Received: 28 March 2022 | Revised: 29 June 2022 | Accepted: 10 August 2022

DOI: 10.1002/nme.7096

RESEARCH ARTICLE

WILEY

Composite Anderson acceleration method with two window sizes and optimized damping

Kewang Chen[1,2] | **Cornelis Vuik**[2]

[1]College of Mathematics and Statistics, Nanjing University of Information Science and Technology, Nanjing, China

[2]Delft Institute of Applied Mathematics, Delft University of Technology, Delft, the Netherlands

Correspondence
Kewang Chen, College of Mathematics and Statistics, Nanjing University of Information Science and Technology, Nanjing 210044, China.
Email: kwchen@nuist.edu.cn

Funding information
National Natural Science Foundation of China, Grant/Award Number: 12001287; Startup Foundation for Introducing Talent of Nanjing University of Information Science and Technology, Grant/Award Number: 2019r106; China Scholarship Council, Grant/Award Number: 202008320191

Abstract

In this article, we propose and analyze a set of fully nonstationary Anderson acceleration (AA) algorithms with two window sizes and optimized damping. Although AA has been used for decades to speed up nonlinear solvers in many applications, most authors are simply using and analyzing the stationary version of AA (sAA) with fixed window size and a constant damping factor. The behavior and potential of the nonstationary version of AA methods remain an open question. Most efficient linear solvers however use composable algorithmic components. Similar ideas can be used for AA to solve nonlinear systems. Thus in the present work, to develop nonstationary AA algorithms, we first propose a systematic way to dynamically alternate the window size m by the multiplicative composite combination, which means we apply sAA(m) in the outer loop and apply sAA(n) in the inner loop. By doing this, significant gains can be achieved. Second, to make AA to be a fully nonstationary algorithm, we need to combine these strategies with our recent work on the nonstationary AA algorithm with optimized damping (AAoptD), which is another important direction of producing nonstationary AA and nice performance gains have been observed. Moreover, we also investigate the rate of convergence of these nonstationary AA methods under suitable assumptions. Finally, our numerical results show that some of these proposed nonstationary AA algorithms converge faster than the stationary sAA method and they may significantly reduce the storage and time to find the solution in many cases.

KEYWORDS

Anderson acceleration, fixed-point iteration, nonstationary

1 | INTRODUCTION

In this part, we first present a literature review on the development of Anderson acceleration (AA) method and its applications. Then we discuss our main motivations and the structure for the present work. In 1962, Anderson[1] developed a technique for accelerating the convergence of the Picard iteration associated with a fixed-point problem which is called extrapolation algorithm. Since that, this technique has enjoyed remarkable success and wide usage, especially in electronic structure computations, where it is known as Anderson mixing. The technique is now called AA in the applied mathematics community. In contrast to Picard iteration, which uses only one previous iterate, AA method proceeds by linearly recombining a list of previous iterates in a way such that approximately minimizes the linearized fixed-point residual. The usual general form of AA with damping is given in Algorithm 1.

Algorithm 1. Anderson acceleration: AA(m)

Given: x_0 and $m \geq 1$.
Set: $x_1 = g(x_0)$.
for $k = 0, 1, 2, \ldots$ **do**
  Set: $m_k = \min\{m, k\}$.
  Set: $F_k = (f_{k-m_k}, \ldots, f_k)$, where $f_i = g(x_i) - x_i$.
  Determine: $\alpha^{(k)} = \left(\alpha_0^{(k)}, \ldots, \alpha_{m_k}^{(k)}\right)^T$ that solves

$$\min_{\alpha=(\alpha_0,\ldots,\alpha_{m_k})^T} \|F_k\alpha\|_2 \text{ s.t. } \sum_{i=0}^{m_k} \alpha_i = 1.$$

  Set: $x_{k+1} = (1-\beta_k)\sum_{i=0}^{m_k} \alpha_i^{(k)} x_{k-m_k+i} + \beta_k \sum_{i=0}^{m_k} \alpha_i^{(k)} g(x_{k-m_k+i})$.
end for

Here, f_k is the residual for the kth iteration; m is the window size which indicates how many history residuals will be used in the algorithm. It is usually a fixed number during the procedure. The value of m is typically no larger than 3 in the early days of applications and now this value could be as large as up to 100.[2] $\beta_k \in (0, 1]$ is a damping factor (or a relaxation parameter) at kth iteration. We have, for a fixed window size m:

$$\beta_k = \begin{cases} 1, & \text{no damping,} \\ \beta \text{ (a constant independent of } k), & \text{stationary Anderson acceleration,} \\ \beta_k \text{ (depending on } k), & \text{nonstationary Anderson acceleration.} \end{cases}$$

We can also formulate the above constrained optimization problem as an equivalent unconstrained least-squares problem:[3,4]

$$\min_{(\omega_1,\ldots,\omega_{m_k})^T} \left\| f_k + \sum_{i=1}^{m_k} \omega_i (f_{k-i} - f_k) \right\|_2. \tag{1}$$

One can easily recover the original problem by setting

$$\omega_0 = 1 - \sum_{i=1}^{m_k} \omega_i.$$

AA methods are "essentially equivalent" to the nonlinear GMRES methods[5-9] and the direct inversion on the iterative subspace method (DIIS).[10-12] It is also in a broad category with methods based on quasi-Newton updating.[13-17] For example, Walker and Ni[9] showed that AA without truncation is equivalent in a certain sense to the GMRES method on linear problems and Fang and Saad[15] had clarified a remarkable relationship of AA to quasi-Newton methods. However, unlike Newton-like methods, one advantage of AA is that it does not require the expensive computation or approximation of Jacobians or Jacobian-vector products. Although AA has been used for decades, convergence analysis has been reported only recently. For the linear case, Toth and Kelley[3] first proved the stationary version of AA (sAA) without damping is locally r-linearly convergent if the fixed point map is a contraction and the coefficients in the linear combination remain bounded. Later, Evans et al.[18] extended the result to AA with damping factors and proved the convergence rate is $\theta_k((1-\beta_{k-1}) + \beta_{k-1}\kappa)$, where κ is the Lipschitz constant for the function $g(x)$ and θ_k is the ratio quantifying the convergence gain provided by AA in step k. Recently, in 2019, Pollock et al.[19] applied sAA to the Picard iteration for solving steady incompressible Navier–Stokes equations and proved that the acceleration improves the convergence rate of the Picard iteration. More recently, De Sterck and He[20] extended the result to more general fixed-point iteration $x = g(x)$, given knowledge of the spectrum of $g'(x)$ at fixed-point x^* and Wang et al.[21] extended the result to study the asymptotic linear convergence speed of sAA applied to alternating direction method of multipliers (ADMMs) method. It is worth mentioning here that the sAA in the papers of De Sterck and He[20] and Wang et al.[21] is stationary in a different sense: in

those papers, the α_i^k of Algorithm 1 are fixed and do not depend on the iteration, so the α_i are stationary. Sharper local convergence results of AA remain a hot research topic in this area. For more related results about AA and its applications, we refer the interested readers to papers[22-28] and references therein.

Although AA has been widely used and studied for decades, most authors are simply using and analyzing the sAA with fixed window size and a constant damping factor. The behavior and potential of the nonstationary versions of the AA method have not been deeply studied and few results have been reported. Anderson[2] suggested a conceptual procedure for adaptively choosing β_k. However, he has not had an opportunity to assess its practical utility. A dynamic approach to damping is also demonstrated by Glowinski et al.[29] Evans et al.[18] developed a new strategy to adaptively choose the damping factors, where those β_k are chosen by a simple heuristic strategy based on the gain θ_k ($\beta_k = 0.9 - 1/2 * \theta_k$). The heuristic choice of damping yields $0.4 \leq \beta_k \leq 0.9$, and leads to fewer iterations to convergence than with the uniform damping factors tested on the p-Laplacian problem, where p-Laplacian is a non-contractive operator. Pollock and Rebholz[30] proposed a strategy to dynamically alternate the window sizes. The window size m_k is kept at a small to moderate value (2–5) until the residual drops below a given threshold, on the order of 10^{-2} or 10^{-3}, then m_k is increased to a higher steady level, for instance $m = 10$. This approach is appropriate for problems where the initial residual is moderately scaled. Besides, the early days of the Anderson Mixing method (the 1980s, for electronic structure calculations) initially dictated the window size $m \leq 3$ due to the storage limitations and costly g evaluations. However, in recent years and a broad range of contexts, the window size m ranging from 20 to 100 has also been considered by many authors. For example, Walker and Ni[9] used $m = 50$ to solve the nonlinear Bratu problem. A natural question is that should we try to further speed up AA method or try to use a larger size of the window? No such comparison results have been reported. As we know, there are two main possible directions for producing nonstationary AA. One is choosing different damping factors β_k in each iteration, see our recent work on the nonstationary AA algorithm with optimized damping (AAoptD).[31] The other way of making AA to be a nonstationary algorithm is to alternate the window size during iterations. But no systematic ways have been proposed to dynamically alternate the window size m. Since most efficient linear solvers use composable algorithmic components,[32,33] similar ideas can be used for AA(m) and AA(n) to solve nonlinear systems. Moreover, the combination of choosing optimized damping factors and alternating window sizes may lead to significant gains over the stationary AA. Therefore, in the present work, we propose and study the fully nonstationary AA algorithms with dynamic window sizes and optimized damping.

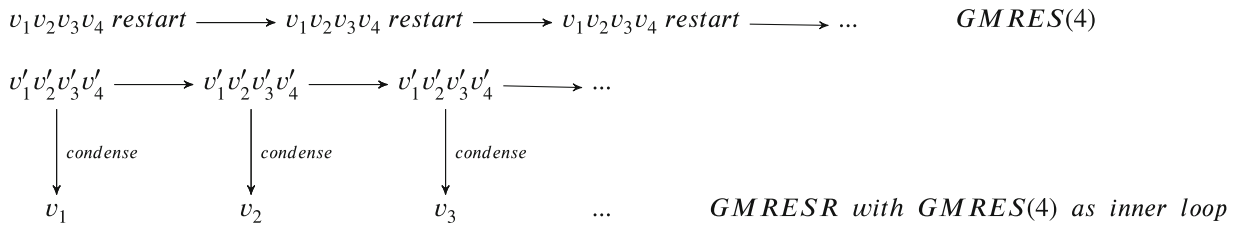
The article is organized as follows. Our new algorithms and analysis are developed in Section 2; the rate of convergence to those algorithms are studied in Section 3; experimental results and discussion are in Section 4; conclusions follow in Section 5.

2 | FULLY NONSTATIONARY AA

2.1 | Nonstationary AA with two window sizes

2.1.1 | Motivation

For linear problems, most efficient linear solvers however use composable algorithmic components. Similar ideas can be used for AA to solve nonlinear systems. van der Vorst and Vuik[34] proposed a hybrid method GMRES recursive (GMRESR) which consists of an outer and inner loop. In the inner loop, one approximates the solution of a linear system by GMRES to find a good search direction. In the following picture, we visualize the strong point of GMRESR in comparison to GMRES(m). A search direction is indicated by v_i. We see for GMRES(4) that after four iterations all information is thrown away. For GMRESR the information after four inner iterations is condensed into one search direction so information does not get lost. This method was also further investigated by Vuik.[35] He proved that GMRESR(m) converges at least as fast as GMRES(m) if GMRES(m) does not stagnate in m iteration steps.



On the other hand, it was proved by Walker and Ni[9] that AA without truncation is equivalent in a certain sense to the GMRES method on linear problems. Therefore, motivated by the GMRESR method, to solve nonlinear problems, we propose a systematic way to dynamically alternate the window size m in sAA by using the multiplicative composition, which means we apply sAA(m) first in the outer loop and then apply sAA(n) in the inner loop.

2.1.2 | Multiplicative composition of two window sizes

We start with composite sAA(m) with sAA(0) (i.e., Picard iteration) in each iteration. This means that after applying one step of sAA(m) without damping, we get,

$$x_{k+1/2} = \sum_{i=0}^{m_k} \alpha_i^{(k)} g(x_{k-m_k+i}).$$

Then, we take the result $x_{k+1/2}$ as an input $\hat{x}_0 = x_{k+1/2}$ and apply Picard iteration in the inner loop:

$$\hat{x}_{j+1} = g(\hat{x}_j).$$

Putting these together, we have the following nonstationary algorithm AA(m, AA(0)) as in Algorithm 2.

Algorithm 2. Anderson acceleration with dynamic window-sizes: AA(m,AA(**0**))

Given: x_0 *iterM*, *iterN*, and $m \geq 1$.
Set: $x_1 = g(x_0)$.
for $k = 1, 2, \ldots,$ *iterM* **do**
  Set: $\mathbf{x^m_{k+1/2}} \leftarrow$ **apply one step of AA(m**; $\{\mathbf{x_k}\}$**) as given in Algorithm 1.**
  Set: $\hat{x}_0 = x^m_{k+1/2}$
  for $j = 0, 1, 2, \ldots,$ *iterN* **do**
    Set: $\hat{\boldsymbol{x}}_{\boldsymbol{j+1}} \leftarrow$ **apply one step of Picard iteration on** $\hat{x}_j$**.**
  end for
  Set: $x_{k+1} = \hat{x}_{iterN}$
end for

Suppose we just do a single inner loop iteration in Algorithm 2, the total amount of work of AA(m, AA(0)) in each iteration is much less than that of applying sAA(m) twice. Algorithm 2 also means that we may "turn off" the acceleration for a while and then turn on the acceleration. However, the performance can be better than sAA(m), see our numerical experiments in Section 4.

More generally, we apply sAA(m) in the outer loop and apply sAA(n) in the inner loop. So, in each iteration, after applying sAA(m), we get,

$$x_{k+1/2} = \sum_{i=0}^{m_k} \alpha_i^{(k)} g(x_{k-m_k+i}).$$

Then we apply sAA(n) with the initial guess $x_0 = x_{k+1/2}$ for *iterN* iterations:

$$x_{k+1} \leftarrow \text{applying sAA}(n) \text{ with initial guess } x_0 = x_{k+1/2}.$$

In other words, the multiplicative composition reads

$$x_{k+1} = \text{sAA}(m, \text{sAA}(n)).$$

We summarize this in the following algorithm in Algorithm 3.

Remark 1. There is a lot of variety here. Let m and n be the window size used in the outer loop and inner loop, respectively. And *iterM* and *iterN* be the total numbers of iterations for the outer loop and inner loop, respectively. In the present work, we report some results for the case where $m > n$ and *iterM* $\gg$ *iterN*, which means the window size used in the inner loop

Algorithm 3. Anderson acceleration with dynamic window-sizes: AA(m,AA($\mathbf{n}$))

Given: x_0, m, n, *iterM*, and *iterN* (with *iterN* $\geq n$).
Set: $x_1 = g(x_0)$.
for $k = 1, 2, \dots,$ *iterM* **do**
  Set: $\mathbf{x_{k+1/2}} \leftarrow$ **apply one step of** AA($\mathbf{m}$; $\{\mathbf{x_k}\}$) **as given in Algorithm 1.**
  Set: $\hat{x}_0 = x_{k+1/2}$
  for $j = 0, 1, 2, \dots,$ *iterN* **do**
    Set: $\hat{\boldsymbol{x}}_{\boldsymbol{j+1}} \leftarrow$ **apply one step of** AA($\mathbf{n}$; $\{\hat{\boldsymbol{x}}_{\boldsymbol{j}}\}$) **as given in Algorithm 1.**
  end for
  Set: $x_{k+1} = \hat{x}_{iterN}$
end for

TABLE 1 Memory requirements

Methods	Memory
AA(m)	$m + 1$ vectors in memory
AA(m, AA(n))	$m + n + 2$ vectors in memory

is smaller than that used in the outer loop and the maximum iterations of the inner loop is much smaller than that of the outer loop. For example, one can choose $n = 1$ and *iterN* $= 1$. In this case, we could compare the convergence results of AA(m, AA(1)) with AA(m) by calculate the residual per function evaluation of $g(x)$. See more discussions in Section 4.

Moreover, we summarize the memory requirements for those algorithms in Section 2.1.2 (Table 1). For some problems, memory storage could be crucial. Our numerical results show that the nonstationary AA methods with smaller window sizes usually perform better than the stationary AA algorithm with very large window sizes, which means our proposed nonstationary AA methods may significantly reduce the storage requirements. See more discussions in our numerical experiments in Section 4.

2.2 | Nonstationary AA with optimized damping

In this part, we briefly summarize some of our recent works on AA with optimized damping (AAoptD), which will be used to produce a fully nonstationary AA in the next section. Suppose that β_k is different at each iteration k, then we have

$$\begin{aligned} x_{k+1} &= (1-\beta_k)\sum_{i=0}^{m_k}\alpha_i^{(k)}x_{k-m_k+i} + \beta_k\sum_{i=0}^{m_k}\alpha_i^{(k)}g(x_{k-m_k+i}) \\ &= \sum_{i=0}^{m_k}\alpha_i^{(k)}x_{k-m_k+i} + \beta_k\left(\sum_{i=0}^{m_k}\alpha_i^{(k)}g(x_{k-m_k+i}) - \sum_{i=0}^{m_k}\alpha_i^{(k)}x_{k-m_k+i}\right). \end{aligned} \tag{2}$$

Let us define the following averages given by the solution α^k to the optimization problem by

$$x_k^\alpha = \sum_{i=0}^{m_k}\alpha_i^{(k)}x_{k-m_k+i}, \quad \tilde{x}_k^\alpha = \sum_{i=0}^{m_k}\alpha_i^{(k)}g(x_{k-m_k+i}). \tag{3}$$

Thus, (2) becomes

$$x_{k+1} = x_k^\alpha + \beta_k(\tilde{x}_k^\alpha - x_k^\alpha). \tag{4}$$

A natural way to choose "best" β_k at this stage is that choosing β_k such that x_{k+1} gives a minimal residual. So we just need to solve the following unconstrained optimization problem:

$$\min_{\beta_k}\|x_{k+1} - g(x_{k+1})\|_2 = \min_{\beta_k}\|x_k^\alpha + \beta_k(\tilde{x}_k^\alpha - x_k^\alpha) - g(x_k^\alpha + \beta_k(\tilde{x}_k^\alpha - x_k^\alpha))\|_2. \tag{5}$$

Using the fact that

$$\begin{aligned} g(x_k^\alpha + \beta_k(\tilde{x}_k^\alpha - x_k^\alpha)) &\approx g(x_k^\alpha) + \beta_k \frac{\partial g}{\partial x}\Big|_{x_k^\alpha}(\tilde{x}_k^\alpha - x_k^\alpha) \\ &\approx g(x_k^\alpha) + \beta_k \left(g(\tilde{x}_k^\alpha) - g(x_k^\alpha)\right). \end{aligned} \tag{6}$$

Therefore, (5) becomes

$$\begin{aligned} \min_{\beta_k} \|x_{k+1} - g(x_{k+1})\|_2 &= \min_{\beta_k} \|x_k^\alpha + \beta_k(\tilde{x}_k^\alpha - x_k^\alpha) - g(x_k^\alpha + \beta_k(\tilde{x}_k^\alpha - x_k^\alpha))\|_2 \\ &\approx \min_{\beta_k} \|x_k^\alpha + \beta_k(\tilde{x}_k^\alpha - x_k^\alpha) - \left[g(x_k^\alpha) + \beta_k(g(\tilde{x}_k^\alpha) - g(x_k^\alpha))\right]\|_2 \\ &\approx \min_{\beta_k} \| \left(x_k^\alpha - g(x_k^\alpha)\right) - \beta_k \left[(g(\tilde{x}_k^\alpha) - g(x_k^\alpha)) - (\tilde{x}_k^\alpha - x_k^\alpha)\right] \|_2. \end{aligned} \tag{7}$$

Thus, we just need to calculate the projection

$$\beta_k = \frac{\left(x_k^\alpha - g(x_k^\alpha)\right) \cdot \left[\left(x_k^\alpha - g(x_k^\alpha)\right) - (\tilde{x}_k^\alpha - g(\tilde{x}_k^\alpha))\right]}{\| \left[\left(x_k^\alpha - g(x_k^\alpha)\right) - (\tilde{x}_k^\alpha - g(\tilde{x}_k^\alpha))\right] \|_2^2}. \tag{8}$$

AA algorithm with optimized damping AAoptD(m) reads as follows:

Algorithm 4. Anderson acceleration with optimized dampings: AAoptD(m)

Given: x_0 and $m \geq 1$.
Set: $x_1 = g(x_0)$.
for $k = 0, 1, 2, \ldots$ **do**
  Set: $m_k = \min\{m, k\}$.
  Set: $F_k = (f_{k-m_k}, \ldots, f_k)$, where $f_i = g(x_i) - x_i$.
  Determine: $\alpha^{(k)} = \left(\alpha_0^{(k}, \ldots, \alpha_{m_k}^{(k)}\right)^T$ that solves

$$\min_{\alpha=(\alpha_0,\ldots,\alpha_{m_k})^T} \|F_k\alpha\|_2 \ \ s.\,t. \sum_{i=0}^{m_k} \alpha_i = 1.$$

  Set: $x_k^\alpha = \sum_{i=0}^{m_k} \alpha_i^{(k)} x_{k-m_k+i}, \quad \tilde{x}_k^\alpha = \sum_{i=0}^{m_k} \alpha_i^{(k)} g(x_{k-m_k+i})$.
  Set: $r_p = \left(x_k^\alpha - g(x_k^\alpha)\right), \ r_q = \left(\tilde{x}_k^\alpha - g(\tilde{x}_k^\alpha)\right)$.
  Set: $\beta_k = \dfrac{(r_p - r_q)^T r_p}{\|r_p - r_q\|_2^2}$.
  Set: $x_{k+1} = x_k^\alpha + \beta_k(\tilde{x}_k^\alpha - x_k^\alpha)$.
end for

Remark 2. This optimized damping step is a "local optimal" strategy at kth iteration. It usually will speed up the convergence rate compared with an undamped one, but not always. Because in $(k+1)$th iteration, it uses a combination of all previous m history information. Besides, if the "local optimal" β_k is not in the interval $(0, 1]$, we set $\beta_k = 1/2$. Moreover, when β_k is very close to zero, the system is overdamped, which sometimes may also slow down the convergence speed. We may need to further modify our β_k to bound it away from zero by using

$$\hat{\beta}_k = \max\{\beta_k, \eta\}, \tag{9}$$

or

$$\hat{\beta}_k = \begin{cases} \beta_k, & \text{if } \beta_k \geq \eta, \\ 1 - \beta_k, & \text{if } \beta_k < \eta, \end{cases} \tag{10}$$

where η is a small positive number such that $0 < \eta < 0.5$. For more details on the implementation of AAoptD(m) and its performance, we refer the readers to our recent paper.[31]

2.3 | Fully nonstationary AA with dynamic window-sizes and optimized damping

At this stage, we are ready to present our final fully nonstationary AA algorithms. Since alternating the window sizes and using different damping factors in each iteration are two main ways to produce nonstationary AA, we need to combine those strategies to make it into a fully nonstationary algorithm. For example, we can obtain the following fully nonstationary algorithm AA(m, AAoptD(n)) as in Algorithm 5 by using multiplicative composite combination.

Algorithm 5. Fully nonstationary Anderson acceleration: AA(m,AAoptD($\mathbf{n}$))

Given: x_0, m, n, *iterM*, and *iterN* (with *iterN* $\geq n$).
Set: $x_1 = g(x_0)$.
for $k = 1, 2, \dots,$ *iterM* **do**
  Set: $\mathbf{x_{k+1/2}} \leftarrow$ **apply one step of AA**($\mathbf{m}$; $\{\mathbf{x_k}\}$) **as given in Algorithm 1.**
  Set: $\hat{x}_0 = x_{k+1/2}$
  for $j = 0, 1, 2, \dots,$ *iterN* **do**
    Set: $\hat{\boldsymbol{x}}_{\boldsymbol{j+1}} \leftarrow$ **apply one step of AAoptD**($\mathbf{n}$; $\{\hat{\boldsymbol{x}}_{\boldsymbol{j}}\}$) **as given in Algorithm 4.**
  end for
  Set: $x_{k+1} = \hat{x}_{iterN}$
end for

Similarly, AAoptD(m, AAoptD(n)) or AAoptD(m, AA(n)) can be easily obtained. Now, we have a set of new nonstationary AA algorithms. We test and compare the performance of some of these methods with stationary AA in Section 4. Suggestions on how to use and choose these methods are provided in our final conclusion part in Section 5.

3 | RESIDUAL BOUNDS

In this section, we investigate the residual bounds of these nonstationary AA methods. The main technical results and assumptions are adopted from papers[3,18] with necessary modifications. Here we provide the convergence theorems for AAoptD(m), AA(m, AA(1)) with inner loop iteration *iterN* $= 1$ and AAoptD(m, AA(1)) with inner loop iteration *iterN* $= 1$. These typical nonstationary AA methods are extensively studied in Section 4. Similarly, one can derive the rate of convergence to other nonstationary AA methods.

To begin with, we summarize the convergence result in Theorem 1 for AA with optimized damping as in Algorithm 4.

Theorem 1. *Assume that*

- $g : R^n \to R^n$ *has a fixed point* $x^* \in R^n$ *such that* $g(x^*) = x^*$.
- g *is uniformly Lipschitz continuously differentiable in the ball* $B(\rho) = \{u | \|x - x^*\|_2 \leq \rho\}$.
- *There exists* $\kappa \in (0, 1)$ *such that* $\|g(y) - g(x)\|_2 \leq \kappa \|y - x\|_2$ *for all* $x, y \in R^n$.
- *Suppose that* $\exists M$ *and* $\epsilon > 0$ *such that for all* $k > m$, $\sum_{i=0}^{m-1} |\alpha_i| < M$ *and* $|\alpha_m| \geq \epsilon$.

Then we have the following convergence result for AAoptD(m) *given in Algorithm 4:*

$$\|f(x_{k+1})\|_2 \leq \theta_{k+1} \left[(1 - \beta_k) + \kappa\beta_k\right] \|f(x_k)\|_2 + \sum_{i=0}^{m} O(\|f(x_{k-m+i})\|_2^2), \tag{11}$$

where the average gain

$$\theta_{k+1} = \frac{\|\sum_{i=0}^{m} \alpha_i^k f(x_{k-m+i})\|_2}{\|f(x_k)\|_2}$$

and

$$\beta_k = \frac{\left(x_k^\alpha - g(x_k^\alpha)\right) \cdot \left[\left(x_k^\alpha - g(x_k^\alpha)\right) - (\tilde{x}_k^\alpha - g(\tilde{x}_k^\alpha))\right]}{\| \left[\left(x_k^\alpha - g(x_k^\alpha)\right) - (\tilde{x}_k^\alpha - g(\tilde{x}_k^\alpha))\right] \|_2^2}$$

with

$$x_k^\alpha = \sum_{i=0}^{m_k} \alpha_i^{(k)} x_{k-m_k+i}, \quad \tilde{x}_k^\alpha = \sum_{i=0}^{m_k} \alpha_i^{(k)} g(x_{k-m_k+i}).$$

Proof. We first note here that assuming α_m bounded away from zero is adopted from Evans et al.[18] and Pollock et al.,[19] where this condition is required in order to obtain some estimation on the residuals. We cannot get rid of this restriction, so we also keep it here. The proof of this theorem can be found in this article[18] for general damping β_k. The key ideas of the analysis are relating the difference of consecutive iterates to residuals based on performing the inner-optimization and explicitly defining the gain in the optimization stage to be the ratio of improvement over a step of the unaccelerated fixed-point iteration. Additionally, here we also need to use (2) and (8) to explicitly calculate these optimized β_k. ∎

Next, we provide the convergence rate for nonstationary AA methods AA(m, AA(1)) with inner loop iteration *iterN* = 1 and AAoptD(m, AA(1)) with inner loop iteration *iterN* = 1, which are extensively studied in Section 4.

Theorem 2. *Assume that $g : R^n \to R^n$ has a fixed point $x^* \in R^n$ such that $g(x^*) = x^*$ and satisfies all assumptions in Theorem 1. Then we have the following convergence rate for* AA(m, AA(1)) *as in Algorithm 3 with iterN* = 1 *inner loop iterations:*

$$\|f(x_{k+1})\|_2 \leq \bar{\theta}_1 \theta_{k+1} \kappa \left[(1-\beta_k) + \kappa\beta_k\right] \|f(x_k)\|_2 + \textit{high order terms}, \tag{12}$$

where

$$\theta_{k+1} = \frac{\|\sum_{i=0}^m \alpha_i^k f(x_{k-m+i})\|_2}{\|f(x_k)\|_2}, \quad \bar{\theta}_1 = \frac{\|\bar{\alpha}_0^1 f(\bar{x}_0) + \bar{\alpha}_1^1 f(\bar{x}_1)\|_2}{\|f(\bar{x}_1)\|_2}$$

with

$$\bar{x}_0 = x_{k+1/2}, \quad \bar{x}_1 = g(\bar{x}_0).$$

Proof. For the outer loop, according to the results in Theorem 1 with any damping factor $\beta_k \in (0, 1]$, we have

$$\|f(x_{k+1/2})\|_2 \leq \theta_{k+1} \kappa \left[(1-\beta_k) + \kappa\beta_k\right] \|f(x_k)\|_2 + \sum_{i=0}^m O(\|f(x_{k-m+i})\|_2^2), \tag{13}$$

where

$$\theta_{k+1} = \frac{\|\sum_{i=0}^m \alpha_i^k f(x_{k-m+i})\|_2}{\|f(x_k)\|_2}.$$

As α_i^k is the solution to the optimization problem in Algorithm 1 and the fact that $\alpha_k^k = 1$, $\alpha_j^k = 0, j \neq k$, is in the feasible set for the optimization problem, we immediately have

$$0 \leq \theta_{k+1} \leq 1.$$

For the inner loop with *iterN* = 1, we have the initial guess $\bar{x}_0 = x_{k+1/2}$, then $\bar{x}_1 = g(\bar{x}_0)$ and

$$f(\bar{x}_0) = g(\bar{x}_0) - \bar{x}_0, \quad f(\bar{x}_1) = g(\bar{x}_1) - \bar{x}_1.$$

Let $\overline{\alpha}_0$ and $\overline{\alpha}_1$ be the solution to the inner loop optimization problem, then applying Theorem 1 with $m = 1$ without damping, we have

$$\|f(\overline{x}_2)\|_2 \le \overline{\theta}_1 \kappa \|f(\overline{x}_1)\| + O(\|f(\overline{x}_1)\|_2^2) + O(\|f(\overline{x}_0)\|_2^2) \tag{14}$$

with

$$\overline{\theta}_1 = \frac{\|\overline{\alpha}_0^1 f(\overline{x}_0) + \overline{\alpha}_1^1 f(\overline{x}_1)\|_2}{\|f(\overline{x}_1)\|_2},$$

where $\overline{\alpha}_0^1$ and $\overline{\alpha}_1^1$ solves the optimization problem of AA(1) in the inner loop iteration. Similarly, since $\overline{\alpha}_0^1 = 0$ and $\overline{\alpha}_1^1 = 1$, is in the feasible set for the related optimization problem, we get

$$0 \le \overline{\theta}_1 \le 1.$$

Using (13) and the fact that the inner loop use $\overline{x}_0 = x_{k+1/2}$ as an initial guess, we have $\overline{x}_1 = g(\overline{x}_0) = g(x_{k+1/2})$. Therefore,

$$\|f(\overline{x}_1)\|_2 \le \theta_{k+1} \kappa \|f(x_k)\|_2 + \sum_{i=0}^{m} O(\|f(x_{k-m+i})\|_2^2). \tag{15}$$

Since $iterN = 1$, so after finishing the inner loop iteration, we will set $x_{k+1} = \overline{x}_2$. Thus, from (14) and (15), we finally obtain

$$\|f(x_{k+1})\|_2 = \|f(\overline{x}_2)\|_2 \le \overline{\theta}_1 \theta_{k+1} \kappa \left[(1 - \beta_k) + \kappa \beta_k\right] \|f(x_k)\|_2 + \text{high order terms}, \tag{16}$$

which completes the proof of this theorem. ∎

Theorem 3. *Assume that $g : R^n \to R^n$ has a fixed point $x^* \in R^n$ such that $g(x^*) = x^*$ and satisfies the assumptions in Theorem 1. Then we have the following convergence rate for* AAoptD(m, AA(1)) *with $iterN = 1$ inner loop iterations:*

$$\|f(x_{k+1})\|_2 \le \overline{\theta}_1 \theta_{k+1} \kappa \left[(1 - \beta_k) + \kappa \beta_k\right] \|f(x_k)\|_2 + \textit{high order terms}, \tag{17}$$

where

$$\beta_k = \frac{\left(x_k^\alpha - g(x_k^\alpha)\right) \cdot \left[\left(x_k^\alpha - g(x_k^\alpha)\right) - \left(\tilde{x}_k^\alpha - g(\tilde{x}_k^\alpha)\right)\right]}{\| \left[\left(x_k^\alpha - g(x_k^\alpha)\right) - \left(\tilde{x}_k^\alpha - g(\tilde{x}_k^\alpha)\right)\right] \|_2^2}$$

and

$$\theta_{k+1} = \frac{\|\sum_{i=0}^{m} \alpha_i f(x_{k-m+i})\|_2}{\|f(x_k)\|_2}, \quad \overline{\theta}_1 = \frac{\|\overline{\alpha}_0 f(\overline{x}_0) + \overline{\alpha}_1 f(\overline{x}_1)\|_2}{\|f(\overline{x}_1)\|_2}$$

with

$$\overline{x}_0 = x_{k+1/2}, \quad \overline{x}_1 = g(\overline{x}_0).$$

Proof. The proof is similar to the proof of Theorem 2 with explicitly expression for β_k values, thus we omit it here. ∎

Remark 3. From the above theorems, we know that the composite AA methods not only converge but also indeed improve the convergence rate of the Picard iteration under some suitable assumptions. Moreover, in our numerical experiments in Section 4, we also explicitly calculate these average gains θ_k and β_k at each iteration as described in Theorems 1–3, so one can see how much the residual is reduced at each iteration.

4 | EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we numerically compare the performance of these fully nonstationary AA algorithms with regular stationary AA (with uniform window size and a constant damping factor). All these experiments are done in the MATLAB 2021b environment. MATLAB codes are available upon request to the authors.

This first example is from Walker and Ni's paper,[9] where a sAA with window size $m = 50$ is used to solve the Bratu problem. This problem has a long history, we refer the reader to Glowinski et al.[29] and Pernice and Walker,[36] and the references in those papers.

Example 1 (The Bratu problem). The Bratu problem is a nonlinear PDE boundary value problem as follows:

$$\Delta u + \lambda\, e^u = 0, \quad \text{in} \quad D = [0,1] \times [0,1],$$
$$u = 0, \quad \text{on} \quad \partial D.$$

In this experiment, we use a centered-difference discretization on a 64×64 grid and take $\lambda = 6$ in the Bratu problem. We first use the zero initial approximate solution in all cases and then we test these methods with different initial approximations at the end. And we also applied the preconditioning such that the basic Picard iteration still works. The preconditioning matrix that we used here is the diagonal inverse of the matrix A, where A is a matrix for the discrete Laplace operator.

We first solve the Bratu problem using the multiplicative composition of nonstationary AA methods with the outer loop window size $m = 20$ and inner loop window size $n = 1$. Here, we use zero initial approximation and set the inner iteration $iterN = 1$ for all composite methods, which means we only apply two iterations in the inner loop. In many applications, the cost of evaluating $g(x)$ usually dominates to the cost of solving the small least-squares problem, in order to compare the performance of nonstationary AA and stationary AA, we count the number of evaluations of $g(x)$ and plot the residual per function evaluation of $g(x)$. The results are shown in Figure 1. The total time used for different methods is shown on the left side of Table 2. Here we use the Matlab commands *tic* and *toc* to record the running time for each of these methods and we have not optimized our Matlab codes yet. From Figure 1, we see that the nonstationary AA methods work better than regular stationary AA. The best one for this case is the fully nonstationary AA AAoptD(20, AA(1)). The convergence results are consistent with the time consumption on the left side of Table 2.

From Theorems 1 to 3, we know that the composite AA methods not only converge but also indeed improve the convergence rate of the Picard iteration. Here, we also explicitly calculate these average gains θ_k and some damping factors β_k at each iteration as in Figures 2 and 3, respectively. Thus one can see how much the residual is reduced at each

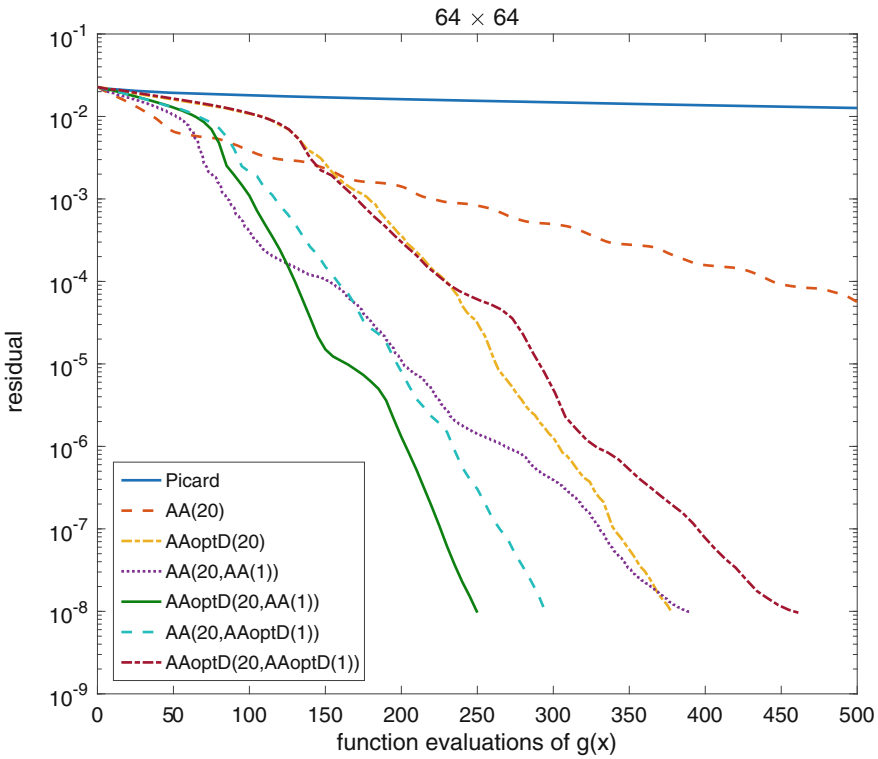


FIGURE 1 Solve the Bratu problem with zero initial approximation, inner loop $m = 20$, and outer loop $n = 1$

TABLE 2 Total time used in solving the Bratu problem on a 64×64 grid with the same initial approximation

Methods	Time (s)	Methods	Time (s)
AAoptD(20, AA(1))	53.19	AA(20, AA(2))	30.38
AA(20, AAoptD(1))	61.77	AAoptD(20, AA(2))	48.89
AAoptD(20)	80.10	AA(20, AAoptD(2))	63.15
AA(20, AA(1))	83.61	AAoptD(20, AAoptD(2))	72.83

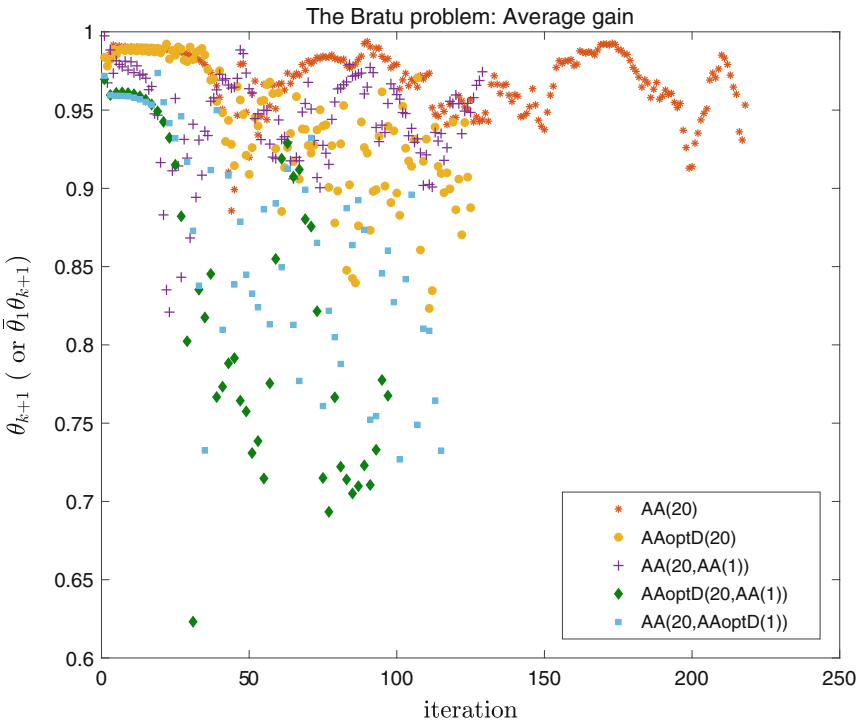


FIGURE 2 Solve the Bratu problem with composite AA methods: Average gain

iteration. For example, as shown in Figure 2, we see that the average gain for AAoptD(AA(1)) and AA(20, AAoptD(1)) at each iteration are much better than stationary AA(20) since a small average gain θ_k means a faster convergence rate. These results are consistent with convergence results shown in Figure 1.

Besides, since there is a lot of freedom in composing AA, we also test other AA(m, AA(n)) on this problem. For example, in Figure 4, we find that the nonstationary AA methods with outer window size $m = 20$ and inner loop window size $n = 2$ give similar or even better convergence improvement, which is not surprising since a larger inner window size is used here. The total time used for different composition methods is shown on the right side of Table 2. Moreover, some of these methods (i.e., AA(20, AA(2)) and AAoptD(20, AA(1))) are better or comparable with the stationary AA(50). This is very important for solving larger-size problems where computer storage is crucial. Our proposed nonstationary AA methods may save a lot of memory storage while maintaining a similar (or even faster) convergence rate.

Lastly, some recent work by De Sterck and He[37] has shown that the asymptotic convergence factor of AA(m) can strongly depend on the initial approximation. So we test these composite methods AA(m, AA(n)) with different initial guesses. Instead of using zero initial approximation (i.e., $x_0 = [0, \dots, 0]$), here we use an unit initial guess (i.e., $x_0 = [1, \dots, 1]$) and a random initial guess in between the zero initial guess and the unit initial guess. The results are show in Figures 5 and 6, respectively. From these two figures, we verify that the asymptotic convergence factor of AA(m) sometimes strongly depends on the initial approximations, so do these composite methods AA(m, AA(n)). However, these composite methods have the same trend for multiple different initial guesses.

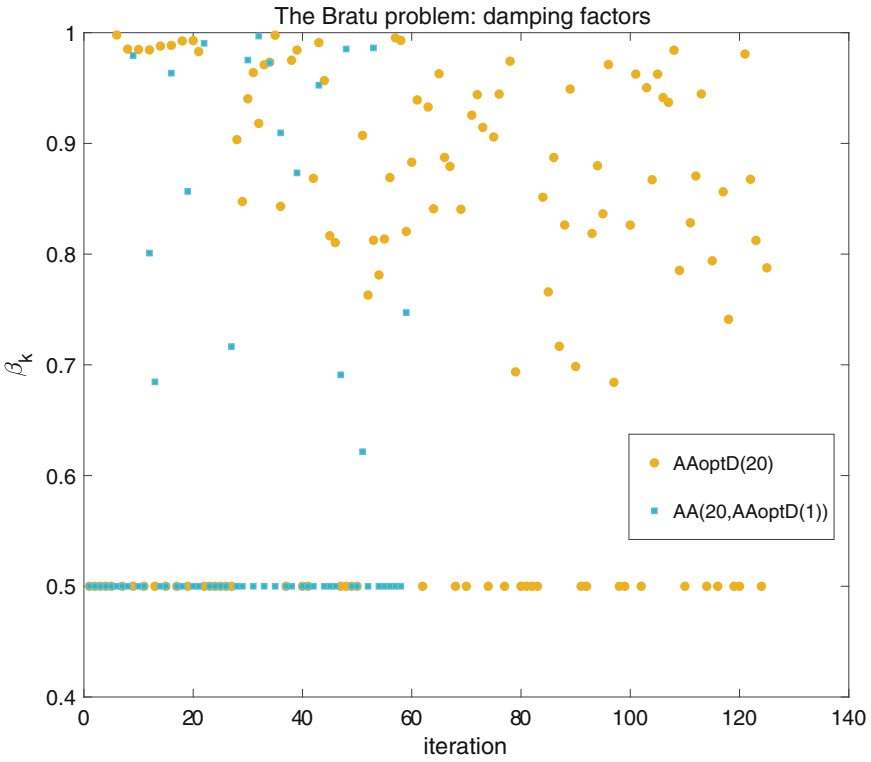


FIGURE 3 Solve the Bratu problem with composite AA methods: Damping factors

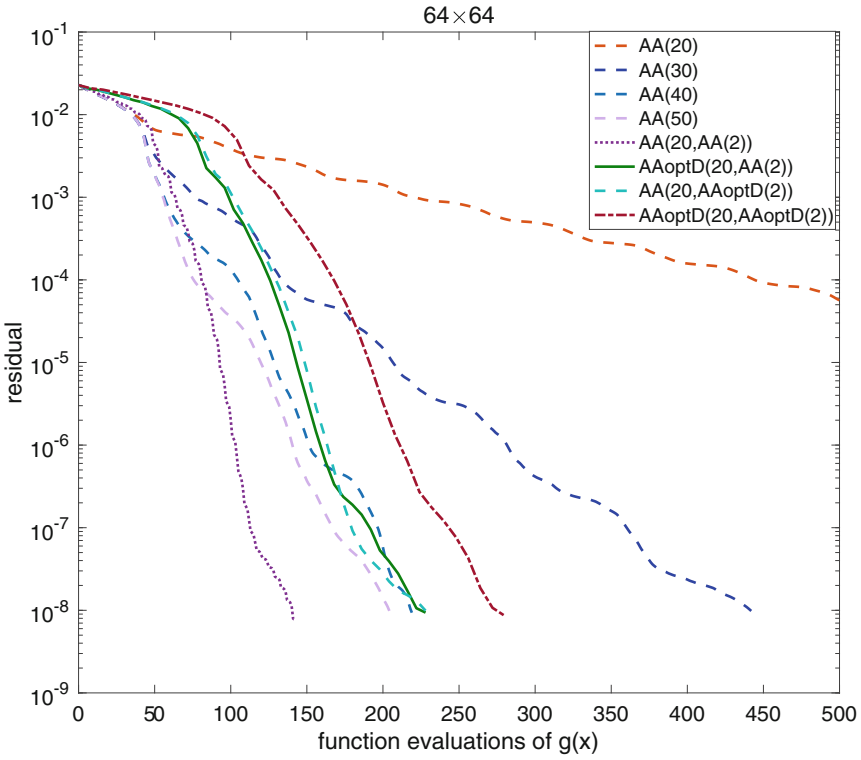


FIGURE 4 Solve the Bratu problem with zero initial approximation, inner loop $m = 20$, and outer loop $n = 2$

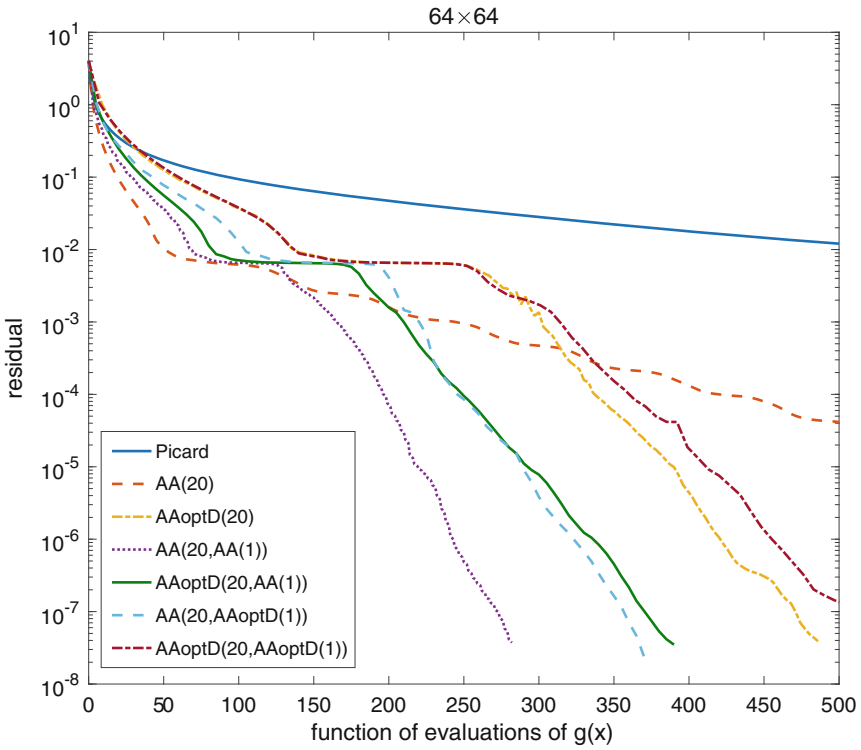


FIGURE 5 Solve the Bratu problem with unit initial approximation $x_0 = [1, \ldots, 1]^T$, inner loop $m = 20$, and outer loop $n = 1$

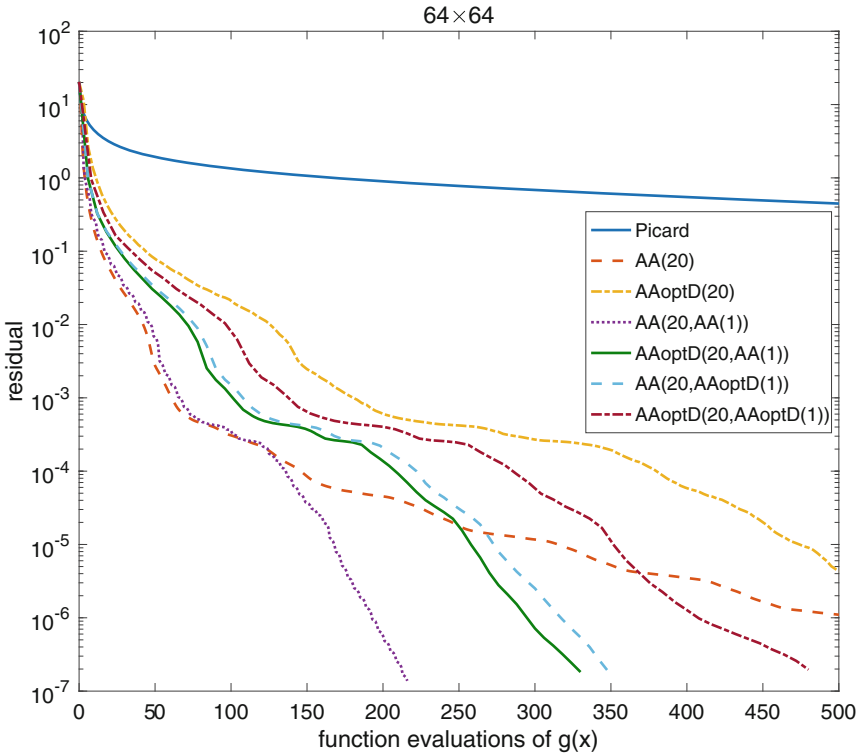


FIGURE 6 Solve the Bratu problem with random initial approximation, inner loop $m = 20$, and outer loop $n = 1$

Example 2 (The stationary nonlinear convection-diffusion problem). Solve the following 2D nonlinear convection-diffusion equation in a square region:

$$\epsilon(-u_{xx} - u_{yy}) + (u_x + u_y) + ku^2 = f(x,y), \quad (x,y) \in D = [0,1] \times [0,1]$$

with the source term

$$f(x,y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$$

and zero boundary conditions: $u(x,y) = 0$ on ∂D.

In this numerical experiment, we first use a centered-difference discretization on a 32×32 grid. We choose $\epsilon = 1$, $\epsilon = 0.1$ and $\epsilon = 0.01$, respectively. Those ϵ values indicate the competition between the diffusion and convection effect. We take $k = 3$ in the above problem and use $u_0 = (1, 1, \ldots, 1)^T$ as an initial approximate solution in all cases. As in solving the Bratu problem, the same preconditioning strategy is used here. The preconditioning matrix that we used here is the diagonal inverse of the matrix A, where A is a matrix for the discrete Laplace operator.

For $\epsilon = 1$, from Figures 7 and 9, we observe similar results that the composite methods perform much better than sAA. Moreover, the average gain as in Figure 8 and the time cost as in Table 3 also confirm the above results. For $\epsilon = 0.1$, from Figure 10, we find that the Picard iteration does not converge anymore. However, AA methods with a small window size already work. Moreover, some of our proposed nonstationary AA methods (e.g., AAoptD(1, AA(1)), AA(1, AAoptD(1)), and AA(1, AA(1))) perform better than the stationary AA(1) method. For $\epsilon = 0.01$, from Figures 11 and 12, we see that the Picard method, the stationary AA(1) method and nonstationary method AA(1, AA(1)) method do not converge while other nonstationary methods still converge. However, for the converging methods, there are some wiggles in the numerical approximation, see the bottom figures in Figure 13. This may result from using the central difference scheme for the convection term. To solve this problem, we then use the upwind scheme (backward difference) for the convection term. The results are shown in Figures 14 and 15. All acceleration methods converge. AA(1, AA(1)) performs best. AAoptD(1, AA(1)) and AA(1, AAoptD(1)) are comparable to the stationary AA(1) method. Moreover, there are no wiggles in the numerical approximation when applying the upwind scheme for the convection term.

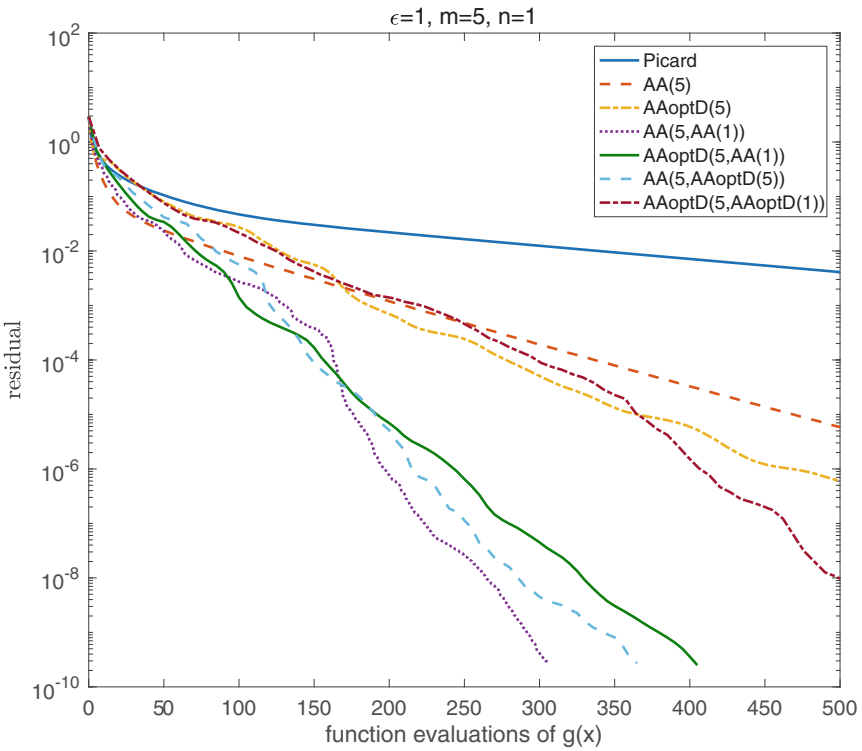


FIGURE 7 Solve the convection-diffusion problem using the central discretization: $\epsilon = 1$, convergence results

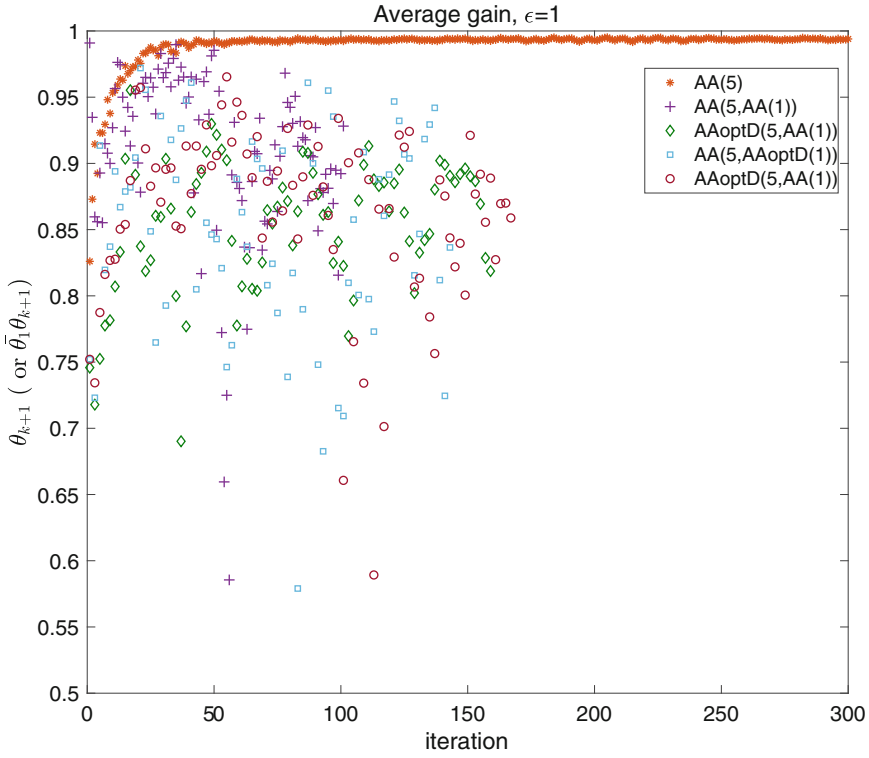


FIGURE 8 Solve the convection-diffusion problem using the central discretization: $\epsilon = 1$, average gain

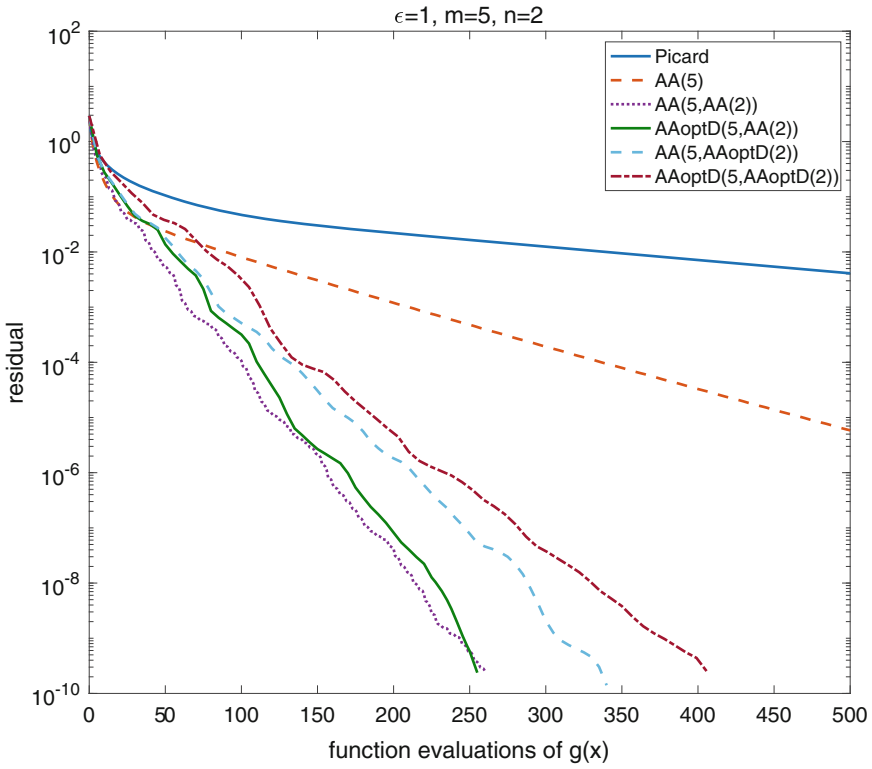


FIGURE 9 Solve the convection-diffusion problem using the central discretization: $\epsilon = 1$, larger inner window size

TABLE 3 Total time used in solving the convection-diffusion problem with $\epsilon = 1$

Methods	Time (s)	Methods	Time (s)
AA(5, AA(1))	2.45	AA(5, AA(2))	2.03
AA(5, AAoptD(1))	2.75	AAoptD(5, AA(2))	2.31
AAoptD(5, AA(1))	3.06	AA(5, AAoptD(2))	3.93
AAoptD(5, AAoptD(1))	4.21	AAoptD(5, AAoptD(2))	4.11

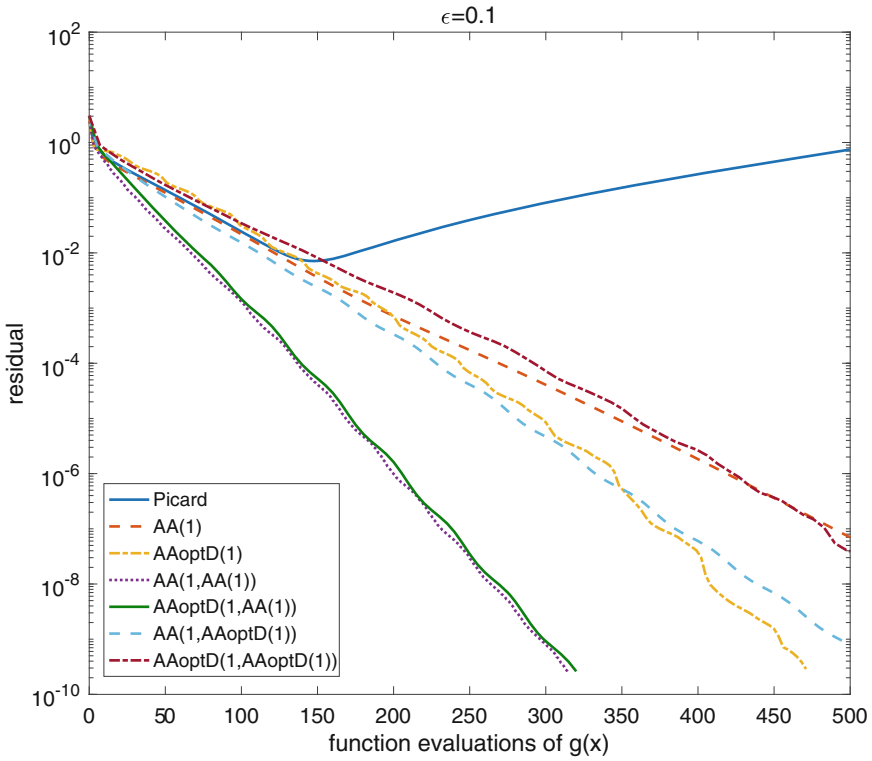


FIGURE 10 Solve the convection-diffusion problem using the central discretization: $\epsilon = 0.1$, convergence results

Our next example is about solving a linear system $Ax = b$. As proved by Walker and Ni,[9] AA without truncation is "essentially equivalent" in a certain sense to the GMRES method for linear problems.

Example 3 (The linear equations). Apply AA and AAoptD to solve the following linear system $Ax = b$, where A is

$$A = \begin{pmatrix} 2 & -1 & \cdots & 0 & 0 \\ -1 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & \cdots & -1 & 2 \end{pmatrix}, \quad A \in R_{N\times N}$$

and

$$b = (1, \ldots, 1)^T.$$

Choose $N = 100$ so that a large window size m is needed in AA. We also note here that the Picard iteration does not work for this problem.

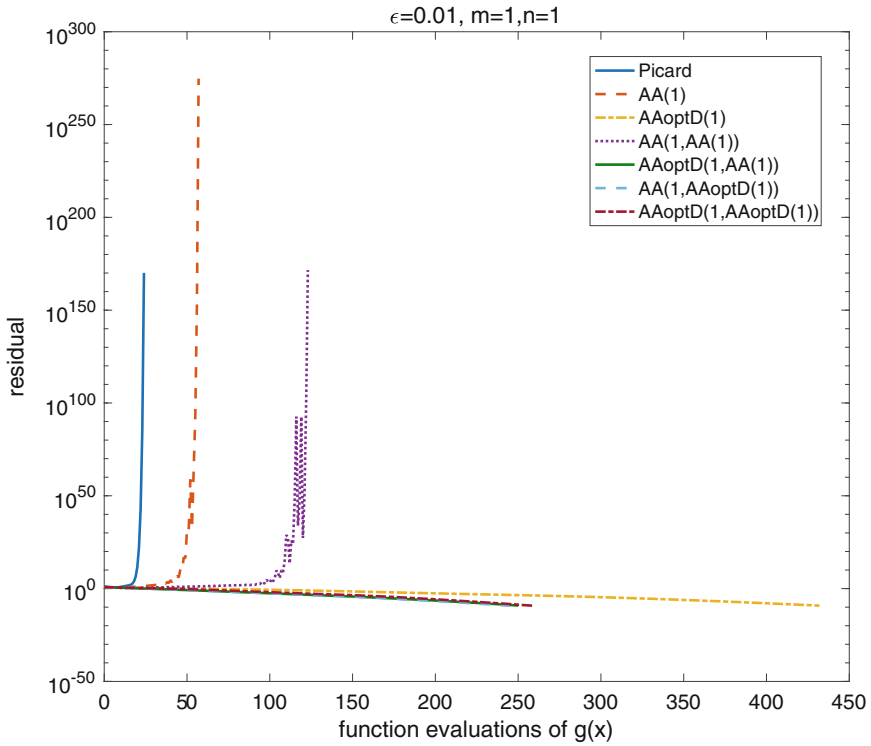


FIGURE 11 Solve the convection-diffusion problem using the central discretization: $\epsilon = 0.01$, convergence results

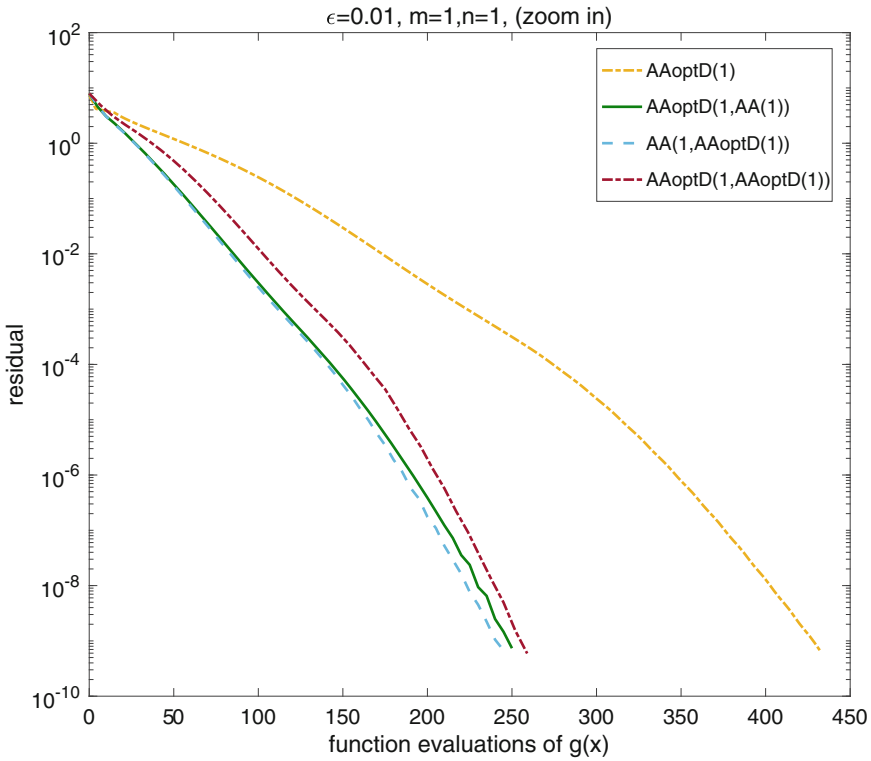


FIGURE 12 Solve the convection-diffusion problem using the central discretization: $\epsilon = 0.01$, convergence results (zoom in)

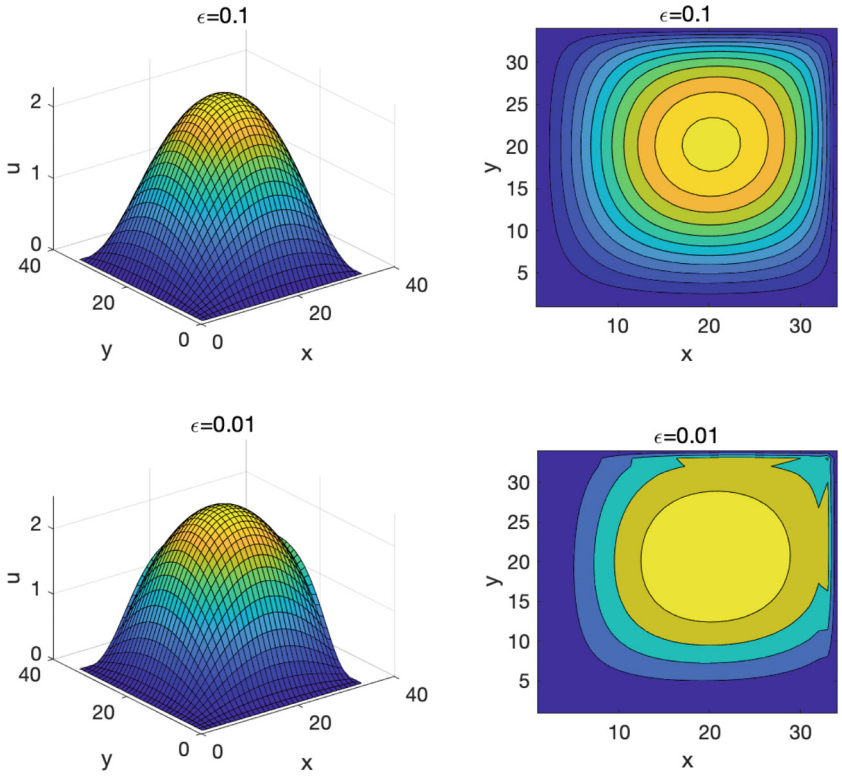


FIGURE 13 Solve the convection-diffusion problem using the central discretization: Solutions for $\epsilon = 0.1$ and $\epsilon = 0.01$

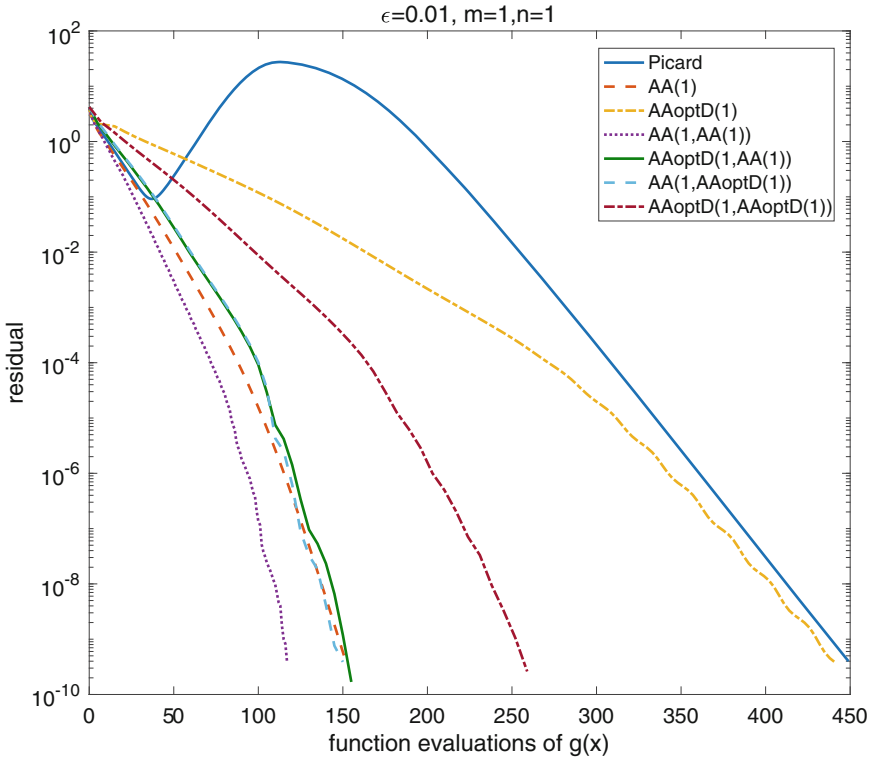


FIGURE 14 Solve the convection-diffusion problem using the upwind method: $\epsilon = 0.01$, convergence results

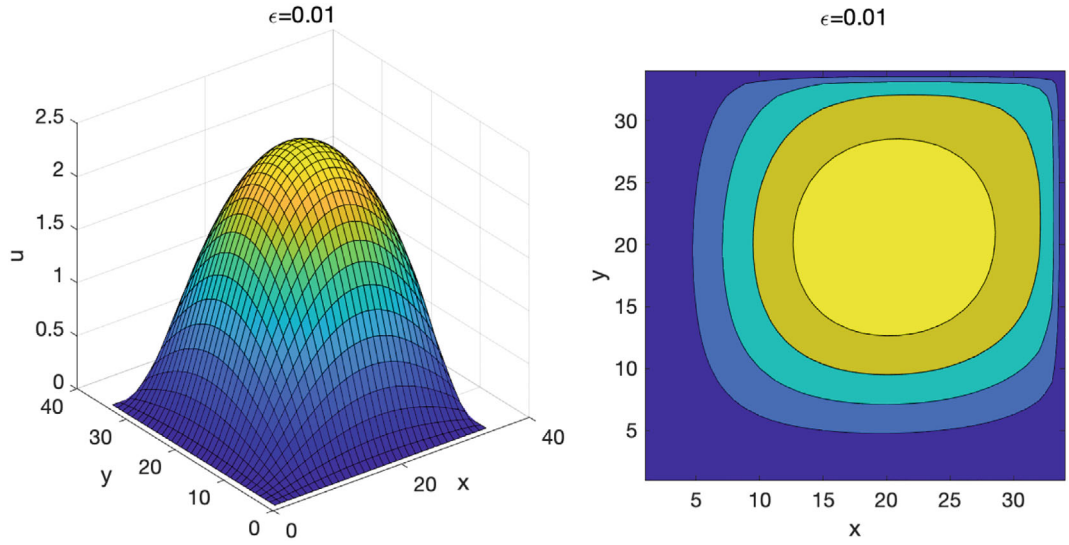


FIGURE 15 Solve the convection-diffusion problem using the upwind method: Solution for $\epsilon = 0.01$

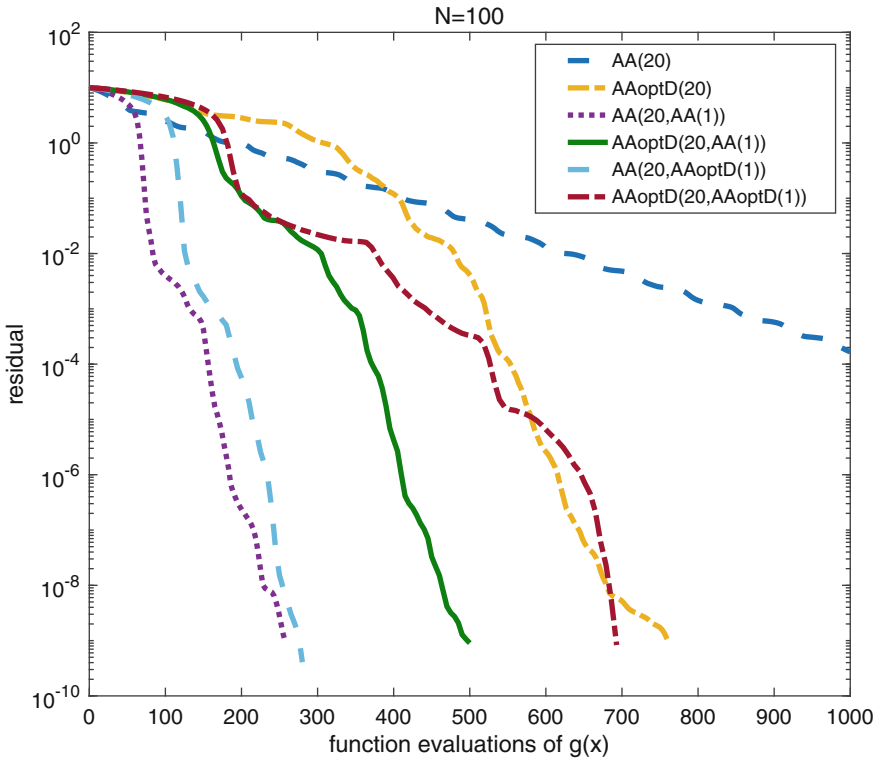


FIGURE 16 Solve a linear problem $Ax = b$ with $N = 100$: Convergence results

In our test, the initial guess is $x_0 = (0, \ldots, 0)^T$. The result is shown in Figure 16. From Figure 16, we see that the fully nonstationary AA methods work much better than the stationary AA method. This example also indicates that our proposed fully nonstationary AA can be also used to solve linear systems. Moreover, we also plot the average gain θ_k for each iteration in Figure 17. The results are roughly consistent with the convergence results. We also notice from Figure 17 that AA(20, AAoptD(1)) should perform better than AA(20, AA(1)), which is slightly inconsistent with the result in Figure 16. This is because the function evaluation for the linear problem may not dominate to cost of solving the least-squares problem.

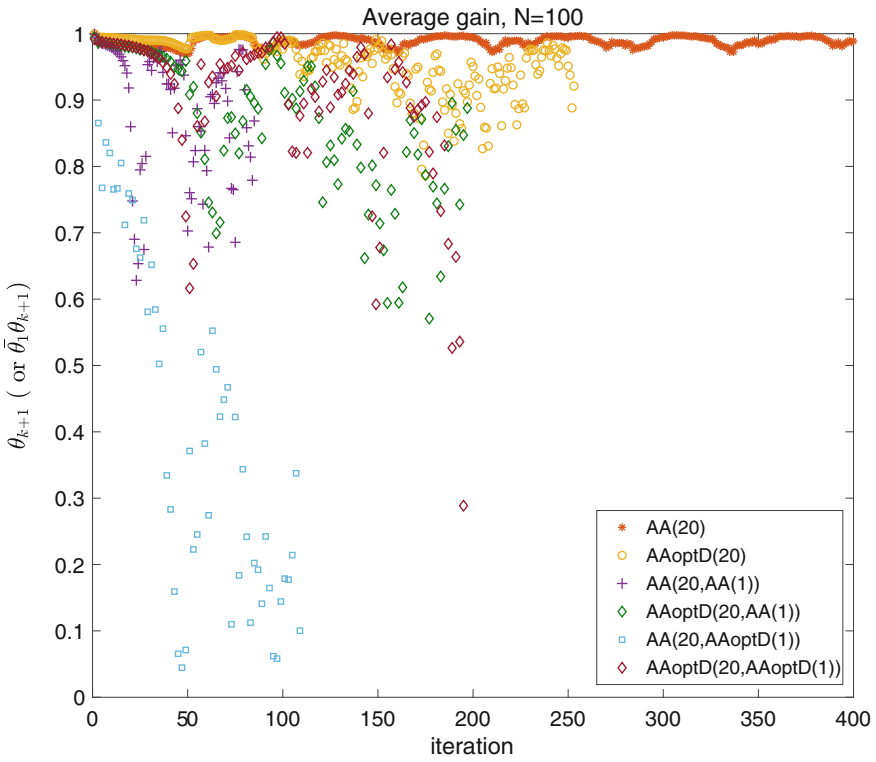


FIGURE 17 Solve a linear problem $Ax = b$ with $N = 100$: Average gain

5 | CONCLUSIONS

In the present work, we propose and analyze a set of fully nonstationary AA algorithms with two window sizes and optimized damping factors to further speed up linear and nonlinear iterations. In general, these nonstationary AA algorithms can converge faster than the stationary AA method and they may significantly reduce the memory requirements and time to find the solution. For future guidance of choosing these nonstationary AA methods, our numerical results indicate that AA(m, AA(n)) or AAoptD(m, AA(n)) (with a very small inner window size $n < m$) usually converges much faster than stationary AA(m). This is not surprising since the local convergence rate of AA with damping factors is $\theta_k((1-\beta_{k-1}) + \beta_{k-1}\kappa)$.[18] There is a lot of variety in these fully nonstationary AA methods, our future work will continue to explore the behaviors of these methods and test them on other broader problems. Moreover, we also verify that the asymptotic convergence factor of AA(m) sometimes strongly depends on the initial approximations, so do these composite methods AA(m, AA(n)). However, these methods still compare to each other in a similar way for multiple different initial guesses.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China (grant number 12001287); the Startup Foundation for Introducing Talent of Nanjing University of Information Science and Technology (grant number 2019r106); the first author Kewang Chen also gratefully acknowledge the financial support for his doctoral study provided by the China Scholarship Council (No. 202008320191). Moreover, the authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Kewang Chen https://orcid.org/0000-0001-5613-3644

REFERENCES

1. Anderson DG. Iterative procedures for nonlinear integral equations. *J Assoc Comput Mach*. 1965;12:547-560. doi:10.1145/321296.321305
2. Anderson DGM. Comments on "Anderson acceleration, mixing and extrapolation". *Numer Algorithms*. 2019;80(1):135-234. doi:10.1007/s11075-018-0549-4
3. Toth A, Kelley CT. Convergence analysis for Anderson acceleration. *SIAM J Numer Anal*. 2015;53(2):805-819. doi:10.1137/130919398
4. Walker HF. Anderson acceleration: algorithms and implementations. WPI Math Sciences Dept Report MS-6-15-50; 2011.
5. Carlson NN, Miller K. Design and application of a gradient-weighted moving finite element code. I. In one dimension. *SIAM J Sci Comput*. 1998;19(3):728-765. doi:10.1137/S106482759426955X
6. Miller K. Nonlinear Krylov and moving nodes in the method of lines. *J Comput Appl Math*. 2005;183(2):275-287. doi:10.1016/j.cam.2004.12.032
7. Oosterlee CW, Washio T. Krylov subspace acceleration of nonlinear multigrid with application to recirculating flows. *SIAM J Sci Comput*. 2000;21:1670-1690.
8. Washio T, Oosterlee CW. Krylov subspace acceleration for nonlinear multigrid schemes. *Electron Trans Numer Anal*. 1997;6:271-290.
9. Walker HF, Ni P. Anderson acceleration for fixed-point iterations. *SIAM J Numer Anal*. 2011;49(4):1715-1735. doi:10.1137/10078356X
10. Lin L, Yang C. Elliptic preconditioner for accelerating the self-consistent field iteration in Kohn-Sham density functional theory. *SIAM J Sci Comput*. 2013;35(5):S277-S298. doi:10.1137/120880604
11. Pulay P. Convergence acceleration of iterative sequences. the case of SCF iteration. *Chem Phys Lett*. 1980;73(2):393-398. doi:10.1016/0009-2614(80)80396-4
12. Pulay P. Improved SCF convergence acceleration. *J Comput Chem*. 1982;3(4):556-560. doi:10.1002/jcc.540030413
13. Eirola T, Nevanlinna O. Accelerating with rank-one updates. *Linear Algebra Appl*. 1989;121:511-520.
14. Eyert V. A comparative study on methods for convergence acceleration of iterative vector sequences. *J Comput Phys*. 1996;124(2):271-285. doi:10.1006/jcph.1996.0059
15. Fang H, Saad Y. Two classes of multisecant methods for nonlinear acceleration. *Numer Linear Algebra Appl*. 2009;16(3):197-221. doi:10.1002/nla.617
16. Haelterman R, Degroote J, Van Heule D, Vierendeels J. On the similarities between the quasi-Newton inverse least squares method and GMRES. *SIAM J Numer Anal*. 2010;47(6):4660-4679. doi:10.1137/090750354
17. Yang C, Meza JC, Lee B, Wang LW. KSSOLV—A MATLAB toolbox for solving the Kohn-Sham equations. *ACM Trans Math Softw*. 2009;36(2):10, 35. doi:10.1145/1499096.1499099
18. Evans C, Pollock S, Rebholz LG, Xiao M. A proof that Anderson acceleration improves the convergence rate in linearly converging fixed-point methods (but not in those converging quadratically). *SIAM J Numer Anal*. 2020;58(1):788-810. doi:10.1137/19M1245384
19. Pollock S, Rebholz LG, Xiao M. Anderson-accelerated convergence of Picard iterations for incompressible Navier-Stokes equations. *SIAM J Numer Anal*. 2019;57(2):615-637. doi:10.1137/18M1206151
20. De Sterck H, He Y. On the asymptotic linear convergence speed of Anderson acceleration, Nesterov acceleration, and nonlinear GMRES. *SIAM J Sci Comput*. 2021;43(5):S21-S46. doi:10.1137/20M1347139
21. Wang D, He Y, De Sterck H. On the asymptotic linear convergence speed of Anderson acceleration applied to ADMM. *J Sci Comput*. 2021;88(2):38, 35. doi:10.1007/s10915-021-01548-2
22. Bian W, Chen X, Kelley CT. Anderson acceleration for a class of nonsmooth fixed-point problems. *SIAM J Sci Comput*. 2021;43(5):S1-S20. doi:10.1137/20M132938X
23. Brune PR, Knepley MG, Smith BF, Tu X. Composing scalable nonlinear algebraic solvers. *SIAM Rev*. 2015;57(4):535-565. doi:10.1137/130936725
24. Peng Y, Deng B, Zhang J, Geng F, Qin W, Liu L. Anderson acceleration for geometry optimization and physics simulation. *ACM Trans Graph (TOG)*. 2018;37(4):1-14. doi:10.1145/3197517.3201290
25. Shi W, Song S, Wu H, Hsu YC, Wu C, Huang G. Regularized Anderson acceleration for off-policy deep reinforcement learning. arXiv preprint arXiv:1909.03245, 2019.
26. Toth A, Ellis JA, Evans T, et al. Local improvement results for Anderson acceleration with inaccurate function evaluations. *SIAM J Sci Comput*. 2017;39(5):S47-S65. doi:10.1137/16M1080677
27. Yang Y. Anderson acceleration for seismic inversion. *Geophysics*. 2021;86(1):R99-R108. doi:10.1190/geo2020-0462.1
28. Zhang J, O'Donoghue B, Boyd S. Globally convergent type-I Anderson acceleration for nonsmooth fixed-point iterations. *SIAM J Optim*. 2020;30(4):3170-3197. doi:10.1137/18M1232772
29. Glowinski R, Keller HB, Reinhart L. Continuation-conjugate gradient methods for the least squares solution of nonlinear boundary value problems. *SIAM J Sci Stat Comput*. 1985;6(4):793-832. doi:10.1137/0906055
30. Pollock S, Rebholz LG. Anderson acceleration for contractive and noncontractive operators. *IMA J Numer Anal*. 2021;41(4):2841-2872. doi:10.1093/imanum/draa095
31. Chen K, Vuik C. Non-stationary Anderson acceleration with optimized damping. arXiv preprint arXiv:2202.05295, 2022. doi: 10.48550/arXiv.2202.05295
32. Brown J, Knepley MG, May DA, McInnes LC, Smith B. Composable linear solvers for multiphysics. Proceedings of the 2012 11th International Symposium on Parallel and Distributed Computing; 2012:55-62; IEEE.
33. Kirby RC, Mitchell L. Solver composition across the PDE/linear algebra barrier. *SIAM J Sci Comput*. 2018;40(1):C76-C98. doi:10.1137/17M1133208

34. van der Vorst HA, Vuik C. GMRESR: a family of nested GMRES methods. *Numer Linear Algebra Appl*. 1994;1(4):369-386. doi:10.1002/nla.1680010404
35. Vuik C. Solution of the discretized incompressible Navier-Stokes equations with the GMRES method. *Int J Numer Methods Fluids*. 1993;16(6):507-523. doi:10.1002/fld.1650160605
36. Pernice M, Walker HF. NITSOL: a Newton iterative solver for nonlinear systems; Vol. 19, 1998:30-318.
37. De Sterck H, He Y. Linear asymptotic convergence of anderson acceleration: fixed-point analysis. arXiv preprint arXiv:2109.14176, 2021 doi: 10.48550/arXiv.2109.14176

How to cite this article: Chen K, Vuik C. Composite Anderson acceleration method with two window sizes and optimized damping. *Int J Numer Methods Eng*. 2022;123(23):5964-5985. doi: 10.1002/nme.7096