ALGEBRAIC COLLOCATION COARSE APPROXIMATION (ACCA) MULTIGRID

RAN ZEMACH, ERAN TREISTER, IRAD YAVNEH

Abstract. Most algebraic multigrid (AMG) methods define the coarse operators by applying the (Petrov-)Galerkin coarse approximation where the sparsity pattern and operator complexity of the multigrid hierarchy is dictated by the multigrid prolongation and restriction. Therefore, AMG algorithms usually must settle on some compromise between the quality of these operators and the aggressiveness of the coarsening, which affects their rate of convergence and operator complexity. In this paper we propose an algebraic generalization of the *collocation coarse approximation* (CCA) approach of Wienands and Yavneh, where the choice of the sparsity pattern of the coarse operators is independent of the choice of the high-quality transfer operators. The new algorithm is based on the aggregation framework (smoothed and non-smoothed). Using a small set of low-energy eigenvectors, it computes the coarse grid operator by a weighted least squares process. Numerical experiments for two dimensional diffusion problems with sharply varying coefficients demonstrate the efficacy and potential of this multigrid algorithm.

1. Introduction

Multigrid methods are well known for their efficiency in solving linear systems arising from the discretization of elliptic partial differential equations (PDEs). See the introductory [11, 12, 30], the comprehensive [26], and the classical [3, 4, 15, 16]. The discretization yields a sparse, typically large system of equations

$$A\mathbf{x} = \mathbf{b}, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$, and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$ are the vector of unknowns and the sampled right-hand side (RHS) of the PDE, respectively.

It is common to distinguish between geometric multigrid, whereby the problem is associated with a regular grid, and algebraic multigrid (AMG) where usually the matrix A in (1) is given explicitly with no other information on the underlying system. Therefore, AMG methods are often chosen for solving discretized PDEs over unstructured grids as well as certain difficult problems on structured grids.

In order to solve (1), AMG methods use two complementary components: local iterative methods, such as Jacobi or Gauss-Seidel, and coarse-grid correction (CGC). The local iterative methods are usually inefficient in handling certain error modes, called "algebraically smooth", and thus they are often referred to as *relaxations* or *smoothers*. CGC aims at handling these algebraically smooth modes, and is done by solving a coarse-grid problem, that is a lower-dimensional version of the error equation $A\mathbf{e} = \mathbf{b} - A\mathbf{x} = \mathbf{r}$, where $\mathbf{r}$ is the residual. In most AMG methods the coarse-grid problem $A_c\mathbf{e}_c = \mathbf{r}_c$ is defined by the (Petrov) Galerkin coarse approximation (GCA)

$$A_c = RAP, \quad \mathbf{r}_c = R(\mathbf{b} - A\mathbf{x}), \tag{2}$$

Date: January 11, 2012.

All authors are from the Department of Computer Science, Technion—Israel Institute of Technology.
Contact emails: ran.zemach@gmail.com, eran@cs.technion.ac.il, irad@cs.technion.ac.il.

Research supported by the Israeli Science Foundation, grant number 795/08.
Eran Treister is grateful to the Azrieli Foundation for the award of an Azrieli Fellowship.

which is a projection of the error equation onto the subspace defined by the full-rank prolongation and restriction operators, $P \in \mathbb{R}^{n \times n_c}$ and $R \in \mathbb{R}^{n_c \times n}$, respectively, with $n_c < n$. The process is applied recursively, resulting in a hierarchy of successively coarser problems and their associated operators.

One drawback of GCA is that the control over the sparsity of A_c is quite limited and is dictated by P and R. This might lead to a growth in the sparsity pattern of the Galerkin coarse-grid operators and high overall computational complexity of the multigrid algorithm. In the case of large-scale parallel computing, this usually leads to high communication overhead on coarse grids [23]. Thus, one must often compromise between the quality of these operators and the aggressiveness of coarsening, which affects the rate of convergence and the operator complexity of the algorithm.

An example for the above discussion appears in aggregation-based AMG methods [8, 10, 27, 18, 17], where the coarsening is done by clustering (aggregating) the grid unknowns. In its simplest form of pure (non-smoothed) aggregation (AGG), P and R are 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 in many cases it requires some sort of acceleration on all levels of the hierarchy [19, 20], often requiring a more elaborate recursive structure (usually W-cycles). Due to this inherent weakness, the approach of *Smoothed Aggregation* (SA) [8, 10, 27, 13] 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.

In [29] Wienands and Yavneh propose an alternative to GCA in (2), called *Collocation Coarse Approximation* (CCA). The idea behind CCA is to assume control over the sparsity pattern of A_c by selecting it explicitly. The coefficients of each row of A_c are chosen such that a certain subspace of the algebraically smooth modes is approximated exactly as in GCA. This subspace is determined by a set of local functions that form a good local basis for the algebraically smooth modes. For example, in [29] mainly the monomials $\{1, x, y, x^2, y^2\}$ were used to locally approximate geometrically smooth modes. The CCA approach is shown to work well in a structured-grid environment, but it is still unclear how to automatically choose the coarse-grid sparsity patterns and the basis functions themselves. These limitations prevent CCA from being able to handle unstructured settings, for example.

In this paper, we present an algebraic generalization of CCA, called *Algebraic Collocation Coarse Approximation* (ACCA), which attempts to satisfy these requirements. Our algorithm uses the aggregation framework as a basis platform, by applying adaptive smoothed aggregation (αSA, [8]) to define the transfer operators, but GCA is employed along with the simple AGG for defining a sufficiently sparse non-zero pattern of A_c. Furthermore, we define our basis functions automatically using the lowest eigenmodes of A, which we calculate using a multilevel eigensolver. This approach is inspired by Bootstrap AMG (BAMG) [5], where the nonzero entries of P are calculated in a similar way. This algorithm belongs to a group of adaptive algorithms, including [8, 9, 5], that have a rather expensive setup phase and may fit scenarios were we solve (1) multiple times with the same matrix A and multiple right hand sides $\mathbf{b}$—then, a one-time expensive setup is worthwhile.

1.1. **Preliminary definitions.** Next, we briefly present some definitions and notations of the adaptive aggregation framework that we use throughout this work.

1.1.1. ***Adaptive multigrid.*** It is known that, when (2) is used, the prolongation must be chosen so as to accurately approximate the algebraically smooth modes. The purpose of adaptive multigrid methods [8, 10, 9] is to treat problems where the local behavior of these smooth modes is not available a priori. Their main idea is to calculate a prototype vector for the smooth modes by

applying relaxations to the homogeneous system $A\mathbf{e} = 0$. Once such a vector is available, we define the prolongation to have it in its range. Once a hierarchy of operators is set, it can be used to further improve the prototype vector by applying V-cycles to the homogeneous system. Finally, the hierarchy is reconstructed using the improved prototype vector. If A in (1) is nonsymmetric, the restriction may be adapted independently of the prolongation [10].

1.1.2. ***Pure (non-smoothed) adaptive aggregation.*** Aggregation methods are defined by a partitioning of the fine-grid index set $\{1, ..., n\}$ into n_c disjoint subsets $\{C_J\}_{J=1}^{n_c}$, which are called aggregates. Given these aggregates and a prototype vector $\mathbf{x}$, we define the simple AGG prolongation, which we denote by P_0:

$$(P_0)_{i,J} = \begin{cases} \frac{x_i}{(x_c)_J} & i \in C_J, \\ 0 & otherwise, \end{cases} \tag{3}$$

where $(x_c)_J = \sqrt{\sum_{j \in C_J} x_j^2}$ is the J-th entry of a coarse vector $\mathbf{x}_c$, which satisfies $\mathbf{x} = P_0\mathbf{x}_c$ ($\mathbf{x}$ is in the range of P_0). By this definition, the columns of P_0 are orthogonal and ${P_0}^T P_0 = I_{n_c}$, where I_{n_c} is the identity matrix of size n_c. For the AGG restriction operator R_0 we use $R_0 = P_0^T$, and we also define a similar *secondary* restriction operator $\hat{R} \in \mathbb{R}^{n_c \times n}$, that is used to generate coarse versions of fine-grid vectors. For example, in a $\mathcal{C}\backslash\mathcal{F}$ splitting framework, and also in [29], $\hat{R}$ is an injection operator. In our case, we want to choose $\hat{R}$ such that $\|\mathbf{x} - P_0(\hat{R}\mathbf{x})\|_2$ is minimized. This leads to

$$\hat{R} = {P_0}^T, \tag{4}$$

which is the Moore-Penrose pseudo-inverse of P_0, given that ${P_0}^T P_0 = I_{n_c}$.

In this work, we define our aggregation by the Bottom-Up approach used in [24, 25], but other aggregation-based coarsening methods such as [27, 18] may also be suitable for our algorithm.

1.1.3. ***Smoothed aggregation operators.*** As mentioned above, SA methods aim at improving the convergence properties of the simple AGG by smoothing the transfer operators P_0 and R_0, which are henceforth dubbed *tentative operators*. More precisely, the SA prolongation is often defined by

$$P = P_\omega = (I - \omega D^{-1} A)P_0, \tag{5}$$

where the matrix D is the diagonal of A, ω is a damping parameter (we use $\omega = 0.75$), and $I - \omega D^{-1}A$ is the error propagation matrix associated with damped-Jacobi relaxation. If A is symmetric then $R = P^T$ is usually used, and if not, then R may be [14]

$$R = R_\omega = {P_0}^T(I - \omega A D^{-1}). \tag{6}$$

2. Algorithm description

Given the problem matrix A, multigrid algorithms employing GCA define suitable P and R, and then use (2) to construct A_c. Therefore, the sparsity pattern of A_c is dictated by P and R. The main idea of CCA is to separate the issue of the coarse-grid operator sparsity pattern from the transfer operators R and P. This allows us to fix the sparsity pattern of A_c and, independently, use high quality transfer operators P and R. In [29], A_c is constructed in such a way that it yields an exact approximation of the GCA operator, RAP, with respect to certain basis vectors. Assume that the current error, $\mathbf{e}$, is in the range of P, i.e., $\mathbf{e} = P\mathbf{e}_c$. Also, assume that A_c is constructed such that $A_c\mathbf{e}_c = RAP\mathbf{e}_c$, implying that $\mathbf{e}_c$ can be represented as a linear combination of the basis vectors that are used to define A_c. Then, the two-level cycle of CCA eliminates the error $\mathbf{e}$ [29]:

$$\begin{array}{rcccccl} \mathbf{e}_{new} & = & \left[I - P(A_c)^{-1}RA\right]\mathbf{e} & = & [I - P(A_c)^{-1}RA]P\mathbf{e}_c & & \\ & = & [P - P(A_c)^{-1}RAP]\mathbf{e}_c & = & P(A_c)^{-1}[A_c - RAP]\mathbf{e}_c & = & 0. \end{array} \tag{7}$$

With this motivation, the CCA algorithm typically chooses the monomials $\{1, x, y, x^2, y^2, ...\}$ as basis functions, because they are able to represent accurately the *geometrically* smooth error modes that comprise $\mathbf{e}$ and $\mathbf{e}_c$. To accomplish this for *algebraically* smooth errors, in ACCA we compute the k^e lowest eigenvectors of A (the ones corresponding to the smallest eigenvalues), where k^e is equal to or larger than the maximal number of non-zeros per row in A_c. These eigenmodes, that comprise the near null-space of A, are known to be related to the sought *algebraically* smooth modes [8, 5, 9].

Our algorithm starts with a preliminary phase (Phase I), where we calculate the set of global basis vectors—approximations for the lowest eigenvectors of A. For this we build a simple aggregation hierarchy (based on subsection 1.1.2 and GCA), and use it to approximate the k^e eigenvectors by applying the PINVIT AMG (PAMG) eigensolver of [2], together with the accelerated AGG K-cycle algorithm of [19]. Once the basis vectors are set, we use a second phase (Phase II), where we define the αSA operators (5)-(6), using lowest of the eigenmodes as a new prototype for our prolongation. Next, we define the ACCA operator A_c to have the sparsity pattern of AGG produced by GCA, and by using the operators R, P, and the basis vectors, we calculate A_c's elements according to the ACCA approach elaborated in this section. This process is repeated recursively, resulting in the ACCA hierarchy.

A detailed description of both phases is given next, but we first note that, as in the αSA framework of [8], we first diagonally scale the fine-level matrix A. That is, we effectively solve an equivalent system $D^{-1/2}AD^{-1/2}\mathbf{y} = D^{-1/2}\mathbf{b}$, and so we henceforth assume that the system (1) is diagonally scaled.

2.1. **Phase I: Multilevel Eigensolver for generating global basis vectors.** In this work, we assume that we should not use high-quality smoothed-aggregation transfer operators along with GCA, because doing so may be inappropriate in our framework, for example, due to an unacceptable increase in the operator complexity of our algorithm. However, to use our method, we need to calculate a near null-space vector set, which is in principle at least as hard as solving (1). Although this can be done by non-multigrid methods (such as Lanczos [21]), we prefer to use a multigrid method that may lead to mesh-independent performance. Inspired by [5], we first build a relatively cheap GCA multigrid hierarchy and use it to develop the multilevel eigensolver. Most such eigensolvers, including the one we use [2], apply V-cycles iteratively. Since we allow only complexity-friendly (low-quality) transfer operators, we need to accelerate our V-cycles to have the best possible convergence rate when calculating our basis vectors.

As noted above, we build a GCA AGG hierarchy as in Section 1.1.2, and use it along with the PAMG eigensolver of [2] together with the K-cycle algorithm of [19]. As described in [2, 5], when coarsening a symmetric eigenproblem by GCA, we get a generalized eigenproblem

$$P_0^T A P_0 \mathbf{v} = \lambda P_0^T P_0 \mathbf{v} = \lambda T \mathbf{v} \tag{8}$$

where T is a positive definite matrix. In this work, $T = P_0^T P_0 = I_{n_c}$, so we have a regular eigenproblem on all grids. Nevertheless, we keep T in our algorithm description for generality.

In the first step of our algorithm, we create an initial prototype vector $\mathbf{x}$ by applying several Gauss-Seidel relaxations on $A\mathbf{x} = 0$ starting with $\mathbf{x} = 1$ (we do 20 relaxations). From this point, Algorithm 1 describes a two-level algorithm for the creation of basis vectors in our setup Phase I of ACCA. The multilevel algorithm is obtained by recursion, where we make sure that our coarsest grid size is equal to or larger than k^e.

In Step 2 of Algorithm 1, we apply the Bottom-Up aggregation procedure of [25], and define the AGG operator P_0 in (3). Then, we coarsen A and $\mathbf{x}$ (Step 3), and go down the multigrid hierarchy. Once the problem is small enough, we stop coarsening and the AGG hierarchy is set. Following that, we apply a direct coarsest level solution and, as we traverse up the multigrid hierarchy, we apply the PAMG eigensolver together with K-cycles, in a Full Multigrid (FMG) style [11] in Steps

Algorithm: $\{\mathbf{v}^i\}_{i=1}^{k^e}$ **= Two-Level-AGG-PAMG-Eigensolver**$(A, T, \mathbf{x}, k^e, \nu_0)$
% Initial prototype vector: $\mathbf{x}$*. Number of eigenmodes needed:* k^e*.*
% Number of relaxations at each level: ν_0*. (We use* $\nu_0 = 3$*)*
% Kcycle for solving $A\mathbf{x} = \mathbf{b}$*: Kcycle*$(A, \mathbf{x}, \mathbf{b}, \nu_1, \nu_2)$ *(2 accelerated W-cycles)*

(1) Apply ν_0 relaxations on $A\mathbf{x} = 0$.
(2) Define AGG P_0 based on A and $\mathbf{x}$.
(3) Apply GCA Coarsening: $A_c = P_0^T A P_0$, $\mathbf{x}_c = \hat{R}\mathbf{x}$, $T = P_0^T P_0$.
(4) Solve the coarse-grid generalized eigenproblem:
    Find the lowest k^e eigenpairs: $\{(\mathbf{v}_c^i, \lambda_c^i)\}_{i=1}^{k^e}$ s.t $A_c\mathbf{v}_c^i = \lambda_c^i T\mathbf{v}_c^i$
(5) Prolong the eigenmodes: $\mathbf{v}^i = P_0\mathbf{v}_c^i$.
(6) Post-smoothing and approximation of eigenvalues:
    for $i = 1, ..., k^e$: Apply ν_0 relaxations on $A\mathbf{v}^i = 0$.
    $\{(\mathbf{v}^i, \lambda^i)\}_{i=1}^{k^e} \leftarrow$ **RitzProjection**$(A, T, \{\mathbf{v}^i\}_{i=1}^{k^e})$.
(7) Apply several PAMG Eigensolver iterations:
    for $i = 1, ..., k^e$: $\mathbf{v}^i \leftarrow$ **Kcycle**$(A, \mathbf{v}^i, \lambda^i T\mathbf{v}^i, 1, 1)$.
    $\{(\mathbf{v}^i, \lambda^i)\}_{i=1}^{k^e} \leftarrow$ **RitzProjection**$(A, T, \{\mathbf{v}^i\}_{i=1}^{k^e})$.

Algorithm 1: Setup phase I - Calculation of Basis Vectors

5-7. Step 6 is where we address the issue of having non-optimal P_0 and R_0. In [2, 5], where high-quality AMG operators are used, the eigenvalues on each level are initialized with the corresponding eigenvalues from the next coarser level (that is, we go from Step 5 to Step 7 directly with $\lambda^i = \lambda_c^i$). In [5], the similarity of eigenvalues between consecutive coarser levels is the convergence criterion of the BAMG process. Since P_0 and R_0 are of low quality, we do not have such similarity—this is in fact the reason why these operators are of low quality. To address this issue, we first smooth the basis vectors with zero RHS (without using a $\lambda^i\mathbf{v}^i$ RHS), so noise produced by the prolongation is significantly damped, and the newly approximated λ's in Step 6 are more accurate. The Ritz projection procedure in Algorithm 1 is rather standard and can be found in the classical [6].

Our convergence criterion for Step 7 of the multilevel version of Algorithm 1 is given by

$$\max_{i=1,2,...,k^e} \left\{ \|A\mathbf{v}^i - \lambda^i\mathbf{v}^i\|_\infty \right\} \leq \epsilon, \tag{9}$$

where all $\mathbf{v}^i$'s are normalized and ϵ is problem size dependent. A bigger problem requires a smaller ϵ. When this criterion is satisfied at the finest level, we achieve a good approximation to the k^e lowest eigenvectors of A, and phase I is completed.

2.1.1. ***An "economic" Phase I.*** The above process may be expensive, especially because of the rather strict convergence criterion in (9). Yet, as in [5], there is no theoretical requirement in our algorithm for having exact low eigenmodes as basis vectors. All we need is a set of algebraically smooth vectors that are locally independent, and we choose them to be our approximation of the lowest eigenmodes of A. Only one of these eigenmodes, which will be used as a prototype vector for improving our prolongation, needs to be highly accurate. Hence, in practice we use an "economic" phase I, where on the finest level we improve only the lowest eigenmode by PAMG, that is, we use $k^e = 1$ in Step 7 and in the criterion (9).

2.2. **Phase II: building the CCA hierarchy.** For describing our algorithm we start with some definitions and notation for our second setup phase. Following that, we describe both the original CCA and our new ACCA algorithms.

2.2.1. ***Definitions and notation.*** Using the transfer operators (3), (5), and (6), we denote the smoothed and non-smoothed GCA aggregation coarse operators by

$$A_c^0 = P_0^T A P_0, \quad A_c^\omega = R_\omega A P_\omega, \tag{10}$$

and by $(A_c^0)_i$ and $(A_c^\omega)_i$ we denote the i-th row vectors of A_c^0 and A_c^ω, respectively. Next, let $\mathcal{N}$ denote the coarse-grid index set $\{1, ..., n_c\}$, and let the indices of the non zeros at lines i of (10) be

$$\mathcal{N}_i^0 = \{j : (A_c^0)_{ij} \neq 0\}, \quad \mathcal{N}_i^\omega = \{j : (A_c^\omega)_{ij} \neq 0\}. \tag{11}$$

Also, denote the sizes of the above sets by $n_i^0 = |\mathcal{N}_i^0|$, $n_i^\omega = |\mathcal{N}_i^\omega|$.

Finally, we denote by $T_i^0 \in \mathbb{R}^{n_i^0 \times n_c}$ and $T_i^\omega \in \mathbb{R}^{n_i^\omega \times n_c}$ the simple restriction matrices from the coarse index set $\mathcal{N}$ to the index sets $\mathcal{N}_i^0$ and $\mathcal{N}_i^\omega$ respectively:

$$(T_i^0)_{J,k} = \begin{cases} 1 & k \in \mathcal{N}_i^0 \\ 0 & otherwise \end{cases}, \quad (T_i^\omega)_{J,k} = \begin{cases} 1 & k \in \mathcal{N}_i^\omega \\ 0 & otherwise \end{cases}, \quad k = 1, ..., n_c, \tag{12}$$

where J is the J-th index in $\mathcal{N}_i^0$ and $\mathcal{N}_i^\omega$. For example, the term $T_i^0 \left[(A_c^0)_i^T\right]$ equals, by definition, a dense vector of size n_i^0 comprised of the non-zero entries of the i-th row of A_c^0.

2.2.2. ***The Phase II algorithm.*** Having finished Phase I, we have the k^e global fine-grid basis vectors $\{\mathbf{v}^i\}_{i=1}^{k^e}$ and, henceforth, we also refer to them as the columns of a basis-vector matrix

$$B = \left[\mathbf{v}^1, \mathbf{v}^2, ..., \mathbf{v}^{k^e}\right] \in \mathbb{R}^{n \times k^e}. \tag{13}$$

Using the lowest eigenvector $\mathbf{v}^1$, we define our final αSA transfer operators P_0, P_ω and R_ω by (3), (5), and (6) respectively. Next, we define the ACCA operator A_c to have the same (attractive) sparsity pattern as A_c^0

$$\text{Sparsity}(A_c) \triangleq \text{Sparsity}(A_c^0), \tag{14}$$

and then calculate its entries. Algorithm 2 describes the "outer" part of Phase II using the *calculateACCARow*() procedure that calculates each row of A_c. This process is repeated recursively until n_c is small enough, resulting in the multilevel ACCA hierarchy. The diagonal matrix W that appears in Step 4 will be discussed later.

Algorithm: ACCA-Setup(A, B, ν_0)
% Basis vectors matrix: $B = \left[\mathbf{v}^1, \mathbf{v}^2, ..., \mathbf{v}^{k^e}\right]$
% Number of relaxations at each level: $\nu_0 = 5$.
(1) **for** $i = 1, ..., k^e$: Apply ν_0 relaxations on $A\mathbf{v}^i = 0$.
(2) Define AGG P_0 based on A and $\mathbf{v}^1$.
(3) Define αSA operators P_ω, R_ω.
(4) Define a diagonal weight matrix W
    for $i = 1, ..., k^e$: $W_{ii} = \frac{\|\mathbf{v}^i\|_2}{\|A\mathbf{v}^i\|_2}$
(5) Apply GCA Coarsening: $A_c^0 = P_0^T A P_0$, $A_c^\omega = P_\omega^T A P_\omega$.
(6) Coarsen the basis-vector matrix: $B_c = \hat{R} B$.
(7) Calculate the ACCA operator:
    for $i = 1, ..., n_c$: Apply $(A_c)_i =$ **calculateACCARow**$((A_c^0)_i, (A_c^\omega)_i, B_c, W)$
(8) Recursive Call: **ACCA-Setup**(A_c, B_c, ν_0)

Algorithm 2: Setup Phase II - Calculation of the ACCA Hierarchy

We next describe an explicit construction of the ACCA operator—the *calculateACCARow*() procedure (Step 7). This process is repeated independently for each row and therefore may be

easily implemented in parallel computing scenarios. By way of motivation, we first briefly recall the construction of the original CCA operator as in [29].

2.2.3. ***Construction of the original CCA operator***. In the CCA algorithm of [29], a matrix of basis vectors $B^{(i)} = \left[b_1^{(i)}, b_2^{(i)}, ..., b_{n_i^0}^{(i)}\right] \in \mathbb{R}^{n_c \times n_i^0}$ is generated geometrically for computing the entries of each row i of A_c. These vectors are chosen a priori, typically as the monomial functions $\{1, x, y, x^2, y^2, ...\}$ centered at the grid point i as the origin. Then, motivated by (7), $(A_c)_i$ is defined as the solution of

$$[(A_c)_i - R_i A P]\, B^{(i)} = 0, \tag{15}$$

where R and P are the chosen high quality transfer operators (for ACCA, we use (5)-(6)).

Assuming that the (chosen a priori) non-zero indices of $(A_c)_i$ correspond to a set $\mathcal{N}_i^0$, then by setting $M^{(i)} = T_i^0 B^{(i)} \in \mathbb{R}^{n_i^0 \times n_i^0}$ and $\mathbf{u}^{(i)} = T_i^0 [(A_c)_i^T] \in \mathbb{R}^{n_i^0}$, we can rewrite (15) as a linear system

$$\left(M^{(i)}\right)^T \mathbf{u}^{(i)} = \left(R_i A P B^{(i)}\right)^T = \left((RAP)_i B^{(i)}\right)^T, \tag{16}$$

with n_i^0 unknowns. Note that $\mathcal{N}_i^0$ corresponds to the non-zero indices of $(A_c)_i$ also in ACCA, following (11) and (14).

Solving the system (16) gives the non-zeros of $(A_c)_i$. By the a priori choice of $B^{(i)}$, we can safely say that $M^{(i)}$ is non singular and well-conditioned, and solving (16) is sufficient for computing the nonzeros of $(A_c)_i$. Effectively, in (15)-(16), we use only the lines of $B^{(i)}$ corresponding to the non-zero indices of $(A_c)_i$ and $(RAP)_i$, and so we refer to these as the *local* basis vectors of $B^{(i)}$.

2.2.4. ***Construction of the ACCA operator***. As in Algorithm 2, when calculating all the rows of the ACCA operator A_c, we have the single coarse basis-vector matrix

$$B_c = \hat{R} B \in \mathbb{R}^{n_c \times k^e}, \tag{17}$$

where k^e is equal to or larger than the number of nonzeros in each row i, and B appears in (13). Analogously to CCA, we may calculate $(A_c)_i$ by choosing the first n_i^0 columns of B_c as $B^{(i)}$ and use (15) as in CCA. However, this is a risky approach since now we have no guarantee that the resulting system is well-posed for all rows. The columns of B_c may be *locally* linearly dependent even though they are *globally* independent, and in that case serious defects may occur in A_c.

To solve this issue, instead of (15), in ACCA we use an overdetermined system of equations for calculating $(A_c)_i$, taking all available factors into account. Our solution consists of a weighted Least Squares minimization involving two quadratic terms

$$\left\| [(A_c)_i - (A_c^\omega)_i]\, B^{(i)} \right\|_{2,W}^2 + \beta \phi\, [(A_c)_i] \longrightarrow \min, \tag{18}$$

where $B^{(i)} \in \mathbb{R}^{n_c \times k^e}$ is a basis-vector matrix, the weighted squared norm $\|\mathbf{v}\|_{2,W}^2$ is given by $\langle \mathbf{v}, W\mathbf{v} \rangle$ for any vector $\mathbf{v}$, and W is a diagonal weight matrix defined in Step 4 of Algorithm 2; ϕ is another quadratic function for determining $(A_c)_i$, and $\beta > 0$ is a balancing scalar between the two terms. If we have $k^e = n_i^0$, and $\beta = 0$, then (18) reduces to (15), but with a different $B^{(i)}$. We now discuss the choice of $B^{(i)}$, and describe ϕ later.

Let $T_i \in \mathbb{R}^{n_i \times n_c}$ be a simple restriction matrix that, similarly to (12), corresponds to the index set $\mathcal{N}_i = \mathcal{N}_i^0 \cup \mathcal{N}_i^\omega$, with $|\mathcal{N}_i| = n_i$. Then, $G \triangleq T_i B_c \in \mathbb{R}^{n_i \times k^e}$ is the sub-matrix of basis vectors that corresponds to the effective indices in the first term of (18). This way, every sampled basis vector (a column of G) has a local scale. This scale is somewhat irrelevant, since it is known that only the ratios between strongly connected elements is the important characteristic of smooth error modes [24]. In the LS minimization in (18), such a scale may act as a fake weight on the local basis vectors, causing irrelevant preference to certain equations. To solve that, we simply normalize the

columns of G. In our next step we apply a (stabilized) Gram Schmidt orthogonalization to G to expose the local characteristics of our basis vectors and by that improve the numerical stability of our minimization. We do not apply normalization in this Gram Schmidt process in order to prevent situations were errors are amplified in cases where G has almost linearly dependent columns. The above process and the final definition of $B^{(i)}$ is described in Steps 1-4 of Algorithm 3.

Procedure: calculateACCARow$((A_c^0)_i, (A_c^\omega)_i, B_c, W)$
% Restricted Basis vectors matrix: $B_c \in \mathbb{R}^{n_c \times k^e}$
% Diagonal weight matrix: W.
(1) Define $G = T_i B_c \in \mathbb{R}^{n_i \times k^e}$
(2) Normalize the columns of G.
(3) Apply an orthogonalization process to G without normalizing its columns.
(4) Define $B^{(i)} = T_i^T G \in \mathbb{R}^{n_c \times k^e}$ *% zero-filling*
(5) Solve the Least-Squares problem (18) with ϕ as in (19).

Algorithm 3: Calculation of the i-th row of an ACCA operator

The second term of (18) adds available information to our mechanism that does not depend on the basis vectors. For that we use the AGG line $(A_c^0)_i$ together with the fact that A_c^0 is known to have a bad scale but good ratios between entries of each row. For example, for the finite differences discretization of a Laplacian operator, plain 2×2 aggregation using a piece-wise constant P_0 causes a factor 2 scale when compared with the fine-grid operator [7] . This property has led to the idea of accelerating AGG by overcorrection techniques [1, 28, 7, 22]. Inspired by this property, we define ϕ as the quadratic distance function

$$\phi\left[(A_c)_i\right] = \left((A_c)_{ii} - \frac{1}{2}(A_c^0)_{ii}\right)^2 + \sum_{j \in \mathcal{N}_i^0,\, j \neq i} \left((A_c)_{ij} - \frac{(A_c^0)_{ij}}{(A_c^0)_{ii}}(A_c)_{ii}\right)^2, \tag{19}$$

where the first term measures the distance between the diagonal element of the ACCA operator and the scaled (by half) diagonal element of AGG, and the second term aims at making the ratios between the off-diagonals of A_c and its diagonal be equal to the same ratios in the AGG operator A_c^0. The parameter $\frac{1}{2}$ was chosen for all the test cases that we consider in this paper.

Lastly, we define the balancing parameter β in (18) as $\beta = 10^{-2} \min_i\{W_{ii}\}$ (W appears in the weighted norm of (18) and is defined in Step 4 of Algorithm 2). This way, if $B^{(i)}$ has enough information for determining $(A_c)_i$ then the second term ϕ will not have a large influence on the result of the minimization of (18). Otherwise, ϕ makes (18) well-posed. We note that as in [29], the ACCA algorithm does not preserve the symmetry of A in A_c.

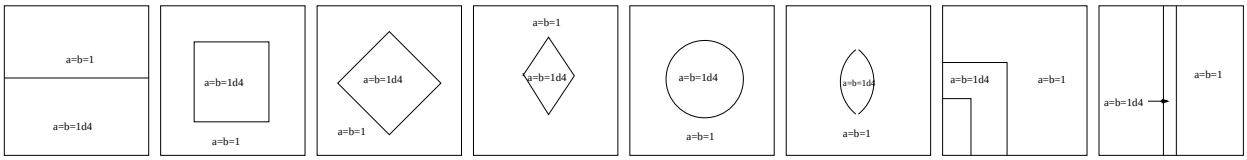
3. Numerical Results

In this section, the efficiency and robustness of the ACCA approach is investigated and compared to both αSA and AGG. We consider the two-dimensional diffusion equation on the unit square with Dirichlet boundary conditions

$$\begin{aligned} &-(a(x,y)u_x(x,y))_x - (b(x,y)u_y(x,y))_y = f(x,y), && (x,y) \in \Omega = (0,1)^2 \\ &u(x,y) = g(x,y), && (x,y) \in \partial\Omega \end{aligned} \quad . \tag{20}$$

The diffusion problem is discretized by the cell-centered finite differences method, leading to a five-point stencil on a discrete domain Ω_h with regular mesh size $h_x = h_y = h$. We present results for $h = 1/256$, $h = 1/512$, and $h = 1/1024$ (i.e., for problem sizes 256^2, 512^2, and 1024^2).

We consider a collection of classical test cases that covers a variety of possible coefficient inhomogeneities, including jumps that are aligned with the grid lines and jumps that are not aligned



(a) $TC1$ (b) $TC2$ (c) $TC3$ (d) $TC4$ (e) $TC5$ (f) $TC6$ (g) $TC7$ (h) $TC8$

FIGURE 1. Types of jumping coefficients problems.

with the grid lines (most of the problems appear in [29]). With each grid geometry, we used a ratio of $1 : 10^4$ of the coefficient discrepancy between the outer and inner shape. The problems are denoted by

TC1: Horizontal jump,
TC2: Inhomogeneous square,
TC3: Inhomogeneous 90° diamond,
TC4: Inhomogeneous narrow diamond,
TC5: Inhomogeneous circle,
TC6: Inhomogeneous ellipse,
TC7: Inhomogeneous L-shaped strip,
TC8: Inhomogeneous narrow (0.04) vertical strip,

and the different choices of shapes and coefficients a, b are specified in Figure 1.

We use V(2,2) cycles, with two pre- and post- Gauss Seidel relaxations for all methods. For all cases, we start with a random initial guess and solve a homogeneous problem where the right-hand-side f and boundary condition g are set to zero. Then, the exact solution is known to be zero and, thus, the numerical experiments are not limited by numerical accuracy. Hence, we compute the asymptotic convergence factor γ by the geometric mean convergence factor per cycle, computed over cycles number 45 to 50. We also compare each method's operator complexity, *op.c.*, 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. Finally, the effective convergence factor, γ_{eff}, is defined as $\gamma_{eff} = \gamma^{1/op.c}$ and is a measure for comparing the effectiveness of the cycles.

All three methods use the calculated lowest eigenvector $\mathbf{v}^1$ as the prototype vector for constructing P and R ((3) for AGG, and (5)-(6) for ACCA and αSA). For AGG and αSA, this vector is calculated using the same Algorithm 1 of ACCA, with $k^e = 1$. For ACCA, we use $k^e = 6$ basis vectors. For all methods, in (9) we use $\epsilon = 10^{-6}, 10^{-7}, 10^{-8}$ for $n = 256^2, 512^2, 1024^2$, respectively.

Table 1 compares the performance of ACCA with AGG and αSA as solvers for the different test cases. Most noticeable is the attractive (and almost identical) operator complexity of ACCA and AGG in comparison with αSA. The convergence factors γ of ACCA are much closer to those of αSA than to those of AGG, with αSA having the best γ values, as expected. The table shows that the performance of ACCA in the problems with grid aligned jumps (TC1,TC2,TC7,TC8) is better than its performance with the rest of the problems (non grid aligned jumps: TC3-TC6). Overall, for all the problems presented, the *effective* convergence factors γ_{eff} of ACCA are comparable to those of αSA, and a lot better than those of AGG.

Interestingly, the problem TC3 is the one reported problem that the original CCA algorithm failed to handle. It is the only problem with non-grid aligned jumps in [29], and so it seems that the ACCA algorithm may be more robust for such problems than CCA. In addition, the structured grids are not preserved on the coarser grids of some of the above problems, indicating that ACCA may handle similar problems on unstructured settings.

Problem	n	*AGG*			ACCA			αSA		
		γ	op.c.	γ_{eff}	γ	op.c.	γ_{eff}	γ	op.c.	γ_{eff}
TC1	256^2	0.921	1.341	0.940	0.161	1.341	0.256	0.072	2.572	0.360
	512^2	0.950	1.337	0.963	0.187	1.337	0.285	0.120	2.633	0.447
	1024^2	0.963	1.335	0.972	0.187	1.335	0.285	0.256	2.680	0.602
TC2	256^2	0.913	1.343	0.934	0.216	1.343	0.320	0.085	2.561	0.382
	512^2	0.949	1.338	0.961	0.235	1.339	0.339	0.165	2.637	0.504
	1024^2	0.968	1.336	0.976	0.256	1.336	0.360	0.283	2.687	0.625
TC3	256^2	0.941	1.445	0.959	0.424	1.443	0.552	0.130	2.949	0.500
	512^2	0.974	1.440	0.982	0.499	1.441	0.617	0.159	3.017	0.544
	1024^2	0.974	1.436	0.982	0.541	1.436	0.652	0.256	3.082	0.642
TC4	256^2	0.941	1.449	0.959	0.473	1.448	0.597	0.156	2.885	0.525
	512^2	0.973	1.443	0.982	0.497	1.443	0.616	0.230	2.968	0.609
	1024^2	0.975	1.440	0.982	0.599	1.440	0.701	0.338	3.019	0.698
TC5	256^2	0.926	1.412	0.947	0.391	1.413	0.515	0.092	2.812	0.428
	512^2	0.969	1.419	0.978	0.478	1.419	0.594	0.116	2.953	0.482
	1024^2	0.975	1.408	0.982	0.588	1.408	0.686	0.167	2.966	0.547
TC6	256^2	0.933	1.424	0.952	0.402	1.425	0.527	0.097	2.904	0.447
	512^2	0.969	1.415	0.978	0.475	1.415	0.591	0.114	2.929	0.476
	1024^2	0.974	1.409	0.982	0.533	1.409	0.640	0.201	2.971	0.583
TC7	256^2	0.909	1.337	0.931	0.172	1.335	0.268	0.071	2.529	0.351
	512^2	0.962	1.336	0.971	0.246	1.336	0.350	0.091	2.623	0.402
	1024^2	0.967	1.334	0.975	0.350	1.334	0.455	0.123	2.672	0.457
TC8	256^2	0.929	1.341	0.947	0.204	1.340	0.306	0.073	2.575	0.362
	512^2	0.962	1.336	0.971	0.294	1.336	0.400	0.087	2.638	0.396
	1024^2	0.967	1.335	0.975	0.389	1.335	0.493	0.413	2.686	0.719

TABLE 1. Performance of ACCA, AGG and αSA V(2,2) cycles as solvers for different problems.

4. Conclusions and future work

In this paper we have presented a new algorithm which is an algebraic generalization for the collocation coarse approximation (CCA) multigrid approach. The main advantage of both the CCA and ACCA approaches is that 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 two approaches particulary 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 having high-quality transfer operators. It computes the non-zero entries of the coarse grid operator using a small set of low-energy eigenvectors, by a weighted least squares process. Numerical experiments show that the algorithm has promising capabilities for 2D diffusion problems. It is quite scalable and robust and may be advantageous in cases where strict sparsity constraints prevent us from using high-quality GCA operators. We expect that our algorithm will have similar performance for such problems also on unstructured settings.

Further research may be aimed at improving the scalability and performance ACCA on harder problems, including non-symmetric problems. Also, investigating the theoretical aspects of the approach may lead to a better way to define the basis vectors of the algorithm, and improve its ability to solve various of other problems.

References

[1] R. Blaheta, *A multilevel method with overcorrection by aggregation for solving discrete elliptic problems*, J. Comput. Appl. Math., 24 (1988), pp. 227–239.

[2] A. Borzi and G. Borzi, *Algebraic multigrid methods for solving generalized eigenvalue problems*, Int. J. Numer. Meth. Eng., 65 (2006), pp. 1186–1196.

[3] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Mathematics of Computation, 31 (1977), pp. 333–390.

[4] ———, *1984 multigrid guide with applications to fluid dynamics*. Monograph, GMD-Studie 85, GMD-FIT, Postfach 1240, D-5205, St. Augustin 1, West Germany, 1985.

[5] A. Brandt, J. Brannick, K. Kahl, and I. Livshits, *Bootstrap amg*, SIAM J. Sci. Comput., 33 (2011), pp. 612–632.

[6] A. Brandt, S. McCormick, and J. Ruge, *Multigrid methods for differential eigenproblems*, SIAM. J. Stat. Sci. Comput., 4 (1983), pp. 655–684.

[7] A. Brandt and I. Yavneh, *Accelerated multigrid convergence and high-reynolds recirculating flows*, SIAM J. Sci. Comput., 14 (1993), pp. 607–626.

[8] M. Brezina, R. Falgout, S. Maclachlan, T. Manteuffel, S. McCormick, and J. Ruge, *Adaptive smoothed aggregation (α SA)*, SIAM J. Sci. Comput., 25 (2004), pp. 1896–1920.

[9] ———, *Adaptive algebraic multigrid*, SIAM J. Sci. Comput., 27 (2006), pp. 1261–1286.

[10] M. Brezina, T. Manteuffel, S. McCormick, J. Ruge, and G. Sanders, *Towards adaptive smooth aggregation (α SA) for nonsymmetric problems*, SIAM J. Sci. Comput., to appear (2009).

[11] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, SIAM, second ed., 2000.

[12] R. D. Falgout, *An introduction to algebraic multigrid*, IEEE: Computing in Science and Engineering, 8 (2006), pp. 24–33.

[13] H. Guillard, A. Janka, and P. V. Ek, *Analysis of an algebraic petrov galerkin smoothed aggregation multigrid method*, Applied Numerical Mathematics, 58 (2008), pp. 1861–1874.

[14] H. De Sterck, T. A. Manteuffel, S. F. McCormick, K. Miller, J. Pearson, J. Ruge, and G. Sanders, *Smoothed aggregation multigrid for markov chains*, Accepted to SIAM J. Sci. Comput., (2009).

[15] W. Hackbusch, *Multi-grid methods and applications*, Springer, Berlin, 1985.

[16] W. Hackbusch and U. Trottenberg, eds., *Multigrid Methods*, Berlin, 1982, Springer-Verlag.

[17] Y. Notay, *Aggregation-based algebraic multilevel preconditioning*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 998–1018.

[18] ———, *An aggregation-based algebraic multigrid method*, Tech. Report Report GANMN 08-02, Universite Libre de Bruxelles, Brussels, Belgium, 2008 (revised 2009).

[19] Y. Notay and P. S. Vassilevski, *Recursive krylov-based multigrid cycles*, 2007.

[20] C. W. Oosterlee and T. Washio, *Krylov subspace acceleration of nonlinear multigrid with application to recirculating flow*, SIAM J. Sci. Comput., 21 (2000), pp. 1670–1690.

[21] Y. Saad, *Numerical methods for large eigenvalue problems, 2^{nd} edition*, SIAM, 2011.

[22] H. D. Sterck, K. Miller, E. Treister, and I. Yavneh, *Fast multilevel methods for markov chains*, Numerical Linear Algebra with Application, 18 (2011), pp. 961–980.

[23] H. D. Sterck, U. M. Yang, Jeffrey, and J. Heys, *Reducing complexity in parallel algebraic multigrid preconditioners*, SIAM J. Matrix Anal. Appl, 27 (2006), pp. 1019–1039.

[24] E. Treister and I. Yavneh, *Square and stretch multigrid for stochastic matrix eigenproblems*, Numerical Linear Algebra with Application, 17 (2010), pp. 229–251.

[25] ———, *On-the-fly adaptive smoothed aggregation multigrid for markov chains*, SIAM Journal on Scientific Computing (SISC), 33 (2011), pp. 2927–2949.

[26] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, London and San Diego, 2001.

[27] P. Vanek, P. V. Ek, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.

[28] P. Vaněk and S. Míka, *Modification of two-level algorithm with overcorrection*, Appl. Math., 37 (1992), pp. 13–28.

[29] R. Wienands and I. Yavneh, *Collocation coarse approximation in multigrid*, SIAM J. Sci. Comput., 31 (2009), pp. 3643–3660.

[30] I. Yavneh, *Why multigrid methods are so efficient*, IEEE: Computing in Science and Engineering, 8 (2006), pp. 12–22.