Package ‘jacobi’

November 19, 2023

Type Package

Title Jacobi Theta Functions and Related Functions

Version 3.1.1

Description Evaluation of the Jacobi theta functions and related functions: Weierstrass elliptic function, Weierstrass sigma function, Weierstrass zeta function, Klein j-function, Dedekind eta function, lambda modular function, Jacobi elliptic functions, Neville theta functions, Eisenstein series, lemniscate elliptic functions, elliptic alpha function, Rogers-Ramanujan continued fractions, and Dixon elliptic functions. Complex values of the variable are supported.

License GPL-3

URL https://github.com/stla/jacobi

BugReports https://github.com/stla/jacobi/issues

Imports Carlson, Rcpp (>= 1.0.8), rgl, Rvcg

Suggests testthat (>= 3.0.0), elliptic, RcppColors

LinkingTo Rcpp

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation yes

Author Stéphane Laurent [aut, cre],
Mikael Fremling [aut] (author of the original Fortran code for the theta functions)

Maintainer Stéphane Laurent <laurent_step@outlook.fr>

Repository CRAN

Date/Publication 2023-11-18 23:50:03 UTC

R topics documented:

`agm` *Arithmetic-geometric mean*

Description

Evaluation of the arithmetic-geometric mean of two complex numbers.

Usage

```
agm(x, y)
```

Arguments

`x, y`	complex numbers

Value

A complex number, the arithmetic-geometric mean of `x` and `y`.

Examples

```
agm(1, sqrt(2))
2*pi^(3/2)*sqrt(2) / gamma(1/4)^2
```

`am` *Amplitude function*

Description

Evaluation of the amplitude function.

Usage

```
am(u, m)
```

Arguments

`u`	complex number
`m`	square of elliptic modulus, a complex number

Value

A complex number.

Examples

```
library(Carlson)
phi <- 1 + 1i
m <- 2
u <- elliptic_F(phi, m)
am(u, m) # should be phi
```

CostaMesh — *Costa surface*

Description

Computes a mesh of the Costa surface.

Usage

```
CostaMesh(nu = 50L, nv = 50L)
```

Arguments

`nu, nv`	numbers of subdivisions

Value

A triangle **rgl** mesh (object of class `mesh3d`).

Examples

```
library(jacobi)
library(rgl)

mesh <- CostaMesh(nu = 250, nv = 250)
open3d(windowRect = c(50, 50, 562, 562), zoom = 0.9)
bg3d("#15191E")
shade3d(mesh, color = "darkred", back = "cull")
shade3d(mesh, color = "orange", front = "cull")
```

disk2H — *Disk to upper half-plane*

Description

Conformal map from the unit disk to the upper half-plane. The function is vectorized.

Usage

```
disk2H(z)
```

Arguments

`z`	a complex number in the unit disk

Value

A complex number in the upper half-plane.

Examples

```
# map the disk to H and calculate kleinj
f <- function(x, y) {
  z <- complex(real = x, imaginary = y)
  K <- rep(NA_complex_, length(x))
  inDisk <- Mod(z) < 1
  K[inDisk] <- kleinj(disk2H(z[inDisk]))
  K
}
n <- 1024L
x <- y <- seq(-1, 1, length.out = n)
Grid <- expand.grid(X = x, Y = y)
K <- f(Grid$X, Grid$Y)
dim(K) <- c(n, n)
# plot
if(require("RcppColors")) {
  img <- colorMap5(K)
} else {
  img <- as.raster(1 - abs(Im(K))/Mod(K))
}
opar <- par(mar = c(0, 0, 0, 0))
plot(NULL, xlim = c(0, 1), ylim = c(0, 1), asp = 1,
     axes = FALSE, xaxs = "i", yaxs = "i", xlab = NA, ylab = NA)
rasterImage(img, 0, 0, 1, 1)
par(opar)
```

`disk2square` *Disk to square*

Description

Conformal map from the unit disk to the square $[-1, 1] \times [-1, 1]$. The function is vectorized.

Usage

```
disk2square(z)
```

Arguments

`z`	a complex number in the unit disk

Value

A complex number in the square $[-1, 1] \times [-1, 1]$.

Examples

```
n <- 70L
r <- seq(0, 1, length.out = n)
theta <- seq(0, 2*pi, length.out = n+1L)[-1L]
Grid <- transform(
  expand.grid(R = r, Theta = theta),
  Z = R*exp(1i*Theta)
)
s <- vapply(Grid$Z, disk2square, complex(1L))
plot(Re(s), Im(s), pch = ".", asp = 1, cex = 2)
#
# a more insightful plot ####
r_ <- seq(0, 1, length.out = 10L)
theta_ <- seq(0, 2*pi, length.out = 33)[-1L]
plot(
  NULL, xlim = c(-1, 1), ylim = c(-1, 1), asp = 1, xlab = "x", ylab = "y"
)
for(r in r_) {
  theta <- sort(
    c(seq(0, 2, length.out = 200L), c(1/4, 3/4, 5/4, 7/4))
  )
  z <- r*(cospi(theta) + 1i*sinpi(theta))
  s <- vapply(z, disk2square, complex(1L))
  lines(Re(s), Im(s), col = "blue", lwd = 2)
}
for(theta in theta_) {
  r <- seq(0, 1, length.out = 30L)
  z <- r*exp(1i*theta)
  s <- vapply(z, disk2square, complex(1L))
  lines(Re(s), Im(s), col = "green", lwd = 2)
}
```

Dixon — *Dixon elliptic functions*

Description

The Dixon elliptic functions.

Usage

```
sm(z)

cm(z)
```

Arguments

z	a real or complex number

Value

A complex number.

Examples

```
# cubic Fermat curve x^3+y^3=1
pi3 <- beta(1/3, 1/3)
epsilon <- 0.7
t_ <- seq(-pi3/3 + epsilon, 2*pi3/3 - epsilon, length.out = 100)
pts <- t(vapply(t_, function(t) {
  c(Re(cm(t)), Re(sm(t)))
}, FUN.VALUE = numeric(2L)))
plot(pts, type = "l", asp = 1)
```

`EisensteinE` *Eisenstein series*

Description

Evaluation of Eisenstein series with weight 2, 4 or 6.

Usage

```
EisensteinE(n, q)
```

Arguments

`n`	the weight, can be 2, 4 or 6
`q`	nome, complex number with modulus smaller than one

Value

A complex number, the value of the Eisenstein series.

`ellipticAlpha` *Elliptic alpha function*

Description

Evaluates the elliptic alpha function.

Usage

```
ellipticAlpha(z)
```

Arguments

`z`	a complex number

Value

A complex number.

References

Weisstein, Eric W. "Elliptic Alpha Function".

`ellipticInvariants` *Elliptic invariants*

Description

Elliptic invariants from half-periods.

Usage

```
ellipticInvariants(omega1omega2)
```

Arguments

`omega1omega2`	the half-periods, a vector of two complex numbers

Value

The elliptic invariants, a vector of two complex numbers.

`eta` *Dedekind eta function*

Description

Evaluation of the Dedekind eta function.

Usage

```
eta(tau)
```

Arguments

`tau`	a vector of complex numbers with strictly positive imaginary parts

Value

A vector of complex numbers.

Examples

```
eta(2i)
gamma(1/4) / 2^(11/8) / pi^(3/4)
```

`halfPeriods` *Half-periods*

Description

Half-periods from elliptic invariants.

Usage

```
halfPeriods(g2g3)
```

Arguments

`g2g3`	the elliptic invariants, a vector of two complex numbers

Value

The half-periods, a vector of two complex numbers.

`jellip` *Jacobi elliptic functions*

Description

Evaluation of the Jacobi elliptic functions.

Usage

```
jellip(kind, u, tau = NULL, m = NULL)
```

Arguments

`kind`	a string with two characters among `"s"`, `"c"`, `"d"` and `"n"`; this string specifies the function: the two letters respectively denote the basic functions sn, cn, dn and 1, and the string specifies the ratio of two such functions, e.g. $ns = 1/sn$ and $cd = cn/dn$
`u`	a complex number, vector or matrix
`tau`	complex number with strictly positive imaginary part; it is related to `m` and only one of them must be supplied
`m`	the "parameter", square of the elliptic modulus; it is related to `tau` and only one of them must be supplied

Value

A complex number, vector or matrix.

Examples

```
u <- 2 + 2i
tau <- 1i
jellip("cn", u, tau)^2 + jellip("sn", u, tau)^2 # should be 1
```

`jtheta1` *Jacobi theta function one*

Description

Evaluates the first Jacobi theta function.

Usage

```
jtheta1(z, tau = NULL, q = NULL)

ljtheta1(z, tau = NULL, q = NULL)
```

Arguments

`z`	complex number, vector, or matrix
`tau`	lattice parameter, a complex number with strictly positive imaginary part; the two complex numbers `tau` and `q` are related by `q = exp(1i*pi*tau)`, and only one of them must be supplied
`q`	the nome, a complex number whose modulus is strictly less than one, but not zero

Value

A complex number, vector or matrix; `jtheta1` evaluates the first Jacobi theta function and `ljtheta1` evaluates its logarithm.

Examples

```
jtheta1(1 + 1i, q = exp(-pi/2))
```

`jtheta2` *Jacobi theta function two*

Description

Evaluates the second Jacobi theta function.

Usage

```
jtheta2(z, tau = NULL, q = NULL)

ljtheta2(z, tau = NULL, q = NULL)
```

Arguments

`z`	complex number, vector, or matrix
`tau`	lattice parameter, a complex number with strictly positive imaginary part; the two complex numbers `tau` and `q` are related by `q = exp(1i*pi*tau)`, and only one of them must be supplied
`q`	the nome, a complex number whose modulus is strictly less than one, but not zero

Value

A complex number, vector or matrix; `jtheta2` evaluates the second Jacobi theta function and `ljtheta2` evaluates its logarithm.

Examples

```
jtheta2(1 + 1i, q = exp(-pi/2))
```

`jtheta3` *Jacobi theta function three*

Description

Evaluates the third Jacobi theta function.

Usage

```
jtheta3(z, tau = NULL, q = NULL)

ljtheta3(z, tau = NULL, q = NULL)
```

Arguments

`z`	complex number, vector, or matrix
`tau`	lattice parameter, a complex number with strictly positive imaginary part; the two complex numbers `tau` and `q` are related by `q = exp(1i*pi*tau)`, and only one of them must be supplied
`q`	the nome, a complex number whose modulus is strictly less than one, but not zero

Value

A complex number, vector or matrix; `jtheta3` evaluates the third Jacobi theta function and `ljtheta3` evaluates its logarithm.

Examples

```
jtheta3(1 + 1i, q = exp(-pi/2))
```

`jtheta4` *Jacobi theta function four*

Description

Evaluates the fourth Jacobi theta function.

Usage

```
jtheta4(z, tau = NULL, q = NULL)

ljtheta4(z, tau = NULL, q = NULL)
```

Arguments

`z`	complex number, vector, or matrix
`tau`	lattice parameter, a complex number with strictly positive imaginary part; the two complex numbers `tau` and `q` are related by `q = exp(1i*pi*tau)`, and only one of them must be supplied
`q`	the nome, a complex number whose modulus is strictly less than one, but not zero

Value

A complex number, vector or matrix; `jtheta4` evaluates the fourth Jacobi theta function and `ljtheta4` evaluates its logarithm.

Examples

```
jtheta4(1 + 1i, q = exp(-pi/2))
```

`jtheta_ab` *Jacobi theta function with characteristics*

Description

Evaluates the Jacobi theta function with characteristics.

Usage

```
jtheta_ab(a, b, z, tau = NULL, q = NULL)
```

Arguments

`a, b`	the characteristics, two complex numbers
`z`	complex number, vector, or matrix
`tau`	lattice parameter, a complex number with strictly positive imaginary part; the two complex numbers `tau` and `q` are related by `q = exp(1i*pi*tau)`, and only one of them must be supplied
`q`	the nome, a complex number whose modulus is strictly less than one, but not zero

Details

The Jacobi theta function with characteristics generalizes the four Jacobi theta functions. It is denoted by $\theta[a,b](z|\tau)$. One gets the four Jacobi theta functions when `a` and `b` take the values `0` or `0.5`:

if `a=b=0.5` then one gets $\vartheta_1(z|\tau)$

if `a=0.5` **and** `b=0` then one gets $\vartheta_2(z|\tau)$

if `a=b=0` then one gets $\vartheta_3(z|\tau)$

if `a=0` **and** `b=0.5` then one gets $\vartheta_4(z|\tau)$

Both $\theta[a,b](z+\pi|\tau)$ and $\theta[a,b](z+\pi\tau|\tau)$ are equal to $\theta[a,b](z|\tau)$ up to a factor - see the examples for the details.

Value

A complex number, vector or matrix, like `z`.

Note

Different conventions are used in the book cited as reference.

References

Hershel M. Farkas, Irwin Kra. *Theta Constants, Riemann Surfaces and the Modular Group: An Introduction with Applications to Uniformization Theorems, Partition Identities and Combinatorial Number Theory*. Graduate Studies in Mathematics, volume 37, 2001.

Examples

```
a   <- 2 + 0.3i
b   <- 1 - 0.6i
z   <- 0.1 + 0.4i
tau <- 0.2 + 0.3i
jab <- jtheta_ab(a, b, z, tau)
# first property ####
jtheta_ab(a, b, z + pi, tau) # is equal to:
jab * exp(2i*pi*a)
# second property ####
jtheta_ab(a, b, z + pi*tau, tau) # is equal to:
jab * exp(-1i*(pi*tau + 2*z + 2*pi*b))
```

`kleinj` *Klein j-function and its inverse*

Description

Evaluation of the Klein j-invariant function and its inverse.

Usage

```
kleinj(tau, transfo = FALSE)

kleinjinv(j)
```

Arguments

`tau`	a complex number with strictly positive imaginary part, or a vector or matrix of such complex numbers; missing values allowed
`transfo`	Boolean, whether to use a transformation of the values of `tau` close to the real line; using this option can fix some failures of the computation (at the cost of speed), e.g. when the algorithm reaches the maximal number of iterations
`j`	a complex number

Value

A complex number, vector or matrix.

Note

The Klein-j function is the one with the factor 1728.

Examples

```
( j <- kleinj(2i) )
66^3
kleinjinv(j)
```

`lambda` *Lambda modular function*

Description

Evaluation of the lambda modular function.

Usage

```
lambda(tau, transfo = FALSE)
```

Arguments

`tau`	a complex number with strictly positive imaginary part, or a vector or matrix of such complex numbers; missing values allowed
`transfo`	Boolean, whether to use a transformation of the values of `tau` close to the real line; using this option can fix some failures of the computation (at the cost of speed), e.g. when the algorithm reaches the maximal number of iterations

Value

A complex number, vector or matrix.

Note

The lambda function is the square of the elliptic modulus.

Examples

```
x <- 2
lambda(1i*sqrt(x)) + lambda(1i*sqrt(1/x)) # should be one
```

`lemniscate` *Lemniscate functions*

Description

Lemniscate sine, cosine, arcsine, arccosine, hyperbolic sine, and hyperbolic cosine functions.

Usage

```
sl(z)

cl(z)

asl(z)

acl(z)

slh(z)

clh(z)
```

Arguments

`z` a real number or a complex number

Value

A complex number.

Examples

```
sl(1+1i) * cl(1+1i) # should be 1
## | the lemniscate ####
# lemniscate parameterization
p <- Vectorize(function(s) {
  a <- Re(cl(s))
  b <- Re(sl(s))
  c(a, a * b) / sqrt(1 + b*b)
})
# lemnniscate constant
ombar <- 2.622 # gamma(1/4)^2 / (2 * sqrt(2*pi))
# plot
s_ <- seq(0, ombar, length.out = 100)
lemniscate <- t(p(s_))
plot(lemniscate, type = "l", col = "blue", lwd = 3)
lines(cbind(lemniscate[, 1L], -lemniscate[, 2L]), col="red", lwd = 3)
```

---

nome *Nome*

Description

The nome in function of the parameter m.

Usage

```
nome(m)
```

Arguments

`m`	the parameter, square of elliptic modulus, real or complex number

Value

A complex number.

Examples

```
nome(-2)
```

`RR` *Rogers-Ramanujan continued fraction*

Description

Evaluates the Rogers-Ramanujan continued fraction.

Usage

```
RR(q)
```

Arguments

`q`	the nome, a complex number whose modulus is strictly less than one, and which is not zero

Value

A complex number

Note

This function is sometimes denoted by R.

`RRa` *Alternating Rogers-Ramanujan continued fraction*

Description

Evaluates the alternating Rogers-Ramanujan continued fraction.

Usage

```
RRa(q)
```

Arguments

`q`	the nome, a complex number whose modulus is strictly less than one, and which is not zero

Value

A complex number

Note

This function is sometimes denoted by S.

`square2disk` *Square to disk*

Description

Conformal map from the unit square to the unit disk. The function is vectorized.

Usage

```
square2disk(z)
```

Arguments

`z`	a complex number in the unit square $[0, 1] \times [0, 1]$

Value

A complex number in the unit disk.

Examples

```
x <- y <- seq(0, 1, length.out = 25L)
Grid <- transform(
  expand.grid(X = x, Y = y),
  Z = complex(real = X, imaginary = Y)
)
u <- square2disk(Grid$Z)
plot(u, pch = 19, asp = 1)
```

---

square2H *Square to upper half-plane*

Description

Conformal map from the unit square to the upper half-plane. The function is vectorized.

Usage

```
square2H(z)
```

Arguments

z	a complex number in the unit square $[0, 1] \times [0, 1]$

Value

A complex number in the upper half-plane.

Examples

```
n <- 1024L
x <- y <- seq(0.0001, 0.9999, length.out = n)
Grid <- transform(
  expand.grid(X = x, Y = y),
  Z = complex(real = X, imaginary = Y)
)
K <- kleinj(square2H(Grid$Z))
dim(K) <- c(n, n)
# plot
if(require("RcppColors")) {
  img <- colorMap5(K)
} else {
  img <- as.raster((Arg(K) + pi)/(2*pi))
}
opar <- par(mar = c(0, 0, 0, 0))
plot(NULL, xlim = c(0, 1), ylim = c(0, 1), asp = 1,
     axes = FALSE, xaxs = "i", yaxs = "i", xlab = NA, ylab = NA)
rasterImage(img, 0, 0, 1, 1)
par(opar)
```

theta.s	*Neville theta functions*

Description

Evaluation of the Neville theta functions.

Usage

```
theta.s(z, tau = NULL, m = NULL)

theta.c(z, tau = NULL, m = NULL)

theta.n(z, tau = NULL, m = NULL)

theta.d(z, tau = NULL, m = NULL)
```

Arguments

z	a complex number, vector, or matrix
tau	complex number with strictly positive imaginary part; it is related to `m` and only one of them must be supplied
m	the "parameter", square of the elliptic modulus; it is related to `tau` and only one of them must be supplied

Value

A complex number, vector or matrix.

wp	*Weierstrass elliptic function*

Description

Evaluation of the Weierstrass elliptic function and its derivatives.

Usage

```
wp(z, g = NULL, omega = NULL, tau = NULL, derivative = 0L)
```

Arguments

`z`	complex number, vector or matrix
`g`	the elliptic invariants, a vector of two complex numbers; only one of `g`, `omega` and `tau` must be given
`omega`	the half-periods, a vector of two complex numbers; only one of `g`, `omega` and `tau` must be given
`tau`	the half-periods ratio; supplying `tau` is equivalent to supply `omega = c(1/2, tau/2)`
`derivative`	differentiation order, an integer between 0 and 3

Value

A complex number, vector or matrix.

Examples

```
omega1 <- 1.4 - 1i
omega2 <- 1.6 + 0.5i
omega <- c(omega1, omega2)
e1 <- wp(omega1, omega = omega)
e2 <- wp(omega2, omega = omega)
e3 <- wp(-omega1-omega2, omega = omega)
e1 + e2 + e3 # should be 0
```

`wpinv` *Inverse of Weierstrass elliptic function*

Description

Evaluation of the inverse of the Weierstrass elliptic function.

Usage

```
wpinv(w, g = NULL, omega = NULL, tau = NULL)
```

Arguments

`w`	complex number
`g`	the elliptic invariants, a vector of two complex numbers; only one of `g`, `omega` and `tau` must be given
`omega`	the half-periods, a vector of two complex numbers; only one of `g`, `omega` and `tau` must be given
`tau`	the half-periods ratio; supplying `tau` is equivalent to supply `omega = c(1/2, tau/2)`

Value

A complex number.

Examples

```
library(jacobi)
omega <- c(1.4 - 1i, 1.6 + 0.5i)
w <- 1 + 1i
z <- wpinv(w, omega = omega)
wp(z, omega = omega) # should be w
```

`wsigma` *Weierstrass sigma function*

Description

Evaluation of the Weierstrass sigma function.

Usage

```
wsigma(z, g = NULL, omega = NULL, tau = NULL)
```

Arguments

`z`	a complex number, vector or matrix
`g`	the elliptic invariants, a vector of two complex numbers; only one of `g`, `omega` and `tau` must be given
`omega`	the half-periods, a vector of two complex numbers; only one of `g`, `omega` and `tau` must be given
`tau`	the half-periods ratio; supplying `tau` is equivalent to supply `omega = c(1/2, tau/2)`

Value

A complex number, vector or matrix.

Examples

```
wsigma(1, g = c(12, -8))
# should be equal to:
sin(1i*sqrt(3))/(1i*sqrt(3)) / sqrt(exp(1))
```

`wzeta` *Weierstrass zeta function*

Description

Evaluation of the Weierstrass zeta function.

Usage

```
wzeta(z, g = NULL, omega = NULL, tau = NULL)
```

Arguments

z	complex number, vector or matrix
g	the elliptic invariants, a vector of two complex numbers; only one of g, omega and tau must be given
omega	the half-periods, a vector of two complex numbers; only one of g, omega and tau must be given
tau	the half-periods ratio; supplying tau is equivalent to supply omega = c(1/2, tau/2)

Value

A complex number, vector or matrix.

Examples

```
# Mirror symmetry property:
z <- 1 + 1i
g <- c(1i, 1+2i)
wzeta(Conj(z), Conj(g))
Conj(wzeta(z, g))
```

Index