Progress In NARS2000
October 2015 to September 2017

Bob Smith
Sudley Place Software
Originally Written
11 Jul 2017
Updated
10 Sep 2017

Released Features

Language Features

Hypercomplex Numbers

Complex, Quaternion, and Octonion numbers are now supported with their 2-, 4-, or 8-dimensional coefficients represented in one of four ways as:

- all fixed-precision 64-bit integers,
- all fixed-precision 64-bit floating point numbers,
- all multiple-precision integer/rational numbers, or
- all multiple-precision floating point numbers.

All primitive functions and operators are sensitive to Hypercomplex Numbers with the exception of the Shriek functions. These functions return results on Real, Complex, and Quaternion Fixed-Precision numbers and Real Multiple-Precision numbers only; otherwise, they signal a `DOMAIN ERROR` on all types of Octonions and a `NONCE ERROR` on Complex and Quaternion Multiple-Precision numbers.

Extending these functions to the remaining datatypes requires calculating the Complex-valued Eigenvalues and Eigenvectors of certain 2×2 or 4×4 Real non-symmetric matrices which in turn requires writing my own Multiple-Precision routines for calculating Eigenvalues and Eigenvectors.

Quaternion and Octonion numbers introduce their own set of challenges because multiplication is non-commutative (where L×R ←/→ R×L). This failed property affects the definition of not just multiplication but several others, and caused me to introduce a new system variable ⎕DQ (Division Quotient – Left or Right) to control how division is done. In particular, there are two ways to interpret L÷R. An additional challenge for Octonions is that multiplication is not associative.

For more details, see “Hypercomplex Numbers in APL”[12], "Hypercomplex GCD in APL[13]", "Hypercomplex Quotients in APL"[14]., "Hypercomplex Notation in APL"[15], and "Hypercomplex Implementation in APL"[16].

Implement Dilate and Condense

One way to manage the coefficients of Hypercomplex numbers uses the left and right carets as monadic functions. The **Condense** function (<R) takes an array whose number of columns is 1, 2, 4, or 8 and creates a new array whose shape is ¯1↓⍴R and whose last coordinate is filled with Hypercomplex numbers of the appropriate dimension. For example,

```
      <2 4⍴⍳8
1i2j3k4 5i6j7k8
```

The **Dilate** function (>R) reverses the effect of the Condense function and creates a new array whose shape is (⍴R),=R, where =R returns an integer scalar of the Hypercomplex dimension (1, 2, 4, or 8) of R:

```
      ><2 4⍴⍳8
1 2 3 4
5 6 7 8
```

This feature was suggested by David A. Rabenhorst.

Implement Preliminary Support For NaNs

A NaN (Not-a Number) is a concept found in the IEEE-754 Standard for Binary Floating Point Arithmetic, among other places. It is meant to represent a state where there is no number that can adequately represent the result of a calculation (e.g., `0×∞`, `1○∞`, etc.).It can also be used as a fill for a missing or unknown value. For the most part, a NaN as an argument to a primitive function/operator either returns another NaN or signals an error. The exceptions are `A=B` and `A≠B` where NaNs compare equally with themselves. The idea is that NaNs propagate throughout a calculation to the end so you know something was missing or unknown.

An interesting example of NaNs is `0*0J1`. You might be tempted to reason that because it's not `0*0`, the answer must be `0`. That's true for Real but not Complex exponents. One way to understand this is to use the folllowing identity which expresses the Power function using simpler functions:

```
L*R    ↔ *R  ×⍟L
0*0J1 ↔ *0J1×⍟0
```

If you don't support infinities, then `⍟0` is a `DOMAIN ERROR` and that's the answer. On the other hand, if your system says `⍟0 ↔ ¯∞`, we get

```
0*0J1 ↔ *0J1×¯∞
```

Applying Euler's formula `*0J1×N ↔ (2○N)+0J1×1○N`, involves calculating `2○¯∞` and `1○¯∞`, both of which are bona fide `DOMAIN`

ERRORs – if you don't suport NaNs, that's the answer. However, the IEEE-754 Standard calls both of those expressions a NaN (the symbol for which is Ø, U+2205), and so the correct answer is

```
0×0J1 ↔ ØJØ
```

This feature was suggested by David A. Rabenhorst.

At the moment, support for this feature is incomplete and experimental.

Implement Combinatorial Operator

This clever idea originated with the late mathematician Gian-Carlo Rota who conceived of "The Twelvefold Way"[4] in a series of lectures at MIT in the 1980s. It consolidates twelve common Combinatorial Algorithms into one 2×2×3 array all within the unifying concept of **Distributing** balls into boxes, subject to a **Capacity** choice. The three dimensions of the array may be described as follows:

- The **Balls** may be labeled or not {2 ways}
- The **Boxes** may be labeled or not {2 ways}
- The **Capacity** of Balls per Box may be one of **At Most One** | **Unrestricted** | **At Least One** {3 ways}

Amazingly, it turns out that if you look closely at all twelve possibilities, each one corresponds to a Combinatorial Algorithm. That is, this very APL-like idea encompasses the Combinatorial Algorithms of Permutations, Combinations, Partitions of a Number, Partitions of a Set, Multisets, and Tuples.

All the user needs to do is specify

- a three-digit Function Selector which chooses the Combinatorial Algorithm

- an optional Boolean value to choose whether you want to enumerate (0 = default) or generate (1) the values
- the # balls and boxes

For example, to count or generate Combinations, use a Function Selector of 010

```
      2!5 ⍝ The original way to enumerate values
10
      10⁞⍨2 5 ⍝ Another way ≡ 10 0⁞⍨2 5
10
      10 1⁞⍨2 5 ⍝ Generate values as 10×2 array
1 2
1 3
2 3
1 4
2 4
3 4
1 5
2 5
3 5
4 5
```

One of the great benefits of this design is that because the programmer simply selects which Combinatorial Algorithm to invoke, she doesn't need to program the fastest corresponding algorithm in APL as there is already a high-quality implementation written in a low-level language backing up her choice. In particular, I found all of the non-trivial Combinatorial Algorithms I needed in D. E. Knuth's TAoCP, Vol 4A, "Combinatorial Algorithms"[10], all of which were easily translated to C.

For more details, see "A Combinatorial Operator in APL"[17].

Grade on Nested and Heterogeneous Arrays

```
      ⍋(1 2) (1 1) (1 3) (¯1 0)
4 2 1 3
      ⍋1 'a' 0 '.'
3 1 4 2
```

where arrays are ordered by Rank, Shape, and Value, etc.

Set Function Fix Time

When the left argument to ⎕FX is a valid seven-element integer vector timestamp, that timestamp is used as the function's fix time. This way, functions stored external to the workspace can be restored to the workspace with their original timestamp. The interpretation of the timestamp is subject to the **User Preference** of "**Use local time (instead of UTC)**".

This feature was suggested by David A. Rabenhorst.

Precision Specifier For VFP Constants

A number after the VFP suffix may be used to specify the precision of the constant such as `1.1v64`. This feature should to be considered experimental and may be deprecated in the future.

This feature was suggested by David A. Rabenhorst.

Get A Variable's Precision

When the left argument to ⎕DR is the value 3, the result is a simple numeric scalar with the right argument's numeric precision:

```
      3 ⎕DR 0 1
1
      3 ⎕DR ⍳3
64
```

```
      3 ⎕DR ÷2 3
64
      3 ⎕DR ÷2 3x
∞
      3 ⎕DR 2.3v ⍝ Returns the value of ⎕FPC when the
                 ⍝   constant was tokenized
128
      3 ⎕DR 1v64
64
      3 ⎕DR 'a'  ⍝ Character width in bits
16
      3 ⎕DR 'a' 1
0
      3 ⎕DR ⊂1 2
0
```

System Function To Retrieve Native File Information

The system function ⎕NINFO is modeled after the function of the same name as defined by Dyalog, where the left argument lists zero or more categories of results such as

0: File Name
1: File Type
2: File Size
3: File Timestamp
4: Owner Security ID
5: Owner Name
6: Boolean Hidden State
7: Symbolic Link

For more details, see the NARS2000 Wiki entry[11] for ⎕NINFO.

Variant Operator Extensions

Syntax	Right Operand Values	Localized Meaning
L(×⍤*C*)R	'⍳' (Interior Product) '⍷' (Exterior Product)	(L R+.×R L)÷2 (L R-.×R L)÷2
(⍎⍤*N*)R	Numeric scalar	⎕FPC←*N*
L(|⍤*C*)R L(÷⍤*C*)R L(∨⍤*C*)R L(∧⍤*C*)R L(○⍤*C*)R L(⊤⍤*C*)R	'⍳' (Left quotient) '⍷' (Right Quotient)	⎕DQ←*C*
(!⍤*N*)R (!⍤*N1 N2*)R	Numeric scalar Numeric vector	Rising/Falling factorial length *N* step 1 (see below) Rising/Falling factorial length *N1* step *N2* (see below)
(⌹⍤*N*)R	1 2 3 4	Return vector of Eigenvalues Return matrix of Eigenvectors Return nested array of both Return nested array of both (see below) along with a Real matrix of the Schur vectors, one per column
(∘⌹⍤*C*)R	'⍳' (Left matrix rep) '⍷' (Right matrix rep) for Octonion scalars	⎕DQ←*C*

Rising/Falling Factorial

Pochhammer k-Symbol (Rising and Falling Factorials)[1] implemented through the Variant operator whose left and right operands are the

Shriek function and a signed integer scalar or one- or two-element integer vector. For example:

Rising Factorial, Default Step

```
      (!⍰3)10
1320
      ×/10 11 12
1320
```

Rising Factorial, Explicit Step

```
      (!⍰3 2)10
1680
      ×/10 12 14
1680
```

Falling Factorial, Default Step

```
      (!⍰¯3)10
720
      ×/10 9 8
720
```

Falling Factorial, Explicit Step

```
      (!⍰¯3 2)10
480
      ×/10 8 6
480
```

Eigenvalues and Eigenvectors

These concepts[5] from Linear Algebra and Matrix Theory define the characteristic values and vectors of the linear transformation represented by a matrix. Every square simple Real numeric matrix has Eigenvalues and Eigenvectors. To calculate these objects, use the Variant operator with a left operand of the Domino function and a right operand of an integer scalar.

`Z←(⍠1) R`	`Z` is a Complex floating point vector of the Eigenvalues
`Z←(⍠2) R`	`Z` is a Complex floating point matrix of the Eigenvectors one per column
`Z←(⍠3) R`	`Z` is a two-element nested vector with a Complex floating point vector of the Eigenvalues in the first element and a Complex floating point matrix of the Eigenvectors in the second
`Z←(⍠4) R`	`Z` is a three-element nested vector with a Complex floating point vector of the Eigenvalues in the first element, a Complex floating point matrix of the Eigenvectors in the second, and a Real matrix of the Schur vectors one per column in the third

For more details, see “A Matrix Operator in APL”[6].

A Matrix Operator

Normally in APL, when applying a scalar function to a matrix, the matrix is viewed as a container of its scalar elements and the scalar function applies to the individual elements. Matrix functions are different – in this case, the scalar function applies to the matrix as a whole[8], that is they treat the matrix as a new datatype. This concept is identical to that of the Multiset[7] operator where functions such as IndexOf (`L⍳⍤1 R`) and MemberOf (`L∈⍤1 R`) take on new meaning in that their arguments are treated as a new datatype, that is, sets with repeated elements where the repetitions play an integral role when calculating the result. Similarly, the Transpose, Matrix Inverse, and Matrix Divide functions and Inner Product and Determinant operators all apply to the matrix as a whole.

In the same manner, the (monadic) Matrix operator applies its (scalar function) operand to its matrix (right) argument as a new datatype to

which the function (left operand) is applied. Not all scalar functions have been extended to matrices, but a large number have and are in use in various scientific fields. For the nonce, this operator applies its function operand to **diagonalizable**[9] **fixed-precision Real** matrices only.

For more details, see “A Matrix Operator in APL”[6].

Extend Expand Function to Signed Integer Left Arguments

The Expand function has been extended to accept signed integer left arguments as per the original NARS implementation. If a negative value occurs in the left argument, the fill item of the right argument is selected for the result; if a positive values occurs in the left argument, an item is selected from the right argument. The magnitude of the values in the left argument indicates the number of times the selected item is repeated in the result, except that `0` has the same effect as `¯1`.

For example,

```
      1 1 2 1 1 ¯1 1 1 1\'BUTERFLY'
BUTTER FLY
```

This function is equivalent to the following:

```
     ∇ Z←L #DydSlope[X] R
[1]    Z←(1⌈|L)/[X] (L>0)\[X] R
     ∇
```

Session Manager

Line Continuations

This feature is available in both the **Session Manager** and the

Function Editor where it allows you to deal with long lines by breaking them into several shorter physical lines which are then treated as one logical line. For example:

```
        f←{
➥α ⍝ Left arg
➥+ ⍝ Function
➥ω ⍝ Right arg
➥}
        3 f 5
8
        h←{s←(+/⍵)÷2
➥  ⋄ √×/s-0,⍵ ⍝ Formula for triangle area
➥}
        h 3 4 5
6
```

The **Line Continuation Marker** ('➥' — U+27A5) is entered by pressing **Shift-Enter**.

In the **Function Editor**, this feature is most useful as a way to organize an unruly function header as well as to create a multi-line Anonymous Function/Operator (AFO).

Editing A User-Defined Function/Operator

Given the following function header

```
[0]     Z←{L1 L2} (LO DOP RO) R;⎕CT;⎕RL;B1;B5;LCL;
GLB;SGL;MUL;LSTACK;LSTACKLEN;RSTACK;RSTACKLEN;⎕IO
```

it can be rewritten using Line Continuations as several short lines

```
[0]    Z←{L1 L2} (LO DOP RO)R;⎕CT ⎕RL
      ➥ B1 B5
      ➥ LCL GLB SGL MUL
      ➥ LSTACK LSTACKLEN
      ➥ RSTACK RSTACKLEN
      ➥ ⎕IO
```

which can then be further organized into groups of related local variables along with trailing comments, as opposed to the typical practice of the function header as a disorganized unordered pile onto which names are tossed.

```
[0]    Z←{L1 L2} (LO DOP RO)R;
      ➥ ⎕CT ⎕RL ⎕IO       ⍝ Sys Vars
      ➥ B1 B5             ⍝ Limits
      ➥ LCL GLB SGL MUL   ⍝ Prefixes
      ➥ LSTACK LSTACKLEN  ⍝ Stack vars
      ➥ RSTACK RSTACKLEN  ⍝ ...
```

Editing An AFO

If the object being edited is an AFO, then **Shift-Enter** may be used (as above) to create additional physical but not logical lines, and **Enter** may be used to create additional both physical and logical lines.

For example, type

```
h←{s←(+/⍵)÷2 ⋄ ⍵/⍨×/s-0,⍵}
```

and then edit this function via ∇h or)EDIT h or double-clicking the name h to display a function editor window containing

```
[0]    h
[1]    s←(+/⍵)÷2 ⋄ ⍵/⍨×/s-0,⍵
```

This function may be edited further by deleting the diamond separator and pressing **Enter** at that point so as to split the line in two, resulting in

```
[0]     h
[1]     s←(+/⍵)÷2
[2]     √×/s-0,⍵ ⍝ Formula for triangle area
```

The **Enter** key is used to create a **multi-physical-line** AFO, whereas **Shift-Enter** creates a **multi-logical-line** AFO. The two types may appear together in a single AFO.

Note that when editing an AFO in this way, the function header line may not be changed.

System Labels In AFOs

Moreover, this method of editing an AFO also allows you to define additional entry points such as for an identity, prototype, or multiset function via the System Labels ⎕ID, ⎕PRO, ⎕MS, respectively. When system labels are used, they must appear at the start of a logical line. For example, the above function h could be edited to include an identity function entry point as follows:

```
[0]     h
[1]     s←(+/⍵)÷2
[2]     √×/s-0,⍵ ⍝ Formula for triangle area
[3]     ⎕ID:0
```

which allows the function to be used where an identity function is required such as

```
      h/⍬
0
```

Similarly

```
      f←{⍺+÷⍵}
```

requires an identity element when used in scan, as in

```
      f\10⍴1
DOMAIN ERROR
```

Because ∞ is the identity element for the function `f`, when it is edited to include the entry point `⎕ID:∞`, the above line produces the expected result now that `∞≡f⍬`:

```
      f\10⍴1
1 2 1.5 1.6667 1.6 1.625 1.6154 1.619 1.6176 1.6182
```

Output Debugging

Sometimes displayed output may appear oddly spaced. This may be due to several reasons one of which is the presence of non-printing ASCII Control Characters (whose character code is in the range 0x00 to 0x1F, excluding CR and LF). To better understand that case, you may turn on **Output Debugging** through the menu items **Edit | Customize... | User Preferences**. When enabled, this feature changes the way in which the above non-printing characters are displayed – instead of the normal behavior of ignoring the character, those characters are displayed using a special font which displays each character with its four digit hex code inside a box. For example,

```
      32⍴⎕AV
```

For ease in visually recognizing non-printing ASCII Control Characters, they are displayed in their own Syntax Color.

Accented Characters

Many international alphabets use diacritical marks as accent symbols on the letters such as á à â ä ã to name but a few such accents. NARS2000 now supports the five most common Latin-based accents:

Acute, Grave, Circumflex, Dieresis, and Tilde for all applicable letters. These new characters may be used anywhere the usual `a-z` and `A-Z` characters are used. For example,

ábćdéfǵhíjḱĺḿńóṕqŕśtúvẃxýź	ÁBĆDÉFǴHÍJḰĹḾŃÓṔQŔŚTÚVẂXÝŹ
àbcdèfghìjklmǹòpqrstùvẁxỳz	ÀBCDÈFGHÌJKLMǸÒPQRSTÙVẀXỲZ
âbĉdêfĝĥîĵklmnôpqrstuvwxyz	ÂBĈDÊFĜĤÎĴKLMNÔPQRŜTÛVŴXŶẐ
äbcdëfgẗïjklmnöpqrsẗüvẅẍÿz	ÄBCDËFGḦÏJKLMNÖPQRSTÜVẄẌŸZ
ãbcdẽfghĩjklmñõpqrstũṽwxỹz	ÃBCDẼFGHĨJKLMÑÕPQRSTŨṼWXỸZ

Entering Accented Characters

Different keyboards provide different ways in which to enter these accented characters. Nearly every keyboard layout that supports accented characters provides one or more dead keys. These keys (e.g. a Dieresis (¨)), when pressed (sometimes in conjunction with a modifier key such as Shift, Alt, or AltGR) do not display a character immediately, but wait for the next keystroke (the base character) to complete the process. If (say) the next key is a capital **U**, then the key displayed is **Ü**. Many of the accented characters in the above list can be entered in this way.

If the base character does not have an accented form, the system produces two symbols: the accent and the base character. This feature can be used to produce the accent character alone by use a space as the base character.

Global Dead keys

However, not all of the above accented characters are supported on keyboards in combination with their software drivers. To aid in entering all of the above accented letters, NARS2000 supports Global Dead Keys through its software drivers. For all of the NARS2000 built-in keyboard layouts (e.g., Danish, French, German, UK, and US), certain keystrokes have been reserved as Dead Keys. The built-in keyboard layouts are divided into two classes: those that use the Alt key to enter

APL characters and those that use the Ctl key. For those layouts in the former class, the reserved keystrokes are Ctl-Shift- in combination with the letters **a c d g t**; the layouts in the latter class reserve Alt-Shift-**a c d g t**. The letters **a c d g t** correspond to the first letter of the accent: **A**cute, **C**ircumflex, **D**ieresis, **G**rave, and **T**ilde. These choices may be overridden in case you need those keystrokes for your own purpose.

For example, to enter the letters **ĵ** or **Ĵ** on an Alt-type keyboard layout, press Ctl-Shift-**c** (for **C**ircumflex), release those keys and then press either **j** to display **ĵ** or **J** to display **Ĵ**.

APL System Functions

From within APL, you have access to all of the above Latin alphabets through the niladic system functions ⎕á ⎕à ⎕â ⎕ä ⎕ã for the lowercase Latin alphabets and ⎕Á ⎕À ⎕Â ⎕Ä ⎕Ã for the uppercase Latin alphabets. Also, ⎕a and ⎕A return the unaccented Latin alphabets. For example,

```
      ⍪⎕a ⎕á ⎕à ⎕â ⎕ä ⎕ã
abcdefghijklmnopqrstuvwxyz
ábćdéfǵhíjḱĺḿńóṕqŕśtúvẃxýź
àbcdèfghìjklmǹòpqrstùvẁxỳz
âbĉdêfĝĥîĵklmnôpqrstuvwxyz
äbcdëfgẗïjklmnöpqrs̈tüvẅẍÿz
ãbcdẽfghĩjklmñõpqrstũṽwxỹz
      ⍪⎕A ⎕Á ⎕À ⎕Â ⎕Ä ⎕Ã
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ÁBĆDÉFǴHÍJḰĹḾŃÓṔQŔŚTÚVẂXÝŹ
ÀBCDÈFGHÌJKLMǸÒPQRSTÙVẀXỲZ
ÂBĈDÊFĜĤÎĴKLMNÔPQRŜTÛVŴXŶẐ
ÄBCDËFGẗÏJKLMNÖPQRSTÜVẄẌŸZ
ÃBCDẼFGHĨJKLMÑÕPQRSTŨṼWXỸZ
```

Also, the niladic system function ⎕CS returns a character array of shape `10 6 2 28` which as the left argument to the Grade functions can act

as a Collating Sequence (or the starting point for your own version of a Collating Sequence) when you need to sort character arrays containing any of the above accented characters.

Drag and Move vs. Drag and Copy

In both the function and session editors, the mouse may **Drag and Move** or **Drag and Copy** a selection to another location in the same window. The former is equivalent to Cut and Paste, while the latter is equivalent to Copy and Paste, both in a single atomic operation.

After selecting the text (which may span multiple lines), release the left mouse button. Then move the mouse back to any point within the selected text, press and hold the left mouse button and Move/Copy the text to its new location. Release the left mouse button at the appropriate insertion point.

During the Move/Copy, the text caret shows exactly the position of the insertion point. and the mouse cursor indicates whether or not the Ctrl-key is pressed. When the left mouse button is released, if the Ctrl-key also is pressed, then the operation becomes **Drag and Copy** which Copies the selected text (without Cutting it) and Pastes it at the insertion point.

The Undo key (Ctrl-Z) restores all of the Cut/Copy/Pasted text to its previous state.

Implementation Features

Elided indices as in `A[I;;J;]` use a temporary APV instead of special treatment throughout indexing. That is the above expression is roughly equivalent to writing `A[I;⍳(⍴A)[2];J;⍳(⍴A)[4]]`. Use of this technique netted about 150 fewer lines of source code (200 of special handling of elided indices less 50 extra lines for the temporary APV).

Miscellaneous Features

- New System Command `)FOPS` to list both `)FNS` and `)OPS`
- Command line arguments for the executable to set Symbol and Hash Table sizes on startup
- New System Command `)SYMB` to display the sizes of the internal Symbol and Hash Tables. Eventually this command will be extended to **set** new sizes for those tables
- Two alternate Hypercomplex notations suggested by David A. Rabenhorst:
  - Unit Normal Radian Point Notation for Complex numbers as in `1au0.25` = `1ad90` where the `0.25` represents a quarter turn, i.e. `90°`, counter-clockwise
  - Digraphs for Octonion numbers to make the notation more regular by using one-letter separators for Complex and Quaternion numbers and two-letter separators for Octonions as in `1i2j3k4os5oi6oj7ok8`
- Optimize Nth prime search (`¯2⍎N`) for `N < 1E5`
- Optimize Nth prime count (`2⍎N`) for `N < 129174`
- Implement niladic system function `⎕SI` to return the contents of the State Indicator
- Support decimal points in base Point Notation such that
  `16bff.f ↔ 255.9375`
  `2b101.01 ↔ 5.25`

Current & Future Work

Extend Iverson's Determinant Operator[2]

While implementing in APL a special case of this operator (`+.×M`) known as the Permanent of a Matrix[3] by H.J. Ryser, I found that Ryser's algorithm when applied to under-determined matrices (i.e., `<⌿⍴M`)

returned results different from Iverson's. Eventually, I traced the difference to the fact that Iverson had defined the determinant of that case to be identical to the determinant of the square matrix obtained by truncating trailing columns of the original matrix, i.e., `(2⍴1⍴M)↑M` – in contrast, the values in those truncated columns are significant in Ryser's algorithm.

This leads me to conjecture that the determinant operator could extend to all under-determined cases for all operands.

MP Floats: Set Precision As Lexical, Not Tokenized

This very subtle issue has caused me considerable pain when I was trying to track down a bug caused by it that appears at first to be far removed. The question revolves around the simple statements

```
      ⎕FPC←128
      ⎕FPC←256 ⋄ A←1.2∨
      3 ⎕DR A
128
```

As you can see, the actual precision of `A` is **not** the expected value of `256`. I'm willing to bet that every APL implementor here whose implementation contains a tokenization phase would code it the way I first did and regret it later. The problem is that the precision of constants is a property you think is set at Lexical time, but is actually set at Tokenization time.

Finish Support for Matrix Operator

The current implementation is incomplete in that it doesn't handle non-diagonalizable matrices. Such matrices can be resolved using Jordan Canonical Form along with successive order derivatives of the function operand on the Eigenvalues.

Implement Derivative Operator

While working on the Matrix operator, I've found the Derivative operator very useful. My first step is to look at what J has done, but in the meantime, a very naive version of this operator can be defined as follows:

```
      Δ←{α←1E¯40x ⋄ ((αα ω+α)-αα ω)÷α}
```

For example,

```
      !Δ 0
¯0.57721566490153286060651209008240243104 21
      ¯1g1x
¯0.57721566490153286060651209008240243104 22
```

That is, the first derivative of the Factorial function at 0 is -γ, where γ is the Euler-Mascheroni constant.

Finish Support for NaNs

The current implementation is incomplete and inconsistent in a few places.

Support User Commands

This excellent idea from APL-Win and later enhanced by Dyalog is sorely missing from NARS2000.

Integrate GnuPlot

Now that NARS2000 supports Quaternions, they are the perfect tool to rotate, scale, and translate multi-dimensional objects.

Support Function Keys

In a previous product (for DOS) which involved allowing the user to edit the screen, I had implemented a simple language which the user could personalize for the function keys and other special keys. Something like this is needed again.

Extend Syntax Coloring To Show Input v. Output

Perhaps by using a background color or a leading line marker?

Access to External Processes

What with Hypercomplex numbers available, there is a greater need to show them off, such as plotting as they rotate a multi-dimensional figure. This is a job for Shared Variables and Name Association and possibly a system function or two.

Online Version

This paper is an ongoing effort and can be out-of-date the next day. To find the most recent version, go to http://sudleyplace.com/APL/ and look for the title of this paper on that page.

References

1. Kenneth E. Iverson (August 2006), "APL in the New Millennium", Vector, 22 (3), archive.vector.org.uk/art10004040
2. Kenneth E. Iverson, “Determinant-Like Functions Produced by the Dot Operator”, SATN-42 (Sharp APL Technical Notes), 1982-04-01, www.jsoftware.com/papers/satn42.htm

3. Wikipedia, “Computing the Permanent”, en.wikipedia.org/wiki/Computing_the_permanent#Ryser_formula
4. Wikipedia, “TwelveFold Way”, en.wikipedia.org/wiki/Twelvefold_way
5. Wikipedia, “Eigenvalues and Eigenvectors”, en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors
6. Smith, Bob, “A Matrix Operator in APL”, www.sudleyplace.com/APL/A Matrix Operator In APL.pdf
7. “Multisets”, wiki.nars2000.org/index.php/Multisets
8. Wikipedia, “Matrix Function”, en.wikipedia.org/wiki/Matrix_function
9. Wikipedia, “Diagonalizable Matrix”, en.wikipedia.org/wiki/Diagonalizable_matrix
10. Knuth, Donald E., “The Art of Computer Programming”, Addison Wesley, Volume 4A, Combinatorial Algorithms, p. 390, ISBN 0-201-89685-0
11. NARS2000 Wiki: ⎕NINFO wiki.nars2000.org/index.php/System_Function_NINFO#.E2.8E.95NINFO
12. Smith, Bob, “Hypercomplex Numbers in APL” www.sudleyplace.com/APL/HyperComplex Numbers in APL.pdf
13. Smith, Bob, "Hypercomplex GCD in APL", www.sudleyplace.com/APL/Hypercomplex GCD in APL.pdf
14. Smith, Bob, "Hypercomplex Quotients in APL", www.sudleyplace.com/APL/Hypercomplex Quotients in APL.pdf
15. Smith, Bob, "Hypercomplex Notation in APL", www.sudleyplace.com/APL/Hypercomplex Notation in APL.pdf
16. Smith, Bob, "Hypercomplex Implementation in APL", www.sudleyplace.com/APL/Hypercomplex Implementation in APL.pdf
17. Smith, Bob, "A Combinatorial Operator in APL", www.sudleyplace.com/APL/A Combinatorial Operator in APL.pdf