

The cool package*

nsetzer

September 24, 2006

This is the cool package: a COntent Oriented L^AT_EX package. That is, it is designed to give L^AT_EX commands the ability to contain the mathematical meaning while retaining the typesetting versatility.

This package requires the following, non-standard L^AT_EX packages (all of which are available on www.ctan.org): coolstr, coollist, forloop

1 Implementation

```
1 \newcounter{COOL@ct} %just a general counter
2 \newcounter{COOL@ct@}%just a general counter
```

1.1 Parenthesis

```
3 \newcommand{\inp}[2][0cm]{\left(#2\parbox[h]{#1}{0cm}{}\right)}
4           % in parentheses ()
5 \newcommand{\inb}[2][0cm]{\left[#2\parbox[h]{#1}{0cm}{}\right]}
6           % in brackets []
7 \newcommand{\inbr}[2][0cm]{\left\{#2\parbox[h]{#1}{0cm}{}\right\}}
8 % in braces {}
9 \newcommand{\inap}[2][0cm]{\left<#2\parbox[h]{#1}{0cm}{}\right>}
10 % in angular parentheses <>
11 \newcommand{\nop}[1]{\left.{#1}\right.}
12 % no parentheses
```

`\COOL@decide@paren` `\COOL@decide@paren[<parenthesis type>]{<function name>}{<contained text>}`.
Since the handling of parentheses is something that will be common to many elements this function will take care of it.

If the optional argument is given, `\COOL@notation@<function name>Paren` is ignored and *<parenthesis type>* is used

<parenthesis type> and `\COOL@notation@<function name>Paren` must be one of none, p for (), b for [], br for {}, ap for <>, inv for \left.\right.

```
13 \let\COOL@decide@paren@no@type=\relax
14 \newcommand{\COOL@decide@paren}[3][\COOL@decide@paren@no@type]{%
15 \ifthenelse{ \equal{#1}{\COOL@decide@paren@no@type} }{%
16 {%
```

*This document corresponds to cool v1.2, dated 2006/09/17.

```

17 \def\COOL@decide@paren@type{\csname COOL@notation@#2Paren\endcsname}%
18 }%
19 % Else
20 {%
21 \def\COOL@decide@paren@type{#1}%
22 }%
23 \ifthenelse{ \equal{\COOL@decide@paren@type}{none} }{%
24 {%
25 #3%
26 }%
27 % Else
28 {%
29 \ifthenelse{ \equal{\COOL@decide@paren@type}{p} }{%
30 {%
31 \inp{#3}%
32 }%
33 % Else
34 {%
35 \ifthenelse{ \equal{\COOL@decide@paren@type}{b} }{%
36 {%
37 \inb{#3}%
38 }%
39 % Else
40 {%
41 \ifthenelse{ \equal{\COOL@decide@paren@type}{br} }{%
42 {%
43 \inbr{#3}%
44 }%
45 % Else
46 {%
47 \ifthenelse{ \equal{\COOL@decide@paren@type}{ap} }{%
48 {%
49 \inap{#3}%
50 }%
51 % Else
52 {%
53 \ifthenelse{ \equal{\COOL@decide@paren@type}{inv} }{%
54 {%
55 \nop{#3}%
56 }%
57 % Else
58 {%
59 \PackageError{cool}{Invalid Parenthesis Option}%
60 {*Paren can only be 'none', 'p', 'b', 'br', 'ap', 'inv'}%
61 }%
62 }%
63 }%
64 }%
65 }%
66 }%

```

67 }

1.2 Indices

`\COOL@decide@indices` `\COOL@decide@indices{\function name}{\local indication}{\indices}`
 Since up or down indices can be as common as the parenthesis decision, this macro is the solution.

\local indication must be either u or d

\indices is very likely to be required to be a comma separated list in the near future

the options for indices are

- `local` allow the indices to be decided by an optional argument to the function (such as `\LeviCivita[u]{i j}`)
- `up` force the indices to appear as superscript
- `down` force the indices to appear as subscript

```

68 \newcommand{\COOL@decide@indices}[3]{%
69 \def\COOL@decide@indices@placement%
70 {\csname COOL@notation@#1Indices\endcsname}%
71 \ifthenelse{\equal{\COOL@decide@indices@placement}{local}}{%
72 {%
73 \ifthenelse{\equal{#2}{u}}{%
74 {^{#3}}%
75 {_{#3}}%
76 }%
77 % Else
78 {%
79 \ifthenelse{\equal{\COOL@decide@indices@placement}{up}}{%
80 {%
81 {^{#3}}%
82 }%
83 % Else
84 {%
85 \ifthenelse{\equal{\COOL@decide@indices@placement}{down}}{%
86 {%
87 {_{#3}}%
88 }%
89 % else
90 {%
91 \PackageError{cool}{Invalid Option Sent}%
92 {#1Indices can only be 'up', 'down', or 'local'}%
93 }%
94 }%
95 }%
96 }
```

1.3 COntent Oriented LaTeX (COOL)

\Style **\Style{⟨options⟩}** sets the style of the output (how to notate particular functions). *⟨options⟩* is a comma delimited list of the form *⟨key⟩=⟨value⟩*, where *⟨key⟩* is the *long* form of the command name without the preceeding backslash (i.e. **Integrate** and not **Int** or **\Int**). The list can be in any order and need only contain the styles that the user desires to set.

There can be multiple **\Style** commands within any document—the styled output of the command depends on the last **\Style** command to have specified its style.

For a list of styling options for a command, see the code where the command is defined

```

97 \newcommand{\Style}[1]{%
98 \COOL@keyeater#1,\COOL@keystop\COOL@keyend%
99 }
100 \newcommand{\COOL@keystop}{@@@}%
101 \def\COOL@keyeater#1=#2,#3\COOL@keyend{%
102 \ifx#3\COOL@keystop%
103 \expandafter\gdef\csname COOL@notation@#1\endcsname{#2}%
104 \else%
105 \expandafter\gdef\csname COOL@notation@#1\endcsname{#2}%
106 \COOL@keyeater#3\COOL@keyend%
107 \fi%
108 }
```

\UseStyleFile Since notational style should be kept consistent and will likely need to span several documents, use this command to input a notation style file that has previously been prepared. (to be implemented in a future release)

```

109 \newcommand{\UseStyleFile}[1]{}
```

1.3.1 Fundamental Constants

see <http://functions.wolfram.com/> for the definitions

\I The square root of minus 1 ($\sqrt{-1}$), *i*.

```

110 \newcommand{\I}{i}
```

\E Euler’s constant and the base of the natural logarithm, *e*.

```

111 \newcommand{\E}{e}
```

\PI Pi—the ratio of the circumference of a circle to its diameter, π .

```

112 \newcommand{\PI}{\pi}
```

\GoldenRatio The Golden Ratio, φ

```

113 \newcommand{\GoldenRatio}{\varphi}
```

\EulerGamma Euler’s Gamma constant, γ

```

114 \newcommand{\EulerGamma}{\gamma}
```

`\Catalan` Catalan constant, C
115 `\newcommand{\Catalan}{C}`

`\Glaisher` Glaisher constant, Glaisher
116 `\newcommand{\Glaisher}{\mathord{\operatorname{Glaisher}}}`

`\Khinchin` Khinchin constant, Khinchin
117 `\newcommand{\Khinchin}{\mathord{\operatorname{Khinchin}}}`

1.3.2 Symbols

`\Infinity` Infinity, ∞
118 `\newcommand{\Infinity}{\infty}`

`\Indeterminant` An indeterminant quantity
119 `\newcommand{\Indeterminant}{%`
120 `\mathchoice%`
121 `{\mbox{\textrm>}}%`
122 `{\mbox{\small>}}%`
123 `{\mbox{\scriptsize>}}%`
124 `{\mbox{\tiny>}}%`
125 `}`

`\DirectedInfinity` Directed Infinity `\DirectedInfinity{<complex number>}` or `\DirInfty{<complex number>}`
`\DirInfty`
126 `\newcommand{\DirectedInfinity}[1]{#1 \, \infty}`
127 `\newcommand{\DirInfty}[1]{\DirectedInfinity{#1}}`

`\ComplexInfinity` Complex infinity, ∞
`\CInfty` 128 `\newcommand{\ComplexInfinity}{\tilde{\infty}}`
129 `\newcommand{\CInfty}{\ComplexInfinity}`

1.3.3 Exponential Functions

`\Exp` Exponential—for use when e^x won't suffice, $\exp(x)$
130 `\newcommand{\COOL@notation@ExpParen}{p}`
131 `\newcommand{\Exp}[1]`
132 `{%`
133 `\exp\COOL@decide@paren{Exp}{#1}%`
134 `}`

`\Log` Logarithm, `\Log{x}`. This function has several options to be set. The usual parentheses, then some about the notation to be used for displaying the symbol.
135 `\newcommand{\COOL@notation@LogParen}{none}`

The following set the symbols:

`LogBaseESymb` can be `ln` or `log`, indicating what symbol should be used for the natural logarithm. If set to `log` then logarithms of base 10 are displayed as \log_{10} .

`LogShowBase` can be either `at will` or `always` and decides whether or not one should show the base, as in $\log_b x$. If this option is set to `always` then `LogBaseESymb` is ignored.

| | | |
|--|---------------|---------------|
| <code>\Log{5}</code> | $\ln 5$ | $\ln 5$ |
| <code>\Log[10]{5}</code> | $\log 5$ | $\log 5$ |
| <code>\Log[4]{5}</code> | $\log_4 5$ | $\log_4 5$ |
| <code>\Style{LogBaseESymb=log}</code> | | |
| <code>\Log{5}</code> | $\log 5$ | $\log 5$ |
| <code>\Log[10]{5}</code> | $\log_{10} 5$ | $\log_{10} 5$ |
| <code>\Log[4]{5}</code> | $\log_4 5$ | $\log_4 5$ |
| <code>\Style{LogShowBase=always}</code> | | |
| <code>\Log{5}</code> | $\log_e 5$ | $\log_e 5$ |
| <code>\Log[10]{5}</code> | $\log_{10} 5$ | $\log_{10} 5$ |
| <code>\Log[4]{5}</code> | $\log_4 5$ | $\log_4 5$ |
| <code>\Style{LogShowBase=at will}</code> | | |
| <code>\Log{5}</code> | $\ln 5$ | $\ln 5$ |
| <code>\Log[10]{5}</code> | $\log 5$ | $\log 5$ |
| <code>\Log[4]{5}</code> | $\log_4 5$ | $\log_4 5$ |
| <code>\Style{LogParen=p}</code> | | |
| <code>\Log[4]{5}</code> | $\log_4 (5)$ | $\log_4 (5)$ |

```

136 \newcommand{\COOL@notation@LogBaseESymb}{\ln}% 'ln', 'log'
137 \newcommand{\COOL@notation@LogShowBase}{at will}% 'at will', 'always'
138 \newcommand{\Log}[2][\E]
139 {%
140 \ifthenelse{ \equal{\COOL@notation@LogShowBase}{at will} }{%
141 {%
142 \ifthenelse{ \equal{#1}{\E} }{%
143 {%
144 \ifthenelse{ \equal{\COOL@notation@LogBaseESymb}{\ln} }{%
145 {%
146 \ln \COOL@decide@paren{Log}{#2}%
147 }%
148 % Else
149 {%
150 \ifthenelse{ \equal{\COOL@notation@LogBaseESymb}{log} }{%
151 {%
152 \log \COOL@decide@paren{Log}{#2}%
153 }%
154 % Else
155 {%
156 \PackageError{cool}{Invalid Option Sent}%
157 {LogBaseESymb can only be 'ln' or 'log'}%
158 }%

```

```

159 }%
160 }%
161 % Else
162 {%
163 \ifthenelse{ \equal{#1}{10} \AND
164 \NOT \equal{\COOL@notation@LogBaseESymb}{log} }{%
165 {%
166 \log \COOL@decide@paren{Log}{#2}%
167 }%
168 % Else
169 {%
170 \log_{#1} \COOL@decide@paren{Log}{#2}%
171 }%
172 }%
173 }%
174 % Else
175 {%
176 \ifthenelse{ \equal{\COOL@notation@LogShowBase}{always} }{%
177 {%
178 \log_{#1}\COOL@decide@paren{Log}{#2}%
179 }%
180 % Else
181 {%
182 \PackageError{cool}{Invalid Option Sent}%
183 {LogShowBase can only be 'at will' or 'always'}%
184 }%
185 }%
186 }

```

1.3.4 Trigonometric Functions

\Sin The sine function, $\text{\Sin{x}}$, $\sin(x)$

```

187 \newcommand{\COOL@notation@SinParen}{p}
188 \newcommand{\Sin}[1]{\sin\COOL@decide@paren{Sin}{#1}}

```

\Cos The cosine function, $\text{\Cos{x}}$, $\cos(x)$

```

189 \newcommand{\COOL@notation@CosParen}{p}
190 \newcommand{\Cos}[1]{\cos\COOL@decide@paren{Cos}{#1}}

```

\Tan The tangent function, $\text{\Tan{x}}$, $\tan(x)$

```

191 \newcommand{\COOL@notation@TanParen}{p}
192 \newcommand{\Tan}[1]{\tan\COOL@decide@paren{Tan}{#1}}

```

\Csc The cosecant function, $\text{\Csc{x}}$, $\csc(x)$

```

193 \newcommand{\COOL@notation@CscParen}{p}
194 \newcommand{\Csc}[1]{\csc\COOL@decide@paren{Csc}{#1}}

```

\Sec The secant function, $\text{\Sec{x}}$, $\sec(x)$

```

195 \newcommand{\COOL@notation@SecParen}{p}
196 \newcommand{\Sec}[1]{\sec\COOL@decide@paren{Sec}{#1}}

```

`\Cot` The cotangent function, `\Cot{x}`, $\cot(x)$

```
197 \newcommand{\COOL@notation@CotParen}{p}
198 \newcommand{\Cot}[1]{\cot\COOL@decide@paren{Cot}{#1}}
```

1.3.5 Inverse Trigonometric Functions

`\COOL@notation@ArcTrig` The inverse trigonometric functions style is governed by this global key. It's options are

`inverse` (default), this displays as \sin^{-1}
`arc`, this displays as `arcsin`

```
199 \def\COOL@notation@ArcTrig{inverse}
```

`\ArcSin` The inverse of the sine function, `\ArcSin{x}`, $\sin^{-1}(x)$

```
200 \newcommand{\COOL@notation@ArcSinParen}{p}
201 \newcommand{\ArcSin}[1]{%
202 \ifthenelse{ \equal{\COOL@notation@ArcTrig}{inverse} }{%
203 {%
204 \sin^{-1}\COOL@decide@paren{ArcSin}{#1}%
205 }
206 % else
207 {
208 \ifthenelse{\equal{\COOL@notation@ArcTrig}{arc}}{%
209 {%
210 \arcsin\COOL@decide@paren{ArcSin}{#1}%
211 }%
212 % else
213 {%
214 \PackageError{cool}{Invalid option sent}{}%
215 }%
216 }%
217 }
```

`\ArcCos` the inverse of the cosine function, `\ArcCos{x}`, $\cos^{-1}(x)$

```
218 \newcommand{\COOL@notation@ArcCosParen}{p}
219 \newcommand{\ArcCos}[1]{%
220 \ifthenelse{ \equal{\COOL@notation@ArcTrig}{inverse} }{%
221 {%
222 \cos^{-1}\COOL@decide@paren{ArcCos}{#1}%
223 }%
224 % else
225 {%
226 \ifthenelse{\equal{\COOL@notation@ArcTrig}{arc}}{%
227 {%
228 \arccos\COOL@decide@paren{ArcCos}{#1}%
229 }%
230 % else
231 {%
232 \PackageError{cool}{Invalid option sent}{}%
233 }%
```

```

234 }%
235 }

\ArcTan The inverse of the tangent function, \ArcTan{x},  $\tan^{-1}(x)$ 
236 \newcommand{\COOL@notation@ArcTanParen}{p}
237 \newcommand{\ArcTan}[1]{%
238 \ifthenelse{ \equal{\COOL@notation@ArcTrig}{inverse} }{%
239 {%
240 \tan^{-1}\COOL@decide@paren{ArcTan}{#1}%
241 }%
242 % else
243 {%
244 \ifthenelse{\equal{\COOL@notation@ArcTrig}{arc}}{%
245 {%
246 \arctan\COOL@decide@paren{ArcTan}{#1}%
247 }%
248 % else
249 {%
250 \PackageError{cool}{Invalid option sent}{}%
251 }%
252 }%
253 }

\ArcCsc The Inverse Cosecant function, \ArcCsc{x},  $\csc^{-1}(x)$ 
254 \newcommand{\COOL@notation@ArcCscParen}{p}
255 \newcommand{\ArcCsc}[1]{\csc^{-1}\COOL@decide@paren{ArcCsc}{#1}}

\ArcSec The inverse secant function, \ArcSec{x},  $\sec^{-1}(x)$ 
256 \newcommand{\COOL@notation@ArcSecParen}{p}
257 \newcommand{\ArcSec}[1]{\sec^{-1}\COOL@decide@paren{ArcSec}{#1}}

\ArcCot The inverse cotangent function, \ArcCot{x},  $\cot^{-1}(x)$ 
258 \newcommand{\COOL@notation@ArcCotParen}{p}
259 \newcommand{\ArcCot}[1]{\cot^{-1}\COOL@decide@paren{ArcCot}{#1}}

```

1.3.6 Hyperbolic Functions

```

\Sinh Hyperbolic sine, \Sinh{x},  $\sinh(x)$ 
260 \newcommand{\COOL@notation@SinhParen}{p}
261 \newcommand{\Sinh}[1]{\sinh\COOL@decide@paren{Sinh}{#1}}

\Cosh Hyperbolic cosine, \Cosh{x},  $\cosh(x)$ 
262 \newcommand{\COOL@notation@CoshParen}{p}
263 \newcommand{\Cosh}[1]{\cosh\COOL@decide@paren{Cosh}{#1}}

\tanh Hyperbolic Tangent, \Tanh{x},  $\tanh(x)$ 
264 \newcommand{\COOL@notation@TanhParen}{p}
265 \newcommand{\Tanh}[1]{\tanh\COOL@decide@paren{Tanh}{#1}}

```

`\Csch` Hyperbolic cosecant `\Csch{x}`, $\operatorname{csch}(x)$
266 `\newcommand{\COOL@notation@CschParen}{p}`
267 `\DeclareMathOperator{\csch}{csch}`
268 `\newcommand{\Csch}[1]{\csch\COOL@decide@paren{Csch}{#1}}`

`\Sech` Hyperbolic secant, `\Sech{x}`, $\operatorname{sech}(x)$
269 `\newcommand{\COOL@notation@SechParen}{p}`
270 `\DeclareMathOperator{\sech}{sech}`
271 `\newcommand{\Sech}[1]{\sech\COOL@decide@paren{Sech}{#1}}`

`\Coth` Hyperbolic Cotangent, `\Coth{x}`, $\operatorname{coth}(x)$
272 `\newcommand{\COOL@notation@CothParen}{p}`
273 `\newcommand{\Coth}[1]{\coth\COOL@decide@paren{Coth}{#1}}`

1.3.7 Inverse Hyperbolic Functions

`\ArcSinh` Inverse hyperbolic sine, `\ArcSinh{x}`, $\sinh^{-1}(x)$
274 `\newcommand{\COOL@notation@ArcSinhParen}{p}`
275 `\newcommand{\ArcSinh}[1]{\sinh^{-1}\COOL@decide@paren{ArcSinh}{#1}}`

`\ArcCosh` Inverse hyperbolic cosine, `\ArcCosh{x}`, $\cosh^{-1}(x)$
276 `\newcommand{\COOL@notation@ArcCoshParen}{p}`
277 `\newcommand{\ArcCosh}[1]{\cosh^{-1}\COOL@decide@paren{ArcCosh}{#1}}`

`\ArcTanh` Inverse hyperbolic tangent, `\ArcTanh{x}`, $\tanh^{-1}(x)$
278 `\newcommand{\COOL@notation@ArcTanhParen}{p}`
279 `\newcommand{\ArcTanh}[1]{\tanh^{-1}\COOL@decide@paren{ArcTanh}{#1}}`

`\ArcCsch` Inverse hyperbolic cosecant, `\ArcCsch{x}`, $\operatorname{csch}^{-1}(x)$
280 `\newcommand{\COOL@notation@ArcCschParen}{p}`
281 `\newcommand{\ArcCsch}[1]{\csch^{-1}\COOL@decide@paren{ArcCsch}{#1}}`

`\ArcSech` Inverse hyperbolic secant, `\ArcSech{x}`, $\operatorname{sech}^{-1}(x)$
282 `\newcommand{\COOL@notation@ArcSechParen}{p}`
283 `\newcommand{\ArcSech}[1]{\sech^{-1}\COOL@decide@paren{ArcSech}{#1}}`

`\ArcCoth` Inverse hyperbolic cotangent, `\ArcCoth{x}`, $\operatorname{coth}^{-1}(x)$
284 `\newcommand{\COOL@notation@ArcCothParen}{p}`
285 `\newcommand{\ArcCoth}[1]{\coth^{-1}\COOL@decide@paren{ArcCoth}{#1}}`

1.3.8 Product Logarithms

`\LambertW` Lambert Function. `\LambertW` is an alias for `\ProductLog` and its properties are therefore set using that function
286 `\newcommand{\LambertW}[1]{\ProductLog{#1}}`

```

\ProductLog  Generalized Lambert Function \ProductLog{[\langle index \rangle],[\langle variable \rangle]}.
              Lambert Function           \ProductLog{x}       $W(x)$ 
              Generalized Lambert Function \ProductLog{k,x}     $W_k(x)$ 

287 \newcommand{\COOL@notation@ProductLogParen}{p}
288 \newcommand{\ProductLog}[1]{%
289 \listval{#1}{0}%
290 \ifthenelse{\value{COOL@listpointer}=1}%
291 {%
292 W\!\COOL@decide@paren{ProductLog}{#1}%
293 }%
294 % else
295 {%
296 \ifthenelse{\value{COOL@listpointer}=2}%
297 {%
298 W_{\listval{#1}{1}}\!\COOL@decide@paren{ProductLog}{\listval{#1}{2}}%
299 }%
300 % else
301 {%
302 \PackageError{cool}{‘ProductLog’ Invaoid Argument}%
303 {Must have a comma separated list of length 1 or 2}
304 }%
305 }%
306 }

```

1.3.9 Max and Min

```

\Max  the maximum function, \Max{x,y,z},  $\max(x,y,z)$ 
307 \newcommand{\COOL@notation@MaxParen}{p}
308 \newcommand{\Max}[1]{\max\COOL@decide@paren{Max}{#1}}

\Min  the minimum function, \Min{x,y,z},  $\min(x,y,z)$ 
309 \newcommand{\COOL@notation@MinParen}{p}
310 \newcommand{\Min}[1]{\min\COOL@decide@paren{Min}{#1}}

```

1.3.10 Bessel Functions

```

\BesselJ  Bessel Function of the first kind, \BesselJ{\nu}{x},  $J_\nu(x)$ 
311 \newcommand{\COOL@notation@BesselJSymb}{J}
312 \newcommand{\COOL@notation@BesselJParen}{p}
313 \newcommand{\BesselJ}[2]%
314 {\COOL@notation@BesselJSymb_{#1}\!\COOL@decide@paren{BesselJ}{#2}}

\BesselY  Bessel Function of the second kind, \BesselY{\nu}{x},  $Y_\nu(x)$ 
315 \newcommand{\COOL@notation@BesselYSymb}{Y}
316 \newcommand{\COOL@notation@BesselYParen}{p}
317 \newcommand{\BesselY}[2]%
318 {\COOL@notation@BesselYSymb_{#1}\!\COOL@decide@paren{BesselY}{#2}}

```

`\BesselI` Modified Bessel Function of the first kind, `\BesselI{\nu}{x}`, $I_\nu(x)$

```

319 \newcommand{\COOL@notation@BesselISymb}{I}
320 \newcommand{\COOL@notation@BesselIParen}{p}
321 \newcommand{\BesselI}[2]%
322 {\COOL@notation@BesselISymb_{#1}\!\COOL@decide@paren{BesselI}{#2}}

```

`\BesselK` Modified Bessel Function of the second kind, `\BesselK{\nu}{x}`, $K_\nu(x)$

```

323 \newcommand{\COOL@notation@BesselKSymb}{K}
324 \newcommand{\COOL@notation@BesselKParen}{p}
325 \newcommand{\BesselK}[2]%
326 {\COOL@notation@BesselKSymb_{#1}\!\COOL@decide@paren{BesselK}{#2}}

```

1.3.11 Airy Functions

`\AiryAi` Airy Ai Function, `\AiryAi{x}`, $\text{Ai}(x)$

```

327 \newcommand{\COOL@notation@AiryAiParen}{p}
328 \DeclareMathOperator{\AiryAiSymb}{Ai}
329 \newcommand{\AiryAi}[1]{\AiryAiSymb\!\COOL@decide@paren{AiryAi}{#1}}

```

`\AiryBi` Airy Bi Function, `\AiryBi{x}`, $\text{Bi}(x)$

```

330 \newcommand{\COOL@notation@AiryBiParen}{p}
331 \DeclareMathOperator{\AiryBiSymb}{Bi}
332 \newcommand{\AiryBi}[1]{\AiryBiSymb\!\COOL@decide@paren{AiryBi}{#1}}

```

1.3.12 Struve Functions

`\StruveH` Struve H function, `\StruveH{\nu}{z}`, $\text{H}_\nu(z)$

```

333 \newcommand{\COOL@notation@StruveHParen}{p}
334 \newcommand{\StruveH}[2]{\bf H_{#1}\!\COOL@decide@paren{StruveH}{#2}}

```

`\StruveL` Struve L function, `\StruveL{\nu}{z}`, $\text{L}_\nu(z)$

```

335 \newcommand{\COOL@notation@StruveLParen}{p}
336 \newcommand{\StruveL}[2]{\bf L_{#1}\!\COOL@decide@paren{StruveL}{#2}}

```

1.3.13 Integer Functions

`\Floor` floor, `\Floor{x}`, $\lfloor x \rfloor$

```

337 \newcommand{\Floor}[1]{\lfloor #1 \rfloor}

```

`\Ceiling` ceiling, `\Ceiling{x}`, $\lceil x \rceil$

```

338 \newcommand{\Ceiling}[1]{\lceil #1 \rceil}

```

`\Round` round, `\Round{x}`, $\lfloor x \rfloor$

```

339 \newcommand{\Round}[1]{\lfloor #1 \rfloor}

```

```

\iPart    The integer part of a real number, \iPart{x}, \IntegerPart{x}, int(x)
\IntegerPart 340 \newcommand{\COOL@notation@IntegerPartParen}{p}
              341 \DeclareMathOperator{\iPartSymb}{int}
              342 \newcommand{\iPart}[1]{\iPartSymb\COOL@decide@paren{IntegerPart}{#1}}
              343 \newcommand{\IntegerPart}[1]{\iPart{#1}}

\fracPart  the fractional part of a real number, \fracPart{x}, \FractionalPart{x}, frac(x)
\FractionalPart 344 \newcommand{\COOL@notation@FractionalPartParen}{p}
                 345 \DeclareMathOperator{\fracPartSymb}{frac}
                 346 \newcommand{\fracPart}[1]{\fracPartSymb\COOL@decide@paren{FractionalPart}{#1}}
                 347 \newcommand{\FractionalPart}[1]{\fracPart{#1}}

\Mod      Modulo, \Mod{n}{m},  $n \bmod m$ 
           348 \newcommand{\COOL@notation@ModDisplay}{mod}
           349 \newcommand{\Mod}[2]{%
           350 \ifthenelse{\equal{\COOL@notation@ModDisplay}{mod}}%
           351 {%
           352 #1 \mod #2%
           353 }%
           354 % ElseIf
           355 { \ifthenelse{\equal{\COOL@notation@ModDisplay}{bmod}}%
           356 {%
           357 #1 \bmod #2%
           358 }%
           359 % ElseIf
           360 { \ifthenelse{\equal{\COOL@notation@ModDisplay}{pmod}}%
           361 {%
           362 #1 \pmod #2%
           363 }%
           364 % ElseIf
           365 {\ifthenelse{\equal{\COOL@notation@ModDisplay}{pod}}%
           366 {%
           367 #1 \pod #2%
           368 }%
           369 % Else
           370 {%
           371 \PackageError{cool}{Invalid Option Sent}%
           372 {ModDisplay can only be 'mod', 'bmod', 'pmod', or 'pod'}}%
           373 }}}}
           374 }

\Quotient  quotient, \Quotient{m}{n}, quotient(m,n)
           375 \newcommand{\COOL@notation@QuotientParen}{p}
           376 \DeclareMathOperator{\QuotientSymb}{quotient}
           377 \newcommand{\Quotient}[2]%
           378 {\QuotientSymb\COOL@decide@paren{Quotient}{#1,#2}}

\GCD      greatest common divisor, \GCD{n_1,n_2,\dots,n_m}, gcd(n_1,n_2,\dots,n_m)
           379 \newcommand{\COOL@notation@GCDParen}{p}
           380 \newcommand{\GCD}[1]{\gcd\COOL@decide@paren{GCD}{#1}}

```

```

\ExtendedGCD Extended Greatest Common Divisor,
\EGCD \EGCD{n}{m}, \ExtendedGCD{n}{m}, egcd( $n, m$ )
381 \newcommand{\COOL@notation@ExtendedGCDParen}{p}
382 \DeclareMathOperator{\ExtendedGCDSymb}{egcd}
383 \newcommand{\ExtendedGCD}[2]%
384 {\ExtendedGCDSymb\COOL@decide@paren{ExtendedGCD}{#1,#2}}
385 \newcommand{\EGCD}[2]{\ExtendedGCD{#1}{#2}}

\LCM Least Common Multiple, \LCM{n_1,n_2,\ldots,n_m}, lcm( $n_1, n_2, \dots, n_m$ )
386 \newcommand{\COOL@notation@LCMParen}{p}
387 \DeclareMathOperator{\LCMSymb}{lcm}
388 \newcommand{\LCM}[1]{\LCMSymb\COOL@decide@paren{LCM}{#1}}

\Fibonacci Fibonacci number, \Fibonacci{n},  $F_n$ , and
Fibonacci Polynomial, \Fibonacci{n,x},  $F_n(x)$ 
389 \newcommand{\COOL@notation@FibonacciParen}{p}
390 \newcommand{\Fibonacci}[1]{%
391 \liststore{#1}{\COOL@Fibonacci@arg@}%
392 \listval{#1}{0}%
393 \ifthenelse{\value{COOL@listpointer} = 1}%
394 {%
395 F_{#1}%
396 }%
397 % ElseIf
398 { \ifthenelse{\value{COOL@listpointer} = 2}%
399 {%
400 F_{\COOL@Fibonacci@arg@i}\!%
401 \COOL@decide@paren{Fibonacci}{\COOL@Fibonacci@arg@ii}%
402 }%
403 % Else
404 {%
405 \PackageError{cool}{Invalid Argument}%
406 {'Fibonacci' can only accept a
407 comma separate list of length 1 or 2}%
408 }%
409 }

\Euler Euler number, \Euler{n},  $E_n$ , and Euler Polynomial, \Euler{n,x},  $E_n(x)$ 
410 \newcommand{\COOL@notation@EulerParen}{p}
411 \newcommand{\Euler}[1]{%
412 \liststore{#1}{\COOL@Euler@arg@}%
413 \listval{#1}{0}%
414 \ifthenelse{\value{COOL@listpointer} = 1}%
415 {%
416 E_{#1}%
417 }%
418 % ElseIf
419 { \ifthenelse{\value{COOL@listpointer} = 2}%
420 {%

```

```

421 E_{\COOL@Euler@arg@i}\!%
422 \COOL@decide@paren{Euler}{\COOL@Euler@arg@ii}%
423 }%
424 % Else
425 {%
426 \PackageError{cool}{Invalid Argument}%
427 {'Euler' can only accept a
428 comma separate list of length 1 or 2}%
429 }}%
430 }

\Bernoulli Bernoulli number, \Bernoulli{n},  $B_n$  and
Bernoulli Polynomial \Bernoulli{n,x},  $B_n(x)$ 
431 \newcommand{\COOL@notation@BernoulliParen}{p}
432 \newcommand{\Bernoulli}[1]{%
433 \liststore{#1}{COOL@Bernoulli@arg@}%
434 \listval{#1}{0}%
435 \ifthenelse{\value{COOL@listpointer} = 1}%
436 {%
437 B_{#1}%
438 }%
439 % ElseIf
440 { \ifthenelse{\value{COOL@listpointer} = 2}%
441 {%
442 B_{\COOL@Bernoulli@arg@i}\!%
443 \COOL@decide@paren{Bernoulli}{\COOL@Bernoulli@arg@ii}%
444 }%
445 % Else
446 {%
447 \PackageError{cool}{Invalid Argument}%
448 {'Bernoulli' can only accept a
449 comma separate list of length 1 or 2}%
450 }}%
451 }

\StirlingSOne Stirling number of the first kind \StirlingSOne{n}{m},  $S_n^{(m)}$ 
452 \newcommand{\StirlingSOne}[2]{S_{#1}^{\inp{#2}}}}

\StirlingSTwo Stirling number of the second kind, \StirlingSTwo{n}{m},  $S_n^{(m)}$ 
453 \newcommand{\StirlingSTwo}[2]{\cal S}_{#1}^{\inp{#2}}}}

\PartitionsP Number of unrestricted partitions of an integer, \PartitionsP{x},  $p(x)$ 
454 \newcommand{\COOL@notation@PartitionsPParen}{p}
455 \newcommand{\PartitionsP}[1]{p\!\!\COOL@decide@paren{PartitionsP}{#1}}

\PartitionsQ number of partitions of an integer into distinct parts, \PartitionsQ{x},  $q(x)$ 
456 \newcommand{\COOL@notation@PartitionsQParen}{p}
457 \newcommand{\PartitionsQ}[1]{q\!\!\COOL@decide@paren{PartitionsQ}{#1}}

```

```

\DiscreteDelta Discrete delta function,
                \DiscreteDelta{n_1,n_2,\ldots,n_m},  $\delta(n_1,n_2,\dots,n_m)$ 
458 \newcommand{\COOL@notation@DiscreteDeltaParen}{p}
459 \newcommand{\DiscreteDelta}[1]%
460 {\delta\COOL@decide@paren{DiscreteDelta}{#1}}

\KroneckerDelta Kronecker Delta, \KroneckerDelta{n_1,n_2,\ldots,n_m},  $\delta^{n_1n_2\dots n_m}$ 
461 \newcommand{\COOL@notation@KroneckerDeltaUseComma}{false}%
462 \newcommand{\COOL@notation@KroneckerDeltaIndicies}{local}
463 \newcommand{\KroneckerDelta}[2][u]{%
464 \liststore{#2}{\COOL@arg@}%
465 \listval{#2}{0}%
466 \def\COOL@arg@temp{}%
467 \forLoop{1}{\value{\COOL@listpointer}}{\COOL@ct}%
468 {%
469 \ifthenelse{\equal{\COOL@notation@KroneckerDeltaUseComma}{true}}{%
470 {%
471 \ifthenelse{\NOT \value{\COOL@ct} = 1}
472 {%
473 \edef\COOL@arg@temp%
474 {\COOL@arg@temp, \csname \COOL@arg@\roman{\COOL@ct}\endcsname}%
475 }%
476 % Else
477 {%
478 \edef\COOL@arg@temp%
479 {\COOL@arg@temp \csname \COOL@arg@\roman{\COOL@ct}\endcsname}%
480 }%
481 }%
482 % Else
483 {%
484 \edef\COOL@arg@temp%
485 {\COOL@arg@temp \csname \COOL@arg@\roman{\COOL@ct}\endcsname}%
486 }%
487 }%
488 \delta\COOL@decide@indicies{KroneckerDelta}{#1}{\COOL@arg@temp}%
489 }

\LeviCivita Levi-Civita totally anti-symmetric Tensor density,
\Signature \LeviCivita{n_1,n_2,\ldots,n_m},  $\epsilon^{n_1n_2\dots n_m}$ 
490 \newcommand{\COOL@notation@LeviCivitaUseComma}{false}
491 \newcommand{\COOL@notation@LeviCivitaIndicies}{local}
492 \newcommand{\LeviCivita}[2][u]{%
493 \liststore{#2}{\COOL@arg@}%
494 \listval{#2}{0}%
495 \def\COOL@arg@temp{}%
496 \forLoop{1}{\value{\COOL@listpointer}}{\COOL@ct}%
497 {%
498 \ifthenelse{\equal{\COOL@notation@LeviCivitaUseComma}{true}}{%
499 {%

```

```

500 \ifthenelse{\NOT \value{COOL@ct} = 1}{,}{}%
501 \edef\COOL@arg@temp%
502 {\COOL@arg@temp \csname COOL@arg@\roman{COOL@ct}\endcsname}%
503 }%
504 % Else
505 {%
506 \edef\COOL@arg@temp%
507 {\COOL@arg@temp \csname COOL@arg@\roman{COOL@ct}\endcsname}%
508 }%
509 }%
510 \epsilon\COOL@decide@indicies{LeviCivita}{#1}{\COOL@arg@temp}%
511 }%
512 \newcommand{\Signature}[2][u]{\LeviCivita[#1]{#2}}

```

1.3.14 Classical Orthogonal Polynomials

\HermiteH Hermite Polynomial, $\backslash\text{HermiteH}\{n\}\{x\}$, $H_n(x)$

```

513 \newcommand{\COOL@notation@HermiteHParen}{p}
514 \newcommand{\COOL@notation@HermiteHSymb}{H}
515 \newcommand{\HermiteH}[2]%
516 {\COOL@notation@HermiteHSymb_{#1}\!\COOL@decide@paren{HermiteH}{#2}}

```

\LagerreL Lagerre Polynomial, $\backslash\text{LagerreL}\{\nu,x\}$, $L_\nu(x)$ and
Generalized Lagerre Polynomial $\backslash\text{LagerreL}\{\nu,\lambda,x\}$, $L_\nu^\lambda(x)$

```

517 \newcommand{\COOL@notation@LagerreLParen}{p}
518 \newcommand{\COOL@notation@LagerreLSymb}{L}
519 \newcommand{\LagerreL}[1]{%
520 \liststore{#1}{COOL@list@temp@}%
521 \listval{#1}{0}%
522 \ifthenelse{\value{COOL@listpointer}=2}%
523 {%
524 \COOL@notation@LagerreLSymb_{\COOL@list@temp@i}%
525 \!\COOL@decide@paren{LagerreL}{\COOL@list@temp@ii}%
526 }%
527 % Else If
528 { \ifthenelse{\value{COOL@listpointer}=3}%
529 {%
530 \COOL@notation@LagerreLSymb_{\COOL@list@temp@i}^{\COOL@list@temp@ii}%
531 \!\COOL@decide@paren{LagerreL}{\COOL@list@temp@iii}%
532 }%
533 % Else
534 {%
535 \PackageError{cool}{Invalid Argument}%
536 {'LagerrL' only accepts a comma separated list of length 2 or 3}%
537 }%
538 }

```

\LegendreP Legendre Polynomials

| | | |
|--------------------------------|--|-------------------------|
| Legendre Polynomial | $\backslash\text{LegendreP}\{n,x\}$ | $P_n(x)$ |
| Associated Legendre Polynomial | | |
| of the first kind of type 2 | $\backslash\text{LegendreP}\{\text{ell},m,x\}$ | $P_\ell^m(x)$ |
| | $\backslash\text{LegendreP}\{\text{ell},m,2,x\}$ | $P_\ell^m(x)$ |
| Associated Legendre Function | | |
| of the first kind of type 3 | $\backslash\text{LegendreP}\{\text{ell},m,3,x\}$ | $\mathcal{P}_\ell^m(x)$ |

```

539 \newcommand{\COOL@notation@LegendrePParen}{p}
540 \newcommand{\COOL@notation@LegendrePSymb}{P}
541 \newcommand{\LegendreP}[1]{%
542 \liststore{#1}{\COOL@LegendreP@arg@}%
543 \listval{#1}{0}%
544 \ifthenelse{\value{\COOL@listpointer} = 2}%
545 {%
546 \COOL@notation@LegendrePSymb_{\COOL@LegendreP@arg@i}\!%
547 \COOL@decide@paren{LegendreP}{\COOL@LegendreP@arg@ii}%
548 }%
549 % ElseIf
550 { \ifthenelse{\value{\COOL@listpointer} = 3}%
551 {%
552 \COOL@notation@LegendrePSymb_{\COOL@LegendreP@arg@i}%
553 ^{\COOL@LegendreP@arg@ii}\!%
554 \COOL@decide@paren{LegendreP}{\COOL@LegendreP@arg@iii}%
555 }%
556 % ElseIf
557 { \ifthenelse{\value{\COOL@listpointer} = 4}%
558 {%
559 \isint{\COOL@LegendreP@arg@iii}%
560 \ifthenelse{\boolean{\COOL@isint}}{%
561 {%
562 \ifcase\COOL@LegendreP@arg@iii\relax%
563 \PackageError{cool}{Invalid Argument}%
564 {'LegendreP' third argument must be $>$ 1}%
565 \or%
566 \PackageError{cool}{Invalid Argument}%
567 {'LegendreP' third argument must be $>$ 1}%
568 \or%
569 \COOL@notation@LegendrePSymb_{\COOL@LegendreP@arg@i}%
570 ^{\COOL@LegendreP@arg@ii}\!%
571 \COOL@decide@paren{LegendreP}{\COOL@LegendreP@arg@iv}%
572 \or%
573 {\cal P}_{\COOL@LegendreP@arg@i}%
574 ^{\COOL@LegendreP@arg@ii}\!%
575 \COOL@decide@paren{LegendreP}{\COOL@LegendreP@arg@iv}%
576 \else%
577 \PackageError{cool}{Invalid Argument}{unsupported}%
578 \fi%
579 }
580 % Else
581 {%

```

```

582 \PackageError{cool}{Invalid Argument}{third arg must be int}%
583 }%
584 }%
585 % Else
586 {%
587 \PackageError{cool}{Invalid Argument}%
588 {'LegendreP' can only accept a%
589 comma separated list of length 2-4}%
590 }}%
591 }

\LegendreQ Legendre Polynomials of the second kind
Legendre Polynomial \LegendreQ{n,x}  $Q_n(x)$ 
Associated Legendre Polynomial
of the second kind of type 2 \LegendreQ{ell,m,x}  $Q_\ell^m(x)$ 
\LegendreQ{ell,m,2,x}  $Q_\ell^m(x)$ 
Associated Legendre Function
of the second kind of type 3 \LegendreQ{ell,m,3,x}  $Q_\ell^m(x)$ 
592 \newcommand{\COOL@notation@LegendreQParen}{p}
593 \newcommand{\COOL@notation@LegendreQSymb}{Q}
594 \newcommand{\LegendreQ}[1]{%
595 \liststore{#1}{COOL@LegendreQ@arg@}%
596 \listval{#1}{0}%
597 \ifthenelse{\value{COOL@listpointer} = 2}%
598 {%
599 \COOL@notation@LegendreQSymb_{\COOL@LegendreQ@arg@i}\!%
600 \COOL@decide@paren{LegendreQ}{\COOL@LegendreQ@arg@ii}%
601 }%
602 % ElseIf
603 { \ifthenelse{\value{COOL@listpointer} = 3}%
604 {%
605 \COOL@notation@LegendreQSymb_{\COOL@LegendreQ@arg@i}%
606 ~{\COOL@LegendreQ@arg@ii}\!%
607 \COOL@decide@paren{LegendreQ}{\COOL@LegendreQ@arg@iii}%
608 }%
609 % ElseIf
610 { \ifthenelse{\value{COOL@listpointer} = 4}%
611 {%
612 \isint{\COOL@LegendreQ@arg@iii}%
613 \ifthenelse{\boolean{COOL@isint}}%
614 {%
615 \ifcase\COOL@LegendreQ@arg@iii\relax%
616 \PackageError{cool}{Invalid Argument}%
617 {'LegendreQ' third argument must be $>$ 1}%
618 \or%
619 \PackageError{cool}{Invalid Argument}%
620 {'LegendreQ' third argument must be $>$ 1}%
621 \or%
622 \COOL@notation@LegendreQSymb_{\COOL@LegendreQ@arg@i}%

```

```

623 ~{\COOL@LegendreQ@arg@ii}\!%
624 \COOL@decide@paren{LegendreQ}{\COOL@LegendreQ@arg@iv}%
625 \or%
626 {\cal Q}_{\COOL@LegendreQ@arg@i}%
627 ~{\COOL@LegendreQ@arg@ii}\!%
628 \COOL@decide@paren{LegendreQ}{\COOL@LegendreQ@arg@iv}%
629 \else%
630 \PackageError{cool}{Invalid Argument}{unsupported}%
631 \fi%
632 }
633 % Else
634 {%
635 \PackageError{cool}{Invalid Argument}{third arg must be int}%
636 }%
637 }%
638 % Else
639 {%
640 \PackageError{cool}{Invalid Argument}%
641 {'LegendreQ' can only accept a%
642 comma separated list of length 2-4}%
643 }}%
644 }

\ChebyshevT Chebyshev Polynomial of the first kind, \ChebyshevT{n}{x}, ChebyshevTnx
645 \newcommand{\COOL@notation@ChebyshevTParen}{p}
646 \newcommand{\COOL@notation@ChebyshevTSymb}{T}
647 \newcommand{\ChebyshevT}[2]%
648 {\COOL@notation@ChebyshevTSymb_{#1}\!\COOL@decide@paren{ChebyshevT}{#2}}

\ChebyshevU , \ChebyshevU{n}{z},  $U_n(z)$  Chebyshev Polynomial of the second kind
649 \newcommand{\COOL@notation@ChebyshevUParen}{p}
650 \newcommand{\COOL@notation@ChebyshevUSymb}{U}
651 \newcommand{\ChebyshevU}[2]%
652 {\COOL@notation@ChebyshevUSymb_{#1}\!\COOL@decide@paren{ChebyshevU}{#2}}

\JacobiP Jacobi Polynomial, \JacobiP{n}{a}{b}{x},  $P_n^{(a,b)}(x)$ 
653 \newcommand{\COOL@notation@JacobiPParen}{p}
654 \newcommand{\COOL@notation@JacobiPSymb}{P}
655 \newcommand{\JacobiP}[4]{%
656 \COOL@notation@JacobiPSymb_{#1}^{\in{#2, #3}}%
657 \!\COOL@decide@paren{JacobiP}{#4}%
658 }

```

1.3.15 Associated Polynomials

\AssocLegendreP Associated Legendre Polynomial of the first kind of type 2
 $\text{\AssocLegendreP}\{\ell\}{m}\{x\}$, $P_\ell^m(x)$
659 \newcommand{\AssocLegendreP}[3]{\LegendreP{#1,#2,#3}}

`\AssocLegendreQ` Associated Legendre Polynomial of the second kind of type 2
`\AssocLegendreQ{\ell}{m}{x}`, $Q_\ell^m(x)$
660 `\newcommand{\AssocLegendreQ}[3]{\LegendreQ{#1,#2,#3}}`

`\GegenbauerC` Gegenbauer Polynomial, `\GegenbauerC{n}{\lambda}{x}`, $C_n^\lambda(x)$
661 `\newcommand{\COOL@notation@GegenbauerCParen}{p}`
662 `\newcommand{\COOL@notation@GegenbauerCSymb}{C}`
663 `\newcommand{\GegenbauerC}[3]{%`
664 `\COOL@notation@GegenbauerCSymb_{#1}^{#2}%`
665 `\!\COOL@decide@paren{GegenbauerC}{#3}%`
666 `}`

`\SphericalHarmonicY` Spherical Harmonic, `\SpHarmY{\ell}{m}{\theta}{\phi}`,
`\SphericalHarmY` `\SphericalHarmY{\ell}{m}{\theta}{\phi}`,
`\SpHarmY` `\SphericalHarmonicY{\ell}{m}{\theta}{\phi}`, $Y_\ell^m(\theta, \phi)$
667 `\newcommand{\COOL@notation@SphericalHarmonicYParen}{p}`
668 `\newcommand{\COOL@notation@SphericalHarmonicYSymb}{Y}`
669 `\newcommand{\SphericalHarmonicY}[4]{%`
670 `\COOL@notation@SphericalHarmonicYSymb_{#1}^{#2}%`
671 `\!\COOL@decide@paren{SphericalHarmonicY}{#3,#4}%`
672 `}`
673 `\newcommand{\SphericalHarmY}[4]{\SphericalHarmonicY{#1}{#2}{#3}{#4}}`
674 `\newcommand{\SpHarmY}[4]{\SphericalHarmonicY{#1}{#2}{#3}{#4}}`

1.3.16 Other Polynomials

`\CyclotomicC` Cyclotomic Polynomial, `\CyclotomicC{n}{z}`, $C_n(z)$
675 `\newcommand{\COOL@notation@CyclotomicCParen}{p}`
676 `\newcommand{\CyclotomicC}[2]{%`
677 `{C_{#1}}\!\COOL@decide@paren{CyclotomicC}{#2}%`

`\FibonacciF` Fibonacci Polynomial, `\FibonacciF{n}{z}`, $F_n(z)$
678 `\newcommand{\FibonacciF}[2]{\Fibonacci{#1,#2}}`

`\EulerE` Euler Polynomial, `\EulerE{n}{z}`, $E_n(z)$
679 `\newcommand{\EulerE}[2]{\Euler{#1,#2}}`

`\BernoulliB` Bernoulli Polynomial, `\BernoulliB{n}{z}`, $B_n(z)$
680 `\newcommand{\BernoulliB}[2]{\Bernoulli{#1,#2}}`

1.3.17 Factorial Functions

`\Factorial` Factorial, `\Factorial{n}`, $n!$
681 `\newcommand{\Factorial}[1]{#1!}`

`\DblFactorial` Double Factorial, `\DblFactorial{n}`, $n!!$
682 `\newcommand{\DblFactorial}[1]{#1!!}`

```

\Binomial binomial, \Binomial{n}{r},  $\binom{n}{r}$ 
683 \newcommand{\Binomial}[2]{ \binom{#1}{#2} }

\Multinomial Multinomial, \Multinomial{n_1,\ldots,n_m},  $(n_1 + \dots + n_m; n_1, \dots, n_m)$ 
684 \newcommand{\Multinomial}[1]%
685 {%
686 \listval{#1}{0}% get the length of the list
687 \setcounter{COOL@listlen}{\value{COOL@listpointer}}% record length
688 \liststore{#1}{COOL@list@temp@}%
689 \isint{\COOL@list@temp@i}% check that the entries are integers
690 \setcounter{COOL@ct}{2}%
691 \whiledo{ \boolean{COOL@isint} \AND
692 \NOT \value{COOL@ct}>\value{COOL@listlen} }{%
693 {%
694 \def\COOL@Multinomial@tempa%
695 {\csname COOL@list@temp@\roman{COOL@ct}\endcsname}%
696 \isint{\COOL@Multinomial@tempa}%
697 \stepcounter{COOL@ct}%
698 }%
699 \ifthenelse{\boolean{COOL@isint}}{%
700 {%
701 % all of them are integers
702 \setcounter{COOL@ct@}{ \COOL@list@temp@i }% records the sum
703 \forLoop{2}{\value{COOL@listlen}}{COOL@ct}%
704 {%
705 \addtocounter{COOL@ct@}%
706 {\csname COOL@list@temp@\roman{COOL@ct}\endcsname}%
707 }%
708 \left(\arabic{COOL@ct@}%
709 }%
710 % Else
711 {%
712 \left(%
713 \listval{#1}{1}%
714 \forLoop{2}{\value{COOL@listlen}}{COOL@ct}%
715 {%
716 + \listval{#1}{\arabic{COOL@ct}}}%
717 }%
718 }%
719 ;#1\right)%
720 }

```

1.3.18 Gamma Functions

| | | | |
|------------|---------------------------------------|-------------------|-------------------|
| \GammaFunc | Gamma Function | | |
| | Gamma Function | \GammaFunc{z} | $\Gamma(z)$ |
| | Incomplete Gamma Function | \GammaFunc{a,z} | $\Gamma(a, z)$ |
| | Generalized Incomplete Gamma Function | \GammaFunc{a,x,y} | $\Gamma(a, x, y)$ |

```

721 \newcommand{\COOL@notation@GammaFuncParen}{p}

```

```

722 \newcommand{\GammaFunc}[1]{%
723 \listval{#1}{0}%
724 \ifthenelse{\value{COOL@listpointer} = 1}%
725 {%
726 \Gamma\!\COOL@decide@paren{\GammaFunc}{#1}%
727 }%
728 % ElseIf
729 { \ifthenelse{\value{COOL@listpointer} = 2}%
730 {%
731 \Gamma\!\COOL@decide@paren{\GammaFunc}{#1}%
732 }%
733 % ElseIf
734 { \ifthenelse{\value{COOL@listpointer} = 3}%
735 {%
736 \Gamma\!\COOL@decide@paren{\GammaFunc}{#1}%
737 }%
738 % Else
739 {%
740 \PackageError{cool}{Invalid Argument}%
741 {'GammaFunc' can only accept a comma separate list of length 1 to 3}%
742 }%
743 }%
744 }

\IncGamma incomplete Gamma function, \IncGamma{a}{x},  $\Gamma(a, x)$ 
745 \newcommand{\IncGamma}[2]{\GammaFunc{#1, #2}}

\GenIncGamma Generalized Incomplete Gamma, \GenIncGamma{a}{x}{y},  $\Gamma(a, x, y)$ 
746 \newcommand{\GenIncGamma}[3]{\GammaFunc{#1, #2, #3}}

\GammaRegularized Regularized Incomplete Gamma
\RegIncGamma \GammaRegularized{a, x}  $Q(a, x)$ 
\GammaReg \RegIncGamma{a}{x}  $Q(a, x)$ 
\GammaReg{a, x}  $Q(a, x)$ 
747 \newcommand{\COOL@notation@GammaRegularizedParen}{p}%
748 \newcommand{\GammaRegularized}[1]{%
749 \listval{#1}{0}%
750 \ifthenelse{\value{COOL@listpointer} = 2}%
751 {%
752 Q\!\COOL@decide@paren{\GammaRegularized}{#1}%
753 }%
754 % ElseIf
755 { \ifthenelse{\value{COOL@listpointer} = 3}%
756 {%
757 Q\!\COOL@decide@paren{\GammaRegularized}{#1}%
758 }%
759 % Else
760 {%
761 \PackageError{cool}{Invalid Argument}%

```

```

762 {'GammaRegularized' can only accept comma%
763 separated lists of length 2 or 3}%
764 }%
765 }%
766 }
767 \newcommand{\RegIncGamma}[2]{\GammaRegularized{#1, #2}}
768 \newcommand{\GammaReg}[1]{\GammaRegularized{#1}}

\RegIncGammaInv Inverse of Regularized Incomplete Gamma,
\InverseGammaRegularized \RegIncGammaInv{a}{x}  $Q^{-1}(a, x)$ 
\GammaRegInv \InverseGammaRegularized{a, x}  $Q^{-1}(a, x)$ 
\GammaRegInv{a, x}  $Q^{-1}(a, x)$ 

769 \newcommand{\COOL@notation@InverseGammaRegularizedParen}{p}
770 \newcommand{\InverseGammaRegularized}[1]{%
771 \listval{#1}{0}%
772 \ifthenelse{\value{COOL@listpointer} = 2}%
773 {%
774  $Q^{-1}$ !\COOL@decide@paren{\InverseGammaRegularized}{#1}%
775 }%
776 % ElseIf
777 { \ifthenelse{\value{COOL@listpointer} = 3}%
778 {%
779  $Q^{-1}$ !\COOL@decide@paren{\InverseGammaRegularized}{#1}%
780 }%
781 % Else
782 {%
783 \PackageError{cool}{Invalid Argument}%
784 {'InverseGammaRegularized' can only accept%
785 a comma separated list of length 2 or 3}%
786 }%
787 }%
788 }
789 \newcommand{\RegIncGammaInv}[2]{\InverseGammaRegularized{#1, #2}}
790 \newcommand{\GammaRegInv}[1]{\InverseGammaRegularized{#1}}

\GenRegIncGamma Generalized Regularized Incomplete Gamma
\GenRegIncGamma{a}{x}{y}  $Q(a, x, y)$ 
\GammaRegularized{a, x, y}  $Q(a, x, y)$ 
791 \newcommand{\GenRegIncGamma}[3]{\GammaRegularized{#1, #2, #3}}

\GenRegIncGammaInv Inverse of Gen. Reg. Incomplete Gamma, \GenRegIncGammaInv{a}{x}{y},
 $Q^{-1}(a, x, y)$ 
792 \newcommand{\GenRegIncGammaInv}[3]{\InverseGammaRegularized{#1, #2, #3}}

\Pochhammer Pochhammer Symbol \Pochhammer{a}{n},  $(a)_n$ 
793 \newcommand{\Pochhammer}[2]{\inp{#1}_{#2}}

\LogGamma Log Gamma Function, \LogGamma{x},  $\log \Gamma(x)$ 

```

```

794 \newcommand{\COOL@notation@LogGammaParen}{p}
795 \DeclareMathOperator{\LogGammaSymb}{\log\Gamma}
796 \newcommand{\LogGamma}[1]{\LogGammaSymb\COOL@decide@paren{LogGamma}{#1}}

```

1.3.19 Derivatives of Gamma Functions

`\DiGamma` Digamma function, `\DiGamma{x}`, $F(x)$

```

797 \newcommand{\COOL@notation@DiGammaParen}{p}
798 \newcommand{\DiGamma}[1]{\digamma\!\COOL@decide@paren{DiGamma}{#1}}

```

PolyGamma function, `\PolyGamma{\nu}{x}`, $\psi^{(\nu)}(x)$

`\PolyGamma`

```

799 \newcommand{\COOL@notation@PolyGammaParen}{p}
800 \newcommand{\PolyGamma}[2]%
801 {\psi^{\in{#1}}\!\COOL@decide@paren{PolyGamma}{#2}}

```

`\HarmNum` Harmonic Number

| | | |
|-------------------------|----------------------------|-------------|
| Harmonic Number | <code>\HarmNum{x}</code> | H_x |
| General Harmonic Number | <code>\HarmNum{x,r}</code> | $H_x^{(r)}$ |

```

802 \newcommand{\HarmNum}[1]{%
803 \listval{#1}{0}%
804 \ifthenelse{\value{COOL@listpointer}=1}%
805 {%
806 H_{#1}
807 }%
808 % Else If
809 { \ifthenelse{\value{COOL@listpointer}=2}%
810 {%
811 \liststore{#1}{COOL@list@temp@}%
812 H^{\in{\COOL@list@temp@ii}}_{\COOL@list@temp@i}%
813 }%
814 % Else
815 {%
816 \PackageError{cool}{Invalid Argument}%
817 {'Harm Num' can only accept a comma separated list of length 1 or 2}%
818 }%
819 }

```

1.3.20 Beta Functions

| | | | |
|--------------------|--------------------------------------|---------------------------------|----------------------|
| | Beta Function | <code>\Beta{a,b}</code> | $B(a,b)$ |
| <code>\Beta</code> | Incomplete Beta Function | <code>\Beta{z,a,b}</code> | $B_z(a,b)$ |
| | Generalized Incomplete Beta Function | <code>\Beta{z_1,z_2,a,b}</code> | $B_{(z_1,z_2)}(a,b)$ |

```

820 \newcommand{\COOL@notation@BetaParen}{p}
821 \newcommand{\Beta}[1]{%
822 \liststore{#1}{COOL@Beta@arg@}%
823 \listval{#1}{0}%
824 \ifthenelse{\value{COOL@listpointer} = 2}%

```

```

825 {%
826 B!\COOL@decide@paren{Beta}{\COOL@Beta@arg@i, \COOL@Beta@arg@ii}%
827 }%
828 % ElseIf
829 { \ifthenelse{\value{COOL@listpointer} = 3}%
830 {%
831 B_{\COOL@Beta@arg@i}\!%
832 \COOL@decide@paren{Beta}{\COOL@Beta@arg@ii, \COOL@Beta@arg@iii}%
833 }%
834 % ElseIf
835 { \ifthenelse{\value{COOL@listpointer} = 4}%
836 {%
837 B_{\inp{\COOL@Beta@arg@i,\COOL@Beta@arg@ii}}\!%
838 \COOL@decide@paren{Beta}{\COOL@Beta@arg@iii, \COOL@Beta@arg@iv}%
839 }%
840 % Else
841 {%
842 \PackageError{cool}{Invalid Argument}%
843 {'Beta' can only accept a comma separated list of length 2 to 4}%
844 }%
845 }%
846 }

\IncBeta Incomplete Beta Function
      \IncBeta{z}{a}{b}    $B_z(a,b)$ 
      \Beta{z,a,b}         $B_z(a,b)$ 
847 \newcommand{\IncBeta}[3]{\Beta{#1,#2, #3}}

\GenIncBeta Generalized Incomplete Beta Function
      \GenIncBeta{x}{y}{a}{b}    $B_{(x,y)}(a,b)$ 
      \Beta{x,y,a,b}             $B_{(x,y)}(a,b)$ 
848 \newcommand{\GenIncBeta}[4]{\Beta{#1,#2,#3,#4}}

\BetaRegularized Regularized Incomplete Beta Function
      \BetaReg      \BetaRegularized{z,a,b}    $I_z(a,b)$ 
\RegIncBeta      \BetaReg{z,a,b}               $I_z(a,b)$ 
      \RegIncBeta{z}{a}{b}    $I_z(a,b)$ 
849 \newcommand{\COOL@notation@BetaRegularizedParen}{p}
850 \newcommand{\BetaRegularized}[1]{%
851 \liststore{#1}{\COOL@BetaRegularized@arg@}%
852 \listval{#1}{0}%
853 \ifthenelse{\value{COOL@listpointer} = 3}%
854 {%
855 I_{\COOL@BetaRegularized@arg@i}\!%
856 \COOL@decide@paren{BetaRegularized}%
857 {\COOL@BetaRegularized@arg@ii, \COOL@BetaRegularized@arg@iii}%
858 }%
859 % ElseIf
860 { \ifthenelse{\value{COOL@listpointer} = 4}%

```

```

861 {%
862 I_{\inp{\COOL@BetaRegularized@arg@i, \COOL@BetaRegularized@arg@ii}}\!%
863 \COOL@decide@paren{BetaRegularized}%
864 {\COOL@BetaRegularized@arg@iii, \COOL@BetaRegularized@arg@iv}%
865 }%
866 % Else
867 {%
868 \PackageError{cool}{Invalid Argument}%
869 {'BetaRegularized' can only accept%
870 a comma separated list of length 3 or 4}%
871 }%
872 }%
873 }
874 \newcommand{\RegIncBeta}[3]{\BetaRegularized{#1,#2,#3}}
875 \newcommand{\BetaReg}[1]{\BetaRegularized{#1}}

\InverseBetaRegularized Inverse of Regularized Incomplete Beta Function
\BetaRegInv \InverseBetaRegularized{z,a,b}  $I_z^{-1}(a,b)$ 
\RegIncBetaInv \BetaRegInv{z,a,b}  $I_z^{-1}(a,b)$ 
\RegIncBetaInv{z}{a}{b}  $I_z^{-1}(a,b)$ 
876 \newcommand{\COOL@notation@InverseBetaRegularizedParen}{p}
877 \newcommand{\InverseBetaRegularized}[1]{%
878 \liststore{#1}{\COOL@InverseBetaRegularized@arg@}%
879 \listval{#1}{0}%
880 \ifthenelse{\value{\COOL@listpointer} = 3}%
881 {%
882 I^{-1}_{\inp{\COOL@InverseBetaRegularized@arg@i}}\!%
883 \COOL@decide@paren{InverseBetaRegularized}%
884 {\COOL@InverseBetaRegularized@arg@ii,%
885 \COOL@InverseBetaRegularized@arg@iii}%
886 }%
887 % ElseIf
888 { \ifthenelse{\value{\COOL@listpointer} = 4}%
889 {%
890 I^{-1}_{\inp{\COOL@InverseBetaRegularized@arg@i,%
891 \COOL@InverseBetaRegularized@arg@ii%
892 }}\!%
893 }\!%
894 \COOL@decide@paren{InverseBetaRegularized}%
895 {\COOL@InverseBetaRegularized@arg@iii,%
896 \COOL@InverseBetaRegularized@arg@iv}%
897 }%
898 % Else
899 {%
900 \PackageError{cool}{Invalid Argument}%
901 {'InverseBetaRegularized' can only accept%
902 a comma separated list of length 3 or 4}%
903 }%
904 }%

```

```

905 }
906 \newcommand{\RegIncBetaInv}[3]{\InverseBetaRegularized{#1,#2,#3}}
907 \newcommand{\BetaRegInv}[1]{\InverseBetaRegularized{#1}}

\GenRegIncBeta   Generalized Regularized Incomplete Beta Func
      \GenRegIncBeta{x}{y}{a}{b}    $B_{(x,y)}(a,b)$ 
      \Beta{x,y,a,b}                $B_{(x,y)}(a,b)$ 
908 \newcommand{\GenRegIncBeta}[4]{\Beta{#1,#2,#3,#4}}

\GenRegIncBetaInv Inverse of Generalized Regularized Incomplete Beta Function
      \GenRegIncBetaInv{x}{y}{z}{b}    $I_{(x,y)}^{-1}(z,b)$ 
      \InverseBetaRegularized{x,y,z,b}  $I_{(x,y)}^{-1}(z,b)$ 
909 \newcommand{\GenRegIncBetaInv}[4]{\InverseBetaRegularized{#1,#2,#3,#4}}

```

1.3.21 Error Functions

```

\Erf   Error Function      \Erf{x}      erf(x)
      Generalized Error Function \Erf{x,y} erf(x,y)

910 \newcommand{\COOL@notation@ErfParen}{p}
911 \DeclareMathOperator{\ErfSymb}{erf}
912 \newcommand{\Erf}[1]{%
913 \liststore{#1}{\COOL@Erf@arg@}%
914 \listval{#1}{0}%
915 \ifthenelse{\value{\COOL@listpointer} = 1}{%
916 {%
917 \ErfSymb\!\COOL@decide@paren{Erf}{#1}
918 }%
919 % ElseIf
920 { \ifthenelse{\value{\COOL@listpointer} = 2}{%
921 {%
922 \ErfSymb\!\COOL@decide@paren{Erf}{#1}
923 }%
924 % Else
925 {%
926 \PackageError{cool}{Invalid Argument}%
927 {'Erf' can only accept a comma separated list of length 1 or 2}%
928 }%
929 }%
930 }

\ErfInv Inverse of Error Function
      \ErfInv{x}      erf-1(x)
      \ErfInv{x,y}    erf-1(x,y)

931 \newcommand{\COOL@notation@ErfInvParen}{p}
932 \newcommand{\ErfInv}[1]{%
933 \liststore{#1}{\COOL@Erf@arg@}%
934 \listval{#1}{0}%
935 \ifthenelse{\value{\COOL@listpointer} = 1}{%

```

```

936 {%
937 \ErfSymb^{-1}\!\COOL@decide@paren{ErfInv}{#1}
938 }%
939 % ElseIf
940 { \ifthenelse{\value{COOL@listpointer} = 2}%
941 {%
942 \ErfSymb^{-1}\!\COOL@decide@paren{ErfInv}{#1}
943 }%
944 % Else
945 {%
946 \PackageError{cool}{Invalid Argument}%
947 {'Erf' can only accept a comma separated list of length 1 or 2}%
948 }%
949 }%
950 }

\GenErf Generalized Error Function and its inverse
\GenErfInv \GenErf{z_1}{z_2} erf(z_1, z_2)
\GenErfInv{z_1}{z_2} erf^{-1}(z_1, z_2)
951 \newcommand{\GenErf}[2]{\Erf{#1,#2}}
952 \newcommand{\GenErfInv}[2]{\ErfInv{#1, #2}}

\Erfc Complimentary Error Function and its inverse
\Erfc{z} erfc(z)
\ErfcInv{z} erfc^{-1}(z)
953 \newcommand{\COOL@notation@ErfcParen}{p}
954 \DeclareMathOperator{\ErfcSymb}{erfc}
955 \newcommand{\Erfc}[1]{\ErfcSymb\!\COOL@decide@paren{Erfc}{#1}}
956 \newcommand{\COOL@notation@ErfcInvParen}{p}
957 \newcommand{\ErfcInv}[1]%
958 {\ErfcSymb^{-1}\!\COOL@decide@paren{ErfcInv}{#1}}

\Erfi Imaginary Error Function, \Erfi{z}, erfi(z)
959 \newcommand{\COOL@notation@ErfiParen}{p}
960 \DeclareMathOperator{\ErfiSymb}{erfi}
961 \newcommand{\Erfi}[1]{\ErfiSymb\!\COOL@decide@paren{Erfi}{#1}}

1.3.22 Fresnel Integrals

\FresnelS Fresnel Integral, \FresnelS{z}, S(z)
962 \newcommand{\COOL@notation@FresnelSParen}{p}
963 \newcommand{\FresnelS}[1]{S\!\COOL@decide@paren{FresnelS}{#1}}

\FresnelC Fresnel Integral, \FresnelC{z}, C(z)
964 \newcommand{\COOL@notation@FresnelCParen}{p}
965 \newcommand{\FresnelC}[1]{C\!\COOL@decide@paren{FresnelC}{#1}}

```

1.3.23 Exponential Integrals

`\ExpIntE` Exponential Integral, `\ExpIntE{\nu}{x}`, $E_\nu(x)$

```

966 \newcommand{\COOL@notation@ExpIntEParen}{p}
967 \newcommand{\ExpIntE}[2]{E_{#1}\!\!\COOL@decide@paren{\ExpIntE}{#2}}

\ExpIntEi Exponential Integral, \ExpIntEi{x},  $Ei(x)$ 
968 \newcommand{\COOL@notation@ExpIntEiParen}{p}
969 \DeclareMathOperator{\ExpIntEiSymb}{Ei}
970 \newcommand{\ExpIntEi}[1]{%
971 {\ExpIntEiSymb\!\!\COOL@decide@paren{\ExpIntEi}{#1}}

\LogInt Logarithmic Integral, \LogInt{x},  $li(x)$ 
972 \newcommand{\COOL@notation@LogIntParen}{p}
973 \DeclareMathOperator{\LogIntSymb}{li}
974 \newcommand{\LogInt}[1]{\LogIntSymb\!\!\COOL@decide@paren{\LogInt}{#1}}

\SinInt Sine Integral, \SinInt{x},  $Si(x)$ 
975 \newcommand{\COOL@notation@SinIntParen}{p}
976 \DeclareMathOperator{\SinIntSymb}{Si}
977 \newcommand{\SinInt}[1]{\SinIntSymb\!\!\COOL@decide@paren{\SinInt}{#1}}

\CosInt Cosine Integral, \CosInt{x},  $Ci(x)$ 
978 \newcommand{\COOL@notation@CosIntParen}{p}
979 \DeclareMathOperator{\CosIntSymb}{Ci}
980 \newcommand{\CosInt}[1]{\CosIntSymb\!\!\COOL@decide@paren{\CosInt}{#1}}

\SinhInt Hyperbolic Sine Integral, \SinhInt{x},  $Shi(x)$ 
981 \newcommand{\COOL@notation@SinhIntParen}{p}
982 \DeclareMathOperator{\SinhIntSymb}{Shi}
983 \newcommand{\SinhInt}[1]{\SinhIntSymb\!\!\COOL@decide@paren{\SinhInt}{#1}}

\CoshInt Hyperbolic Cosine Integral, \CoshInt{x},  $Chi(x)$ 
984 \newcommand{\COOL@notation@CoshIntParen}{p}
985 \DeclareMathOperator{\CoshIntSymb}{Chi}
986 \newcommand{\CoshInt}[1]{\CoshIntSymb\!\!\COOL@decide@paren{\CoshInt}{#1}}
```

1.3.24 Hypergeometric Functions

`\COOL@Hypergeometric@pq@ab@value` This macro is a decision maker that decides what to return for the Hypergeometric function since its results vary based on the nature of the input. This macro is called as

```

\COOL@Hypergeometric@pq@ab@value {‘p’|‘q’} {‘p-input’|‘q-input’} {‘a’|‘b’}
{‘a-input’|‘b-input’}
987 \newcommand{\COOL@Hypergeometric@pq@ab@value}[4]{%
988 \ifthenelse{\boolean{COOL@#1@isint} \AND \boolean{COOL@#3@islist}}{%
989 {% #1 is an INT and #3 is a LIST
990 \ifthenelse{ #2 = 0 }{
```

```

991 {%
992 \PackageWarning{cool}{‘#3’-arg ignored}%
993 }%
994 % Else
995 {%
996 \ifthenelse{ #2 = 1 }{%
997 {%
998 \PackageError{cool}{‘Hypergeometric’ ‘#1’-arg mismatch with ‘#3’-arg}{}%
999 }%
1000 % Else
1001 {%
1002 #4%
1003 }%
1004 }%
1005 }%
1006 % Else
1007 {}%
1008 \ifthenelse{ \boolean{COOL@#1@isint} \AND
1009 \NOT \boolean{COOL@#3@islist} }{%
1010 {%
1011 \ifthenelse{ #2 = 0 }{%
1012 {%
1013 % return nothing
1014 }%
1015 % Else
1016 {%
1017 \ifthenelse{ #2 = 1 }{%
1018 {%
1019 % return
1020 #4%
1021 }%
1022 % Else
1023 {%
1024 \forLoop{1}{#2}{COOL@ct}
1025 {%
1026 \ifthenelse{ \value{COOL@ct} = 1 }{,}%
1027 #4_{\arabic{COOL@ct}}}%
1028 }% end for loop
1029 }%
1030 }%
1031 }%
1032 % else
1033 {}%
1034 \ifthenelse{ \NOT \boolean{COOL@#1@isint} \AND
1035 \boolean{COOL@#3@islist} }{%
1036 {%
1037 \PackageError{cool}{Invalid Argument}%
1038 {‘Hypergeometric’: ‘#1’-arg is not int but ‘#3’-arg is list}
1039 }%
1040 % else

```

```

1041 {}%
1042 \ifthenelse{ \NOT \boolean{COOL@#1@isint} \AND
1043 \NOT \boolean{COOL@#3@islist} }%
1044 {}%
1045 %return
1046 #4_1,\ldots,#4_{#2}%
1047 }%
1048 % else
1049 {}%
1050 }%

\Hypergeometric Generalized Hypergeometric function.  ${}_pF_q(a_1, \dots, a_p; b_1, \dots, b_q; x)$ 
\Hypergeometric{0}{0}{}{}{x}  ${}_0F_0(; ; x)$ 
\Hypergeometric{0}{1}{}{b}{x}  ${}_0F_1(; b; x)$ 
\Hypergeometric{1}{1}{a}{b}{x}  ${}_1F_1(a; b; x)$ 
\Hypergeometric{1}{1}{1}{1}{x}  ${}_1F_1(1; 1; x)$ 
\Hypergeometric{3}{5}{a}{b}{x}  ${}_3F_5(a_1, a_2, a_3; b_1, b_2, b_3, b_4, b_5; x)$ 
\Hypergeometric{3}{5}{1,2,3}{1,2,3,4,5}{x}  ${}_3F_5(1, 2, 3; 1, 2, 3, 4, 5; x)$ 
\Hypergeometric{p}{5}{a}{b}{x}  ${}_pF_5(a_1, \dots, a_p; b_1, b_2, b_3, b_4, b_5; x)$ 
\Hypergeometric{p}{3}{a}{1,2,3}{x}  ${}_pF_3(a_1, \dots, a_p; 1, 2, 3; x)$ 
\Hypergeometric{p}{q}{a}{b}{x}  ${}_pF_q(a_1, \dots, a_p; b_1, \dots, b_q; x)$ 

1051 \newcommand{\COOL@notation@HypergeometricParen}{p}
1052 \newcommand{\COOL@notation@HypergeometricSymb}{F}
1053 \newcommand{\Hypergeometric}[6][F]{%
1054 \provideboolean{COOL@p@isint}%
1055 \provideboolean{COOL@q@isint}%
1056 \provideboolean{COOL@a@islist}%
1057 \provideboolean{COOL@b@islist}%
1058 \isint{#2}%
1059 \ifthenelse{\boolean{COOL@isint}}%
1060 {\setboolean{COOL@p@isint}{true}}%
1061 % Else
1062 {\setboolean{COOL@p@isint}{false}}%
1063 \isint{#3}%
1064 \ifthenelse{\boolean{COOL@isint}}%
1065 {\setboolean{COOL@q@isint}{true}}%
1066 % Else
1067 {\setboolean{COOL@q@isint}{false}}%
1068 \listval{#4}{0}%
1069 \ifthenelse{\value{COOL@listpointer}>1}%
1070 {\setboolean{COOL@a@islist}{true}}%
1071 % Else
1072 {\setboolean{COOL@a@islist}{false}}%

```

```

ensure that the submitted list is the same length as p
1073 \ifthenelse{ \boolean{COOL@p@isint} \AND
1074 \boolean{COOL@a@islist} \AND
1075 \NOT \{ #2 = \value{COOL@listpointer} \} }%
1076 {%
1077 \PackageError{cool}{‘Hypergeometric’ ‘p’-arg mismatch with ‘a’-arg}{}%
1078 }%
1079 % else
1080 {%
1081 \listval{#5}{0}%
1082 \ifthenelse{\value{COOL@listpointer}>1}%
1083 {\setboolean{COOL@b@islist}{true}}%
1084 % Else
1085 {\setboolean{COOL@b@islist}{false}}%
ensure that the submitted ‘b’ list is the same length as q
1086 \ifthenelse{ \boolean{COOL@q@isint} \AND
1087 \boolean{COOL@b@islist} \AND
1088 \NOT \{ #3 = \value{COOL@listpointer} \} }%
1089 {%
1090 \PackageError{cool}{‘Hypergeometric’ ‘q’-arg mismatch with ‘b’-arg}%
1091 {‘b’ list is not the same length as ‘q’}%
1092 }%
1093 % else
1094 {%
1095 % troubleshoot
1096 \ifthenelse{ \boolean{COOL@a@islist} \AND \NOT \boolean{COOL@p@isint} }%
1097 {%
1098 \PackageError{cool}{‘Hypergeometric’ ‘a’-arg mismatch with ‘p’-arg}%
1099 {happens if ‘a’-arg is a list and ‘p’-arg isn’t an integer}%
1100 }%
1101 % else
1102 {%
1103 \ifthenelse{ \boolean{COOL@b@islist} \AND \NOT \boolean{COOL@q@isint} }%
1104 {%
1105 \PackageError{cool}{‘Hypergeometric’ ‘b’-arg mismatch with ‘q’-arg}%
1106 {happens if ‘b’-arg is a list and ‘q’-arg isn’t an integer}%
1107 }%
1108 % else
1109 {%
First print the  ${}_pF_q$ 
1110 \}_{#2}\COOL@notation@HypergeometricSymb}_{#3}\!%
1111 \COOL@decide@paren{Hypergeometric}%
1112 {%
1113 \COOL@Hypergeometric@pq@ab@value{p}{#2}{a}{#4};%
1114 \COOL@Hypergeometric@pq@ab@value{q}{#3}{b}{#5};%
1115 #6%
1116 }%
1117 }

```

```

\RegHypergeometric Regularized hypergeometric function  ${}_p\tilde{F}_q(a_1, \dots, a_p; b_1, \dots, b_q; x)$ 
1118 \newcommand{\COOL@notation@RegHypergeometricParen}{p}
1119 \newcommand{\COOL@notation@RegHypergeometricSymb}{\tilde{F}}
1120 \newcommand{\RegHypergeometric}[6][\tilde{F}]{%
1121 \provideboolean{COOL@p@isint}%
1122 \provideboolean{COOL@q@isint}%
1123 \provideboolean{COOL@a@islist}%
1124 \provideboolean{COOL@b@islist}%
1125 \isint{#2}%
1126 \ifthenelse{\boolean{COOL@isint}}{%
1127 {\setboolean{COOL@p@isint}{true}}%
1128 % Else
1129 {\setboolean{COOL@p@isint}{false}}%
1130 \isint{#3}%
1131 \ifthenelse{\boolean{COOL@isint}}{%
1132 {\setboolean{COOL@q@isint}{true}}%
1133 % Else
1134 {\setboolean{COOL@q@isint}{false}}%
1135 \listval{#4}{0}%
1136 \ifthenelse{\value{COOL@listpointer}>1}%
1137 {\setboolean{COOL@a@islist}{true}}%
1138 % Else
1139 {\setboolean{COOL@a@islist}{false}}%
    ensure that the submitted list is the same length as p
1140 \ifthenelse{ \boolean{COOL@p@isint} \AND
1141 \boolean{COOL@a@islist} \AND
1142 \NOT \(\ #2 = \value{COOL@listpointer} \) }{%
1143 {%
1144 \PackageError{cool}%
1145 {'RegHypergeometric' 'p'-arg mismatch with 'a'-arg}{}%
1146 }%
1147 % else
1148 {%
1149 \listval{#5}{0}%
1150 \ifthenelse{\value{COOL@listpointer}>1}%
1151 {\setboolean{COOL@b@islist}{true}}%
1152 % Else
1153 {\setboolean{COOL@b@islist}{false}}%
    ensure that the submitted 'b' list is the same length as q
1154 \ifthenelse{ \boolean{COOL@q@isint} \AND
1155 \boolean{COOL@b@islist} \AND
1156 \NOT \(\ #3 = \value{COOL@listpointer} \) }{%
1157 {%
1158 \PackageError{cool}%
1159 {'RegHypergeometric' 'q'-arg mismatch with 'b'-arg}%
1160 {'b' list is not the same length as 'q'}%
1161 }%
1162 % else

```

```

1163 {}%
1164 % troubleshoot
1165 \ifthenelse{ \boolean{COOL@a@islist} \AND \NOT \boolean{COOL@p@isint} }%
1166 {%
1167 \PackageError{cool}%
1168 {'RegHypergeometric' 'a'-arg mismatch with 'p'-arg}%
1169 {happens if 'a'-arg is a list and 'p'-arg isn't an integer}%
1170 }%
1171 % else
1172 {}%
1173 \ifthenelse{ \boolean{COOL@b@islist} \AND \NOT \boolean{COOL@q@isint} }%
1174 {%
1175 \PackageError{cool}%
1176 {'RegHypergeometric' 'b'-arg mismatch with 'q'-arg}%
1177 {happens if 'b'-arg is a list and 'q'-arg isn't an integer}%
1178 }%
1179 % else
1180 {}%

```

First print the ${}_pF_q$

```

1181 {}_{#2}\COOL@notation@RegHypergeometricSymb}_{#3}\!%
1182 \COOL@decide@paren{RegHypergeometric}%
1183 {%
1184 \COOL@Hypergeometric@pq@ab@value{p}_{#2}{a}_{#4};%
1185 \COOL@Hypergeometric@pq@ab@value{q}_{#3}{b}_{#5};%
1186 #6%
1187 }%
1188 }

```

`\AppellFOne` Appell Hypergeometric Function

`\AppellFOne{a}{b_1,b_2}{c}{z_1,z_2}` $F_1(a; b_1, b_2; c; z_1, z_2)$

```

1189 \newcommand{\COOL@notation@AppellFOneParen}{p}
1190 \newcommand{\AppellFOne}[4]%
1191 {F_{#1}\!\COOL@decide@paren{AppellFOne}{#1; #2; #3; #4}}

```

`\HypergeometricU` Tricomi confluent hypergeometric function

`\HypergeometricU{a}{b}{z}` $U(a, b, z)$

```

1192 \newcommand{\COOL@notation@HypergeometricUSymb}{U}
1193 \newcommand{\HypergeometricU}[3]%
1194 {\COOL@notation@HypergeometricUSymb\!\inp{#1, #2, #3}}

```

`\COOL@MeijerG@anp@value` This macro is a decision maker for the `\MeijerG` macro. Despite the name it is used for both p and q . It is called as

`\COOL@MeijerG@anp@value {<a|b>} {<n|m>} {<p|q>}`

```

1195 \newcommand{\COOL@MeijerG@anp@value}[3]{%
1196 \isint{#3}%
1197 \ifthenelse{\boolean{COOL@isint}}%
1198 {%
1199 \isint{#2}%

```

```

1200 \ifthenelse{\boolean{COOL@isint}}{%
1201 {%
1202 \forLoop{1}{#3}{COOL@ct}%
1203 {%
1204 \ifthenelse{\value{COOL@ct}=1}{\{,\}}{%
1205 #1_{\arabic{COOL@ct}}}%
1206 }%
1207 }%
1208 % else
1209 {%
1210 #1_1,\ldots,#1_{#2},#1_{#2+1},\ldots,#1_{#3}%
1211 }%
1212 }%
1213 % else
1214 {%
1215 \isint{#2}%
1216 \ifthenelse{\boolean{COOL@isint}}{%
1217 {%
1218 \forLoop{1}{#2}{COOL@ct}%
1219 {%
1220 \ifthenelse{\value{COOL@ct}=1}{\{,\}}{%
1221 #1_{\arabic{COOL@ct}}}%
1222 }%
1223 \setcounter{COOL@ct}{#2}%
1224 \addtocounter{COOL@ct}{1}%
1225 ,#1_{\arabic{COOL@ct}}, \ldots, #1_{#3}%
1226 }%
1227 % else
1228 {%
1229 #1_1,\ldots,#1_{#2},#1_{#2+1},\ldots,#1_{#3}%
1230 }%
1231 }%
1232 }

\MeijerG \MeijerG{a_1,\ldots,a_n}{a_{n+1},\ldots,a_p}{b_1,\ldots,b_m}{b_{m+1},\ldots,b_q}{\langle x \rangle}
\MeijerG[\langle a \text{ list symbol} \rangle,\langle b \text{ list symbol} \rangle]{\langle n \rangle}{\langle p \rangle}{\langle m \rangle}{\langle q \rangle}{\langle x \rangle}
\MeijerG[\langle a \text{ list symbol} \rangle]{\langle n \rangle}{\langle p \rangle}{b_1,\ldots,b_m}{b_{m+1},\ldots,b_q}{\langle x \rangle}
\MeijerG[, \langle b \text{ list symbol} \rangle]{a_1,\ldots,a_n}{a_{n+1},\ldots,a_p}{\langle m \rangle}{\langle q \rangle}{\langle x \rangle}

```

Meijer G -Function

$$\backslash\text{MeijerG}[a,b]\{n\}\{p\}\{m\}\{q\}\{z\} \quad G_{p,q}^{m,n} \left(z \left| \begin{matrix} a_1, \dots, a_n, a_{n+1}, \dots, a_p \\ b_1, \dots, b_m, b_{m+1}, \dots, b_q \end{matrix} \right. \right)$$

Meijer G -Function

$$\backslash\text{MeijerG}\{1,2\}\{3\}\{a,b\}\{c,d\}\{z\} \quad G_{3,4}^{2,2} \left(z \left| \begin{matrix} 1,2,3 \\ a,b,c,d \end{matrix} \right. \right)$$

Generalized Meijer G -Function

$$\backslash\text{MeijerG}[a,b]\{n\}\{p\}\{m\}\{q\}\{z,r\} \quad G_{p,q}^{m,n} \left(z, r \left| \begin{matrix} a_1, \dots, a_n, a_{n+1}, \dots, a_p \\ b_1, \dots, b_m, b_{m+1}, \dots, b_q \end{matrix} \right. \right)$$

Generalized Meijer G -Function

$$\backslash\text{MeijerG}\{1,2\}\{3\}\{a,b\}\{c,d\}\{z,r\} \quad G_{3,4}^{2,2} \left(z, r \left| \begin{matrix} 1,2,3 \\ a,b,c,d \end{matrix} \right. \right)$$

```

1233 \newcommand{\COOL@notation@MeijerGSymb}{G}
1234 \newcommand{\MeijerG}[6][@, @]{%
1235 \listval{#1}{0}
1236 \ifthenelse{\value{COOL@listpointer}>2 \OR \value{COOL@listpointer}<1}%
1237 {%
1238 \PackageError{cool}{‘MeijerG’ Invalid Optional Argument}%
1239 {Must be a comma separated list of length 1 or 2}%
1240 }%
1241 % else
1242 {%
1243 }%
1244 \COOL@notation@MeijerGSymb%
1245 \ifthenelse{\equal{#1}{@, @}}{%
1246 {%
1247 \listval{#2}{0}% n
1248 \setcounter{COOL@ct}{\value{COOL@listpointer}}%
1249 \listval{#4}{0}% m
1250 \setcounter{COOL@ct@}{\value{COOL@listpointer}}%
1251 ~{\arabic{COOL@ct@}, \arabic{COOL@ct}}%
1252 \listval{#3}{0}% p - n
1253 \addtocounter{COOL@ct}{\value{COOL@listpointer}}%
1254 \listval{#5}{0}% q - m
1255 \addtocounter{COOL@ct@}{\value{COOL@listpointer}}%
1256 _{\arabic{COOL@ct}, \arabic{COOL@ct@}}%
1257 \!\left(%
1258 #6%
1259 \left| %
1260 { {#2, #3} \@@atop {#4, #5} }%
1261 \right)\right.%
1262 }%
1263 % else
1264 {%
1265 \listval{#1}{0}%
1266 \ifthenelse{\value{COOL@listpointer}=2}%
1267 {%
1268 \provideboolean{COOL@MeijerG@opt@one@blank}%
1269 \def\COOL@MeijerG@sniffer##1,##2\COOL@MeijerG@sniffer@end{%
1270 \ifthenelse{\equal{##1}{}}%
1271 {%

```

```

1272 \setboolean{COOL@MeijerG@opt@one@blank}{true}%
1273 }%
1274 % else
1275 {%
1276 \setboolean{COOL@MeijerG@opt@one@blank}{false}%
1277 }%
1278 }%
1279 \expandafter\COOL@MeijerG@sniffer#1\COOL@MeijerG@sniffer@end\relax%
1280 \ifthenelse{\boolean{COOL@MeijerG@opt@one@blank}}%
1281 {%
    this is \MeijerG[,b]{a_1,\dots,a_n}{a_{n++},\dots,a_p}{m}{q}{x}
1282 \listval{#2}{0}% n
1283 \setcounter{COOL@ct}{\value{COOL@listpointer}}%
1284 ^{#4,\arabic{COOL@ct}}%
1285 \listval{#3}{0}% p
1286 \addtocounter{COOL@ct}{\value{COOL@listpointer}}%
1287 _{\arabic{COOL@ct},#5}%
1288 \!\left(
1289 #6%
1290 \left|
1291 {%
1292 {#2,#3} \@@atop {\COOL@MeijerG@anp@value{\listval{#1}{2}}{#4}{#5}}
1293 }%
1294 \right)\right.%
1295 }%
1296 % else
1297 {%
1298 ^{#4,#2}_{#3,#5}%
1299 \!\left(
1300 #6%
1301 \left|
1302 {%
1303 {\COOL@MeijerG@anp@value{\listval{#1}{1}}{#2}{#3}}%
1304 \@@atop%
1305 {\COOL@MeijerG@anp@value{\listval{#1}{2}}{#4}{#5}}
1306 }%
1307 \right)\right.%
1308 }%
1309 }%
1310 % else
1311 {%
    this is \MeijerG[a]{n}{p}{b_1,\dots,b_m}{b_{m++},\dots,a_p}{x}
1312 \listval{#4}{0}% m
1313 \setcounter{COOL@ct}{\value{COOL@listpointer}}%
1314 ^{\arabic{COOL@ct}, #2}%
1315 \listval{#5}{0}% q
1316 \addtocounter{COOL@ct}{\value{COOL@listpointer}}%
1317 _{#3, \arabic{COOL@ct}}%
1318 \!\left(

```

```

1319 #6%
1320 \left| %
1321 { %
1322 {\COOL@MeijerG@anp@value{#1}{#2}{#3}} \@@atop {#4,#5}
1323 } %
1324 \right)\right.%
1325 } %
1326 } %
1327 } %

```

1.3.25 Angular Momentum Functions

\ClebschGordon Clebsch-Gordon Coefficients

```

\ClebschGordon{j_1,m_1}{j_2,m_2}{j,m} \langle j_1,j_2;m_1,m_2|j_1,j_2;j,m\rangle
http://functions.wolfram.com/HypergeometricFunctions/ClebschGordan/
1328 \newcommand{\ClebschGordon}[3]{ %
1329 \listval{#1}{0} %
1330 \ifthenelse{\NOT \value{COOL@listpointer}=2} %
1331 { %
1332 \PackageError{cool}{‘ClebschGordon’ Invalid Argument} %
1333 {Must have a comma separated list of length two} %
1334 } %
1335 % else
1336 {} %
1337 \listval{#2}{0} %
1338 \ifthenelse{\NOT \value{COOL@listpointer}=2} %
1339 { %
1340 \PackageError{cool}{‘ClebschGordon’ Invalid Argument} %
1341 {Must have a comma separated list of length two} %
1342 } %
1343 % else
1344 {} %
1345 \listval{#3}{0} %
1346 \ifthenelse{\NOT \value{COOL@listpointer}=2} %
1347 { %
1348 \PackageError{cool}{‘ClebschGordon’ Invalid Argument} %
1349 {Must have a comma separated list of length two} %
1350 } %
1351 % else
1352 {} %
1353 \left< %
1354 \listval{#1}{1},\listval{#2}{1}; %
1355 \listval{#1}{2},\listval{#2}{2} %
1356 \left| %
1357 \listval{#1}{1},\listval{#2}{1}; %
1358 \listval{#3}{1},\listval{#3}{2} %
1359 \right>\right.%
1360 }

```

\ThreeJSymbol Wigner 3-j Symbol

```

\ThreeJSymbol{j_1,m_1}{j_2,m_2}{j_3,m_3} 
$$\begin{pmatrix} j_1 & j_2 & j_3 \\ m_1 & m_2 & m_3 \end{pmatrix}$$

http://functions.wolfram.com/HypergeometricFunctions/ThreeJSymbol/
1361 \newcommand{\ThreeJSymbol}[3]{%
1362 \listval{#1}{0}%
1363 \ifthenelse{\NOT \value{COOL@listpointer}=2}%
1364 {%
1365 \PackageError{cool}{‘ThreeJSymbol’ Invalid Argument}%
1366 {Must have comma separated list of length 2}%
1367 }%
1368 % else
1369 {%
1370 \listval{#2}{0}%
1371 \ifthenelse{\NOT \value{COOL@listpointer}=2}%
1372 {%
1373 \PackageError{cool}{‘ThreeJSymbol’ Invalid Argument}%
1374 {Must have comma separated list of length 2}%
1375 }%
1376 % else
1377 {%
1378 \listval{#3}{0}%
1379 \ifthenelse{\NOT \value{COOL@listpointer}=2}%
1380 {%
1381 \PackageError{cool}{‘ThreeJSymbol’ Invalid Argument}%
1382 {Must have comma separated list of length 2}%
1383 }%
1384 % else
1385 {%
1386 \mathchoice{%
1387 % displaystyle
1388 \inp{\!%
1389 \begin{array}{ccc}%
1390 \listval{#1}{1} & \listval{#2}{1} & \listval{#3}{1} \\
1391 \listval{#1}{2} & \listval{#2}{2} & \listval{#3}{2} \\
1392 \end{array}%
1393 \!}%
1394 }%
1395 {%
1396 % inline
1397 \inp{\!%
1398 {\listval{#1}{1} \@@atop \listval{#1}{2}}%
1399 {\listval{#2}{1} \@@atop \listval{#2}{2}}%
1400 {\listval{#3}{1} \@@atop \listval{#3}{2}}%
1401 \!}%
1402 }%
1403 {%
1404 % subscript
1405 \inp{\!%
1406 {\listval{#1}{1} \@@atop \listval{#1}{2}}%
1407 {\listval{#2}{1} \@@atop \listval{#2}{2}}%

```

```

1408 {\listval{#3}{1} \@@atop \listval{#3}{2}}%
1409      \!}%
1410    }%
1411    {%
1412 % subsubscript
1413 \inp{\!%
1414 {\listval{#1}{1} \@@atop \listval{#1}{2}}%
1415 {\listval{#2}{1} \@@atop \listval{#2}{2}}%
1416 {\listval{#3}{1} \@@atop \listval{#3}{2}}%
1417      \!}%
1418    }%
1419 }

```

\SixJSymbol Racah 6-j Symbol

```

\SixJSymbol{j_1,j_2,j_3}{j_4,j_5,j_6} \left\{ \begin{matrix} j_1 & j_2 & j_3 \\ j_4 & j_5 & j_6 \end{matrix} \right\}
http://functions.wolfram.com/HypergeometricFunctions/SixJSymbol/
1420 \newcommand{\SixJSymbol}[2]{%
1421 \listval{#1}{0}%
1422 \ifthenelse{\NOT \value{COOL@listpointer}=3}%
1423 {%
1424 \PackageError{cool}{‘SixJSymbol’ Invalid Argument}%
1425 {Must have a comma separated list of length 3}%
1426 }%
1427 %else
1428 {%
1429 \listval{#2}{0}%
1430 \ifthenelse{\NOT \value{COOL@listpointer}=3}%
1431 {%
1432 \PackageError{cool}{‘SixJSymbol’ Invalid Argument}%
1433 {Must have a comma separated list of length 3}%
1434 }%
1435 %else
1436 {%
1437 \mathchoice{%
1438 % displaystyle
1439 \inbr{\!%
1440 \begin{array}{ccc}%
1441 \listval{#1}{1} & \listval{#1}{2} & \listval{#1}{3} \\
1442 \listval{#2}{1} & \listval{#2}{2} & \listval{#2}{3} \\
1443 \end{array}%
1444      \!}%
1445    }%
1446    {%
1447 % inline
1448 \inbr{\!%
1449 {\listval{#1}{1} \@@atop \listval{#2}{1}}%
1450 {\listval{#1}{2} \@@atop \listval{#2}{2}}%
1451 {\listval{#1}{3} \@@atop \listval{#2}{3}}%
1452      \!}%

```

```

1453 }%
1454 {%
1455 % superscript
1456 \inbr{\!%
1457 {\listval{#1}{1} \@@atop \listval{#2}{1}}%
1458 {\listval{#1}{2} \@@atop \listval{#2}{2}}%
1459 {\listval{#1}{3} \@@atop \listval{#2}{3}}%
1460 \!}%
1461 }%
1462 {%
1463 % supersuperscript
1464 \inbr{\!%
1465 {\listval{#1}{1} \@@atop \listval{#2}{1}}%
1466 {\listval{#1}{2} \@@atop \listval{#2}{2}}%
1467 {\listval{#1}{3} \@@atop \listval{#2}{3}}%
1468 \!}%
1469 }%
1470 }

```

1.3.26 Complete Elliptic Integrals

\EllipticK Complete Elliptic Integral of the First Kind

\EllipticK{x} $K(x)$

```

1471 \newcommand{\COOL@notation@EllipticKParen}{p}
1472 \newcommand{\COOL@notation@EllipticKSymb}{K}
1473 \newcommand{\EllipticK}[1]%
1474 {\COOL@notation@EllipticKSymb\!\COOL@decide@paren{EllipticK}{#1}}%

```

\EllipticE Complete Elliptic Integral of the Second Kind

\EllipticE{x} $E(x)$

```

1475 \newcommand{\COOL@notation@EllipticEParen}{p}
1476 \newcommand{\COOL@notation@EllipticESymb}{E}
1477 \newcommand{\EllipticE}[1]{%
1478 \liststore{#1}{\COOL@EllipticE@arg@}%
1479 \listval{#1}{0}%
1480 \ifthenelse{\value{\COOL@listpointer} = 1}%
1481 {%
1482 \COOL@notation@EllipticESymb\!\COOL@decide@paren{EllipticE}{#1}%
1483 }%
1484 % ElseIf
1485 { \ifthenelse{\value{\COOL@listpointer} = 2}%
1486 {%
1487 \COOL@notation@EllipticESymb\!%
1488 \COOL@decide@paren{EllipticE}%
1489 {\COOL@EllipticE@arg@i \left| \, , \COOL@EllipticE@arg@ii \!\!\right.}%
1490 }%
1491 % Else
1492 {%
1493 \PackageError{Invalid Argument}%

```

```

1494 {'EllipticE' can only accept a comma separated list of length 1 or 2}%
1495 }%
1496 }%
1497 }

```

\EllipticPi Complete Elliptic Integral of the Third Kind

```

      \EllipticPi{n,m}    $\Pi(n|m)$ 
1498 \newcommand{\COOL@notation@EllipticPiParen}{p}
1499 \newcommand{\COOL@notation@EllipticPiSymb}{\Pi}
1500 \newcommand{\EllipticPi}[1]{%
1501 \liststore{#1}{\COOL@EllipticPi@arg@}%
1502 \listval{#1}{0}%
1503 \ifthenelse{\value{\COOL@listpointer} = 2}%
1504 {%
1505 \COOL@notation@EllipticPiSymb%
1506 \!\COOL@decide@paren{\EllipticPi}%
1507 {\COOL@EllipticPi@arg@i \left| \,, \COOL@EllipticPi@arg@ii \!\!\right.}%
1508 }%
1509 % ElseIf
1510 { \ifthenelse{\value{\COOL@listpointer} = 3}%
1511 {%
1512 \COOL@notation@EllipticPiSymb%
1513 \!\COOL@decide@paren{\EllipticPi}%
1514 { \COOL@EllipticPi@arg@i; \,,%
1515 \COOL@EllipticPi@arg@ii \left| \,,%
1516 \COOL@EllipticPi@arg@iii \!\!\right.}%
1517 }%
1518 }%
1519 % Else
1520 {%
1521 \PackageError{cool}{Invalid Argument}%
1522 {'EllipticPi' can only accept a comma separated list of length 2 or 3}%
1523 }%
1524 }%
1525 }

```

1.3.27 Incomplete Elliptic Integrals

\EllipticF Incomplete Elliptic Integral of the First Kind

```

\IncEllipticF      \EllipticF{z,m}       $F(z|m)$ 
                  \IncEllipticF{z}{m}     $F(z|m)$ 
1526 \newcommand{\COOL@notation@EllipticFParen}{p}
1527 \newcommand{\COOL@notation@EllipticFSymb}{F}
1528 \newcommand{\EllipticF}[1]{%
1529 \liststore{#1}{\COOL@EllipticF@arg@}%
1530 \listval{#1}{0}%
1531 \ifthenelse{\value{\COOL@listpointer} = 2 }%
1532 {%
1533 \COOL@notation@EllipticFSymb%

```

```

1534 \!\COOL@decide@paren{EllipticF}%
1535 {\COOL@EllipticF@arg@i \left| \, \COOL@EllipticF@arg@ii \!\!\right.}%
1536 }%
1537 % Else
1538 {%
1539 \PackageError{cool}{Invalid Argument}%
1540 {'EllipticF' can only accept a comma separated list of length 2}%
1541 }%
1542 }
1543 \newcommand{\IncEllipticF}[2]{\EllipticF{#1,#2}}

\IncEllipticE   Incomplete Elliptic Integral of the Second Kind
                \IncEllipticE{z}{m}    $E(z|m)$ 
                \EllipticE{z,m}        $E(z|m)$ 
1544 \newcommand{\IncEllipticE}[2]{\EllipticE{#1,#2}}

\IncEllipticPi  Incomplete Elliptic Integral of the Third Kind
\EllipticPi     \IncEllipticPi{n}{z}{m}    $\Pi(n; z|m)$ 
                \EllipticPi{n,z,m}        $\Pi(n; z|m)$ 
1545 \newcommand{\IncEllipticPi}[3]{\EllipticPi{#1,#2,#3}}

\JacobiZeta     Jacobi Zeta Function
                \JacobiZeta{z}{m}    $Z(z|m)$ 
1546 \newcommand{\COOL@notation@JacobiZetaParen}{p}
1547 \newcommand{\COOL@notation@JacobiZetaSymb}{Z}
1548 \newcommand{\JacobiZeta}[2]{%
1549 \COOL@notation@JacobiZetaSymb
1550 \!\COOL@decide@paren{JacobiZeta}{#1 \left| \, \, #2 \right.\!\!\!}%
1551 }

```

1.3.28 Jacobi Theta Functions

```

\EllipticTheta  Jacobi Theta Functions
\JacobiTheta     \JacobiTheta{1}{z}{q}    $\vartheta_1(z,q)$ 
                \JacobiTheta{2}{z}{q}    $\vartheta_2(z,q)$ 
                \JacobiTheta{3}{z}{q}    $\vartheta_3(z,q)$ 
                \JacobiTheta{4}{z}{q}    $\vartheta_4(z,q)$ 
1552 \newcommand{\COOL@notation@EllipticThetaParen}{p}
1553 \newcommand{\EllipticTheta}[3]%
1554 {\vartheta_{#1}\!\COOL@decide@paren{EllipticTheta}{#2, #3}}
1555 \newcommand{\JacobiTheta}[3]{\EllipticTheta{#1}{#2}{#3}}

```

1.3.29 Neville Theta Functions

```

\NevilleThetaC  Neville Theta Function, \NevilleThetaC{z}{m},  $\vartheta_c(z|m)$ 
1556 \newcommand{\COOL@notation@NevilleThetaCParen}{p}
1557 \newcommand{\NevilleThetaC}[2]{%
1558 \vartheta_{#1}\!\COOL@decide@paren{NevilleThetaC}{%

```

```

1559 {#1 \left| \, #2 \right.\!\!\}%
1560 }

\NevilleThetaD Neville Theta Function, \NevilleThetaD{z}{m},  $\vartheta_d(z|m)$ 
1561 \newcommand{\COOL@notation@NevilleThetaDParen}{p}
1562 \newcommand{\NevilleThetaD}[2]{%
1563 \vartheta_{#1}\!\!\COOL@decide@paren{NevilleThetaD}%
1564 {#1 \left| \, #2 \right.\!\!\}%
1565 }

\NevilleThetaN Neville Theta Function, \NevilleThetaN{z}{m},  $\vartheta_n(z|m)$ 
1566 \newcommand{\COOL@notation@NevilleThetaNParen}{p}
1567 \newcommand{\NevilleThetaN}[2]{%
1568 \vartheta_{#1}\!\!\COOL@decide@paren{NevilleThetaN}%
1569 {#1 \left| \, #2 \right.\!\!\}%
1570 }

\NevilleThetaS Neville Theta Function, \NevilleThetaS{z}{m},  $\vartheta_s(z|m)$ 
1571 \newcommand{\COOL@notation@NevilleThetaSParen}{p}
1572 \newcommand{\NevilleThetaS}[2]{%
1573 \vartheta_{#1}\!\!\COOL@decide@paren{NevilleThetaS}%
1574 {#1 \left| \, #2 \right.\!\!\}%
1575 }

```

1.3.30 Weierstrass Functions

```

\WeierstrassP Weierstrass Elliptic Function
\WeiP      \WeierstrassP{z}{g_2,g_3}     $\wp(z;g_2,g_3)$ 
           \WeiP{z}{g_2,g_3}           $\wp(z;g_2,g_3)$ 
1576 \newcommand{\COOL@notation@WeierstrassPParen}{p}
1577 \newcommand{\WeierstrassP}[2]{%
1578 \liststore{#2}{\COOL@WeiP@arg@g}%
1579 \listval{#2}{0}%
1580 \ifthenelse{\NOT \value{\COOL@listpointer} = 2}{%
1581 {%
1582 \PackageError{cool}{Invalid Argument}%
1583 {'WeierstrassP' second argument must be%
1584 a comma separated list of length 2}%
1585 }
1586 % Else
1587 {%
1588 \wp\!\!\COOL@decide@paren{WeierstrassP}{#1; #2}
1589 }%
1590 }
1591 \newcommand{\WeiP}[2]{\WeierstrassP{#1}{#2}}

\WeierstrassPInv Inverse of Weierstrass Elliptic Function
\WeiPInv      Inverse      \WeiPInv{z}{g_2,g_3}     $\wp^{-1}(z;g_2,g_3)$ 
              Generalized Inverse \WeiPInv{z_1,z_2}{g_2,g_3}  $\wp^{-1}(z_1,z_2;g_2,g_3)$ 

```

```

1592 \newcommand{\COOL@notation@WeierstrassPInvParen}{p}
1593 \newcommand{\WeierstrassPInv}[2]{%
1594 \liststore{#1}{\COOL@WeiPInv@arg@z@}%
1595 \liststore{#1}{\COOL@WeiPInv@arg@g@}%
1596 \listval{#2}{0}%
1597 \ifthenelse{\NOT \value{\COOL@listpointer} = 2}%
1598 {%
1599 \PackageError{cool}{Invalid Argument}%
1600 {'WeierstrassPInv' second argument must be%
1601 a comma separated list of length 2}%
1602 }
1603 % Else
1604 {
1605 \listval{#1}{0}%
1606 \ifthenelse{\value{\COOL@listpointer} = 1}%
1607 {%
1608 \wp^{-1}\!\!\COOL@decide@paren{WeierstrassPInv}{#1; #2}%
1609 }%
1610 % ElseIf
1611 { \ifthenelse{\value{\COOL@listpointer} = 2}%
1612 {%
1613 \wp^{-1}\!\!\COOL@decide@paren{WeierstrassPInv}{#1; #2}%
1614 }%
1615 % Else
1616 {%
1617 \PackageError{cool}{Invalid Argument}%
1618 {'WeierstrassPInv' first argument must be%
1619 a comma separate list of length 1 or 2}%
1620 }}%
1621 }%
1622 }
1623 \newcommand{\WeiPInv}[2]{\WeierstrassPInv{#1}{#2}}

\WeierstrassPGenInv Generalized Inverse of Weierstrass Elliptic Function
\WierstrassPGenInv{z_1}{z_2}{g_1}{g_2}
1624 \newcommand{\WeierstrassPGenInv}[4]{\WeierstrassPInv{#1,#2}{#3,#4}}

\WeierstrassSigma Wierstrass Sigma Function
\WeiSigma Sigma \WeierstrassSigma{z}{g_2,g_3}  $\sigma(z; g_2, g_3)$ 
\WeiSigma{z}{g_2,g_3}  $\sigma(z; g_2, g_3)$ 
Associated Sigma \WeierstrassSigma{n,z}{g_2,g_3}  $\sigma_n(z; g_2, g_3)$ 
\WeiSigma{n,z}{g_2,g_3}  $\sigma_n(z; g_2, g_3)$ 
1625 \newcommand{\WeierstrassSigma}[2]{%
1626 \liststore{#1}{\COOL@WeiSigma@arg@z@}%
1627 \liststore{#2}{\COOL@WeiSigma@arg@g@}%
1628 \listval{#2}{0}%
1629 \ifthenelse{\NOT \value{\COOL@listpointer} = 2}
1630 {%
1631 \PackageError{cool}{Invalid Argument}%

```

```

1632 {'WeierstrassSigma' second argument must be%
1633 a comma separated list of length 2}%
1634 }%
1635 % Else
1636 {%
1637 \listval{#1}{0}%
1638 \ifthenelse{\value{COOL@listpointer} = 1}%
1639 {%
1640 \sigma\!\!\inp{#1; #2}%
1641 }%
1642 % ElseIf
1643 { \ifthenelse{\value{COOL@listpointer} = 2}%
1644 {%
1645 \sigma_{\COOL@WeiSigma@arg@z@i}\!\!\inp{\COOL@WeiSigma@arg@z@ii; #2}%
1646 }%
1647 % Else
1648 {%
1649 \PackageError{cool}{Invalid Argument}%
1650 {'WeierstrassSigma' first argument must be%
1651 a comma separated list of length 1 or 2}%
1652 }}%
1653 }%
1654 }
1655 \newcommand{\WeiSigma}[2]{\WeierstrassSigma{#1}{#2}}

\AssocWeierstrassSigma Associated Weierstrass Sigma Function
\AssocWeierstrassSigma{n}{z}{g_2}{g_3}  $\sigma_n(z; g_2, g_3)$ 
\WeiSigma{n,z}{g_2,g_3}  $\sigma_n(z; g_2, g_3)$ 
1656 \newcommand{\AssocWeierstrassSigma}[4]{\WeierstrassSigma{#1,#2}{#3,#4}}

\WeierstrassZeta Weierstrass Zeta Function
\WeiZeta \WeierstrassZeta{z}{g_2,g_3}  $\zeta(z; g_2, g_3)$ 
\WeiZeta{z}{g_2,g_3}  $\zeta(z; g_2, g_3)$ 
1657 \newcommand{\COOL@notation@WeierstrassZetaParen}{p}%
1658 \newcommand{\WeierstrassZeta}[2]{%
1659 \listval{#2}{0}%
1660 \ifthenelse{\NOT \value{COOL@listpointer} = 2}%
1661 {%
1662 \PackageError{cool}{Invalid Argument}%
1663 {'WeierstrassZeta' second argument must be%
1664 a comma separated list of length 2}%
1665 }%
1666 % Else
1667 {%
1668 \zeta\!\!\COOL@decide@paren{WeierstrassZeta}{#1; #2}%
1669 }%
1670 }
1671 \newcommand{\WeiZeta}[2]{\WeierstrassZeta{#1}{#2}}

```

```

\WeierstrassHalfPeriods Weierstrass half-periods
\WeiHalfPeriods \WeierstrassHalfPeriods{g_2,g_3} { $\omega_1(g_2,g_3), \omega_3(g_2,g_3)$ }
\WeiHalfPeriods{g_2,g_3} { $\omega_1(g_2,g_3), \omega_3(g_2,g_3)$ }

1672 \newcommand{\WeierstrassHalfPeriods}[1]{%
1673 \listval{#1}{0}%
1674 \ifthenelse{\NOT \value{COOL@listpointer} = 2}%
1675 {%
1676 \PackageError{cool}{Invalid Argument}%
1677 {'WeierstrassHalfPeriods' can only accept%
1678 a comma separated list of length 2}%
1679 }%
1680 % Else
1681 {%
1682 \{ \omega_1!\!\!\in{#1}, \omega_3!\!\!\in{#1} \}%
1683 }%
1684 }
1685 \newcommand{\WeiHalfPeriods}[1]{\WeierstrassHalfPeriods{#1}}

\WeierstrassInvariants Weierstrass Invariants
\WeierstrassInvariants{\omega_1,\omega_3} { $g_2(\omega_1,\omega_3), g_3(\omega_1,\omega_3)$ }
\WeiInvars{\omega_1,\omega_3} { $g_2(\omega_1,\omega_3), g_3(\omega_1,\omega_3)$ }

1686 \newcommand{\WeierstrassInvariants}[1]{%
1687 \listval{#1}{0}%
1688 \ifthenelse{\NOT \value{COOL@listpointer} = 2}%
1689 {%
1690 \PackageError{cool}{Invalid Argument}%
1691 {'WeierstrassInvariants' can only accept%
1692 a comma separated list of length 2}%
1693 }%
1694 % Else
1695 {%
1696 \{ g_2!\!\!\in{#1}, g_3!\!\!\in{#1} \}%
1697 }%
1698 }
1699 \newcommand{\WeiInvars}[1]{\WeierstrassInvariants{#1}}

\COOL@hideOnSF Used to hide inputs or other when style is sf
sf short form
ff full form

1700 \newcommand{\COOL@hideOnSF}[2]
1701 {%
1702 \ifthenelse{ \equal{\csname COOL@notation@#1\endcsname}{sf} }%
1703 {%
1704 % Else
1705 {#2}%
1706 }

\WeierstrassPHalfPeriodValues Weierstrass elliptic function values at half-periods
\WeiPHalfPeriodVal

```

```

\Style{WeierstrassPHalfPeriodValuesDisplay=sf} (Default)
\WeierstrassPHalfPeriodValues{g_2,g_3}
\WeiPHalfPeriodVal{g_2,g_3}
{e_1,e_2,e_3}

\Style{WeierstrassPHalfPeriodValuesDisplay=ff}
\WeierstrassPHalfPeriodValues{g_2,g_3}
\WeiPHalfPeriodVal{g_2,g_3}
{e_1(g_2,g_3),e_2(g_2,g_3),e_3(g_2,g_3)}
1707 \newcommand{\COOL@notation@WeierstrassPHalfPeriodValuesDisplay}{sf}
1708 \newcommand{\WeierstrassPHalfPeriodValues}[1]
1709 {%
1710 \listval{#1}{0}%
1711 \ifthenelse{\NOT \value{COOL@listpointer} = 2}%
1712 {%
1713 \PackageError{cool}{Invalid Argument}%
1714 {'WeierstrassPHalfPeriodValues' can only accept%
1715 a comma separated list of length 2}%
1716 }%
1717 % Else
1718 {%
1719 \{ e_1\COOL@hideOnSF{WeierstrassPHalfPeriodValuesDisplay}{\!\inp{#1}},%
1720 e_2\COOL@hideOnSF{WeierstrassPHalfPeriodValuesDisplay}{\!\inp{#1}},%
1721 e_3\COOL@hideOnSF{WeierstrassPHalfPeriodValuesDisplay}{\!\inp{#1}}%
1722 \}%
1723 }%
1724 }
1725 \newcommand{\WeiPHalfPeriodVal}[1]{\WeierstrassPHalfPeriodValues{#1}}

WeierstrassZetaHalfPeriodValues Weierstrass zeta function values at half-periods
\WeiZetaHalfPeriodVal \Style{WeierstrassZetaHalfPeriodValuesDisplay=sf} (Default)
\WeierstrassZetaHalfPeriodValues{g_2,g_3}
\WeiZetaHalfPeriodVal{g_2,g_3}
{\eta_1,\eta_2,\eta_3}

\Style{WeierstrassZetaHalfPeriodValuesDisplay=ff}
\WeierstrassZetaHalfPeriodValues{g_2,g_3}
\WeiZetaHalfPeriodVal{g_2,g_3}
{\eta_1(g_2,g_3),\eta_2(g_2,g_3),\eta_3(g_2,g_3)}
1726 \newcommand{\COOL@notation@WeierstrassZetaHalfPeriodValuesDisplay}{sf}
1727 \newcommand{\WeierstrassZetaHalfPeriodValues}[1]
1728 {%
1729 \listval{#1}{0}%
1730 \ifthenelse{\NOT \value{COOL@listpointer} = 2}%
1731 {%
1732 \PackageError{cool}{Invalid Argument}%
1733 {'WeierstrassZetaHalfPeriodValues' can only accept%
1734 a comma separated list of length 2}%

```

```

1735 }%
1736 % Else
1737 {%
1738 \{%
1739 \eta_1\COOL@hideOnSF%
1740 {WeierstrassZetaHalfPeriodValuesDisplay}{\!\inp{#1}},%
1741 \eta_2\COOL@hideOnSF%
1742 {WeierstrassZetaHalfPeriodValuesDisplay}{\!\inp{#1}},%
1743 \eta_3\COOL@hideOnSF%
1744 {WeierstrassZetaHalfPeriodValuesDisplay}{\!\inp{#1}}%
1745 }%
1746 }%
1747 }
1748 \newcommand{\WeiZetaHalfPeriodVal}[1]%
1749 {\WeierstrassZetaHalfPeriodValues{#1}}

```

1.3.31 Jacobi Functions

```

\JacobiAmplitude Amplitude, \JacobiAmplitude{z}{m},  $\text{am}(z|m)$ 
1750 \newcommand{\COOL@notation@JacobiAmplitudeParen}{p}
1751 \DeclareMathOperator{\JacobiAmplitudeSymb}{am}
1752 \newcommand{\JacobiAmplitude}[2]{%
1753 \JacobiAmplitudeSymb\!\COOL@decide@paren%
1754 {JacobiAmplitude}{#1 \left| \, , #2 \right.\!\!\}}%
1755 }

\JacobiCD Jacobi elliptic function and its inverse
\JacobiCDInv \JacobiCD{z}{m}  $\text{cd}(z|m)$ 
\JacobiCDInv{z}{m}  $\text{cd}^{-1}(z|m)$ 
1756 \newcommand{\COOL@notation@JacobiCDParen}{p}
1757 \newcommand{\COOL@notation@JacobiCDInvParen}{p}
1758 \DeclareMathOperator{\JacobiCDSymb}{cd}
1759 \newcommand{\JacobiCD}[2]{%
1760 \JacobiCDSymb\!\COOL@decide@paren%
1761 {JacobiCD}{#1 \left| \, , #2 \right.\!\!\}}%
1762 }
1763 \newcommand{\JacobiCDInv}[2]{%
1764 \JacobiCDSymb^{-1}\!\COOL@decide@paren%
1765 {JacobiCDInv}{#1 \left| \, , #2 \right.\!\!\}}%
1766 }

\JacobiCN Jacobi elliptic function and its inverse
\JacobiCNInv \JacobiCN{z}{m}  $\text{cn}(z|m)$ 
\JacobiCNInv{z}{m}  $\text{cn}^{-1}(z|m)$ 
1767 \newcommand{\COOL@notation@JacobiCNParen}{p}
1768 \newcommand{\COOL@notation@JacobiCNInvParen}{p}
1769 \DeclareMathOperator{\JacobiCNSymb}{cn}
1770 \newcommand{\JacobiCN}[2]{%
1771 \JacobiCNSymb\!\COOL@decide@paren%

```

```

1772 {JacobiCN}{#1 \left| \, , #2 \right.\!\!\}%
1773 }
1774 \newcommand{\JacobiCNInv}[2]{%
1775 \JacobiCNSymb^{-1}\!\COOL@decide@paren%
1776 {JacobiCNInv}{#1 \left| \, , #2 \right.\!\!\}%
1777 }

\JacobiCS   Jacobi elliptic function and its inverse
\JacobiCSInv \JacobiCS{z}{m}      cs(z|m)
              \JacobiCSInv{z}{m}  cs^{-1}(z|m)

1778 \newcommand{\COOL@notation@JacobiCSParen}{p}
1779 \newcommand{\COOL@notation@JacobiCSInvParen}{p}
1780 \DeclareMathOperator{\JacobiCSSymb}{cs}
1781 \newcommand{\JacobiCS}[2]{%
1782 \JacobiCSSymb\!\COOL@decide@paren%
1783 {JacobiCS}{#1 \left| \, , #2 \right.\!\!\}%
1784 }
1785 \newcommand{\JacobiCSInv}[2]{%
1786 \JacobiCSSymb^{-1}\!\COOL@decide@paren%
1787 {JacobiCSInv}{#1 \left| \, , #2 \right.\!\!\}%
1788 }

\JacobiDC   Jacobi elliptic function and its inverse
\JacobiDCInv \JacobiDC{z}{m}      dc(z|m)
              \JacobiDCInv{z}{m}  dc^{-1}(z|m)

1789 \newcommand{\COOL@notation@JacobiDCParen}{p}
1790 \newcommand{\COOL@notation@JacobiDCInvParen}{p}
1791 \DeclareMathOperator{\JacobiDCSymb}{dc}
1792 \newcommand{\JacobiDC}[2]{%
1793 \JacobiDCSymb\!\COOL@decide@paren%
1794 {JacobiDC}{#1 \left| \, , #2 \right.\!\!\}%
1795 }
1796 \newcommand{\JacobiDCInv}[2]{%
1797 \JacobiDCSymb^{-1}\!\COOL@decide@paren%
1798 {JacobiDCInv}{#1 \left| \, , #2 \right.\!\!\}%
1799 }

\JacobiDN   Jacobi elliptic function and its inverse
\JacobiDNInv \JacobiDN{z}{m}      dn(z|m)
              \JacobiDNInv{z}{m}  dn^{-1}(z|m)

1800 \newcommand{\COOL@notation@JacobiDNParen}{p}
1801 \newcommand{\COOL@notation@JacobiDNInvParen}{p}
1802 \DeclareMathOperator{\JacobiDNSymb}{dn}
1803 \newcommand{\JacobiDN}[2]{%
1804 \JacobiDNSymb\!\COOL@decide@paren%
1805 {JacobiDN}{#1 \left| \, , #2 \right.\!\!\}%
1806 }
1807 \newcommand{\JacobiDNInv}[2]{%
1808 \JacobiDNSymb^{-1}\!\COOL@decide@paren%

```

```

1809 {JacobiDNInv}{#1 \left| \, #2 \right.\!\!\}%
1810 }

\JacobiDS Jacobi elliptic function and its inverse
\JacobiDSInv \JacobiDS{z}{m} ds(z|m)
\JacobiDSInv{z}{m} ds^{-1}(z|m)
1811 \newcommand{\COOL@notation@JacobiDSParen}{p}
1812 \newcommand{\COOL@notation@JacobiDSInvParen}{p}
1813 \DeclareMathOperator{\JacobiDSymb}{ds}
1814 \newcommand{\JacobiDS}[2]{%
1815 \JacobiDSymb\!\COOL@decide@paren%
1816 {JacobiDS}{#1 \left| \, #2 \right.\!\!\}%
1817 }
1818 \newcommand{\JacobiDSInv}[2]{%
1819 \JacobiDSymb^{-1}\!\COOL@decide@paren%
1820 {JacobiDSInv}{#1 \left| \, #2 \right.\!\!\}%
1821 }

\JacobiNC Jacobi elliptic function and its inverse
\JacobiNCInv \JacobiNC{z}{m} nc(z|m)
\JacobiNCInv{z}{m} nc^{-1}(z|m)
1822 \newcommand{\COOL@notation@JacobiNCParen}{p}
1823 \newcommand{\COOL@notation@JacobiNCInvParen}{p}
1824 \DeclareMathOperator{\JacobiNCSymb}{nc}
1825 \newcommand{\JacobiNC}[2]{%
1826 \JacobiNCSymb\!\COOL@decide@paren%
1827 {JacobiNC}{#1 \left| \, #2 \right.\!\!\}%
1828 }
1829 \newcommand{\JacobiNCInv}[2]{%
1830 \JacobiNCSymb^{-1}\!\COOL@decide@paren%
1831 {JacobiNCInv}{#1 \left| \, #2 \right.\!\!\}%
1832 }

\JacobiND Jacobi elliptic function and its inverse
\JacobiNDInv \JacobiND{z}{m} nd(z|m)
\JacobiNDInv{z}{m} nd^{-1}(z|m)
1833 \newcommand{\COOL@notation@JacobiNDParen}{p}
1834 \newcommand{\COOL@notation@JacobiNDInvParen}{p}
1835 \DeclareMathOperator{\JacobiNDSymb}{nd}
1836 \newcommand{\JacobiND}[2]{%
1837 \JacobiNDSymb\!\COOL@decide@paren%
1838 {JacobiND}{#1 \left| \, #2 \right.\!\!\}%
1839 }
1840 \newcommand{\JacobiNDInv}[2]{%
1841 \JacobiNDSymb^{-1}\!\COOL@decide@paren%
1842 {JacobiNDInv}{#1 \left| \, #2 \right.\!\!\}%
1843 }

\JacobiNS Jacobi elliptic function and its inverse
\JacobiNSInv

```

```

\JacobiNS{z}{m}      ns(z|m)
\JacobiNSInv{z}{m}   ns-1(z|m)
1844 \newcommand{\COOL@notation@JacobiNSParen}{p}
1845 \newcommand{\COOL@notation@JacobiNSInvParen}{p}
1846 \DeclareMathOperator{\JacobiNSSymb}{ns}
1847 \newcommand{\JacobiNS}[2]{%
1848 \JacobiNSSymb\!\COOL@decide@paren%
1849 {JacobiNS}{#1 \left| \, , #2 \right.\!\!}%
1850 }
1851 \newcommand{\JacobiNSInv}[2]{%
1852 \JacobiNSSymb-1\!\COOL@decide@paren%
1853 {JacobiNSInv}{#1 \left| \, , #2 \right.\!\!}%
1854 }

\JacobiSC Jacobi elliptic function and its inverse
\JacobiSCInv \JacobiSC{z}{m}      sc(z|m)
\JacobiSCInv{z}{m}   sc-1(z|m)
1855 \newcommand{\COOL@notation@JacobiSCParen}{p}
1856 \newcommand{\COOL@notation@JacobiSCInvParen}{p}
1857 \DeclareMathOperator{\JacobiSCSymb}{sc}
1858 \newcommand{\JacobiSC}[2]{%
1859 \JacobiSCSymb\!\COOL@decide@paren%
1860 {JacobiSC}{#1 \left| \, , #2 \right.\!\!}%
1861 }
1862 \newcommand{\JacobiSCInv}[2]{%
1863 \JacobiSCSymb-1\!\COOL@decide@paren%
1864 {JacobiSCInv}{#1 \left| \, , #2 \right.\!\!}%
1865 }

\JacobiSD Jacobi elliptic function and its inverse
\JacobiSDInv \JacobiSD{z}{m}      sd(z|m)
\JacobiSDInv{z}{m}   sd-1(z|m)
1866 \newcommand{\COOL@notation@JacobiSDParen}{p}
1867 \newcommand{\COOL@notation@JacobiSDInvParen}{p}
1868 \DeclareMathOperator{\JacobiSDSymb}{sd}
1869 \newcommand{\JacobiSD}[2]{%
1870 \JacobiSDSymb\!\COOL@decide@paren%
1871 {JacobiSD}{#1 \left| \, , #2 \right.\!\!}%
1872 }
1873 \newcommand{\JacobiSDInv}[2]{%
1874 \JacobiSDSymb-1\!\COOL@decide@paren%
1875 {JacobiSDInv}{#1 \left| \, , #2 \right.\!\!}%
1876 }

\JacobiSN Jacobi elliptic function and its inverse
\JacobiSNInv \JacobiSN{z}{m}      sn(z|m)
\JacobiSNInv{z}{m}   sn-1(z|m)
1877 \newcommand{\COOL@notation@JacobiSNParen}{p}

```

```

1878 \newcommand{\COOL@notation@JacobiSNInvParen}{p}
1879 \DeclareMathOperator{\JacobiSNSymb}{sn}
1880 \newcommand{\JacobiSN}[2]{%
1881 \JacobiSNSymb\!\COOL@decide@paren%
1882 {JacobiSN}{#1 \left| \, , #2 \right.\!\!}%
1883 }
1884 \newcommand{\JacobiSNInv}[2]{%
1885 \JacobiSNSymb^{-1}\!\COOL@decide@paren%
1886 {JacobiSNInv}{#1 \left| \, , #2 \right.\!\!}%
1887 }

```

1.3.32 Modular Functions

\DedekindEta Dedekind eta modular function, $\backslash\text{DedekindEta}\{z\}$, $\eta(z)$

```

1888 \newcommand{\COOL@notation@DedekindEtaParen}{p}
1889 \newcommand{\DedekindEta}[1]{\eta\!\COOL@decide@paren{DedekindEta}{#1}}

```

\KleinInvariantJ Klein invariant modular function, $\backslash\text{KleinInvariantJ}\{z\}$, $J(z)$

```

1890 \newcommand{\COOL@notation@KleinInvariantJParen}{p}
1891 \newcommand{\KleinInvariantJ}[1]{%
1892 {J\!\COOL@decide@paren{KleinInvariantJ}{#1}}

```

\ModularLambda Modular lambda function, $\backslash\text{ModularLambda}\{z\}$, $\lambda(z)$

```

1893 \newcommand{\COOL@notation@ModularLambdaParen}{p}
1894 \newcommand{\ModularLambda}[1]{%
1895 {\lambda\!\COOL@decide@paren{ModularLambda}{#1}}

```

\EllipticNomeQ Nome and its inverse

\EllipticNomeQInv

| | |
|--|-------------|
| $\backslash\text{EllipticNomeQ}\{m\}$ | $q(m)$ |
| $\backslash\text{EllipticNomeQInv}\{m\}$ | $q^{-1}(m)$ |

```

1896 \newcommand{\COOL@notation@EllipticNomeQParen}{p}
1897 \newcommand{\COOL@notation@EllipticNomeQInvParen}{p}
1898 \newcommand{\EllipticNomeQ}[1]{%
1899 {q\!\COOL@decide@paren{EllipticNomeQ}{#1}}
1900 \newcommand{\EllipticNomeQInv}[1]{%
1901 {q^{-1}\!\COOL@decide@paren{EllipticNomeQ}{#1}}

```

1.3.33 Arithmetic Geometric Mean

\ArithGeoMean Arithmetic Geometric Mean

| | |
|---|--------------------|
| $\backslash\text{ArithGeoMean}\{a\}\{b\}$ | $\text{agm}(a, b)$ |
| $\backslash\text{AGM}\{a\}\{b\}$ | $\text{agm}(a, b)$ |

```

1902 \newcommand{\COOL@notation@ArithGeoMeanParen}{p}
1903 \DeclareMathOperator{\ArithGeoMeanSymb}{agm}
1904 \newcommand{\ArithGeoMean}[2]{%
1905 {\ArithGeoMeanSymb\!\COOL@decide@paren{ArithGeoMean}{#1, #2}}
1906 \newcommand{\AGM}[2]{\ArithGeoMean{#1}{#2}}

```

1.3.34 Elliptic Exp and Log

```

\EllipticExp  Elliptic exponential
\EEExp      \EllipticExp{z}{a,b}    eexp(z; a, b)
           \EEExp{z}{a,b}          eexp(z; a, b)

1907 \newcommand{\COOL@notation@EllipticExpParen}{p}
1908 \DeclareMathOperator{\EllipticExpSymb}{eexp}
1909 \newcommand{\EllipticExp}[2]{%
1910 \liststore{#2}{COOL@EllipticExp@arg@}
1911 \listval{#2}{0}%
1912 \ifthenelse{\NOT \value{COOL@listpointer} = 2}%
1913 {%
1914 \PackageError{cool}{Invalid Argument}%
1915 {'EllipticExp' second argument must be
1916 a comma separated list of length 2}%
1917 }%
1918 % Else
1919 {%
1920 \EllipticExpSymb!\COOL@decide@paren{EllipticExp}{#1; #2}%
1921 }%
1922 }
1923 \newcommand{\EEExp}[2]{\EllipticExp{#1}{#2}}

\EllipticLog  Elliptic logarithm
\ELog        \EllipticLog{z_1,z_2}{a,b}    elog(z_1, z_2; a, b)
           \ELog{z_1,z_2}{a,b}          elog(z_1, z_2; a, b)

1924 \newcommand{\COOL@notation@EllipticLogParen}{p}
1925 \DeclareMathOperator{\EllipticLogSymb}{elog}
1926 \newcommand{\EllipticLog}[2]{%
1927 \liststore{#1}{COOL@EllipticLog@arg@z@}%
1928 \liststore{#2}{COOL@EllipticLog@arg@a@}%
1929 \listval{#1}{0}%
1930 \ifthenelse{\NOT \value{COOL@listpointer} = 2}%
1931 {%
1932 \PackageError{cool}{Invalid Argument}%
1933 {'EllipticLog' first argument must be
1934 a comma separated list of length 2}%
1935 }%
1936 % Else
1937 {%
1938 \listval{#2}{0}%
1939 \ifthenelse{\NOT \value{COOL@listpointer} = 2}%
1940 {%
1941 \PackageError{cool}{Invalid Argument}%
1942 {'EllipticLog' second argument must be%
1943 a comma separated list of length 2}%
1944 }%
1945 % Else
1946 {%

```

```

1947 \EllipticLogSymb\!\COOL@decide@paren{EllipticLog}{#1; #2}%
1948 }%
1949 }%
1950 }
1951 \newcommand{\ELog}[2]{\EllipticLog{#1}{#2}}

```

1.3.35 Zeta Functions

```

\RiemannZeta  Riemann Zeta Function
              \RiemannZeta{s}   $\zeta(s)$ 
              \Zeta{s}          $\zeta(s)$ 
1952 \newcommand{\RiemannZeta}[1]{\Zeta{#1}}

\HurwitzZeta  Hurwitz Zeta Function
              \HurwitzZeta{s}{a}   $\zeta(s, a)$ 
              \Zeta{s, a}          $\zeta(s, a)$ 
1953 \newcommand{\HurwitzZeta}[2]{\Zeta{#1, #2}}

\Zeta  Riemann and Hurwitz Zeta
       Riemann Zeta  \Zeta{s}   $\zeta(s)$ 
       Hurwitz Zeta  \Zeta{s, a}   $\zeta(s, a)$ 
1954 \newcommand{\COOL@notation@ZetaParen}{p}
1955 \newcommand{\Zeta}[1]{%
1956 \liststore{#1}{\COOL@Zeta@arg@}%
1957 \listval{#1}{0}% get the list length
1958 \ifthenelse{\value{\COOL@listpointer} = 2}%
1959 {%
1960 \zeta\!\COOL@decide@paren{Zeta}{\COOL@Zeta@arg@i, \COOL@Zeta@arg@ii}%
1961 }%
1962 % else
1963 {%
1964 \ifthenelse{\value{\COOL@listpointer} = 1}%
1965 {%
1966 \zeta\!\COOL@decide@paren{Zeta}{#1}%
1967 }%
1968 % else
1969 {%
1970 \PackageError{cool}{‘Zeta’ Invalid Argument}%
1971 {the Zeta function can only accept%
1972 a comma delimited list of length 1 or 2}
1973 }%
1974 }%
1975 }%

\RiemannSiegelTheta  Riemann-Siegel Theta Function, \RiemannSiegelTheta{z},  $\vartheta(z)$ 
1976 \newcommand{\COOL@notation@RiemannSiegelThetaParen}{p}
1977 \newcommand{\RiemannSiegelTheta}[1]%
1978 {\vartheta\!\COOL@decide@paren{RiemannSiegelTheta}{#1}}

```

```

\RiemannSiegelZ  Riemann-Siegel Z Function, \RiemannSiegelZ{z},  $Z(z)$ 
1979 \newcommand{\COOL@notation@RiemannSiegelZParen}{p}
1980 \newcommand{\RiemannSiegelZ}[1]{%
1981 {Z\!\COOL@decide@paren{RiemannSiegelZ}{#1}}

\StieltjesGamma  Stieltjes Constant, \StieltjesGamma{n},  $\gamma_n$ 
1982 \newcommand{\StieltjesGamma}[1]{\gamma_{#1}}

\LerchPhi  Lerch transcendent, \LerchPhi{z}{s}{a},  $\Phi(z, s, a)$ 
1983 \newcommand{\COOL@notation@LerchPhiParen}{p}
1984 \newcommand{\LerchPhi}[3]{\Phi\!\COOL@decide@paren{LerchPhi}{#1,#2,#3}}

```

1.3.36 Polylogarithms

```

\NielsenPolyLog  Nielsen Polylogarithm, \NielsenPolyLog{\nu}{p}{z},  $S_\nu^p(z)$ 
1985 \newcommand{\COOL@notation@NielsenPolyLogParen}{p}
1986 \newcommand{%
1987 \NielsenPolyLog}[3]{S_{#1}^{#2}\!\COOL@decide@paren{NielsenPolyLog}{#3}}%
1988 \COOL@decide@paren{NielsenPolyLog}{#3}%
1989 }

\PolyLog  Polylogarithm
          Nielsen PolyLog  \PolyLog{\nu,p,z}   $S_\nu^p(z)$ 
          PolyLog          \PolyLog{\nu,z}     $Li_\nu(z)$ 
1990 \newcommand{\COOL@notation@PolyLogParen}{p}
1991 \DeclareMathOperator{\PolyLogSymb}{Li}
1992 \newcommand{\PolyLog}[1]{%
1993 \liststore{#1}{COOL@PolyLog@arg@}%
1994 \listval{#1}{0}%
1995 \ifthenelse{\value{COOL@listpointer} = 3}{%
1996 {%
1997 \NielsenPolyLog{\COOL@PolyLog@arg@i}%
1998 {\COOL@PolyLog@arg@ii}{\COOL@PolyLog@arg@iii}%
1999 }%
2000 % else
2001 {%
2002 \ifthenelse{\value{COOL@listpointer} = 2 }{%
2003 {%
2004 \PolyLogSymb_{\COOL@PolyLog@arg@i}\!\COOL@decide@paren{PolyLog}{\COOL@PolyLog@arg@ii}%
2005 \COOL@decide@paren{PolyLog}{\COOL@PolyLog@arg@ii}%
2006 }%
2007 % else
2008 {%
2009 \PackageError{cool}{‘PolyLog’ Invalid Argument}%
2010 {This function returns either the Polylogarithm or the%
2011 Nielsen Polylogarithm. It therefore only accepts a comma%
2012 delimited list of length two or three (1 or 2 commas)}%
2013 }%
2014 }%

```

2015 }

\DiLog Dilogarithm (alias for \PolyLog{2,x}); \DiLog{x}, $\text{Li}_2(x)$
2016 \newcommand{\DiLog}[1]{\PolyLog{2,#1}}

1.3.37 Mathieu Functions

\MathieuC Even Mathieu Function, \MathieuC{a}{q}{z}, $\text{Ce}(a, q, z)$
2017 \newcommand{\COOL@notation@MathieuCParen}{p}
2018 \DeclareMathOperator{\MathieuCSymb}{Ce}
2019 \newcommand{\MathieuC}[3]%
2020 {\MathieuCSymb!\COOL@decide@paren{\MathieuC}{#1,#2,#3}}

\MathieuS Odd Mathieu Function, \MathieuS{a}{q}{z}, $\text{Se}(a, q, z)$
2021 \newcommand{\COOL@notation@MathieuSParen}{p}
2022 \DeclareMathOperator{\MathieuSSymb}{Se}
2023 \newcommand{\MathieuS}[3]%
2024 {\mathord{\MathieuSSymb}\!\COOL@decide@paren{\MathieuS}{#1,#2,#3}}

1.3.38 Mathieu Characteristics

\MathieuCharacteristicA Characteristic Value of Even Mathieu Function
\MathieuCharisticA \MathieuCharacteristicA{r}{q} $a_r(q)$
\MathieuCharisticA{r}{q} $a_r(q)$
2025 \newcommand{\COOL@notation@MathieuCharacteristicAParen}{p}
2026 \newcommand{\MathieuCharacteristicA}[2]%
2027 {a_{#1}}!\COOL@decide@paren{\MathieuCharacteristicA}{#2}}
2028 \newcommand{\MathieuCharisticA}[2]{\MathieuCharacteristicA{#1}{#2}}

\MathieuCharacteristicB Characteristic Value of Even Mathieu Fuction
\MathieuCharisticB \MathieuCharacteristicB{r}{q} $b_r(q)$
\MathieuCharisticB{r}{q} $b_r(q)$
2029 \newcommand{\COOL@notation@MathieuCharacteristicBParen}{p}
2030 \newcommand{\MathieuCharacteristicB}[2]%
2031 {b_{#1}}!\COOL@decide@paren{\MathieuCharacteristicB}{#2}}
2032 \newcommand{\MathieuCharisticB}[2]{\MathieuCharacteristicB{#1}{#2}}

\MathieuCharacteristicExponent Characteristic Exponent of a Mathieu Fuction
\MathieuCharisticExp \MathieuCharateristicExponent{a}{q} $r(a, q)$
\MathieuCharisticExp{a}{q} $r(a, q)$
2033 \newcommand{\COOL@notation@MathieuCharacteristicExponentParen}{p}
2034 \newcommand{\MathieuCharacteristicExponent}[2]%
2035 {r!\COOL@decide@paren{\MathieuCharacteristicExponent}{#1,#2}}
2036 \newcommand{\MathieuCharisticExp}[2]%
2037 {\MathieuCharacteristicExponent{#1}{#2}}

1.3.39 Complex variables

```

\Abs Absolute value, \Abs{z}, |z|
2038 \newcommand{\Abs}[1]{ \left|#1\right| }

\Arg Argument, \Arg{z}, arg(z)
2039 \newcommand{\Arg}[1]{ \arg\!\!\in{#1} }

\Conjugate Complex Conjugate
\Conj          \Conj{z}          z*
               \Conjugate{z}      z*

2040 \def\COOL@notation@Conjugate{star}
2041 \newcommand{\COOL@notation@ConjugateParen}{inv}
2042 \newcommand{\Conjugate}[1]{\Conj{#1}}
2043 \newcommand{\Conj}[1]{%
2044 \ifthenelse{\equal{\COOL@notation@Conjugate}{bar}}%
2045 {%
2046 \bar{#1}%
2047 }%
2048 % ElseIf
2049 { \ifthenelse{\equal{\COOL@notation@Conjugate}{overline}}%
2050 {%
2051 \overline{#1}%
2052 }%
2053 % ElseIf
2054 { \ifthenelse{\equal{\COOL@notation@Conjugate}{star}}%
2055 {%
2056 \COOL@decide@paren{Conjugate}{#1}^*%
2057 }%
2058 % Else
2059 {%
2060 \PackageError{cool}{Invalid Option Sent}%
2061 {'Conjugate' can only be set at 'star', 'bar', or 'overline'}}%
2062 }%
2063 }%
2064 }

\Real Real Part, \Real{z}, Re z
2065 \newcommand{\COOL@notation@RealParen}{none}
2066 \DeclareMathOperator{\RealSymb}{Re}
2067 \newcommand{\Real}[1]{%
    we put a space if there is no parentheses, or leave it out if there are
2068 \ifthenelse{\equal{\COOL@notation@ImagParen}{none}}%
2069 {%
2070 \RealSymb{#1}%
2071 }%
2072 % Else
2073 {%
2074 \RealSymb\!\!\COOL@decide@paren{Imag}{#1}%

```

```

2075 }%
2076 }

\Imag Imaginary Part, \Imag{z},  $\operatorname{Im} z$ 
2077 \newcommand{\COOL@notation@ImagParen}{none}
2078 \DeclareMathOperator{\ImagSymb}{\operatorname{Im}}
2079 \newcommand{\Imag}[1]{%
    we put a space if there is no parentheses, or leave it out if there are
2080 \ifthenelse{\equal{\COOL@notation@ImagParen}{none}}{%
2081 {%
2082 \ImagSymb{#1}%
2083 }%
2084 % Else
2085 {%
2086 \ImagSymb\!\COOL@decide@paren{Imag}{#1}%
2087 }%
2088 }

```

```

\Sign Sign function, \Sign{x},  $\operatorname{sgn}(x)$ 
2089 \newcommand{\COOL@notation@SignParen}{p}
2090 \newcommand{\Sign}[1]{\operatorname{sgn}\!\COOL@decide@paren{Sign}{#1}}

```

1.3.40 Number Theory Functions

```

\FactorInteger Prime decomposition, \Factors{n},  $\operatorname{factors}(n)$ 
\Factors 2091 \newcommand{\COOL@notation@FactorIntegerParen}{p}
2092 \DeclareMathOperator{\FactorIntegerSymb}{\operatorname{factors}}
2093 \newcommand{\FactorInteger}[1]{%
2094 {\FactorIntegerSymb\!\COOL@decide@paren{FactorInteger}{#1}}
2095 \newcommand{\Factors}[1]{\FactorInteger{#1}}

```

```

\Divisors Divisors, \Divisors{n},  $\operatorname{divisors}(n)$ 
2096 \newcommand{\COOL@notation@DivisorsParen}{p}
2097 \DeclareMathOperator{\DivisorsSymb}{\operatorname{divisors}}
2098 \newcommand{\Divisors}[1]{%
2099 {\mathord{\DivisorsSymb}\!\COOL@decide@paren{Divisors}{#1}}

```

```

\Prime The  $n$ th Prime, \Prime{n},  $\operatorname{prime}(n)$ 
2100 \newcommand{\COOL@notation@PrimeParen}{p}
2101 \DeclareMathOperator{\PrimeSymb}{\operatorname{prime}}
2102 \newcommand{\Prime}[1]{%
2103 {\mathord{\PrimeSymb}\!\COOL@decide@paren{Prime}{#1}}

```

```

\PrimePi Prime counting function, \PrimePi{x},  $\pi(x)$ 
2104 \newcommand{\COOL@notation@PrimePiParen}{p}
2105 \newcommand{\PrimePi}[1]{\pi\!\COOL@decide@paren{PrimePi}{#1}}

```

```

\DivisorSigma  Sum of divisor powers, \DivisorSigma{k}{n},  $\sigma_k(n)$ 
2106 \newcommand{\COOL@notation@DivisorSigmaParen}{p}
2107 \newcommand{\DivisorSigma}[2]%
2108 {\sigma_{#1}\!\!\COOL@decide@paren{DivisorSigma}{#2}}

\EulerPhi  Euler Totient Function, \EulerPhi{x},  $\varphi(x)$ 
2109 \newcommand{\COOL@notation@EulerPhiParen}{p}
2110 \newcommand{\EulerPhi}[1]{\varphi\!\!\COOL@decide@paren{EulerPhi}{#1}}

\MoebiusMu  Moebius Function, \MoebiusMu{x},  $\mu(x)$ 
2111 \newcommand{\COOL@notation@MoebiusMuParen}{p}
2112 \newcommand{\MoebiusMu}[1]{\mu\!\!\COOL@decide@paren{MoebiusMu}{#1}}

\JacobiSymbol  Jacobi Symbol, \JacobiSymbol{n}{m},  $\left(\frac{n}{m}\right)$ 
2113 \newcommand{\JacobiSymbol}[2]{\inp{\frac{#1}{#2}}}

\CarmichaelLambda  Carmichael Lambda Function, \CarmichaelLambda{x},  $\lambda(x)$ 
2114 \newcommand{\COOL@notation@CarmichaelLambdaParen}{p}
2115 \newcommand{\CarmichaelLambda}[1]%
2116 {\lambda\!\!\COOL@decide@paren{CarmichaelLambda}{#1}}

\DigitCount  Count the digits of an integer n for a given base b
               \DigitCount{n}{b}
                $\{s_b^{(1)}(n), s_b^{(2)}(n), \dots, s_b^{(b)-1}(n), s_b^{(0)}(n)\}$ 
2117 \newcommand{\DigitCount}[2]{%
2118 \isint{#2}%
2119 \ifthenelse{\boolean{COOL@isint}}{%
2120 {%
2121 \{%
2122 \setcounter{COOL@ct}{#2}%
2123 \addtocounter{COOL@ct}{-1}%
2124 \forLoop{1}{\arabic{COOL@ct}}{COOL@ct}%
2125 {%
2126 s^{\arabic{COOL@ct}}_{#2}\!\!\inp{#1},
2127 }%
2128 s^{\inp{0}}_{#2}\!\!\inp{#1}%
2129 \}%
2130 }%
2131 % else
2132 {%
2133 \{%
2134 s^{\inp{1}}_{#2}\!\!\inp{#1},%
2135 s^{\inp{2}}_{#2}\!\!\inp{#1},%
2136 \ldots,%
2137 s^{\inp{#2} - 1}_{#2}\!\!\inp{#1},%
2138 s^{\inp{0}}_{#2}\!\!\inp{#1}%
2139 \}%
2140 }%
2141 }

```

1.3.41 Generalized Functions

`\DiracDelta` Dirac Delta Function, `\DiracDelta{x}`, $\delta(x)$

```

2142 \newcommand{\COOL@notation@DiracDeltaParen}{p}
2143 \newcommand{\DiracDelta}[1]{\delta\!\!\COOL@decide@paren{DiracDelta}{#1}}
```

`\HeavisideStep` Heaviside Step Function

`\UnitStep` `\HeavisideStep{x}` $\theta(x)$
`\UnitStep{x}` $\theta(x)$

```

2144 \newcommand{\COOL@notation@HeavisideStepParen}{p}
2145 \newcommand{\HeavisideStep}[1]{%
2146 {\theta\!\!\COOL@decide@paren{HeavisideStep}{#1}}
2147 \newcommand{\UnitStep}[1]{\HeavisideStep{#1}}
```

1.3.42 Calculus

`\COOL@notation@DDisplayFunc` Both `\D` and `\pderiv` are controlled by these keys.

`\COOL@notation@DShorten` `DDisplayFunc` controls how the function is displayed, it can take the values:

inset Display as $\frac{df}{dx}$
outset Display as $\frac{d}{dx}f$

`DShorten` is for multiple derivatives. it can take the values

true force derivatives to be consolidated, as in $\frac{d^2}{dxdy}f$
false expand derivatives as in $\frac{d}{dx}\frac{d}{dx}f$

```

2148 \newcounter{COOL@multideriv}
2149 \newcommand{\COOL@notation@DDisplayFunc}{inset}
2150 \newcommand{\COOL@notation@DShorten}{true}
```

`\COOL@derivative` Both `\D` and `\pderiv` have the same basic operation, so a macro is defined that does the internals

`\COOL@derivative{\langle derivative power(s)\rangle}{\langle function\rangle}{\langle wrt\rangle}{\langle symbol\rangle}`
`\langle wrt\rangle` is a comma separated list of length ≥ 1 .
`\langle symbol\rangle` is passed by `\D` or `\pderiv` and is ‘d’ or ‘ ∂ ’ respectively

| | |
|---|--|
| $\backslash\text{COOL@derivative}\{2,3\}\{f\}\{x,y,z\}\{d\}$ | $\frac{d^8 f}{dx^2 dy^3 dz^3}$ |
| $\backslash\text{COOL@derivative}\{2,3,4,5\}\{f\}\{x,y,z\}\{d\}$ | $\frac{d^9 f}{dx^2 dy^3 dz^4}$ |
| $\backslash\text{COOL@derivative}\{2,n,1\}\{f\}\{x,y,z\}\{d\}$ | $\frac{d^{2+n+1} f}{dx^2 dy^n dz}$ |
| $\backslash\text{COOL@derivative}\{2,n\}\{f\}\{x,y,z\}\{d\}$ | $\frac{d^{2+n+n} f}{dx^2 dy^n dz^n}$ |
| $\backslash\text{Style}\{\text{DDisplayFunc}=\text{outset}\}$ | |
| $\backslash\text{COOL@derivative}\{2,n\}\{f\}\{x,y,z\}\{d\}$ | $\frac{d^{2+n+n} f}{dx^2 dy^n dz^n}$ |
| $\backslash\text{Style}\{\text{DShorten}=\text{false}, \text{DDisplayFunc}=\text{inset}\}$ | |
| $\backslash\text{COOL@derivative}\{2,n\}\{f\}\{x,y,z\}\{d\}$ | $\frac{d^2}{dx^2} \frac{d^n}{dy^n} \frac{d^n}{dz^n} f$ |
| $\backslash\text{COOL@derivative}\{2,3,4,5\}\{f\}\{x,y,z\}\{d\}$ | $\frac{d^2}{dx^2} \frac{d^3}{dy^3} \frac{d^4}{dz^4} f$ |
| $\backslash\text{Style}\{\text{DShorten}=\text{false}, \text{DDisplayFunc}=\text{outset}\}$ | |
| $\backslash\text{COOL@derivative}\{2,n\}\{f\}\{x,y,z\}\{d\}$ | $\frac{d^2}{dx^2} \frac{d^n}{dy^n} \frac{d^n}{dz^n} f$ |

2151 $\backslash\text{newcommand}\{\backslash\text{COOL@derivative}\}[4]\{\%$
 Get the length of $\langle wrt \rangle$ argument. $\backslash\text{listval}\{\#3\}\{0\}$ gives the length of the list
 since lists begin indexing at 1.
 2152 $\backslash\text{listval}\{\#3\}\{0\}\%$
 2153 $\backslash\text{setcounter}\{\text{COOL@listlen}\}\{\backslash\text{value}\{\text{COOL@listpointer}\}\}\%$
 Store the $\langle wrt \rangle$ list and get the length of $\langle derivative\ power(s) \rangle$.
 2154 $\backslash\text{liststore}\{\#3\}\{\text{COOL@deriv@wrt@}\}\%$
 2155 $\backslash\text{listval}\{\#1\}\{0\}\%$
 2156 $\backslash\text{setcounter}\{\text{COOL@ct}\}\{\backslash\text{value}\{\text{COOL@listpointer}\}\}\%$
 2157 $\backslash\text{ifthenelse}\{\backslash\text{value}\{\text{COOL@ct}\}>\backslash\text{value}\{\text{COOL@listlen}\}\}\%$
 2158 $\{\backslash\text{setcounter}\{\text{COOL@ct}\}\{\backslash\text{value}\{\text{COOL@listlen}\}\}\}\{\}\%$
 2159 $\backslash\text{liststore}\{\#1\}\{\text{COOL@deriv@powers@}\}\%$
 Check to see if all of the powers are integers—if they are, then we may sum them
 in the usual sense
 2160 $\backslash\text{isint}\{\backslash\text{COOL@deriv@powers@i}\}\%$
 2161 $\backslash\text{setcounter}\{\text{COOL@multideriv}\}\{2\}\%$
 2162 $\backslash\text{whiledo}\{\ \backslash\text{boolean}\{\text{COOL@isint}\}\ \backslash\text{AND}$
 2163 $\backslash\text{NOT}\ \backslash\text{value}\{\text{COOL@multideriv}\}>\backslash\text{value}\{\text{COOL@ct}\}\ \}\%$
 2164 $\{\%$
 2165 $\backslash\text{def}\backslash\text{COOL@tempd}\%$
 2166 $\{\backslash\text{csname}\ \text{COOL@deriv@powers@}\backslash\text{roman}\{\text{COOL@multideriv}\}\backslash\text{endcsname}\}\%$
 2167 $\backslash\text{isint}\{\backslash\text{COOL@tempd}\}\%$
 2168 $\backslash\text{stepcounter}\{\text{COOL@multideriv}\}\%$
 2169 $\}\%$
 If the length of $\langle derivative\ power(s) \rangle$ is less than the length of $\langle wrt \rangle$, then we
 assume that the last value applies to *all* the remaining derivatives.
 2170 $\backslash\text{ifthenelse}\{\ \backslash\text{equal}\{\backslash\text{COOL@notation@DShorten}\}\{\text{true}\}\ \backslash\text{AND}$
 2171 $\backslash\text{equal}\{\backslash\text{COOL@notation@DDisplayFunc}\}\{\text{inset}\}\ \}\%$

```

2172 {%
2173 \ifthenelse{ \boolean{COOL@isint} }{%
2174 {%
2175 \def\COOL@temp@D@bot{%
2176 \setcounter{COOL@ct}{0}%
2177 \forLoop{1}{\value{COOL@ct}}{COOL@multideriv}%
2178 {%
2179 \edef\COOL@power@temp%
2180 {\csname COOL@deriv@powers@\roman{COOL@multideriv}\endcsname}%
2181 \edef\COOL@wrt@temp%
2182 {\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
2183 \addtocounter{COOL@ct}{\COOL@power@temp}%
2184 \ifthenelse{ \value{COOL@multideriv}=1 }{%
2185 {\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}%
2186 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2187 {%
2188 \edef\COOL@temp@D@bot%
2189 {\COOL@temp@D@bot #4 \COOL@wrt@temp}%
2190 }%
2191 % Else
2192 {%
2193 \edef\COOL@temp@D@bot%
2194 {\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
2195 }%
2196 }%

```

we're done with the length of the $\langle derivative\ power(s) \rangle$ argument, and we want to start at it + 1 to add the remainders

```

2197 \ifthenelse{\value{COOL@ct}<\value{COOL@listlen}}{%
2198 {%
2199 \edef\COOL@power@temp%
2200 {\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
2201 \stepcounter{COOL@ct}%
2202 \forLoop{\value{COOL@ct}}{\value{COOL@listlen}}{COOL@multideriv}%
2203 {%
2204 \edef\COOL@wrt@temp%
2205 {\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
2206 \addtocounter{COOL@ct}{\COOL@power@temp}%
2207 \ifthenelse{ \value{COOL@multideriv}=1 }{%
2208 {\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}%
2209 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2210 {%
2211 \edef\COOL@temp@D@bot%
2212 {\COOL@temp@D@bot #4 \COOL@wrt@temp}%
2213 }%
2214 % Else
2215 {%
2216 \edef\COOL@temp@D@bot%
2217 {\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
2218 }%

```

```

2219 }%
2220 }%
2221 % Else
2222 {%
2223 \ifthenelse{\value{COOL@ct@}=1}%
2224 {%
2225 \frac{#4 #2}{\COOL@temp@D@bot}%
2226 }%
2227 % Else
2228 {%
2229 \frac{#4^{\arabic{COOL@ct@}} #2}{\COOL@temp@D@bot}%
2230 }%
2231 }%
2232 % Else
2233 {%
    Powers are not all Integers
2234 \edef\COOL@temp@D@bot{%
2235 \def\COOL@temp@D@top@power{%
2236 \forLoop{1}{\value{COOL@ct@}}{\COOL@multideriv}%
2237 {%
2238 \edef\COOL@power@temp%
2239 {\csname COOL@deriv@powers\roman{COOL@multideriv}\endcsname}%
2240 \edef\COOL@wrt@temp%
2241 {\csname COOL@deriv@wrt\roman{COOL@multideriv}\endcsname}%
2242 \ifthenelse{\value{COOL@multideriv} = 1}%
2243 {%
2244 \edef\COOL@temp@D@top@power{\COOL@power@temp}%
2245 }%
2246 % Else
2247 {%
2248 \edef\COOL@temp@D@top@power%
2249 {\COOL@temp@D@top@power + \COOL@power@temp}%
2250 \edef\COOL@temp@D@bot{\COOL@temp@D@bot \,%
2251 }%
2252 \ifthenelse{\equal{\COOL@power@temp}{1} }%
2253 {%
2254 \edef\COOL@temp@D@bot%
2255 {\COOL@temp@D@bot #4 \COOL@wrt@temp}%
2256 }%
2257 % Else
2258 {%
2259 \edef\COOL@temp@D@bot%
2260 {\COOL@temp@D@bot #4 \COOL@wrt@temp^{\COOL@power@temp}%
2261 }%
2262 }%
    we're done with the length of the  $\langle derivative\ power(s) \rangle$  argument, and we want to
    start at it + 1 to add the remainders
2263 \ifthenelse{\value{COOL@ct@}<\value{COOL@listlen}}%
2264 {%

```

```

2265 \edef\COOL@power@temp%
2266 {\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
2267 \stepcounter{COOL@ct}%
2268 \forLoop{\value{COOL@ct}}{\value{COOL@listlen}}{COOL@multideriv}%
2269 {%
2270 \edef\COOL@wrt@temp%
2271 {\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
2272 \ifthenelse{ \value{COOL@multideriv} = 1}%
2273 {%
2274 \edef\COOL@temp@D@top@power{\COOL@power@temp}%
2275 }%
2276 % Else
2277 {%
2278 \edef\COOL@temp@D@top@power%
2279 {\COOL@temp@D@top@power + \COOL@power@temp}%
2280 \edef\COOL@temp@D@bot{\COOL@temp@D@bot \,%
2281 }%
2282 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2283 {%
2284 \edef\COOL@temp@D@bot%
2285 {\COOL@temp@D@bot #4 \COOL@wrt@temp}%
2286 }%
2287 % Else
2288 {%
2289 \edef\COOL@temp@D@bot%
2290 {\COOL@temp@D@bot #4 \COOL@wrt@temp^{\COOL@power@temp}%
2291 }%
2292 }%
2293 }%
2294 % Else
2295 }%
2296 \frac{#4^{\COOL@temp@D@top@power} #2}{\COOL@temp@D@bot}%
2297 }%
2298 }%

2299 % Else If
2300 { \ifthenelse{ \equal{\COOL@notation@DShorten}{true} \AND
2301 \equal{\COOL@notation@DDisplayFunc}{outset} }%
2302 {%
2303 \ifthenelse{ \boolean{COOL@isint} }%
2304 {%
2305 \def\COOL@temp@D@bot{}%
2306 \setcounter{COOL@ct}{0}%
2307 \forLoop{1}{\value{COOL@ct}}{COOL@multideriv}%
2308 {%
2309 \edef\COOL@power@temp%
2310 {\csname COOL@deriv@powers@\roman{COOL@multideriv}\endcsname}%
2311 \edef\COOL@wrt@temp%
2312 {\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
2313 \addtocounter{COOL@ct}{\COOL@power@temp}%

```

```

2314 \ifthenelse{ \value{COOL@multideriv}=1 }{}%
2315 {\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}%
2316 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2317 {%
2318 \edef\COOL@temp@D@bot%
2319 {\COOL@temp@D@bot #4 \COOL@wrt@temp}%
2320 }%
2321 % Else
2322 {%
2323 \edef\COOL@temp@D@bot%
2324 {\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
2325 }%
2326 }%

```

we're done with the length of the $\langle derivative\ power(s) \rangle$ argument, and we want to start at it + 1 to add the remainders

```

2327 \ifthenelse{\value{COOL@ct}<\value{COOL@listlen}}%
2328 {%
2329 \edef\COOL@power@temp%
2330 {\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
2331 \stepcounter{COOL@ct}%
2332 \forLoop{\value{COOL@ct}}{\value{COOL@listlen}}{COOL@multideriv}%
2333 {%
2334 \edef\COOL@wrt@temp%
2335 {\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
2336 \addtocounter{COOL@ct}{\COOL@power@temp}%
2337 \ifthenelse{ \value{COOL@multideriv}=1 }{}%
2338 {\edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}%
2339 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2340 {%
2341 \edef\COOL@temp@D@bot%
2342 {\COOL@temp@D@bot #4 \COOL@wrt@temp}%
2343 }%
2344 % Else
2345 {%
2346 \edef\COOL@temp@D@bot%
2347 {\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
2348 }%
2349 }%
2350 }%
2351 % Else
2352 {}%
2353 \ifthenelse{\value{COOL@ct}=1}%
2354 {%
2355 \frac{#4}{\COOL@temp@D@bot} #2%
2356 }%
2357 % Else
2358 {%
2359 \frac{#4^{\arabic{COOL@ct}}}{\COOL@temp@D@bot} #2%
2360 }%

```

```

2361 }%
2362 % Else
2363 {%
    Powers are not all Integers
2364 \edef\COOL@temp@D@bot{}%
2365 \def\COOL@temp@D@top@power{}%
2366 \forLoop{1}{\value{COOL@ct}}{\COOL@multideriv}%
2367 {%
2368 \edef\COOL@power@temp%
2369 {\csname COOL@deriv@powers@\roman{\COOL@multideriv}\endcsname}%
2370 \edef\COOL@wrt@temp%
2371 {\csname COOL@deriv@wrt@\roman{\COOL@multideriv}\endcsname}%
2372 \ifthenelse{ \value{COOL@multideriv} = 1}%
2373 {%
2374 \edef\COOL@temp@D@top@power{\COOL@power@temp}%
2375 }%
2376 % Else
2377 {%
2378 \edef\COOL@temp@D@top@power%
2379 {\COOL@temp@D@top@power + \COOL@power@temp}%
2380 \edef\COOL@temp@D@bot{\COOL@temp@D@bot \,%
2381 }%
2382 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2383 {%
2384 \edef\COOL@temp@D@bot%
2385 {\COOL@temp@D@bot #4 \COOL@wrt@temp}%
2386 }%
2387 % Else
2388 {%
2389 \edef\COOL@temp@D@bot%
2390 {\COOL@temp@D@bot #4 \COOL@wrt@temp^\COOL@power@temp}%
2391 }%
2392 }%

    we're done with the length of the  $\langle derivative\ power(s) \rangle$  argument, and we want to
    start at it + 1 to add the remainders
2393 \ifthenelse{\value{COOL@ct}<\value{COOL@listlen}}%
2394 {%
2395 \edef\COOL@power@temp%
2396 {\csname COOL@deriv@powers@\roman{\COOL@ct}\endcsname}%
2397 \stepcounter{COOL@ct}%
2398 \forLoop{\value{COOL@ct}}{\value{COOL@listlen}}{\COOL@multideriv}%
2399 {%
2400 \edef\COOL@wrt@temp%
2401 {\csname COOL@deriv@wrt@\roman{\COOL@multideriv}\endcsname}%
2402 \ifthenelse{ \value{COOL@multideriv} = 1}%
2403 {%
2404 \edef\COOL@temp@D@top@power{\COOL@power@temp}%
2405 }%
2406 % Else

```

```

2407 {%
2408 \edef\COOL@temp@D@top@power%
2409 {\COOL@temp@D@top@power + \COOL@power@temp}%
2410 \edef\COOL@temp@D@bot{\COOL@temp@D@bot \,}%
2411 }%
2412 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2413 {%
2414 \edef\COOL@temp@D@bot%
2415 {\COOL@temp@D@bot #4 \COOL@wrt@temp}%
2416 }%
2417 % Else
2418 {%
2419 \edef\COOL@temp@D@bot%
2420 {\COOL@temp@D@bot #4 \COOL@wrt@temp~\COOL@power@temp}%
2421 }%
2422 }%
2423 }%
2424 % Else
2425 {%
2426 \frac{#4^{\COOL@temp@D@top@power} }{\COOL@temp@D@bot} #2%
2427 }%
2428 }%

2429 % Else If
2430 { \ifthenelse{ \equal{\COOL@notation@DShorten}{false} \AND
2431 \equal{\COOL@notation@DDisplayFunc}{inset} }%
2432 {%
2433 \def\COOL@temp@D@result{}%
2434 \def\COOL@temp@D@bot{}%
2435 \def\COOL@temp@D@top{}%
2436 \setcounter{COOL@ct@}{\value{COOL@ct}}%
2437 \addtocounter{COOL@ct@}{-1}
2438 \forLoop{1}{\value{COOL@ct@}}{\COOL@multideriv}%
2439 {%
2440 \edef\COOL@power@temp%
2441 {\csname COOL@deriv@powers@\roman{COOL@multideriv}\endcsname}%
2442 \edef\COOL@wrt@temp%
2443 {\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
2444 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2445 {%
2446 \edef\COOL@temp@D@top{#4}%
2447 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
2448 }%
2449 % Else
2450 {%
2451 \edef\COOL@temp@D@top{#4^{\COOL@power@temp}%
2452 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp~\COOL@power@temp}%
2453 }%
2454 \edef\COOL@temp@D@result%
2455 {\COOL@temp@D@result \frac{\COOL@temp@D@top}{\COOL@temp@D@bot}}%

```

```

2456 }%

    we're done with the length of the  $\langle derivative\ power(s) \rangle$  argument, and we want to
    start at it + 1 to add the remainders
2457 \ifthenelse{\value{COOL@ct}<\value{COOL@listlen}}%
2458 {%
    Must pick up the one for \value{COOL@ct}
2459 \edef\COOL@power@temp%
2460 {\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
2461 \edef\COOL@wrt@temp%
2462 {\csname COOL@deriv@wrt@\roman{COOL@ct}\endcsname}%
2463 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2464 {%
2465 \edef\COOL@temp@D@top{#4}%
2466 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
2467 }%
2468 % Else
2469 {%
2470 \edef\COOL@temp@D@top{#4^\COOL@power@temp}%
2471 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
2472 }%
2473 \edef\COOL@temp@D@result%
2474 {\COOL@temp@D@result \frac{\COOL@temp@D@top}{\COOL@temp@D@bot}}%

    Now add the ones beyond
2475 \stepcounter{COOL@ct}%
2476 \setcounter{COOL@ct}{\value{COOL@listlen}}%
2477 \addtocounter{COOL@ct}{-1}%
2478 \forLoop{\value{COOL@ct}}{\value{COOL@ct}}{\COOL@multideriv}%
2479 {%
2480 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2481 {%
2482 \edef\COOL@temp@D@top{#4}%
2483 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
2484 }%
2485 % Else
2486 {%
2487 \edef\COOL@temp@D@top{#4^\COOL@power@temp}%
2488 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
2489 }%
2490 \edef\COOL@temp@D@result%
2491 {\COOL@temp@D@result \frac{\COOL@temp@D@top}{\COOL@temp@D@bot}}%
2492 }%

    Must pick up the one for \value{COOL@listlen}
2493 \edef\COOL@wrt@temp%
2494 {\csname COOL@deriv@wrt@\roman{COOL@listlen}\endcsname}%
2495 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2496 {%
2497 \edef\COOL@temp@D@top{#4}%

```

```

2498 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
2499 }%
2500 % Else
2501 {%
2502 \edef\COOL@temp@D@top{#4^\COOL@power@temp}%
2503 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
2504 }%
2505 \edef\COOL@temp@D@result%
2506 {\COOL@temp@D@result \frac{\COOL@temp@D@top #2}{\COOL@temp@D@bot}}%
2507 }%
2508 % Else
2509 {%
    Must pick up the one for \value{COOL@ct}
2510 \edef\COOL@power@temp%
2511 {\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
2512 \edef\COOL@wrt@temp%
2513 {\csname COOL@deriv@wrt@\roman{COOL@ct}\endcsname}%
2514 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2515 {%
2516 \edef\COOL@temp@D@top{#4}%
2517 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
2518 }%
2519 % Else
2520 {%
2521 \edef\COOL@temp@D@top{#4^\COOL@power@temp}%
2522 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
2523 }%
2524 \edef\COOL@temp@D@result%
2525 {\COOL@temp@D@result \frac{\COOL@temp@D@top #2}{\COOL@temp@D@bot}}%
2526 }%
2527 \COOL@temp@D@result%
2528 }%

2529 % Else If
2530 { \ifthenelse{ \equal{\COOL@notation@DShorten}{false} \AND
2531 \equal{\COOL@notation@DDisplayFunc}{outset} }%
2532 {%
2533 \def\COOL@temp@D@result{}%
2534 \def\COOL@temp@D@bot{}%
2535 \def\COOL@temp@D@top{}%
2536 \forLoop{1}{\value{COOL@ct}}{\COOL@multideriv}%
2537 {%
2538 \edef\COOL@power@temp%
2539 {\csname COOL@deriv@powers@\roman{COOL@multideriv}\endcsname}%
2540 \edef\COOL@wrt@temp%
2541 {\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
2542 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2543 {%
2544 \edef\COOL@temp@D@top{#4}%
2545 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%

```

```

2546 }%
2547 % Else
2548 {%
2549 \edef\COOL@temp@D@top{#4^\COOL@power@temp}%
2550 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
2551 }%
2552 \edef\COOL@temp@D@result%
2553 {\COOL@temp@D@result \frac{\COOL@temp@D@top}{\COOL@temp@D@bot}}%
2554 }%

```

we're done with the length of the $\langle derivative\ power(s) \rangle$ argument, and we want to start at it + 1 to add the remainders

```

2555 \ifthenelse{\value{COOL@ct}<\value{COOL@listlen}}%
2556 {%
2557 \edef\COOL@power@temp%
2558 {\csname COOL@deriv@powers@\roman{COOL@ct}\endcsname}%
2559 \stepcounter{COOL@ct}%
2560 \forLoop{\value{COOL@ct}}{\value{COOL@listlen}}{\COOL@multideriv}%
2561 {%
2562 \edef\COOL@wrt@temp%
2563 {\csname COOL@deriv@wrt@\roman{COOL@multideriv}\endcsname}%
2564 \ifthenelse{ \equal{\COOL@power@temp}{1} }%
2565 {%
2566 \edef\COOL@temp@D@top{#4}%
2567 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp}%
2568 }%
2569 % Else
2570 {%
2571 \edef\COOL@temp@D@top{#4^\COOL@power@temp}%
2572 \edef\COOL@temp@D@bot{#4 \COOL@wrt@temp^\COOL@power@temp}%
2573 }%
2574 \edef\COOL@temp@D@result%
2575 {\COOL@temp@D@result \frac{\COOL@temp@D@top}{\COOL@temp@D@bot}}%
2576 }%
2577 }%
2578 % Else
2579 {%
2580 }%
2581 \COOL@temp@D@result #2
2582 }%

2583 % Else
2584 {%
2585 \PackageError{cool}{Invalid Option Sent}%
2586 {DShorten can only be 'true' or 'false';%
2587 DDisplayFunc can only be 'inset' or 'outset'}}%
2588 }%
2589 }}}}
2590 }

```

\D Derivatives

\pderiv

| | |
|--|---|
| $\backslash D\{f\}\{x\}$ | $\frac{d}{dx} f$ |
| $\backslash D[n]\{f\}\{x\}$ | $\frac{d^n}{dx^n} f$ |
| $\backslash D\{f\}\{x,y,z\}$ | $\frac{d}{dx} \frac{d}{dy} \frac{d}{dz} f$ |
| $\backslash D[1,2,1]\{f\}\{x,y,z\}$ | $\frac{d}{dx} \frac{d^2}{dy^2} \frac{d}{dz} f$ |
| $\backslash pderiv\{f\}\{x\}$ | $\frac{\partial}{\partial x} f$ |
| $\backslash pderiv[n]\{f\}\{x\}$ | $\frac{\partial^n}{\partial x^n} f$ |
| $\backslash pderiv\{f\}\{x,y,z\}$ | $\frac{\partial}{\partial x} \frac{\partial}{\partial y} \frac{\partial}{\partial z} f$ |
| $\backslash pderiv[1,2,1]\{f\}\{x,y,z\}$ | $\frac{\partial}{\partial x} \frac{\partial^2}{\partial y^2} \frac{\partial}{\partial z} f$ |

2591 $\backslash newcommand{\backslash D}[3][1]{\backslash COOL@derivative\{#1\}\{#2\}\{#3\}\{d\}}$

2592 $\backslash newcommand{\backslash pderiv}[3][1]{\backslash COOL@derivative\{#1\}\{#2\}\{#3\}\{\backslash partial\}}$

$\backslash Integrate$ Integrate

$\backslash Int$ This has the option `IntegrateDisplayFunc` which can be inset or outset:

$\backslash Style\{IntegrateDisplayFunc=inset\}$ (Default)

$$\backslash Integrate\{f\}\{x\} \quad \int f \, dx$$

$$\backslash Int\{f\}\{x\} \quad \int f \, dx$$

$$\backslash Integrate\{f\}\{x,A\} \quad \int_A f \, dx$$

$$\backslash Int\{f\}\{x,A\} \quad \int_A f \, dx$$

$$\backslash Integrate\{f\}\{x,a,b\} \quad \int_a^b f \, dx$$

$$\backslash Int\{f\}\{x,a,b\} \quad \int_a^b f \, dx$$

$\backslash Style\{IntegrateDisplayFunc=outset\}$

$$\backslash Integrate\{f\}\{x\} \quad \int dx \, f$$

$$\backslash Int\{f\}\{x\} \quad \int dx \, f$$

$$\backslash Integrate\{f\}\{x,A\} \quad \int_A dx \, f$$

$$\backslash Int\{f\}\{x,A\} \quad \int_A dx \, f$$

$$\backslash Integrate\{f\}\{x,a,b\} \quad \int_a^b dx \, f$$

$$\backslash Int\{f\}\{x,a,b\} \quad \int_a^b dx \, f$$

2593 $\backslash newcommand{\backslash COOL@notation@IntegrateDisplayFunc}\{inset\}$

2594 $\backslash newcommand{\backslash Integrate}[2]\{%$

2595 $\backslash listval\{#2\}\{0\}\%$

record the length of the list

2596 $\backslash setcounter\{COOL@listlen\}\{value\{COOL@listpointer\}\}\%$

2597 $\backslash ifthenelse\{value\{COOL@listlen\} = 1\}\{%$

2598 $\{%$

2599 $\backslash ifthenelse\{equal\{\backslash COOL@notation@IntegrateDisplayFunc\}\{outset\}\}\%$

2600 $\{%$

```

2601 \int \! d#2 \, #1%
2602 }%
2603 % ElseIf
2604 { \ifthenelse{\equal{\COOL@notation@IntegrateDisplayFunc}{inset}}}%
2605 {%
2606 \int #1 \, d#2%
2607 }%
2608 % Else
2609 {%
2610 \PackageError{cool}{Invalid Option Sent}%
2611 {'DisplayFunc' can only be 'inset' or 'outset'}%
2612 }}%
2613 }%
2614 % ElseIf
2615 { \ifthenelse{ \value{COOL@listlen} = 2 }%
2616 {%
2617 \ifthenelse{\equal{\COOL@notation@IntegrateDisplayFunc}{outset}}}%
2618 {%
2619 \int_{\listval{#2}{2}} \! d{\listval{#2}{1}} \, #1%
2620 }%
2621 % ElseIf
2622 { \ifthenelse{\equal{\COOL@notation@IntegrateDisplayFunc}{inset}}}%
2623 {%
2624 \int_{\listval{#2}{2}} #1 \, d{\listval{#2}{1}}%
2625 }%
2626 % Else
2627 {%
2628 \PackageError{cool}{Invalid Option Sent}%
2629 {'DisplayFunc' can only be 'inset' or 'outset'}%
2630 }}%
2631 }%
2632 % ElseIf
2633 { \ifthenelse{ \value{COOL@listlen} = 3 }%
2634 {%
2635 \ifthenelse{\equal{\COOL@notation@IntegrateDisplayFunc}{outset}}}%
2636 {%
2637 \int_{\listval{#2}{2}}^{\listval{#2}{3}} \! d{\listval{#2}{1}} \, #1%
2638 }%
2639 % ElseIf
2640 { \ifthenelse{\equal{\COOL@notation@IntegrateDisplayFunc}{inset}}}%
2641 {%
2642 \int_{\listval{#2}{2}}^{\listval{#2}{3}} #1 \, d{\listval{#2}{1}}%
2643 }%
2644 % Else
2645 {%
2646 \PackageError{cool}{Invalid Option Sent}%
2647 {'DisplayFunc' can only be 'inset' or 'outset'}%
2648 }}%
2649 }%
2650 % Else

```

```

2651 {%
2652 \PackageError{cool}{‘Integrate’ has invalid parameter list}%
2653   {this happens when the second argument has more than two commas}%
2654 }}}}
2655 }%
2656 \newcommand{\Int}[2]{\Integrate{#1}{#2}}

\Sum Sum
      \Sum{a_n}{n}           $\sum_n a_n$ 
      \Sum{a_n}{n,1,N}       $\sum_{n=1}^N a_n$ 
2657 \newcommand{\Sum}[2]{%
2658 \listval{#2}{0}%
      record the length of the list
2659 \setcounter{COOL@listlen}{\value{COOL@listpointer}}
2660 \ifthenelse{ \value{COOL@listlen} = 1 }{%
2661 {%
2662 \sum_{#2} #1%
2663 }%
2664 % else
2665 {%
2666 \ifthenelse{ \value{COOL@listlen} = 3 }{%
2667 {%
2668 \sum_{ \listval{#2}{1} = \listval{#2}{2} }^{\listval{#2}{3}} #1
2669 }%
2670 % else
2671 {%
2672 \PackageError{cool}{Invalid list length for ‘Sum’}%
2673 {can only have none or two commas for second argument}%
2674 }%
2675 }%
2676 }

\Prod Product
      \Prod{a_n}{n}           $\prod_n a_n$ 
      \Prod{a_n}{n,1,N}       $\prod_{n=1}^N a_n$ 
2677 \newcommand{\Prod}[2]{%
2678 \listval{#2}{0}%
      record the length of the list
2679 \setcounter{COOL@listlen}{\value{COOL@listpointer}}
2680 \ifthenelse{ \value{COOL@listlen} = 1 }{%
2681 {%
2682 \prod_{#2} #1%
2683 }%
2684 % else
2685 {%
2686 \ifthenelse{ \value{COOL@listlen} = 3 }{%
2687 {%

```

```

2688 \prod_{\listval{#2}{1}} = \listval{#2}{2}}^{\listval{#2}{3}} } #1
2689 }%
2690 % else
2691 {%
2692 \PackageError{cool}{Invalid list length for ‘Prod’}%
2693 {can only have none or two commas for second argument}%
2694 }%
2695 }%
2696 }

```

1.3.43 Vector Operators

\DotProduct The dot product, $\text{\DotProduct}\{\text{\vec{A}}\}\{\text{\vec{B}}\}$, $\vec{A} \cdot \vec{B}$
2697 `\newcommand{\DotProduct}[2]{#1 \cdot #2}`

\Cross The cross product, $\text{\Cross}\{\text{\vec{A}}\}\{\text{\vec{B}}\}$, $\vec{A} \times \vec{B}$
2698 `\newcommand{\Cross}[2]{#1 \times #2}`

\Div the divergence, $\text{\Div}\{\text{\vec{A}}\}$, $\nabla \cdot \vec{A}$
2699 `\newcommand{\Div}[1]{\nabla \cdot #1}`

\Grad The gradient, $\text{\Grad}\{f\}$, ∇f
2700 `\newcommand{\Grad}[1]{\nabla #1}`

\Curl The curl, $\text{\Curl}\{\text{\vec{A}}\}$, $\nabla \times \vec{A}$
2701 `\newcommand{\Curl}[1]{\nabla \times #1}`

\Laplacian The laplacian, $\text{\Laplacian}\{f\}$, $\nabla^2 f$
2702 `\newcommand{\Laplacian}[1]{\nabla^2 #1}`

1.3.44 Matrix Operations

\Transpose Transpose of a matrix, $\text{\Transpose}\{A\}$, A^T
2703 `\newcommand{\COOL@notation@TransposeParen}{inv}`
2704 `\newcommand{\Transpose}[1]{\COOL@decide@paren{Transpose}{#1}^T }`

\Dagger Conjugate Transpose of a matrix, $\text{\Dagger}\{A\}$, A^\dagger
2705 `\newcommand{\COOL@notation@DaggerParen}{inv}`
2706 `\newcommand{\Dagger}[1]{\COOL@decide@paren{Dagger}{#1}^\dagger }`

\Det determinant of a matrix
`\Style{DetDisplay=det}` (Default)
 $\text{\Det}\{A\}$ $\det A$
`\Style{DetDisplay=barenc}`
 $\text{\Det}\{A\}$ $|A|$
2707 `\newcommand{\COOL@notation@DetParen}{none}`
2708 `\newcommand{\COOL@notation@DetDisplay}{det}`
2709 `\newcommand{\Det}[1]{%`

```

2710 \ifthenelse{\equal{\COOL@notation@DetDisplay}{det}}{%
2711 {%
2712 \det\COOL@decide@paren{Det}{#1}%
2713 }%
2714 % ElseIf
2715 { \ifthenelse{\equal{\COOL@notation@DetDisplay}{barenc}}{%
2716 {%
2717 \left|#1\right|%
2718 }%
2719 % Else
2720 {%
2721 \PackageError{cool}{Invalid Option Sent}%
2722 {'DetDisplay' can only be 'det' or 'barenc'}%
2723 }%
2724 }

```

`\Tr` Trace of a Matrix, `\Tr{A}`, `Tr A`

```

2725 \newcommand{\COOL@notation@TrParen}{none}
2726 \newcommand{\Tr}[2][]{%
2727 \ifthenelse{\equal{#1}{}}{
2728 {%
2729 \operatorname{Tr}\COOL@decide@paren{Tr}{#2}%
2730 }%
2731 % Else
2732 {%
2733 \operatorname{Tr}_{#1}\COOL@decide@paren{Tr}{#2}%
2734 }%
2735 }

```

1.3.45 Matricies

`\IdentityMatrix` The Identity Matrix

$$\begin{array}{l} \text{\code{\IdentityMatrix}} \quad \mathbb{I} \\ \text{\code{\IdentityMatrix[2]}} \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}$$

```

2736 \newcommand{\COOL@notation@IdentityMatrixParen}{p}
2737 \newcounter{COOL@row}%
2738 \newcounter{COOL@col}%
2739 \newcommand{\IdentityMatrix}[1][0]{%
2740 \isint{#1}%
2741 \ifthenelse{\boolean{COOL@isint}}{%
2742 {%
2743 \ifthenelse{#1=0}{
2744 {%
2745 \mathbb{I}%
2746 }%
2747 % Else
2748 {%
2749 \setcounter{COOL@ct}{\value{MaxMatrixCols}}%

```

```

2750 \setcounter{MaxMatrixCols}{#1}%
2751 \ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{p}}%
2752 {%
2753 \begin{pmatrix}%
2754 }%
2755 % ElseIf
2756 { \ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{b}}%
2757 {%
2758 \begin{bmatrix}%
2759 }%
2760 % ElseIf
2761 { \ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{br}}%
2762 {%
2763 \begin{Bmatrix}%
2764 }%
2765 % Else
2766 {%
2767 \begin{matrix}%
2768 }}%
2769 \forLoop{1}{#1}{\COOL@row}%
2770 {%
2771 \ifthenelse{\NOT \value{\COOL@row} = 1}{\{\}}%
2772 \forLoop{1}{#1}{\COOL@col}%
2773 {%
2774 \ifthenelse{\NOT \value{\COOL@col} = 1}{&}{}%
2775 \ifthenelse{\value{\COOL@row}=\value{\COOL@col}}{1}{0}%
2776 }%
2777 }%
2778 \ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{p}}%
2779 {%
2780 \end{pmatrix}%
2781 }%
2782 % ElseIf
2783 { \ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{b}}%
2784 {%
2785 \end{bmatrix}%
2786 }%
2787 % ElseIf
2788 { \ifthenelse{\equal{\COOL@notation@IdentityMatrixParen}{br}}%
2789 {%
2790 \end{Bmatrix}%
2791 }%
2792 % Else
2793 {%
2794 \end{matrix}%
2795 }}%
2796 \setcounter{MaxMatrixCols}{\value{\COOL@ct}}%
2797 }%
2798 }%
2799 % Else

```

2800 {%
2801 \mathbb{I}%
2802 }%
2803 }%

Change History

| | | |
|--|--|---|
| v0 | | age to allow storing of the list length 1 |
| General: pre-Initial version [tentative edition] 1 | | |
| v1.0 | | v1.2 |
| General: Initial Release 1 | | General: Split off the list, string, and forloop parts to separate packages 1 |
| v1.1 | | |
| General: Added listlenstore to pack- | | |

Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in *roman* refer to the code lines where the entry is used.

| Symbols | | |
|-------------|---------------------------------------|--|
| \{ | 7, 1682, 1696, 1719, 1738, 2121, 2133 | \ArithGeoMean 1902 |
| \} | 7, 1682, 1696, 1722, 1745, 2129, 2139 | \ArithGeoMeanSymb 1903, 1905 |
| | | \AssocLegendreP 659 |
| | | \AssocLegendreQ 660 |
| | | \AssocWeierstrassSigma 1656 |
| A | | |
| \Abs | 2038 | B |
| \AGM | 1902 | \Bernoulli 431, 680 |
| \AiryAi | 327 | \BernoulliB 680 |
| \AiryAiSymb | 328, 329 | \BesselI 319 |
| \AiryBi | 330 | \BesselJ 311 |
| \AiryBiSymb | 331, 332 | \BesselK 323 |
| \AppellFOne | 1189 | \BesselY 315 |
| \ArcCos | 218 | \Beta 820, 847, 848, 908 |
| \ArcCosh | 276 | \BetaReg 849 |
| \ArcCot | 258 | \BetaRegInv 876 |
| \ArcCoth | 284 | \BetaRegularized 849 |
| \ArcCsc | 254 | \Binomial 683 |
| \ArcCsch | 280 | C |
| \ArcSec | 256 | \CarmichaelLambda 2114 |
| \ArcSech | 282 | \Catalan 115 |
| \ArcSin | 200 | \Ceiling 338 |
| \ArcSinh | 274 | \ChebyshevT 645 |
| \ArcTan | 236 | \ChebyshevU 649 |
| \ArcTanh | 278 | \CInfty 128 |
| \Arg | 2039 | |

| | | | |
|----------------------------------|--|-------------------|-------------------------|
| \ClebschGordon | 1328 | \DiGamma | 797 |
| \ComplexInfinity | 128 | \DigitCount | 2117 |
| \Conj | 2040 | \DiLog | 2016 |
| \Conjugate | 2040 | \DiracDelta | 2142 |
| \COOL@arg@temp | 466, 473, 474, 478, 479, 484, 485, 488, 495, 501, 502, 506, 507, 510 | \DirectedInfinity | 126 |
| \COOL@Bernoulli@arg@i | 442 | \DirInfty | 126 |
| \COOL@Bernoulli@arg@ii | 443 | \DiscreteDelta | 458 |
| \COOL@decide@indicies | 68 | \Div | 2699 |
| \COOL@decide@indicies@placement | 69, 71, 79, 85 | \Divisors | 2096 |
| \COOL@decide@paren | 13 | \DivisorSigma | 2106 |
| \COOL@decide@paren@no@type | 13–15 | \DivisorsSymb | 2097, 2099 |
| \COOL@decide@paren@type | 17, 21, 23, 29, 35, 41, 47, 53 | \DotProduct | 2697 |
| \COOL@deriv@powers@i | 2160 | | |
| \COOL@derivative | 2151, 2591, 2592 | E | |
| \COOL@hideOnSF | 1700, 1719–1721, 1739, 1741, 1743 | \E | 111, 138, 142 |
| \COOL@Hypergeometric@pq@ab@value | 987 | \EExp | 1907 |
| \COOL@keyeater | 98, 101, 106 | \EGCD | 381 |
| \COOL@keyend | 98, 101, 106 | \EllipticE | 1475, 1544 |
| \COOL@keystop | 98, 100, 102 | \EllipticExp | 1907 |
| \COOL@MeijerG@anp@value | 1195, 1292, 1303, 1305, 1322 | \EllipticExpSymb | 1908, 1920 |
| \COOL@notation@ArcTrig | 199 | \EllipticF | 1526 |
| \COOL@notation@DDisplayFunc | 2148 | \EllipticK | 1471 |
| \COOL@notation@DShorten | 2148 | \EllipticLog | 1924 |
| \COOL@tempd | 2165, 2167 | \EllipticLogSymb | 1925, 1947 |
| \Cos | 189 | \EllipticNomeQ | 1896 |
| \Cosh | 262 | \EllipticNomeQInv | 1896 |
| \CoshInt | 984 | \EllipticPi | 1498, 1545 |
| \CoshIntSymb | 985, 986 | \EllipticTheta | 1552 |
| \CosInt | 978 | \ELog | 1924 |
| \CosIntSymb | 979, 980 | \Erf | 910, 951 |
| \Cot | 197 | \Erfc | 953 |
| \Coth | 272 | \ErfcInv | 957 |
| \Cross | 2698 | \ErfcSymb | 954, 955, 958 |
| \Csc | 193 | \ErFi | 959 |
| \Csch | 266 | \ErfInv | 931, 952 |
| \Curl | 2701 | \ErFiSymb | 960, 961 |
| \CyclotomicC | 675 | \ErfSymb | 911, 917, 922, 937, 942 |
| | | \Euler | 410, 679 |
| D | | \EulerE | 679 |
| \D | 2591 | \EulerGamma | 114 |
| \Dagger | 2705 | \EulerPhi | 2109 |
| \DblFactorial | 682 | \Exp | 130 |
| \DedekindEta | 1888 | \ExpIntE | 966 |
| \Det | 2707 | \ExpIntEi | 968 |
| | | \ExpIntEiSymb | 969, 971 |
| | | \ExtendedGCD | 381 |
| | | \ExtendedGCDSymb | 382, 384 |
| | | F | |
| | | \Factorial | 681 |

| | | | |
|--------------------------------|------------------|--|--|
| \JacobiSymbol | 2113 | | |
| \JacobiTheta | 1552 | | |
| \JacobiZeta | 1546 | | |
| K | | | |
| \Khinchin | 117 | | |
| \KleinInvariantJ | 1890 | | |
| \KroneckerDelta | 461 | | |
| L | | | |
| \LambertW | 286 | | |
| \Laplacian | 2702 | | |
| \LaugerreL | 517 | | |
| \LCM | 386 | | |
| \LCMSymb | 387, 388 | | |
| \LegendreP | 539, 659 | | |
| \LegendreQ | 592, 660 | | |
| \LerchPhi | 1983 | | |
| \LeviCivita | 490 | | |
| \Log | 135 | | |
| \LogGamma | 794 | | |
| \LogGammaSymb | 795, 796 | | |
| \LogInt | 972 | | |
| \LogIntSymb | 973, 974 | | |
| M | | | |
| \MathieuC | 2017 | | |
| \MathieuCharacteristicA | 2025 | | |
| \MathieuCharacteristicB | 2029 | | |
| \MathieuCharacteristicExponent | 2033 | | |
| \MathieuCharisticA | 2025 | | |
| \MathieuCharisticB | 2029 | | |
| \MathieuCharisticExp | 2033 | | |
| \MathieuCSymb | 2018, 2020 | | |
| \MathieuS | 2021 | | |
| \MathieuSSymb | 2022, 2024 | | |
| \Max | 307 | | |
| \MeijerG | 1233 | | |
| \Min | 309 | | |
| \Mod | 348 | | |
| \ModularLambda | 1893 | | |
| \MoebiusMu | 2111 | | |
| \Multinomial | 684 | | |
| N | | | |
| \NevilleThetaC | 1556 | | |
| \NevilleThetaD | 1561 | | |
| \NevilleThetaN | 1566 | | |
| \NevilleThetaS | 1571 | | |
| \NielsenPolyLog | 1985, 1997 | | |
| P | | | |
| \PartitionsP | 454 | | |
| \PartitionsQ | 456 | | |
| \pderiv | 2591 | | |
| \PI | 112 | | |
| \Pochhammer | 793 | | |
| \PolyGamma | 799 | | |
| \PolyLog | 1990, 2016 | | |
| \PolyLogSymb | 1991, 2004 | | |
| \Prime | 2100 | | |
| \PrimePi | 2104 | | |
| \PrimeSymb | 2101, 2103 | | |
| \Prod | 2677 | | |
| \ProductLog | 286, 287 | | |
| Q | | | |
| \Quotient | 375 | | |
| \QuotientSymb | 376, 378 | | |
| R | | | |
| \Real | 2065 | | |
| \RealSymb | 2066, 2070, 2074 | | |
| \RegHypergeometric | 1118 | | |
| \RegIncBeta | 849 | | |
| \RegIncBetaInv | 876 | | |
| \RegIncGamma | 747 | | |
| \RegIncGammaInv | 769 | | |
| \RiemannSiegelTheta | 1976 | | |
| \RiemannSiegelZ | 1979 | | |
| \RiemannZeta | 1952 | | |
| \Round | 339 | | |
| S | | | |
| \Sec | 195 | | |
| \Sech | 269 | | |
| \Sign | 2089 | | |
| \Signature | 490 | | |
| \Sin | 187 | | |
| \Sinh | 260 | | |
| \SinhInt | 981 | | |
| \SinhIntSymb | 982, 983 | | |
| \SinInt | 975 | | |
| \SinIntSymb | 976, 977 | | |
| \SixJSymbol | 1420 | | |
| \SpHarmY | 667 | | |
| \SphericalHarmonicY | 667 | | |
| \SphericalHarmY | 667 | | |
| \StieltjesGamma | 1982 | | |
| \StirlingSOne | 452 | | |
| \StirlingSTwo | 453 | | |

