Algorithms for the rational approximation
of matrix-valued functions

Ion Victor Gosea[*]   Stefan Güttel[†]

March 16, 2020

Abstract

A selection of algorithms for the rational approximation of matrix-valued functions are discussed, including variants of the interpolatory AAA method, the RKFIT method based on approximate least squares fitting, vector fitting, and a method based on low-rank approximation of a block Loewner matrix. A new method, called the block-AAA algorithm, based on a generalized barycentric formula with matrix-valued weights is proposed. All algorithms are compared in terms of obtained approximation accuracy and runtime on a set of problems from model order reduction and nonlinear eigenvalue problems, including examples with noisy data. It is found that interpolation-based methods are typically cheaper to run, but they may suffer in the presence of noise for which approximation-based methods perform better.

Keywords: rational approximation, block rational function, Loewner matrix.

1   Introduction

Rational approximation is a powerful tool in applied science and engineering. To give just two examples, it is very commonly used for model order reduction [2–4] and the solution of nonlinear eigenvalue problems [16, Section 6]. Recently, several new algorithms for the rational approximation and interpolation of scalar-valued functions have been proposed, including (in reverse chronological order) the AAA algorithm [21], the RKFIT algorithm [7, 8], vector fitting [13, 15], and methods based on the Loewner matrices [20]. The aim of this paper is to explore extensions of these methods for the purpose of approximating a matrix-valued (or block) function $F : \Lambda \to \mathbb{C}^{m \times n}$ on a discrete set Λ in the complex plane.

The paper contains two key contributions. Firstly, we propose an extension of a barycentric formula to non-scalar weights and develop it into a new algorithm for the computation of rational approximants of matrix-valued functions. This algorithm, referred to as the block-AAA algorithm, generalizes the AAA algorithm in [21]. Secondly, we perform extensive numerical comparisons of several rational approximation algorithms proposed by different communities (model reduction, numerical linear algebra, approximation theory). To the best of our knowledge, this is the first paper that offers a comprehensive study of these algorithms. As part of our

---

[*]Max Planck Institute for Dynamics of Complex Technical Systems Magdeburg, 39106 Magdeburg, Germany, gosea@mpi-magdeburg.mpg.de

[†]The University of Manchester, Department of Mathematics, Manchester M13 9PL, United Kingdon, stefan.guettel@manchester.ac.uk. This author has been supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1.

experiments, we identify potential problems of interpolation-based algorithms when the data samples are polluted by noise.

The paper is structured as follows. Section 2 contains a brief review of existing algorithms for scalar rational approximation. In Section 3 we introduce new representations of matrix-valued rational approximants and the new block-AAA algorithm. In Section 4 we briefly review other popular algorithms for rational approximation, including methods based on partial fractions, rational Krylov spaces, and state space representations. Section 5 contains a collection of numerical experiments and discussions, guided by six examples from different application areas. Section 6 further discusses the performance of the different methods for each numerical example. We conclude in Section 7 with some ideas for future work.

2 Scalar rational approximation

In this section we summarize several algorithms for the rational approximation of a scalar function $f : \Lambda \to \mathbb{C}$ sampled at a nonempty set of points $\Lambda := \{\lambda_1, \dots, \lambda_\ell\} \subset \mathbb{C}$. This serves as an introduction of our notation, but also as a template for the block variants in Sections 3 and 4.

2.1 Adaptive Antoulas–Anderson (AAA) algorithm

The AAA algorithm proposed in [21] is a practically robust and easy-to-use method for scalar rational interpolation. The degree d interpolant r_d obtained after d iterations of the AAA algorithm is of the form

$$r_d(z) = \frac{\displaystyle\sum_{k=0}^{d} \frac{w_k f_k}{z - z_k}}{\displaystyle\sum_{k=0}^{d} \frac{w_k}{z - z_k}}, \tag{1}$$

with nonzero barycentric weights $w_k \in \mathbb{C}$, pairwise distinct support points $z_k \in \mathbb{C}$, and function values $f_k = f(z_k)$. A key ingredient of the AAA algorithm is its greedy choice of the support points, one per iteration $j = 0, 1, \dots, d$, intertwined with the solution of a least squares problem to determine the barycentric weights $w_0^{(j)}, \dots, w_j^{(j)}$ at each iteration j. The core AAA algorithm can be summarized as follows:

1. Set $j = 0$, $\Lambda^{(0)} := \Lambda$, and $r_{-1} :\equiv \ell^{-1} \sum_{i=1}^{\ell} f(\lambda_i)$.

2. Select a point $z_j \in \Lambda^{(j)}$ where $|f(z) - r_{j-1}(z)|$ is maximal, with

$$r_{j-1}(z) := \frac{\displaystyle\sum_{k=0}^{j-1} \frac{w_k^{(j-1)} f_k}{z - z_k}}{\displaystyle\sum_{k=0}^{j-1} \frac{w_k^{(j-1)}}{z - z_k}} \quad \text{for } j \geq 1.$$

3. If $|f(z_j) - r_{j-1}(z_j)|$ is small enough, return r_{j-1} and stop.

4. Set $f_j := f(z_j)$ and $\Lambda^{(j+1)} := \Lambda^{(j)} \setminus \{z_j\}$.

5. Compute weights $w_0^{(j)}, \dots, w_j^{(j)}$ with $\sum_{k=0}^{j} |w_k^{(j)}|^2 = 1$ such that

$$\sum_{k=0}^{j} \frac{w_k^{(j)}}{z - z_k} f(z) \approx \sum_{k=0}^{j} \frac{w_k^{(j)} f_k}{z - z_k}$$

   is solved with least squares error over all $z \in \Lambda^{(j+1)}$.

6. Set $j := j + 1$ and go to step 2.

MATLAB implementations of the AAA algorithm can be found in [21] and the Chebfun package [11] available at

https://www.chebfun.org/

In [21, Section 10] a non-interpolatory variant of the AAA algorithm is also mentioned, which is obtained by allowing the weights in the numerator and denominator of (1) to be different.

2.2 Rational Krylov Fitting (RKFIT)

The RKFIT method introduced in [7, 8] is based on iteratively relocating the poles of a rational function in the so-called RKFUN representation. An RKFUN is a triple $r_d \equiv (\underline{H_d}, \underline{K_d}, \mathbf{c})$ with upper-Hessenberg matrices $\underline{H_d}, \underline{K_d} \in \mathbb{C}^{(d+1) \times d}$ and a coefficient vector $\mathbf{c} \in \mathbb{C}^{d+1}$. For a given point $z \in \mathbb{C}$, the RKFUN $r_d(z)$ is evaluated as a dot-product $r_d(z) := \mathbf{n}(z) \cdot \mathbf{c}$, where $\mathbf{n}(z)$ is the unique left nullvector of $z \underline{K_d} - \underline{H_d}$ normalized such that its first component is 1 (note that the unique existence of this nullvector is guaranteed by the fact that $z \underline{K_d} - \underline{H_d}$ is an unreduced upper-Hessenberg matrix).

The matrices $\underline{H_d}$ and $\underline{K_d}$ of the RKFUN representation satisfy a rational Arnoldi decomposition $A V_{d+1} \underline{K_d} = V_{d+1} \underline{H_d}$ associated with a rational Krylov space

$$\mathcal{Q}_{d+1}(A, \mathbf{b}) := q(A)^{-1} \text{span}\{\mathbf{b}, A\mathbf{b}, \dots, A^d \mathbf{b}\},$$

with $A \in \mathbb{C}^{\ell \times \ell}$, $\mathbf{b} \in \mathbb{C}^{\ell}$, $V_{d+1} \in \mathbb{C}^{\ell \times (d+1)}$, and a monic polynomial $q \in \mathcal{P}_d$ such that $q(A)$ is invertible. It can be shown that the columns of $V_{d+1} = [\mathbf{v}_1, \dots, \mathbf{v}_{d+1}]$ are all of the form $\mathbf{v}_{k+1} = p_k(A) q(A)^{-1} \mathbf{b}$ with polynomials $p_k \in \mathbb{P}_d$ for $k = 0, 1, \dots, d$. In other words, the rational Arnoldi decomposition encodes a basis of scalar rational functions $r_k := p_k / q$ all sharing the same denominator q. Further, one can show that the roots ξ_j of $q(z) = \prod_{\xi_j \neq \infty} (z - \xi_j)$ correspond to the quotients of the subdiagonal elements of the upper-Hessenberg matrices $\underline{H_d}$ and $\underline{K_d}$, i.e., $\xi_j = h_{j+1,j} / k_{j+1,j}$ for $j = 1, \dots, d$. These quotients are referred to as the *poles* of the rational Arnoldi decomposition [7].

Given a matrix $F \in \mathbb{C}^{\ell \times \ell}$, the RKFIT algorithm attempts to identify an RKFUN $r_d \equiv (\underline{H_d}, \underline{K_d}, \mathbf{c})$ such that $r_d(A)\mathbf{b} \approx F\mathbf{b}$ in the least squares sense. Starting with an initial rational Arnoldi decomposition $A V_{d+1} \underline{K_d} = V_{d+1} \underline{H_d}$ having poles $\xi_1^{(0)}, \dots, \xi_d^{(0)}$, one RKFIT iteration consists of solving a least squares problem for finding a unit-norm vector $\widehat{\mathbf{v}} \in \text{span}(V_{d+1})$ such that

$$\widehat{\mathbf{v}} = \underset{\substack{\mathbf{v} = V_{d+1} \mathbf{c} \\ \|\mathbf{c}\|_2 = 1}}{\arg \min} \| (I - V_{d+1} V_{d+1}^*) F \mathbf{v} \|_2,$$

and to use the roots of $\widehat{r} = \widehat{p}/q$ associated with $\widehat{\mathbf{v}} = \widehat{r}(A)\mathbf{b}$ as the new poles $\xi_1^{(1)}, \ldots, \hat{\xi}_d^{(1)}$ for the next iteration. This process is iterated until a convergence criterion is satisfied.

A MATLAB implementation of RKFIT is provided in the Rational Krylov Toolbox [6] available at

<div align="center">

http://www.rktoolbox.org/

</div>

RKFIT can naturally deal with scalar rational approximation by choosing diagonal matrices $A = \text{diag}(\lambda_1, \ldots, \lambda_\ell)$, $F = \text{diag}(f_1, \ldots, f_\ell)$, and $\mathbf{b} = [1, 1, \ldots, 1]^T$. We emphasize that RKFIT is a *non-interpolatory* method.

2.3 Vector Fitting (VF)

The VF algorithm, originally proposed in [15], seeks to fit a rational function in partial fraction (pole–residue) form

$$r_d(z) = \delta + \sum_{k=1}^{d} \frac{\gamma_k}{z - \xi_k} = \frac{p(z)}{q(z)}.$$

The iterative algorithm is initiated by choosing the degree d of the rational approximant and an initial guess for the poles $\{\xi_1^{(0)}, \ldots, \xi_d^{(0)}\}$. At iteration $j = 0, 1, \ldots$ one determines parameters $c_k^{(j)}$ and $d_k^{(j)}$ such that

$$\underbrace{\sum_{k=1}^{d} \frac{c_k^{(j)}}{z - \xi_k^{(j-1)}} + c_0^{(j)}}_{p^{(j)}(z)} \approx \Big(\underbrace{\sum_{k=1}^{d} \frac{d_k^{(j)}}{z - \xi_k^{(j-1)}} + 1}_{q^{(j)}(z)} \Big) f(z)$$

is solved with least squares error over all $z \in \Lambda$. Afterwards, the next set of poles $\{\xi_1^{(j)}, \ldots, \xi_d^{(j)}\}$ is computed as the roots of the polynomial $q^{(j)}(z)$ by solving a linear eigenvalue problem. The iteration continues until a convergence criterion is satisfied. Vector fitting is a *non-interpolatory* method. A MATLAB implementation of this method is available at

<div align="center">

https://www.sintef.no/projectweb/vectorfitting/

</div>

2.4 Loewner framework (LF)

This LF method uses the full data set in matrix format by forming (possibly very large) Loewner matrices. The original method, introduced in [20], is based on constructing a reduced-order rational function in state-space representation. In recent years, the Loewner framework has been extended to classes of mildly nonlinear systems, including bilinear systems [5].

Assuming that ℓ, the number of sampling points, is an even positive integer, the first step in the Loewner framework is to partition $\Lambda = \Lambda^L \cup \Lambda^R$ into two disjoint sets of the same cardinality. The set Λ^L contains the left points $\{x_1, \ldots, x_{\ell/2}\}$, while Λ^R contains the right points $\{y_1, \ldots, y_{\ell/2}\}$. Similarly, the set of scalar samples (evaluations of the function f on Λ) is partitioned into two sets.

One then defines matrices $\mathbb{L} \in \mathbb{C}^{\ell/2 \times \ell/2}$ (the Loewner matrix) and $\mathbb{L}^s \in \mathbb{C}^{\ell/2 \times \ell/2}$ (the shifted Loewner matrix) with entries

$$\mathbb{L}_{i,j} = \frac{f(x_i) - f(y_j)}{x_i - y_j} \quad \text{and} \quad \mathbb{L}_{i,j}^s = \frac{x_i f(x_i) - y_j f(y_j)}{x_i - y_j},$$

respectively. Additionally, one defines vectors $V \in \mathbb{C}^{\ell/2 \times 1}$, $W \in \mathbb{C}^{1 \times \ell/2}$ as

$$V = \left[f(x_1), f(x_2), \ldots, f(x_{\ell/2}) \right]^T, \quad W = \left[f(y_1), f(y_2), \ldots, f(y_{\ell/2}) \right]. \quad (2)$$

The next step is to compute a rank-d truncated singular value decomposition of the Loewner matrix $\mathbb{L} \approx XSZ^*$ with $X \in \mathbb{C}^{\ell/2 \times d}$, $S \in \mathbb{C}^{d \times d}$ and $Z \in \mathbb{C}^{\ell/2 \times d}$. Finally, by means of projecting with the matrices X and Z, the fitted rational function is

$$r_d(z) = WZ \left(X^*(\mathbb{L}^s - z\mathbb{L})Z \right)^{-1} X^*V. \quad (3)$$

This is a subdiagonal rational function of type $(d-1, d)$ whose d poles are given by the generalized eigenvalues of the matrix pair $(X^*\mathbb{L}^s Z, X^*\mathbb{L}Z)$.

3 Matrix-valued barycentric forms and block-AAA

The simplest matrix-valued barycentric form is obtained from (1) by replacing the function values f_k with matrices $F_k := F(z_k)$:

$$R_d(z) = \frac{\displaystyle\sum_{k=0}^{d} \frac{w_k F_k}{z - z_k}}{\displaystyle\sum_{k=0}^{d} \frac{w_k}{z - z_k}}. \quad \text{(bary-A)}$$

Provided that all weights w_k are nonzero, the function R_d interpolates the function F at all the support points z_k. Each (i, j) entry of R_d is a rational function of the form $p_{ij}(z)/q(z)$, where $p_{ij}, q \in \mathcal{P}_d$ are polynomials of degree d. Note that all these entries share the same *scalar* denominator q.

A slight modification of (bary-A) yields a new matrix-valued barycentric formula

$$R_d(z) = \left(\sum_{k=0}^{d} \frac{W_k}{z - z_k} \right)^{-1} \left(\sum_{k=0}^{d} \frac{W_k F(z_k)}{z - z_k} \right) \quad \text{(bary-B)}$$

with weight matrices $W_k \in \mathbb{C}^{m \times m}$. If all these W_k are nonsingular, R_d interpolates F at all the support points z_k. Given a set of support points $z_0, \ldots, z_k$ different from any point in Λ, a linearized version of the approximation problem $R_d(\lambda_i) \approx F(\lambda_i)$ is

$$\sum_{i=1}^{\ell} \left\| \left(\sum_{k=0}^{d} \frac{W_k}{\lambda_i - z_k} \right) F(\lambda_i) - \sum_{k=0}^{d} \frac{W_k F(z_k)}{\lambda_i - z_k} \right\|_F^2 \to \min_{W_k}.$$

The weight matrices can now be obtained from a trailing left-singular block vector $[W_0, \ldots, W_d]$ of unit norm such that

$$[W_0, \ldots, W_d] \begin{bmatrix} \frac{F(\lambda_1)-F(z_0)}{\lambda_1-z_0} & \cdots & \frac{F(\lambda_\ell)-F(z_0)}{\lambda_\ell-z_0} \\ \vdots & & \vdots \\ \frac{F(\lambda_1)-F(z_d)}{\lambda_1-z_d} & \cdots & \frac{F(\lambda_\ell)-F(z_d)}{\lambda_\ell-z_d} \end{bmatrix} =: \mathbb{W}\mathbb{L}$$
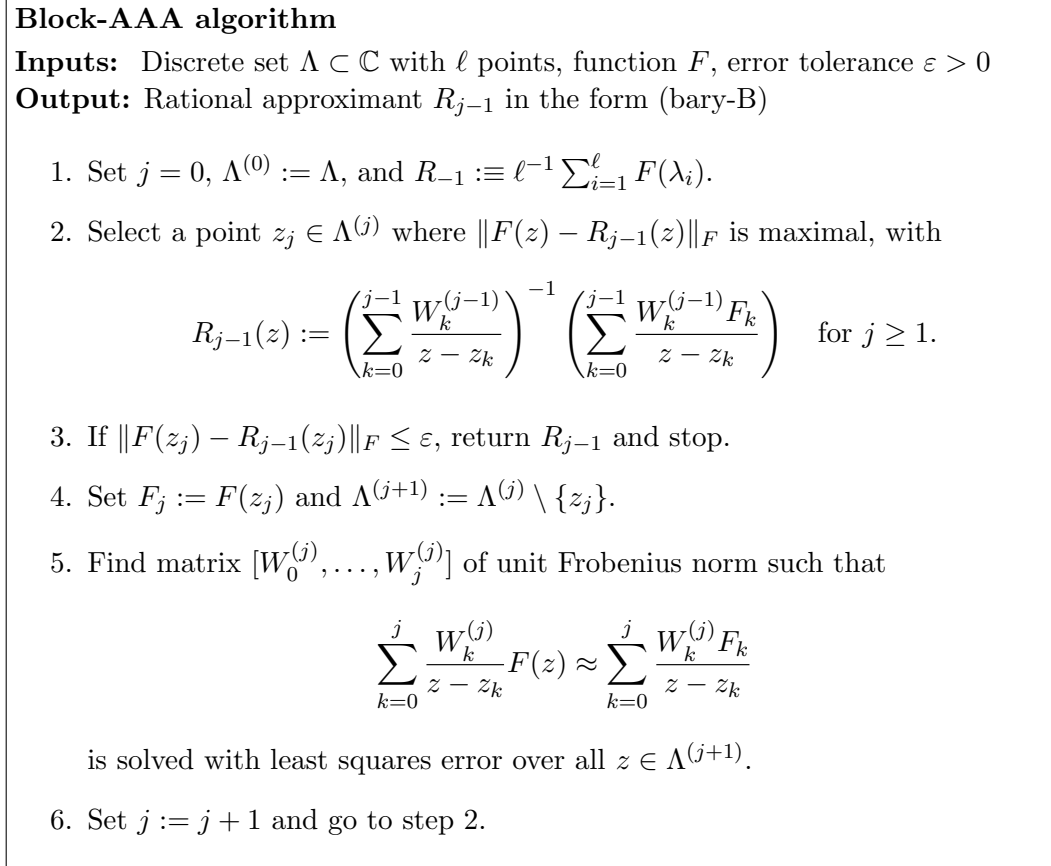

Block-AAA algorithm

Inputs: Discrete set $\Lambda \subset \mathbb{C}$ with ℓ points, function F, error tolerance $\varepsilon > 0$

Output: Rational approximant R_{j-1} in the form (bary-B)

1. Set $j = 0$, $\Lambda^{(0)} := \Lambda$, and $R_{-1} :\equiv \ell^{-1} \sum_{i=1}^{\ell} F(\lambda_i)$.

2. Select a point $z_j \in \Lambda^{(j)}$ where $\|F(z) - R_{j-1}(z)\|_F$ is maximal, with

$$R_{j-1}(z) := \left(\sum_{k=0}^{j-1} \frac{W_k^{(j-1)}}{z - z_k} \right)^{-1} \left(\sum_{k=0}^{j-1} \frac{W_k^{(j-1)} F_k}{z - z_k} \right) \quad \text{for } j \geq 1.$$

3. If $\|F(z_j) - R_{j-1}(z_j)\|_F \leq \varepsilon$, return R_{j-1} and stop.

4. Set $F_j := F(z_j)$ and $\Lambda^{(j+1)} := \Lambda^{(j)} \setminus \{z_j\}$.

5. Find matrix $[W_0^{(j)}, \ldots, W_j^{(j)}]$ of unit Frobenius norm such that

$$\sum_{k=0}^{j} \frac{W_k^{(j)}}{z - z_k} F(z) \approx \sum_{k=0}^{j} \frac{W_k^{(j)} F_k}{z - z_k}$$

   is solved with least squares error over all $z \in \Lambda^{(j+1)}$.

6. Set $j := j + 1$ and go to step 2.

Figure 1: Pseudocode of the block-AAA algorithm

has smallest possible norm. Note that the matrix $\mathbb{L}$ is a block Loewner matrix.[1] It is now easy to derive an AAA-like algorithm based on (bary-B). It is shown in Figure 3 and referred to as the *block-AAA algorithm*.

A MATLAB implementation of the block-AAA algorithm is available at

`https://github.com/nla-group/block_aaa`

As it will become clearer from the discussions in Section 4, the block-AAA algorithm is different from the set-valued AAA [19] and the fast-AAA [18] algorithms: the entries of a block-AAA approximant R_d *do not* share a common scalar denominator of degree d. Indeed, a block-AAA approximant R_d of order d can have a larger number of up to dm poles, i.e., its McMillan degree can be as high as dm. For this reason we refer to the *order* of a block-AAA approximant instead of its degree.

Finally, we mention for completeness two other barycentric formulas which might be of interest elsewhere. One is given as

$$R_d(z) = \left(\sum_{k=0}^{d} \frac{D_k}{z - z_k} \right)^{-1} \left(\sum_{k=0}^{d} \frac{C_k}{z - z_k} \right) \tag{bary-C}$$

---

[1] In the Loewner framework discussed in Section 2.4, the Loewner matrix entries

$$\mathbb{L}_{ij} = \frac{\mathbf{v}_i^* \mathbf{r}_j - \boldsymbol{\ell}_i^* \mathbf{w}_j}{\mu_i - \lambda_j}$$

are defined in terms of left and right directions $(\mu_i, \boldsymbol{\ell}_i, \mathbf{v}_i)$ and $(\lambda_j, \mathbf{r}_j, \mathbf{w}_j)$, but here we have a special case where the left and right directions $\boldsymbol{\ell}_i$ and $\mathbf{r}_j$ are chosen as the unit vectors.

with the matrices $C_k \in \mathbb{C}^{m \times n}$ and $D_k \in \mathbb{C}^{m \times m}$ chosen such that $R_\ell(\lambda_i) \approx F(\lambda_i)$. We assume that the z_k and λ_i are pairwise distinct and consider the linearized problem

$$\sum_{i=1}^{\ell} \left\| \left(\sum_{k=0}^{d} \frac{D_k}{\lambda_i - z_k} \right) F(\lambda_i) - \sum_{k=0}^{d} \frac{C_k}{\lambda_i - z_k} \right\|_F^2 \to \min_{C_k, D_k} .$$

This least squares problem is solved by a trailing left-singular block vector given by $[C_0, \ldots, C_d, D_0, \ldots, D_d]$ of unit norm such that

$$[C_0, \ldots, C_d, D_0, \ldots, D_d] \begin{bmatrix} \frac{-I}{\lambda_1 - z_0} & \cdots & \frac{-I}{\lambda_\ell - z_0} \\ \vdots & & \vdots \\ \frac{-I}{\lambda_1 - z_d} & \cdots & \frac{-I}{\lambda_\ell - z_d} \\ \frac{F(\lambda_1)}{\lambda_1 - z_0} & \cdots & \frac{F(\lambda_\ell)}{\lambda_\ell - z_0} \\ \vdots & & \vdots \\ \frac{F(\lambda_1)}{\lambda_1 - z_d} & \cdots & \frac{F(\lambda_\ell)}{\lambda_\ell - z_d} \end{bmatrix}$$

has smallest possible norm. Provided that the matrices D_k are nonsingular, we have $R_d(z_k) = D_k^{-1} C_k$ and in general the approximation is *non-interpolatory*. Such approximants might be useful for the solution of nonlinear eigenvalue problems; see also the appendix.

It is also possible to have matrix-valued "support points" $Z_k \in \mathbb{C}^{m \times m}$, resulting in

$$R_d(z) = \left(\sum_{k=0}^{d} (zI - Z_k)^{-1} D_k \right)^{-1} \left(\sum_{k=0}^{d} (zI - Z_k)^{-1} C_k \right). \qquad \text{(bary-D)}$$

This leads to tangential interpolation determined by the eigenvalues and eigenvectors of the matrices Z_k. We do not further explore this form here.

4 Other block methods

In this section we list methods for the rational approximation of matrix-valued functions using the barycentric formula (bary-A), as well as methods based on other representations of their approximants. These methods will be compared numerically in Sections 5 and 6.

Set-valued AAA (fast-AAA): The set-valued AAA algorithm presented in [19], and similarly the fast-AAA algorithm in [18], applies the standard AAA algorithm to each component of F using common weights and support points for all of them, thereby effectively producing a barycentric interpolant in the form (bary-A).

Surrogate AAA: This method presented in [12] applies the standard AAA algorithm to a scalar surrogate function $f(z) := \mathbf{a}^T F(z) \mathbf{b}$, with vectors $\mathbf{a}, \mathbf{b}$ chosen at random. The resulting block interpolant is of the form (bary-A), obtained by replacing the scalar values $f_k = f(z_k)$ by the matrices $F_k = F(z_k)$.

RKFIT (block): The RKFIT algorithm has been generalized in [8] to the problem of finding a family $r_d^{[k]}(z) = p^{[k]}(z)/q(z)$ of scalar rational functions such that $r_d^{[k]}(z) \approx f^{[k]}(z)$ in the least squares sense over all $z \in \Lambda$. All computed functions

$r_d^{[k]}$ are represented in the RKFUN format and share the common scalar denominator q of degree d. In our setting, we simply associate with each (i, j) entry of F a corresponding function $f^{[k]}(z)$ and run RKFIT on that family.

Matrix Fitting: The extension of vector fitting to matrix-valued data has been implemented in the MATLAB Matrix Fitting Toolbox available at

> https://www.sintef.no/projectweb/vectorfitting/downloads/
> matrix-fitting-toolbox/

As described in the manual [14] associated with the toolbox, the primary intention of this software is the rational multi-port modelling of data in the frequency domain. In particular, the toolbox deals with fitting so-called admittance values (known also as Y parameters) and scattering values (known also as S parameters).

The fitted rational function is provided in the pole–residue form

$$R_d(z) = D + \sum_{k=1}^{d} \frac{C_k}{z - \xi_k},$$

or as an equivalent state space model

$$R_d(z) = D + C(zI - A)^{-1}B,$$

where $D, C_k \in \mathbb{C}^{n \times n}$ for $k = 1, \ldots, d$, $C \in \mathbb{C}^{n \times nd}$, $A \in \mathbb{C}^{nd \times nd}$, and $B \in \mathbb{C}^{nd \times n}$. For both representations, stable poles can be enforced. Note that the matrix elements of R_d share a common set of d poles.

The above mentioned matrix fitting implementation [14] works with the elements of the upper triangular part of the matrix samples $F_i := F(\lambda_i)$ $(i = 1, \ldots, \ell)$, i.e., the entries $[F_i]_{a,b}$ for $a \leq b$, which are stacked into a single vector. Hence, this implementation can be used only for symmetric matrix-valued functions. The elements of the vector are fitted using an implementation of the "relaxed version" of vector fitting [13], provided in the function `vectfit3.m` of the toolbox. The main driver function `VFdriver.m` accepts as input arguments sample points $\Lambda = \overline{\Lambda} = \{\lambda_1, \ldots, \lambda_\ell\}$ on the imaginary axis in complex conjugate pairs (see Section 3.1, page 7 of the toolbox manual). In our context of fitting on arbitrary discrete sets Λ in the complex plane, this requirement might represent an inconvenience. Also, the available driver seems to be intended to be run for even orders d only, although this is not an inherent restriction of vector fitting.

Block Loewner framework: The extension of the Loewner framework to matrix-valued functions is covered in [20]. Therein, the case of directionally-sampled matrix data is treated in detail. Introducing left directional vectors $\boldsymbol{\ell}_i \in \mathbb{C}^m$ and right directional vectors $\mathbf{r}_j \in \mathbb{C}^n$, one defines a Loewner matrix $\mathbb{L} \in \mathbb{C}^{\ell/2 \times \ell/2}$ and a shifted Loewner matrix as follows:

$$\mathbb{L}_{i,j} = \frac{\boldsymbol{\ell}_i^* F(x_i)\mathbf{r}_j - \boldsymbol{\ell}_i^* F(y_j)\mathbf{r}_j}{x_i - y_j},$$

$$\mathbb{L}_{i,j}^s = \frac{x_i \boldsymbol{\ell}_i^* F(x_i)\mathbf{r}_j - y_j \boldsymbol{\ell}_i^* F(y_j)\mathbf{r}_j}{x_i - y_j}.$$

The definition of the matrices $V \in \mathbb{C}^{\ell/2 \times n}$, $W \in \mathbb{C}^{m \times \ell/2}$ is also different from those in the scalar case in (2), namely the directional vectors appear in the form

$$V^* = \left[F(x_1)^* \boldsymbol{\ell}_1, F(x_2)^* \boldsymbol{\ell}_2, \ldots, F(x_{\ell/2})^* \boldsymbol{\ell}_{\ell/2} \right],$$
$$W = \left[F(y_1)\mathbf{r}_1, F(y_2)\mathbf{r}_2, \ldots, F(y_{\ell/2})\mathbf{r}_{\ell/2} \right].$$

The rational matrix-valued function R_d is then computed exactly as in the scalar case (3) in the form

$$R_d(z) = WZ\left(X^*(\mathbb{L}^s - z\mathbb{L})Z\right)^{-1} X^*V,$$

where the truncated singular vector matrices X, Z are the same as before.

5 Numerical comparisons

In all our tests we assume that $\Lambda = \{\lambda_1, \ldots, \lambda_\ell\}$ and all the samples $F(\lambda_i)$ are given. We will compare the six algorithms introduced in the previous sections, using MATLAB implementations provided by their authors whenever available. In order to compare the algorithms with respect to their approximation performance, we use the root mean squared error

$$\texttt{RMSE} = \left(\frac{1}{\ell}\sum_{i=1}^{\ell}\|F(\lambda_i) - R_d(\lambda_i)\|_F^2\right)^{1/2}.$$

As before, d generally refers to the order of the rational approximant R_d, but it is the same as the degree for all algorithms except block-AAA. The RMSE values are reported for various examples and different different orders d in Tables 1, 2, 3, 4, and 5.

We also report timings for all algorithms under consideration. Although MATLAB timings might not always be reliable, the differences in the runtimes are usually large enough to give some indication of algorithmic complexity. In order to reduce random fluctuations in the timings, we run each algorithm for 20 times and compute the average execution time for a single run. All experiments were performed on a desktop computer with 8 GB RAM and an Intel(R) Core(TM) i7-7500U CPU running at 2.70 GHz.

5.1 Two toy examples

We first consider a rational function given by

$$F(z) = \begin{bmatrix} \frac{2}{z+1} & \frac{3-z}{z^2+z-5} \\ \frac{3-z}{z^2+z-5} & \frac{2+z^2}{z^3+3z^2-1} \end{bmatrix}.$$

The entries of F can be written with a common scalar denominator of degree $d = 6$, e.g., $q(z) = (z + 1)(z^2 + z - 5)(z^3 + 3z^2 - 1)$. We use $\ell = 100$ logarithmically spaced samples of F on the imaginary axis in the interval $[1, 100]$i. As in addition the samples of F are symmetric matrices, this example is suitable for the Matrix Fitting Toolbox [14].

The RMSE values obtained with all discussed algorithms for varying orders $d = 0, 1, \ldots, 20$ are shown in Figure 2 (left). Note that all methods eventually recover the function accurately. The Loewner approach requires a slightly higher order than the other methods, i.e., $d_1 = 8$. The block-AAA algorithm correctly identifies F when the order is $d_2 = 5$.

Next, we modify the (1,2) entry of the function F by replacing its denominator by $z^2 + z + 5$. The entries of F can then be written with a common denominator of degree $d = 8$, e.g., $q(z) = (z+1)(z^2 + z - 5)(z^2 + z + 5)(z^3 + 3z^2 - 1)$. The RMSE values are shown in Figure 2 (right). Now the VF approach fails as the modified
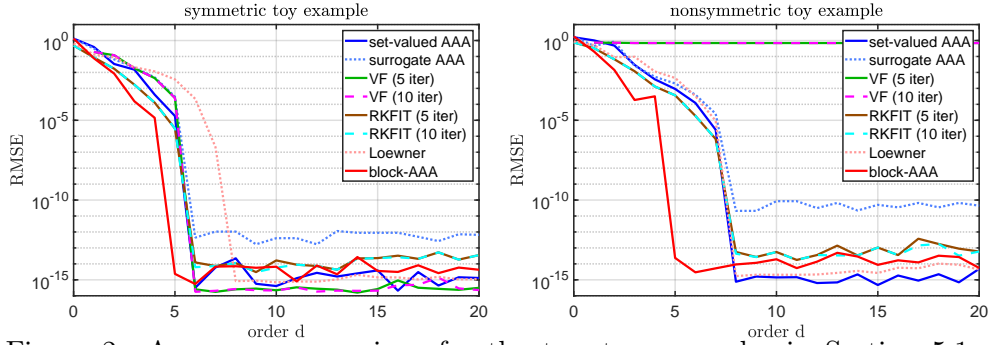
Figure 2: Accuracy comparison for the two toy examples in Section 5.1. Left: approximating a symmetric 2×2 matrix-valued function. Right: modified non-symmetric case.

function F is no longer symmetric. The block-AAA algorithm again identifies F correctly with an order of $d_1 = 5$, while all other methods require a degree of $d_2 = 8$ as expected.

5.2 An example from the MF Toolbox

We now choose an example from the Matrix Fitting Toolbox [14]. We use the file `ex2_Y.m` provided therein, containing 300 samples of 3×3 admittance matrices for a three-conductor overhead line. The sampling points are all on the imaginary axis within the interval $62.8319 \cdot [1, 10^4]$i. Figure 3 (left) depicts the magnitude of the nine matrix entries over the sampled frequency range.
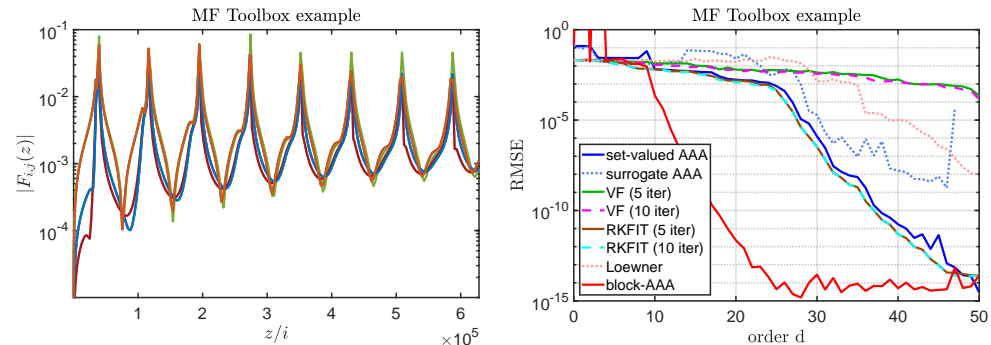
Figure 3: Example with admittance matrices from the MF Toolbox, described in Section 5.2. Left: entries of the 3×3 matrix $F(z)$ over the frequency range. Right: accuracy performance.

Figure 3 (right) shows the RMSEs achieved by each of the tested algorithms for varying orders $d = 0, 1, \ldots, 50$. For two particular orders, namely $d = 10$ and $d = 20$, we report the numerical RMSE values and the corresponding timings in Table 1.

5.3 A first model order reduction example

We now consider the `CD player` example from the SLICOT benchmark collection [1]. The mechanism to be modelled is a swing arm on which a lens is mounted by means of two horizontal leaf springs. The challenge is to find a low-cost controller that can make the servo-system faster and less sensitive to external shocks. The LTI system that models the dynamics has 60 vibration modes, hence the dimension

Table 1: Selected RMSE values and timings for all tested algorithms — MF Toolbox example

	RMSE		Runtime (ms)	
	$d = 10$	$d = 20$	$d = 10$	$d = 20$
Set-valued AAA	$6.522 \cdot 10^{-3}$	$1.707 \cdot 10^{-3}$	14.7	24.7
Surrogate AAA	$1.278 \cdot 10^{-2}$	$2.501 \cdot 10^{-2}$	11.6	18.5
VF (5 iter)	$1.374 \cdot 10^{-2}$	$6.143 \cdot 10^{-3}$	38.3	69.4
VF (10 iter)	$1.139 \cdot 10^{-2}$	$6.069 \cdot 10^{-3}$	55.1	103.6
RKFIT (5 iter)	$6.619 \cdot 10^{-3}$	$1.568 \cdot 10^{-3}$	80.2	182.6
RKFIT (10 iter)	$6.246 \cdot 10^{-3}$	$1.233 \cdot 10^{-3}$	132.5	302.2
Loewner	$1.795 \cdot 10^{-2}$	$1.675 \cdot 10^{-2}$	21.7	33.3
block-AAA	$1.421 \cdot 10^{-4}$	$8.625 \cdot 10^{-12}$	122.3	288.0

$n = 120$. Additionally, there are $m = 2$ inputs and $p = 2$ outputs. See [10] for more details.

The system's transfer function is a rational function of size 2×2 which we sample at 200 logarithmically spaced points in the interval $[10^1, 10^5]$i. Figure 4 (left) shows the magnitude of the 4 matrix entries over the frequency range. Figure 4 (right) shows the RMSE values achieved by each of the algorithms for varying orders $d = 0, 1, \ldots, 80$. The numerical RMSE values and the corresponding timings for orders $d = 10$ and $d = 20$ are listed in Table 2.
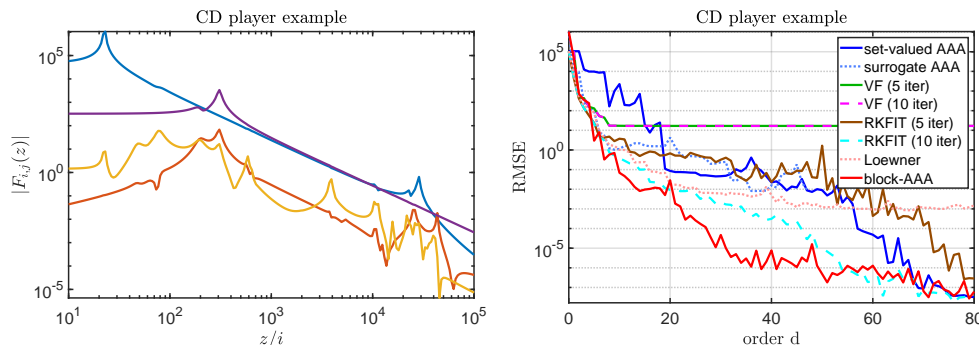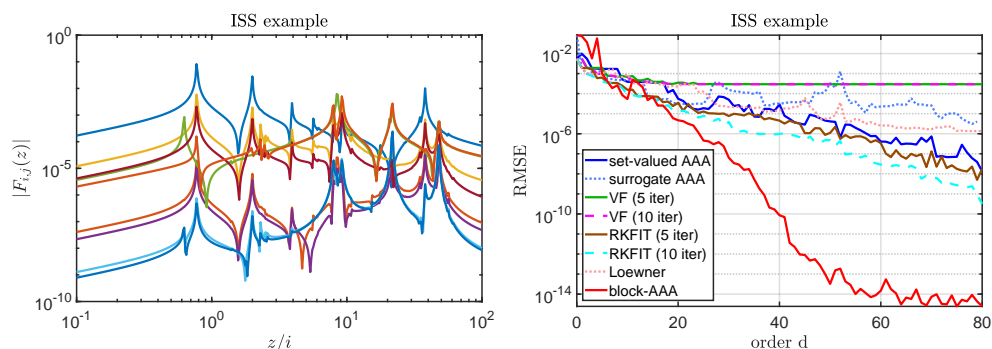

Figure 4: `CD player` example from Section 5.3. Left: entries of the 2×2 matrix $F(z)$ evaluated at 200 points on the imaginary axis. Right: accuracy performance.

5.4 A second model order reduction example

We consider another model reduction example, namely the ISS model from the SLICOT benchmark collection [1]. There, an LTI system is used as structural model for the component 1R (Russian service module) in the International Space Station (ISS). The state space dimension of the linear system is 270 with $m = n = 3$ inputs and outputs. The matrix transfer function of this system is sampled at 400 logarithmically spaced points in the interval $[10^{-1}, 10^2]$i. In Figure 5 (left) we depict the absolute value of the matrix entries. Figure 5 (right) shows the RMSE achieved by each of the algorithms for varying orders $d = 0, 1, \ldots, 80$, and selected numerical RMSE values and corresponding timings for orders $d = 10$ and $d = 20$ are given in Table 3.

Table 2: Selected RMSE values and timings for all tested algorithms — `CD player` example

	RMSE		Runtime (ms)	
	$d = 10$	$d = 20$	$d = 10$	$d = 20$
Set-valued AAA	$2.258 \cdot 10^3$	$8.564 \cdot 10^{-2}$	15.3	16.9
Surrogate AAA	$1.129 \cdot 10^0$	$4.301 \cdot 10^0$	8.7	13.8
VF (5 iter)	$1.702 \cdot 10^1$	$1.675 \cdot 10^1$	35.7	42.4
VF (10 iter)	$1.683 \cdot 10^1$	$1.674 \cdot 10^1$	75.4	59.5
RKFIT (5 iter)	$3.675 \cdot 10^0$	$5.270 \cdot 10^{-1}$	70.8	143.6
RKFIT (10 iter)	$3.806 \cdot 10^{-1}$	$9.061 \cdot 10^{-3}$	114	245.1
Loewner	$1.011 \cdot 10^0$	$1.784 \cdot 10^{-2}$	11.3	17
block-AAA	$6.897 \cdot 10^{-2}$	$2.863 \cdot 10^{-2}$	66	166.9

Figure 5: `ISS` example from Section 5.4. Left: entries of the 3×3 matrix $F(z)$ evaluated at 400 points on the imaginary axis. Right: accuracy performance.

5.5 A nonlinear eigenvalue problem

We consider the `buckling` example in [17], a 3×3 nonlinear eigenvalue problem that arises from a buckling plate model. Since we are interested in the approximation of non-rational functions by means of rational functions, we select only the non-constant part of $F(z)$. Hence, we consider the following 2×2 symmetric matrix-valued function

$$F(z) = \begin{bmatrix} \frac{z(1-2z \cot 2z)}{\tan(z)-z} + 10 & \frac{z(2\lambda - \sin 2z)}{\sin(2z)(\tan(z)-z)} \\ \frac{z(2\lambda - \sin 2z)}{\sin(2z)(\tan(z)-z)} & \frac{z(1-2z \cot 2z)}{\tan(z)-z} + 4 \end{bmatrix}.$$

We choose 500 logarithmically spaced sampling points in the interval $[10^{-2}, 10]$i. Figure 6 shows the RMSE values achieved by the tested algorithms for varying orders $d = 0, 1, \ldots, 30$. As before, we report the numerical RMSE values and the corresponding timings for two selected orders $d = 10$ and $d = 20$ in Table 4.

5.6 A scalar example with noise

In this experiment we investigate the effects of noisy perturbations on the approximation quality. As a test case, we use a scalar function $f(z) = (z-1)/(z^2 + z + 2)$. Only two methods will be considered for this experiment, namely RKFIT (a non-interpolatory method) and AAA (an interpolatory method). We sample the function

Table 3: Selected RMSE values and timings for all tested algorithms — `ISS` example

	RMSE		Runtime (ms)	
	$d = 10$	$d = 20$	$d = 10$	$d = 20$
Set-valued AAA	$3.895 \cdot 10^{-4}$	$5.543 \cdot 10^{-5}$	18.8	36.5
Surrogate AAA	$4.662 \cdot 10^{-4}$	$3.209 \cdot 10^{-5}$	14.6	24.9
VF (5 iter)	$6.811 \cdot 10^{-4}$	$3.523 \cdot 10^{-4}$	49.2	97.1
VF (10 iter)	$6.729 \cdot 10^{-4}$	$3.016 \cdot 10^{-4}$	72.3	135.4
RKFIT (5 iter)	$1.555 \cdot 10^{-4}$	$2.345 \cdot 10^{-5}$	101.8	240
RKFIT (10 iter)	$8.735 \cdot 10^{-5}$	$1.507 \cdot 10^{-5}$	164.9	363.8
Loewner	$9.419 \cdot 10^{-4}$	$2.225 \cdot 10^{-4}$	54.8	56.5
block-AAA	$5.378 \cdot 10^{-5}$	$4.678 \cdot 10^{-6}$	165.4	418.9
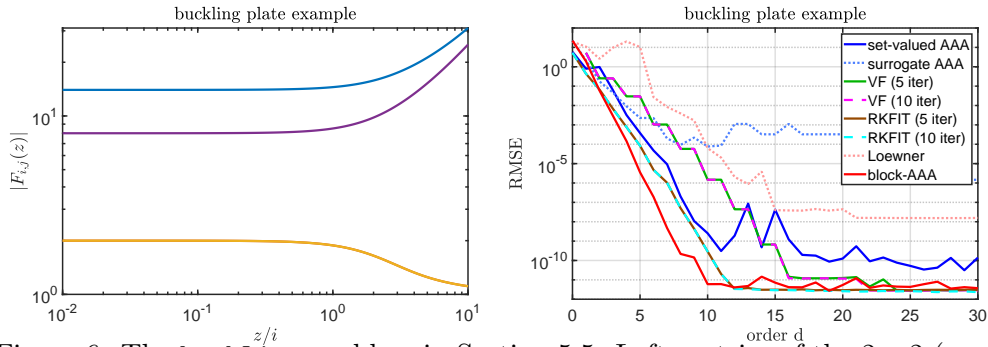
Figure 6: The `buckling` problem in Section 5.5. Left: entries of the 2×2 (symmetric) matrix $F(z)$ evaluated at 500 points on the imaginary axis. Right: accuracy performance.

$f(z)$ at 500 logarithmically spaced points in the interval $[10^{-1}, 10^1]$i, and then add normally distributed noise with a standard deviation of $\tau = 10^{-2}$ to these samples.

We compute rational approximants of degree $d = 5$ using the (scalar) RKFIT and AAA algorithms. For RKFIT, we perform three iterations starting with the default initialization of all poles at infinity. As it can be observed in Figure 7 (left), the red curve corresponding to the RKFIT approximants follows the noisy measurements very well on average. At the same time, the blue curve corresponding to the AAA approximant shows considerable deviations from the measurements. By inspecting the deviation between the two rational approximants and the original function $f(z)$ in Figure 7 (right), we find that RKFIT approximation error for is comparable to the noise level. The RKFIT approximant has effectively estimated the additive noise rather accurately, which is also confirmed visually by the true noise curve overlaid as a dotted line on top of the RKFIT error curve. On the other hand, the approximation error attained by the AAA method is between 1–2 orders of magnitude larger than the added noise level.

Finally, we vary the degree $d = 0, 1, \ldots, 5$ and compute the corresponding RMSE values of the RKFIT and AAA approximants. The results are depicted in Figure 8. Note that the RKFIT method achieves lower RMSE values than the AAA method and exhibits a more "regular" convergence. The RKFIT error stagnates at the noise level of approximately $\tau = 10^{-2}$ when the degree $d = 2$ is reached. We believe that this preferable approximation performance is due to the non-interpolatory nature of RKFIT, and that AAA suffers from the fact that noisy values are being interpolated

Table 4: Selected RMSE values and timings for all tested algorithms — `buckling` problem

	RMSE		Runtime (ms)	
	$d = 10$	$d = 20$	$d = 10$	$d = 20$
Set-valued AAA	$2.543 \cdot 10^{-9}$	$1.258 \cdot 10^{-10}$	18.4	27.3
Surrogate AAA	$7.640 \cdot 10^{-5}$	$3.132 \cdot 10^{-4}$	20.7	31.8
VF (5 iter)	$1.476 \cdot 10^{-4}$	$1.270 \cdot 10^{-11}$	49.2	108.1
VF (10 iter)	$1.476 \cdot 10^{-4}$	$2.216 \cdot 10^{-11}$	63.6	146.4
RKFIT (5 iter)	$2.924 \cdot 10^{-10}$	$2.969 \cdot 10^{-12}$	112.9	269.1
RKFIT (10 iter)	$2.924 \cdot 10^{-10}$	$2.389 \cdot 10^{-12}$	171.7	341.2
Loewner	$6.309 \cdot 10^{-5}$	$4.397 \cdot 10^{-8}$	83.8	86.8
block-AAA	$6.272 \cdot 10^{-12}$	$3.530 \cdot 10^{-12}$	143.8	330.8
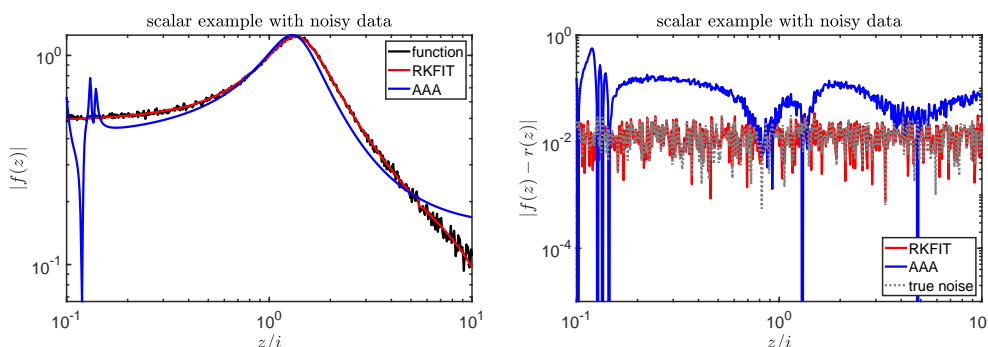
Figure 7: Scalar example with noisy measurements from Section 5.6. Left: the original scalar function $f(z)$ and the computed RKFIT and AAA approximants of order $d = 5$. Right: the deviation between the two approximants and the original function $f(z)$.

(resulting in the observed oscillations of the approximants).

5.7 A second example with noise

For this case, we analyze a noisy version of the `ISS` model considered in Section 5.4. We modify the original LTI system (A, B, C) by choosing $C = B^T$ in order to obtain a symmetric transfer function and be able to apply the Matrix Fitting Toolbox.

The sample values are corrupted by additive normally distributed noise with standard deviation $\tau = 10^{-2}$. In Figure 9 (left) we depict the magnitude of each entry of the perturbed 3×3 matrix transfer function.

The RMSE values for varying orders $d = 0, 1, \ldots, 80$ are plotted in Figure 9 (right). Now, for $d = 10$ and $d = 20$ we record the numerical RMSE values for all methods in Table 5. In accordance with our observations on the previous scalar example (Section 5.6) we find that the non-interpolatory methods like RKFIT and VF exhibit the most steady convergence behaviour, while the interpolation-based methods (AAA, Loewner) produce approximants whose accuracy varies wildly as the order d changes. The only methods that reliably attain an RMSE close to the noise level of $\tau = 10^{-2}$ are RKFIT and VF, with RKFIT (10 iterations) consistently attaining the lowest RMSE values for all considered orders.

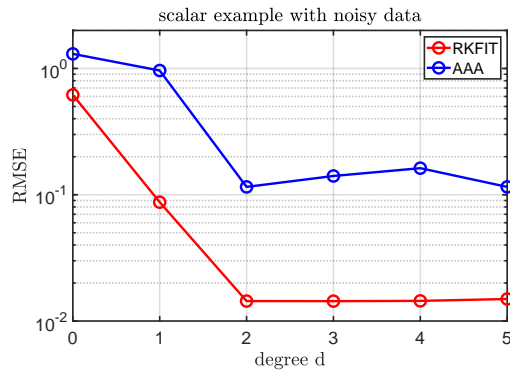Recall that we considered the original (nonsymmetric) `ISS` model without noise

Figure 8: RMSE convergence of the approximants computed via RKFIT (non-interpolatory) and AAA (interpolatory) for increasing degree d. The original function is rational of degree 2, and noise with a standard deviation of $\tau = 10^{-2}$ has been added to perturb it.

in Section 5.4. As the timings of all methods are similar to those reported in Table 2, we do not report them separately for the noisy case considered here.
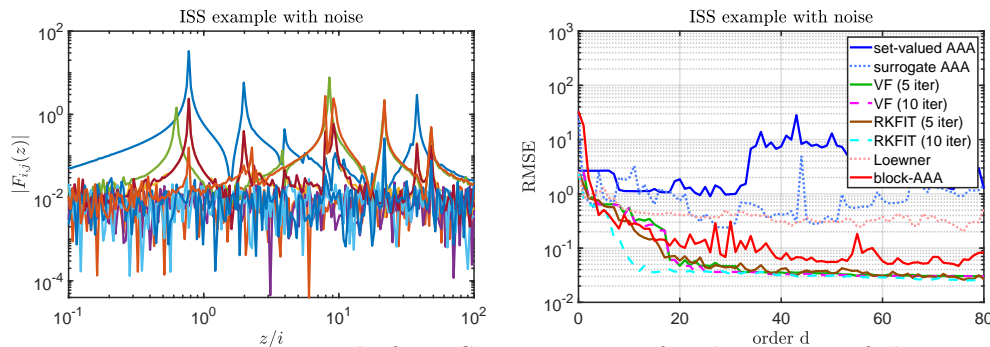

Figure 9: Noisy `ISS` example from Section 5.7. Left: the entries of the perturbed 3×3 matrix $F(z)$. Right: performance comparison.

6 Discussion

In this section we present a more detailed discussion of the numerical results reported in Section 5. The discussion will address each method separately, taking into consideration both the approximation quality and also the amount of time needed to run the method (runtime).

6.1 Set-valued AAA

A main benefit of the set-valued AAA algorithm is its fast runtime. With one exception, the set-valued AAA method was the second fastest method in our tests, surpassed only by the surrogate AAA method.

With respect to the approximation quality, this method produced models with similar accuracy to RKFIT for the example in Section 5.2 (see Figure 3, right). Similar behaviour is observed for the following three examples, producing good and very good results. For the example in Section 5.3 it even produced results comparable in accuracy to the block-AAA method when the order d was very high; see Figure 4 (right). For the examples in Sections 5.6 and 5.7 with added noise, however, the

Table 5: RMSE values for all tested methods - noisy `ISS` example

	$d = 10$	$d = 20$
Set-valued AAA	$1.124 \cdot 10^0$	$1.857 \cdot 10^0$
Surrogate AAA	$1.961 \cdot 10^0$	$7.438 \cdot 10^{-1}$
VF(5 iter)	$5.556 \cdot 10^{-1}$	$4.702 \cdot 10^{-2}$
VF (10 iter)	$5.557 \cdot 10^{-1}$	$4.693 \cdot 10^{-2}$
RKFIT (5 iter)	$3.197 \cdot 10^{-1}$	$5.862 \cdot 10^{-2}$
RKFIT (10 iter)	$6.280 \cdot 10^{-2}$	$3.798 \cdot 10^{-2}$
Loewner	$4.998 \cdot 10^{-1}$	$4.140 \cdot 10^{-1}$
block-AAA	$2.721 \cdot 10^{-1}$	$1.121 \cdot 10^{-1}$

method produced poor results. In particular for the noisy ISS example, the RMSE values produced by the set-valued AAA method are the highest (see Figure 9, right). As discussed above, we believe that the reason for this poor approximation is the interpolation of noisy data values.

6.2 Surrogate AAA

The surrogate AAA method performs best in terms of timings, and it might be an attractive approach when the problem dimension (m, n) is very large and approximation accuracy is not the main concern. The algorithm produces, in general, quite poor approximants compared to the other methods. For the example in Section 5.2, it did not reach an RMSE value below 10^{-9}, while the set-valued AAA, RKFIT, and block-AAA methods were able to converge below 10^{-13}; see Figure 3 (right).

A similar behaviour was observed for both MOR examples in Section 5.3 and 5.4. For the later, the method was the second worst (after VF). For the buckling plate example in Section 5.5 it produced the poorest approximation results (see Figure 6). Finally, for the ISS example with noise in Section 5.7, we observed that the method produced quite poor results, comparable to those produced by the Loewner framework.

6.3 Vector Fitting

For this method we used the implementation provided in [14]. One limitation of this toolbox is that it can only be applied for symmetric matrix-valued functions, i.e., the samples must be symmetric matrices. Consequently, this implementation of VF could not be applied for two examples, namely the ones in Sections 5.3 and 5.4.

For the example in Section 5.2 we observed that the VF method produced poor approximants compared to some of the other methods. On the other hand, for the buckling plate example in Section 5.5 the VF method surpassed three methods and reached an RMSE of about 10^{-12} for the degree $d = 30$ (see Figure 6). Note also that VF together with RKFIT are the only methods that reliably attain an RMSE value ($\tau = 0.03$) close to that of the noise level ($\tau = 10^{-2}$), as could be observed in Figure 9 (right).

Another observation is that it is generally not worth performing 10 VF iterations instead of just 5. We did not observe any significant approximation enhancement and the total runtime of VF will approximately double for 10 VF iterations. Throughout

the experiments performed in Section 5, we found that VF was generally slower than the set-valued and surrogate AAA methods, but faster than the RKFIT and block-AAA methods.

6.4 RKFIT

For the RKFIT method we used the implementation in the RKToolbox. For each experiment, we ran the RKFIT algorithm for a fixed prescribed number of iterations. When stopping after five iterations, RKFIT was the second slowest method for the experiments in Sections 5.2, 5.3, and 5.5 (faster only than the block-AAA method) and the slowest for the experiments in Sections 5.4 and 5.7.

On the other hand, the RKFIT method proved to yield the second lowest RMSE values for the first five experiments, beaten only by block-AAA. This is not surprising given that RKFIT produces approximants R_d where all matrix entries share a common scalar denominator of degree d, say, while a block-AAA approximant of the same order d can have up to dm singularities. Hence, among the scalar denominator methods, RKFIT was the most accurate. Moreover, RKFIT proved to be the most robust when dealing with noise. For both variants (with 5 and 10 iterations), the RMSE values obtained by RKFIT were the lowest among all methods (see Figure 9, right), comparable only to those of VF.

The runtime of the RKFIT method linearly depends on the number of iterations. It was observed in some cases that the RMSE values would not significantly decrease when increasing the number of RKFIT iterations from five to ten. For example, it can be seen in Figure 3 (right) and also in Figure 6 (right), that the two RMSE curves for RKFIT (5 iterations) and RKFIT (10 iterations) are practically indistinguishable. In practice, a dynamic stopping criterion based on the stagnation of the RMSE should therefore be used.

6.5 The Loewner framework

The Loewner framework, as introduced in Section 2.4, is the only direct method out of the six included in this study, i.e., it does not rely on an iteration. This could be an advantage with respect to the speed of execution. In general, this method performs very well in terms of running times for small, and medium to large data sets. The reason for this is that it relies on computing a full singular value decomposition of a matrix with dimension half of the data set.

It was observed that, with some exceptions, the Loewner framework was the third fastest method, surpassed only by the set-valued and surrogate AAA methods. This is not surprising since all the three methods rely on multiplying the matrix-valued samples with left and right tangential vectors, thereby reducing the dimension of the SVD problem to be solved at each iteration.

In terms of approximation accuracy, the Loewner framework produced rather poor results as compared to other methods. For example, in Section 5.2 it was observed that the RMSE values achieved by this method were only lower than those of VF (see Figure 3, right). For the experiments in Sections 5.4 and 5.5, it produced results better only than those of surrogate AAA. Note that, for the CD player example in Section 5.3, the approximation quality was comparable to that of the set-valued AAA method and the RKFIT (with 5 iterations). Finally, the Loewner method failed for the example with added noise (see Figure 9).

6.6 Block-AAA

The block-AAA method introduced in Section 4 is the only method in this comparison which produces rational approximants with *nonscalar* denominators. As such, the approximation quality obtained for a certain order d might not be fully comparable to that of another method. In terms of the execution times, the method proved to be the second slowest for the examples in Sections 5.2, 5.3, and 5.5 (in these cases, only the RKFIT method with 10 iterations was slower). Additionally, for the example presented in Section 5.4, the block-AAA method was indeed the slowest. The reason for this inefficiency is the repeated solution of the SVD problem in Step 5 of the algorithm (see Figure 3), involving a matrix of size $m(j+1) \times \ell n$ at each iteration $j = 0, 1, \ldots, d$, as well as the greedy search in Step 2 which requires the evaluation of $R_{j-1}(z)$ at all remaining sampling points in $\Lambda^{(j)}$ to determine the next interpolation point z_j. We have tried to speed up Step 5 using the updating trick of the SVD decomposition described in [19], but noticed some numerical instabilities for sufficiently large degrees. We have therefore not used any SVD updating strategy for the block-AAA results reported here.

In terms of approximation accuracy, the block-AAA method usually produced the best results for a given order d, with the noisy ISS MOR example being the only exception. A representative illustration is Figure 3 (right), where block-AAA requires an order of $d = 25$ to reach an RMSE value of 10^{-14}, while RKFIT and the set-valued AAA method require $d = 50$ (twice the order). The other methods do not even reach this accuracy. A similar behaviour is observed in Figure 4 (right) for the CD player example and in Figure 5 for the ISS example. For the noisy data experiment in Section 5.7, block-AAA was outperformed by RKFIT and, again, we believe this is due to RKFIT's non-interpolatory character.

7 Conclusion

We proposed an extension of the AAA algorithm to the case of matrix-valued interpolation. The block-AAA method uses a new generalized barycentric representation of the interpolant with matrix-valued weights. Our numerical experiments indicate that this modification allows for the computation of accurate approximants of low order. Having a low order might be particularly interesting in cases where the approximant needs to be linearized for further processing, e.g., in the context of nonlinear eigenvalue computations. In the appendix we show how the generalized barycentric interpolants can be linearized into a corresponding eigenvalue problem of size proportional to the order. The use of block-AAA for this application will be explored in future work. In terms of algorithmic complexity and runtime, the block-AAA method is currently inefficient and can be practically applied only for problems of small dimensions. Further work is required to improve its performance. One approach to deal with this shortcoming could be to replace the full singular value decomposition performed at each step of block-AAA with a CUR decomposition (selecting only a small number of relevant columns and rows in the block Loewner matrix), and to somehow restrict the search for the next interpolation point to a smaller set.

Our comparisons indicated that there was no method that always performed best with respect to accuracy and runtime. Interpolation-based methods are often cheaper to run, but they may suffer in the presence of noise for which approximation-based methods perform better. It seems fair to say that, despite the exciting recent

developments in the area, there is still a lot of work to be done to design robust and fast methods for the rational approximation of matrix-valued functions.

Acknowledgements

References

[1] *SLICOT Benchmark Examples for Model Reduction.* `http://slicot.org/20-site/126-benchmark-examples-for-model-reduction`, 2002.

[2] A. C. Antoulas, *Approximation of Large-Scale Dynamical Systems*, vol. 6, SIAM, 2005.

[3] A. C. Antoulas, C. A. Beattie, and S. Gugercin, *Interpolatory model reduction of large-scale dynamical systems*, in Efficient Modeling and Control of Large-Scale Systems, Springer, Boston, MA, 2010, pp. 3–58.

[4] A. C. Antoulas, C. A. Beattie, and S. Gugercin, *Interpolatory Methods for Model Reduction*, SIAM, 2020.

[5] A. C. Antoulas, I. V. Gosea, and A. C. Ionita, *Model reduction of bilinear systems in the Loewner framework*, SIAM J. Sci. Comput., 38(5) (2016), pp. B889–B916.

[6] M. Berljafa, S. Elsworth, and S. Güttel, *A Rational Krylov Toolbox for MATLAB*, MIMS EPrint 2014.56, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, 2014. `http://rktoolbox.org/`.

[7] M. Berljafa and S. Güttel, *Generalized rational Krylov decompositions with an application to rational approximation*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 894–916.

[8] ——, *The RKFIT algorithm for nonlinear rational approximation*, SIAM J. Sci. Comput., 39 (2017), pp. A2049–A2071.

[9] J.-P. Berrut and L. N. Trefethen, *Barycentric Lagrange interpolation*, SIAM Rev., 46 (2004), pp. 501–517.

[10] Y. Chahlaoui and P. Van Dooren, *Benchmark examples for model reduction of linear time-invariant dynamical systems*, in Dimension Reduction of Large-Scale Systems, Lecture Notes in Computational Science and Engineering, Springer, Berlin, Heidelberg, 2005.

[11] T. A. Driscoll, N. Hale, and L. N. Trefethen, *Chebfun Guide.* `https://www.chebfun.org/docs/guide/`, 2014.

[12] S. ELSWORTH AND S. GÜTTEL, *Conversions between barycentric, RKFUN, and Newton representations of rational interpolants*, Linear Algebra Appl., 576 (2019), pp. 246–257.

[13] B. GUSTAVSEN, *Improving the pole relocating properties of vector fitting*, IEEE Trans. Power Delivery, 21 (2006), pp. 1587–1592.

[14] ———, *Matrix Fitting Toolbox.* `https://www.sintef.no/projectweb/vectorfitting/downloads/matrix-fitting-toolbox/`, 2009.

[15] B. GUSTAVSEN AND A. SEMLYEN, *Rational approximation of frequency domain responses by vector fitting*, IEEE Trans. Power Delivery, 14 (1999), pp. 1052–1061.

[16] S. GÜTTEL AND F. TISSEUR, *The nonlinear eigenvalue problem*, Acta Numer., 26 (2017), pp. 1–94.

[17] N. J. HIGHAM, G. M. NEGRI PORZIO, AND F. TISSEUR, *An updated set of nonlinear eigenvalue problems*, Tech. Rep. MIMS EPrint: 2019.5, Manchester, United Kingdom, 2019. `http://eprints.maths.manchester.ac.uk/`.

[18] A. HOCHMAN, *FastAAA: A fast rational-function fitter*, in Proceedings of the 26th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS), IEEE, 2017.

[19] P. LIETAERT, J. PÉREZ, B. VANDEREYCKEN, AND K. MEERBERGEN, *Automatic rational approximation and linearization of nonlinear eigenvalue problems*, arXiv preprint arXiv:1801.08622, (2018).

[20] A. J. MAYO AND A. C. ANTOULAS, *A framework for the solution of the generalized realization problem*, Linear Algebra Appl., 425 (2007), pp. 634–662.

[21] Y. NAKATSUKASA, O. SÈTE, AND L. N. TREFETHEN, *The AAA algorithm for rational approximation*, SIAM J. Sci. Comput., 40 (2018), pp. A1494–A1522.

[22] R. VAN BEEUMEN, W. MICHIELS, AND K. MEERBERGEN, *Linearization of Lagrange and Hermite interpolating matrix polynomials*, IMA J. Numer. Anal., 35 (2015), pp. 909–930.

Appendix: Linearization of generalized barycentric forms

While we have been primarily concerned with evaluating the various algorithms in terms of their RMSE approximation performance, in the context of nonlinear eigenvalue problems one also requires the computed approximants to be suitable for pole and "root" computations. More precisely, in nonlinear eigenvalue problems one is interested in the points $\xi \in \mathbb{C}$ where one or more entries of the rational function $R_d(z)$ have a pole, and the nonlinear eigenvalues $\lambda \in \mathbb{C}$ where $R_d(\lambda)$ is a singular matrix. Naturally the latter problem requires $R_d(z)$ to be a square matrix.

Let us discuss the suitability of the matrix-valued barycentric forms in Section 3 for nonlinear eigenvalue computations based on linearization; see, e.g., [16, Section 6]. First note that (bary-A) and (bary-B) are special cases of (bary-C), respectively: we obtain (bary-A) from (bary-C) by setting $C_k = w_k F(z_k)$ and $D_k = w_k I$,

while we obtain (bary-B) from (bary-C) by setting $C_k = W_k F(z_k)$ and $D_k = W_k$. Hence it is sufficient to focus on the most general representation (bary-C).

If $\lambda \in \mathbb{C}$ is a nonlinear eigenvalue of $R_d(z)$, then clearly $R_d(z)^{-1}$ has a pole at $z = \lambda$, and vice versa. It is therefore sufficient to be able to find the nonlinear eigenvalues of R_d defined in (bary-C) as those points $\lambda \in \mathbb{C}$ for which the "numerator"

$$N(z) = \frac{\displaystyle\sum_{k=0}^{d} \frac{w_k[C_k/w_k]}{z - z_k}}{\displaystyle\sum_{k=0}^{\ell} \frac{w_k}{z - z_k}}$$

is a singular matrix. It is well known that for given support points $z_0, z_1, \ldots, z_d$, we can choose nonzero weights $w_0, w_1, \ldots, w_d$ such that $N(z)$ is an interpolating matrix polynomial:

$$w_k = \frac{1}{\prod_{j \neq k}(z_j - z_k)};$$

see, e.g., [9, eq. (3.2)] and [22, eq. (3.4)]. With this choice of the weights, we can immediately apply [22, Thm. 4.8], which states that the matrix pencil $L(z) = L_0 - zL_1$, where

$$L_0 = \begin{bmatrix} z_1 C_0 & z_2 C_1 & \cdots & z_{d-1}C_{d-2} & z_\ell C_{d-1} + z_{d-1}\theta_d^{-1}C_d \\ z_0 I & -z_2\theta_1 I & & & \\ & \ddots & \ddots & & \\ & & z_{d-3}I & -z_{d-1}\theta_{d-2}I & \\ & & & z_{d-2}I & -z_d\theta_{d-1}I \end{bmatrix}$$

and

$$L_1 = \begin{bmatrix} C_0 & C_1 & \cdots & C_{d-2} & C_{d-1} + \theta_d^{-1}C_d \\ I & -\theta_1 I & & & \\ & \ddots & \ddots & & \\ & & I & -\theta_{d-2}I & \\ & & & I & -\theta_{d-1}I \end{bmatrix}$$

with $\theta_j = w_{j-1}/w_j$ for $j = 1, \ldots, d$ is a strong linearization of $N(z)$. In other words, the nonlinear eigenvalues of $N(z)$ can be computed as the generalized eigenvalues of the matrix pair (L_0, L_1). As long as R_d does not also have a pole at a nonlinear eigenvalue λ of $N(z)$, λ will be a nonlinear eigenvalue of $R_d(z)$.

Finally, it is perhaps interesting to note that we have freedom to choose the nonzero weights w_j and this choice will likely influence the numerical stability of the linearization (L_0, L_1). This might be explored in some future work.