juliacon

RationalFunctionApproximation.jl: Rational Approximation On Discrete and Continuous Domains

Tobin A. Driscoll[1]

[1]University of Delaware

ABSTRACT

Unlike polynomials, rational functions can represent functions having poles or branch cuts with root-exponential convergence and no Runge phenomenon. Recent developments of the AAA and greedy Thiele algorithms have sparked renewed interest in computational rational approximation. The RationalFunctionApproximation package supplies the fastest known implementations of these methods and the only arbitrary-precision ones. Combined with the ComplexRegions package, it can produce compact and accurate representations of a huge variety of functions over intervals, circles, or other domains in the complex plane.

Keywords

Julia, Approximation, Complex variables, Rational functions

1. Introduction

Approximation of functions is a foundational technology for scientific computing. As of this writing, the Julia ecosystem has excellent options for approximation in packages such as Interpolations [14] for piecewise polynomials, ApproxFun [21] for global polynomials, KernelInterpolation [15] for radial basis functions, and many others.

A long-studied class of approximating functions are the rational functions, which are ratios of polynomials. Unlike polynomials, they can represent functions having branch cuts with root-exponential convergence [20, 12] and have no Runge requirement for crowding interpolation nodes near boundaries [22]. Rational functions have been used in applications such as nonlinear eigenvalue problems, reduced-order modeling, and the solution of PDEs [1, 4, 10, 11, 16, 23, 27].

An important connection between rational and polynomial interpolants is their common representation in the barycentric formula [2, 26], which allows for fast, stable evaluation. A major recent development in computational rational approximation is the AAA algorithm [18]. This approach chooses the barycentric weights to minimize least-squares error over a test set and uses an iterative greedy selection of new interpolation nodes. The method has inspired many extensions and variations [8, 13, 19, 29]. Of particular interest to this work is the extension of AAA to adaptive node selection in continuous domains in order to avert underresolution near singularities [7].

Rational functions have an alternative classical representation in terms of continued fractions, a fact used by Thiele at least 100 years ago to derive an interpolation scheme via inverse differences [17]. Because of the connection to divided differences, Thiele's method has long been known to be vulnerable to numerical instability, depending strongly on the selection and ordering of interpolation nodes. But Salazar [24, 25] has recently shown that a greedy iterative approach to selecting nodes can yield stable Thiele approximations. Adding a single node to a Thiele interpolant on n nodes requires just $O(n)$ work, compared to $O(n^3)$ for AAA; the possibility of a faster method is intriguing, although the stability situation is not yet fully understood.

The RationalFunctionApproximation package implements three rational approximation methods: the AAA algorithm, the greedy Thiele algorithm, and linear least-squares approximation using prescribed poles. It builds on the ComplexRegions package to provide a convenient interface for approximating functions on complex domains, such as the unit disk or the exterior of a polygon. The package is compatible with extended-precision arithmetic as provided, e.g., by BigFloat or MultiFloats. The package is intended for use both as a library for applications or for research into rational approximation methods.

This paper is written about and using version 0.2.4.

2. Mathematical background

2.1 AAA

The barycentric representation of a rational interpolant is

$$r(z) = \frac{\displaystyle\sum_{j=1}^{n} \frac{y_j\, w_j}{z - z_j}}{\displaystyle\sum_{j=1}^{n} \frac{w_j}{z - z_j}}, \tag{1}$$

where $z_1, \ldots, z_n$ are distinct interpolation nodes, $y_1, \ldots, y_n$ are the values of r at those nodes, and $w_1, \ldots, w_n$ are the barycentric weights. Generically, the function r is of degree n in both the numerator and denominator. For a particular choice of the weights, however, r is actually a polynomial of degree at most $n - 1$ [2]. We can instead use the weights to improve the global quality of the approximation of r to a function f with values $y_j = f(z_j)$.

We define the linearized residual

$$\delta(z) = \sum_{j=1}^{n} \frac{y_j\, w_j}{z - z_j} - f(z) \sum_{j=1}^{n} \frac{w_j}{z - z_j}. \tag{2}$$

Given test points $t_1, \ldots, t_m$ in the domain of f, we can express the evaluation of d at those points as

$$\begin{bmatrix} \delta(t_1) \\ \delta(t_2) \\ \vdots \\ \delta(t_m) \end{bmatrix} = \boldsymbol{L} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, \tag{3}$$

where $\boldsymbol{L}$ is the $m \times n$ *Loewner matrix* with entries

$$L_{ij} = -\frac{f(t_i) - y_j}{t_i - z_j}. \tag{4}$$

Since the barycentric formula (1) is unaffected by a scaling of the weight vector, we can recognize the least-squares minimization of the vector $\boldsymbol{\delta}$ as equivalent to finding the least-significant right singular vector of $\boldsymbol{L}$. Once the weights are determined, the true residual $f(z) - r(z)$ be evaluated at the test points. The AAA algorithm selects the test point with the largest magnitude of this residual as the next interpolation node.

In the original discrete AAA algorithm [18], the test points are selected prior to the iteration. In the continuum version [7], a small initial set of test points is selected to discretize the boundary of the domain, and with each transfer of a test point to the node set, a few additional test points are added along the boundary to either side of the new node. This adaptivity becomes important when f has a singularity very close to or on the boundary of the domain.

2.2 Greedy Thiele

The Thiele representation of a rational interpolant is [24, 17]

$$r(z) = d_1 + \cfrac{z - z_1}{d_2 + \cfrac{z - z_2}{d_3 + \cdots}}, \tag{5}$$

where, given nodes $z_1, \ldots, z_n$ and inductively $d_1, \ldots, d_{n-1}$, the new weight d_n is defined as the result u_n of the iteration

$$\begin{aligned} u_1 &= f(z_n), \\ u_{k+1} &= \frac{z_n - z_k}{u_k - d_k} \quad \text{for } k = 1, \ldots, n-1. \end{aligned} \tag{6}$$

Generically, r interpolates the values $(z_j, f(z_j))$ for $j = 1, \ldots, n$ and is a rational function of type (m, m) if $n = 2m - 1$ or type $(m + 1, m)$ if $n = 2m$.

The Thiele representation has had modest computational use due to the inverse difference appearing in (6), which is numerically unstable for many node distributions/orderings. However, Salazar [24, 25] observed that if one follows the structure of AAA by evaluating a candidate r at test points and greedily selecting the worst test point as a new node, the method is often stable. A key attraction of Salazar's method is that it requires only $O(n)$ work to add a new node, comparing favorably to $O(mn^2)$ for AAA with n nodes and m test points.

2.3 Prescribed poles

When prior information about the singularity structure of a function is available, it can be useful to use a rational approximation with prescribed poles, yielding a linear problem. Given poles $\zeta_1, \ldots, \zeta_\nu$, consider the partial fractions representation

$$r(z) = p(z) + \sum_{j=1}^{\nu} \frac{c_j}{z - \zeta_j}, \tag{7}$$

where q is a polynomial. Evaluation of r at the test points $t_1, \ldots, t_m$ yields an overdetermined linear system for the coefficients of p and the residues $c_1, \ldots, c_\nu$.

A potential difficulty with this approach is finding a well-conditioned basis for p. On a circle, the monomial basis is ideal, and on an interval, the Chebyshev basis is a natural choice, but on a more general domain, the choice is not as apparent. To resolve this problem, we use the Vandermonde–Arnoldi approach [3], which is a stable method to orthogonalize the monomial basis $\{1, z, z^2, \ldots, z^N\}$ as evaluated at the test points $\boldsymbol{t}$:

$\boldsymbol{q}_0 \leftarrow [1, 1, \ldots, 1]$
for $j = 0, \ldots, N-1$ **do**
  $\boldsymbol{v} = \boldsymbol{t} \odot \boldsymbol{q}_j$ ▷ Hadamard product
  for $k = 0, \ldots, j-1$ **do**
    $H_{k,j} = \boldsymbol{q}_j^* \boldsymbol{v}/m$ ▷ Inner product
    $\boldsymbol{v} \leftarrow \boldsymbol{v} - H_{k,j}\boldsymbol{q}_k$ ▷ Orthogonalize
  end for
  $H_{j+1,j} = \|\boldsymbol{v}\|/\sqrt{m}$ ▷ Normalize
  $\boldsymbol{q}_{j+1} \leftarrow \boldsymbol{v}/H_{j+1,j}$ ▷ Next basis vector
end for

The vectors $\boldsymbol{q}_0, \ldots, \boldsymbol{q}_N$ form an orthonormal basis for the evaluated monomials, while the upper Hessenberg $\boldsymbol{H}$ holds values needed to reconstruct the orthogonalization process at new points. Given coefficients $a_0, \ldots, a_N$ of polynomial q in the orthonormal basis, we evaluate $y = p(z)$ via

$y \leftarrow a_0$
$q_0 \leftarrow 1$
for $j = 0, \ldots, N-1$ **do**
  $v \leftarrow zq_j$
  for $k = 0, \ldots, j-1$ **do**
    $v \leftarrow v - H_{k,j}q_k$
  end for
  $q_{j+1} \leftarrow v/H_{j+1,j}$
  $y \leftarrow y + a_{j+1}q_{j+1}$
end for

3. Key data types

3.1 DiscretizedPath

The DiscretizedPath type manages the details of choosing nodes and test points along a Curve or Path from the ComplexRegions package. Both kinds of points are stored in an array that is allocated at the time of construction, with nodes appearing in the first column. The `add_node!` method designates one of the test points as a new node, and additional test points are added to either side of the new node. The `collect` method extends `Base.collect` to return a vector of all nodes, all test points, or all points, depending on the `which` keyword.

3.2 ArnoldiBasis and ArnoldiPolynomial

An ArnoldiBasis applies the Arnoldi iteration to a given vector of points, as described in subsection 2.3. The ArnoldiPolynomial type collects an ArnoldiBasis and a vector of coefficients, providing an evaluation method. As a convenience, the backslash operator is overloaded to solve a linear least-squares problem for a function, as in the following:

Code 1: Least-squares for an ArnoldiBasis.

```
z = point(Circle(0, 1), range(0, 1, 800))
B = ArnoldiBasis(z, 10)
p = B \ cos    # creates an ArnoldiPolynomial
maximum(abs, p.(z) - cos.(z))
```

The result from the above is about 2.78×10^{-7}.

Table 1. : AbstractRationalFunction interface

Method	Description	Implemented?
`show(io, r)`	Pretty-print details	yes
`r(z)`	Evaluate at a point	no
`eltype(r)`	Floating-point type for values	yes
`isempty(r)`	Is the interpolant empty?	yes
`degrees(r)`	Degree of numerator, denominator	no
`degree(r)`	Degree of the denominator	no
`poles(r)`	Poles of the rational function	no
`residues(r)`	Poles and residues	no
`roots(r)`	Roots of the rational function	no
`Res(r, z)`	Residue by trapezoidal rule	yes

Table 2. : AbstractRationalInterpolant interface

Method	Description	Implemented?
`show(io, r)`	Pretty-print details	yes
`nodes(r)`	Interpolation nodes	no
`values(r)`	Values at the nodes	no

3.3 AbstractRationalFunction and AbstractRationalInterpolant

The AbstractRationalFunction type is an abstract type providing a common interface for rational functions. The type is parameterized by the floating-point type of the function's values. It provides stubs or universal implementations for the methods shown in Table 1.
The AbstractRationalInterpolant type is an abstract subtype of AbstractRationalFunction that provides an interface for rational interpolants. It is parameterized by the floating-point type of the interpolation nodes and the floating-point type of the values at the nodes. The type provides stubs or universal implementations for the methods shown in Table 2.
Each following concrete realization of one of these abstract types also implements two signatures of the `approximate` function that is described in subsection 3.4, one for approximation of a function on a continuum domain, and one for fully discrete approximation.

3.3.1 Barycentric. Barycentric is a concrete subtype of AbstractRationalInterpolant that implements the barycentric representation of a rational interpolant, as described in subsection 2.1. It stores the nodes, values, and weights, and provides implementations for the methods shown in Table 2. When precision beyond standard `Float64` is detected, the pole computation relies on `GenericSchur` from GenericLinearAlgebra to perform the necessary generalized eigenvalue computation. T

3.3.2 Thiele. Thiele is a concrete subtype of AbstractRationalInterpolant that implements the Thiele form of a rational interpolant, as described in subsection 2.2. It stores the nodes, values, and weights, and provides implementations for the methods shown in Table 2. Currently, however, the implementation of `residues` is based only on trapezoidal integration rather than any special property of the Thiele representation.

3.3.3 PartialFractions. The PartialFractions type is a concrete subtype of AbstractRationalFunction that collects an ArnoldiPolynomial and vectors of poles and residues, representing a rational function with prescribed poles, as given in Equation 7. It implements all the methods shown in Table 1.

3.4 Approximation

The Approximation type is the primary type that a user interacts with directly. It collects a function, information about its domain, a rational approximant, and an iterative history. The standard way to create an Approximation is via the `approximate` function, which has several different calling signatures.
Interpolation. For greedy iterations with the barycentric (i.e., AAA) or Thiele representations, `approximate` takes a `method` keyword that is either `Barycentric` (the default) or `Thiele`. Nodes are added until the error at all test points is less than a tolerance that can be chosen with the keyword `tol`, or until a stagnation criterion is met for a number of iterations that can be set with the keyword `stagnation`. The algorithm only returns a result whose poles are all allowed, using a Boolean-valued function that can be set with the `allowed` keyword. The following signatures are available:

`approximate(f,d)` for Curve or Path `d` creates a rational interpolant of the function `f` on the domain `d` using a continuum approach. No poles in `d` are allowed.

`approximate(f,d)` for SimplyConnectedRegion `d` creates a rational interpolant of the function `f` on the domain `d` using a continuum approach. The nodes are chosen on the boundary of `d`, and no poles are allowed in `d`, which may be an interior or exterior region.

`approximate(f,z)` for a vector of numbers `z` creates a rational interpolant of the function `f` in a discrete approach, using `z` as the fixed set of test points and potential nodes. By default, all poles are allowed.

`approximate(y,z)` for vectors `y` and `z` uses the entries of `y` as the values to be taken at the entries of `z`. This method returns a subtype of AbstractRationalInterpolant, as there is presumed to be no information about an originating function.

Because both the AAA and Salazar iterations rely on pairwise differences between nodes, test points, and function values in various combinations, care was taken to reuse such information between iterations. The representation-specific references in the iteration are the representation constructor and the methods `add_nodes!` and `update_test_values!`, which the representations must implement. A first call to `update_test_values!` allocates working space. Subsequent calls update the working arrays at any new test points, updates the weights of the rational interpolant (in the barycentric case only), and evaluates the interpolant at all test points. Once a new node is chosen, a call to `add_nodes!` updates the node and value properties of the interpolant (which, in the Thiele case, also updates the weights), allocates space for new weights, and updates differencing information between the new node(s) and the existing test points.
Least squares. For a least-squares approximation with prescribed poles using the partial fraction representation, a `degree` keyword to `approximate` can be used to specify the degree of the polynomial part. The following signatures are available:

`approximate(f,d,zeta)` for Curve or Path `d` creates an approximation of the function `f` on the domain `d` with the prescribed poles in the vector `zeta`. To discretize `d`, equally spaced nodes are placed (whose number is selectable by the `init` keyword), and test points are promoted to nodes iteratively until the distance between adjacent nodes is no greater than one-half the distance to the nearest pole.

`approximate(f,z,zeta)` for a vector of numbers `z` uses the entries of `z` as the discretization of the domain.

`approximate(y,z,zeta)` for vectors `y` and `z` uses the entries of `y` as the values to take at the entries of `z`. This method returns a subtype of AbstractRationalFunction, as there is presumed to be no information about an originating function.

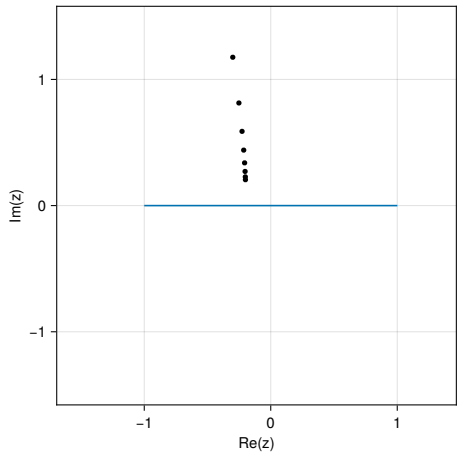

Fig. 1: Pole plot of the AAA rational interpolant of $f(z) = \log(1+i+5iz)$ on the unit interval. The poles cluster near the branch point at $(i-1)/5$.

4. Examples

The package defines `unit_interval` as a Segment from -1 to 1. We can approximate the function $f(z) = \log(1 + i + 5iz)$ on this interval as follows:

Code 2: Continuum AAA on the unit interval.

```
using RationalFunctionApproximation, CairoMakie
f = z -> log(1 + 1im + 5im*z)
r = approximate(f, unit_interval)
poleplot(r)
```

The result is a Barycentric interpolant of type $(12, 12)$ and the plot in Figure 1. The poles of this interpolant cluster near the branch point of f at $(i-1)/5$. We can check the size of the error via `check(r)`, which returns vectors of test points and error values and prints a message with the maximum error; here, that maximum is about 1.6×10^{-13}.
The usefulness of the continuum methods is apparent when singularities of f creep close to the domain. If, for instance, we approximate $\sqrt{x + 10^{-6}i}$ on equally spaced points in $[1, 1]$, we will miss important details at the origin:

Code 3: Discrete AAA for $\sqrt{x + 10^{-6}i}$.

```
f = z -> sqrt(z + 1e-6im)
x = range(-1, 1, 1001)
r = approximate(f, x)
lines(-0.001..0.001, x->real(f(x) - r(x)))
```

Although the discrete iteration finishes successfully, with a reported maximum error of 4.5×10^{-14} at the entries of `x`, we see in Figure 2 that the error near the origin exceeds 10^{-3} at other locations. With advance knowledge of the singularity structure, a discretization could be chosen manually that would properly resolve the singularity. But the continuum approach offers this resolution automatically:

Code 4: Continuum AAA for $\sqrt{x + 10^{-6}i}$.

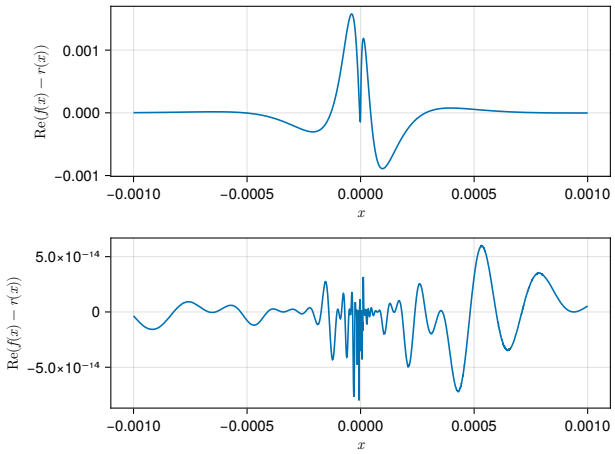

Fig. 2: Top: In the discrete AAA iteration, accuracy close to a singularity will suffer if the nodes have insufficient resolution. Bottom: Continuum AAA adaptively chooses nodes and test points in order to resolve the singularity.

```
r = approximate(f, unit_interval)
lines(-0.001..0.001, x->real(f(x) - r(x)))
```

Now, as confirmed in Figure 2, the error is less than 10^{-13} everywhere on the interval.
The convergence of AAA for Newman's example of $f(x) = |x|$ is interesting:

Code 5: Continuum AAA for $|x|$.

```
r = approximate(abs, unit_interval)
convergenceplot(r)
```

As seen in the upper plot of Figure 3, the iteration was halted when convergence stalled a bit short of machine roundoff. In order to continue past the plateau, we can use extended precision supplied by the DoubleFloats package. Since the eigenvalue computation needed to check for disallowed poles is much slower in extended precision, we distable the checks with the keyword `allowed=true`.

Code 6: Continuum AAA in extended precision.

```
using DoubleFloats, ComplexRegions
uiquad = Segment{Double64}(-1, 1)
r = approximate(abs, uiquad;
        allowed=true, max_iter=200)
```

This computation took 30.5 seconds to compute on an Apple M4 Pro processor, compared to 0.23 seconds in Code 5. The lower plot of Figure 3 shows that the convergence continues smoothly at a root-exponential rate. Figure 4 illustrates that the positive node locations cluster with a tapered exponential pattern at the singularity, much as the poles do [28].
Any of the above examples can be computed using the greedy Thiele iteration in place of AAA by adding `method=Thiele` to the `approximate` call. For example, we can use it in extended precision for $|x|$:In order to exploit its speed for Code 5, we use

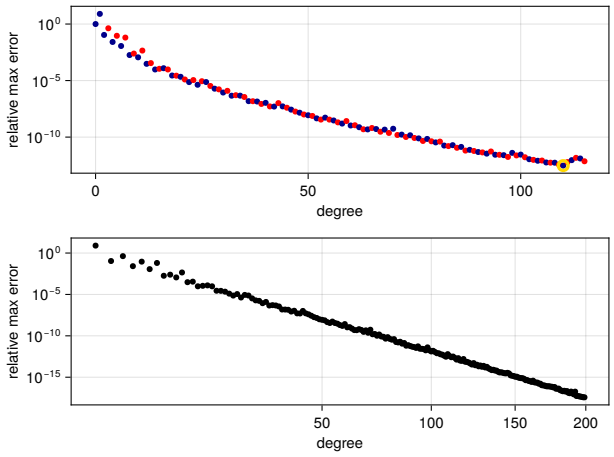

Fig. 3: Convergence of continuum AAA to the function $|x|$ over $[-1, 1]$. Top: Red dots indicate an interpolant having a pole in the interval, while blue dots indicate no disallowed poles. The gold halo shows the interpolant returned as the result, chosen when the iteration stagnated. Bottom: Using extended precision, the iteration continues successfully (without pole checking); the square-root scale on the degree axis illustrates root-exponential convergence.

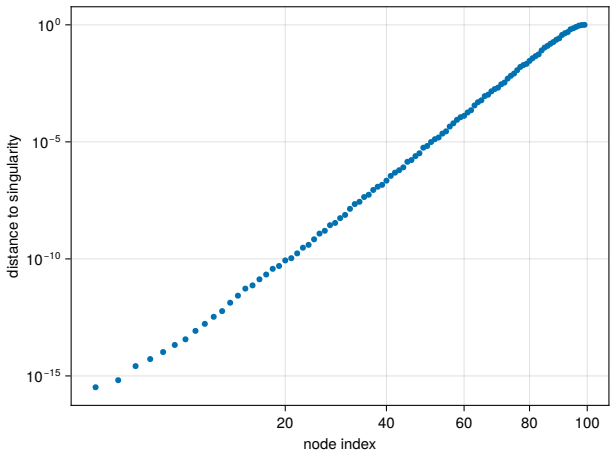

Fig. 4: Positive nodes selected by continuum AAA for the function $|x|$ over $[-1, 1]$. They cluster near the singualrity at $x = 0$.

`allowed=true` to disable pole checking and add `stagnation=20` to make it persist through plateaus. We also double the number of iterations, since the degree is roughly half the number of nodes:

Code 7: Continuum Thiele for $|x|$.

```
approximate(abs, uiquad; allowed=true,
        method=Thiele, max_iter=400, stagnation=20)
```

The above code takes just 1.69 seconds to reach a similar degree and accuracy as the AAA result. This speedup is dramatic because AAA relies on generic implementations of the SVD for DoubleFloats, while Thiele requires no linear algebra at all. However, as seen in Figure 5, the convergence of the Thiele iteration is far less

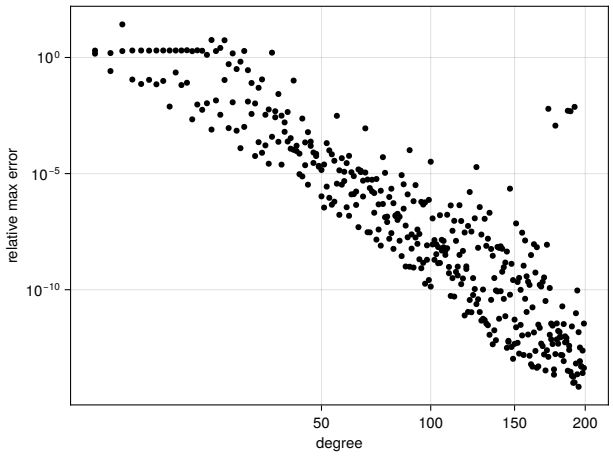

Fig. 5: Convergence of continuum Thiele to the function $|x|$ over $[-1, 1]$. The smallest errors are similar to those in Figure 3 for AAA, but the convergence is far less smooth.

smooth here, which is why `stagnation=20` was used to make it continue past apparent plateaus.

An advantage of the AAA approach is that it can be modified to construct near-best approximations in the max-norm sense [19], which classically is of greater interest than the 2-norm. For instance, the error for $f(x) = |x - 0.5 + 0.05i|$ is rather nonuniform:

Code 8: Continuum AAA for $|x - 0.5 + 0.05i|$.

```
f = z -> abs(z - 0.5 + 0.05im)
r = approximate(f, unit_interval, max_iter=20)
lines(check(r)...)
```

As seen on the top of Figure 6, the error is larger on the right end of the interval than on the left by orders of magnitude. The `minimax` function uses iteratively weighted norms to place more emphasis where the error is largest, resulting in near-uniform error, as shown in the bottom of the figure.

Code 9: Minimax AAA for $|x - 0.5 + 0.05i|$.

```
r_inf = minimax(r, 20);
lines(check(r_inf)...)
```

Other domains of analyticity may easily be specified. One predefined domain is the unit circle:

Code 10: Approximation on the unit circle.

```
using SpecialFunctions: zeta
f = z -> 1/ zeta(11z)
r = approximate(f, unit_circle)
poleplot(r)
```

The poles of the resulting interpolant of degree 21 are shown in Figure 7. The maximum error in the approximation on the circle is about 1.2×10^{-12}. The ComplexRegions package defines a Shapes module with other predefined curves, as well as functions `interior` and `exterior` that can be used to create SimplyConnectedRegion objects.

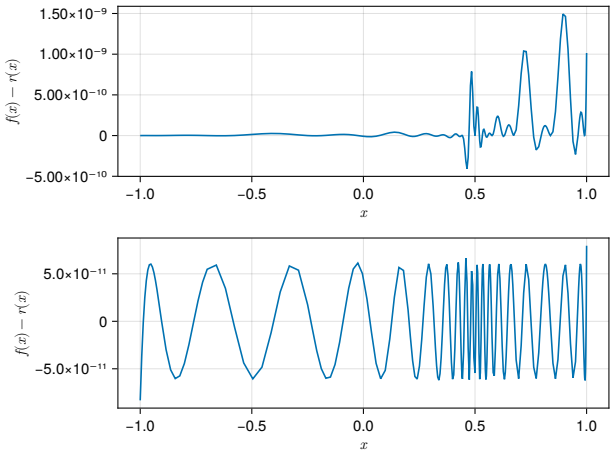

Fig. 6: Top: Nonuniform error is produced by the AAA iteration, which uses a least-squares criterion. Bottom: After Lawson-style iterations, the error can be made nearly uniform across the interval.

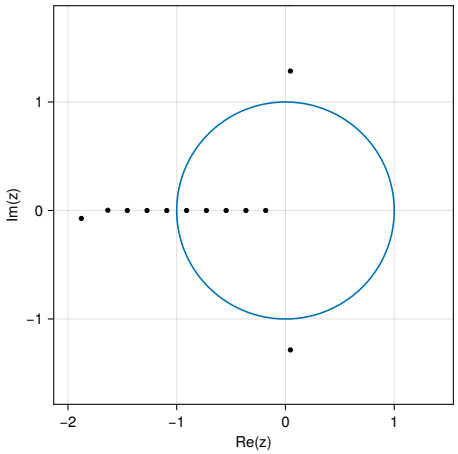

Fig. 7: Poles of a rational interpolant of $f(z) = 1/\zeta(11z)$ on the unit circle.

Code 11: Approximation on an exterior domain.

```
f = z -> coth(1/z^3)
r = approximate(f, exterior(Shapes.squircle))
poleplot(r)
```

The resulting plot is given in Figure 8.
If prior knowledge of the singularity structure is available, we can use the partial fraction representation and solve a linear problem without iteration. For example, suppose we return to the log function in Code 2 and extract the poles from a rational interpolant. We can use the same poles to approximate a different function with the same branch point:

Code 12: Partial fraction approximation of $\log(1 + i + 5iz)$.

```
f = z -> log(1 + 1im + 5im*z)
r = approximate(f, unit_interval)
ζ = poles(r)
g = z -> sqrt(1 + 1im + 5im*z)
s = approximate(g, unit_interval, ζ; degree=20)
```

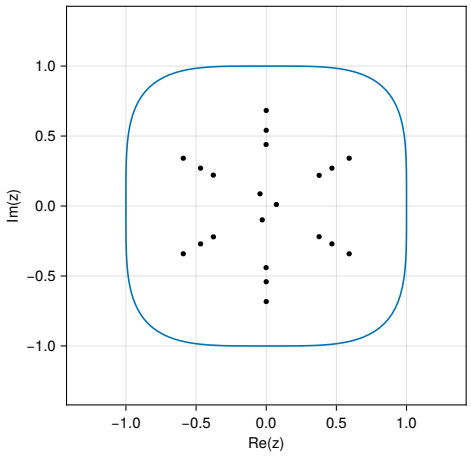

Fig. 8: Poles of a rational approximation of $\coth(z^{-3})$ in the exterior of a squircle.

The resulting approximation reports an error of size 5.3×10^{-10} at its test points.

5. Future work

It should be possible to implement operations on the rational function types to enable basic arithmetic, composition, differentiation, and integration, in the style of Chebfun [6] and ApproxFun [21]. A systematic comparison of the AAA and Thiele methods is also needed, as well as a more complete understanding of the stability of the Thiele method. Least-squares approximations with prescribed poles have been fruitful in the solution of Laplace and related PDEs [5, 10, 9], which requires imposing conditions other than direct approximation of a known function.

6. Acknowledgments

I thank Nick Trefethen and Oliver Salazar Celis for their valuable feedback on a draft of this paper.

7. References

[1] A. C. Antoulas, C. A. Beattie, and S. Güğercin. *Interpolatory Methods for Model Reduction*. Society for Industrial and Applied Mathematics, Philadelphia, PA, January 2020. doi:10.1137/1.9781611976083.

[2] J.-P. Berrut and L. N. Trefethen. Barycentric Lagrange interpolation. *SIAM Review*, 46(3):501–517, January 2004. doi:10.1137/S0036144502417715.

[3] P. D. Brubeck, Y. Nakatsukasa, and L. N. Trefethen. Vandermonde with Arnoldi. *SIAM Review*, 63(2):405–415, January 2021. doi:10.1137/19M130100X.

[4] A. Budisa, X. Hu, M. Kuchta, K.-A. Mardal, and L. Zikatanov. Rational approximation preconditioners for multiphysics problems. In I. Georgiev, M. Datcheva, K. Georgiev, and G. Nikolov, editors, *Numerical Methods and Applications*, pages 100–113. Springer Nature Switzerland, 2023. doi:10.1007/978-3-031-32412-3_9.

[5] S. Costa and L. N. Trefethen. AAA-least squares rational approximation and solution of Laplace problems. In A. Hujdurović, K. Kutnar, D. Marušič, Š Miklavič, T. Pisan-

ski, and P. Šparl, editors, *European Congress of Mathematics*, pages 511–534. EMS Press, 1 edition, July 2023. doi:10.4171/8ecm/16.

[6] T. A. Driscoll, N. Hale, and L. N. Trefethen. *Chebfun Guide*. Pafnuty Publications, 2014.

[7] T. A. Driscoll, Y. Nakatsukasa, and L. N. Trefethen. AAA rational approximation on a continuum. *SIAM Journal on Scientific Computing*, 46(2):A929–A952, April 2024. doi:10.1137/23M1570508.

[8] S.-I. Filip, Y. Nakatsukasa, L. N. Trefethen, and B. Beckermann. Rational minimax approximation via adaptive barycentric representations. *SIAM Journal on Scientific Computing*, 40(4):A2427–A2455, January 2018. doi:10.1137/17M1132409.

[9] A. Gopal and L. N. Trefethen. Representation of conformal maps by rational functions. *Numerische Mathematik*, 142(2):359–382, June 2019. doi:10.1007/s00211-019-01023-z.

[10] A. Gopal and L. N. Trefethen. Solving Laplace problems with corner singularities via rational functions. *SIAM Journal on Numerical Analysis*, 57(5):2074–2094, January 2019. doi:10.1137/19m125947x.

[11] T. Haut, G. Beylkin, and L. Monzón. Solving Burgers' equation using optimal rational approximations. *Applied and Computational Harmonic Analysis*, 34(1):83–95, January 2013. doi:10.1016/j.acha.2012.03.004.

[12] A. Herremans, D. Huybrechs, and L. N. Trefethen. Resolution of singularities by rational functions. *SIAM Journal on Numerical Analysis*, 61(6):2580–2600, December 2023. doi:10.1137/23M1551821.

[13] A. Hochman. FastAAA: A fast rational-function fitter. In *2017 IEEE 26th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, pages 1–3, October 2017. doi:10.1109/EPEPS.2017.8329756.

[14] M. Kittisopikul, T. E. Holy, and T. Aschan. Interpolations.jl. `https://github.com/JuliaMath/Interpolations.jl`, May 2025.

[15] J. Lampert. Kernelinterpolation.jl: Multivariate (generalized) scattered data interpolation with symmetric (conditionally) positive definite kernel functions in arbitrary dimension. `https://dx.doi.org/10.5281/zenodo.12599880`, June 2024.

[16] P. Lietaert, K. Meerbergen, J. Pérez, and B. Vandereycken. Automatic rational approximation and linearization of nonlinear eigenvalue problems. *IMA Journal of Numerical Analysis*, 42(2):1087–1115, April 2022. doi:10.1093/imanum/draa098.

[17] L. Milne-Thompson. *The Calculus of Finite Differences*. Macmillan and Company, 1933.

[18] Y. Nakatsukasa, O. Sète, and L. N. Trefethen. The AAA algorithm for rational approximation. *SIAM Journal on Scientific Computing*, 40(3):A1494–A1522, January 2018. doi:10.1137/16m1106122.

[19] Y. Nakatsukasa and L. N. Trefethen. An algorithm for real and complex rational minimax approximation. *SIAM Journal on Scientific Computing*, 42(5):A3157–A3179, January 2020. doi:10.1137/19M1281897.

[20] D. J. Newman. Rational approximation to |x|. *Michigan Math Journal*, 11:11–14, 1964. doi:10.1307/mmj/1028999029.

[21] S. Olver and A. Townsend. A practical framework for infinite-dimensional linear algebra. In *Proceedings of the 1st Workshop for High Performance Technical Computing in Dynamic Languages – HPTCDL '14*, 2014.

[22] R. B. Platte, L. N. Trefethen, and A. B. J. Kuijlaars. Impossibility of fast stable approximation of analytic functions from equispaced samples. *SIAM Review*, 53(2):308–318, January 2011. doi:10.1137/090774707.

[23] Andrea Carracedo Rodriguez, Linus Balicki, and Serkan Gugercin. The p-AAA algorithm for data-driven modeling of parametric dynamical systems. *SIAM Journal on Scientific Computing*, 45(3):A1332–A1358, 2023. doi:10.1137/20M1322698. https://doi.org/10.1137/20M1322698.

[24] O. Salazar Celis. Adaptive Thiele interpolation. *ACM Commun. Comput. Algebra*, 56(3):125–132, April 2023. doi:10.1145/3594252.3594254.

[25] O. Salazar Celis. Numerical continued fraction interpolation. *Ukrainian Mathematical Journal*, 76(4):635–648, September 2024. doi:10.1007/s11253-024-02344-5.

[26] C. Schneider and W. Werner. Some new aspects of rational interpolation. *Mathematics of Computation*, 47(175):285–299, 1986. doi:10.1090/S0025-5718-1986-0842136-8.

[27] L. N. Trefethen. Numerical conformal mapping with rational functions. *Computational Methods and Function Theory*, 20(3):369–387, November 2020. doi:10.1007/s40315-020-00325-w.

[28] L. N. Trefethen, Y. Nakatsukasa, and J. A. C. Weideman. Exponential node clustering at singularities for rational approximation, quadrature, and PDEs. *Numerische Mathematik*, 147(1):227–254, January 2021. doi:10.1007/s00211-020-01168-2.

[29] H. Wilber, A. Damle, and A. Townsend. Data-driven algorithms for signal processing with trigonometric rational functions. *SIAM Journal on Scientific Computing*, 44(3):C185–C209, June 2022. doi:10.1137/21M1420277.