`SPECFN`: Special Functions Package for REDUCE

Chris Cannam, et. al.
Konrad–Zuse–Zentrum für Informationstechnik Berlin
Takustrasse 7
D–14195 Berlin – Dahlem
Federal Republic of Germany
E–mail: neun@zib.de

Version 2.5, October 1998

1 Introduction

This package provides the 'common' special functions for REDUCE. The names of the operators and implementation details can be found in this document.

Due to the enormous number of special functions a package for special functions is never complete. Several users pointed out that important classes of special functions were missing in the first version. These comments and other hints from a number of contributors and users were very helpful.

The first version of this package was developed while the author worked as a student exchange grantee at ZIB Berlin in 1992/93. The package is maintained by ZIB Berlin after the author left the ZIB. Therefore, please direct comments, hints and bug reports etc. to `neun@zib.de`. Numerous contributions have been integrated after the release with version 3.5 of REDUCE.

This package is designed to provide algebraic and numeric manipulations of several common special functions, namely:

- Bernoulli numbers and Polynomials;
- Euler numbers and Polynomials;
- Fibonacci numbers and Polynomials;

- Stirling numbers;
- Binomial Coefficients;
- Pochhammer notation;
- The Gamma function;
- The psi function and its derivatives;
- The Riemann Zeta function;
- The Bessel functions J and Y of the first and second kinds;
- The modified Bessel functions I and K;
- The Hankel functions H1 and H2;
- The Kummer hypergeometric functions M and U;
- The Beta function, and Struve, Lommel and Whittaker functions;
- The Airy funcions;
- The Exponential Integral, the Sine and Cosine Integrals;
- The Hyperbolic Sine and Cosine Integrals;
- The Fresnel Integrals and the Error function;
- The Dilog function;
- The Polylogarithm and Lerch Phi function;
- Hermite Polynomials;
- Jacobi Polynomials;
- Legendre Polynomials;
- Associated Legendre Functions (Spherical and Solid Harmonics)
- Laguerre Polynomials;
- Chebyshev Polynomials;
- Gegenbauer Polynomials;
- Lambert's ω function;
- (Jacobi's) Elliptic Functions;
- Elliptic Integrals;
- 3j and 6j symbols , Clebsch-Gordan coefficients;
- and some well-known constants.

All algorithms whose sources are uncredited are culled from series or expressions found in the Dover Handbook of Mathematical Functions[**?**].

There is a nice collection of plot calls for special functions in the file $reduce/plot/specplot.tst. These examples will reproduce a number of well-known pictures from [**?**].

2 Compatibility with earlier REDUCE versions

For PSL versions, this package is intended to be used with the new REDUCE bigfloat mechanisms which is distributed together with REDUCE 3.5 and later versions. The package does work with the earlier bigfloat implementations, but in order to ensure that it works efficiently with the new versions, it has not been optimized for the old.

3 Simplification and Approximation

All of the operators supported by this package have certain algebraic simplification rules to handle special cases, poles, derivatives and so on. Such rules are applied whenever they are appropriate. However, if the `ROUNDED` switch is on, numeric evaluation is also carried out. Unless otherwise stated below, the result of an application of a special function operator to real or complex numeric arguments in rounded mode will be approximated numerically whenever it is possible to do so. All approximations are to the current precision.

Most algebraic simplifications within the special function package are defined in the form of a REDUCE ruleset. Therefore, in order to get a quick insight into the simplification rules one can use the ShowRules operator, e.g.

```
ShowRules BesselI;

                          1          ~z     - ~z
{besseli(~n,~z) => ---------------*(e   - e     )
                   sqrt(pi*2*~z)
```

```
                          1
 when numberp(~n) and ~n=---,
                          2

                         1          ~z     - ~z
besseli(~n,~z) => ---------------*(e   + e     )
                  sqrt(pi*2*~z)

                            1
 when numberp(~n) and ~n= - ---,
                            2

besseli(~n,~z) => 0

 when numberp(~z) and ~z=0 and numberp(~n) and ~n neq 0,

besseli(~n,~z) => besseli( - ~n,~z) when numberp(~n)

 and impart(~n)=0 and ~n=floor(~n) and ~n<0,

besseli(~n,~z) => do*i(~n,~z)

 when numberp(~n) and numberp(~z) and *rounded,

df(besseli(~n,~z),~z)

     besseli(~n - 1,~z) + besseli(~n + 1,~z)
 => ----------------------------------------,
                        2

df(besseli(~n,~z),~z)

 => besseli(1,~z) when numberp(~n) and ~n=0}
```

Several REDUCE packages (such as Sum or Limits) obtain different (hopefully better) results for the algebraic simplifications when the SPECFN package is loaded, because the later package contains some information which may be useful and directly applicable for other packages, e.g.:

```
sum(1/k^s,k,1,infinity); % will be evaluated to

zeta(s)
```

A record is kept of all values previously approximated, so that should a value be required which has already been computed to the current precision or greater, it can be simply looked up. This can result in some storage overheads, particularly if many values are computed which will not be needed again. In this case, the switch `savesfs` may be turned off in order to inhibit the storage of approximated values. The switch is on by default.

4 Constants

Some well-known constants are defined in the special function package. Important properties of these constants which can be used to define them are also known. Numerical values are computed at arbitrary precision if the switch ROUNDED is on.

- Euler_Gamma : Euler's constants, also available as -$\psi(1)$;
- Catalan : Catalan's constant;
- Khinchin : Khinchin's constant , defined in [**?**]. (takes a lot of time to compute);
- Golden_Ratio : $\frac{1+\sqrt{5}}{2}$

5 Bernoulli Numbers and Euler Numbers

The unary operator `Bernoulli` provides notation and computation for Bernoulli numbers. `Bernoulli(n)` evaluates to the nth Bernoulli number; all of the odd Bernoulli numbers, except `Bernoulli(1)`, are zero.

The algorithms are based upon those by Herbert Wilf, presented by Sandra Fillebrown [**?**]. If the `ROUNDED` switch is off, the algorithms are exactly those; if it is on, some further rounding may be done to prevent computation of redundant digits. Hence, these functions are particularly fast when used to approximate the Bernoulli numbers in rounded mode.

Euler numbers are computed by the unary operator Euler, which return

the nth Euler number. The computation is derived directly from Pascal's triangle of binomial coefficients.

6 Fibonacci Numbers and Fibonacci Polynomials

The unary operator `Fibonacci` provides notation and computation for Fibonacci numbers. `Fibonacci(n)` evaluates to the nth Fibonacci number. If n is a positive or negative integer, it will be evaluated following the definition:

$F_0 = 0; F_1 = 1; F_n = F_{n-1} + F_{n-2}$

Fibonacci Polynomials are computed by the binary operator FibonacciP. FibonacciP(n,x) returns the nth Fibonaccip polynomial in the variable x. If n is a positive or negative integer, it will be evaluated following the definition:

$F_0(x) = 0; F_1(x) = 1; F_n(x) = xF_{n-1}(x) + F_{n-2}(x)$

7 Stirling Numbers

Stirling numbers of the first and second kind are computed by the binary operators `Stirling1` and `Stirling2` using explicit formulae.

8 The Γ Function, and Related Functions

8.1 The Γ Function

This is represented by the unary operator `Gamma`.

Initial transformations applied with `ROUNDED` off are: $\Gamma(n)$ for integral n is computed, $\Gamma(n + 1/2)$ for integral n is rewritten to an expression in $\sqrt{\pi}$, $\Gamma(n + 1/m)$ for natural n and m a positive integral power of 2 less than or equal to 64 is rewritten to an expression in $\Gamma(1/m)$, expressions with arguments at which there is a pole are replaced by `INFINITY`, and those with a negative (real) argument are rewritten so as to have positive arguments.

The algorithm used for numerical approximation is an implementation of an asymptotic series for $ln(\Gamma)$, with a scaling factor obtained from the

Pochhammer functions.

An expression for $\Gamma'(z)$ in terms of Γ and ψ is included.

8.2 The Pochhammer Notation

The Pochhammer notation $(a)_k$ is supported by the binary operator `Pochhammer`. With `ROUNDED` off, this expression is evaluated numerically if a and k are both integral, and otherwise may be simplified where appropriate. The simplification rules are based upon algorithms supplied by Wolfram Koepf [**?**].

8.3 The Digamma Function, ψ

This is represented by the unary operator `PSI`.

Initial transformations for ψ are applied on a similar basis to those for Γ; where possible, $\psi(x)$ is rewritten in terms of $\psi(1)$ and $\psi(\frac{1}{2})$, and expressions with negative arguments are rewritten to have positive ones.

Numerical evaluation of ψ is only carried out if the argument is real. The algorithm used is based upon an asymptotic series, with a suitable scaler.

Relations for the derivative and integral of ψ are included.

8.4 The Polygamma Functions, $\psi^{(n)}$

The nth derivative of the ψ function is represented by the binary operator `Polygamma`, whose first argument is n.

Initial manipulations on $\psi^{(n)}$ are few; where the second argument is 1 or 3/2, the expression is rewritten to one involving the Riemann ζ function, and when the first is zero it is rewritten to ψ; poles are also handled.

Numerical evaluation is only carried out with real arguments. The algorithm used is again an asymptotic series with a scaling factor; for negative (second) arguments, a Reflection Formula is used, introducing a term in the nth derivative of $\cot(z\pi)$.

Simple relations for derivatives and integrals are provided.

8.5 The Riemann ζ Function

This is represented by the unary operator `Zeta`.

With `ROUNDED` off, $\zeta(z)$ is evaluated numerically for even integral arguments in the range $-31 < z < 31$, and for odd integral arguments in the range $-30 < z < 16$. Outside this range the values become a little unwieldy.

Numerical evaluation of ζ is only carried out if the argument is real. The algorithms used for ζ are: for odd integral arguments, an expression relating $\zeta(n)$ with $\psi^{n-1}(3)$; for even arguments, a trivial relationship with the Bernoulli numbers; and for other arguments the approach is either (for larger arguments) to take the first few primes in the standard over-all-primes expansion, and then continue with the defining series with natural numbers not divisible by these primes, or (for smaller arguments) to use a fast-converging series obtained from [**?**].

There are no rules for differentiation or integration of ζ.

9 Bessel Functions

Support is provided for the Bessel functions J and Y, the modified Bessel functions I and K, and the Hankel functions of the first and second kinds. The relevant operators are, respectively, `BesselJ`, `BesselY`, `BesselI`, `BesselK`, `Hankel1` and `Hankel2`, which are all binary operators.

The following initial transformations are performed:

- trivial cases or poles of J, Y, I and K are handled;
- J, Y, I and K with negative first argument are transformed to have positive first argument;
- J with negative second argument is transformed for positive second argument;
- Y or K with non-integral or complex second argument is transformed into an expression in J or I respectively;
- derivatives of J, Y and I are carried out;
- derivatives of K with zero first argument are carried out;
- derivatives of Hankel functions are carried out.

Also, if the `COMPLEX` switch is on and `ROUNDED` is off, expressions in Hankel functions are rewritten in terms of Bessels.

No numerical approximation is provided for the Bessel K function, or for the Hankel functions for anything other than special cases. The algorithms used for the other Bessels are generally implementations of standard ascending series for J, Y and I, together with asymptotic series for J and Y; usually, the asymptotic series are tried first, and if the argument is too small for them to attain the current precision, the standard series are applied. An obvious optimization prevents an attempt with the asymptotic series if it is clear from the outset that it will fail.

There are no rules for the integration of Bessel and Hankel functions.

10 Hypergeometric and Other Functions

This package also provides some support for other functions, in the form of algebraic simplifications:

- The Beta function, a variation upon the Γ function[**?**], with the binary operator `Beta`;
- The Struve **H** and **L** functions, through the binary operators `StruveH` and `StruveL`, for which manipulations are provided to handle special cases, simplify to more readily handled functions where appropriate, and differentiate with respect to the second argument;
- The Lommel functions of the first and second kinds, through the ternary operators `Lommel1` and `Lommel2`, for which manipulations are provided to handle special cases and simplify where appropriate;
- The Kummer confluent hypergeometric functions M and U (the hypergeometric ${}_1F_1$ or Φ, and $z^{-a}{}_2F_0$ or Ψ, respectively), with the ternary operators `KummerM` and `KummerU`, for which there are manipulations for special cases and simplifications, derivatives and, for the M function, numerical approximations for real arguments;
- The Whittaker M and W functions, variations upon the Kummer functions, which, with the ternary operators `WhittakerM` and `WhittakerW`, simplify to expressions in the Kummer functions.

11 Integral Functions

The SPECFN package includes manipulation and a limited numerical evaluation for some Integral functions, namely

erf, erfc, Si, Shi, si, Ci, Chi, Ei, li, Fresnel_C and Fresnel_S.

The definitions from integral, the derviatives and some limits are known together with some simple properties such as symmetry conditions.

The numerical approximation for the Integral functions suffer from the fact that the precision is not set correctly for values of the argument above 10.0 (approx.) and from the usage of summations even for large arguments.

li is simplified towards $Ei(ln(z))$.

12 Airy Functions

Support is provided for the Airy Functions Ai and Bi and for the Airyprime Functions Aiprime and Biprime. The relevant operators are respectively `Airy_Ai`, `Airy_Bi`, `Airy_Aiprime`, and `Airy_Biprime`, which are all unary operators with one argument.

The following cases can be performed:

- Trivial cases of Airy_Ai and Airy_Bi and their primes are handled.
- All cases can handle both complex and real arguments.
- The Airy Functions can also be represented in terms of Bessel Functions by activating an inactive rule set.

In order to activate the Airy Function to Bessel Rules one should type: `let Airy2Bessel_rules;`. As a result the Airy_Ai function, for example will be calculated using the formula :-

`Ai(z) =` $\frac{1}{3}\sqrt{z}[I_{-1/3}(\zeta)$ - $I_{1/3}(\zeta)]$, where $\zeta = \frac{2}{3}z^{\frac{2}{3}}$

Note:- In order to obtain satisfactory approximations to results both the `COMPLEX` and `ROUNDED` switches must be on.

The algorithms used for the Airy Functions are implementations of standard ascending series, together with asymptotic series. At some point it is

better to use the asymptotic approach, rather than the series. This value is calculated by the program and depends on the given precision.

There are no rules for the integration of Airy Functions.

13 Polynomial Functions

Two groups are defined, some well-known orthogonal Polynomials (Hermite, Jacobi, Legendre, Laguerre, Chebyshev, Gegenbauer) and Euler and Bernoulli Polynomials. The names of the REDUCE operator are build by adding a P to the name of the polynomials, e.g. EulerP implements the Euler polynomials. Most definitions are equivalent to [**?**], except for the ternary (associated) Legendre Polynomials.

```
P(n,m,x) = (-1)^m *(1-x^2)^(m/2)*df(legendreP (n,x),x,m)
```

14 Spherical and Solid Harmonics

The relevant operators are, respectively,
`SolidHarmonicY` and `SphericalHarmonicY`.

The SolidHarmonicY operator implements the Solid Harmonics described below. It expects 6 parameter, namely n,m,x,y,z and r2 and returns a polynomial in x,y,z and r2.

The operator SphericalHarmonicY is a special case of SolidHarmonicY with the usual definition:

```
algebraic procedure SphericalHarmonicY(n,m,theta,phi);
        SolidHarmonicY(n,m,sin(theta)*cos(phi),
                sin(theta)*sin(phi),cos(theta),1)$
```

Solid Harmonics of order n (Laplace polynomials) are homogeneous polynomials of degree n in x,y,z which are solutions of Laplace equation:-

```
df(P,x,2) + df(P,y,2) + df(P,z,2) = 0.
```

There are 2*n+1 independent such polynomials for any given $n >= 0$ and with:-

```
w!0 = z, w!+ = i*(x-i*y)/2, w!- = i*(x+i*y)/2,
```

they are given by the Fourier integral:-

```
S(n,m,w!-,w!0,w!+) =

       (1/(2*pi)) *
       for u:=-pi:pi integrate (w!0 + w!+ * exp(i*u) + w!- *
           exp(-i*u))^n * exp(i*m*u) * du;
```

which is obviously zero if $|m| > n$ since then all terms in the expanded integrand contain the factor exp(i*k*u) with k neq 0,

S(n,m,x,y,z) is proportional to

```
r^n * Legendre(n,m,cos theta) * exp(i*phi)
```

Let r2 = $x^2 + y^2 + z^2$ and r = sqrt(r2).

The spherical harmonics are simply the restriction of the solid harmonics to the surface of the unit sphere and the set of all spherical harmonics $n >= 0; -n <= m =< n$ form a complete orthogonal basis on it, i.e. $< n, m|n', m' >$ = Kronecker_delta(n,n') * Kronecker_delta(m,m') using $<$ ...|... $>$ to designate the scalar product of functions over the spherical surface.

The coefficients of the solid harmonics are normalised in what follows to yield an ortho-normal system of spherical harmonics.

Given their polynomial nature, there are many recursions formulae for the solid harmonics and any recursion valid for Legendre functions can be 'translated' into solid harmonics. However the direct proof is usually far simpler using Laplace's definition.

It is also clear that all differentiations of solid harmonics are trivial, qua polynomials.

Some substantial reduction in the symbolic form would occur if one maintained throughout the recursions the symbol r2 (r cannot occur as it is not rational in x,y,z). Formally the solid harmonics appear in this guise as more compact polynomials in (x,y,z,r2).

Only two recursions are needed:-

(i) along the diagonal (n,n);

(ii) along a line of constant n: (m,m),(m+1,m),...,(n,m).

Numerically these recursions are stable.

For $m < 0$ one has:-

```
S(n,m,x,y,z) = (-1)^m * S(n,-m,x,-y,z);
```

15 Jacobi's Elliptic Functions

The following functions have been implemented:

- The Twelve Jacobi Functions
- Arithmetic Geometric Mean
- Descending Landen Transformation

15.1 Jacobi Functions

The following Jacobi functions are available:-

- Jacobisn(u,m)
- Jacobidn(u,m)
- Jacobicn(u,m)
- Jacobicd(u,m)
- Jacobisd(u,m)
- Jacobind(u,m)
- Jacobidc(u,m)
- Jacobinc(u,m)
- Jacobisc(u,m)
- Jacobins(u,m)
- Jacobids(u,m)
- Jacobics(u,m)

They will be evaluated numerically if the `rounded` switch is used.

15.2 Amplitude

The amplitude of u can be evaluated using the `JacobiAmplitude(u,m)` command.

15.3 Arithmetic Geometric Mean

A procedure to evaluate the AGM of initial values a_0, b_0, c_0 exists as `AGM_function(`a_0, b_0, c_0`)` and will return
$\{N, AGM, \{a_N, \ldots, a_0\}, \{b_N, \ldots, b_0\}, \{c_N, \ldots, c_0\}\}$, where N is the number of steps to compute the AGM to the desired acuracy.

To determine the Elliptic Integrals K(m), E(m) we use initial values $a_0 = 1$; $b_0 = \sqrt{1-m}$; $c_0 = \sqrt{m}$.

15.4 Descending Landen Transformation

The procedure to evaluate the Descending Landen Transformation of phi and alpha uses the following equations:

$(1 + sin\alpha_{n+1})(1 + cos\alpha_n) = 2$ where $\alpha_{n+1} < \alpha_n$
$tan(\phi_{n+1} - \phi_n) = cos\alpha_n tan\phi_n$ where $\phi_{n+1} > \phi_n$

It can be called using `landentrans(`ϕ_0`,`α_0`)` and will return
$\{\{\phi_0, \ldots, \phi_n\}, \{\alpha_0, \ldots, \alpha_n\}\}$.

16 Elliptic Integrals

The following functions have been implemented:

- Elliptic Integrals of the First Kind
- Elliptic Integrals of the Second Kind
- Jacobi θ Functions
- Jacobi ζ Function

16.1 Elliptic F

The Elliptic F function can be used as `EllipticF(`ϕ`,m)` and will return the value of the Elliptic Integral of the First Kind.

16.2 Elliptic K

The Elliptic K function can be used as `EllipticK(m)` and will return the value of the Complete Elliptic Integral of the First Kind, K. It is often used in the calculation of other elliptic functions

16.3 Elliptic K′

The Elliptic K′ function can be used as `EllipticK!'(m)` and will return the value $\mathrm{K}(1-m)$.

16.4 Elliptic E

The Elliptic E function comes with two different numbers of arguments:

It can be used with two arguments as `EllipticE(`ϕ`,m)` and will return the value of the Elliptic Integral of the Second Kind.

The Elliptic E function can also be used as `EllipticE(m)` and will return the value of the Complete Elliptic Integral of the Second Kind, E.

16.5 Elliptic Θ Functions

This can be used as `EllipticTheta(a,u,m)`, where a is the index for the theta functions ($a = 1, 2, 3$ or 4) and will return H; H_1; Θ_1; Θ. (Also denoted in some texts as ϑ_1; ϑ_2; ϑ_3; ϑ_4.)

16.6 Jacobi's Zeta Function Z

This can be used as `JacobiZeta(u,m)` and will return Jacobi Zeta. Note: the operator `Zeta` will invoke Riemann's ζ function.

17 Lambert's W function

Lambert's W function is the inverse of the function $w * e^w$. Therefore it is an important contribution for the solve package. The function is studied

extensively in [**?**]. The current implementation will compute the principal branch in ROUNDED mode only.

18 3j symbols and Clebsch-Gordan Coefficients

The operators `ThreeJSymbol`, `Clebsch_Gordan` are defined like in [**?**] or [**?**]. ThreeJSymbol expects as arguments three lists of values $\{j_i, m_i\}$, e.g.

```
ThreeJSymbol({J+1,M},{J,-M},{1,0});
Clebsch_Gordan({2,0},{2,0},{2,0});
```

19 6j symbols

The operator `SixJSymbol` is defined like in [**?**] or [**?**]. SixJSymbol expects two lists of values $\{j_1, j_2, j_3\}$ and $\{l_1, l_2, l_3\}$ as arguments, e.g.

```
SixJSymbol({7,6,3},{2,4,6});
```

In the current implementation of the SixJSymbol, there is only a limited reasoning about the minima and maxima of the summation using the INEQ package, such that in most cases the special 6j-symbols (see e.g. [**?**]) will not be found.

20 Acknowledgements

The contributions of Kerry Gaskell, Matthew Rebbeck, Lisa Temme, Stephen Scowcroft and David Hobbs (all students from the University of Bath on placement in ZIB Berlin for one year) were very helpful to augment the package. The advise of René Grognard (CSIRO , Australia) for the development of the module for Clebsch-Gordan and 3j, 6j symbols and the module for spherical and solid harmonics was very much appreciated.

21 Table of Operators and Constants

Function	Operator
$J_\nu(z)$	`BesselJ(nu,z)`
$Y_\nu(z)$	`BesselY(nu,z)`
$I_\nu(z)$	`BesselI(nu,z)`
$K_\nu(z)$	`BesselK(nu,z)`
$H_\nu^{(1)}(z)$	`Hankel1(n,z)`
$H_\nu^{(2)}(z)$	`Hankel2(n,z)`
$\mathbf{H}_\nu(z)$	`StruveH(nu,z)`
$\mathbf{L}_\nu(z)$	`StruveL(n,z)`
$s_{a,b}(z)$	`Lommel1(a,b,z)`
$S_{a,b}(z)$	`Lommel2(a,b,z)`
$Ai(z)$	`Airy_Ai(z)`
$Bi(z)$	`Airy_Bi(z)`
$Ai'(z)$	`Airy_Aiprime(z)`
$Bi'(z)$	`Airy_Biprime(z)`
$M(a,b,z)$ or ${}_1F_1(a,b;z)$ or $\Phi(a,b;z)$	`KummerM(a,b,z)`
$U(a,b,z)$ or $z^{-a}{}_2F_0(a,b;z)$ or $\Psi(a,b;z)$	`KummerU(a,b,z)`
$M_{\kappa,\mu}(z)$	`WhittakerM(kappa,mu,z)`
$W_{\kappa,\mu}(z)$	`WhittakerW(kappa,mu,z)`
Fibonacci Numbers F_n	`Fibonacci(n)`
Fibonacci Polynomials $F_n(x)$	`FibonacciP(n)`
$B_n(x)$	`BernoulliP(n,x)`
$E_n(x)$	`EulerP(n,x)`
$C_n^{(\alpha)}(x)$	`GegenbauerP(n,alpha,x)`
$H_n(x)$	`HermiteP(n,x)`
$L_n(x)$	`LaguerreP(n,x)`
$L_n^{(m)}(x)$	`LaguerreP(n,m,x)`
$P_n(x)$	`LegendreP(n,x)`
$P_n^{(m)}(x)$	`LegendreP(n,m,x)`

Function	Operator
$P_n^{(\alpha,\beta)}(x)$	`JacobiP(n,alpha,beta,x)`
$U_n(x)$	`ChebyshevU(n,x)`
$T_n(x)$	`ChebyshevT(n,x)`
$Y_n^m(x,y,z,r2)$	`SolidHarmonicY(n,m,x,y,z,r2)`
$Y_n^m(\theta,\phi)$	`SphericalHarmonicY(n,m,theta,phi)`
$\begin{pmatrix} j_1 & j_2 & j_3 \\ m_1 & m_2 & m_3 \end{pmatrix}$	`ThreeJSymbol({j1,m1},{j2,m2},{j3,m3})`
$(j_1 m_1 j_2 m_2 \vert j_1 j_2 j_3 - m_3)$	`Clebsch_Gordan({j1,m1},{j2,m2},{j3,m3})`
$\begin{Bmatrix} j_1 & j_2 & j_3 \\ l_1 & l_2 & l_3 \end{Bmatrix}$	`SixJSymbol({j1,j2,j3},{l1,l2,l3})`
$sn(u\vert m)$	`Jacobisn(u,m)`
$dn(u\vert m)$	`Jacobidn(u,m)`
$cn(u\vert m)$	`Jacobicn(u,m)`
$cd(u\vert m)$	`Jacobicd(u,m)`
$sd(u\vert m)$	`Jacobisd(u,m)`
$nd(u\vert m)$	`Jacobind(u,m)`
$dc(u\vert m)$	`Jacobidc(u,m)`
$nc(u\vert m)$	`Jacobinc(u,m)`
$sc(u\vert m)$	`Jacobisc(u,m)`
$ns(u\vert m)$	`Jacobins(u,m)`
$ds(u\vert m)$	`Jacobids(u,m)`
$cs(u\vert m)$	`Jacobics(u,m)`
$F(\phi\vert m)$	`EllipticF(phi,m)`
$K(m)$	`EllipticK(m)`
$E(\phi\vert m) or E(m)$	`EllipticE(phi,m) or EllipticE(m)`
$H(u\vert m), H_1(u\vert m), \Theta_1(u\vert m), \Theta(u\vert m)$	`EllipticTheta(a,u,m)`
$\theta_1(u\vert m), \theta_2(u\vert m), \theta_3(u\vert m), \theta_4(u\vert m)$	`EllipticTheta(a,u,m)`
$Z(u\vert m)$	`Zeta_function(u,m)`
Lambert $\omega(z)$	`Lambert_W(z)`

Constant	REDUCE name
Euler's γ constant	`Euler_gamma`
Catalan's constant	`Catalan`
Khinchin's constant	`Khinchin`
Golden ratio	`Golden_ratio`

Function	Operator
$\binom{n}{m}$	`Binomial(n,m)`
Motzkin(n)	`Motzkin(n)`
Bernoulli(n) or B_n	`Bernoulli(n)`
Euler(n) or E_n	`Euler(n)`
$S_n^{(m)}$	`Stirling1(n,m)`
$\mathbf{S}_n^{(m)}$	`Stirling2(n,m)`
$B(z,w)$	`Beta(z,w)`
$\Gamma(z)$	`Gamma(z)`
incomplete Beta $B_x(a,b)$	`iBeta(a,b,x)`
incomplete Gamma $\Gamma(a,z)$	`iGamma(a,z)`
$(a)_k$	`Pochhammer(a,k)`
$\psi(z)$	`Psi(z)`
$\psi^{(n)}(z)$	`Polygamma(n,z)`
Riemann's $\zeta(z)$	`Zeta(z)`
$Si(z)$	`Si(z)`
$si(z)$	`s_i(z)`
$Ci(z)$	`Ci(z)`
$Shi(z)$	`Shi(z)`
$Chi(z)$	`Chi(z)`
$erf(z)$	`erf(z)`
$erfc(z)$	`erfc(z)`
$Ei(z)$	`Ei(z)`
$li(z)$	`li(z)`
$C(x)$	`Fresnel_C(x)`
$S(x)$	`Fresnel_S(x)`
$dilog(z)$	`dilog(z)`
$Li_n(z)$	`Polylog(n,z)`
Lerch $\Phi(z,s,a)$	`Lerch_Phi(z,s,a)`

References