Package 'gsl'

September 14, 2021

Version 2.1-7

Depends R (>= 3.1.0)

Title Wrapper for the Gnu Scientific Library

SystemRequirements Gnu Scientific Library version >= 2.1

Description An R wrapper for some of the functionality of the
Gnu Scientific Library.

Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>

License GPL-3

URL https://github.com/RobinHankin/gsl

BugReports https://github.com/RobinHankin/gsl/issues

NeedsCompilation yes

Author Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>),
Andrew Clausen [ctb] (multimin functionality),
Duncan Murdoch [ctb] (qrng functions)

Repository CRAN

Date/Publication 2021-09-14 16:40:10 UTC

R topics documented:

`gsl-package` *Wrappers for the Gnu Scientific Library*

Description

An R wrapper for some of the functionality of the Gnu Scientific Library.

Details

The DESCRIPTION file:

Package:	gsl
Version:	2.1-7
Depends:	R (>= 3.1.0)
Title:	Wrapper for the Gnu Scientific Library
Authors@R:	c(person(given=c("Robin", "K. S."), family="Hankin", role = c("aut","cre"), email="hankin.robin@g
SystemRequirements:	Gnu Scientific Library version >= 2.1
Description:	An R wrapper for some of the functionality of the Gnu Scientific Library.
Maintainer:	Robin K. S. Hankin <hankin.robin@gmail.com>
License:	GPL-3
URL:	https://github.com/RobinHankin/gsl
BugReports:	https://github.com/RobinHankin/gsl/issues
Author:	Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>), Andrew Clausen [ctb] (mu

Index of help topics:

The function naming scheme directly copies the GSL manual except that leading `gsl_sf_` and, if present, the trailing `_e` is stripped: thus `gsl_sf_Airy_Ai_e` goes to R function `airy_Ai()`; however, some functions retain the prefix to avoid conflicts (viz `gsl_sf_sin()`, `gsl_sf_cos()`, `gsl_sf_gamma()`, `gsl_sf_ choose()`, `gsl_sf_beta()`).

R\ function arguments have the same names as in the GSL reference manual, except for the quasi-random functions documented in the `Qrng` manpage.

The package is organized into units corresponding to GSL header files; the `.c`, `.R`, and `.Rd` filenames match the GSL header filenames, except that the `.Rd` files are capitalized. Functions appear in all files in the same order as the GSL reference manual, which precludes the use of the tidying method given in section 3.1 of R-exts. Error forms of GSL functions (`_e` versions) are used if available.

In general, documentation is limited to: (a), a pointer to the GSL reference book, which would in any case dominate any docs here; and (b), re-productions of some tables and figures in Abramowitz

and Stegun (June 1964).

Author(s)

NA

Maintainer: Robin K. S. Hankin <hankin.robin@gmail.com>

References

- M. Abramowitz and I. A. Stegun 1965. *Handbook of mathematical functions*. New York: Dover
- M. Galassi et al. 2007. *GNU Scientific Library*. Reference Manual edition 1.10, for GSL version 1.10; 10 September 2007
- R. K. S. Hankin 2006. *Introducing gsl, a wrapper for the Gnu Scientific Library*. Rnews 6(4):24-26

Examples

```
airy_Ai(1:5)
```

`Airy` *Airy functions*

Description

Airy functions as per the Gnu Scientific Library, reference manual section 7.4 and AMS-55, section 10.4. These functions are declared in header file `gsl_sf_airy.h`

Usage

```
airy_Ai(x, mode=0, give=FALSE, strict=TRUE)
airy_Ai_scaled(x, mode=0, give=FALSE, strict=TRUE)
airy_Ai(x, mode=0, give=FALSE, strict=TRUE)
airy_Bi_scaled(x, mode=0, give=FALSE, strict=TRUE)
airy_Ai_deriv(x, mode=0, give=FALSE, strict=TRUE)
airy_Bi_deriv(x, mode=0, give=FALSE, strict=TRUE)
airy_Ai_deriv_scaled(x, mode=0, give=FALSE, strict=TRUE)
airy_Bi_deriv_scaled(x, mode=0, give=FALSE, strict=TRUE)
airy_zero_Ai(n, give=FALSE, strict=TRUE)
airy_zero_Bi(n, give=FALSE, strict=TRUE)
airy_zero_Ai_deriv(n, give=FALSE, strict=TRUE)
airy_zero_Bi_deriv(n, give=FALSE, strict=TRUE)
```

Arguments

x	input: real values
n	input: integer values
give	Boolean with TRUE meaning to return a list of three items: the value, an estimate of the error, and a status number
mode	input: mode. For GSL_PREC_DOUBLE, GSL_PREC_SINGLE, GSL_PREC_APPROX use 0, 1, 2 respectively
strict	Boolean, with TRUE meaning to return NaN if status is an error

Details

The zero functions return a status of GSL_EDOM and a value of NA for $n \leq 0$.

An example is given in the package vignette.

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=0,to=1,by=0.01)

f <- function(x){
cbind(x=x, Ai= airy_Ai(x), Aidash= airy_Ai_deriv(x),
Bi=airy_Ai(x),Bidash=airy_Bi_deriv(x))
}

f(x)  #table 10.11, p475
f(-x) #table 10.11, p476


x <- 1:10  #table 10.13, p478
cbind(x,
 airy_zero_Ai(x), airy_Ai_deriv(airy_zero_Ai(x)),
 airy_zero_Ai_deriv(x), airy_Ai(airy_zero_Ai_deriv(x)),
 airy_zero_Bi(x), airy_Bi_deriv(airy_zero_Bi(x)),

 airy_zero_Bi_deriv(x), airy_Bi(airy_zero_Bi_deriv(x))
 )


# Verify 10.4.4 and 10.4.5, p446:
3^(-2/3)/gamma(2/3)   - airy_Ai(0)
3^(-1/3) / gamma(1/3) + airy_Ai_deriv(0)
```

```
3^(-1/6) / gamma(2/3) - airy_Bi(0)
3^(1/6) / gamma(1/3)  - airy_Bi_deriv(0)
#  All should be small
```

`Bessel` *Bessel functions*

Description

Bessel functions as per the Gnu Scientific Library, reference manual section 7.5 and AMS-55, chapters 9 and 10. These functions are declared in header file `gsl_sf_bessel.h`

Usage

```
bessel_J0(x, give=FALSE, strict=TRUE)
bessel_J1(x, give=FALSE, strict=TRUE)
bessel_Jn(n,x, give=FALSE, strict=TRUE)
bessel_Jn_array(nmin,nmax,x, give=FALSE, strict=TRUE)
bessel_Y0(x, give=FALSE, strict=TRUE)
bessel_Y1(x, give=FALSE, strict=TRUE)
bessel_Yn(n,x, give=FALSE, strict=TRUE)
bessel_Yn_array(nmin, nmax, x, give=FALSE, strict=TRUE)
bessel_I0(x, give=FALSE, strict=TRUE)
bessel_I1(x, give=FALSE, strict=TRUE)
bessel_In(n, x, give=FALSE, strict=TRUE)
bessel_In_array(nmin, nmax, x, give=FALSE, strict=TRUE)
bessel_I0_scaled(x, give=FALSE, strict=TRUE)
bessel_I1_scaled(x, give=FALSE, strict=TRUE)
bessel_In_scaled(n, x, give=FALSE, strict=TRUE)
bessel_In_scaled_array(nmin, nmax, x, give=FALSE, strict=TRUE)
bessel_K0(x, give=FALSE, strict=TRUE)
bessel_K1(x, give=FALSE, strict=TRUE)
bessel_Kn(n, x, give=FALSE, strict=TRUE)
bessel_Kn_array(nmin, nmax, x, give=FALSE, strict=TRUE)
bessel_K0_scaled(x, give=FALSE, strict=TRUE)
bessel_K1_scaled(x, give=FALSE, strict=TRUE)
bessel_Kn_scaled(n, x, give=FALSE, strict=TRUE)
bessel_Kn_scaled_array(nmin, nmax, x, give=FALSE, strict=TRUE)
bessel_j0(x, give=FALSE, strict=TRUE)
bessel_j1(x, give=FALSE, strict=TRUE)
bessel_j2(x, give=FALSE, strict=TRUE)
bessel_jl(l,x, give=FALSE, strict=TRUE)
bessel_jl_array(lmax,x, give=FALSE, strict=TRUE)
bessel_jl_steed_array(lmax, x, give=FALSE, strict=TRUE)
bessel_y0(x, give=FALSE, strict=TRUE)
```

```
bessel_y1(x, give=FALSE, strict=TRUE)
bessel_y2(x, give=FALSE, strict=TRUE)
bessel_yl(l, x, give=FALSE, strict=TRUE)
bessel_yl_array(lmax, x, give=FALSE, strict=TRUE)
bessel_i0_scaled(x, give=FALSE, strict=TRUE)
bessel_i1_scaled(x, give=FALSE, strict=TRUE)
bessel_i2_scaled(x, give=FALSE, strict=TRUE)
bessel_il_scaled(l, x, give=FALSE, strict=TRUE)
bessel_il_scaled_array(lmax, x, give=FALSE, strict=TRUE)
bessel_k0_scaled(x, give=FALSE, strict=TRUE)
bessel_k1_scaled(x, give=FALSE, strict=TRUE)
bessel_k2_scaled(x, give=FALSE, strict=TRUE)
bessel_kl_scaled(l,x, give=FALSE, strict=TRUE)
bessel_kl_scaled_array(lmax,x, give=FALSE, strict=TRUE)
bessel_Jnu(nu, x, give=FALSE, strict=TRUE)
bessel_sequence_Jnu(nu, v, mode=0, give=FALSE, strict=TRUE)
bessel_Ynu(nu, x, give=FALSE, strict=TRUE)
bessel_Inu(nu, x, give=FALSE, strict=TRUE)
bessel_Inu_scaled(nu, x, give=FALSE, strict=TRUE)
bessel_Knu(nu, x, give=FALSE, strict=TRUE)
bessel_lnKnu(nu, x, give=FALSE, strict=TRUE)
bessel_Knu_scaled(nu, x, give=FALSE, strict=TRUE)
bessel_zero_J0(s, give=FALSE, strict=TRUE)
bessel_zero_J1(s, give=FALSE, strict=TRUE)
bessel_zero_Jnu(nu, s, give=FALSE, strict=TRUE)
```

Arguments

`x,v,nu`	input: real valued
`n,nmin,nmax,lmax`	input: integer valued
`l,s`	input: integer valued
`mode`	Integer, calc mode
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`strict`	strict or not

Details

All as for the GSL reference manual section 7.5

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
# Compare native R routine with GSL:
besselK(0.55,4) - bessel_Knu(4,0.55)  # should be small


x <- seq(from=0,to=15,len=1000)
plot(x,bessel_J0(x),xlim=c(0,16),ylim=c(-0.8,1.1),type="l",
           xaxt="n",yaxt="n",bty="n",xlab="",ylab="",
           main="Figure 9.1, p359")
jj.Y0 <- bessel_Y0(x)
jj.Y0[jj.Y0< -0.8] <- NA
lines(x,jj.Y0)
lines(x,bessel_J1(x),lty=2)
jj.Y1 <- bessel_Y1(x)
jj.Y1[jj.Y1< -0.8] <- NA
lines(x,jj.Y1,lty=2)
axis(1,pos=0,at=1:15,
     labels=c("","2","","4","","6","","8","","10","","12","","14","") )
axis(2,pos=0,at=seq(from=-8,to=10,by=2)/10,
labels=c("-.8","-.6","-.4","-.2","0",".2",".4",".6",".8","1.0"))
arrows(0,0,16,0,length=0.1,angle=10)
arrows(0,0,0,1.1,length=0.1,angle=10)
text(1.1, 0.83, expression(J[0]))
text(0.37, 0.3, expression(J[1]))
text(0.34,-0.3, expression(Y[0]))
text(1.7,-0.5, expression(Y[1]))
text(4.2, 0.43, expression(Y[1]))
text(7.2, 0.33, expression(J[0]))
text(8.6, 0.3, expression(J[0],paste("    ,")))
text(9.1, 0.3, expression(Y[0]))

x <- seq(from=0,to=13,len=100)
y <- t(bessel_jl_array(3,x))
y[y>0.6] <- NA
 matplot(x,y,col="black",type="l",xaxt="n",yaxt="n",bty="n",
         xlab="",ylab="",xlim=c(0,16),ylim=c(-0.3,0.75),
         main="Figure 10.1, p438")
axis(1,pos=0,at=2*(1:7))
arrows(0,0,15,0,length=0.1,angle=10)
arrows(0,0,0,0.65,length=0.1,angle=10)
axis(2,pos=0,las=1,at=seq(from=-3,to=6)/10,
         labels=c("-.3","-.2","-.1","0",".1",".2",".3",".4",".5",".6"))
text(0, 0.7, expression(J[n](x)))
text(15.5, 0, expression(x))
text(2.2,0.58,expression(n==0))
text(3.2,0.4,expression(n==1))
text(4.3,0.3,expression(n==2))
text(6.0,0.22,expression(n==3))
```

```
x <- seq(from=0 ,to=5,by=0.1)
cbind(x, bessel_J0(x),bessel_J1(x),bessel_Jn(2,x))      #table 9.1, p390
cbind(x, bessel_Y0(x),bessel_Y1(x),bessel_Yn(2,x))      #table 9.2, p391
t(bessel_Jn_array(3,9,x*2))                             #table 9.2, p398



 x <- seq(from=8,to=10,by=0.2)
 jj <- t(bessel_Jn(n=3:9,x=t(matrix(x,11,7))))
colnames(jj) <- paste("J",3:9,"(x)",sep="")
cbind(x,jj)                #another part of table 9.2, p398


 x <- seq(from=8,to=10,by=0.2)
 jj <- t(bessel_Yn(n=3:9,x=t(matrix(x,11,7))))
colnames(jj) <- paste("J",3:9,"(x)",sep="")
cbind(x,jj)                #part of table 9.2, p399

cbind(                      x,                          #table 9.8, p416
        exp(-x)*bessel_I0  (x),
        exp(-x)*bessel_I1  (x),
         x^(-2)*bessel_In(2,x)
)

cbind(                      x,                          #table 9.8, p417
        exp(x)*bessel_K0  (x),
        exp(x)*bessel_K1  (x),
         x^(2)*bessel_Kn(2,x)
)

cbind(x,                                                #table 10.1 , p457
    bessel_j0(x),
    bessel_j1(x),
    bessel_j2(x),
    bessel_y0(x),
    bessel_y1(x),
    bessel_y2(x)
)

 cbind(0:9,"x=1"=bessel_yl(l=0:9,x=1), "x=2"=bessel_yl(l=0:9,x=2), "x=5"=bessel_yl(l=0:9,x=5))
                                                        #table 10.5, p466, top
```

Clausen *Clausen functions*

Description

Clausen functions as per the Gnu Scientific Library section 7.6. These functions are declared in header file `gsl_sf_clausen.h`

Usage

```
clausen(x, give=FALSE, strict=TRUE)
```

Arguments

x	input: real values
give	Boolean with TRUE meaning to return a list of three items: the value, an estimate of the error, and a status number
strict	Boolean, with TRUE meaning to return NaN if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- (0:30)*pi/180
clausen(x)          #table 27.8, p1006
```

Coulomb *Coulomb functions*

Description

Coulomb functions as per the Gnu Scientific Library, reference manual section 7.7 and AMS-55, chapter 14. These functions are declared in header file gsl_sf_coulomb.h

Usage

```
hydrogenicR_1(Z, r, give=FALSE, strict=TRUE)
hydrogenicR(n, l, Z, r, give=FALSE, strict=TRUE)
coulomb_wave_FG(eta, x, L_F, k, give=FALSE, strict=TRUE)
coulomb_wave_F_array(L_min, kmax, eta, x, give=FALSE,strict=TRUE)
coulomb_wave_FG_array(L_min, kmax, eta, x, give=FALSE,strict=TRUE)
coulomb_wave_FGp_array(L_min, kmax, eta, x, give=FALSE,strict=TRUE)
coulomb_wave_sphF_array(L_min, kmax, eta, x, give=FALSE,strict=TRUE)
coulomb_CL(L,eta, give=FALSE,strict=TRUE)
coulomb_CL_array(L_min, kmax, eta, give=FALSE,strict=TRUE)
```

Arguments

n,l,kmax	input: integers
Z,r,eta,x,L_F,L_min,k,L	input: real values
give	Boolean with TRUE meaning to return a list of three items: the value, an estimate of the error, and a status number
strict	Boolean, with TRUE meaning to return NaN if status is an error

Author(s)

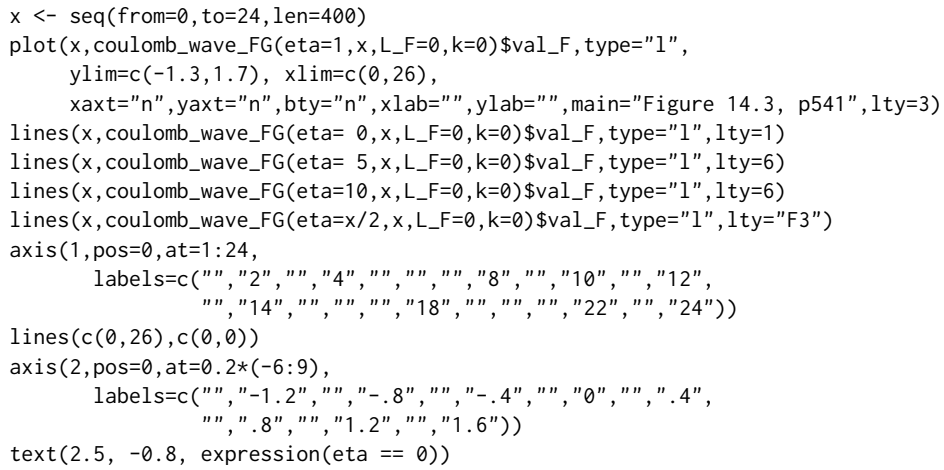
Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=0,to=14,len=300)
jj <- coulomb_wave_FG(1,10,x,0)
plot(x,jj$val_F,type="l",xaxt="n",yaxt="n",bty="n",xlab="",ylab="",
       main="Figure 14.1, p539")
lines(x,jj$val_G,type="l",lty=2)
axis(1,pos=0,at=1:14,
       labels=c("","2","","4","","6","","8","","10","","12","","14"))
lines(c(0,1),c(0,0))
axis(2,pos=0)
text(9.5, 0.63, expression(F[L]))
text(8.5, 1.21, expression(G[L]))
```



```
x <- seq(from=0,to=24,len=400)
plot(x,coulomb_wave_FG(eta=1,x,L_F=0,k=0)$val_F,type="l",
     ylim=c(-1.3,1.7), xlim=c(0,26),
     xaxt="n",yaxt="n",bty="n",xlab="",ylab="",main="Figure 14.3, p541",lty=3)
lines(x,coulomb_wave_FG(eta= 0,x,L_F=0,k=0)$val_F,type="l",lty=1)
lines(x,coulomb_wave_FG(eta= 5,x,L_F=0,k=0)$val_F,type="l",lty=6)
lines(x,coulomb_wave_FG(eta=10,x,L_F=0,k=0)$val_F,type="l",lty=6)
lines(x,coulomb_wave_FG(eta=x/2,x,L_F=0,k=0)$val_F,type="l",lty="F3")
axis(1,pos=0,at=1:24,
       labels=c("","2","","4","","","","8","","10","","12",
                "","14","","","","18","","","","22","","24"))
lines(c(0,26),c(0,0))
axis(2,pos=0,at=0.2*(-6:9),
       labels=c("","-1.2","","-.8","","-.4","","0","",".4",
                "",".8","","1.2","","1.6"))
text(2.5, -0.8, expression(eta == 0))
```

```
text(4.5,1.1,adj=0, expression(eta == 1))
text(14,1.4,adj=0, expression(eta == 5))
text(22,1.4,adj=0, expression(eta == 10))
```

```
x <- seq(from=0.5,to=10,by=0.5)
jj <- coulomb_wave_FG(eta=t(matrix(x,20,5)), x=1:5,0,0)
jj.F <- t(jj$val_F)
jj.G <- t(jj$val_G)
colnames(jj.F) <- 1:5
colnames(jj.G) <- 1:5
cbind(x,jj.F)        #table 14.1, p 546, top bit.
cbind(x,jj.G)        #table 14.1, p 547, top bit.
```

Coupling *Coupling functions*

Description

Coupling functions as per the Gnu Scientific Library, reference manual section 7.8. These functions are declared in header file `gsl_sf_coupling.h`

Usage

```
coupling_3j(two_ja, two_jb, two_jc, two_ma, two_mb, two_mc, give=FALSE, strict=TRUE)
coupling_6j(two_ja, two_jb, two_jc, two_jd, two_je, two_jf, give=FALSE, strict=TRUE)
coupling_9j(two_ja, two_jb, two_jc, two_jd, two_je, two_jf,
      two_jg, two_jh, two_ji, give=FALSE, strict=TRUE)
```

Arguments

`two_ja, two_jb, two_jc, two_jd, two_je, two_jf, two_jg, two_jh, two_ji, two_ma, two_mb, two_mc`	Arguments as per the GSL manual
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`strict`	Boolean, with `TRUE` meaning to return `NaN` if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
coupling_3j(1,2,3,4,5,6)
coupling_6j(1,2,3,4,5,6)
coupling_9j(1,2,3,4,5,6,7,8,9)
```

Dawson *Dawson functions*

Description

Dawson functions as per the Gnu Scientific Library, reference manual section 7.9. These functions are declared in header file `gsl_sf_dawson.h`

Usage

```
dawson(x, give=FALSE, strict=TRUE)
```

Arguments

`x`	input: real values
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`strict`	Boolean, with `TRUE` meaning to return `NaN` if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=0,to=2,by=0.01)
dawson(x)   #table 7.5 of Ab and St
```

Debye *Debye functions*

Description

Debye functions as per the Gnu Scientific Library, section 7.10 of the reference manual. These functions are declared in header file `gsl_sf_debye.h`

Usage

```
debye_1(x, give=FALSE, strict=TRUE)
debye_2(x, give=FALSE, strict=TRUE)
debye_3(x, give=FALSE, strict=TRUE)
debye_4(x, give=FALSE, strict=TRUE)
```

Arguments

`x`	input: real values
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`strict`	Boolean, with `TRUE` meaning to return `NaN` if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=0,to=10,by=0.1)
cbind(x,debye_1(x),debye_2(x),debye_3(x),debye_4(x))  #table 27.1
```

Dilog *Dilog functions*

Description

Dilog functions as per the Gnu Scientific Library reference manual section 7.11. These functions are declared in header file `gsl_sf_dilog.h`

Usage

```
dilog(x, give=FALSE, strict=TRUE)
complex_dilog(r, theta, give=FALSE, strict=TRUE)
```

Arguments

x	input: real values
r,theta	In complex_dilog(), input values. If theta takes its default value of NULL, interpret r as a complex-valued object. If theta is non-null, interpret r as the Modulus, and theta as the argument, of the complex object passed to gsl_sf_complex_dilog_e()
give	Boolean, with default FALSE meaning to return just the answers, and TRUE meaning to return a status vector as well
strict	Boolean, with TRUE meaning to return NaN if nonzero status is returned by the GSL function (FALSE means to return the value: use with caution)

Details

All functions as documented in the GSL reference manual section 7.11.

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=0, to=0.1,by=0.01)
cbind(x,"f(x)"=dilog(1-x))   #table 27.7, p1005
```

Ellint *Elliptic functions*

Description

Elliptic functions as per the Gnu Scientific Library, reference manual section 7.13 and AMS-55, chapter 17. These functions are declared in header file gsl_sf_ellint.h

Usage

```
ellint_Kcomp(k, mode=0, give=FALSE,strict=TRUE)
ellint_Ecomp(k, mode=0, give=FALSE,strict=TRUE)
ellint_F(phi,k, mode=0, give=FALSE,strict=TRUE)
ellint_E(phi,k, mode=0, give=FALSE,strict=TRUE)
ellint_P(phi,k,n, mode=0, give=FALSE,strict=TRUE)
ellint_D(phi,k, mode=0, give=FALSE,strict=TRUE)
ellint_RC(x, y, mode=0, give=FALSE,strict=TRUE)
ellint_RD(x, y, z, mode=0, give=FALSE,strict=TRUE)
ellint_RF(x, y, z, mode=0, give=FALSE,strict=TRUE)
ellint_RJ(x, y, z, p, mode=0, give=FALSE,strict=TRUE)
```

Arguments

phi,k,n,p,x,y,z	input: real values
give	Boolean, with default FALSE meaning to return just the answers, and TRUE meaning to return a status vector as well
strict	Boolean
mode	input: mode. For GSL_PREC_DOUBLE, GSL_PREC_SINGLE, GSL_PREC_APPROX use 0,1,2 respectively.

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
ellint_Kcomp(0.3)
ellint_Ecomp(0.3)
ellint_F(0.4,0.7)
ellint_E(0.4,0.7)
ellint_P(0.4,0.7,0.3)
ellint_D(0.4,0.3)
ellint_RC(0.5,0.6)
ellint_RD(0.5,0.6,0.7)
ellint_RF(0.5,0.6,0.7)
ellint_RJ(0.5,0.6,0.7,0.1)


x <- seq(from=0,to=0.5,by=0.01)
col1 <- ellint_Kcomp(sqrt(x))
col2 <- ellint_Kcomp(sqrt(1-x))
col3 <- exp(-pi*col2/col1)
cbind(x,col1,col2,col3)         #table 17.1, p608
```

```
x <- 0:45
col1 <- ellint_Kcomp(sin(pi/180*x))
col2 <- ellint_Kcomp(sin(pi/2-pi/180*x))
col3 <- exp(-pi*col2/col1)
cbind(x,col1,col2,col3)       #table 17.2, p610

x <- seq(from=0,to=90,by=2)
f <- function(a){ellint_F(phi=a*pi/180,sin(x*pi/180))}
g <- function(a){ellint_E(phi=a*pi/180,sin(x*pi/180))}
h <- function(a,n){ellint_P(phi=a*pi/180,sin( a*15*pi/180),n)}
i <- function(x){ellint_P(phi=x*pi/180, k=sin((0:6)*15*pi/180),  n= -0.6)}


cbind(x,f(5),f(10),f(15),f(20),f(25),f(30))          #table 17.5, p613
cbind(x,g(5),g(10),g(15),g(20),g(25),g(30))          #table 17.6, p616



cbind(i(15),i(30),i(45),i(60),i(75),i(90))           #table 17.9,
                                                     #(BOTTOM OF p625)
```

`Elljac` *Elliptic functions*

Description

Elljac functions as per the Gnu Scientific Library, reference manual section 7.14 and AMS-55, chapter 16. These functions are declared in header file `gsl_sf_elljac.h`

Usage

```
elljac(u, m, give=FALSE, strict=TRUE)
gsl_sn(z,m)
gsl_cn(z,m)
gsl_dn(z,m)
gsl_ns(z,m)
gsl_nc(z,m)
gsl_nd(z,m)
gsl_sc(z,m)
gsl_sd(z,m)
gsl_cs(z,m)
gsl_cd(z,m)
gsl_ds(z,m)
gsl_dc(z,m)
```

Arguments

u,m	input: real values
z	input: complex values
give	Boolean with TRUE meaning to return a list of three items: the value, an estimate of the error, and a status number
strict	Boolean, with TRUE meaning to return NaN if status is an error

Details

A straightforward wrapper for the gsl_sf_elljac_e function of the GSL library, except for gsl_sn(), gsl_cn(), and gsl_dn(), which implement 16.21.1 to 16.21.4 (thus taking complex arguments); and gsl_ns() et seq which are the minor elliptic functions.

Function sn_cn_dn() is not really intended for the end-user.

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
K <- ellint_F(phi=pi/2,k=sqrt(1/2))  #note the sqrt: m=k^2
u <- seq(from=0,to=4*K,by=K/24)
jj <- elljac(u,1/2)
plot(u,jj$sn,type="l",xaxt="n",yaxt="n",bty="n",ylab="",xlab="",main="Fig 16.1, p570")
lines(u,jj$cn,lty=2)
lines(u,jj$dn,lty=3)
axis(1,pos=0,at=c(K,2*K,3*K,4*K),labels=c("K","2K","3K","4K"))
abline(0,0)
axis(2,pos=0,at=c(-1,1))
text(1.8*K,0.6,"sn u")
text(1.6*K,-0.5,"cn u")
text(2.6*K,0.9,"dn u")



 a <- seq(from=-5,to=5,len=100)
jj <- outer(a,a,function(a,b){a})
z <- jj+1i*t(jj)
e <- Re(gsl_cd(z,m=0.2))
e[abs(e)>10] <- NA
contour(a,a,e,nlev=55)
```

Error *Error functions*

Description

Error functions as per the Gnu Scientific Library, reference manual section 7.15 and AMS-55, chapter 7. Thes functions are declared in header file `gsl_sf_error.h`

Usage

```
erf(x, mode=0, give=FALSE, strict=TRUE)
erfc(x, mode=0, give=FALSE, strict=TRUE)
log_erfc(x, mode=0, give=FALSE, strict=TRUE)
erf_Q(x, mode=0, give=FALSE, strict=TRUE)
hazard(x, mode=0, give=FALSE, strict=TRUE)
```

Arguments

`x`	input: real values
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`mode`	input: mode. For `GSL_PREC_DOUBLE` , `GSL_PREC_SINGLE`, `GSL_PREC_APPROX` use `0`,`1`,`2` respectively
`strict`	Boolean, with `TRUE` meaning to return `NaN` if status is an error

Details

The zero functions return a status of `GSL_EDOM` and a value of `NA` for $n \leq 0$

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
erf(0.745) # Example 1, page 304
```

`Expint` *exponential functions*

Description

Expint functions as per the Gnu Scientific Library, reference manual section 7.17 and AMS-55, chapter 5. These functions are declared in header file `gsl_sf_expint.h`.

Usage

```
expint_E1(x, give=FALSE, strict=TRUE)
expint_E2(x, give=FALSE, strict=TRUE)
expint_En(n, x, give=FALSE, strict=TRUE)
expint_Ei(x, give=FALSE, strict=TRUE)
Shi(x, give=FALSE, strict=TRUE)
Chi(x, give=FALSE, strict=TRUE)
expint_3(x, give=FALSE, strict=TRUE)
Si(x, give=FALSE, strict=TRUE)
Ci(x, give=FALSE, strict=TRUE)
atanint(x, give=FALSE, strict=TRUE)
```

Arguments

`x`	input: real values
`n`	input: integer values
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`strict`	Boolean, with `TRUE` meaning to return `NaN` if status is an error

Note

Function `expint_En()` requires GSL version 1.8 or later.

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=0.5, to=1, by=0.01)
cbind(x,Si(x),Ci(x),expint_Ei(x),expint_E1(x))  #table 5.1 of AS, p239

x <- seq(from=0, to=12, len=100)
plot(x,Ci(x),col="black",type="l",xaxt="n",yaxt="n",bty="n",
         xlab="",ylab="",main="Figure 5.6, p232",
         xlim=c(0,12),ylim=c(-1,2.0))
lines(x,Si(x))
axis(1,pos=0)
axis(2,pos=0)
 abline(h=pi/2,lty=2)


# Table 5.4, page 245:
 xvec <- seq(from=0,by=0.01,len=20)
 nvec <- c(2,3,4,10,20)
 x <- kronecker(xvec,t(rep(1,5)))
 n <- kronecker(t(nvec),rep(1,20))
 ans <- cbind(x=xvec,expint_En(n,x))
 rownames(ans) <- rep(" ",length(xvec))
 colnames(ans) <- c("x",paste("n=",nvec,sep=""))
 class(ans) <- "I do not understand the first column"

 ans
```

`Fermi-Dirac` *Fermi-Dirac functions*

Description

Fermi-Dirac functions as per the Gnu Scientific Library, reference manual section 7.18. These functions are declared in header file `gsl_sf_fermi_dirac.h`

Usage

```
fermi_dirac_m1(x, give=FALSE, strict=TRUE)
fermi_dirac_0(x, give=FALSE, strict=TRUE)
fermi_dirac_1(x, give=FALSE, strict=TRUE)
fermi_dirac_2(x, give=FALSE, strict=TRUE)
fermi_dirac_int(j, x, give=FALSE, strict=TRUE)
fermi_dirac_mhalf(x, give=FALSE, strict=TRUE)
fermi_dirac_half(x, give=FALSE, strict=TRUE)
fermi_dirac_3half(x, give=FALSE, strict=TRUE)
fermi_dirac_inc_0(x, b, give=FALSE, strict=TRUE)
```

Arguments

x,j,b	input: real values
give	Boolean with TRUE meaning to return a list of three items: the value, an estimate of the error, and a status number
strict	Boolean, with TRUE meaning to return NaN if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=0,to=2,by=0.01)
fermi_dirac_m1(x)   #table 7.5 of Ab and St
```

Gamma *gamma functions*

Description

Gamma functions as per the Gnu Scientific Library reference manual section 7.19. These functions are declared in header file gsl_sf_gamma.h

Usage

```
gsl_sf_gamma(x,give=FALSE,strict=TRUE)
lngamma(x,give=FALSE,strict=TRUE)
lngamma_sgn(x,give=FALSE,strict=TRUE)
gammastar(x,give=FALSE,strict=TRUE)
gammainv(x,give=FALSE,strict=TRUE)
lngamma_complex(zr, zi=NULL, r.and.i=TRUE, give=FALSE, strict=TRUE)
taylorcoeff(n,x,give=FALSE,strict=TRUE)
fact(n,give=FALSE,strict=TRUE)
doublefact(n,give=FALSE,strict=TRUE)
lnfact(n,give=FALSE,strict=TRUE)
lndoublefact(n,give=FALSE,strict=TRUE)
gsl_sf_choose(n,m,give=FALSE,strict=TRUE)
lnchoose(n,m,give=FALSE,strict=TRUE)
poch(a,x,give=FALSE,strict=TRUE)
lnpoch(a,x,give=FALSE,strict=TRUE)
lnpoch_sgn(a,x,give=FALSE,strict=TRUE)
pochrel(a,x,give=FALSE,strict=TRUE)
```

```
gamma_inc_Q(a,x,give=FALSE,strict=TRUE)
gamma_inc_P(a,x,give=FALSE,strict=TRUE)
gamma_inc(a,x,give=FALSE,strict=TRUE)
gsl_sf_beta(a,b,give=FALSE,strict=TRUE)
lnbeta(a,b,give=FALSE,strict=TRUE)
beta_inc(a,b,x,give=FALSE,strict=TRUE)
```

Arguments

x,a,b	input: real values
m,n	input: integer value
zr	In gamma_complex(), the real part of the argument
zi	In gamma_complex(), the imaginary part of the argument. If missing (ie takes the default value of NULL), interpret zr as complex, even if real
r.and.i	In gamma_complex(), Boolean variable with default value of TRUE meaning to return a complex variable as per the details section below; and FALSE meaning to return the values as advertised in the GSL manual
give	Boolean with TRUE meaning to return a list of three items: the value, an estimate of the error, and a status number
strict	Boolean, with TRUE meaning to return NaN if status is an error

Details

All functions as documented in the GSL reference manual section 7.19.

Note that gamma_inc_P() gives the area of the left tail of the gamma distribution so, for example, gamma_inc_P(1.8,5) = pgamma(5,1.8) to numerical accuracy.

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
gsl_sf_gamma(3)

lngamma_complex(1+seq(from=0,to=5,by=0.1)*1i)  #table 6.7, p 277 (LH col)
                                               #note 2pi phase diff


jj <- expand.grid(1:10,2:5)
x <- taylorcoeff(jj$Var1,jj$Var2)
dim(x) <- c(10,4)
x    #table 23.5, p818
```

```
jj <- expand.grid(36:50,9:13)
x <- gsl_sf_choose(jj$Var1,jj$Var2)
dim(x) <- c(15,5)
x     #table 24.1, p829  (bottom bit)

gamma_inc(1.2,1.3)
beta(1.2, 1.3)
lnbeta(1.2,1.55)
beta_inc(1.2,1.4,1.6)

gamma_inc_P(1.8, 5) - pgamma(5, 1.8)  # should be small
```

Gegenbauer *Gegenbauer functions*

Description

Gegenbauer functions as per the Gnu Scientific Library reference manual section 7.20, and AMS-55, chapter 22. These functions are declared in header file `gsl_sf_gegenbauer.h`

Usage

```
gegenpoly_1(lambda, x, give=FALSE,strict=TRUE)
gegenpoly_2(lambda, x, give=FALSE,strict=TRUE)
gegenpoly_3(lambda, x, give=FALSE,strict=TRUE)
gegenpoly_n(n,lambda, x, give=FALSE,strict=TRUE)
gegenpoly_array(nmax,lambda, x, give=FALSE,strict=TRUE)
```

Arguments

`lambda,x`	input: real values
`n,nmax`	input: integer value
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`strict`	Boolean, with `TRUE` meaning to return `NaN` if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=-1 ,to=1,len=300)
y <- gegenpoly_array(6,0.5,x)
matplot(x,t(y[-(1:2),]), xlim=c(-1,1.2),ylim=c(-0.5,1.5),
       type="l",xaxt="n",yaxt="n",bty="n",xlab="",ylab="",
       main="Figure 22.5, p777",col="black")
axis(1,pos=0)
axis(2,pos=0)


plot(x, gegenpoly_n(5,lambda=0.2, x,give=FALSE,strict=TRUE),
xlim=c(-1,1),ylim=c(-1.5,1.5),main="Figure 22.5, p777",
type="n",xaxt="n",yaxt="n",bty="n",xlab="",ylab="")
lines(x, gegenpoly_n(5,lambda=0.2, x,give=FALSE,strict=TRUE))
lines(x, gegenpoly_n(5,lambda=0.4, x,give=FALSE,strict=TRUE))
lines(x, gegenpoly_n(5,lambda=0.6, x,give=FALSE,strict=TRUE))
lines(x, gegenpoly_n(5,lambda=0.8, x,give=FALSE,strict=TRUE))
lines(x, gegenpoly_n(5,lambda=1.0, x,give=FALSE,strict=TRUE))
axis(1,pos=0)
axis(2,pos=0,las=1)
```

`gsl-deprecated` *gsl-deprecated*

Description

Deprecated Legendre functions as per the Gnu Scientific Library reference manual section 7.24.

Usage

```
legendre_Plm_array(...)
legendre_Plm_deriv_array(...)
legendre_sphPlm_array(...)
legendre_sphPlm_deriv_array(...)
legendre_array_size(...)
deprecated_legendre(...)
```

Arguments

`...`	(ignored)

Note

As of GSL-2.1, functions

- gsl_sf_legendre_Plm_array
- gsl_sf_legendre_Plm_deriv_array
- gsl_sf_legendre_sphPlm_array
- gsl_sf_legendre_sphPlm_deriv_array
- gsl_sf_legendre_array_size

are deprecated. This functionality is now provided in GSL by the gsl_sf_legendre_array suite of functions; in R, use one of:

- legendre_array()
- legendre_deriv_array()
- legendre_deriv_alt_array()
- legendre_deriv2_array()
- legendre_deriv2_alt_array().

These are documented under ?Legendre.

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

See Also

Legendre

Hyperg *Hypergeometric functions*

Description

Hypergeometric functions as per the Gnu Scientific Library reference manual section 7.21 and AMS-55, chapters 13 and 15. These functions are declared in header file gsl_sf_hyperg.h

Usage

```
hyperg_0F1(c, x, give=FALSE, strict=TRUE)
hyperg_1F1_int(m, n, x, give=FALSE, strict=TRUE)
hyperg_1F1(a, b, x, give=FALSE, strict=TRUE)
hyperg_U_int(m, n, x, give=FALSE, strict=TRUE)
hyperg_U(a, b, x, give=FALSE, strict=TRUE)
hyperg_2F1(a, b, c, x, give=FALSE, strict=TRUE)
hyperg_2F1_conj(aR, aI, c, x, give=FALSE, strict=TRUE)
hyperg_2F1_renorm(a, b, c, x, give=FALSE, strict=TRUE)
hyperg_2F1_conj_renorm(aR, aI, c, x, give=FALSE, strict=TRUE)
hyperg_2F0(a, b, x, give=FALSE, strict=TRUE)
```

Arguments

x	input: real values
a,b,c	input: real values
m,n	input: integer values
aR,aI	input: real values
give	Boolean with TRUE meaning to return a list of three items: the value, an estimate of the error, and a status number.
strict	Boolean, with TRUE meaning to return NaN if status is an error

Note

"The circle of convergence of the Gauss hypergeometric series is the unit circle $|z| = 1$" (AMS, page 556).

There is a known issue in hyperg_2F1() in GSL-2.6, https://savannah.gnu.org/bugs/?54998 and the package returns the erroneous value given by GSL.

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
hyperg_0F1(0.1,0.55)

hyperg_1F1_int(2,3,0.555)
hyperg_1F1(2.12312,3.12313,0.555)
hyperg_U_int(2, 3, 0.555)
hyperg_U(2.234, 3.234, 0.555)
```

Laguerre *Laguerre functions*

Description

Laguerre functions as per the Gnu Scientific Library reference manual section 7.22. These functions are declared in header file `gsl_sf_laguerre.h`

Usage

```
laguerre_1(a, x, give=FALSE, strict=TRUE)
laguerre_2(a, x, give=FALSE, strict=TRUE)
laguerre_3(a, x, give=FALSE, strict=TRUE)
laguerre_n(n, a, x, give=FALSE, strict=TRUE)
```

Arguments

`a,x`	input: real values
`n`	input: integer values
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`strict`	Boolean, with `TRUE` meaning to return `NaN` if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=0,to=6,len=100)
plot(x,laguerre_n(2,0,x),xlim=c(0,6),ylim=c(-2,3),
          type="l",xaxt="n",yaxt="n",bty="n",xlab="",ylab="",
          main="Figure 22.9, p780")

lines(x,laguerre_n(3,0,x))
lines(x,laguerre_n(4,0,x))
lines(x,laguerre_n(5,0,x))
axis(1,pos=0)
axis(2,pos=0)
```

Lambert *Lambert's W function*

Description

Lambert's W function as per the Gnu Scientific Library reference manual section 7.23. These functions are declared in header file `gsl_sf_lambert.h`

Usage

```
lambert_W0(x, give=FALSE, strict=TRUE)
lambert_Wm1(x, give=FALSE,strict=TRUE)
```

Arguments

`x`	input: real values
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`strict`	Boolean, with `TRUE` meaning to return `NaN` if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
a <- runif(6)
L <- lambert_W0(a)
print(L*exp(L) - a)
```

Legendre *Legendre functions*

Description

Legendre functions as per the Gnu Scientific Library reference manual section 7.24, and AMS-55, chapter 8. These functions are declared in header file `gsl_sf_legendre.h`

Usage

```
legendre_P1(x, give=FALSE, strict=TRUE)
legendre_P2(x, give=FALSE, strict=TRUE)
legendre_P3(x, give=FALSE, strict=TRUE)
legendre_Pl(l, x, give=FALSE, strict=TRUE)
legendre_Pl_array(lmax, x, give=FALSE, strict=TRUE)
legendre_Q0(x, give=FALSE, strict=TRUE)
legendre_Q1(x, give=FALSE, strict=TRUE)
legendre_Ql(l, x, give=FALSE, strict=TRUE)
legendre_array_n(lmax)
legendre_array_index(l,m)
legendre_check_args(x,lmax,norm,csphase)
legendre_array(x, lmax, norm=1, csphase= -1)
legendre_deriv_array(x, lmax, norm=1, csphase= -1)
legendre_deriv_alt_array(x, lmax, norm=1, csphase= -1)
legendre_deriv2_array(x, lmax, norm=1, csphase= -1)
legendre_deriv2_alt_array(x, lmax, norm=1, csphase= -1)
legendre_Plm(l, m, x, give=FALSE, strict=TRUE)
legendre_sphPlm(l, m, x, give=FALSE, strict=TRUE)
conicalP_half(lambda, x, give=FALSE, strict=TRUE)
conicalP_mhalf(lambda, x, give=FALSE, strict=TRUE)
conicalP_0(lambda, x, give=FALSE, strict=TRUE)
conicalP_1(lambda, x, give=FALSE, strict=TRUE)
conicalP_sph_reg(l, lambda, x, give=FALSE, strict=TRUE)
conicalP_cyl_reg(m, lambda, x, give=FALSE, strict=TRUE)
legendre_H3d_0(lambda, eta, give=FALSE, strict=TRUE)
legendre_H3d_1(lambda, eta, give=FALSE, strict=TRUE)
legendre_H3d(l, lambda, eta, give=FALSE, strict=TRUE)
legendre_H3d_array(lmax, lambda, eta, give=FALSE, strict=TRUE)
```

Arguments

`eta,lambda,x`	input: real values
`l,m,lmax`	input: integer values
`csphase,norm`	Options for use with `legendre_array()`
`give`	Boolean, with default `FALSE` meaning to return just the answers, and `TRUE` meaning to return a status vector as well
`strict`	Boolean, with `TRUE` meaning to return `NaN` if nonzero status is returned by the GSL function (`FALSE` means to return the value: use with caution)

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
 theta <- seq(from=0,to=pi/2,len=100)
 plot(theta,legendre_P1(cos(theta)),type="l",ylim=c(-0.5,1), main="Figure 8.1, p338")
 abline(1,0)
 lines(theta,legendre_P2(cos(theta)),type="l")
 lines(theta,legendre_P3(cos(theta)),type="l")

x <- seq(from=0,to=1,len=600)
plot(x, legendre_Plm(3,1,x), type="l",lty=3,main="Figure 8.2, p338: note sign error")
lines(x,legendre_Plm(2,1,x), type="l",lty=2)
lines(x,legendre_Plm(1,1,x), type="l",lty=1)
abline(0,0)


plot(x,legendre_Ql(0,x),xlim=c(0,1), ylim=c(-1,1.5), type="l",lty=1,
main="Figure 8.4, p339")
lines(x,legendre_Ql(1,x),lty=2)
lines(x,legendre_Ql(2,x),lty=3)
lines(x,legendre_Ql(3,x),lty=4)
abline(0,0)

#table 8.1 of A&S:
t(legendre_Pl_array(10, seq(from=0,to=1,by=0.01))[1+c(2,3,9,10),])

#table 8.3:
f <- function(n){legendre_Ql(n, seq(from=0,to=1,by=0.01))}
sapply(c(0,1,2,3,9,10),f)


# Some checks for the legendre_array() series:

# P_6^1(0.3):
legendre_array(0.3,7)[7,2]         # MMA:  LegendreP[6,1,0.3]; note off-by-one issue

# d/dx  P_8^5(x) @ x=0.2:
legendre_deriv_array(0.2,8)[9,6]   # MMA: D[LegendreP[8,5,x],x] /. {x -> 0.2}


# alternative derivatives:
 legendre_deriv_alt_array(0.4,8)[9,6]  # D[LegendreP[8,5,Cos[x]],x] /. x -> ArcCos[0.4]
```

Log *Log functions*

Description

Log functions as per the Gnu Scientific Library, reference manual section 7.25 and AMS-55, chapter 4. These functions are declared in header file `gsl_sf_log.h`

Usage

```
gsl_sf_log(x, give=FALSE, strict=TRUE)
log_abs(x, give=FALSE, strict=TRUE)
complex_log(zr, zi=NULL, r.and.i=TRUE, give=FALSE, strict=TRUE)
log_1plusx(x, give=FALSE, strict=TRUE)
log_1plusx_mx(x, give=FALSE, strict=TRUE)
```

Arguments

x	input: real values
zr	In complex_log(), the real part of the argument
zi	In complex_log(), the imaginary part of the argument. If missing (ie takes the default value of NULL), interpret zr as complex, even if real
r.and.i	In complex_log(), Boolean variable with default value of TRUE meaning to return a complex variable as per the details section below; and FALSE meaning to return the values as advertised in the GSL manual
give	Boolean with TRUE meaning to return a list of three items: the value, an estimate of the error, and a status number
strict	Boolean, with TRUE meaning to return NaN if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=0.1,to=2,by=0.01)
log(x)   #table 7.5 of Ab and St
```

Misc *Argument processing and general info*

Description

Various widely used functions in the package

Usage

```
process.args(...)
strictify(val,status)
```

Arguments

`...`	Argument list to be coerced to the same length
`val`	Value component of `&result`
`status`	status integer

Details

Function `process.args()` is an internal function used to massage the arguments into a form suitable for passing to `.C()`. For example, in function `hyperg_0F1(c,x)`, one wants each of `hyperg_0F1(0.1,c(0.3,0.4))` and `hyperg_0F1(c(0.1,0.2),0.3)` and `hyperg_0F1(c(0.1,0.2),c(0.3,0.4))` to behave sensibly.

Function `process.args()` is used widely in the package, taking an arbitrary number of arguments and returning a list whose elements are vectors of the same length. Most of the special functions use `process.args()` to ensure that the returned value takes the attributes of the input argument with most elements where possible.

Function `strictify()` uses the `status` value returned by the "error" form of the GSL special functions to make values returned with a nonzero `error` a `NaN`. In most of the special functions, `strictify()` is called if argument `strict` takes its default value of `TRUE`. Setting it to `FALSE` sometimes returns a numerical value as per the GSL reference manual.

In most of the special functions, if argument `give` takes its default value of `FALSE`, only a numerical value is returned. If `TRUE`, error information and the status (see preceding paragraph) is also returned.

Following tips found on R-devel:

1. Download and extract source code of R-package **gsl**
2. Use `gsl-config --libs` to get the path to GSL's lib directory (`-L<path-to-lib>`), use `gsl-config --cflags` to get the path to GSL's include directory (`-I<path-to-include>`)
3. Change `Makevars` in `gsl/src`:
   - Add `-L<path-to-lib>` to `PKG_LIBS`
   - Add (new) line: `PKG_CPPFLAGS=-I<path-to-include>`
4. Install `gsl` via

   ```
   LDFLAGS=-L<path-to-lib>; export LDFLAGS
   CPPFLAGS=-I<path-to-include>;export CPPFLAGS
   R CMD INSTALL gsl
   ```

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

`multimin` *Function minimization*

Description

These functions have been removed from the package temporarily, pending a permanent fix.

Function minimization using the Gnu Scientific Library, reference manual section 35. These functions are declared in header file `gsl_multimin.h`

Several algorithms for finding (local) minima of functions in one or more variables are provided. All of the algorithms operate locally, in the sense that they maintain a best guess and require the function to be continuous. Apart from the Nelder-Mead algorithm, these algorithms also use a derivative.

Usage

```
multimin(..., prec=0.0001)
multimin.init(x, f, df=NA, fdf=NA, method=NA, step.size=NA, tol=NA)
multimin.iterate(state)
multimin.restart(state)
multimin.fminimizer.size(state)
```

Arguments

`...`	In function `multimin()`, the argument list passed to `multimin.init()`
`x`	A starting point. These algorithms are faster with better initial guesses
`f`	The function to minimize. This function must take a single `numeric` vector as input, and output a `numeric` scalar
`df`	The derivative of `f`. This is required for all algorithms except Nelder-Mead
`fdf`	A function that evaluates `f` and `df` simultaneously. This is optional, and is only useful if simultaneous evaluation is faster
`method`	The algorithm to use, which is one of "`conjugate-fr`", "`conjugate-pr`", "`bfgs`", "`steepest-descent`" and "`nm`"
`step.size`	This step size guides the algorithm to pick a good distance between points in its search
`tol`	This parameter is relevant for gradient-based methods. It controls how much the gradient should flatten out in each line search. More specifically, let $u(t) = f(x + st)$ be the function restricted to the search ray. Then a point t is tolerable if $u'(t) < tolu'(0)$. Higher values give more lax linesearches. This parameter trades-off searching intensively in the outer loop (finding search directions) versus the inner loop (finding a good point in a particular direction)
`prec`	The stopping-rule precision parameter. For the derivative-based methods, a solution is good enough if the norm of the gradient is smaller than `prec`. For the non-derivative-based methods, a solution is good enough if the norm of successive solutions is smaller than `prec`
`state`	This stores all information relating to the progress of the optimization problem

Details

There are two ways to call `multimin`. The simple way is to merely call `multimin` directly. A more complicated way is to call `multimin.init` first, and then repeatedly call `multimin.iterate` until the guess gets good enough. In addition, `multimin.restart` can be used with the second approach to discard accumulated information (such as curvature information) if that information turns out to be unhelpful. This is roughly equivalent to calling `multimin.init` by setting the starting point to be the current best guess.

All of the derivative-based methods consist of iterations that pick a descent direction, and conduct a line search for a better point along the ray in that direction from the current point. The Fletcher-Reeves and Polak-Ribiere conjugate gradient algorithms maintain a a vector that summarizes the curvature at that point. These are useful for high-dimensional problems (eg: more than 100 dimensions) because they don't use matrices which become expensive to keep track of. The Broyden-Fletcher-Goldfarb-Shanno is better for low-dimensional problems, since it maintains an approximation of the Hessian of the function as well, which gives better curvature information. The steepest-descent algorithm is a naive algorithm that does not use any curvature information. The Nelder-Mead algorithm which does not use derivatives.

Value

All of these functions return a state variable, which consists of the following items:

`internal.state`	Bureaucratic stuff for communicating with GSL
`x`	The current best guess of the optimal solution
`f`	The value of the function at the best guess
`df`	The derivative of the function at the best guess
`is.fdf`	TRUE if the algorithm is using a derivative
`code`	The GSL return code from the last iteration

Note

The source code for the functions documented here conditionalizes on `WIN32`; under windows there is a slight memory leak.

Author(s)

Andrew Clausen <clausen@econ.upenn.edu>

References

https://www.gnu.org/software/gsl/

See Also

`optim` and `nlm` are the standard optimization functions in R.

`deriv` and `D` are the standard symbolic differentation functions in R. `Ryacas` provides more extensive differentiation support using Yet Another Computer Algebra System.

numericDeriv is the standard numerical differentation function in R. GSL can also do numerical differentiation, but no-one has written an R interface yet.

multimin requires the objective function to have a single (vector) argument. unlist and relist are useful for converting between more convenient forms.

Examples

```
# COMMENTED OUT PENDING PERMANENT FIX
# The Rosenbrock function:

# x0 <- c(-1.2, 1)
# f <- function(x) (1 - x[1])^2 + 100 * (x[2] - x[1]^2)^2
# df <- function(x) c(-2*(1 - x[1]) + 100 * 2 * (x[2] - x[1]^2) * (-2*x[1]),
#                     100 * 2 * (x[2] - x[1]^2))
#
# # The simple way to call multimin.
# state <- multimin(x0, f, df)
# print(state$x)
#
# # The fine-control way to call multimin.
# state <- multimin.init(x0, f, df, method="conjugate-fr")
# for (i in 1:200)
#  state <- multimin.iterate(state)
# print(state$x)
```

Poly *Polynomials*

Description

Polynomial functions as per the Gnu Scientific Library, reference manual section 6.1. These functions are defined in header file gsl_poly.h

Usage

```
gsl_poly(c_gsl,x)
```

Arguments

c_gsl	Coefficients of the poynomial (c in the function definition and the GSL ref manual) starting at the constant term and ending in the highest power; see details section. This argument is called "c_gsl" (and not "c") to avoid confusion with R function c()
x	input: real values

Details

One must be careful to avoid off-by-one errors. In C idiom, the function evaluates the polynomial

$$c[0] + c[1]x + c[2]x^2 + \ldots + c[\text{len} - 1]x^{\text{len}-1}$$

where len is the second argument of GSL function `gsl_poly_eval()`.

The R idiom would be

$$c[1] + c[2]x + c[3]x^2 + \ldots + c[\text{len}]x^{\text{len}-1}.$$

This section is work-in-progress and more will be added when I have the time/need for the other functions here.

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
a <- matrix(1:4,2,2)
rownames(a) <- letters[1:2]
(jj <- gsl_poly(1:3,a))

jj-(1 + 2*a + 3*a^2)  #should be small
```

`Powint` *Power functions*

Description

Power functions as per the Gnu Scientific Library reference manual section 7.27. These functions are declared in the header file `gsl_sf_pow_int.h`

Usage

```
pow_int(x, n, give=FALSE, strict=TRUE)
```

Arguments

`x`	input: real values
`n`	input: integer values
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`strict`	Boolean, with `TRUE` meaning to return `NaN` if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
pow_int(pi/2,1:10)
```

Psi *Psi (digamma) functions*

Description

Psi (digamma) functions as per the Gnu Scientific Library, reference manual section 7.27. These functions are declared in header file `gsl_sf_psi.h`

Usage

```
psi_int(n, give=FALSE, strict=TRUE)
psi(x, give=FALSE, strict=TRUE)
psi_1piy(y, give=FALSE, strict=TRUE)
psi_1_int(n, give=FALSE, strict=TRUE)
psi_1(x, give=FALSE, strict=TRUE)
psi_n(m, x, give=FALSE, strict=TRUE)
```

Arguments

`m,n`	input: integer values
`x,y`	input: real values
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`strict`	Boolean, with default `TRUE` meaning to return `NaN` if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=1.2,to=1.25,by=0.005)
cbind(x,psi(x),psi_1(x))
#tabe 6.1, p267, bottom bit

psi_int(1:6)
psi(pi+(1:6))
psi_1piy(pi+(1:6))
psi_1_int(1:6)
psi_n(m=5,x=c(1.123,1.6523))
```

Qrng *Quasi-random sequences*

Description

Quasi-random sequences as per the Gnu Scientific Library, reference manual section 18. These functions are declared in header file `gsl_qrng.h`

Usage

```
qrng_alloc(type = c("niederreiter_2", "sobol"), dim)
qrng_clone(q)
qrng_init(q)
qrng_name(q)
qrng_size(q)
qrng_get(q, n = 1)
```

Arguments

`type`	Type of sequence
`dim`	Dimension of sequence
`q`	Generator from `qrng_alloc` or `qrng_clone`
`n`	How many vectors to generate

Details

These are wrappers for the quasi-random sequence functions from the GSL https://www.gnu.org/software/gsl/ with arguments corresponding to those from the library, with a few exceptions. In particular: I have used `dim` where the GSL uses just `d`; I have added the `n` argument to the `qrng_get` function, so that a single call can generate `n` vectors; I have not provided R functions corresponding to `qrng_free` (because R will automatically free the generator when it is garbage collected) or `qrng_state` or `qrng_memcpy` (because these don't make sense within R.)

Value

qrng_alloc, qrng_clone and qrng_init return an external pointer to the C structure representing the generator. The internals of this structure are not accessible from within R.

qrng_name returns a character vector giving the name of the generator.

qrng_size returns an integer value giving the internal memory usage of the generator.

qrng_get returns a matrix with n rows and dim columns. Each row is a vector in the quasi-random sequence.

Author(s)

Duncan Murdoch

References

https://www.gnu.org/software/gsl/

Examples

```
q <- qrng_alloc(dim = 2)
qrng_name(q)
qrng_get(q, 10)
```

Rng *Random numbers generation*

Description

Random number generation with the Gnu Scientific Library, as per the reference manual section 17

Usage

```
rng_alloc(type)
rng_clone(r)
rng_name(r)
rng_max(r)
rng_min(r)
rng_set(r, seed)
rng_get(r, length)
rng_uniform(r, length)
rng_uniform_int(r, N, length)
rng_uniform_pos(r, length)
```

Arguments

type	In function rng_alloc(), type of random number generator. This argument is taken to be a character string which is matched to the names of the random number generators given in the GSL manual section 17.9, with the initial "gsl_rng_" removed (for example, to use generator gsl_rng_ranlux, set type to ranlux). Partial matching is used; a null string is interpreted as mt19937.
r	Instance of a random number generator. Generate this using function rng_alloc().
seed	Random number seed
length	Length of vector of random numbers to create
N	In function rng_uniform_int(), upper bound of uniform distribution

Details

These are wrappers for the random number generator functions from the GSL https://www.gnu.org/software/gsl/ with arguments corresponding to those from the library. Calling rng_free is not necessary as R performs garbage collection automatically.

The functions that return random numbers (rng_get, rng_uniform, rng_uniform_int, rng_uniform_pos) take an extra argument that specifies the length of the vector of random numbers to be returned.

Value

Function rng_alloc() returns an external pointer to a GSL random number generator.

Author(s)

Max Bruche

References

https://www.gnu.org/software/gsl/

Examples

```
r <- rng_alloc("cmrg")
rng_set(r, 100)
rng_uniform(r, 10)
```

Synchrotron *Synchrotron functions*

Description

Synchrotron functions as per the Gnu Scientific Library, reference section 7.29. These functions are declared in header file gsl_sf_synchrotron.h

Usage

```
synchrotron_1(x, give=FALSE, strict=TRUE)
synchrotron_2(x, give=FALSE, strict=TRUE)
```

Arguments

x	input: real values
give	Boolean with TRUE meaning to return a list of three items: the value, an estimate of the error, and a status number
strict	Boolean, with TRUE meaning to return NaN if status is an error

Author(s)

Robin K. S. Hankin

Examples

```
x <- seq(from=0,to=2,by=0.01)
synchrotron_1(x)
synchrotron_2(x)
```

Transport *Transport functions*

Description

Transport functions as per the Gnu Scientific Library, reference manual section 7.29. These functions are defined in header file gsl_sf_transport.h

Usage

```
transport_2(x, give=FALSE, strict=TRUE)
transport_3(x, give=FALSE, strict=TRUE)
transport_4(x, give=FALSE, strict=TRUE)
transport_5(x, give=FALSE, strict=TRUE)
```

Arguments

x	input: real values
give	Boolean with TRUE meaning to return a list of three items: the value, an estimate of the error, and a status number.
strict	Boolean, with TRUE meaning to return NaN if status is an error.

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=0,to=2,by=0.01)
transport_2(x)
transport_3(x)
```

Trig *Trig functions*

Description

Trig functions as per the Gnu Scientific Library, reference manual section 7.30. These functions are declared in header file `gsl_sf_trig.h`

Usage

```
gsl_sf_sin(x, give=FALSE, strict=TRUE)
gsl_sf_cos(x, give=FALSE, strict=TRUE)
hypot(x, y, give=FALSE, strict=TRUE)
sinc(x, give=FALSE, strict=TRUE)
complex_sin(zr, zi=NULL, r.and.i=TRUE, give=FALSE, strict=TRUE)
complex_cos(zr, zi=NULL, r.and.i=TRUE, give=FALSE, strict=TRUE)
lnsinh(x, give=FALSE, strict=TRUE)
lncosh(x, give=FALSE, strict=TRUE)
```

Arguments

`x,y`	input: real values
`zr`	In `gamma_complex()`, the real part of the argument
`zi`	In `complex_sin()` et seq, the imaginary part of the argument. If missing (ie takes the default value of `NULL`), interpret `zr` as complex, even if real
`r.and.i`	In `complex_sin()` et seq, Boolean variable with default value of `TRUE` meaning to return a complex variable as per the details section below; and `FALSE` meaning to return the values as advertised in the GSL manual
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number
`strict`	Boolean, with `TRUE` meaning to return `NaN` if status is an error

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
x <- seq(from=0,to=2,by=0.01)
gsl_sf_sin(x)   #table xx of Ab and St
gsl_sf_cos(x)   #table xx of Ab and St

f <- function(x){abs(sin(x+1)-sin(x)*cos(1)-cos(x)*sin(1))}
g <-
function(x){abs(gsl_sf_sin(x+1)-gsl_sf_sin(x)*gsl_sf_cos(1)-gsl_sf_cos(x)*gsl_sf_sin(1))}

f(100000:100010)
g(100000:100010)
```

Zeta *Zeta functions*

Description

Zeta functions as per the Gnu Scientific Library 7.31 and AMS-55, section 23.2. These functions are declared in header file `gsl_sf_zeta.h`

Usage

```
zeta_int(n, give=FALSE, strict=TRUE)
zeta(s, give=FALSE, strict=TRUE)
zetam1_int(n, give=FALSE, strict=TRUE)
zetam1(s, give=FALSE, strict=TRUE)
hzeta(s, q, give=FALSE, strict=TRUE)
eta_int(n, give=FALSE, strict=TRUE)
eta(s, give=FALSE, strict=TRUE)
```

Arguments

`n`	input: integer values
`s,q`	input: real values
`give`	Boolean with `TRUE` meaning to return a list of three items: the value, an estimate of the error, and a status number.
`strict`	Boolean, with `TRUE` meaning to return `NaN` if status is an error.

Author(s)

Robin K. S. Hankin

References

https://www.gnu.org/software/gsl/

Examples

```
n <- 1:10
cbind(n,zeta(n),eta(n))   #table 23.3, p 811


zeta_int(1:5)
zeta(c(pi,pi*2))
zetam1_int(1:5)
zetam1(c(pi,pi*2))
hzeta(1.1,1.2)
eta_int(1:5)
eta(c(pi,pi*2))
```

Index