Package: minimaxApprox (via r-universe)

October 15, 2026

Type Package

Title Minimax Approximation of Functions by Polynomials and Rational Functions

Version 0.6.0

Date 2026-07-17

Description Implements minimax approximation of functions via the Remez (1962) algorithm for polynomials and the Cody-Fraser-Hart (1968) <doi:10.1007/BF02162506> algorithm for rational functions, as well as their barycentric formulations: the Pachón-Trefethen (2009) <doi:10.1007/s10543-009-0240-1> algorithm for polynomials and the Filip-Nakatsukasa-Trefethen-Beckermann (2018) <doi:10.1137/17M1132409> algorithm for rational functions, which provide improved numerical stability at higher degrees and on wider intervals.

License MPL-2.0

URL https://github.com/aadler/minimaxApprox

BugReports https://github.com/aadler/minimaxApprox/issues

Imports stats, graphics

Suggests tinytest, covr, knitr, rmarkdown

VignetteBuilder knitr

ByteCompile yes

NeedsCompilation yes

Encoding UTF-8

UseLTO yes

Repository https://aadler.r-universe.dev

Date/Publication 2026-07-17 05:54:59 UTC

RemoteUrl https://github.com/aadler/minimaxapprox

RemoteRef HEAD

RemoteSha 9eb7ff3496b548d651c82f86bad2f85546533003

Contents

minimaxApprox-package 2
coef.minimaxApprox 3
minimaxApprox 4
minimaxErr 11
minimaxEval 12
plot.minimaxApprox 13
print.minimaxApprox 14

Index **15**

`minimaxApprox-package` *Minimax Approximation of Functions by Polynomials and Rational Functions*

Description

Implements minimax approximation of functions via the Remez (1962) algorithm for polynomials and the Cody-Fraser-Hart (1968) <doi:10.1007/BF02162506> algorithm for rational functions, as well as their barycentric formulations: the Pachón-Trefethen (2009) <doi:10.1007/s10543-009-0240-1> algorithm for polynomials and the Filip-Nakatsukasa-Trefethen-Beckermann (2018) <doi:10.1137/17M1132409> algorithm for rational functions, which provide improved numerical stability at higher degrees and on wider intervals.

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

Avraham Adler [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-3039-0703>)

Maintainer: Avraham Adler <Avraham.Adler@gmail.com>

`coef.minimaxApprox` *Extract coefficients from a* `"minimaxApprox"` *object*

Description

Extracts the numerator and denominator vectors from a `"minimaxApprox"` object. For objects fitted with the Chebyshev or barycentric basis, it will also extract the corresponding monomial-basis coefficients.

Usage

```
## S3 method for class 'minimaxApprox'
coef(object, ...)
```

Arguments

`object`	An object inheriting from class `"minimaxApprox"`.
`...`	Other arguments.

Value

Coefficients extracted from the `"minimaxApprox"` object. A list containing:

`a`	The polynomial coefficients or the rational numerator coefficients.
`b`	The rational denominator coefficients. Missing for polynomial approximation.
`aMono`	The polynomial coefficients or the rational numerator coefficients for the monomial basis when the approximation was done using Chebyshev polynomials or the barycentric basis. Missing if only the monomial basis was used.
`bMono`	The rational denominator coefficients for the monomial basis when the approximation was done using Chebyshev polynomials or the barycentric basis. Missing if either only the monomial basis was used or for polynomial approximation.

Author(s)

Avraham Adler <Avraham.Adler@gmail.com>

See Also

`minimaxApprox`

Examples

```
PP <- minimaxApprox(exp, 0, 1, 5)
coef(PP)
identical(unlist(coef(PP), use.names = FALSE), c(PP$a, PP$aMono))

RR <- minimaxApprox(exp, 0, 1, c(2, 3), basis = "m")
coef(RR)
identical(coef(RR), list(a = RR$a, b = RR$b))
```

`minimaxApprox` *Minimax Approximation of Functions*

Description

Calculates minimax approximations to functions. Polynomial approximation uses either the Remez (1962) algorithm—with the basis polynomials evaluated using monomials or Chebyshev polynomials of the first kind—or the barycentric variant of Pachón & Trefethen (2009). Rational approximation uses either the Cody–Fraser–Hart (Cody et al., 1968) version of the Remez algorithm or the barycentric variant of Filip et al. (2018). When using monomials as the polynomial basis, the Compensated Horner Scheme of Langlois et al. (2006) is used.

Usage

```
minimaxApprox(fn, lower, upper, degree, relErr = FALSE, basis ="Chebyshev",
              xi = NULL, opts = list())
```

Arguments

`fn`	function; A vectorized univariate function having `x` as its first argument. This could be a built-in R function, a predefined function, or an anonymous function defined in the call; see **Examples**.
`lower`	numeric; The lower bound of the approximation interval.
`upper`	numeric; The upper bound of the approximation interval.
`degree`	integer; Either a single value representing the requested degree for polynomial approximation or a vector of length 2 representing the requested degrees of the numerator and denominator for rational approximation.
`relErr`	logical; If `TRUE`, calculate the minimax approximation using *relative* error. The default is `FALSE` which uses *absolute* error.
`basis`	character; Which polynomial basis to use in the analysis. `"Monomial"` uses the standard x^k basis. `"Chebyshev"` uses the Chebyshev polynomials of the first kind, T_k. `"Barycentric"` for polynomials uses the barycentric-Remez algorithm of Pachón and Trefethen (2009), which represents the trial polynomial by its values at the reference rather than by coefficients in a fixed basis, avoiding the ill-conditioned linear solve entirely (see **Barycentric Basis** below). For rational approximation, `"Barycentric"` uses the adaptive barycentric algorithm of Filip, Nakatsukasa, Trefethen, and Beckermann (2018) and supports **absolute** error only; requesting it with `relErr = TRUE` is an error. The default is `"Chebyshev"`, and the parameter is case-insensitive and may be abbreviated.
`xi`	numeric; **Deprecated as of 0.6.0 and scheduled for removal in the next release**—the current exchange algorithm derives its reference from the error curve directly, so a user-supplied initial reference is no longer needed. Until removal: for rational approximation, a vector of initial points of the correct length—$\sum(\texttt{degree}) + 2$. If missing, the approximation will use the appropriate Chebyshev nodes, except that rational approximation in the barycentric basis initializes

from the error extrema of an AAA approximant (see **Barycentric Basis** below); a supplied `xi` of the correct length overrides that initialization. Polynomial approximation **always** uses Chebyshev nodes and will ignore `xi` with a message.

`opts` list; Configuration options including:

- `maxiter`: integer; The maximum number of iterations to attempt convergence. Defaults to 100.
- `miniter`: integer; The minimum number of iterations before allowing convergence. Defaults to 10.
- `conviter`: integer; The number of successive iterations with the same results allowed before assuming no further convergence is possible. Defaults to 30. Will overwrite `maxiter` and `miniter` if `conviter` is explicitly passed and is larger than either one.
- `showProgress`: logical; If `TRUE` will print error values at each iteration.
- `convrat`: numeric; The convergence ratio tolerance. Defaults to $1 + 1 \times 10^{-9}$. See **Details**.
- `tol`: numeric; The absolute difference tolerance. Defaults to 1×10^{-14}. See **Details**.
- `tailtol`: numeric; The tolerance of the coefficient of the largest power of `x` to be ignored when performing the polynomial approximation a second time. Defaults to the smaller of 1×10^{-10} or $\frac{\texttt{upper} - \texttt{lower}}{10^6}$. Set to `NULL` to skip the `degree + 1` check completely. See **Details**.
- `ztol`: numeric; The tolerance for each polynomial or rational numerator or denominator coefficient's contribution to **not** to be set to 0. Similar to polynomial `tailtol` but applied at each step of the algorithm. Defaults to `NULL` which leaves all coefficients as they are regardless of magnitude. See **Details**.

Details

Convergence: The function implements the Remez algorithm using linear approximation, chiefly as described by Cody et al. (1968). Convergence is considered achieved when all three of the following criteria are met:

1. The observed error magnitudes are within tolerance of the expected error—the **Distance Test**.
2. The observed error magnitudes are within tolerance of each other—the **Magnitude Test**.
3. The observed error signs oscillate—the **Oscillation Test**.

"Within tolerance" can be met in one of two ways:

1. **Difference**: The difference between the absolute magnitudes is less than or equal to `tol`.
2. **Ratio**: The ratio between the absolute magnitudes of the larger and smaller is less than or equal to `convrat`.

For efficiency, the **Distance Test** is taken between the absolute value of the largest observed error and the absolute value of the expected error. Similarly, the **Magnitude Test** is taken between the absolute value of the largest observed error and the absolute value of the smallest observed error. Both tests can be passed by **either** being within `tol` or `convrat` as described above. However, when the **Difference** test returns values less than machine precision, it is ignored in favor of the **Ratio** test.

When the error values remain within tolerance of each other over `conviter` iterations, the algorithm will stop, as it is expected that no further precision will be gained by continued iterations.

Polynomial Evaluation: Monomial polynomials are evaluated using the Compensated Horner Scheme of Langlois et al. (2006) to enhance both stability and precision. Chebyshev polynomials are evaluated on an internally affine-mapped domain (see **Mapped Chebyshev Basis** below). There may be cases where the algorithm will fail using the monomial basis but succeed using Chebyshev polynomials and vice versa. The default is to use the Chebyshev polynomials.

Mapped Chebyshev Basis: Chebyshev polynomials T_k are only well-conditioned on $[-1, 1]$; off that interval, conditioning degrades severely and grows with both distance from $[-1, 1]$ and degree (for example, degree 10 on $[5, 6]$ has a condition number around 10^{24} unmapped versus about 1.4 mapped). To avoid this, the Chebyshev basis internally maps the requested range $[l, u]$ to $[-1, 1]$ via $z = \frac{2x-(l+u)}{u-l}$ and evaluates T_k on z. The returned `a` (and `b`) are therefore coefficients of $T_k(z)$, **not** $T_k(x)$, whenever the fitted range is not $[-1, 1]$; on $[-1, 1]$ itself the map is the identity and `a`/ `b` are unchanged from prior releases.

`aMono`/`bMono` remain coefficients in the natural, raw x in every case, and are the stable way to consume a Chebyshev-basis result off $[-1, 1]$ without needing to know about the internal map. They are obtained by converting the mapped-Chebyshev coefficients to monomials in z as before and then composing that polynomial with the affine map above to recover monomial coefficients in x.

This composition step is itself a monomial-basis operation and inherits monomial ill-conditioning at high degree, the same way `aMono` always has. On ranges far from $[-1, 1]$ at higher degrees, the composition's own error can grow to dominate the underlying fit's accuracy − for example, a degree-15 fit on $[10, 20]$ can have a well-conditioned Chebyshev-basis error near 10^{-7} while the composed `aMono` coefficients disagree with the Chebyshev-basis evaluation by closer to 10^{-6} on the same range. When degree is high and the range is both wide and far from $[-1, 1]$, prefer evaluating with the Chebyshev basis directly (the default for `minimaxEval`/`minimaxErr`) over consuming `aMono` directly, if accuracy at that level matters.

Barycentric Basis:

Polynomial Approximation: Selecting `basis = "Barycentric"` for **polynomial** approximation uses the barycentric-Remez algorithm of Pachón and Trefethen (2009). Rather than solving an $(n+2) \times (n+2)$ linear system in a fixed basis at each iteration—the source of the conditioning failures that cap the achievable degree in the classical bases—it represents the trial polynomial by its *values* at the reference points, with barycentric weights, and computes the leveled error h in closed form. Evaluation uses the second barycentric formula, which is forward-stable for well-distributed nodes such as these (Higham, 2004). Weights are assembled with capacity scaling so that wide or far-off-center intervals and high degrees do not over- or underflow. As a result the barycentric basis typically converges at degrees where the classical bases fail with a singular-matrix error, and is well-conditioned off $[-1, 1]$ without a separate mapping step.

Functions that are exactly representable, or resolved to machine precision at the requested degree (the cases described in **Machine-Precision-Resolved Functions** below), are handled *natively*: the closed-form leveled error is 0 and the trial polynomial interpolates `fn`, so no singular-solve rescue is needed. Relative error is supported via the analogous closed form; however, if `fn` is **exactly** 0 at a reference point (typically an endpoint of $[l, u]$, which is always a reference node), the relative error there is $0/0$ and the relative minimax does not exist—this is reported as a clear error rather than a silent or crashed result. Use absolute error, or an interval whose endpoints are not zeros of `fn`.

The returned object stores the barycentric representation (reference nodes, weights, and nodal values) as its primary form; `a`, `aMono`, and the convenience methods are derived from it. Because the coefficient recovery is the one step that can lose accuracy at high degree, a conversion residual is reported (see `convResid` in **Value**); for functions with poles near the approximation interval this residual can be noticeably larger at some intermediate degrees (for example, a degree-20 fit of $1/(1+x^2)$ on $[-1,1]$ shows a residual near 10^{-8} while degrees 10 and 50 are near 10^{-15}). The `tailtol` degree-`n + 1` retry and the `ztol` coefficient-zeroing test do not apply to the barycentric basis, which has no coefficient solve to restart or prune.

Rational Approximation: **Rational approximation** in the barycentric basis implements the adaptive-barycentric minimax algorithm of Filip, Nakatsukasa, Trefethen, and Beckermann (2018): the trial rational is represented as a quotient of barycentric forms over *support points* chosen as every other reference point, and the leveled error at each iteration is an eigenvalue of a small *symmetric* matrix, so there is again no ill-conditioned linear solve. The initial reference is taken from the error extrema of an AAA approximant (Nakatsukasa, Sète, and Trefethen, 2018) computed on a dense sample grid, falling back to Chebyshev points when the AAA error curve does not supply enough alternations. The returned `a`/`b` (and `aMono`/`bMono`) are recovered from the barycentric representation with the denominator normalized to a constant term of 1, matching the classical rational path, and `convResid` is a named length-2 vector reporting the relative conversion residual of the numerator (`"a"`) and denominator (`"b"`) separately.
The rational barycentric implementation is a deliberate subset of the published algorithm, with the following limitations. Only **absolute** error is supported; `relErr = TRUE` is rejected with an error. Degenerate or *defective* problems—for example, requesting even numerator and denominator degrees for an even function, where the true best approximation has fewer effective parameters than the requested type—are **detected and reported as errors** rather than repaired by degree reduction as in the full published algorithm; the error message identifies the detected cause (a rank-deficient basis matrix, no pole-free solution of the leveled-error equations, or a collapsed barycentric weight). A trial denominator with a zero inside the approximation interval is likewise reported as an error, mirroring the classical rational path. Finally, for *non-normal* requests (degrees for which the true best approximation equioscillates on more than $m+n+2$ points, as happens for even or odd functions at generic degrees) the fixed-size exchange can converge to a leveled fixed point that is not the global minimax; the returned approximation is then valid and its expected error is a lower bound on the true minimax error (de la Vallée Poussin). To guard against this, every approximation path verifies a converged result against a dense evaluation grid and raises a warning whenever the dense-grid error materially exceeds the leveled error; the check is skipped only where its ratio would be meaningless noise (results at the machine-precision floor, and relative-error results when `fn` has an exact zero on the grid). For the rational barycentric path this warning is a structural outcome of non-normal requests. For the polynomial and classical rational paths, the exchange maintains the invariants (whole-interval root search, retention of the global extremum) under which reference-local convergence should not occur, so the check there is a safety net: if it fires, please report it as a bug.

Polynomial Algorithm "Singular Error" Response: When too high of a degree is requested for the tolerance of the algorithm, it often fails with a singular matrix error. In this case, for the *polynomial* version, the algorithm will try looking for an approximation of degree `n + 1`. If it finds one, **and** the contribution of that coefficient to the approximation is $\leq$ `tailtol`, it will ignore that coefficient and return the resulting degree `n` polynomial, as the largest coefficient is effectively 0. The contribution is measured by multiplying that coefficient by the endpoint with the larger absolute magnitude raised to the `n + 1` power. This is done to prevent errors in cases where a very small coefficient is found on a range with very large absolute values and the resulting

contribution to the approximation is **not** *de minimis*. Setting `tailtol` to `NULL` will skip the `n + 1` test completely.

Machine-Precision-Resolved Functions: When the requested degree is high enough that `fn` is resolved to (or extremely near) machine double precision—including functions that are *exactly* representable, such as a quadratic requested at degree 2—there may be no minimax problem left to solve: any polynomial passing through a correctly-chosen reference already matches `fn` to the limits of floating point, so the Remez iteration has nothing left to optimize. This is detected either as a singular linear solve at both degree `n` and degree `n + 1` (see above), or, on some platforms, as the iteration running to `maxiter` or stalling "unchanging" while the observed error is already at the machine-precision floor. In that situation, `minimaxApprox` returns the degree-`n` polynomial that *interpolates* `fn` through the Chebyshev reference, rather than a Remez-optimized polynomial, together with a warning stating that the result is not technically a Remez result. This interpolant equals the true minimax polynomial when `fn` is exactly representable, and is indistinguishable from it (to within floating point) otherwise; `ExpErr` and `ObsErr` are equal in this case, since there is no separate leveled error to report. If the observed error is **not** at the machine floor, the singularity or stall has a genuine cause unrelated to representability, and the original error or convergence warning is raised as before—this fallback never masks real non-convergence.

Close-to-Zero Tolerance: For each step of the algorithms' iterations, the contribution of the found coefficient to the total sum (as measured in the above section) is compared to the `ztol` option. When less than or equal to `ztol`, that coefficient is set to 0. Setting `ztol` to `NULL` skips the test completely. For intervals near or containing zero, setting this option to anything other than `NULL` may result in either non-convergence or poor results. It is recommended to keep it as `NULL`, although there are edge cases where it may allow convergence where a standard call may fail.

Value

`minimaxApprox` returns an object of class `"minimaxApprox"` which inherits from the class list.

The generic accessor function `coef` will extract the numerator and denominator vectors. There are also default `print` and `plot` methods.

An object of class `"minimaxApprox"` is a list containing the following components:

`a` The polynomial or rational numerator coefficients. When using Chebyshev polynomials, these are the coefficients for T_k of the internally mapped variable z (see **Mapped Chebyshev Basis** above) whenever the fitted range is not $[-1, 1]$; on $[-1, 1]$ itself, $z = x$ and these are unchanged from prior releases. When using the barycentric basis, these are the mapped-Chebyshev coefficients recovered from the barycentric representation (see **Barycentric Basis** above). When using monomials, these are the coefficients for x^k.

`b` The rational denominator coefficients. Same semantics as `a` above. Missing for polynomial approximation.

`aMono` When using Chebyshev or barycentric polynomials, these are the polynomial or rational numerator coefficients for monomial expansion in x^k (natural, raw x), regardless of the internal map. See **Mapped Chebyshev Basis** above for an accuracy caveat at high degree on wide, far-off-$[-1, 1]$ ranges; the barycentric basis reports the size of this conversion loss directly in `convResid`. Missing for monomial-based approximation.

`bMono` When using Chebyshev polynomials, these are the rational denominator coefficients for monomial expansion in x^k (natural, raw x; same caveat as `aMono`). Missing for both polynomial and monomial-based rational approximation.

`ExpErr` The absolute value of the expected error as calculated by the Remez algorithms. For a machine-precision-resolved result (see **Machine-Precision-Resolved Functions** above), there is no separate Remez-computed expected error; `ExpErr` equals `ObsErr` in that case.

`ObsErr` The absolute value of largest observed error between the function and the approximation at the extremal points. For a result returned with `Warning = TRUE`, the final reference points need not track the approximation's true extrema, so `ObsErr` can materially understate the true maximum error; verify non-converged results with `minimaxErr` on a dense grid before use.

`iterations` The number of iterations of the algorithm. This does not include any iterations required to converge the error value in rational approximation.

`Extrema` The extrema at which the minimax error was achieved.

`Warning` A logical flag indicating if any warnings were thrown.

`convResid` Barycentric basis only. The conversion residual: the maximum absolute difference between the (accurate) barycentric evaluation of the fitted polynomial and its reconstruction from the recovered coefficients, measured on a dense probe grid. A diagnostic of how much accuracy the coefficient recovery lost; see **Barycentric Basis** above. For rational approximation it is a named length-2 vector giving the *relative* residual of the numerator (`"a"`) and denominator (`"b"`) polynomials separately.

The object also contains the following attributes:

`type` "Rational" or "Polynomial".

`basis` "Monomial", "Chebyshev", or "Barycentric".

`func` The function being approximated.

`range` The range on which the function is being approximated.

`relErr` A logical indicating that relative error was used. If `FALSE`, then absolute error was used.

`tol` The tolerance used for the **Distance Test**.

`convrat` The tolerance used for the **Magnitude Test**.

Note

So long as the package remains in an experimental state—noted by a 0 major version—the API may change at any time; see the 'NEWS' file for breaking changes.

Author(s)

Avraham Adler <Avraham.Adler@gmail.com>

References

Remez, E. I. (1962) *General computational methods of Chebyshev approximation: The problems with linear real parameters*. US Atomic Energy Commission, Division of Technical Information. AEC-tr-4491

Fraser W. and Hart J. F. (1962) "On the computation of rational approximations to continuous functions", *Communications of the ACM*, **5**(7), 401–403, doi:10.1145/368273.368578

Cody, W. J. and Fraser W. and Hart J. F. (1968) "Rational Chebyshev approximation using linear equations", *Numerische Mathematik*, **12**, 242–251, doi:10.1007/BF02162506

Langlois, P. and Graillat, S. and Louvet, N. (2006) "Compensated Horner Scheme", in *Algebraic and Numerical Algorithms and Computer-assisted Proofs*. Dagstuhl Seminar Proceedings, **5391**, doi:10.4230/DagSemProc.05391.3

Pachón, R. and Trefethen, L. N. (2009) "Barycentric-Remez algorithms for best polynomial approximation in the chebfun system", *BIT Numerical Mathematics*, **49**(4), 721–741, doi:10.1007/s1054300902401

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

Higham, N. J. (2004) "The numerical stability of barycentric Lagrange interpolation", *IMA Journal of Numerical Analysis*, **24**(4), 547–556, doi:10.1093/imanum/24.4.547

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

Nakatsukasa, Y. and Sète, O. and Trefethen, L. N. (2018) "The AAA algorithm for rational approximation", *SIAM Journal on Scientific Computing*, **40**(3), A1494–A1522, doi:10.1137/16M1106122

See Also

`minimaxEval`, `minimaxErr`

Examples

```
minimaxApprox(exp, 0, 1, 5)                              # Built-in & polynomial

fn <- function(x) sin(x) ^ 2 + cosh(x)                   # Pre-defined
minimaxApprox(fn, 0, 1, c(2, 3), basis = "m")            # Rational

minimaxApprox(function(x) x ^ 3 / sin(x), 0.7, 1.6, 6L)  # Anonymous

fn <- function(x) besselJ(x, nu = 0)                     # More than one input
b0 <- 0.893576966279167522                               # Zero of besselY
minimaxApprox(fn, 0, b0, c(3L, 3L))                      # Cf. DLMF 3.11.19

minimaxApprox(exp, -1, 1, 14, basis = "b")              # Barycentric: converges
                                                         # at degrees where the
                                                         # classical bases fail
```

minimaxErr *Evaluate the Minimax Approximation Error*

Description

Evaluates the difference between the function and the minimax approximation at x.

Usage

```
minimaxErr(x, mmA)
```

Arguments

x	a numeric vector
mmA	a "minimaxApprox" return object

Details

This is a convenience function to evaluate the approximation error at x. It will use the same polynomial basis as was used in the approximation; see minimaxApprox for more details.

Value

A vector of the same length as x containing the approximation error values.

Author(s)

Avraham Adler <Avraham.Adler@gmail.com>

See Also

minimaxApprox, minimaxEval

Examples

```
# Show results
x <- seq(0, 0.5, length.out = 11L)
mmA <- minimaxApprox(exp, 0, 0.5, 5L)
err <- minimaxEval(x, mmA) - exp(x)
all.equal(err,  minimaxErr(x, mmA))

# Plot results
x <- seq(0, 0.5, length.out = 1001L)
plot(x, minimaxErr(x, mmA), type = "l")
```

`minimaxEval` *Evaluate Minimax Approximation*

Description

Evaluates the rational or polynomial approximation stored in `mmA` at `x`.

Usage

```
minimaxEval(x, mmA, basis = "Chebyshev")
```

Arguments

`x`	a numeric vector
`mmA`	a `"minimaxApprox"` return object
`basis`	character; Which polynomial basis to use in to evaluate the function; see `minimaxApprox` for more details. If Chebyshev is requested but the analysis used only monomials, the calculation will proceed using the monomials with a message. If the analysis used the barycentric basis, evaluation defaults to the (most accurate) stored barycentric representation; requesting `"Chebyshev"` or `"Monomial"` instead evaluates through the converted coefficients with a message, as that conversion can be less accurate (see the object's `convResid`). For a rational barycentric approximation the stored representation and both coefficient routes all evaluate the full numerator/denominator quotient. The default is `"Chebyshev"`, and the parameter is case-insensitive and may be abbreviated.

Details

This is a convenience function to evaluate the approximation at `x`.

Value

A vector of the same length as `x` containing the approximated values.

Author(s)

Avraham Adler <Avraham.Adler@gmail.com>

See Also

`minimaxApprox`, `minimaxErr`

Examples

```
# Show results
x <- seq(0, 0.5, length.out = 11L)
mmA <- minimaxApprox(exp, 0, 0.5, 5L)
apErr <- abs(exp(x) - minimaxEval(x, mmA))
all.equal(max(apErr), mmA$ExpErr)

# Plot results
curve(exp, 0.0, 0.5, lwd = 2)
curve(minimaxEval(x, mmA), 0.0, 0.5, add = TRUE, col = "red", lty = 2L, lwd = 2)
```

---

plot.minimaxApprox *Plot errors from a* "minimaxApprox" *object*

Description

Produces a plot of the error of the "minimaxApprox" object, highlighting the error extrema and bounds.

Usage

```
## S3 method for class 'minimaxApprox'
plot(x, y, ...)
```

Arguments

x	An object inheriting from class "minimaxApprox".
y	Ignored. In call as required by R in Writing R Extensions:chapter 7.
...	Further arguments to plot. Specifically to pass ylim to allow for zooming in or out.

Value

No return value; called for side effects.

Author(s)

Avraham Adler <Avraham.Adler@gmail.com>

See Also

minimaxApprox

Examples

```
PP <- minimaxApprox(exp, 0, 1, 5)
plot(PP)
```

`print.minimaxApprox` *Print method for a* `"minimaxApprox object"`

Description

Provides a more human-readable output of a `"minimaxApprox"` object.

Usage

```
## S3 method for class 'minimaxApprox'
print(x, digits = 14L, ...)
```

Arguments

`x`	An object inheriting from class `"minimaxApprox"`.
`digits`	integer; Number of digits to which to round the ratio.
`...`	Further arguments to `print`.

Details

To print the raw `"minimaxApprox"` object use `print.default`.

Value

No return value; called for side effects.

Author(s)

Avraham Adler <Avraham.Adler@gmail.com>

See Also

`minimaxApprox`

Examples

```
PP <- minimaxApprox(sin, 0, 1, 8)
PP
print(PP, digits = 2L)
print.default(PP)
```

Index

∗ **NumericalMathematics**
  coef.minimaxApprox, 3
  minimaxApprox, 4
  minimaxErr, 11
  minimaxEval, 12
  plot.minimaxApprox, 13
  print.minimaxApprox, 14
∗ **hplot**
  plot.minimaxApprox, 13
∗ **methods**
  coef.minimaxApprox, 3
  plot.minimaxApprox, 13
  print.minimaxApprox, 14
∗ **optimize**
  minimaxApprox, 4
∗ **package**
  minimaxApprox-package, 2
∗ **print**
  print.minimaxApprox, 14

class, *3*, *8*, *13, 14*
coef.minimaxApprox, 3

list, *3*, *5*, *8*

minimaxApprox, *3*, 4, *11–14*
minimaxApprox-package, 2
minimaxErr, *6*, *9, 10*, 11, *12*
minimaxEval, *6*, *10, 11*, 12

plot.minimaxApprox, 13
print.minimaxApprox, 14