TRUE BASIC

The Original BASIC

Huge Arithmetic Toolkit

Reference Guide

Huge Arithmetic Toolkit

by Thomas E. Kurtz
Co-inventor of BASIC

Introduction

This toolkit allows you to write True BASIC programs in the usual way except that certain numeric variables and arrays may be declared to be of type **HUGE**.

A **DO** program then revises your True BASIC program into one that can be run directly and that will perform the calculations using huge number arithmetic, if necessary.

You may use most of the True BASIC statements and structures including modules, internal and external subroutines, public and shared variables, etc. There are some restrictions.

1. All variables and arrays that are intended to be huge must be so declared early on in the main program, subroutine, or module, and before any other statements, such as **LOCAL** or **PUBLIC**, that contain these variables. There is a special requirement for subroutines.
2. A huge declaration must be included just after the **SUB** statement in the definition if some or all of the parameters in the **SUB** statement are to be huge. The same is true for multiple-line **DEF** structures. (Single-line **DEF**s cannot be of type huge nor can their arguments be huge. The solution is to make them into multiple-line **DEF**s.)
3. Only simple **MAT** statements may be used.

The toolkit operates as a **DO** program, revising the contents of the source program in the current editing window. The subroutines of the toolkit may be pre-loaded, using

```
LOAD loadhuge
```

but this is not necessary. After a successful revision, the resulting modified source program can be run directly, or saved.

This toolkit combines that portion of the former **Mathematician's Toolkit** that dealt with huge number arithmetic for simple and matrix entities.

Example

```
! Tests an integer by Fermat's Theorem to see whether prime.
! Integer must be positive, greater than 2.
! Checks whether 2^(n-1) is congruent to 1 mod n.

DECLARE HUGE n, n1, power

PRINT "Test for prime numbers"
PRINT "Type '0' to stop"
PRINT
DO
   PRINT "Number";
   INPUT n
   IF n = 0 then STOP                   ! Type '0' to stop
   LET n1 = n - 1                       ! n-1
   LET power = mod(2^n1,n)              ! 2^(n-1) mod n
   IF power = 1 then
      PRINT "Probably a prime"


   ELSE
      PRINT "Not a prime"

   END IF

LOOP

END
```

Notice that all you really have to do is to insert a

```
DECLARE HUGE n, n1, power
```

into the main program, early on, to notify the toolkit that the variables ‘n’, ‘n1’, and ‘power’ are going to be huge numbers.

The next step is to revise the program using

```
do huge
```

After the revision, this will be your program in the current editing window:

```
DECLARE DEF h_in$,h_isequal,h_diff2$,h_pwr1$,h_mod$
LIBRARY "HugeLibs.trc"
! Tests an integer by Fermat's Theorem to see whether prime.
! Integer must be positive, greater than 2.
! Checks whether 2^(n-1) is congruent to 1 mod n.

! declare huge n, n1, power

PRINT "Test for prime numbers"
PRINT "Type '0' to stop"
PRINT
DO
   PRINT "Number";
```

```
     INPUT h_N$
     LET h_N$ = h_in$(h_N$)
     IF h_isequal(h_N$,h_in$(str$(0))) = 1 then STOP
     LET h_N1$ = h_diff2$(h_N$,1)! n-1
     LET h_POWER$ = h_MOD$(h_pwr1$(2,h_N1$), h_N$)
     IF h_isequal(h_POWER$,h_in$(str$(1))) = 1 then
        PRINT "Probably a prime"

     ELSE
        PRINT "Not a prime"

     END IF

  LOOP

  END
```

Notice these changes:

1. Two statements (**DECLARE DEF** and **LIBRARY**) have been added near the top of the program. If the huge toolkit has been "loaded," these statements are not needed. The **DECLARE DEF** statement names all the defined functions used by the program. (A defined function is defined by a **DEF** statement, in contrast with, for example, the **SIN** function, which is builtin.) The special functions used in the main program are: **h_in$**, **h_isequal**, **h_diff2$**, **h_pwr1$**, **h_mod$**.
2. The variables **n**, **n1**, and **power** have been declared to be of type **HUGE**. The **DO** program changes them to **h_N$**, **h_N1$**, and **h_POWER$**, respectively. All numeric variable names declared to be huge are preceded with "**h_**" and followed by "**$**".
3. All numeric and relational operations involving huge numbers have been converted to function invocations.

How it is Done

Huge numbers are stored as strings. The first three bytes of the string contain the algebraic sign of the number and the number of decimal places. Each subsequent group of three bytes contains six digits of the number, with the least significant part coming first. The advantage is that a single huge number can be stored and manipulated as a single string rather than as a large numeric vector.

You write your program in the usual way. To convert it to huge, type

```
do huge
```

This **DO program** converts the current program in the editing window into a huge one.

If you are not using the **Gold Edition** of True BASIC, you should now type

```
rename _huge_
```

or something similar, to prevent accidently saving the revised program over your original saved version.

Now type the command

```
run
```

or simply select **Run** from the Run menu.

The huge number tool kit runs equally well whether set up as a loaded workspace or not. If a loaded workspace, the startup time for the do program is much shorter. To load the huge toolkit workspace, enter the huge toolkit directory and type

```
script loadhuge
```

If your huge toolkit is not loaded as a workspace, just make sure that the compiled toolkit file **HugeWork.trc** is in your current directory, or that there is an 'alias' to allow True BASIC to find it.

The file **HugeWork.trc** contains the library file **HugeLibs.trc**, so don't also load the latter file.

MAT Operations

The following **MAT** operations are permitted. (These form a subset of the **MAT** operations allowed in True BASIC.)

```
MAT C = A + B
MAT C = A - B
MAT C = A * B
MAT C = K * A                 ! k a scalar variable or constant
MAT C = INV(A)
MAT C = TRN(A)
MAT C = CONJ(A)
MAT C = IDN
MAT C = CON
MAT C = ZER
```

It is a strict requirement that the dimensions and subscript ranges must match.

Adding and subtracting are allowed for vectors and matrices. Products are allowed for two matrices, a vector and a matrix, a matrix and a vector, and for two vectors. Again, the subscript ranges must conform! If you multiply a vector times a matrix, the vector will be interpreted as a row vector. If you multiply a matrix times a vector, the vector will be interpreted as a column vector. If you multiply two vectors, the dot product will be assumed.

Scalar multiplication is allowed for both vectors and matrices.

INV, **TRN**, and **IDN** are allowed only for matrices.

The **PUBLIC** variable **h_det$** will be the huge determinant of the most recently successfully inverted matrix.

Input and Printed Output

Input is always in terms of real numbers. Ordinary INPUT and READ statements can be used. To input a huge number, use something like

```
DECLARE HUGE n
INPUT prompt "Enter constant: ": n
...
```

The same approach can be used to **READ** a huge constant

```
DECLARE HUGE c
READ c
DATA 17
```

To build a huge matrix, you can use the **READ** and **DATA** statements to create

```
DECLARE HUGE A
DIM A(3,3)
MAT READ A
DATA 1, 2, 3
DATA 2, 3, 4
DATA 3, 4, 5
```

You can output numbers in the usual way.

```
PRINT x, y
```

If the value is huge, then

```
DECLARE HUGE z
...
PRINT z
```

will work. This is converted into **PRINT h_out$(z)**, where **h_out$** is a function that converts a huge number into a string, suitable for printing.

You can control the number of decimals places of accuracy with the **DECIMALS** statement.

```
DECIMALS 4
```

will cause all subsequent uses of **h_out$** to round to four decimals places (actually six, since decimal places are added in groups of six.)

You can include several **DECIMALS** statements in your program; see the demonstration program **DemHilb.tru**.

In addition, there is another function **h_outd$(z,d)** that allows you to convert to a string a huge value rounded to '**d**' decimal places.

Demonstration Programs

DemE.tru

Computes the mathematical constant e to the number of decimal places prescribed. Uses the infinite series, not the builtin constant.

DemHilb.tru

Generates a Hilbert matrix, which is known to be extremely ill-conditioned, and then inverts it. H(i,j) = 1/(i+j-1). Prints the inverse and the determinant. The computation is done with a large number of decimal places, enough to guarantee reasonable accuracy.

DemMat.tru

Demonstrates several matrix operations, including converting from strings of digits to huge, and vice versa

DemPrime.tru

Applies Fermat's Theorem to determine is the integer inputed is prime or not. If 2^(n-1) MOD n = 1, then n could be a prime; otherwise, n is most certainly not a prime. This requires raising 2 to a large power. If n > 54 or 55, 2^(n-1) cannot be done accurately with IEEE eight-byte double precision.

Reference List

Here is a list of all available huge functions.

h_zero$	Zero
h_one$	One
h_two$	Two
h_pi$	π to 126 decimal places
h_twopi$	2π
h_pi2$	π/2
h_pi4$	π/4
h_e$	e to 126 decimal places
h_eps$	1 in last place
h_in$(s$)	String of digits to huge
h_conv$(n)	Converts a real to a huge
h_out$ (n$)	Huge to string of digits
h_outd$(n$,d)	Huge to string of digits, d decimals
h_sum$(a$,b$)	Addition: huge plus huge
h_sum1$(a,b$)	Addition: real plus huge
h_sum2$(a$,b)	Addition: huge plus real
h_diff$(a$,b$)	Subtraction: huge minus huge

h_diff1$(a,b$)	Subtraction: real minus huge
h_diff2$(a$,b)	Subtraction: huge minus real
h_prod$(a$,b$)	Multiply: huge times huge
h_prod1$(b,a$)	Multiply: real times huge
h_prod2$(a$,b)	Multiply: huge times real
h_quot$(a$,b$)	Division: huge divided by huge
h_quot1$(a,b$)	Division: real divided by huge
h_quot2$(a$,b)	Division: huge divided by real
h_pwr$(n$,p$)	Power: huge ^ huge integer
h_pwr1$(n,p$)	Power: real ^ huge integer
h_pwr2$(n$,p)	Power: huge ^ integer
h_pwrmod$ (n$, p$, m$)	Huge ^ huge MOD huge
h_IsZero(n$)	Equals 1 if n = 0, 0 otherwise
h_IsAbsGreater(a$,b$)	Equals 1 if |a| > |b|, 0 otherwise
h_IsEqual(a$,b$)	Equals 1 if a = b, 0 otherwise
h_IsNotEqual(a$,b$)	Equals 1 if a ≠ b, 0 otherwise
h_IsGreater(a$,b$)	Equals 1 if a > b, 0 otherwise
h_IsLess(a$,b$)	Equals 1 if a < b, 0 otherwise
h_gcd$(a$,b$)	Euclidean algorithm, gcd
h_mod$(a$,m$)	Mod, a MOD m
h_abs$(a$)	Absolute value ABS
h_int$(a$)	Integer part INT
h_neg$(a$)	Negation
h_sqr$(n$)	Square root function
h_sin$(x$)	Sine function
h_cos$(x$)	Cosine function
h_tan$(x$)	Tangent function
h_atn$(x$)	Arctangent function
h_exp$(x$)	Exponential function
h_log$(x$)	Natural logarithm function

The following subroutines carry out the huge matrix operations. In all cases, the last argument is the result matrix or value.

SUB H_MatIn (A$(,))	Convert matrix of string of digits to matrix of huge
SUB H_MatSum (A$(,), B$(,), C$(,))	MAT C = A + B, matrices
SUB H_MatDiff (A$(,), B$(,), C$(,))	MAT C = A - B, matrices

SUB H_MatProd (A$(,), B$(,), C$(,))	MAT C = A*B, matrices
SUB H_MatPrint (A$(,))	MAT PRINT A, matrix
SUB H_MatPrintd (A$(,), d)	MAT PRINT A, d decimals
SUB H_MatIdn (A$(,))	MAT A = IDN
SUB H_MatTrn (A$(,), B$(,))	MAT B = TRN(A)
SUB H_MatScmH (A$(,), h$, B$(,))	MAT B = h*A, matrices
SUB H_MatInv (A$(,), B$(,))	MAT B = INV(A)
SUB H_VecIn (V$())	Convert vector of string of digits to vector of huge
SUB H_VecSum (V$(), W$(), Z$())	MAT Z = V + W, vectors
SUB H_VecDiff (V$(), W$(), Z$())	MAT Z = V - W, vectors
SUB H_VecPrint (V$())	MAT PRINT V, vector
SUB H_VecPrintd (V$(), d)	MAT PRINT V, d decimals
SUB H_MatVecProd (A$(,), W$(), Z$())	MAT Z = A*W
SUB H_VecMatProd (W$(), A$(,), Z$())	MAT Z = V*A
SUB H_VecScmH (V$(), h$, Z$())	MAT Z = h*V, vectors
SUB H_DotProd (V$(), W$(), n$)	LET n = DOT(V,W)

The following array relational functions are also available.

h_MatIsEqual (A$(,), B$(,))	Returns 1 if A$ = B$, 0 otherwise
h_VecIsEqual (V$(), W$())	Returns 1 if V$ = W$, 0 otherwise

Error Messages

100, "String is not a number: "
200, "Illegal argument"
210, "Must be positive integer"
220, "Illegal argument for decimals"
230, "Squareroot of a negative number"
240, "Zero or negative argument for h_log$"
250, "h_gcd$ must have integer arguments"
260, "Negative number to non-integer power"
270, "Arguments must be positive integers"
280, "Bad value for h_tan$"
400, "Illegal number for decimals"
500, "Dimensions do not match"
501, "Lower bounds are not equal"
505, "Must be square matrix"
510, "Determinant is 0"PRINT

December 11, 2000