baryrat

Mar 11, 2022

Contents:

1 Indices and tables **9**

Python Module Index **11**

Index **13**

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 != 0.

__call__(*x*)

Evaluate rational function at all points of *x*.

degree(*tol=1e-12*)

Compute the pair *(m,n)* of true degrees of the numerator and denominator.

degree_denom(*tol=1e-12*)

Compute the true degree of the denominator polynomial.

Uses a result from [Berrut, Mittelmann 1997].

degree_numer(*tol=1e-12*)

Compute the true degree of the numerator polynomial.

Uses a result from [Berrut, Mittelmann 1997].

denominator()

Return a new `BarycentricRational` which represents the denominator polynomial.

eval_deriv(*x*, *k=1*)

Evaluate the *k*-th derivative of this rational function at a scalar node *x*, or at each point of an array *x*. Only the cases *k* <= *2* are currently implemented.

Note that this function may incur significant numerical error if *x* is very close (but not exactly equal) to a node of the barycentric rational function.

References

https://doi.org/10.1090/S0025-5718-1986-0842136-8 (C. Schneider and W. Werner, 1986)

gain()

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

jacobians(*x*)

Compute the Jacobians of *r(x)*, where *x* may be a vector of evaluation points, with respect to the node, value, and weight vectors.

The evaluation points *x* may not lie on any of the barycentric nodes (unimplemented).

Returns A triple of arrays with as many rows as *x* has entries and as many columns as the barycentric function has nodes, representing the Jacobians with respect to `self.nodes`, `self.values`, and `self.weights`, respectively.

numerator()

Return a new `BarycentricRational` which represents the numerator polynomial.

order

The order of the barycentric rational function, that is, the maximum degree that its numerator and denominator may have, or the number of interpolation nodes minus one.

poles(*use_mp=False*)
Return the poles of the rational function.

If `use_mp` is `True`, uses the `flamp` multiple precision package to compute the result. This option is automatically enabled if `uses_mp()` is True.

polres(*use_mp=False*)
Return the poles and residues of the rational function.

If `use_mp` is `True`, uses the `flamp` multiple precision package to compute the result. This option is automatically enabled if `uses_mp()` is True.

reciprocal()
Return a new `BarycentricRational` which is the reciprocal of this one.

reduce_order()
Return a new `BarycentricRational` which represents the same rational function as this one, but with minimal possible order.

See (Ionita 2013), PhD thesis.

uses_mp()
Checks whether any of the data of this rational function uses extended precision.

zeros(*use_mp=False*)
Return the zeros of the rational function.

If `use_mp` is `True`, uses the `flamp` multiple precision package to compute the result. This option is automatically enabled if `uses_mp()` is True.

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
> https://doi.org/10.1137/16M1106122

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.bpane`(*f*, *f_deriv*, *interval*, *deg*, *tol=1e-08*, *maxiter=1000*, *verbose=0*, *info=False*)

Best polynomial approximation using Newton's algorithm.

Compute the best uniform polynomial approximation of degree *deg* of the function *f* with derivative *f_deriv* in the given *interval*.

References

https://www.ricam.oeaw.ac.at/files/reports/21/rep21-46.pdf

Parameters

- **f** – the scalar function to be approximated. Must be able to operate on arrays of arguments.
- **f_deriv** – the derivative of *f*. If *None* is passed, a central finite difference quotient is used to approximate the derivative.
- **interval** – the bounds *(a, b)* of the approximation interval
- **deg** – the degree of the approximating polynomial
- **tol** – the maximum allowed deviation from equioscillation
- **maxiter** – the maximum number of iterations
- **verbose** – if greater than 0, the progress is printed in each iteration
- **info** – whether to return an additional object with details

Returns the computed polynomial 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:

- **error** (float): the maximum error of the approximation
- **lam** (float): the quantity lambda (signed error)
- **deviation** (float): the relative error between the smallest and the largest equioscillation peak. The convergence criterion is **deviation** <= **tol**.
- **nodes** (array): the abscissae of the interpolation nodes (*deg* + 1)
- **iterations** (int): the number of iterations used

`baryrat.brane`(*f*, *f_deriv*, *interval*, *deg*, *tol=1e-16*, *maxiter=1000*, *initial_nodes=None*, *verbose=0*, *info=False*)

Best rational approximation using Newton's algorithm.

Compute the best uniform rational approximation of the function *f* with derivative *f_deriv* in the given *interval*.

References

https://www.ricam.oeaw.ac.at/files/reports/22/rep22-02.pdf

Parameters

- **f** – the scalar function to be approximated. Must be able to operate on arrays of arguments.
- **f_deriv** – the derivative of *f*. If *None* is passed, a central finite difference quotient is used to approximate the derivative.
- **interval** – the bounds *(a, b)* of the approximation interval

- **deg** – the degree *(m, n)* of the approximating rational function
- **tol** – the maximum allowed deviation from equioscillation
- **maxiter** – the maximum number of iterations
- **initial_nodes** – an array of length *m + n + 1* with the starting interpolation nodes. If not given, Chebyshev nodes of the first kind are used.
- **verbose** – if greater than 0, the progress is printed in each iteration
- **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:

- **error** (float): the maximum error of the approximation
- **lam** (float): the quantity lambda (signed error)
- **deviation** (float): the relative error between the smallest and the largest equioscillation peak. The convergence criterion is **deviation <= tol**.
- **nodes** (array): the abscissae of the interpolation nodes (*m + n + 1*)
- **iterations** (int): the number of iterations used

Note: This function requires the `gmpy2` and `flamp` packages for extended precision. Remember to set the precision by `flamp.set_dps(...)` before use.

`baryrat.`**`brasil`**(*f*, *interval*, *deg*, *tol=0.0001*, *maxiter=1000*, *max_step_size=0.1*, *step_factor=0.1*, *npi=-30*, *init_steps=100*, *info=False*)

Best Rational Approximation by Successive Interval Length adjustment.

Computes best rational or polynomial approximations in the maximum norm by the BRASIL algorithm (see reference below).

References

https://doi.org/10.1007/s11075-020-01042-0

Parameters

- **f** – the scalar function to be approximated. Must be able to operate on arrays of arguments.
- **interval** – the bounds *(a, b)* of the approximation interval
- **deg** – the degree of the numerator *m* and denominator *n* of the rational approximation; either an integer (*m=n*) or a pair *(m, n)*. If *n = 0*, a polynomial best approximation is computed.
- **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. For high-accuracy results, *npi=-30* is typically a good choice.
- **init_steps** – how many steps of the initialization iteration to run
- **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 <= tol**.
- **nodes** (array): the abscissae of the interpolation nodes (2*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.

Note: This function supports `gmpy2` for extended precision. To enable this, specify the interval *(a, b)* as *mpfr* numbers, e.g., `interval=(mpfr(0), mpfr(1))`. Also make sure that the function *f* consumes and outputs arrays of *mpfr* numbers; the Numpy function `numpy.vectorize()` may help with this.

`baryrat.`**`chebyshev_nodes`**(*num_nodes*, *interval=(-1.0, 1.0)*, *use_mp=False*)

Compute *num_nodes* Chebyshev nodes of the first kind in the given interval.

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

Compute the Floater-Hormann rational interpolant for the given nodes and values.

Parameters

- **nodes** (*array*) – the interpolation nodes (length *n*)
- **values** (*array*) – the function values at the interpolation nodes (length *n*)
- **blending** (*int*) – the blending parameter (usually called *d* in the literature), an integer between 0 and *n-1* (inclusive). For functions with higher smoothness, the blending parameter may be chosen higher. For *d=n-1*, the result is the polynomial interpolant.

Returns the rational interpolant

Return type *BarycentricRational*

References

(Floater, Hormann 2007): https://doi.org/10.1007/s00211-007-0093-y

`baryrat.`**`interpolate_poly`**(*nodes*, *values*)

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

Parameters

- **nodes** (*array*) – the interpolation nodes
- **values** (*array*) – the function values at the interpolation nodes

Returns the polynomial interpolant

Return type *BarycentricRational*

`baryrat.`**`interpolate_rat`**(*nodes*, *values*, *use_mp=False*)

Compute a rational function which interpolates the given nodes/values.

Parameters

- **nodes** (*array*) – the interpolation nodes; must have odd length and be passed in strictly increasing or decreasing order
- **values** (*array*) – the values at the interpolation nodes
- **use_mp** (*bool*) – whether to use `gmpy2` for extended precision. Is automatically enabled if *nodes* or *values* use `gmpy2`.

Returns the rational interpolant. If there are $2n + 1$ nodes, both the numerator and denominator have degree at most n.

Return type *BarycentricRational*

References

https://doi.org/10.1109/LSP.2007.913583

`baryrat.`**`interpolate_with_degree`**(*nodes*, *values*, *deg*, *use_mp=False*)

Compute a rational function which interpolates the given nodes/values with given degree m of the numerator and n of the denominator.

Parameters

- **nodes** (*array*) – the interpolation nodes
- **values** (*array*) – the values at the interpolation nodes
- **deg** – a pair (m, n) of the degrees of the interpolating rational function. The number of interpolation nodes must be $m + n + 1$.
- **use_mp** (*bool*) – whether to use `gmpy2` for extended precision. Is automatically enabled if *nodes* or *values* use `gmpy2`.

Returns the rational interpolant

Return type *BarycentricRational*

References

https://doi.org/10.1016/S0377-0427(96)00163-X

`baryrat.`**`interpolate_with_poles`**(*nodes*, *values*, *poles*, *use_mp=False*)

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

Parameters

- **nodes** (*array*) – the interpolation nodes (length *n*)
- **values** (*array*) – the function values at the interpolation nodes (length *n*)
- **poles** (*array*) – the locations of the poles of the rational function (length *n-1*)
- **use_mp** (*bool*) – whether to use `gmpy2` for extended precision

Returns the rational interpolant with the given poles

Return type *BarycentricRational*

CHAPTER 1

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

baryrat, 1

Index

Symbols

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

A

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

B

`BarycentricRational` (*class in baryrat*), 1
`baryrat` (*module*), 1
`bpane()` (*in module baryrat*), 2
`brane()` (*in module baryrat*), 3
`brasil()` (*in module baryrat*), 4

C

`chebyshev_nodes()` (*in module baryrat*), 5

D

`degree()` (*baryrat.BarycentricRational method*), 1
`degree_denom()` (*baryrat.BarycentricRational method*), 1
`degree_numer()` (*baryrat.BarycentricRational method*), 1
`denominator()` (*baryrat.BarycentricRational method*), 1

E

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

F

`floater_hormann()` (*in module baryrat*), 5

G

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

I

`interpolate_poly()` (*in module baryrat*), 5
`interpolate_rat()` (*in module baryrat*), 6
`interpolate_with_degree()` (*in module baryrat*), 6
`interpolate_with_poles()` (*in module baryrat*), 6

J

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

N

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

O

`order` (*baryrat.BarycentricRational attribute*), 1

P

`poles()` (*baryrat.BarycentricRational method*), 2
`polres()` (*baryrat.BarycentricRational method*), 2

R

`reciprocal()` (*baryrat.BarycentricRational method*), 2
`reduce_order()` (*baryrat.BarycentricRational method*), 2

U

`uses_mp()` (*baryrat.BarycentricRational method*), 2

Z

`zeros()` (*baryrat.BarycentricRational method*), 2