
baryrat

Jun 16, 2021

Contents:

1	Indices and tables	5
	Python Module Index	7
	Index	9

A Python package for barycentric rational approximation.

class `baryrat.BarycentricRational` (*z, f, w*)

A class representing a rational function in barycentric representation.

Parameters

- **z** (*array*) – the interpolation nodes
- **f** (*array*) – the values at the interpolation nodes
- **w** (*array*) – the weights

The rational function has the interpolation property $r(z_j) = f_j$ at all nodes where $w_j \neq 0$.

__call__ (*x*)

Evaluate rational function at all points of *x*.

gain ()

The gain in a poles-zeros-gain representation of the rational function, or equivalently, the value at infinity.

poles (*use_mp=False*)

Return the poles of the rational function.

If *use_mp* is *True*, uses the `mpmath` package to compute the result using 100-digit precision arithmetic.
If an integer is passed, uses that number of digits to compute the result.

polres (*use_mp=False*)

Return the poles and residues of the rational function.

The *use_mp* argument has the same meaning as for `poles()` and is only used during computation of the poles.

reciprocal ()

Return a new *BarycentricRational* which is the reciprocal of this one.

zeros (*use_mp=False*)

Return the zeros of the rational function.

If *use_mp* is *True*, uses the `mpmath` package to compute the result using 100-digit precision arithmetic.
If an integer is passed, uses that number of digits to compute the result.

`baryrat.aaa` (*Z, F, tol=1e-13, mmax=100, return_errors=False*)

Compute a rational approximation of *F* over the points *Z* using the AAA algorithm.

Parameters

- **Z** (*array*) – the sampling points of the function. Unlike for interpolation algorithms, where a small number of nodes is preferred, since the AAA algorithm chooses its support points adaptively, it is better to provide a finer mesh over the support.
- **F** – the function to be approximated; can be given as a function or as an array of function values over *Z*.
- **tol** – the approximation tolerance
- **mmax** – the maximum number of iterations/degree of the resulting approximant
- **return_errors** – if *True*, also return the history of the errors over all iterations

Returns an object which can be called to evaluate the rational function, and can be queried for the poles, residues, and zeros of the function.

Return type *BarycentricRational*

For more information, see the paper

The AAA Algorithm for Rational Approximation

Yuji Nakatsukasa, Olivier Sete, and Lloyd N. Trefethen

SIAM Journal on Scientific Computing 2018 40:3, A1494-A1522

as well as the Chebfun package <<http://www.chebfun.org>>. This code is an almost direct port of the Chebfun implementation of aaa to Python.

```
baryrat.brasil(f, interval, deg, tol=0.0001, maxiter=1000, max_step_size=0.1, step_factor=0.1,  
              npi=100, init_steps=100, poly=False, info=False)
```

Best Rational Approximation by Successive Interval Length adjustment.

Parameters

- **f** – the real scalar function to be approximated
- **interval** – the bounds (a, b) of the approximation interval
- **deg** – the degree of the numerator and denominator of the rational approximation
- **tol** – the maximum allowed deviation from equioscillation
- **maxiter** – the maximum number of iterations
- **max_step_size** – the maximum allowed step size
- **step_factor** – factor for adaptive step size choice
- **npi** – points per interval for error calculation. If $npi < 0$, golden section search with $-npi$ iterations is used instead of sampling
- **init_steps** – how many steps of the initialization iteration to run
- **poly** – if true, compute polynomial best approximation instead
- **info** – whether to return an additional object with details

Returns the computed rational approximation. If *info* is True, instead returns a pair containing the approximation and an object with additional information (see below).

Return type *BarycentricRational*

The *info* object returned along with the approximation if *info=True* has the following members:

- **converged** (bool): whether the method converged to the desired tolerance **tol**
- **error** (float): the maximum error of the approximation
- **deviation** (float): the relative error between the smallest and the largest equioscillation peak. The convergence criterion is **deviation** \leq **tol**.
- **nodes** (array): the abscissae of the interpolation nodes ($2 \cdot \text{deg} + 1$)
- **iterations** (int): the number of iterations used, including the initialization phase
- **errors** (array): the history of the maximum error over all iterations
- **deviations** (array): the history of the deviation over all iterations
- **stepsizes** (array): the history of the adaptive step size over all iterations

Additional information about the resulting rational function, such as poles, residues and zeroes, can be queried from the *BarycentricRational* object itself.

```
baryrat.floater_hormann(nodes, values, blending)
```

Compute the Floater-Hormann rational interpolant for the given nodes and values. See (Floater, Hormann 2007), DOI 10.1007/s00211-007-0093-y.

The blending parameter (usually called d in the literature) is an integer between 0 and n (inclusive), where $n+1$ is the number of interpolation nodes. For functions with higher smoothness, the blending parameter may be chosen higher. For $d=n$, the result is the polynomial interpolant.

Returns an instance of *BarycentricRational*.

`baryrat.interpolate_poly(nodes, values)`

Compute the interpolating polynomial for the given nodes and values in barycentric form.

`baryrat.interpolate_rat(nodes, values)`

Compute a rational function which passes through all given node/value pairs. The number of nodes must be odd, and they should be passed in strictly increasing or strictly decreasing order.

`baryrat.interpolate_with_poles(nodes, values, poles)`

Compute a rational function which interpolates the given values at the given nodes and which has the given poles.

The arrays `nodes` and `values` should have length n , and `poles` should have length $n - 1$.

CHAPTER 1

Indices and tables

- `genindex`
- `modindex`
- `search`

b

baryrat, [1](#)

Symbols

`__call__()` (*baryrat.BarycentricRational method*), 1

A

`aaa()` (*in module baryrat*), 1

B

`BarycentricRational` (*class in baryrat*), 1

`baryrat` (*module*), 1

`brasil()` (*in module baryrat*), 2

F

`floater_hormann()` (*in module baryrat*), 2

G

`gain()` (*baryrat.BarycentricRational method*), 1

I

`interpolate_poly()` (*in module baryrat*), 3

`interpolate_rat()` (*in module baryrat*), 3

`interpolate_with_poles()` (*in module baryrat*),
3

P

`poles()` (*baryrat.BarycentricRational method*), 1

`polres()` (*baryrat.BarycentricRational method*), 1

R

`reciprocal()` (*baryrat.BarycentricRational method*),
1

Z

`zeros()` (*baryrat.BarycentricRational method*), 1