The Short

`Hyperoperations` v0.1 Manual

Andy Groeneveld

July 18, 2012

Package copyright ©2012, Andy Groeneveld. Mathematica is a copyright of Wolfram Research.

1 Introduction

The `Hyperoperations` package has one purpose: allowing the use of hyperoperation sequences in Mathematica. A hyperoperation sequence consists of operations that are related by a recursion scheme. The familiar operations of addition, multiplication, and exponentiation are part of the main hyperoperation sequence. Of course, variations on this sequence exist, such as Albert Bennett's commutative hyperoperations, Clément Frappier's balanced hyperoperations, and Andrzej Grzegorczyk's functions used in defining the classes $\mathcal{E}^n$. Within each hyperoperation sequence, an individual operation is identified by its rank. The ranks of the familiar operations of algebra are integers, so that we have addition (rank 1), multiplication (rank 2), and exponentiation (rank 3). The inverses of a hyperoperation are the hyperlogarithm and hyper-root; when the hyperoperation is commutative, both inverses are the same. An example is addition, whose inverse is subtraction. Naturally, mathematicians have wondered what lower ranks, higher ranks, and even real or complex ranks correspond to. In some cases, they have been succesful, although there is still much room for progress in this area.

With that said, `Hyperoperations` is not a very sophisticated package; it is a small collection of hyperoperations and related functions. Hyperoperations and hyperlogarithms of rank 0 through 4 are fully supported; hyper-roots are fully supported through rank 3. For rank 4, the square super-root can be calculated exactly; other super-roots are approximated numerically when possible. Symbolic manipulation can be done on any rank.

Other hyperoperation sequences are also included:

- Bennett's commutative hyperoperations (including real ranks),
- Frappier's balanced hyperoperations,
- Goodstein's G function (closely related to the main sequence), and
- Grzegorczyk's f_k, used in defining the hierarchy that bears his name.

If you find anything that needs correcting, from a typo to a bug to a serious mathematical blunder, feel free to tell me by emailing `abgroene@mtu.edu`.

2 Using `Hyperoperations`

Once you have downloaded the zip file `Hyperoperations.zip`, all you need to do to begin using the package is:

1. Unzip `Hyperoperations.zip`. On Windows, this can be accomplished by right-clicking on the file and selecting "Extract all..."

2. Place the files `Hyperoperations.m` and `s25` in your Mathematica working directory. You can find out what your working directory is with `Directory[]`, or set a new one with `SetDirectory[`*new path*`]`

3. Load the package with the command

   ```
   Needs["Hyperoperations`"]
   ```

   It is also possible to use `Get` or its shorthand form `<<` to load the package. You may encounter a "shadow" error if you have already used or defined symbols with the same names as symbols in the package. If you do, run `Remove[`*symbol*`]` for each name that appears with an error message.

4. For brief help on a function, type `?`*function*. For a list of all functions in the `Hyperoperations` package, type `?Hyperoperations`*`. For a list of all functions with more substantial documentation, you can see the next section.

3 Functions

`Bennett[`n`, `a`, `b`]`	gives the commutative hyperoperation $a\,{}_n^0\,b$. n need not be an integer. For $n = 0, 1$, the commutative hyperoperations correspond to the main sequence: $a\,{}_0^0\,b = a+b$ and $a\,{}_1^0\,b = ab$, but $a\,{}_2^0\,b \neq a^b$.
`Delta[`x`]`	represents the delta number Δx (not to be confused with a differential).
`Deltate[`x`, `y`]`	gives the deltation $x \,\Delta\, y$. Deltation is the inverse of (Rubtsov and Romerio's) zeration, so if $z = x \circ y$, then $y = z \,\Delta\, x$.
`Epsilon[`s`, `a`]`	gives the left identity element $\epsilon_s(a)$, where s is the hyperoperation rank and a is the right operand.
`Eta[`s`]`	gives the right identity element η_s, where s is the hyperoperation rank. Note that the right identity element does not depend on the other operand.
`FrappierBox[`x`, `y`]`	gives Clément Frappier's box operation $x \,\square\, y$.
`GoodsteinG[`k`, `a`, `n`]`	gives R. L. Goodstein's function $G(k, a, n)$, which is equal to $a \boxed{k} n$ for integers k and n both greater than zero.

`HierarchyF[`n`, `x`, `y`]` gives Andrzej Grzegorczyk's function $f_n(x, y)$. f_n arises in the definition of the Grzegorczyk hierarchy, which contains *every* computable function at some level. The name `HierarchyF` was chosen since few people can spell "Grzegorczyk" the same way twice.

`Hyper[`s`, `a`, `b`]` gives the general hyperoperation $a\boxed{s}b$, also written as $a{\otimes}^s b$ or $a[s]b$. For integer ranks $0 \le s \le 4$, the function evaluates numerically for all numeric arguments. Note that for $k \ge 4$, extremely large values can be produced easily (see `TetrateDigits`).

`HyperLog[`s`, `a`, `b`]` gives the base-b hyperlogarithm of a for rank s, usually denoted as ${}^{b}\!\underset{s}{\underline{|a}}$.

`HyperRoot[`s`, `a`, `b`]` gives the b-th hyper-root of a for rank s, handily denoted by ${}_{b}\!\overset{s}{|a}$.

`SuperLog[`x`, `y`]` gives the base-x superlogarithm of y. Note that this is the reverse order from `HyperLog`; `SuperLog[`x`, `y`]` is the same as `HyperLog[4, `y`, `x`]`.

`SuperRoot[`x`, `y`]` gives the y-th super-root of x. For $y = 2$, the result is exact. For $y > 2$, the root is calculated numerically. For $y < 2$, the process takes too long to be practical. Be advised that non-integer roots are also time-consuming, but still possible. Commands like `SuperRoot[2, 1.5]` will simply evaluate to `SuperRoot[2, 1.5]` since computation is impractical. However, the package is smart enough to handle these "symbolic" super-roots properly.

`SuperSqrt[`x`]` gives the square super-root of x, provided that x is real and $x > e^{-1/e}$.

`Tetrate[`x`, `y`]` gives the tetration ${}^{y}x$, provided $x > 1$. For those curious, the extension method used is that of Andrew Robbins. Be aware that even `Tetrate[4, 4]` causes an overflow. However, the next command provides a way to approach these large numbers.

`TetrateDigits[`x`, `y`, `n`]` gives the n rightmost digits of ${}^{y}x$. Using this, one can compute, say, the last 100,000 digits of ${}^{4}4$. For non-integer values of x, Mathematica uses machine-precision numbers, so this limits the number of digits that can be obtained, typically to 7 or less.

`Tower[`x`, `n`]` gives a power tower of n copies of x. For integer n, this is the same as `Tetrate[`x`, `n`]`.

`Tower[`x`, `n`, `s`]` gives a power tower of n copies of x with s at the top. Those who feel so inclined could also use `Tetrate[`x`, `n` + SuperLog[`x`, `s`]]`, which may *not* be the same since `Tetrate` and `SuperLog` use approximations!

`TowerMod[`x`, `n`, `m`]`	is the same as `Tower[`x`, `n`]`, but evaluates using exponentiation modulo m.
`TowerMod[`x`, `n`, `s`, `m`]`	is the same as `Tower[`x`, `n`, `s`]`, but evaluates using exponentiation modulo m.
`TrappmannZerate[`x`, `y`]`	gives the zeration $x \circ y$ according to Henryk Trappmann's counterproposal.
`Zerate[`x`, `y`]`	gives the zeration $x \circ y$ according to Rubtsov and Romerio's definition.

4 Examples

As shown here, `Bennett` takes non-integer ranks, although whether these non-integer ranks have any meaning is another question.

```
In[1]:= Bennett[0.5, 3, 4]
Out[1]= 9.692
```

Since `Bennett` performs commutative hyperoperations, its version of exponentiation is not what one might expect:

```
In[2]:= Bennett[2, 0, 0]
Out[2]= Infinity

In[3]:= N[Bennett[2, 3, 4]]
Out[3]= 4.58596
```

The following examples illustrate the unusual results `Deltate` can give.

```
In[4]:= Zerate[10, 5]
Out[4]= 11

In[5]:= Deltate[11, 10]
Out[5]= Interval[{-Infinity, 10}]

In[6]:= Deltate[11, 5]
Out[6]= 10
```

To get a better idea of what left identity elements are and how they work, see these examples:

```
In[7]:= Epsilon[0, 77]
Out[7]= -Infinity
In[8]:= Hyper[0, %, 77]
Out[8]= 77

In[9]:= Epsilon[1, 77]
Out[9]= 0
In[10]:= Hyper[1, %, 77]
Out[10]= 77
```

```
In[11]:= Epsilon[3, 77]
Out[11]= 77^(1/77)
In[12]:= Hyper[3, %, 77]
Out[12]= 77
```

Frappier's box operations have a simple structure of repeated exponentiation:

```
In[13]:= Clear[x]
In[14]:= Table[FrappierBox[n, x], {n, 0, 3}]
Out[14]= {x, x^x, (x^x)^x^x, ((x^x)^x^x)^(x^x)^x^x}
```

Tetration is famous for producing large numbers, but the function `TetrateDigits` provides a way to see the rightmost digits of a tetration that would otherwise be too large to handle:

```
In[15]:= Tetrate[4, 4]
General::ovfl: Overflow occurred in computation. >>
Out[15]= Overflow[]
In[16]:= TetrateDigits[4, 4, 50]
Out[16]= 27238232605843019607448189676936860456095261392896
```

Tetration and its two inverses also work for real numbers:

```
In[17]:= Tetrate[E, Pi]
Out[17]= 3.71529×10^10

In[18]:= SuperLog[3, 19]
Out[18]= 1.90824
In[19]:= Tetrate[3, %]
Out[19]= 19.

In[20]:= SuperRoot[44, 3]
Out[20]= 2.12993
In[21]:= Tetrate[%, 3]
Out[21]= 44.
```

If you can understand how these next expressions for the golden ratio ϕ and its square ϕ^2 work, you have taken a great stride in understanding hyperoperations. You'll certainly be able to get use out of this package.

```
In[22]:= Hyper[1, Hyper[2, Hyper[3, 5, 1/2], 1/2], 1/2]
Out[22]= 1/2+Sqrt[5]/2
In[23]:= % == GoldenRatio
Out[23]= True

In[24]:= Hyper[0, Hyper[1, Hyper[2, Hyper[3, 5, 1/2], 1/2], 1/2], 1/2]
Out[24]= 3/2+Sqrt[5]/2
In[25]:= % == GoldenRatio^2
Out[25]= True
```

5 Creating the Critical Function

There is nothing magical about the file `s25`. It contains the critical function for the superlogarithm, `s25`. I generated it using this short snippet:

```
n = 25;
temp = LinearSolve[Table[Table[m^k/m! - If[m == k, Log[x]^(-k), 0], {m, 1, n}],
  {k, 0, n - 1}], Join[{1}, Table[0, {n - 1}]]];
body = -1 + Sum[z^k/k! temp[[k]], {k, 1, n}];
s25[x_, z_] := Evaluate[Simplify[body]]
```

It isn't necessary to use `Simplify`; I only did this to reduce the size of the function for distribution. For personal use, the extra time simplifcation takes is probably not worth it. You can use this code, with slight modification, to produce any approximation you desire. Just change the value of `n` (and the function name!), but beware that you may need to wait awhile for values over 20. To use your own critical function in the package, write it to a file (using `Put` or `>>`, for example) and then change the line in `Hyperoperations.m` that reads:

```
Get["s25"]
```

to `Get["`*your file*`"]`.

6 Legal Information

Copyright ©2012, Andy Groeneveld.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Michigan Technological University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.