The AAA Algorithm for Rational Approximation

王士銘

November 2025

1 Problem Description

In this paper, the author wants to construct a rational approximant $r(z)$ that accurately represents a function $f(z)$ given at discrete points $Z = \{z_1, z_2, \ldots, z_M\} \subset \mathbb{C}$. The challenge is to achieve high accuracy, numerical stability, and adaptability without requiring user intervention or prior knowledge of the number or location of poles.

This problem is important for several reasons. First, rational functions provide exponentially faster convergence than polynomials when the target function has nearby poles or singularities. Second, rational models can be used to extrapolate and infer the properties of f beyond the sampled region, which is valuable for the analytic continuation and identification of the system. Finally, a stable rational representation facilitates robust computation of poles and residues, enabling applications in reduced-order modeling, control, and interpolation.

In contrast to many previous algorithms, AAA operates on arbitrary sets of sample points. It is not restricted to an interval, disk, or specific geometry, and hence serves as a flexible tool for general rational approximation problems.

2 Method Overview

2.1 Barycentric Representation

The core of the AAA algorithm is the barycentric form of a rational function:

$$r(z) = \frac{\displaystyle\sum_{j=1}^{m} \frac{w_j f_j}{z - z_j}}{\displaystyle\sum_{j=1}^{m} \frac{w_j}{z - z_j}},$$

where z_j are distinct support points, $f_j = f(z_j)$ are known data values and w_j are weights determined by solving a linear least-squares problem. This expression ensures interpolation at the support points, i.e. $r(z_j) = f_j$ whenever $w_j \neq 0$. The barycentric form is numerically stable because the basis functions $1/(z - z_j)$ remain well-conditioned even when m is large.

2.2 Greedy Support-Point Selection

The AAA algorithm constructs $r(z)$ adaptively. Starting with no support points, it repeats the following steps:

1. Evaluate the residual $|f(z) - r_{m-1}(z)|$ at all sample points.
2. Choose the next support point z_m at which the residual is maximal.
3. Solve a least-squares problem for the weights $w_1, \ldots, w_m$ to minimize

$$\|fd - n\|_{Z(m)}, \quad \text{where } n(z) = \sum_{j=1}^{m} \frac{w_j f_j}{z - z_j},\ d(z) = \sum_{j=1}^{m} \frac{w_j}{z - z_j}.$$

4. Form the new approximant $r_m(z) = n(z)/d(z)$.

The algorithm ends when the residual is less than a prescribed tolerance (default 10^{-13} relative to $\|f(Z)\|_\infty$). The resulting rational function is of type $(m-1, m-1)$ and is typically free of numerical results.

2.3 Avoiding Froissart Doublets

Spurious pole–zero pairs can arise in any rational approximation because of numerical round-off. The AAA algorithm detects and removes them using a simple cleanup step: any pole with residue magnitude smaller than 10^{-13} is considered spurious, and the corresponding support point is removed. A final least-squares update then yields a clean rational function with only the genuine poles retained.

2.4 Computational Aspects

At each iteration, the method solves a small singular value decomposition (SVD) of the so-called Loewner matrix. If M is the number of sample points and m the number of support points, the cost is $O(Mm^3)$ operations. Because m rarely exceeds 50 in practice, the computational expense is modest. The algorithm exhibits excellent numerical stability and scalability, showing reliably even for data sampled on complex geometries or multiple disjoint regions.

3 Simple Implementation Example

To illustrate the algorithm, we consider approximating the function $f(x) = \tan(x)$ on $[-1.5, 1.5]$. This function has poles near $x = \pm\pi/2$, which makes the polynomial approximation inefficient, but ideal for rational methods. The following pseudocode outlines a simplified AAA implementation.

```
import numpy as np

Z = np.linspace(-1.5, 1.5, 200)
F = np.tan(Z)
R = np.mean(F)
support, f_support = [], []
```

```
tol = 1e-12

for m in range(1, 20):
    j = np.argmax(np.abs(F - R))
    support.append(Z[j]); f_support.append(F[j])
    C = 1.0 / (Z[:, None] - np.array(support))
    SF, Sf = np.diag(F), np.diag(f_support)
    A = SF @ C - C @ Sf
    _, _, Vh = np.linalg.svd(A)
    w = Vh[-1, :]
    N, D = C @ (w * np.array(f_support)), C @ w
    R = N / D
    if np.max(np.abs(F - R)) < tol * np.max(np.abs(F)):
        break
```

After a few iterations, the approximation reproduces the function to near machine precision over most of the sampled interval. Because the sampling interval excludes the actual poles of $tan(x)$, the algorithm behaves as a smooth rational interpolant rather than identifying singularities. Even though this pseudocode omits optimization details, it captures the essential logic of the AAA iteration.

4 Experimental Demonstration and Observations

The authors tested the algorithm on a lot of problems, including analytic and meromorphic functions, functions with branch points, and data sampled on disconnected sets. A few representative results can be summarized as follows:

- For analytic functions such as $f(z) = \tan(z)$ or $\log(1.1 - z)$, the AAA approximants achieve exponential convergence in the number of support points, often outperforming polynomial interpolation by many orders of magnitude.
- For meromorphic functions inside the unit disk, the method accurately reproduces the correct poles and residues, providing insight into analytic continuation and model reduction problems.
- In more complex geometries or disconnected domains, the barycentric form remains well conditioned, whereas polynomial-based methods would suffer from severe ill-conditioning.
- The algorithm successfully approximates real functions like $|x|$ and $\exp(x)$ on unbounded or semi-infinite intervals, achieving results close to theoretical minimax rates.

A key observation from these experiments is that the AAA algorithm is robust across diverse settings without modification of parameters. It also provides a natural balance between simplicity and accuracy: though not strictly optimal in any norm, its practical performance rivals that of specialized optimization-based methods.

5 Summary or conclusion

5.1 Advantages

The AAA algorithm has some advantages:

- The entire procedure fits in a short and transparent code, easy to implement in MATLAB, Python, or Julia.
- Support points are chosen greedily, eliminating the need for user-defined parameters or initial guesses.
- The barycentric representation avoids large intermediate coefficients, and the use of SVD ensures robustness in floating-point arithmetic.
- It applies to real or complex functions, arbitrary point sets, and even disconnected domains.

5.2 Disadvantages

Despite its success, several limitations remain:

- The resulting rational function is not guaranteed to be optimal in L_2 or L_∞ norms.
- The algorithm can introduce non-physical poles if the tolerance is too strict or if data contain noise.
- In problems requiring symmetry (e.g., real-valued, even, or odd functions), modifications are needed to enforce such constraints explicitly.
- The computational cost, though modest, grows cubically with the number of support points and can become significant for very large datasets.

5.3 Extensions and conclusion

Several extensions have been explored since this paper. Variants such as Lawson–AAA and AAA-LeastSquares aim to achieve better norm optimality. Other versions enforce real or even/odd symmetry or improve performance on noisy data through regularization. The AAA framework also serves as a foundation for model-reduction techniques in dynamical systems, where rational approximations describe frequency responses efficiently.

Overall, the algorithm demonstrates that a simple adaptive approach can achieve high accuracy, robustness, and universality in rational approximation—qualities that have made AAA a standard tool in modern numerical software such as Chebfun.

By combining the barycentric form with greedy adaptive selection and least-squares updates, it delivers accurate and stable rational fits with minimal user effort. Its flexibility across domains, immunity to most numerical instabilities, and ease of implementation have made it widely adopted in numerical analysis, signal processing, and scientific computing.

Although not theoretically optimal, its balance between simplicity and performance makes it an outstanding example of practical numerical innovation. Future research will likely extend AAA toward high-dimensional data, noisy measurements, and real-time model identification, continuing its role as a bridge between mathematical theory and computational practice.

References

[1] Y. Nakatsukasa, O. Sète, and L. N. Trefethen. *The AAA Algorithm for Rational Approximation*. SIAM Journal on Scientific Computing, 40(3), 2018.