Safeguarded Anderson acceleration for parametric nonexpansive operators

Michael Garstka, Mark Cannon & Paul Goulart[1]

***Abstract*— This paper describes the design of a safeguarding scheme for Anderson acceleration to improve its practical performance and stability when used for first-order optimisation methods. We show how the combination of a non-expansiveness condition, conditioning constraints, and memory restarts integrate well with solver algorithms that can be represented as fixed point operators with dynamically varying parameters. The performance of the scheme is demonstrated on seven different QP and SDP problem types, including more than 500 problems. The safeguarded Anderson acceleration scheme proposed in this paper is implemented in the open-source ADMM-based conic solver `COSMO`.**

I. INTRODUCTION

Solutions of large convex optimisation problems of the form

$$\begin{array}{ll}\text{minimize} & \frac{1}{2}x^\top Px + q^\top x \\ \text{subject to} & Ax + s = b,\ (x, s) \in \mathbb{R}^n \times \mathcal{K}\end{array} \tag{1}$$

with $P \in \mathbb{S}^n_+$, $q \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $\mathcal{K}$ a convex cone, are vital to numerous application areas, including robust and optimal control, structural design, operations research, and signal processing [1], [2], [3], [4]. Moreover, recent interest in machine learning has seen convex optimisation being used in many novel applications, including neural network verification against adversarial attacks [5], sparse principal component analysis [6], graph clustering [7], and kernel matrix learning [8]. Many of these applications require the solution of very large-scale problem instances, which presents a challenge for established interior-point solution methods (IPMs). This is because IPMs solve a Newton system at each iteration that grows with the problem dimension n and has a per iteration computational cost of $\mathcal{O}(n^3)$. This drawback led to a renewed interest in first-order methods (FOMs) that trade-off moderate accuracy solutions for a lower per-iteration computational cost, allowing them to solve larger problems. Popular FOMs that have been developed into solver packages include the *Alternating Direction Method of Multipliers (ADMM)* [9], [10], [11], *Douglas-Rachford splitting* [12], and the *Augmented Lagrangian method* [13]. Moreover, the *proximal gradient method* [14] is a popular tool to design custom solution algorithms.

The minimisation problem solved by FOMs can be recast as the problem of finding a fixed point of a nonexpansive operator $F\colon \mathcal{D} \subseteq \mathbb{R}^n \to \mathbb{R}^n$

$$v = F(v), \tag{2}$$

where $v \in \mathbb{R}^n$. Several recent publications consider acceleration methods for FOMs (see the survey in [15]). Well-known acceleration methods based on the gradient descent method include *Nesterov's accelerated gradient method*, *accelerated proximal gradient method* and the *Heavy ball method*. However, all of these methods require differentiability of at least part of the objective of the underlying optimisation problem.

A line search method for general nonexpansive operators is developed in [16]. This searches in the direction of the fixed-point residual and takes larger steps than classical line search methods. Although it is possible to skip many iterations with this approach, it requires the operator F to be evaluated at each candidate point, incurring the same cost as a full iteration. Therefore, this method is only practical if the operator can be expressed in the form $F(v) = F_2\big(F_1(v)\big)$ where F_2 is cheap to evaluate and F_1 is affine, which allows many points in the search direction to be checked cheaply.

Many acceleration approaches view (2) as the problem of finding the zeros $0 \in r(v)$ of the residual operator $r(v) = F(v) - v$, and apply Newton's method due to its fast practical asymptotic convergence. For example, in [17] a semismooth Newton method is applied to the fixed-point operator that is used in the `SCS` [12] solver. The method assumes a semismooth operator and relies on expensive line search steps. Similarly, the *SuperMann* acceleration scheme [18] globalizes fixed-point iterations of non-expansive operators and enjoys superlinear convergence under certain conditions. This approach uses a limited memory Broyden method to approximate the Jacobian with Powell regularisation to keep the Jacobian approximation non-singular. The original fixed point iteration is employed if the accelerated candidate point does not provide an improved solution estimate.

A similar approach uses *Anderson acceleration* (AA) [19], which does not require a line search, to update the Jacobian approximation instead of using Newton's or Broyden's method. The method has been applied to electronic structure computation, known as *Pulay mixing* [20], *direct inversion in the iterative subspace (DIIS)* [21], and *Anderson mixing*, but it has only recently gained attention in the optimisation community (e.g. [22], [23]).

The AA algorithm uses a combination of past iterates to find the accelerated point, where the weights are determined by minimizing a weighted sum of past residuals. Unfortunately, convergence guarantees for AA are available only if additional assumptions such as contractivity [24], linearity [25], or differentiability [26] are made on the operator or on the memory-length of the scheme. However, FOMs for convex conic optimisation problems have none

[1]The authors are with the Department of Engineering Science, University of Oxford, Oxford, OX1 3PJ, UK. Email: {michael.garstka, mark.cannon, paul.goulart}@eng.ox.ac.uk

of these properties, and for large problems only a limited memory variant of AA is economical. Moreover, Mai and Johansson [27] prove that the limited-memory AA cannot guarantee global convergence. Thus, additional safeguarding measures are required. An implementation to safeguard AA in combination with the Douglas-Rachford method employed by SCS is discussed in [28]. This combines (type-I) AA steps with the execution of the original algorithm whenever the residual decreases sufficiently. In order to ensure a non-singular Jacobian, a Gram-Schmidt orthogonalization strategy is used, and a *rolling-memory* approach is also employed.

Contributions of this paper:

1. We design a safeguarding mechanism for AA based on a relaxed non-expansiveness condition on the norm of the residual operator.
2. We investigate a restarted limited-memory scheme that allows the use of a solver algorithm whose representation is a parametric operator and which relies periodically on non-accelerated steps for infeasibility detection.
3. We provide evidence from more than 500 QP and SDP test problems demonstrating the performance of our implementation against the non-accelerated operator. The mean number of iterations for each problem set is reduced by a factor between 1.7 and 8.5 and the mean solve time by a factor of up to 6 for higher accuracy solutions. We also show that the additional time required to calculate the Anderson directions can be kept between 3% to 15% for large QPs and SDPs. Thus, the results make a particularly strong case for using AA to solve SDPs and large QPs.

Outline: In Section II we review the relationship between solving a convex conic problem with a FOM and applying a fixed point iteration to a particular operator. Then we introduce the classical AA method and explain how it can be viewed as a multisecant Broyden's method. In Section III we detail the design decisions of our safeguarding scheme for AA. Section IV shows benchmark results of different variants of the COSMO solver: the classic algorithm, the accelerated algorithm, and the safeguarded *and* accelerated algorithm. Section V concludes the paper.

Notation: Denote the space of real numbers $\mathbb{R}$, the n-dimensional real space $\mathbb{R}^n$, and denote a convex proper cone as $\mathcal{K}$. The identity operator is denoted as Id. We say that an operator $F\colon \mathcal{D} \subseteq \mathbb{R}^n \to \mathbb{R}^n$ is:

a) *nonexpansive* if $\|Fx - Fy\| \leq \|x - y\|$ for all $x, y \in \mathbb{R}^n$;
b) *α-averaged* if there exists a nonexpansive operator $G\colon \mathbb{R}^n \to \mathbb{R}^n$ such that $F = (1-\alpha)\mathrm{Id} + \alpha G$ for $\alpha \in (0, 1)$;
c) *firmly nonexpansive* if it is $\frac{1}{2}$-averaged.

The scaled *proximal operator* of a convex, closed and proper function $f\colon \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ with $\gamma > 0$ is given by

$$\mathrm{prox}_{\gamma f}(v) \coloneqq \underset{y}{\mathrm{argmin}}\{f(y) + \tfrac{1}{2\gamma}\|y - v\|_2^2\}. \tag{3}$$

Denote the *reflected proximal operator* as $R_{\gamma f}(v) \coloneqq (2\,\mathrm{prox}_{\gamma f} - \mathrm{Id})(v)$. We note that the proximal operator (3) is firmly nonexpansive and the reflected proximal operator is nonexpansive [29].

II. BACKGROUND

Before introducing the AA method, we review the relationship between solving an optimisation problem via a FOM and finding the fixed points of a firmly nonexpansive operator.

A. FOMs as fixed-point iterations

In this paper we propose a safeguarded AA method to improve the slow convergence of FOMs when solving large convex optimisation problems of the form (1). A popular FOM to solve (1) to moderate accuracy is *Douglas-Rachford splitting (DRS)*. This solves problems of the form

$$\text{minimize} \quad f(v) + g(v), \tag{4}$$

where both $f\colon \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ and $g\colon \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ are convex, closed, and proper. It is well known that the Fenchel dual of (1) is in the form of (4). Applying DRS to the problem gives the following algorithm

$$z^k \coloneqq \mathbf{prox}_{\gamma f}(v^k) \tag{5a}$$
$$x^k \coloneqq \mathbf{prox}_{\gamma g}(2z^k - v^k) \tag{5b}$$
$$v^{k+1} \coloneqq v^k + (x^k - z^k) \tag{5c}$$

with $\gamma > 0$, or using the more compact operator form

$$v^{k+1} \coloneqq F(v^k) = \left(\frac{1}{2}\mathrm{Id} + \frac{1}{2}R_{\gamma f}R_{\gamma g}\right)(v^k), \tag{6a}$$
$$z^{k+1} \coloneqq \mathbf{prox}_{\gamma f}(v^{k+1}). \tag{6b}$$

Notice that (6a,b) has the fixed-point operator form (2). Moreover F is a $\frac{1}{2}$-averaged iteration of the nonexpansive operator $R_{\gamma f}R_{\gamma g}$ and therefore firmly-nonexpansive. Hence applying the *Picard iteration* (2) will converge linearly to a fixed point (if one exists) that is coincident with the optimal solution of the underlying optimisation problem [30].

We consider the fixed-point operator F in (6a) as an abstraction of the actual FOM solver used to solve (1).

B. Anderson acceleration

AA calculates an accelerated candidate point v_k^{acc} as a weighted combination of $m_k + 1$ previous iterates,

$$v_k^{\mathrm{acc}} = \sum_{i=0}^{m_k} \alpha_k^i F(v_{k-m_k+i}). \tag{7}$$

Anderson's main idea was to choose the weights $\alpha_k = \left[\alpha_k^0, \ldots, \alpha_k^{m_k}\right]$ by minimizing the norm of past residual vectors:

$$\begin{array}{ll} \text{minimize} & \|R_k\alpha_k\|_2^2 \\ \text{subject to} & \mathbf{1}_{m_k+1}^\top \alpha_k = 1, \end{array} \tag{8}$$

where $R_k = \begin{bmatrix} r_{k-m_k} & \cdots & r_k \end{bmatrix}$ is the matrix of past residual vectors, $r_k \coloneqq r(v_k)$. Eyert [31] established a connection between AA and multisecant Broyden's method which allows AA to be considered a Quasi-Newton method. To clarify the relationship one can make a change of variables [23] by defining $\eta = (\eta^0, \ldots, \eta^{m_k-1}) \in \mathbb{R}^{m_k}$, which implicitly encodes the constraint in (8)

$$\alpha^0 = \eta^0,\ \alpha^i = \eta^i - \eta^{i-1}, \ldots,\ \alpha^{m_k} = 1 - \eta^{m_k-1}. \tag{9}$$

Next, η is substituted for α and we use the differences between iterates $\Delta v_k = v_{k+1} - v_k$, $\Delta r_k = r(v_{k+1}) - r(v_k)$ to rewrite (7) and (8) as

$$v_k^{\text{acc}} = v_k - r_k - (\mathcal{V}_k - \mathcal{R}_k)\eta_k, \tag{10}$$

$\mathcal{V}_k = [\Delta v_{k-m_k}, \dots, \Delta v_{k-1}]$, $\mathcal{R}_k = [\Delta r_{k-m_k}, \dots, \Delta r_{k-1}]$. The Anderson coefficients η are calculated by solving the least-squares problem $\|r_k - \mathcal{R}_k\eta\|_2^2$. Assuming $\mathcal{R}_k$ is full-rank we can substitute the solution for η and write AA as a multisecant Broyden type-II method:

$$v_k^{\text{acc}} = v_k - H_k^{II} r_k \tag{11}$$

where $H_k^{II} = I + (\mathcal{V}_k - \mathcal{R}_k)(\mathcal{R}_k^\top \mathcal{R}_k)^{-1}\mathcal{R}_k^\top$ can be viewed as the rank-m_k update formula to approximate the inverse of the Jacobian of the residual operator. A corresponding type-I method is obtained by using the approximation update rule $H_k^I = I + (\mathcal{V}_k - \mathcal{R}_k)(\mathcal{V}_k^\top \mathcal{R}_k)^{-1}\mathcal{V}_k^\top$.

III. SAFEGUARDED ANDERSON ACCELERATION

We make no assumptions about the smoothness of the operator F, which means global convergence of the unaltered acceleration method cannot be guaranteed. Section II explains how Anderson acceleration can be seen as a Broyden's method which starts with I as an estimate for the Jacobian inverse H_k of the operator and then uses rank-1 updates to modify it based on input and output differences of the residual operator. A source of instability of this Jacobian update step is the tendency of FOMs to eventually reach regimes of slow convergence. This causes the vectors used in the updates to become nearly collinear and leads to poor conditioning of the Jacobian approximation. Like other Quasi-Newton methods, AA also suffers from a lack of global convergence guarantees. Therefore further safeguarding measures are needed, usually to check the quality of candidate iterates before they are accepted.

Solvers based on FOMs rarely stick to the classic algorithm but instead use heuristic rules that adapt some of the algorithm's parameters. Thus, the corresponding operator will change at different points in the solution process. We denote the parameter-dependent operator F_ρ with changing parameter vector ρ. Our AA scheme needs to be able to accommodate changing operators. The following subsections detail the design choices that are summarized in Algorithm 1.

A. Anderson acceleration variant

As discussed in Section II and shown in [23] two types of AA can be derived from the corresponding Broyden's methods. These differ in the way that the Anderson coefficients η are computed. The type-I variant uses $\eta_k = (\mathcal{V}_k^\top \mathcal{R}_k)^{-1}\mathcal{V}_k^\top r_k$ whereas type-II uses $\eta_k = (\mathcal{R}_k^\top \mathcal{R}_k)^{-1}\mathcal{R}_k^\top r_k$. For neither of the two variants we observed consistently faster convergence when benchmarked against standard problem sets IV. However, the type-II variant has the advantage that the computation of η_k can be solved using QR decomposition. Moreover, the QR factorisation $\mathcal{R}_k = Q_k R_k$ can be efficiently updated from Q_{k-1}, R_{k-1} as at each step only one column is added to $\mathcal{R}_k$ [22]. Compared to the type-I computation of the coefficients, this results in a significant speed-up.

Algorithm 1: Safeguarded AA with memory restarts and scheduling.

Input: v_0, f_0, fixed-point iteration $F_\rho\colon \mathbb{R}^n \to \mathbb{R}^n$ with $v_{k+1} = f_k = F_\rho(v_k)$, allocated memory for $\mathcal{V}_0$, $\mathcal{R}_0$, Q, and R, column pointer $j = 1$, parameters: $\eta_{\max}, \tau, \epsilon, m_{\max}$

```
acc_success = false;
while ‖v_{k+1} − v_k‖ > ε do
    Update history: V_j ← [V_{j−1}, Δv_{k−1}],
      R_j ← [R_{j−1}, Δr_{k−1}], j ← j + 1;
    if j > 2 then
        Update QR factors Q_k, R_k of R_k;
        Compute η_k from R_k η_k = Q_k^⊤ r_k;
        if ‖η_k‖_2 > η_max then
            acc_success ← false;
        else
            Accelerate: v_k^acc = f_k − (V_j − R_j)η_k;
            acc_success ← true;
        if acc_success then
            Fixed-point iteration: f_k^acc = F_ρ(v_k^acc);
            Compute residual: r_k^acc = v_k^acc − f_k^acc;
            if ‖r_k^acc‖_2 ≤ τ ‖r(v_{k−1})‖_2 then
                v_{k+1} ← v_k^acc, f_{k+1} ← f_k^acc, and
                  r_{k+1} ← r_k^acc;
            else
                acc_success ← false;
    if not acc_success or j ≤ 2 then
        if operator change scheduled then
            ρ* ← u(F_ρ, v_k, f_k, r_k, ρ) ;
            Update operator: F_ρ ← F_{ρ*};
            j ← 1;
        Safeguarding step: v_{k+1} ← f_k,
          f_{k+1} = F_ρ(v_{k+1}), r_{k+1} = v_{k+1} − f_{k+1};
    if infeasibility detection scheduled and j = 2 then
        perform infeasibility checks;
    if j > m_max then
        j ← 1 ;
```

B. Safeguarding mechanism

One source of instability of AA is that not every acceleration candidate point is guaranteed to be closer to the fixed-point than the current iterate. To mitigate this issue, we perform a safeguarding step at each iteration. Assume that at iteration k, AA produced an accelerated candidate point v_k^{acc} using (11). An intuitive way to assess the quality of v_k^{acc} is by comparing the resulting residual operator norm with the last accepted step and imposing the condition

$$\|r(v_k^{\text{acc}})\|_2 = \|v_k^{\text{acc}} - F_\rho(v_k^{\text{acc}})\|_2 \le \tau \|r(v_k)\|_2 , \tag{12}$$

where $\tau \in (0, 1)$ is an expansiveness tolerance. If (12) holds, then v_k^{acc} is accepted as the next iterate, $v_{k+1} = v_k^{\text{acc}}$. Otherwise, the method resorts to $v_{k+1} = F_\rho(v_k)$. Checking condition (12) is expensive as it involves both the evaluation

of $F_\rho(v_k^{\text{acc}})$ and $F_\rho(v_k)$, which means two iterations of the underlying FOM. This makes the condition inefficient in practice. We also observed from standard benchmark testing that enforcing strict monotonicity of the residual norm tends to make the performance of the solver worse. The authors in [18] therefore only enforce the condition periodically or when past progress has stalled. On the other hand, the term $\tau \|r(v_k)\|_2$ is replaced in [32] by a summable and exponentially decaying series based on $\|r(v_0)\|_2$.

We use the relaxed safeguarding condition

$$\|v_k^{\text{acc}} - F_\rho(v_k^{\text{acc}})\|_2 \leq \tau \|r(v_{k-1})\|_2 \tag{13}$$

using the previous residual norm $\|r(v_{k-1})\|_2$ and $\tau \in (0, 2]$. This means that every evaluation of (13) only requires the evaluation of $F_\rho(v_k^{\text{acc}})$. We note that in the majority of iterations the safeguarding check (13) passes, in which case $F_\rho(v_k^{\text{acc}})$ can be used in the next evaluation of AA. Consequently, the safeguarding step only leads to additional operator evaluations if the candidate point is rejected.

Another source of instability is the tendency of the iterates $r(v_k)$ to become nearly co-aligned, which tends to happen in regions where the underlying algorithm does not make much progress. A consequence is bad conditioning of the matrix $\mathcal{R}_k$, which then results in unusable estimates of the coefficients η_k. To avoid this scenario we monitor the norm of the coefficients. If $\|\eta_k\|_2 > \eta_{\max}$ we abort the acceleration step and perform an ordinary fixed-point iteration.

C. Scheduling of operator changes

Solver packages rarely use the same operator at each iteration but instead employ heuristics to tune parameters during the solve process. For example, the ADMM step size parameter is often adapted based on the ratio of primal and dual residuals at the current iterate [33]. However, AA relies on the history of past input vectors and operator outputs to approximate the inverse of its Jacobian and therefore requires a non-changing operator. This history is invalid if the operator changes, and the method must then be restarted. Another issue is that some solver packages rely on pure operator steps to monitor convergence behaviour. For example [34] uses successive differences in FOM iterates and separating hyperplane conditions to detect infeasible problems.

We accommodate these two requirements by using a restarted memory approach for AA which adds information of past iterates up to a maximum memory length $m_{\max}$. Then the memory is reset and begins building a new history starting with the last iterate. Operator changes, i.e. computing a new F_ρ based on a parameter update rule $\rho^* \leftarrow u(F_\rho, v_k, f_k, r_k, \rho)$, are scheduled after the AA method has been restarted or the safeguarding check (13) fails, so that any changes in the operator are taken into account when new iterates are collected. Similarly, the infeasibility detection is scheduled after AA restarts, because each restart is followed by at least two non-accelerated iterations.

IV. NUMERICAL RESULTS

To evaluate the impact of safeguarding and acceleration on a FOM, we implemented Algorithm 1 in v0.8 of the conic ADMM-based solver `COSMO`. Different variants of AA are implemented in a standalone package[1]. Here we compare three different configurations of `COSMO`: without acceleration, with *unsafe* acceleration, and the safeguarded acceleration detailed in Algorithm 1. The experiments were run using Julia v1.5 on computing nodes of the University of Oxford ARC-HTC cluster with 16 logical Intel Xeon E5-2560 cores and 64GB of DDR3 RAM. We used the default parameters of `COSMO` and set the accuracy to $\epsilon = 10^{-6}$ for QPs and $\epsilon = 10^{-5}$ for SDPs, checking for convergence every 25 iterations. The acceleration method was configured with maximum memory length $m_{\max} = 15$. This value was chosen to obtain the best trade-off between convergence benefits, singularity issues, and computation overhead of higher memory lengths. For the safeguarding parameter we chose $\tau = 2$, and for the maximum norm of the Anderson parameters $\eta_{\max} = 10^4$, as these values work reasonably well on many different problem types.

Our benchmark tests used more than 500 problems from seven different QP and SDP problem sets to evaluate the impact of acceleration on the number of iterations needed to achieve higher accuracy solutions. In each case we evaluated how much extra computation time the acceleration scheme required. The tests compare three QP problem sets: the Maros and Mészáros problem set [35]; model predictive control problems based on the MPC Benchmarking Collection [36]; and Markowitz portfolio optimisation problems of the form described in [9]. Furthermore, problems from four SDP problem sets are included: a relaxed sparse principal component analysis (SPCA) problem [6]; computation of the Lovász theta function [37] for a set of undirected graphs from the SuiteSparse Matrix collection [38]; randomly generated SDPs with chordal block arrowhead sparsity pattern [39]; and a set of smaller non-decomposable problems used in Hans Mittelmann's SDP benchmarks [40].

To compare the three solver methods, we calculate for each problem set the mean and median total solve time, the number of problems solved, and the mean time used to calculate the Anderson candidate points as a fraction of the solve time. The results are shown in Table I.

As each solver configuration solved a different number of problems, we determined the average iteration counts and the average solve times using the subset of problems that was solved by every solver configuration. This skews the results slightly in favour of solver configurations that solved fewer problems as problems not solved by each configuration tend to converge slower. For the safeguarded method, Table I also shows in brackets the number of additional operator evaluations due to declined candidate points that failed the safeguarding check (13). Another metric shown in Table I, and commonly used in solver benchmarks, is the normalized shifted geometric mean $\mu_{g,s}$ of the solve time, defined by

$$\mu_{g,s} := \left[\prod_p (t_{p,s} + \text{sh})\right] - \text{sh}\Big]^{1/n} \tag{14}$$

[1] https://github.com/oxfordcontrol/COSMOAccelerators.jl

TABLE I

RESULTS FOR VANILLA, ACCELERATED, AND SAFEGUARDED & ACCELERATED ADMM FOR VARIOUS QP AND SDP PROBLEM SETS.

	algorithm	solved	iter[1]	solve time[2]	% acc time[3]	gmean[4]
Maros	vanilla	75	1046.0	2.10 (0.02)		2.33
	accelerated	98	676.8	1.27 (0.02)	24.7	1.08
	safeguarded	100	505.8 (22.9)	1.16 (0.02)	25.0	1.00
MPC	vanilla	255	820.3	0.020 (0.0026)		2.74
	accelerated	230	296.9	0.012 (0.0025)	27.5	4.72
	safeguarded	285	272.0 (115.3)	0.015 (0.0029)	26.5	1.00
Portfolio	vanilla	10	3460.0	171.51 (134.16)		3.04
	accelerated	10	1042.5	69.85 (48.96)	4.8	1.31
	safeguarded	10	725.0 (30.7)	49.67 (32.29)	3.6	1.00
SPCA	vanilla	9	5586.1	132.81 (95.69)		6.72
	accelerated	10	1013.9	46.92 (15.60)	14.6	1.57
	safeguarded	10	652.8 (32.3)	23.67 (12.20)	13.5	1.00
Block	vanilla	20	3136.3	99.31 (42.11)		5.40
	accelerated	20	308.8	11.49 (8.47)	2.9	1.00
	safeguarded	20	396.3 (66.9)	18.85 (8.97)	2.3	1.27
Lovász	vanilla	12	2220.8	17.01 (7.21)		5.72
	accelerated	15	1360.4	5.79 (3.01)	8.7	1.09
	safeguarded	15	1283.3 (15.0)	4.92 (2.82)	8.5	1.00
Mittelm.	vanilla	22	1853.4	179.50 (34.81)		1.72
	accelerated	29	744.3	79.57 (31.84)	3.7	1.00
	safeguarded	31	792.0 (5.9)	88.68 (22.51)	3.4	1.01

[1] mean iteration and (extra safeguarding iterations);
[2] mean and median (based on subset of problems where all solver configurations solved the problem);
[3] geometric mean of fraction of total solve time spent in acceleration-related functions;
[4] normalized shifted geometric mean of solve time, see (14) (based on all problems in the problem set);

with total solver time $t_{p,s}$ of solver s and problem p, shifting factor sh and size of the problem set n. We used a shifting factor $\mathrm{sh} = 10$ and a maximum allowable time of $5\,\mathrm{min}$ for QPs and $60\,\mathrm{min}$ for SDPs. Compared to the vanilla (i.e. unaccelerated) method, both accelerated methods provide a significant reduction in both the mean number of iterations and solve time. For the Maros and MPC problem sets, the median solve time stays fairly similar to the vanilla method. This is due to the presence of a number of easy problems that are solved in only a few iterations by each method.

The results also show that the safeguarded acceleration leads to a greater number of problems solved. While the impact of the safeguarded vs. non-safeguarded method appears small for most problem sets, we see a much more robust behavior for the MPC problems, with an additional 55 problems solved. The different numbers of problems solved is taken into account by the shifted geometric mean of the solve time. The impact of the safeguarded acceleration vs. the vanilla method ranges between 1.72 for the Mittelmann SDPs and 6.72 for the SPCA problems. To evaluate the additional time spent inside acceleration related functions, we compute the time spent on acceleration as a fraction of the total solve time. For a memory size of 15 this fraction varies from $2\,\%$ to $15\,\%$ for the large problems, the Markowitz Portfolio and SDP problems. For the smaller QP problem sets this fraction grows to $25\,\%$. Consequently, for smaller problems a significant reduction in iterations is needed to reduce the overall solve time. On average this seems to be the case for the Maros and MPC problem sets, but for some problems the acceleration might slow the solver down.

Figure 1 shows the convergence of the ADMM and operator residuals for the vanilla and the safeguarded accelerated method for the SDP problem `ros_500`. This behavior is typical when AA works successfully. Initially the residuals of both methods decrease in a similar fashion until an accuracy of 10^{-2} or 10^{-4} is reached. Then the residuals of the accelerated method (solid lines) drop sharply relative to the vanilla method's slow convergence from iteration 400 onwards. This is likely due to the algorithm reaching the region where AA approximates the Jacobian well and achieves superlinear convergence. It also indicates that the impact of AA will be fairly low when used to accelerate the convergence only to a low accuracy of e.g. 10^{-3}.

V. CONCLUSIONS

This paper uses a combination of scheduled memory restarts, a safeguarding rule based on the residual operator norm, and least-squares condition checking, to safeguard Anderson acceleration. We show that the approach works well for a FOM-based solver that allows adaptation of its operator form to improve convergence behaviour and which relies on successive unaltered iterates for infeasibility detection. We provide an efficient AA implementation using an updated QR decomposition in the latest version of `COSMO`. The effectiveness of our approach in reducing both the mean number of iterations and the solve time while increasing the number of solved problems is shown for a large number of QPs and SDPs taken from different application domains. Instead of using m_k+1 past iterates in the acceleration scheme, it seems

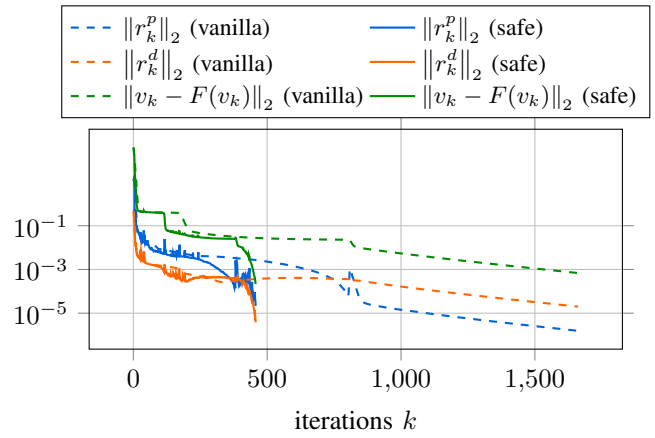

Fig. 1. Norms of primal residual $\|r_p^k\|_2$, dual residual $\|r_d^k\|_2$, and fixed point residual $\|v_k - F_\rho(v_k)\|_2$ of the vanilla and the safeguarded accelerated method (Mittelmann: `ros_500`).

promising to investigate whether performance improvements are achievable using the same number of iterates but spread out over a longer history of past iterations.

References

[1] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[2] H. Wolkowicz, R. Saigal, and L. Vandenberghe, *Handbook of Semidefinite Programming: Theory, Algorithms, and Applications*. Springer Science & Business Media, 2012, vol. 27.

[3] A. Ben-Tal and A. Nemirovski, *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. SIAM, 2001.

[4] J. Mattingley and S. Boyd, "Real-time convex optimization in signal processing," *IEEE Signal processing magazine*, vol. 27, no. 3, pp. 50–61, 2010.

[5] A. Raghunathan, J. Steinhardt, and P. S. Liang, "Semidefinite relaxations for certifying robustness to adversarial examples," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 877–10 887.

[6] A. d'Aspremont, L. El Ghaoui, M. I. Jordan, and G. R. G. Lanckriet, "A direct formulation for sparse PCA using semidefinite programming," *Advances in Neural Information Processing Systems*, vol. 17, pp. 41–48, 2004.

[7] T. De Bie and N. Cristianini, "Fast SDP relaxations of graph cut clustering, transduction, and other combinatorial problems," *Journal of Machine Learning Research*, vol. 7, no. Jul, pp. 1409–1436, 2006.

[8] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. I. Jordan, "Learning the kernel matrix with semidefinite programming," *Journal of Machine learning research*, vol. 5, no. Jan, pp. 27–72, 2004.

[9] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An Operator Splitting Solver for Quadratic Programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, Oct. 2020.

[10] Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn, "Chordal decomposition in operator-splitting methods for sparse semidefinite programs," *Mathematical Programming*, vol. 180, no. 1, pp. 489–532, 2020.

[11] M. Garstka, M. Cannon, and P. Goulart, "COSMO: A conic operator splitting method for convex conic problems," *Journal of Optimization Theory and Applications*, vol. 190, no. 3, pp. 779–810, 2021.

[12] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, Jun. 2016.

[13] X. Zhao, D. Sun, and K. Toh, "A Newton-CG augmented Lagrangian method for semidefinite programming," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1737–1765, 2010.

[14] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[15] A. d'Aspremont, D. Scieur, and A. Taylor, "Acceleration methods," *arXiv preprint arXiv:2101.09545*, 2021.

[16] P. Giselsson, M. Fält, and S. Boyd, "Line search for averaged operator iteration," *IEEE 55th Conference on Decision and Control (CDC)*, pp. 1015–1022, 2016.

[17] A. Ali, E. Wong, and J. Z. Kolter, "A semismooth newton method for fast, generic convex programming," in *International Conference on Machine Learning*. PMLR, 2017, pp. 70–79.

[18] A. Themelis and P. Patrinos, "SuperMann: a superlinearly convergent algorithm for finding fixed points of nonexpansive operators," *IEEE Transactions on Automatic Control*, vol. 64, no. 12, pp. 4875–4890, 2019.

[19] D. G. Anderson, "Iterative procedures for nonlinear integral equations," *Journal of the ACM*, vol. 12, no. 4, pp. 547–560, 1965.

[20] P. Pulay, "Convergence acceleration of iterative sequences. the case of SCF iteration," *Chemical Physics Letters*, vol. 73, no. 2, pp. 393–398, 1980.

[21] ——, "Improved SCF convergence acceleration," *Journal of Computational Chemistry*, vol. 3, no. 4, pp. 556–560, 1982.

[22] H. F. Walker and P. Ni, "Anderson acceleration for fixed-point iterations," *SIAM Journal on Numerical Analysis*, vol. 49, no. 4, pp. 1715–1735, 2011.

[23] H. Fang and Y. Saad, "Two classes of multisecant methods for nonlinear acceleration," *Numerical Linear Algebra with Applications*, vol. 16, no. 3, pp. 197–221, 2009.

[24] A. Toth and C. T. Kelley, "Convergence analysis for Anderson acceleration," *SIAM Journal on Numerical Analysis*, vol. 53, no. 2, pp. 805–819, 2015.

[25] F. A. Potra and H. Engler, "A characterization of the behavior of the anderson acceleration on linear problems," *linear Algebra and its Applications*, vol. 438, no. 3, pp. 1002–1011, 2013.

[26] D. M. Gay and R. B. Schnabel, "Solving systems of nonlinear equations by Broyden's method with projected updates," in *Nonlinear Programming*. Elsevier, 1978, pp. 245–281.

[27] V. Mai and M. Johansson, "Anderson acceleration of proximal gradient methods," in *International Conference on Machine Learning*. PMLR, 2020, pp. 6620–6629.

[28] J. Zhang, B. O'Donoghue, and S. Boyd, "Globally convergent type-I Anderson acceleration for nonsmooth fixed-point iterations," *SIAM Journal on Optimization*, vol. 30, no. 4, pp. 3170–3197, 2020.

[29] H. Bauschke and P. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*, 1st ed. Springer, 2011.

[30] E. Ryu and S. Boyd, "Primer on monotone operator methods," *Applied and Computational Mathematics*, vol. 15, no. 1, pp. 3–43, 2016.

[31] V. Eyert, "A comparative study on methods for convergence acceleration of iterative vector sequences," *Journal of Computational Physics*, vol. 124, no. 2, pp. 271–285, 1996.

[32] A. Fu, J. Zhang, and S. Boyd, "Anderson accelerated Douglas–Rachford splitting," *SIAM Journal on Scientific Computing*, vol. 42, no. 6, pp. A3560–A3583, 2020.

[33] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[34] G. Banjac, P. Goulart, B. Stellato, and S. Boyd, "Infeasibility detection in the alternating direction method of multipliers for convex optimization," *Journal of Optimization Theory and Applications*, vol. 183, no. 2, pp. 490–519, 2019.

[35] I. Maros and C. Mészáros, "A repository of convex quadratic programming problems," *Optimization Methods and Software*, vol. 11, no. 1-4, pp. 671–681, 1999.

[36] J. Ferreau, "MPC benchmarking collection: Open collection of model predictive control (MPC) benchmarking problems," 2020. [Online]. Available: https://github.com/ferreau/mpcBenchmarking

[37] L. Lovász, "On the Shannon capacity of a graph," *IEEE Transactions on Information Theory*, vol. 25, no. 1, pp. 1–7, 1979.

[38] T. A. Davis, "Suitesparse: A suite of sparse matrix software," 2015, http://faculty.cse.tamu.edu/davis/suitesparse.html.

[39] M. Garstka, M. Cannon, and P. Goulart, "A clique graph based merging strategy for decomposable SDPs," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 7355–7361, 2020, 21th IFAC World Congress.

[40] H. Mittelmann, "Benchmarks for optimization software," 2020, http://plato.asu.edu/bench.html.