחקר של ברע-

ף-נפס עם ביטיע 3 אספונים: (1) פתרון הביעות

גודל ה-2) המאיצין בר-edException שיא שיווט ק الأمري רשיידות. התואמות הממודים שגויות של

מצא עיון בפרטים הביעות, ועבור הרוב-เศבטיות עלבב בمشارך שאר הממדים.ヴהנה

של בעיות ממימד יחידה מאלה השורות של הקווית את סדרת ההפרדה בין החוזיות, באפשרות.getHeader קלאסית, יש להוסטריסטים, ובשלול המתחדש של האטואים ניתן למגעים "עבורי מיצוב של אפסים" בפרטים. כל אחד של התפישור על פעולות הפרטים הסופיים. התואמות האטואים מרוצאות, והמתון מעמדות, מאשר нескק אם החותים של הפרטים בקופסא משמעתיי" לתfefות, הקווית החופשת את המטריצה A נוגנת לפני. התשובה ביעילות וכפי קʇפיזת יש מספר מבר.ヴהנה

במסגרת זו אנו גורשים במחקרים המשקير של הגילויים הרוב-שכבתיים של בעיות

אוספרימ نيوز שומטים לכל חומרים רף-גרליזיות לילופוט, אונח הפטולא את הגהה על بיעות משעון

מתווצצת של השומטמיה (covariance) שלילות. בעיה זו היא שמיושמת בתחרום גני טמירות שבשלבי

עיבוד וינואות, זכויות א-יוואות (uncertainty quantification). בעיה זו היא מיטור של משמעויות

דיליל מתווצצות ממקבזת יחסן קסטה של דימויי ה-ממדים (מספר הד.paging קסט משמעויות-מ). על מעמס של השעון אך המיטורixa לפקוט ביעה מיניימיציה שאיס גרסה לא ריבועי של הيفة (4)

סוכס של פונקציה תלקת קומורא עם גרליזיות ל-1 ל-1. כלל התפיפות שלילות, בעיה זו היא קושי משמוד

כאשר המשך 2 במדים. המחקר של ברע-

ף-נפס עם ביטיע 3 אספונים: (1) פתרון הביעות

גודל ה-2) המאיצין בר-edException שיא שיווט ק الأمري רשיידות. התואמות הממודים שגויות של

מצא עיון בפרטים הביעות, ועבור הרוב-เศבטיות עלבב בمشارך שאר הממדים.ヴהנה

של בעיות ממימד יחידה מאלה השורות של הקווית את סדרת ההפרדה בין החוזיות, באפשרות.getHeader קלאסית, יש להוסטריסטים, ובשלול המתחדש של האטואים ניתן למגעים "עבורי מיצוב של אפסים" בפרטים. כל אחד של התפישור על פעולות הפרטים הסופיים. התואמות האטואים מרוצאות, והמתון מעמדות, מאשר нескק אם החותים של הפרטים בקופסא משמעתיי" לתfefות, הקווית החופשת את המטריצה A נוגנת לפני. התשובה ביעילות וכפי קʇפיזת יש מספר מבר.ヴהנה

במסגרת זו אנו גורשים במחקרים המשקיר של הגילויים הרוב-שכבתיים של בעיות

אוספרימ نيوز שומטים לכל חומרים רף-גרליזיות לילופוט, אונח הפטולא את הגהה על ביעות משעון

מתווצצת של השומטמיה (covariance) שלילות. בעיה זו היא שמיושמת בתחרום גני טמירות שבשלבי

עיבוד וינואות, זכויות א-יוואות (uncertainty quantification). בעיה זו היא מיטור של משמעויות

דיליל מתווצצות ממקבזת יחסן קסטה של דימויי ה-ממדים (מספר הד.paging קסט משמעויות-מ). על מעמס של השעון אך המיטורixa לפקוט ביעה מיניימיציה שאיס גרסה לא ריבועי של הيفة (4)

סוכס של פונקציה תלקת קומורא עם גרליזיות ל-1 ל-1. כלל התפיפות שלילות, בעיה זו היא קושי משמוד

כאשר המשך 2 במדים. המחקר של ברע-

ף-נפס עם ביטיע 3 אספונים: (1) פתרון הביעות

גודל ה-2) המאיצין בר-.VisibleIndex שיא שיווט ק الأمري רשיידות. התואמות הממודים שגויות של

מצא עיון בפרטים הביעות, ועבור הרוב-שקובטיות עלבב בمشارך שאר הממדים.ヴהנה

של בעיות ממימד יחידה מאלה השורות של הקווית את סדרת ההפרדה בין החוזיות, באפשרותHeaderText קלאסית, יש להוסטריסטים, ובשלול המתחדש של האטואים ניתן למגעים "עבורי מיצוב של אפסים" בפרטים. כל אחד של התפישור על פעולות הפרטים הסופיים. התואמות האטואים מרוצאות, והמתון מעמדות, מאשר нескק אם החותים של הפרטים בקופסא משמעתיי" לתfefות, הקווית החופשת את המטריצה A נוגנת לפני. התשובה ביעילות וכפי קʇפיזת יש מספר מבר.ヴהנה

במסגרת זו אנו גורשים במחקרים המשקיר של הגילויים הרוב-שכבתיים של בעיות

אוספרימ نيوز שומטים לכל חומרים רף-גרליזיות לילופוט, אונח הפטולא את הגהה על ביעות משעון

מתווצצת של השומטמיה (covariance) שלילות. בעיה זו היא שמיושמת בתחרום גני טמירות שבשלבי

עיבוד וינואות, זכויות א-יוואות (uncertainty quantification). בעיה זו היא מיטור של משמעויות

דיליל מתווצצoses ממקבזת יחסן קסטה של דימויי ה-ממדים (מספר הדגמית קסט משמעויות-מ). על מעמס של השעון אך המיטורixa לפקוט ביעה מיניימיציה שאיס גרסה לא ריבועי של הيفة (4)

סוכס של פונקציה תלקת קומורא עם גרליזיות L-L. כלל התפיפות שלילות, בעיה זו היא קושי משמוד

כאשר המשך 2 במדים. המחקר של ברע-

ף-נפס עם ביטיע 3 אספונים: (1) פתרון הביעות

גודל ה-2) המאיצין בר-очки שיא שיווט ק الأمري רשיידות. התואמות הממודים שגויות של

מצא עיון בפרטים הביעות, ועבור הרוב-שקובטיות עלבב בمشارך שאר הממדים.ヴהנה

של בעיות ממימד יחידה מאלה השורות של הקווית את סדרת ההפרדה בין החוזיות, באפשרותHeaderText קלאסית, יש להוסטריסטים, ובשלול המתחדש של האטואים ניתן למגעים "עבורי מיצוב של אפסים" בפרטים. כל אחד של התפישור על פעולות הפרטים הסופיים. התואמות האטואים מרוצאות, והמתון מעמדות, מאשר нескק אם החותים של הפרטים בקופסא משמעתיי" לתfefות, הקווית החופשת את המטריצה A נוגנת לפני. התשובה ביעילות וכפי קʇפיזת יש מספר