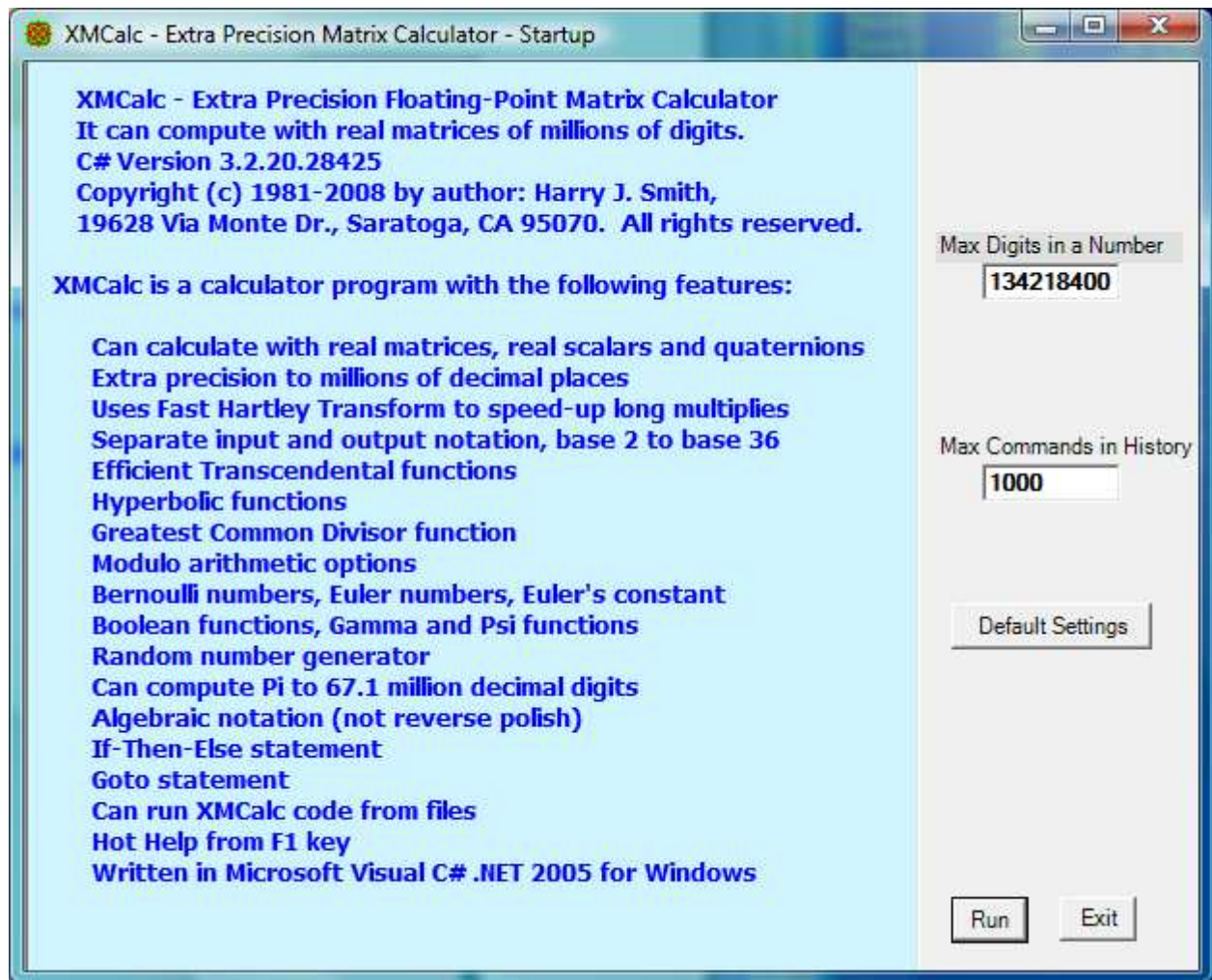


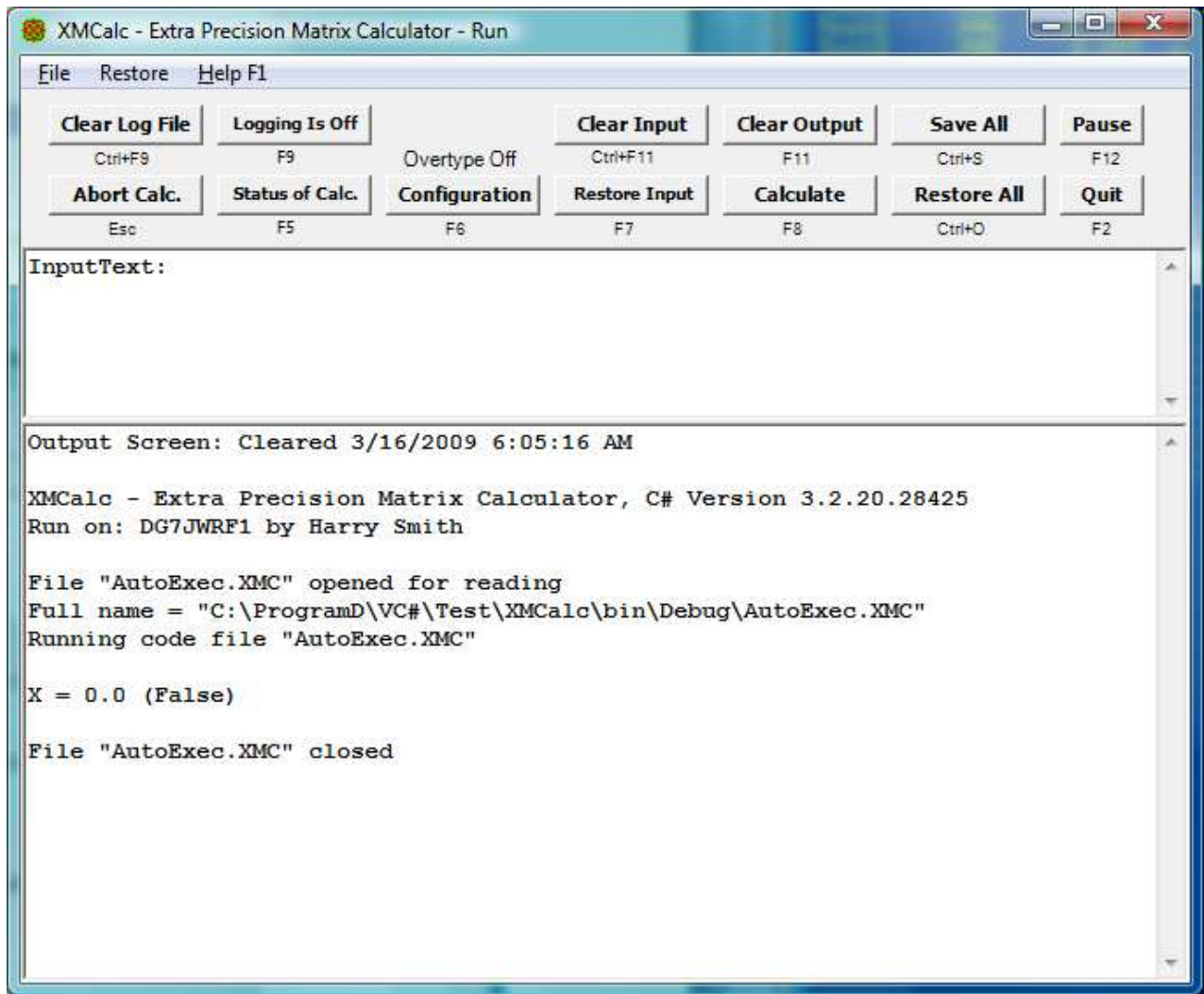
This document applies to the C# Version **3.2.20.28425** of XMCalc,  
Copyright (c) 1981-2008 by author: Harry J. Smith, Saratoga, CA.

Introduction -

When you execute the program XMCalc.exe it responds by displaying the Startup form:



Just click the Run button and the Run form is displayed:



Numbers are stored in memory in decimal; actually they are stored in base 100,000,000 as an array of super-digits. Each super-digit is stored as a 32-bit integer between 0 and 99,999,999. As variables change in value, memory is dynamically reallocated so no more memory is used than is needed to represent their current precision.

At any given time there is a current number of decimal digits that will be computed for a mantissa, a max decimal digits allow in a mantissa, and a max decimal digits ever allowed in a mantissa. These values are initialized to 56, 134218400, and 134218400 respectively and can be changed after the program is running. The current can be changed by the M primitive and the max allowed can be changed by the SetMax procedure. The max ever allowed can only be changed by changing the Max Digits in a Number field on the Startup form.

The Run form has two large text boxes. The upper one is for command input. Commands may be entered one at a time each followed by a return, or several separated by one or more blank spaces or semicolons. A separator is never needed between primitive op codes and is only needed otherwise to prevent ambiguity of meaning. Commands are not case sensitive, upper and lower case letters are always interpreted the same.

There are four basic types of commands: 1) Enter a number, 2) Execute a primitive op code, 3) Evaluate an equation, and 4) Do a procedure.

The calculator contains a list of named numbers or variables. Initially the list contains only the item X = 0.0. Its name is X and its value is 0.0. Items can be added to the list by evaluating an equation. Equations are assignment statements

like {variable} = {expression}. A {variable} is a name of a variable and an {expression} is an expression of terms, factors, functions, variables, constants, and {expression}s. Item names are alphanumeric with the first character alphabetic, have all characters significant but not case sensitive.

A given named numbers on the list can be a scalar or a matrix. A scalar is a real number. A matrix is a rectangular array of scalars arranged in r rows and c columns. The element in row i and column j is labeled as element i, j with  $1 \leq i \leq r$ ,  $1 \leq j \leq c$ . A one by one matrix is not considered to be a scalar and a scalar is not a matrix.

An m by 1 matrix is a column vector. A 1 by n matrix is a row vector. A polynomial  $a(n)*x^n + a(n-1)*x^{(n-1)} + \dots + a(1)*x + a(0)$  is stored as a (n + 1) by 1 column vector {a(n); a(n-1); ..., a(1); a(0)}.

Parentheses can be nested to any level in expressions. Any number of closing parentheses can be replaced with a single semi-colon or an end-of-line. Thus  $x = (a / (b * (c + d);$  is a legal assignment statement and is interpreted as  $(a / (b * (c + d)))$ .

Any time a variable is referenced that is not currently on the list, it is added to the list with a value of 0.0.

At any given time, one item on the list is the active item. This is referred as Top, the item on top of the list. Initially item X is the active item. When an expression is evaluated, the variable being assigned a value becomes the active item. If the {expression} part of an assignment statement is left blank, like X=, the referenced variable becomes the active item without changing its value.

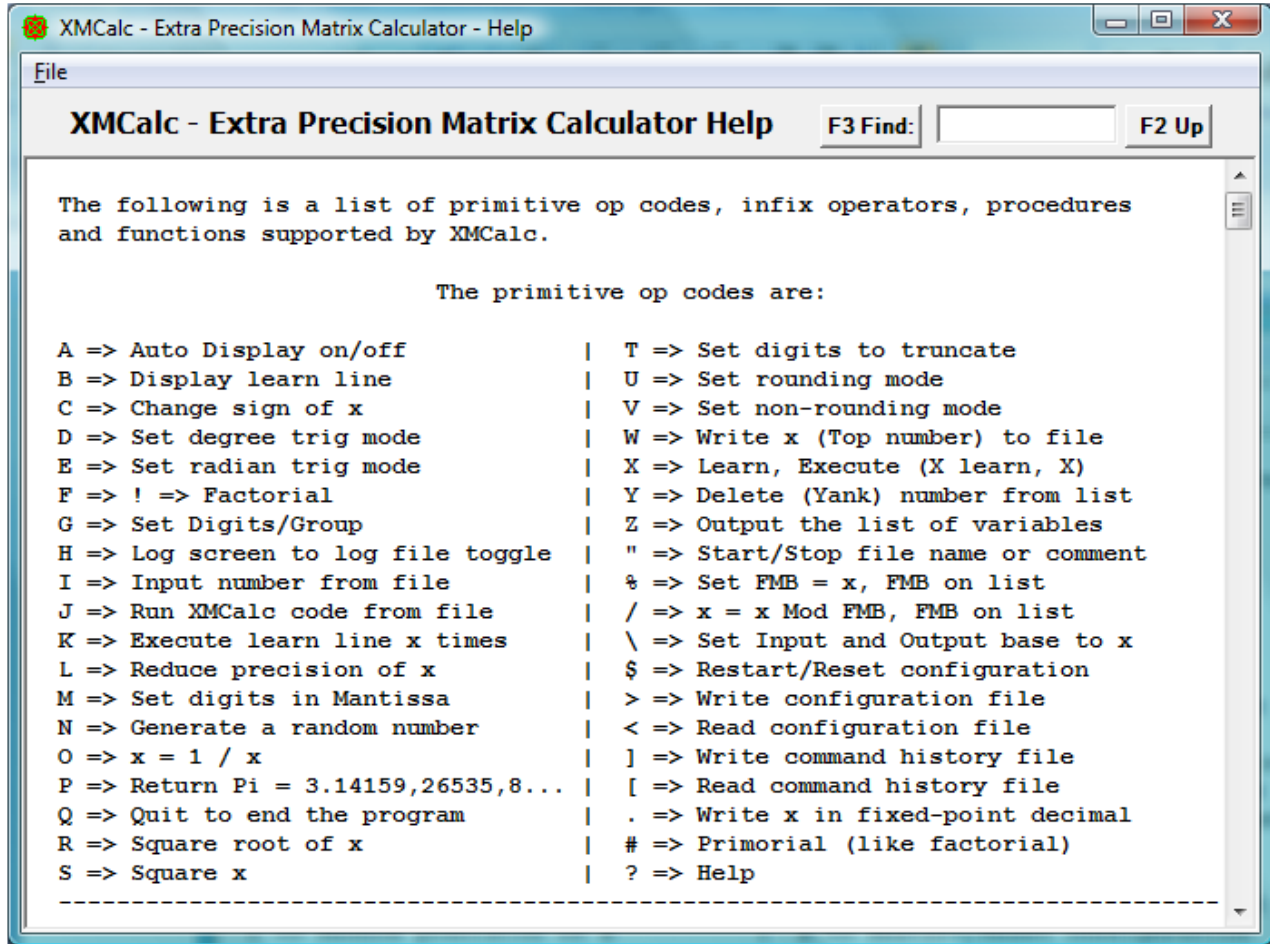
When a number is entered, it replaces the value of the active item. Numbers (constants) may have a leading sign and embedded commas. An example of a constant is -12,345.678,9E+1,234. The commas, plus signs, decimal point, and the E power of 10 factor are optional. The numbers 1.0E+1,50323,85525 is at the upper end of the dynamic range of the calculator. Because commas are allowed in input constants to make them readable, a single comma cannot be used to separate numeric arguments in functions calls. An example of this is: X = ATan2(12,345'78,901). A tic mark separates the two arguments instead of a comma as is normally done. A ", " also works as in ATan2(12,345, 78,901).

There is a special feature that allows for the entry of an equation without the "X=" preceding it. If the first character of a command is ">= '0' and <= '9' and <= MaxDigit" or equal to "(", "-", "+", or "=", the command will be prepended with "Top.Nm=" (no == though), where Top.Nm is the name of the item on top of the number list. This is done for each command of a multi-command command line. This also works for the other assignment operators (+=. -=, \*=, /=, and %=). For example, +=Y will be treated as  $X = X + (Y)$ . Y is evaluated and added to the top item on the list.

If the program is executed from the DOS prompt with one or more arguments, the initial Startup form is not displayed and the arguments are taken as an initial XMCalc command line. This allows you to control the execution of XMCalc from batch files and XMCalc code files with no operator intervention. The XMCalc code file AutoExec.XMC is always run first, even before the DOS command line commands. The ASCII Tab character (9) is allowed in XMCalc code files and in the initial command arguments and treated as a blank space.

Special handling is given to the first argument on the DOS command line. If it ends in .XMC, it is changed to Run("... .XMC") so this XMCalc code file will be run. If it ends in .XMN, it is changed to ReadN("... .XMN") so this XMCalc number file will be read and added to the list of items. This allows XMCalc to be run by Windows or a program, like ZTree, by associating the file XMCalc.Exe with the extensions XMC and XMN, and then opening a file with one of these extensions.

? => Help: the "?" primitive causes the Help form to be displayed:



The Help form is scrollable and resizable. It contains a list of primitive op codes, infix operators, procedures, functions supported, Function key actions, and commands used in XMCalc code files:

The primitive op codes are:

A => Auto Display on/off		T => Set digits to truncate
B => Display learn line		U => Set rounding mode
C => Change sign of x		V => Set non-rounding mode
D => Set degree trig mode		W => Write x (Top number) to file
E => Set radian trig mode		X => Learn, Execute (X learn, X)
F => ! => Factorial		Y => Delete (Yank) number from list
G => Set Digits/Group		Z => Output the list of variables
H => Log screen to log file toggle		" => Start/Stop file name or comment
I => Input number from file		% => Set FMB = x, FMB on list
J => Run XMCalc code from file		/ => x = x Mod FMB, FMB on list
K => Execute learn line x times		\ => Set Input and Output base to x
L => Reduce precision of x		\$ => Restart/Reset configuration
M => Set digits in Mantissa		> => Write configuration file
N => Generate a random number		< => Read configuration file
O => x = 1 / x		] => Write command history file
P => Return Pi = 3.14159,26535,8...		[ => Read command history file
Q => Quit to end the program		. => Write x in fixed-point decimal
R => Square root of x		# => Primorial (like factorial)
S => Square x		? => Help

---

The infix operators are:

```
A = X + Y => Set A to X plus Y
A = X - Y => Set A to X minus Y
A = X * Y => Set A to X times Y
A = X / Y => Set A to X divided by Y
A = X ^ Y => Set A to X to the power Y
A = Y @ X => Set A to ATan2(Y over X)
A = X # Y => Set A to Mag(X, Y) = SqRt(Sq(X) + Sq(Y))
A = X % Y => Set A to Mod(X, Y) = X Modulo Y
A = X \ Y => Set A to Floor(X/Y), integer divide
A = X & Y => Set A to 1 if X and Y are not 0, else set A to 0
A = X | Y => Set A to 1 if X or Y, is not 0, else set A to 0
A = X < Y => Set A to 1 if X < Y, else set A to 0
A = X > Y => Set A to 1 if X > Y, else set A to 0
A = X = Y => Set A to 1 if X = Y, else set A to 0, same as ==
A = X == Y => Set A to 1 if X = Y, else set A to 0
A = X <= Y => Set A to 1 if X <= Y, else set A to 0
A = X != Y => Set A to 1 if X not = Y, else set A to 0
A = X <> Y => Set A to 1 if X not = Y, else set A to 0, same as !=
A = X >= Y => Set A to 1 if X >= Y, else set A to 0
A = X -- Y => Set A to X * Repunit(Y), 3--5 = 33333, five threes
A = X ** Y => Set A to PolMul(X, Y)
```

---

The assignment operators are:

```
= => Set Top to Top (no-op)
X = => Bring X to top of list
= Y => Set Top to Y
+= Y => Set Top to Top plus Y
-= Y => Set Top to Top minus Y
*= Y => Set Top to Top times Y
/= Y => Set Top to Top divided by Y
%= Y => Set Top to Mod(Top, Y) = Top Modulo Y
X = Y => Set X to Y
X += Y => Set X to X plus Y
X -= Y => Set X to X minus Y
X *= Y => Set X to X times Y
X /= Y => Set X to X divided by Y
X %= Y => Set X to Mod(X, Y) = X Modulo Y
```

---

The procedures supported are:

```
AllD(X) => Compute all divisors of X
AllowFHT(X) => Set Allow FHT multiple on if X != 0, else off
AutoDisplay(X) => Set Auto display on if X != 0, else off
Base(X) => Set Input and Output base to X, [2, 36]
BaseI(X) => Set Input base to X, [2, 36]
BaseO(X) => Set Output base to X, [2, 36]
BernM => Returns the Max index of saved Bernoulli numbers
Cat => Returns Catalan's constant G = 0.91596,55941,77219...
ChDir(F) => Change directory to F = "ccc...c", F optional
ClearBern => Clear storage of saved Bernoulli numbers
ClearEuler => Clear storage of saved Euler numbers
ClearHist => Clear history of previous operator entries
ClearLog => Clear the log file
Diag(X) => Set diagnostic mode on or off, (X) is optional
Ee => Returns e = Exp(1) = 2.71828,18284,59045...
EulerC => Euler's constant gamma = 0.57721,56649,01532...
EulerM => Returns the Max index of saved Euler numbers
Exit => Totally quit the program with no questions asked
```

ForceFHT(X) => Set Force FHT multiple on if X != 0, else off  
 GenBern(X) => Generate and save Bernoulli number upto B(X)  
 GenEuler(X) => Generate and save Euler number upto E(X)  
 HelpH(X) => Set Height of Help form in pixels  
 HelpW(X) => Set Width of Help form in pixels  
 HistH(X) => Set Height of History form in pixels  
 HistW(X) => Set Width of History form in pixels  
 Ln10 => Returns the natural log of 10 = 2.30258,50929,94045...  
 LogScreen(X) => Log screen to log file mode, on or off  
 LX => LT => Restore LastTop to top of the list  
 Next => Move to next item on the list (no argument)  
 Pause => Pause the calculations to free the processor  
 PFA(X) => Run prime factor algorithm A on X  
 PFB(X) => Run prime factor algorithm B on X  
 PFE(X) => Run prime factor algorithm ECM on X (fastest)  
 PTab(X) => Write prime table to XICalcPTab.txt, X primes  
 Phi => Returns Golden Ratio = (1 + SqRt(5)) / 2 = 1.61803...  
 PhiP => Returns Phi prime = (1 - SqRt(5)) / 2 = -0.61803,39...  
 Pi => Same as the P command, Returns Pi = 3.14159,26535,8...  
 PiAa => Compute x = Pi by Borwein algorithm a  
 PiAb => Compute x = Pi by Borwein algorithm b  
 PiAGM => Compute x = Pi by Schoenhage AGM algorithm  
 PiCh => Compute x = Pi by Chudnovsky binary splitting algo  
 PiGL => Compute x = Pi by Gauss-Legendre algorithm  
 Pri(X) => Set the execution priority in the operating system  
 Qi => Returns the Quaternion i = {1; 0; 0; 0}  
 Qj => Returns the Quaternion j = {0; 1; 0; 0}  
 Qk => Returns the Quaternion k = {0; 0; 1; 0}  
 QId => Returns the Quaternion I = {0; 0; 0; 1}, identity Q  
 Quiet(X) => Set the quiet mode on or off, (X) is optional  
 Ran => Randomly start a new random number sequence  
 ReadN(F) => Read number from file, F = "ccc...c" optional  
 Restore => Restore Configuration, History, & List  
 Run(F) => Run XMCalc code from file F, F is optional  
 RunH(X) => Set Height of Run form in pixels  
 RunW(X) => Set Width of Run form in pixels  
 Save => Save Configuration, History, & List  
 SaveTop(X) => Set "save top value in LastTop" on or off  
 ScieN(X) => Force scientific notation on iff X != 0  
 ScientificN(X) => Force scientific notation on iff X != 0  
 SetC(X) => Set max commands in history  
 SetD(X) => Set max decimal digits in display  
 SetM(X) => Set digits in Mantissa  
 SetMax(X) => Set max decimal digits allowed in mantissa  
 Time => Set timing mode on without other diags  
 Ubiq => Returns the ubiquitous constant U = 0.84721,30847,9...  
 Write(X) => Output X, (X may be "ccc...c", X is optional)  
 WriteLn(X) => Write(X) and a line feed  
 WriteN(F) => Write X to file F = "ccc...c", F is optional  
 XMCIIn(F) => Enter file name F = "ccc...c" for J command  
 XMLOut(F) => Enter file name F = "ccc...c" for H command  
 XMNIIn(F) => Enter file name F = "ccc...c" for I command  
 XMNOOut(F) => Enter file name F = "ccc...c" for W command

Note: Top is the item currently on top of the named number list.

---

The functions supported are:

Abs(X) = AbsoluteValue(X) = |X|  
 ACos(X) = ArcCoSine(X)  
 ACosh(X) = ArcHyperbolicCoSine(X)  
 ACot(X) = ArcCoTangent(X)

ACoth(X) = ArcHyperbolicCoTangent(X)  
 ACsc(X) = ArcCoSecant(X)  
 ACsch(X) = ArcHyperbolicCoSecant(X)  
 Adj(M) = Adjoint matrix of square matrix M  
 AFib(X) = ArcFibonacciNumber(|X|), an integer  
 AGM(X, Y) = Arithmetic Geometric Mean = MeanC(X, Y)  
 ASec(X) = ArcSecant(X)  
 ASech(X) = ArcHyperbolicSecant(X)  
 ASin(X) = ArcSin(X)  
 ASinh(X) = ArcHyperbolicSine(X)  
 ATan(X) = ArcTangent(X)  
 ATan2(Y, X) = ArcTangent(Y over X)  
 ATanh(X) = ArcHyperbolicTangent(X)  
 Balance(M) = Return a balanced matrix with same eigenvalues  
 Bern(X) = Bernoulli number B(Int(X))  
 BernD(X) = Denominator of Bernoulli number B(Int(X))  
 BernDL(X) = Denominator of Large Bernoulli number B(Int(X))  
 BernG(X) = Generalized Bernoulli number B(X)  
 BernN(X) = Numerator of Bernoulli number B(Int(X))  
 BernNL(X) = Numerator of Large Bernoulli number B(Int(X))  
 Beta(X) = Dirichlet beta function  
 BetaC(X, Y) = The complete beta function  
 Bino(X, Y) = Binomial coefficient (X, Y), generalized  
 BinoS(X, Y) = Binomial coefficient (X, Y) by standard method  
 Ceiling(X) = Least integer  $\geq$  X  
 CharPoly(M) = Characteristic polynomial of square matrix M  
 Cheby(X) = Chebyshev polynomial of Int(X)  
 Chin(X, Y) = add to Chinese remainder problem  $z \equiv X \pmod{Y}$   
 Chin1(X, Y) = Initialize Chinese remainder with X1, Y1  
 CompM(P) = Companion matrix of polynomial P  
 Concat(X, Y) = Concatenate the columns of matrix X and Y  
 ConcatR(X, Y) = Concatenate the rows of matrix X and Y  
 ContFrac(X) = Continued fraction expansion of x  
 ContFrac(X, Y) = Continued fraction expansion of x with y terms  
 ContFracV(X) = Value of continued fraction x  
 Cos(X) = CoSine(X)  
 Cosh(X) = HyperbolicCoSine(X)  
 Cot(X) = CoTangent(X)  
 Coth(X) = HyperbolicCoTangent(X)  
 Csc(X) = CoSecant(X)  
 Csch(X) = HyperbolicCoSecant(X)  
 CuRt(X) = CubeRoot(X) =  $X^{(1/3)}$   
 Decomp(M) = The LU decomposition of matrix M  
 Del(M, C) = Delete column C of matrix M  
 DelR(M, R) = Delete row R of matrix M  
 Deriv(P) = Derivative of polynomial P  
 Det(M) = Determinate of matrix M  
 Diagonal(V) = Diagonal matrix from a column or row vector V  
 Dig(X, Y) = Number of base Y digits in X,  $Y \geq 2$   
 DigD(X) = Number of decimal digits in X  
 Dilog(X) = Dilogarithm(X)  
 DivInt(X, Y) = Floor(X/Y), integer divide, element by element  
 DivRem(X, Y) = Floor(X/Y) and set Re to remainder, element wise  
 E1(X) = Exponential integral function one  
 Ei(X) = Exponential integral  $Ei(x) = -E1(-x)$   
 EqQ(Q) = Quaternion equivalent to quaternion Q  
 Erf(X) = Error function  
 ErfC(X) = Complementary error function  
 Eta(X) = Dirichlet eta function  
 Euler(X) = Euler number E(X)  
 EulerG(X) = Generalized Euler number E(X)  
 EulerN(X) = Numerator of E(X) = E(X)  
 Exp(X) = eToThePower(X)

ExpL(X) = eToThePower(X) - 1  
 Extract(M, C) = Extract column C from matrix M  
 ExtractR(M, R) = Extract row R from matrix M  
 Fac(X) = Factorial of Int(X)  
 Fac2(X, Y) = Fac(Int(X)) / Fac(Int(Y)) by binary splitting  
 FacM(X) = (Factorial of Int(X)) Mod FMB  
 FacM2(X, Y) = Fac(Int(X)) / Fac(Int(Y)) Mod FMB by binary split  
 FacMS(X) = (Factorial of X) Mod FMB by standard method  
 FacS(X) = Factorial of Int(X) by standard method  
 Fib(X) = FibonacciNumber(X), Fib(0) = 0  
 Floor(X) = Greatest integer  $\leq$  X  
 Frac(X) = FractionalPart(X)  
 Gam(X) = GammaFunction(X) = (X-1)!  
 Gam1(X) = 16-digit GammaFunction(X)  
 GamL(A, X) = Lower incomplete gamma function  
 GamP(A, X) = lower regularized incomplete gamma function  
 GamQ(A, X) = Upper regularized incomplete gamma function  
 GamU(A, X) = Upper incomplete gamma function  
 GCD(X, Y) = Greatest Common Divisor X, Y  
 GCDe(X, Y) = Extended GCD(X, Y) = X\*X1 + Y\*Y1  
 Get(M, R, C) = Returns element at row R, column C of matrix M  
 Hess(M) = Return upper Hessenberg matrix, same eigenvalues  
 Hilbert(X) = Equals an X by X Hilbert matrix  
 Image(M) = Returns a basis for the image/range of matrix M  
 Insert(M, V, C) = Insert column vector V at column C of matrix M  
 InsertR(M, V, R) = Insert row vector V at row R of matrix M  
 Int(X) = IntegerPart(X)  
 Integ(P) = Formal integral of polynomial P  
 Inv(X) = 1 / X  
 Inv(X, Y) = Z = Inverse of X Mod Y, X\*Z == 1 Mod Y  
 InvQ(Q) = Inverse quaternion of quaternion Q  
 IsDiag(M) = Returns 1 if M is a diagonal matrix, else 0  
 IsFib(X) = 1 (True) if X is a Fibonacci number, else 0  
 IsFib2(X) = IsFib(X) by second method  
 IsSq(X) = 1 (True) if X is a square, else 0  
 Kron(X, Y) = Kronecker-Legendre symbol X over Y  
 Lam(X) = Dirichlet lambda function  
 LCM(X, Y) = Least Common Multiple X, Y  
 Legen(X) = Legendre polynomial of order Int(X)  
 Lerch(X, S, A) = LerchPhi(X, S, A), preferred method  
 Lerch1(X, S, A) = 16-digit LerchPhi(X, S, A)  
 Lerch2(X, S, A) = LerchPhi(X, S, A) by simple sum  
 LerchT(X, S, A) = LerchPhiT(X, S, A), traditional  
 LerchT1(X, S, A) = 16-digit LerchPhiT(X, S, A)  
 LerchT2(X, S, A) = LerchPhiT(X, S, A) by simple sum  
 Li(X) = Logarithmic integral = Ei(Ln(X))  
 Ln(X) = NaturalLog(X)  
 LnL(X) = NaturalLog(X + 1)  
 Log(X) = LogBase10(X)  
 Lop(X) = ReducePrecision(X)  
 Mag(X, Y) = SqRt(Sq(X), Sq(Y))  
 Mat(R, C) = Return a zero matrix with R rows and C columns  
 MatId(X) = Equals an X by X identity matrix  
 Max(X, Y) = Greater of X and Y  
 MeanC(X, Y) = Common mean of X and Y = AGM(X, Y)  
 MEq(X) = Mersenne equation =  $2^X - 1$   
 Min(X, Y) = Lesser of X and Y  
 Mod(X, Y) = X - (Floor(X/Y) \* Y)  
 Mord(A, N) = Multiplicative order of base A (mod N) or 0  
 MPG(X) = The X'th Mersenne Prime Generator, MPG(1) = 3  
 MPP(X) = Mersenne Prime Power, MPP(1) = 2  
 MPrime(X) = 1 (True) if  $2^X - 1$  is a Mersenne Prime else 0  
 MToQ(M) = Quaternion equivalent of 3 by 3 matrix M



MToR(M) = Rotation vector from 3 by 3 matrix M  
 Mu(X) = Moebius Mu(X) function  
 MulQ(X, Y) = Quaternion multiply, X \* Y  
 NormC(X, M, S) = Normal Cumulative distribution function (cdf)  
 NormQ(Q) = Quaternion Q normalized  
 NormR(R) = Rotation vector R normalized  
 NormS(X) = Standard Normal Cumulative distribution function  
 NormV(V) = Norm of vector V, its length  
 P(X) = The X'th prime  
 Pascal(X) = Equals an X by X Pascal triangle,  $P_{ij} = \text{Bino}(i, j)$   
 PEq(X) = Perfect equation =  $(2^X - 1) * 2^{(X-1)}$   
 PGT(X) = First prime > X  
 Phi(X) = Euler's totient function  
 PhiL(X, A) = Legendre's formula  
 Pi(X) = Number of primes  $\leq X$  by sieve or Lehmer  
 PiL(X) = number of primes  $\leq X$  by Lehmer's formula  
 PiL1(X) = number of primes  $\leq X$  by Legendre's formula  
 PiM(X) = number of primes  $\leq X$  by Meissel's formula  
 PLT(X) = Largest prime < X  
 PNG(X) = The X'th Perfect Number Generator, PNG(1) = 6  
 PolAdd(X, Y) = Polynomial X + Y  
 PolDisc(P) = Discriminant of polynomial P  
 PolDiv(X, Y) = Polynomial X / Y, "Re" = remainder  
 PolEval(P, X) = Polynomial P evaluated at X  
 PolMod(X, Y) = Returns "Re" = remainder of polynomial X / Y  
 PolMonic(P) = Monic polynomial of P, leading coeff. = 1 or {0}  
 PolMul(X, Y) = Polynomial X \* Y  
 PolNorm(P) = Norm of polynomial P, leading coeff. not 0 or {0}  
 PolRecip(P) = Reciprocal polynomial of P,  $x^{\text{deg}(P)} * P(1/x)$   
 PolSturm(P) = Number of real roots in real polynomial P  
 PolSturmI(P, X, Y) = Number of real roots in P in interval (X, Y)  
 PolSub(X, Y) = Polynomial X - Y  
 Polylog(S, X) = Polylogarithm(S, X)  
 Pow(X, Y) = X to the Y  
 PowM(X, Y) = (X to the Y) Mod FMB  
 Prime(X) = 1 (True) if X is Prime else 0  
 Primo(X) = Primorial, product of all primes  $\leq X$   
 PrimR(X) = Largest and smallest primitive root of X or 0  
 PrimRP(X) = Largest and smallest prime primitive root of X  
 PrimRQ(X, Y) = 1 (True) if X is a primitive root of Y, else 0  
 PrinQ(Q) = Principal equivalent quaternion of quaternion Q  
 Psi(X) = DigammaFunction(X)  
 QToM(Q) = Matrix equivalent of quaternion Q  
 QToR(Q) = Rotation vector equivalent of quaternion Q  
 QuadR(a, b, c) = Roots of  $a*x^2 + b*x + c = 0$ , sets X1 and X2  
 Rank(M) = Rank of matrix M  
 Rev(X) = Digit Reversal of X base 10  
 Rev(X, Y) = Digit Reversal of X base Y  
 Ri(X) = Riemann prime counting function  
 RInt(X, Y) = RandomInteger between Int(X) and Int(Y) inclusive  
 RN(X) = RandomNumber(X=Seed or matrix)  
 RotateM(M, V) = 3-D vector V rotated by matrix M  
 RotateQ(Q, V) = 3-D vector V rotated by quaternion Q  
 Round(X) = Integer nearest to X  
 RPYM(R, P, Y) = Rotation matrix, R = roll, P = pitch, Y = yaw  
 RREF(M) = Return Reduced Row Echelon Form of matrix M  
 RREFa(M, V) = RREF solution U to augmented matrix [M,V],  $M*U=V$   
 RToM(R) = 3 by 3 matrix from rotation vector R  
 RToQ(R) = Quaternion from rotation vector R  
 Sec(X) = Secant(X)  
 Sech(X) = HyperbolicSecant(X)  
 Set(M, R, C, Y) = Return M with element at row R, column C set to Y  
 SetAll(M, Y) = Return matrix M with all elements set to Y

```

    Sig(X) = Sum of divisors of X
    Sig0(X) = Sum of divisors of X (-X)
    Sign(X) = 0 if X=0, else = X / |X|
    Sin(X) = Sine(X)
    Sinh(X) = HyperbolicSine(X)
    Size(X) = {r, c} where r = rows in X, c = columns in X
    Solve(M, V) = Solve for vector U where square matrix M * U = V
    Solve(X, Y, N) = Solve for z, X * z == Y Mod N
    SolveH(M) = Solve for vectors U != 0 where matrix M * U = 0
    SolveS(M, V) = Solve for minimum vector U where matrix M * U = 0
    Sord(A, N) = Multiplicative suborder of base A (mod N) or 0
    SortC(X) = Sort each column of matrix X in numerical order
    SortCS(X, Y, Z) = Sort each column of matrix X, row Y thru Z
    SortR(X) = Sort each row of matrix X in numerical order
    SortRS(X, Y, Z) = Sort each row of matrix X, column Y thru Z
    Sq(X) = X Squared
    SqFree(X) = 1 (True) if X is a squarefree, else 0
    SqRt(X) = SquareRoot(X)
    SqRtRem(X) = Floor(SquareRoot(X)) and set Re to remainder
    SumD(X, Y) = Sum of base Y digits in X, Y >= 2
    SumDD(X) = Sum of decimal digits in X
    SVD(M) = Singular Value Decomposition, M = u * w * v^T
    Tan(X) = Tangent(X)
    Tanh(X) = HyperbolicTangent(X)
    Tau(X) = Number of divisors of X
    ToDeg(X) = RadiansToDegrees(X)
    ToRad(X) = DegreesToRadians(X)
    Trace(M) = Sum of elements on the principal diagonal of M
    Tran(M) = Transpose of matrix M
    Vander(V) = Vandermonde's matrix from vector V
    Zeta(X) = Riemann zeta function
    ZetaH(S, A) = Hurwitz zeta function
    {M11, M12, ...; ..., Mrc} = Enter a matrix with r rows and c columns
    -X = Negative of X, 0 - X
    +X = Positive of X, 0 + X
    !X = Not X, 0 -> 1 else 0

```

Note that the ' character can be used to separate the arguments in functions since commas are allowed in numeric input, ", " works also.

---

The constant identifiers supported are:

```

Cat = Catalan's constant G = 0.91596,55941,77219...
Ee = Exp(1) = e = 2.71828,18284,59045...
EulerC = Euler's constant gamma = -Psi(1) = 0.57721,56649,01532...
Ln10 = The natural log of 10 = 2.30258,50929,94045...
Phi = Golden Ratio = (1 + SqRt(5)) / 2 = 1.61803,39887,49894...
PhiP = Phi prime = (1 - SqRt(5)) / 2 = -0.61803,39887,49894...
Pi = Archimedes' Constant Pi = 3.14159,26535,89793...
Qi = Quaternion i = {1; 0; 0; 0}
Qj = Quaternion j = {0; 1; 0; 0}
Qk = Quaternion k = {0; 0; 1; 0}
QId = Quaternion I = {0; 0; 0; 1}, identity quaternion
Ubiq = The ubiquitous constant U = 0.84721,30847,93979...

```

Note that these constant identifiers can be used in equations. If the constant is already on the list, the list item is used. If it is not on the, it is generated and put on the list.

---

```

+-----Function Keys on Run form-----+
| F1 => Display Help form (?) |

```

```

| F2 => Totally Quit/end the program (Q) |
| F3 => Restore previous input command |
| F4 => Restore previous input and accept |
| F5 => Get Status of calculation |
| F6 => Display Configuration form |
| F7 => Display Restore Input History form |
| F8 => Accept input and Calculate |
| F9 => Toggle Logging to Log file on/off (H) |
| F11 => Clear output text box |
| F12 => Pause (Pause) |
| Ctrl+F2 => Restart ($) |
| Ctrl+F9 => Clear Log File (ClearLog) |
| Ctrl+F11 => Clear input text box |
| Ctrl+S => Save All (Save) |
| Ctrl+O => Restore All (Restore) |
| ESC => Clear Run form Input Text Box |
| ESC => Interrupt/Abort a long calculation |
| PgUp => Display previous newer command |
| PgDn => Display previous older command |
+-----+

```

---

Commands used in XMCalc code files:

```

If {expression} Then {statements} Else {statements}
GoTo {label} (label is a name without a colon)
GoUpTo {label}
Labels: A name followed by a colon (:)
Continuation lines ending with + or -
Batch Commands (Echo, @Echo, Pause, and Rem)
Echo On/Off
@Echo On/Off
Pause
Rem ... or //... (for remarks)

```

---

Primitive op codes -

Primitives that act on a number, act on the currently active item. In the following description of primitives, the currently active item is called x for convenience.

A => Auto Display on/off:

Normally, after each command line is executed, the name and numerical value of the currently active item on the list is displayed. When computing with numbers with many significant digits, the time spent in producing this display can be excessively large. It is desirable then to be able to prevent this automatic display. Each time the A command is given the selection status of this option is reversed (toggled).

A word about the displayed value is in order. As an example, if the first command line you enter after starting the program is 3o, the response will be:

```
X = 3.33333,33333,33333,33333,33333,33333,33333,33333,33333,33333,33E-1 (48) [56]
```

The E-1 means that  $X = 3.3\dots$  times 10 to the minus one, the 48 in parentheses means there are 48 decimal digits displayed, and the 56 in brackets means that X is stored in memory with 56 decimal digits of precision. Leading zeros in the most significant super-digit and trailing zeros in the least significant super-digit are not counted.

Internally the mantissa is considered as an integer and the exponent or characteristic is a count of super-digits to determine the decimal point (actually the base  $10^8$  point). The characteristic is carried as a 64-bit signed integer. This allows for very large exponents, for example, numbers near the limit are:

Big = 1.0E+73786,97629,48382,06463 [1]

and

Small = 1.0E-73786,97629,48382,06456 [1]

B => Display learn line:

The calculator can contains a learned line, see the X primitive to enter and execute the learned line. The B command displays the current contents of the learned line. After the B command is executed, the F3 and F4 keys will restore the input line to the contents of the learned line instead of the previously typed command line. The B command enters the learned line command into the top of the history list of commands.

C => Change sign of x:

This is the same as multiplying x by minus one. Negative numbers can be entered by preceding them with a minus sign. The x referenced here is the current active item on the list of variables.

D => Set degree trig mode (nominal):

The trigonometric functions, Sin(X), Cos(X), ASin(X), ACos(X), Tan(X), ATan(X), ATan2(Y, X), etc. normally assume the angle involved in either the input or output is expressed in degrees. If radians are desired, use the E command. When degrees are desired, use the D command. The degree trig mode stays selected until changed by the E command.

E => Set radian trig mode:

The trigonometric functions, Sin(X), Cos(X), ASin(X), ACos(X), Tan(X), ATan(X), ATan2(Y, X), etc. normally assume the angle involved in either the input or output is expressed in degrees. If radians are desired, use the E command. When degrees are desired, use the D command. The radian trig mode stays selected until changed by the D command.

F => ! => Factorial:

Replaces x with the factorial of  $x = 1 * 2 * 3 * \dots * x$ . Only the integer portion of x is used in the calculation.

G => Set Digits/Group:

The G command will set the number of digits per group to the current value of x. If this is set to 3, numbers will be displayed with a comma after every 3rd digit like 1.234,567,89 E+34,457. If this is set to less than 1, no commas will be displayed. Only the integer portion of x is used.

H => Log screen to log file toggle:

The H command causes all output to the screen to be logged to a disk file. See the XMLOut(F) procedure for specifying a file name for this purpose. Each time the H command is given the selection status of this option is reversed/toggled.

The output text box on the Run form is referred to as the output screen or merely as the screen when the context is clear.

I => Input number from file:

The I command will use the last entered comment as a file name and read this file as an XMCalc formatted number and assign it to the current active item. It is assumed that the file was created by the W command or the WriteN procedure.

See the W command for the format of file names. If a comment has not been entered, the file name NoName.XMN is used. The XMNIn procedure can be used to give an override file name. File names can also be assigned using the Configuration form.

The files input by the I command are assumed to have all ASCII text characters with a numerical value less than 128. See the ReadN procedure. The DOS VPCalc program deleted the upper bit of characters greater than 127.

J => Run XMCalc code from file:

The J command will use the last entered comment as a file name and read this file as a text file. Each line of the file will be interpreted as an XMCalc command line and executed. If comment commands and J commands exist in the text file, these other referenced files will be opened and processed. The only limitation to this nesting of code files is the availability of memory and buffers. If a comment has not been entered, the file name NoName.XMC is used. The XMCIn procedure can be used to give an override file name for the J command. The files input by the J command are assumed to have all ASCII text characters with a numerical value less than 128.

K => Execute learn line x times:

This will cause the learned line to be executed x time. x must be in the range  $0 <= x <= 2,147,483,647 (2^{31} - 1)$ . The x referenced here is the current active item on the list of variables. If x is larger than this max, then the max will be used. A long repetition of a learned line can always be interrupted by using the ESC key or selecting the Abort Calc. button on the Run form.

L => Reduce precision of x:

This command removes or Lops off the least significant super-digit of x. If rounding is turned on, the removed super-digit is used to round into the new least significant super-digit. In XMCalc numbers are normalized from both sides. If a calculation results in a number with some trailing zero super-digits, these super-digits are removed, the count of the number of super-digits in the mantissa is reduced and memory is reallocated. The L command can result in many super-digits being removed if removing one super-digit results in many trailing zeros.

For matrices some or all of the elements will have their precision reduced depending on the position of their least significant super digit.

M => Set digits in Mantissa:

The M command will set the current value of the maximum number of decimal digits allowed in a floating-point number to the current value of x. If there are items on the list containing more than this number of digits, they will be reduced to contain at most this number of digits. If x is not a multiple of 8, when the M command is given, then the next higher multiple of 8 is used. If x is less than 16, it is set to 16. Only the integer portion of x is used.

Some messages output by the calculator contain the name FMC. For example, the message "Error in Pi, FMC = 125" would be given if you were running at 1000 decimal digits of precision and the value of Pi stored in file "Pi.XPN" had less than 125 super-digits. FMC is the current max number of super-digits in a floating-point number. The Pi.XPN file delivered with XMCalc has 1048772 decimal digits.

N => Generate a random number:

The N command generates a random number between zero and 1.0 and assigns it to the current active item on the list. This number will never have more than 35 significant decimal digits. Theoretically the random number generator will cycle after  $10^{35}$  numbers, but the earth will not last that long. The items RN, RNA, and RNC are put on the list by the random number command. The equation used is:  $x = RN = (RNA * RN * 10^{35} + RNC) \bmod (10^{35}) / 10^{35}$ , where RNA and RNC are 35 digit integers.

O =>  $x = 1 / x$ :

Replace x with 1.0 divided by x, error if  $x = 0$ .

P => Return Pi = 3.14159,26535,89793...:

If Pi is on the list, then  $x = Pi$ . If Pi is not on the list, the file Pi.XPN is read-in, Pi added to the list, and  $x = Pi$ . If the file Pi.XPN is not found, Pi is computed by algorithm b. Algorithm b is documented in Scientific American, Feb 1988, Ramanujan and Pi, by Jonathan M. Borwein and Peter B. Borwein.

Pi = 3.14159,26535,89793,23846,26433,83279,50288,41972E+0 (41) [73]

Q => To totally Quit/end the program:

The program exits after displaying a message block to allow the operator to OK or Cancel the request.

R => Square root of x:

x is replaced with the positive square root of x, error if  $x < 0$ .

S => Square x:

x is replaced with the square of x.

T => Set digits to truncate:

The T command will set the number of decimal digits to truncate for display to the current value of x. The calculator is initialized with this set to 8 decimal digits. Only the integer portion of x is used.

U => Set rounding mode:

This command sets rounding on. When rounding is on, the results of all numerical operations are rounded to the maximum number of super-digits in mantissa. When rounding is off, these results are truncated to the maximum number of super-digits in mantissa. Use the M command to set the maximum number of super-digits in mantissa. The IEEE standard of round to even is used, e.g., all numbers in the closed interval [11.5, 12.5] round to 12.

V => Set non-rounding mode:

This command sets rounding off. When rounding is off, the results of all numerical operations are truncated to the maximum number of super-digits in mantissa. Use the M command to set the maximum number of super-digits in mantissa.

W => Write x (Top number) to file:

The W command will use the last entered comment as a file name and write register x into this file as an XMCalc formatted number. This number can be reread into x by the I command or ReadN procedure. If the file already exists, it will be erased and recreated. For example, the following are valid file names:

"File.Ext" File.Ext is on default drive and directory, "B:FileName.Ext"  
FileName.Ext is on B: drive, current B: directory, "P.XMN" P.XMN is on default  
drive and directory, "C:\Direct\File.Ext" File.Ext is on C: drive, Direct  
directory

If the file name does not have a period, the extension .XMN is added. If a comment has not been entered, the file name NoName.XMN is used. The XMNOut procedure can be used to give an override file name for the W command. The file written is a text file and can easily be browsed and read by other programs. The content of the file for 1/7 is:

```

                                {3 blank lines}
OneOver7 = m.n E-1, m.n =

1.
42857 14285 71428 57142 85714 28571 42857 14285 71428 57142
85714
E-1 (56)
                                {51 blank lines}
                                Page 1
                                {4 blank lines}
```

The content of the file for a base 36 random number is:

```

{1 blank line}
Base(36)

X = m.n -1, m.n =

0V.
31TAE,KCAHW,QVE5N,NTT26,K0Q2O,6RELM,7LM6C
-1 (36 Digits Base 36)
```

A base 36 number not in scientific notation looks like this:

```
{1 blank line}
Base(36)
```

X =

```
13DVQ,MR26F.ZZZZZ,ZZZZZ,MU5TS,106Y2,LCT3A,1F
(10.27 Digits Base 36)
```

Note, the back-tick character (`) is used as a prefix to exponents for the scientific notation format of numbers in a base larger than ten. The character E is used if base is less than 11. For input with base less than 11, the characters E, e, and ` can be used interchangeably.

X => Learn, Execute (X learn, X):

If this is the last command on a command line, then it caused the learned line to be executed once. If not the last command on the line, this command stores all the commands following on the same line or text box as this one into the learned line. Execution of the current line is stopped. Type the learned line:

```
N=0 Fact=1 X N=N+1 Fact=Fact*N Z X
```

and then do an X commands. You might want to key in an H command before the X command to turn on logging. This will print a table of factorials from 2! to (1.70854E+9)! or so, if you wait long enough. Hit the ESC key to interrupt and abort the operation if you get tired of waiting. After the X command is executed, the F3 and F4 keys will restore the input line to the contents of the learned line instead of the previously typed command line.

Y => Delete (Yank) number from list:

The Y command removes the currently active item from the list and makes the next older item the active item. The age of an item is judged by when it was created. X is always the oldest item and is never removed from the list. If the Y command is executed, when X is the active item, X is not removed, but the youngest item becomes the active item. Thus, a long string of Y commands will always remove all items from the list except X.

Z => Output the list of variables:

The Z command will display the name and value of all items on the list. Some items may be found on the list that were not explicitly put there. The items RN, RNA, and RNC are put on the list by the random number command N and the random number function RN(X). The item File: "comment" is put on the list by the "comment" command. The item Lrn: {learned line} is put on the list by the X command. The items File: "comment" and the item Lrn: {learned line} also have a value associated with them, normally = 0. This value has no meaning and is not used. The item Re is put on the list by the DivRem function.

If diags are turned on by Diag(1) when the Z command is executed, extra items are displayed. For example the output:

```
1: X = 0 N=1 M=1 U=1
```

says that item 1: on the list is X = 0, it has N=1 super-digits, memory for M=1 super-digits, and the upper bound of the array for its super-digits U=1.

" => Start/Stop file name or comment:



Comments can be entered anywhere on the command line. The comment is started with a " mark. The comment is ended with a " mark or the end of the line. All blank spaces between the " marks become part of the comment. Comments are also used as file names; see the XMCIIn, XMNIIn, XMNOOut, and XMLOut procedures. The item File: {comment} is put on the list by this "{comment}" command.

% => Set FMB = x, FMB on list:

The % primitive command is equivalent to  $FMB = x$ , where  $x$  is the currently active item. FMB stands for Floating Modulo Base. The / primitive command and the PowM(X, Y) function use FMB from the list. Normally  $FMB > 0$ , but it can be  $< 0$  to give negative residuals.

All modulo and integer division operations are performed to make the remainder of the division to have the same sign as the divisor or be zero. For  $q = x \setminus y$  with remainder  $r$  ( $y \neq 0$ ),  $0 \leq |r| < |y|$  and  $\text{sign}(r) = \text{sign}(y)$  or  $r = 0$ , and  $x = q*y + r$ . For example:

```
14.3\5.2 = 2, r = 3.9,
14.3\(-5.2) = -3, r = -1.3,
(-14.3)\5.2 = -3, r = 1.3,
(-14.3)\(-5.2) = 2, r = -3.9 .
```

/ =>  $x = x \text{ Mod } FMB$ , FMB on list:

The / primitive command replaces  $x$  with  $x$  modulo FMB, where  $x$  is the currently active item and FMB is an item on the list. If FMB is not on the list, it is added to the list with a value of zero. If FMB is zero, the value of  $x$  is not changed. See the % primitive.

\ => Set Input and Output base to x, [2, 36]:

The primitive command \ sets both the input and output base to  $x$ . This is the same as the Base(X) procedure described more fully below. Only the integer portion of  $x$  is used.

\$ => Restart/Reset configuration:

This reinitializes the program, the same as reloading except total running time is not reset, the history of previous operator entries is not cleared, and the log screen to log file state is not changed. The parameters set by the D, E, M, T, U, and V commands are reset to their nominal values, and all items on the list are deleted except X and it is cleared. This also reset the random number generator and FMB is effectively zero.

> => Write configuration to file Config.XMC:

The file Config.XMC is written to disk. It contains the XMCalc commands that will restore the configuration of XMCalc to its current state.

An example of the contents of a Config.XMC is:

```
@Echo Off
Rem Start of file Config.XMC
BaseI(0)
SetMax(134218400)
LastTop=; 54525952 M; 8 T; 5 G; U; D
SetC(1000)
```

```

SetD(1001)
SaveTop(1)
AllowFHT(1)
ForceFHT(0)
Diag(0)
Time
AutoDisplay(1)
ChDir("C:\ProgramD\VC#\Harry\XMCalc\bin\Debug")
ScientificN(0)
XMCIIn("CODE.XMC")
XMLOut("Test4.XML")
XMNIIn("Test3.XMN")
XMNOOut("NoName.XMN")
LogScreen(1)
BaseO(8)
BaseI(16)
X=
Quiet(1)
Pri(-1)
Rem End of file Config.XMC
Echo On

```

< => Read configuration from file Config.XMC:

The file Config.XMC is read and run as an XMCalc code file. This will restore the configuration of XMCalc to its configuration when the file was written by the > command.

] => Write entry command history to file XMCalcHist.txt:

The file XMCalcHist.txt is written to disk. This is a text file and contains a copy of the current history of operator entries.

[ => Read entry command history from file XMCalcHist.txt:

The file XMCalcHist.txt is read and used to restore the history of operator entries with the history when the file was written by the ] command. The current history is not cleared, but some or all of it may be lost since only "Max Commands in History" entries are saved. Duplicate commands are deleted as the new copy is entered.

. => Write x in fixed-point decimal:

Write X (Top item) in fixed-point decimal.

# => Primorial (like factorial):

Replaces x with the primorial of x. See the Primo(X) function.

? => Display Help form as presented above.

Infix operators -

Infix operators +, -, \*, /, ^, @, #, %, \, &, |, <, >, =, ==, <=, !=, <>, >=, --, and \*\* are the operators that appear between operands in an expression. Infix operators do not change the value of their operands, but produce a single result

that can be used to further complete the evaluation of the expression that contains the infix operator. The infix operator precedence classes, from highest to lowest, are:

- 1) ^
- 2) \*, /, @, #, %, \, &, \*\*
- 3) +, -, |, --
- 4) <, >, =, ==, <=, !=, <>, >=

Operators of the same class are evaluated from left to right. Thus  $(2 * 10)^2 = 20^2$ , but  $2 * 10^2 = 2 * 100$ . Also,  $A + B * C = A + (B * C)$ .

$A = X + Y \Rightarrow$  Set A to X plus Y:

Addition operator. To add two matrices, X and Y must be of the same size, same number of rows and same number of columns,  $X.m = Y.m$  and  $X.n = Y.n$ . The sum A will also be that size of a matrix with each element of A being the sum of the corresponding elements of X and Y,  $a_{ij} = x_{ij} + y_{ij}$  for all ij with  $1 \leq i \leq X.m$ ,  $1 \leq j \leq X.n$ .

$A = X - Y \Rightarrow$  Set A to X minus Y:

Subtraction operator. For matrices, same as sum except  $a_{ij} = x_{ij} - y_{ij}$ .

$A = X * Y \Rightarrow$  Set A to X times Y:

Multiplication operator. To multiply two matrices,  $k = X.n$  must equal  $Y.m$ , and the product A will be an  $X.m$  by  $Y.n$  matrix,  $a_{ij} = x_{i1}y_{1j} + x_{i2}y_{2j} + \dots + x_{ik}y_{kj}$ .

If one of X or Y is a scalar and the other is a matrix, this is a scalar multiply. All of the elements of the matrix are multiplied by the scalar.

See: <http://mathworld.wolfram.com/MatrixMultiplication.html> from MathWorld.

$A = X / Y \Rightarrow$  Set A to X divided by Y:

Division operator, error if  $y = 0$ . To divide two matrices, they must be square matrices of the same size and the divisor Y must have a matrix inverse  $Inv(Y)$ . Then  $A = X * Inv(Y)$ .

If X is a matrix and Y is a scalar  $\neq 0$ , this is a scalar multiply,  $A = X * (1/Y)$ . If X is a scalar and Y is a square matrix that has an inverse, this is also a scalar multiply,  $A = X * Inv(Y)$ .

$A = X ^ Y \Rightarrow$  Set A to X to the power Y:

Exponential operator. This operator operates differently depending on whether y is an exact integer. If y is an exact integer, the peasants' method is used in which up to  $2 * \text{Log base 2 of } y$  multiplies of powers of x are done to compute the result. If y is not an exact integer, the result is computed by  $\text{Exp}(y * \text{Ln}(x))$ . An error message is generated in two cases: 1)  $x$  is  $< 0$  and y is not an integer. 2)  $x = 0$  and y is  $< 0$ . If  $x = 0$  and  $y = 0$ , an answer of 1.0 will be given. The ^ operator is evaluated from right to left:  $3^3^3 = 3^{27} = 762,55974,84987$ . My MS Dos program VPCalc evaluated it as  $(3^3)^3 = 27^3 = 19683$ , which is not the normal convention.

$A = Y @ X \Rightarrow$  Set A to  $ATan2(Y \text{ over } X)$ :

ArcTangent of y over x operator. Used to find the Polar coordinates angle coordinate of the Cartesian coordinates (x, y). If the degree mode is set, the answer, A, will be in the range  $-180 < a \leq 180$ . If the radian mode is set, the answer will be in the range  $-\pi < a \leq \pi$ . If both x and y are zero, an answer of zero will be given.

$A = X \# Y \Rightarrow$  Set A to  $Mag(X, Y) = \text{Sqrt}(\text{Sq}(X) + \text{Sq}(Y))$ :

Magnitude of (x, y) operator. Used to find the Polar coordinates radius coordinate of the Cartesian coordinates (x, y).

$A = X \% Y \Rightarrow$  Set A to  $\text{Mod}(X, Y) = X \text{ Modulo } Y$ :

Modulo operator.  $a = x \% y = x - (\text{Floor}(x/y) * y)$ . Where  $\text{Floor}(x/y)$  is the greatest integer  $\leq x/y$ . The sign(a) = sign(y) or a = 0,  $0 \leq |a| < |y|$ .  $a = 5 \% 3.5 = 1.5$ ,  $a = 5 \% (-3.5) = -2$ . An error message is generated if  $y = 0$ .

$A = X \setminus Y \Rightarrow$  Set A to  $\text{Floor}(X/Y)$ , integer divide

Integer divide,  $a = x \setminus y = \text{Floor}(x/y)$ . An error message is given if  $y = 0$ .  $x = a * y + \text{Re}$ . The remainder always has the same sign as y or equal to 0,  $0 \leq |\text{Re}| < |y|$ .

$A = X \& Y \Rightarrow$  Set A to 1 if X and Y are both not 0, else set A to 0:

Logical And operator. For all logical operations, 0 is considered False and all other values are considered True. When the result of a logical operation is True, the value 1 will be produced. When the result of a logical operation is False, the value 0 will be produced.

$A = X | Y \Rightarrow$  Set A to 1 if X or Y, is not 0, else set A to 0:

Logical Or operator.

$A = X < Y \Rightarrow$  Set A to 1 if  $X < Y$ , else set A to 0:

Numerical Less-than operator. For all numerical equivalence operators, the operands are considered as signed numbers and the result is either 1 (True) or 0 (False).

$A = X > Y \Rightarrow$  Set A to 1 if  $X > Y$ , else set A to 0:

Numerical Greater-than operator.

$A = X = Y \Rightarrow$  Set A to 1 if  $X = Y$ , else set A to 0, same as  $==$ :

Numerical Equal-to operator.

$A = X == Y \Rightarrow$  Set A to 1 if  $X = Y$ , else set A to 0:

Numerical Equal-to operator.

$A = X \leq Y \Rightarrow$  Set A to 1 if  $X \leq Y$ , else set A to 0:

Numerical Less-than-or-equal-to operator.

$A = X \neq Y \Rightarrow$  Set A to 1 if  $X \neq Y$ , else set A to 0, same as  $\lt;$ :

Numerical Not-equal-to operator.

$A = X \geq Y \Rightarrow$  Set A to 1 if  $X \geq Y$ , else set A to 0:

Numerical Greater-than-or-equal-to operator.

$A = X \text{ -- } Y \Rightarrow$  Set A to  $X * \text{Repunit}(Y)$ ,  $3\text{--}5 = 33333$ , five threes:

A repunit is a number consisting of copies of the single digit 1.  $\text{Repunit}(n)$  base B is  $(B^n - 1)/(B - 1)$  for base 10 this is  $(10^n - 1)/9$ . The output base (Base0) is used for this calculation. This allows you to have one value for the base of the input numbers and another value for the repunit base. The arguments x and y are not restricted to being integers,  $x\text{--}y = x * ((B^y - 1) / (B - 1))$ .

$A = X ** Y \Rightarrow$  Set A to  $\text{PolMul}(X, Y)$ :

Polynomial multiplication operator. Can multiply two polynomials or a polynomial and a scalar,

Assignment operators -

$= \Rightarrow$  Set Top to Top (no-op):

Same as  $\text{Top} = \text{Top}$ , a no-op.

$X = \Rightarrow$  Bring X to top of list:

The item X is made the current active item.

$= Y \Rightarrow$  Set Top to Y:

Same as  $\text{Top} = Y$ . Evaluate Y and store it on the list as Top, where Top is the current active item on top of the list.

$+ = Y \Rightarrow$  Set Top to Top plus Y:

Same as  $\text{Top} = \text{Top} + (Y)$ . Evaluate Y and add it to Top, where Top is the current active item on top of the list.

$- = Y \Rightarrow$  Set Top to Top minus Y:

Same as  $\text{Top} = \text{Top} - (Y)$ .

$* = Y \Rightarrow$  Set Top to Top times Y:

Same as  $Top = Top * (Y)$ .

$/= Y \Rightarrow$  Set Top to Top divided by Y:

Same as  $Top = Top / (Y)$ .

$\%= Y \Rightarrow$  Set Top to  $Mod(Top, Y) = Top$  Modulo Y:

Same as  $Top = Top \% (Y)$ .

$X = Y \Rightarrow$  Set X to Y:

Basic assignment operator, evaluate Y and store it on the list as X.

$X += Y \Rightarrow$  Set X to X plus Y:

Same as  $X = X + (Y)$ . Evaluate Y and add it to X.

$X -= Y \Rightarrow$  Set X to X minus Y:

Same as  $X = X - (Y)$ .

$X *= Y \Rightarrow$  Set X to X times Y:

Same as  $X = X * (Y)$ .

$X /= Y \Rightarrow$  Set X to X divided by Y and set Re to remainder:

Same as  $X = X / (Y)$ .

$X \% = Y \Rightarrow$  Set X to  $Mod(X, Y) = X$  Modulo Y:

Same as  $X = X \% (Y)$ .

Procedures -

Procedures are invoked by a statement starting with a procedure name followed by its argument. Arguments are numerical expressions that are evaluated before the procedure is performed. Procedures do not change the value of their arguments. For the procedures Write and WriteLn, arguments are optional and may be literal like: WriteLn("Now is the time"). For some procedures like Next, arguments are not allowed.

AllD(X)  $\Rightarrow$  Compute all divisors of X:

Uses the prime factor algorithm A to factor x and then compute all of the positive integral divisors of x. For example, if  $x = 12$  the output is:

All 6 Divisors: 1; 2; 4; 3; 6; 12.

The first divisor is always 1 and the last divisor is always x. The number of divisors is displayed in decimal but the divisors are displayed in the current output base.

AllowFHT(X) => Set Allow FHT multiple on if X != 0, else off:

The allow FHT mode is turned on if x != 0 and is turned off if x = 0. When this mode is on, long multiplications are speeded up by using the fast Hartley transform method to do the convolution. FHT is used if the numbers are greater than about 236 decimal digits. For about 100,000 digits numbers the FHT multiply runs in 0.1% of the time of a normal multiply (a factor of 1000). This mode is initially on.

For fast Hartley transform multiply  
fxt subroutines converted from C++ to C#  
including FHT Convolution with zero padded data  
by Harry J. Smith.

C++ author = Joerg Arndt           email: arndt@jjj.de  
the C++ software is online at   <http://www.jjj.de/>

----- \*\*\* LEGAL NOTICE: \*\*\* -----

This program is free software; you can redistribute it and/or modify  
it under the terms of the GNU General Public License (GPL) as published  
by the Free Software Foundation. cf. the file COPYING.txt.

----- \*\*\* end of legal notice \*\*\* -----

Also, when this mode is on, divides and square roots are speeded up by using Newton-Raphson iterations. For u/d, 1/d is computed first. No divides are performed except one at low precision to get the first guess. This is referred to as "divisionless divide".

$x = 1/d: x = (x + x) - (x * x) * d$

For SqRt(s), there is a long divide each iteration, but this is performed by the divisionless divide. The divide by 2 is a fast short division.

$x = \text{SqRt}(s): x = (x + s/x)/2$

In both cases, the precision used to calculate x is doubled with each iteration.

AutoDisplay(X) => Set Auto display on if X != 0, else off:

Same as the A primitive op code, but instead of being a toggle, sets Auto display on if x != 0, and sets it off if x = 0.

Base(X) => Set Input and Output base to X, [2, 36]:

There are two configuration items, BaseI and BaseO, that determine the numerical base of input numbers and output numbers. This command sets both of them to integer part of x. If x is less than 2, they are set to 10. If larger than 36, they are set to 36. The letter A through Z are used to represent digits larger than 9. A represents 10 decimal and Z represents 35 decimal. This gives hexadecimal numbers their normal representation. Only the integer portion of x is used.

All the numbers in input commands are affected by the input base, so the command Base(10) is always a no-op. To change the base to 10 decimal use the command Base(0). Digits in the exponent portion of a number (input or output) are always in decimal.

Not all output digits are displayed in BaseO, only the numerical value of the mantissa of numbers on the list. When BaseI is not 10 decimal, the command prompt will contain the input base expressed in decimal, and when BaseO is not 10 decimal, the displayed numbers will have the output base displayed in decimal. For example, if both BaseI and BaseO = 16 decimal, the input of X=0FF would get the response:

```
Command (Base 16): X=0FF
```

```
X = 0FF.0 (Base 16) [3]
```

The leading zero in the input 0FF was needed to distinguish it from the primitive F command. In general, if a number starts with a digit larger than 9, precede it with a zero (0).

BaseI(X) => Set Input base to X, [2, 36]:

This is like the Base(X) command, but only the Input base is affected. Only the integer portion of x is used.

BaseO(X) => Set Output base to X, [2, 36]:

This is like the Base(X) command, but only the Output base is affected. Only the integer portion of x is used.

BernM => Returns the Max index of saved Bernoulli numbers:

When Bernoulli numbers are generated by the BernD or BernN functions, the exact numerator and denominator of all even indexed Bernoulli numbers up to the maximum index generated are saved. This command returns this maximum index currently in storage.

Cat => Returns Catalan's constant  $G = 0.91596,55941,77219\dots$ :

Catalan's constant  $G = 1 - 1/9 + 1/25 - 1/49 + \dots = \text{Beta}(2)$ .

ChDir(F) => Change directory to F = "ccc...c", F optional:

Changes the directory used for commands H, I, J, W, ReadN(F), and Run(F). The original directory when the program starts is called the Home directory and is always used for the commands >, <, ], [, ?, ClearHist, ClearLog, LogScreen, Restore, and Save, commands. A ChDir without F will not change the directory, but will tell you how it is currently set.

It is probably easier to use the Configuration form "Change File Path" command button to change the disk directory. This procedure was included so it could be used by the Config.XMC code file for the >, <, Save, and Restore commands.

ClearBern => Clear storage of saved Bernoulli numbers.

ClearEuler => Clear storage of saved Euler numbers.

ClearHist => Clear history of previous operator entries:



The history of up to "Max Commands in History" previous operator entries are saved and can be retrieved by selecting the "Restore Input" button on the Run form. The ClearHist procedure removes all operator entries currently saved and makes this memory available to the calculator. Even though no argument is needed for this and some other procedures, it is usually better to use the parentheses, e.g., ClearHist() or ClearHist( to prevent unexpected results if the procedure name is misspelled.

ClearLog => Clear the log file:

If the log file is open, the file is closed and reopened. If it is currently closed, it opened and then closed. In either case it is cleared. Initially the log file name is NoName.XML.

The cleared log file will have up to three lines of data like:

```
Log file NoName.XML Cleared 1/3/2006 1:55:54 PM
XMCalc - Extra Precision Calculator, C# Version 3.2.2194.20607
Run on: Harry's Intel 3 GHz Pentium 4 - Dell DGV4T641 - Windows XM Pro SP2
```

The third line is generated by having something like:

```
SET SYSTEM=Harry's Intel 3 GHz Pentium 4 - Dell DGV4T641 - Windows XM Pro SP2
```

in your AutoExec.Bat file.

Diag(X) => Set diagnostic mode on or off:

The diagnostic mode is turned on if  $x \neq 0$  and is turned off if  $x = 0$ . When the diagnostic mode is on, all command line executions will be timed by the computer clock and the time spent executing the command will be displayed. The timing data is displayed as:

```
T = xxx.xx DT = xx.xx sec. Start execution
{command output, if any}
T = xxx.xx DT = xx.xx sec. End of execution
```

The DT value on the End of execution line is the time spent executing the command. The DT on the Start execution line is the time spent waiting for the operator to compose the command. The T values are the total running time since the program was started and can only be reset by terminating and reentering the program. The Quit button followed by the Run button will do it.

To turn diags on Diag(1) can be shortened to Diag.

Ee => Returns  $e = \text{Exp}(1) = 2.71828,18284,59045\dots$ :

$e$  is the base of the natural logarithms  $= 1 + 1/1! + 1/2! + 1/3! + \dots$

EulerC => Euler's constant  $\gamma = -\text{Psi}(1) = 0.57721,56649,01532\dots$ :

The asymptotic formula with Bernoulli numbers is used to compute this. See the  $\text{Psi}(X) = \text{DigammaFunction}(X)$ .

EulerM => Returns the Max index of saved Euler numbers:

When Euler numbers are generated, the exact integer value of all even indexed Euler numbers up to the maximum index generated are saved. This command returns this maximum index currently in storage.

Exit => Totally quit the program with no questions asked.

ForceFHT(X) => Set Force FHT multiple on if X != 0, else off:

The force FHT mode is turned on if x != 0 and is turned off if x = 0. When this mode is on, all multiplications are done using the fast Hartley transform method to do the convolution. This mode is initially off.

If ForceFHT is turned on when AllowFHT is off, AllowFHT is turned on also. If AllowFHT is turned off when ForceFHT is turned on, ForceFHT is turned off also.

GenBern(X) => Generate and save Bernoulli number upto B(X):

The exact numerator and denominator of all even indexed Bernoulli numbers up to the B(x) are generated and saved in computer storage. If x is less than 4, it is taken to be 4. If x is an odd positive integer, the next even integer is used. If x is very large, an error message is generated: "GenBern: n > 10000000, too large for Bernoulli number". But it would take "forever" for an x = 10,000,000. Only the integer portion of x is used.

GenEuler(X) => Generate and save Euler number upto E(X):

The exact integer value of all even indexed Euler numbers up to the E(x) are generated and saved in computer storage. If x is less than 4, it is taken to be 4. If x is an odd positive integer, the next even integer is used. If x is very large, an error message is generated: "GenEuler: n > 10000000, too large for Euler number". But it would take "forever" for an x = 10,000,000. Only the integer portion of x is used.

HelpH(X) => Set Height of Help form in pixels:

It is handy to put these commands in the AutoExec.XMC file. For example, the lines:

```
HelpW(674) HelpH(900)
RunW(674) RunH(950)
```

Will set the forms to larger than the default size for a 1280 by 1024 screen resolution. Only the integer portion of x is used.

HelpW(X) => Set Width of Help form in pixels. Only the integer portion of x is used.

HistH(X) => Set Height of History form in pixels.

HistW(X) => Set Width of History form in pixels.

Ln10 => Returns the natural log of 10 = 2.30258,50929,94045... .

LogScreen(X) => Log screen to Log file, on or off:

Same as the H primitive op code, but instead Of being a toggle, sets Log screen to Log file on if  $x \neq 0$ , and sets it off if  $x = 0$ . This procedure was called EchoScreen(X) in my VPCalc DOS program, and that command still works.

LX => LT => Restore LastTop to top of the list:

This sets the current active item equal to the value of the item named LastTop. If LastTop does not exist, it is created with a value of zero. Normally, before each command line is executed the value of the current active item is saved on the list in an item named LastTop. If a command line is entered that changes the value of the current active item, it can be restored to its previous value if the LX or LT procedure is performed immediately. This procedure should be entered as a command by itself to prevent LastTop from being changed before it is retrieved.

The value of the current active item, Top, is not saved in LastTop if: 1) The command line is: LX 2) The command line is: LT 3) The command line is: LastTop= 4) The SaveTop option is turned off by SaveTop(0).

Next => Move to next item on the list (no argument):

This changes which item on the list is the active item from the current active item to the next item from top to bottom. If X is the active item, which is always at the bottom of the list, the top item will become the active item. A command line with the single command Next followed by several F4 function keys will move through the whole list one item at a time. Note, this procedure does not take an argument.

Pause => Pause the calculations to free the processor. See function key F12:

Pause is actually a batch command. All commands on the line after Pause are ignored.

PFA(X) => Run prime factor algorithm A on X:

Computes the prime factorization of x using algorithm A. For a command of PFA(60) the output is:

$$60 = 2^2 * 3^1 * 5^1.$$

If the input base is 10 and the output base is 16, for a command of PFA( $2^{15} * 13^1$ ) the output is:

$$68000 \text{ (Base 16)} = 2^{15} * 0D^1.$$

The number and the prime divisors are displayed in the current output base. The exponents are always displayed in decimal

Algorithm A uses an implementation of a method described in Knuth, Vol. 2, Seminumerical Algorithms, Page 348 for the factorization.

PFB(X) => Run prime factor algorithm B on X:

Computes the prime factorization of x using algorithm B. The output is the same as PFA(X). Algorithm B uses Algorithm 357 of the Collected Algorithms from ACM, An Efficient Prime Number Generator using the sieve of Eratosthenes, to generate consecutive primes and find the prime divisors of x.

PFE(X) => Run prime factor algorithm ECM on X (fastest):

Computes the prime factorization of x using the Elliptic Curve Method (ECM) algorithm. The output is the same as PFA(X).

PTab(X) => Write prime table to XICalcPTab.txt, X primes:

Writes a table of x primes to file XICalcPTab.txt. If XICalcPTab.txt already exists it is renamed XICalcPTab.Bak before the new file is written. For example, a command of PTab(100) generates a file containing:

```
XICalc - Prime Number Table - 6/09/2004 11:45:20 AM
First column is prime index the rest are consecutive primes

1 2 3 5 7 11 13 17 19 23
10 29 31 37 41 43 47 53 59 61 67
20 71 73 79 83 89 97 101 103 107 109
30 113 127 131 137 139 149 151 157 163 167
40 173 179 181 191 193 197 199 211 223 227
50 229 233 239 241 251 257 263 269 271 277
60 281 283 293 307 311 313 317 331 337 347
70 349 353 359 367 373 379 383 389 397 401
80 409 419 421 431 433 439 443 449 457 461
90 463 467 479 487 491 499 503 509 521 523
100 541
End of file XICalcPTab.txt
```

All of the numbers in the table are always in decimal.

Phi => Returns the Golden Ratio =  $(1 + \text{Sqrt}(5)) / 2 = 1.61803,39887,49894\dots$ :

Phi (fee) and PhiP (fip, Phi prime) are the two solutions of the quadratic equation:

$$x^2 - x - 1 = 0.$$

Also,  $\text{Phi} = 2*\text{Cos}(36 \text{ degrees}) = 2*\text{Cos}(\text{Pi}/5) = 2*\text{Sin}(3*\text{Pi}) = -1/\text{PhiP}$ .

PhiP => Returns Phi prime =  $(1 - \text{Sqrt}(5)) / 2 = -0.61803,39887,49894\dots$ :

Also,  $\text{PhiP} = -2*\text{Cos}(72 \text{ degrees}) = -2*\text{Cos}(2*\text{Pi}/5) = -2*\text{Sin}(\text{Pi}/10) = -1/\text{Phi}$ .

The exact closed form solution for N'th Fibonacci number is called the Binet formula:

$$\text{FIB}(n) = (\text{Phi}^n - \text{PhiP}^n) / \text{Sqrt}(5).$$

This formula is good for all integers n, 0, positive or negative.

Pi => Same as the P command, Returns  $\text{Pi} = 3.14159,26535,89793\dots$  .

PiAa => Compute  $x = \text{Pi}$  by Borwein algorithm a.

PiAb => Compute x = Pi by Borwein algorithm b.

PiGL => Compute x = Pi by Gauss-Legendre algorithm.

PiAGM => Compute x = Pi by Schoenhage AGM algorithm.

PiCh => Compute x = Pi by Chudnovsky brothers' binary splitting algorithm.

It is recommended that, if you compute Pi to more places than you can verify to be correct by an independent source, you should compute Pi to this number of places by two methods and compare the files output to verify that the same value was computed by both algorithms.

The Borwein algorithms are documented in Scientific American, February 1988, Ramanujan and Pi, by Jonathan M. Borwein and Peter B. Borwein.

Algorithm a:

$$\begin{aligned} \text{Let } y[0] &= \text{SqRt}(1/2), & x[0] &= 1/2 \\ y[n] &= (1 - \text{SqRt}(1 - y[n-1]^2)) / (1 + \text{SqRt}(1 - y[n-1]^2)) \\ x[n] &= ((1 + y[n])^2 * x[n-1]) - 2^n * y[n] \end{aligned}$$

Algorithm b:

$$\begin{aligned} \text{Let } y[0] &= \text{SqRt}(2) - 1, & x[0] &= 6 - 4 * \text{SqRt}(2) \\ y[n] &= (1 - \text{SqRt}(\text{SqRt}(1 - y[n-1]^4))) / (1 + \text{SqRt}(\text{SqRt}(1 - y[n-1]^4))) \\ x[n] &= ((1 + y[n])^4 * x[n-1]) - 2^{(2n+1)} * y[n] * (1 + y[n] + y[n]^2) \end{aligned}$$

For both algorithms, x[n] converges to 1/Pi. Algorithm a is quadratically convergent and algorithm b is quartically convergent. The following table shows how many iterations are needed to compute Pi to a given number of significant digits:

Iterations of algo. a	Iterations of algo. b	Digits matching Pi
-----	-----	-----
1		0
2		3
3	1	8
4		19
5	2	41
6		84
7	3	171
8		345
9	4	694
10		1392
11	5	2788

Algorithm b converges to a given number of digits about twice as fast as algorithm a, but takes about twice as much work per iteration. Usually algorithm b is 10 to 20 percent faster than algorithm a.

The Gauss-Legendre algorithm:

1. Initial value setting;

$$a = 1 \quad b = 1 / \text{SqRt}(2) \quad t = 1/4 \quad x = 1$$

2. Repeat the following statements until the difference of a and b is within the desired accuracy;

$$y = a$$

$$a = (a+b) / 2$$

$$b = \text{SqRt}(b*y)$$

$$t = t - x * (y-a)^2$$

$$x = 2 * x$$

3. Pi is approximated with a, b and t as;

$$\text{Pi} = ((a+b)^2) / (4*t)$$

The algorithm has second order convergent nature. Then if you want to calculate up to n digits, iteration count of the order  $\log_2 n$  is sufficient. E.g. 19 times for 1 million decimal digits, 31 times for 3.2 billion decimal digits.

Note: The text for Gauss-Legendre algorithm is from the program SuperPi.Exe.

The Schoenhage's AGM algorithm:

1. Initial value setting;

$$a = 1 \quad A = 1 \quad B = 1/2 \quad t = 1/2 \quad k = 0$$

2. Repeat the following statements until the difference of a and b is within the desired accuracy;

$$S = (A+B) / 4$$

$$b = \text{SqRt}(B) \quad (\text{Full square root})$$

$$a = (a+b) / 2$$

$$A = a^2 \quad (\text{Full square multiply})$$

$$B = 2 * (A-S)$$

$$C = A - B$$

$$t = t - 2^{(k+1)} * C$$

$$k = k + 1$$

3. Pi is approximated with the final a and t as;

$$\text{Pi} = 2*a*a/t$$

The algorithm has second order convergent nature. Then if you want to calculate up to n digits, iteration count of the order  $\log_2 n$  is sufficient. E.g. 20 times for 1 million decimal digits, 32 times for 3.2 billion decimal digits.

One iteration can be saved by a more complicated initial value setting:

$$B = \text{SqRt}(1/2) \quad (\text{can be done by the SqRt inverse function})$$

$$a = \text{SqRt}(1/2)/2 + 1/2$$

$$A = \text{SqRt}(1/2)/2 + 3/8$$

$$t = \text{SqRt}(1/2) - 1/4$$

k = 1

This speedup for PiAGM is used by XMCalc, and this method appears to be the fastest of the four Pi algorithms supported; just a little faster than PiAb.

The text for Schoenhage's AGM algorithm is adapted from material by Joerg Arndt email: [arndt@jjj.de](mailto:arndt@jjj.de). The C++ software is online at <http://www.jjj.de/>.

Chudnovsky brothers' binary splitting algorithm:

The basic equation is  $\text{Pi} = (k3^{3/2} / (12*k1)) / S$ , where S is the sum of  $((c1 + n) * \text{Fac}(6*n) * (-1)^n) / (\text{Fac}(3*n) * (\text{Fac}(n))^3 * (k3)^{(3*n)})$  for n = 0 to m, and

```
k1 = 5451,40134 // = 2 * 3^2 * 7 * 11 * 19 * 127 * 163
k2 = 135,91409 // = 13 * 10,45493
k3 = 6,40320 // = 2^6 * 3 * 5 * 23 * 29
c1 = k2 / k1.
```

When m = 7, Pi can be computed to 112 decimal places or 113 digits. The number of good decimal places is at least 14\*(m+1).

This formula can be improved by rearranging terms to  $\text{Pi} = k6 * \text{Sqrt}(k3) / S$ , where S is the sum of  $((k2 + k1*n) * \text{Fac}(6*n) * (-1)^n) / (\text{Fac}(3*n) * (\text{Fac}(n))^3 * (8 * k7)^n)$  for n = 0 to m, and

```
k4 = 1001,00025 // = 3^2 * 5^2 * 23^2 * 29^2
k5 = 3278,43840 // = 2^15 * 3 * 5 * 23 * 29
k6 = 53360 // = 2^4 * 5 * 23 * 29
k7 = k4*k5 // = 2^15*3^3*5^3*23^3*29^3 = 3,20160^3 = 32,81717,65800,96000.
```

The value of m is first determined based on the precision desired, then the second form of the sum is evaluated by the binary splitting method. See:

<http://numbers.computation.free.fr/Constants/Algorithms/splitting.html>

and

<http://home.istar.ca/~lyster/pi.html>

Pri(X) => Set the execution priority in the operating system:

The execution priority is set according to x:

```
x = 2 => Highest
x = 1 => AboveNormal
x = 0 => Normal
x = -1 => BelowNormal
x = -2 => Lowest
Else => Normal;
```

The Pri command without the (X) will display the current priority. Only the integer portion of x is used.

Qi => Returns the Quaternion i = {1; 0; 0; 0}:

There are four quaternion units i, j, k, and I.  $i^2 = j^2 = k^2 = -I$ .  $i*j = -j*i = k$ ,  $j*k = -k*j = I$ ,  $k*i = -i*k = j$ ,  $I*I = I$ . Every quaternion can be written as  $a*i + b*j + c*k + d*I$ , where a, b, c, d are real numbers (scalars) and are

stored as a column vector {a; b; c; d}. This is called the scalar last convention.

See my webpage: <http://www.geocities.com/hjsmithh/Quatdoc/index.html> .

Qj => Returns the Quaternion  $j = \{0; 1; 0; 0\}$ :

See the Qi procedure.

Qk => Returns the Quaternion  $k = \{0; 0; 1; 0\}$ :

See the Qi procedure.

QId => Returns the Quaternion  $I = \{0; 0; 0; 1\}$ , identity Q:

See the Qi procedure.

Quiet(X) => Set the quiet mode on or off:

The quiet mode is turned on if  $x \neq 0$  and is turned off if  $x = 0$ . When the quiet mode is on, some of the status messages all not displayed. The (X) is optional, Quiet, Quiet(1, and Quiet(1) are interpreted as an on command.

Ran => Randomly start a new random number sequence:

This is like then Rn(X) function, but instead of the user supplying a seed; a seed is automatically generated from the current date and time.

ReadN(F) => Read file F = "ccc...c", F is optional:

This will use the argument F = "ccc...c" as a file name and read this file as an XMCalc formatted number and assign it to the item with the name stored in the file. This is the name it had when it was written. It is assumed that the file was created by the W command or the WriteN(F) procedure. See the W command for the format of file names. If no argument is given, and a comment has not been entered, the file name NoName.XMN is used. The files input by the ReadN command are assumed to have all ASCII text characters with a numerical value less than 128. The ReadN proc is similar to the I command. The ReadN command will not change the current active item unless an equal sign "=" is not found in the file or the name found is the same as the current active item.

Restore/Save => Restore or Save Configuration, History, & List:

Save will write the entry history file XMCalcHist.txt like the ] command, write the configuration file Config.XMC like the > command, write each items on the list to a separate file (Save0000.XMN, Save0001.XMN, ...), and write an XMCalc code file Restore.XMC that can be run by XMCalc to restore all of the saved items.

Restore will read the entry history file XMCalcHist.txt like the [ command, run the configuration file Config.XMC like the < command, and run the Restore.XMC restore file to read in each items that was on the list at save time. Restore does not clear the command history or the list before it executes, so they may grow larger than they were at save time.



Run(F) => Run XMCalc code from file F, F is optional:

This will use the argument F = "ccc...c" as a file name and read and run this file as an XMCalc code file. If no argument is given, defaults are like ReadN(F). To see examples of how XMCalc primitives, procedures, and functions are used, inspect the delivered \*.XMC files. It will be noted that they are in plain text.

RunH(X) => Set Height of Run form in pixels. Only the integer portion of x is used.

RunW(X) => Set Width of Run form in pixels. Only the integer portion of x is used.

Save => Save Configuration, History, & List:

Same as Ctrl+S, Save All.

SaveTop(X) => Set "save top value in LastTop" on or off:

This sets the "save top value in LastTop" option on if x != 0, and sets it off if x = 0.

ScieN(X) => ScientificN(X) => Force scientific notation on iff X != 0:

Normally numbers with no more than 16 significant digits to the left and no more than 16 to the right of the decimal point are displayed in fixed notation (e.g., 12.34). If the ScientificN(X) procedure is executed with x != 0, all numbers will be displayed in scientific notation (e.g. 1.234E+1 [4]). The normal method is restored after the ScientificN(X) procedure is executed with x = 0.

To turn scientific notation on ScieN(1) can be shortened to ScieN.

SetC(X) => Set max commands in history [1, 100000]:

The SetC(X) procedure sets the maximum number of commands that will be saved in the command history list. If X evaluates to a number greater than 100000, the max commands is changed to 100000. If x is less than 1, the max commands is not changed and the status message "Max commands in history not changed from {max}" is displayed. This message is also displayed if x is the same as the value already being used.

If the value is actually changed, the message "Max commands in history changed from {old max} to {new max}" is displayed. If there are more than x commands already in history, all but the latest x commands are removed. Only the integer portion of x is used.

SetD(X) => Set max decimal digits in display:

The SetD(X) procedure sets the maximum number of decimal digits to display to the evaluated value of X. If this is set larger than the number of digits set by the M command minus the number of digits set by the T command, the smaller value will be used to determine the number of digits to display. This maximum only applies when the display is in scientific notation. The values set by the M and T commands are always carried as a multiple of four (4), but the value set by the SetD(X) procedure can be any integer >= two (2). If this maximum is in

effect, the last digit may not be rounded. Only the integer portion of x is used.

SetM(X) => Set digits in Mantissa:

Same as the M primitive op codes.

SetMax(X) => Set max decimal digits allowed in mantissa:

The SetMax(X) procedure sets the max decimal digits allowed in any value to the evaluated value of X. If x is not a multiple of 8, then the next higher multiple of 8 is used. If x is less than 40, the value 40 is used. Only the integer portion of x is used.

Time => Set timing node on without other diags:

This selects the timing information that you get when diags are turned on, but without the other diagnostic messages. Turning diags off will turn timing off also.

Ubiq => Returns the ubiquitous constant  $U = 0.84721,30847,93979\dots$  :

$U = \text{common mean of } 1 \text{ and } \text{Sqrt}(0.5) = \text{BetaC}(.75, .75) / 2$ . The common mean function is used to compute U. See the MeanC(X, Y) function.

Write(X) => Output X, (X may be "ccc...c", X is optional):

The Write(X) procedure outputs the evaluated value of X to the console. The H command and the LogScreen(X) procedure can be used to log this output to the Log file. This procedure is mainly useful in XMCalc code files.

WriteLn(X) => Write(X) and a line feed:

The WriteLn(X) procedure is the same as the Write(X) procedure except that the output generated is followed by an end-of-line indicator.

WriteN(F) => Write X to file F = "ccc...c", F is optional):

This will use the argument F = "ccc...c" as a file name and write the current active item as an XMCalc formatted number exactly like the W command. See the W command for the format of file names. If no argument is given, and a comment has not been entered, the file name NoName.XMN is used.

XMCIIn(F) => Enter file name F = "ccc...c" for J command:

This establishes the file name of the XMCalc code file that will be read by the next J command. If the ("filename") is missing, the file name input with the last "comment" command will be used. If no comment entered, the name NoName.XMC will be used.

XMLOut(F) => Enter file name F = "ccc...c" for H command:

This establishes the file name of the XMCalc log file that will be opened by the next H command that opens a file. If the ("filename") is missing, the file

name input with the last "comment" command will be used. If no comment entered, the name NoName.XML will be used.

If the log file name is changed while it is open, the new name will not be used until logging is turned of and then turned on again. Whenever a log file is opened it is opened for append.

XMNIn(F) => Enter file name F = "ccc...c" for I command:

This establishes the file name of the XMCalc number file that will be read by the next I command. If the ("filename") is missing, the file name input with the last "comment" command will be used. If no comment entered, the name NoName.XMN will be used.

XMNOut(F) => Enter file name F = "ccc...c" for W command:

This establishes the file name of the XMCalc number file that will be written by the next W command. If the ("filename") is missing, the file name input with the last "comment" command will be used. If no comment entered, the name NoName.XMN will be used.

Functions -

Functions are used on the right hand side of an equation or assignment statement. Functions do not change the value of their arguments, but produce a single result that can be used to further complete the evaluation of the expression that contains the function reference.

If a statement starts with a function reference like a procedure, then the function is evaluated and this value is assigned to the current active item. It is best not to have one of the calculator created items like FMB, LastTop, Ln10, Pi, RN, RNA, or RNC as the current active item when using a function as if it were a procedure. For example P; Pi=; Sin(60); will clobber the value of Pi.

Abs(X) = AbsoluteValue(X):

Absolute value function = |X|.

ACos(X) = ArcCoSine(X):

Inverse of Trigonometric CoSine function, error if  $|x| > 1$ . If the degree mode is set, the answer, A, will be in the range  $0 \leq A \leq 180$ . If the radian mode is set, the answer will be in the range  $0 \leq A \leq \text{Pi}$ .

ACosh(X) = ArcHyperbolicCoSine(X):

The positive inverse of Hyperbolic CoSine function, error if  $x < 1$ .

ACot(X) = ArcCoTangent(X):

ACot(X) = ATan(1/X). ACot(0) = Pi/2 or 90 degrees.

ACoth(X) = ArcHyperbolicCoTangent(X):

ACoth(X) = ATanh(1/X). Error if  $|X| \leq 1$ .

ACsc(X) = ArcCoSecant(X):

ACsc(X) = ASin(1/X). Error if |X| < 1.

ACsch(X) = ArcHyperbolicCoSecant(X):

ACsch(X) = ASinh(1/X). Error if X = 0.

Adj(M) = Adjoint matrix of square matrix M:

The short definition is the adjoint matrix A of square n by n matrix M is the transpose of the cofactor matrix of M.  $M * A = \text{Det}(M) * \text{MatId}(n)$ . M may be singular.

Let  $M_{hj}$  denote the (n - 1) by (n - 1) submatrix of M obtained by deleting row h and column j, Then the scalar  $c_{ij} = (-1)^{(h+j)} * \text{Det}(M_{hj})$  is called the cofactor of  $M_{hj}$  in M. The n by n matrix  $\text{Tran}(c_{hj})$  is called the adjoint of M and denoted  $\text{Adj}(M)$ .

For example if  $M = \{1, 2, 3; 0, 4, 5; 1, 0, 6\}$ ,

$$M_{11} = +\text{Det}(\{4, 5; 0, 6\}) = 24,$$

$$M_{12} = -\text{Det}(\{0, 5; 1, 6\}) = 5,$$

$$M_{13} = +\text{Det}(\{0, 4; 1, 0\}) = -4,$$

$$M_{21} = -\text{Det}(\{2, 3; 0, 6\}) = -12,$$

$$M_{22} = +\text{Det}(\{1, 3; 1, 6\}) = 3,$$

$$M_{23} = -\text{Det}(\{1, 2; 1, 0\}) = 2,$$

$$M_{31} = +\text{Det}(\{2, 3; 4, 5\}) = -2,$$

$$M_{32} = -\text{Det}(\{1, 3; 0, 5\}) = -5,$$

$$M_{33} = +\text{Det}(\{1, 2; 0, 4\}) = 4,$$

So the cofactor matrix is  $\{24, 5, -4; -12, 3, 2; -2, -5, 4\}$ .

The adjoint of M is the transpose of the cofactor matrix

$$A = \text{Adj}(M) = \{24, -12, -2; 5, 3, -5; -4, 2, 4\}$$

$$M * A = A * M = \{22, 0, 0; 0, 22, 0; 0, 0, 22\}$$

$$\text{Det}(M) = 22.$$

See: <http://www.mathwords.com/a/adjoint.htm> from Mathwords.

AFib(X) = ArcFibonacciNumber(|X|), an integer:

This is the functional inverse of the Fibonacci number function. It is computed by

$$\text{AFib}(x) = \text{round}(\text{Ln}(\text{sqrt}(5) * |x|) / \text{Ln}(\text{Phi})).$$

For example,  $\text{AFib}(144) = 12.0$ . It is only accurate if x is a Fibonacci number.

AGM(X, Y) = Arithmetic Geometric Mean = MeanC(X, Y):

Exactly the same as  $\text{MeanC}(X, Y) = \text{Common mean of X and Y}$ .

ASec(X) = ArcSecant(X):

ASec(X) = ACos(1/x). Error if  $|X| < 1$ .

ASech(X) = ArcHyperbolicSecant(X):

ASech(X) = ACosh(1/x). Error if  $X \leq 0$  or  $X > 1$ .

ASin(X) = ArcSin(X):

Inverse of Trigonometric Sine function, error if  $|x| > 1$ . If the degree mode is set, the answer, A, will be in the range  $-90 \leq A \leq 90$ . If the radian mode is set, the answer will be in the range  $-\pi/2 \leq A \leq \pi/2$ .

ASinh(X) = ArcHyperbolicSine(X):

Inverse of Hyperbolic Sine function.

ATan(X) = ArcTangent(X):

Inverse of Trigonometric Tangent function. If the degree mode is set, the answer, A, will be in the range  $-90 \leq A \leq 90$ . If the radian mode is set, the answer will be in the range  $-\pi/2 \leq A \leq \pi/2$ .

ATan2(Y, X) = ArcTangent(Y over X):

Trigonometric ArcTangent function. Used to find the Polar coordinates angle coordinate of the Cartesian coordinates (x, y). If the degree mode is set, the answer, A, will be in the range  $-180 < A \leq 180$ . If the radian mode is set, the answer will be in the range  $-\pi < A \leq \pi$ . If both x and y are zero, an answer of zero will be given.

ATanh(X) = ArcHyperbolicTangent(X):

Inverse of Hyperbolic Tangent function, error if  $|x| \geq 1$ .

Balance(M) = Return a balanced matrix with same eigenvalues:

```
#include <math.h>
#define RADIX 2.0
void balance(float **a, int n)
// Given a matrix a[1..n][1..n], this routine replaces it by a balanced matrix
// with identical eigenvalues. A symmetric matrix is already balanced and is
// unaffected by this procedure. The parameter RADIX should be the machine's
// floating-point radix.
{
    int last, j, i;
    float s, r, g, f, c, sqrdx;
    sqrdx = RADIX * RADIX;
    last = 0;
    while (last == 0)
    {
        last = 1;
        for (i = 1; i <= n; i++) // Calculate row and column norms.
        {
```

```

r = c = 0.0;
for (j = 1; j <= n; j++)
    if (j != i)
    {
        c += fabs(a[j][i]);
        r += fabs(a[i][j]);
    }
if (c && r) // If both are nonzero,
{
    g = r / RADIX;
    f = 1.0;
    s = c + r;
    while (c < g) // find the integer power of the machine radix that
    { // comes closest to balancing the matrix.
        f *= RADIX;
        c *= sqrdx;
    }
    g = r * RADIX;
    while (c > g)
    {
        f /= RADIX;
        c /= sqrdx;
    }
    if ((c + r) / f < 0.95*s)
    {
        last = 0;
        g = 1.0 / f;
        for (j = 1; j <= n; j++) // Apply similarity transformation.
            a[i][j] *= g;
        for (j = 1; j <= n; j++)
            a[j][i] *= f;
    }
}
}
}
}

```

See: <http://www.library.cornell.edu/nr/bookcpdf/c11-5.pdf>

Bern(X) = Bernoulli number B(X):

This function returns the value of the Bernoulli number B(x). If x is less than zero, zero is returned. If x is not an integer, the integer portion of x is used. Bern(0) = 1. Bern(1) = -0.5. Bern(2\*n+1) = 0 for n > 0. If x > 4.05997,83000,01705,888E+18, an alarm message is displayed. If n < 150 + 0.41 \* decimal-digits, or if Bern(n) is in storage, Bern(n) = BernN(n) / BernD(n). If n is larger than this, Bern(n) = -n \* Zeta(1-n). See the Zeta(X) function.

Question: How many Bernoulli numbers are needed to compute a Bernoulli number Bern(n)? It depends on the precision desired and n. For a given precision, if n is not large enough, Bern(n) is needed to compute Bern(n) using the gamma and zeta functions. For n < 150 + 0.41 \* decimal-digits, use method by Knuth & Buckholtz, Math. Comp. 21 (1967). This is the method of BernD, BernN, and GenBern.

BernD(X) = Denominator of B(X):

This function returns the denominator of the Bernoulli number B(x), a rational number reduced to its lowest terms. If x < 0, 1 is returned. If x > 10,000,000, 1 is returned and an error message is generated. If x is not an integer, the

integer portion of  $x$  is used. This uses GenBern to generate and save all Bernoulli number upto  $Bern(x)$  if they are not already saved.

$BernDL(X)$  = Denominator of Large Bernoulli number  $B(Int(X))$ :

This function returns the denominator of the Bernoulli number  $B(x)$ , a rational number reduced to its lowest terms. If  $x < 0$ , 1 is returned. If  $x$  is not an integer, the integer portion of  $x$  is used.  $BernDL(0) = 1$ ,  $BernDL(1) = 2$ ,  $BernDL(\text{odd number} > 1) = 1$ , otherwise:

$BernDL(n)$  is computed as the product of all primes  $p$  where  $p-1$  evenly divides  $n$ .

See: [http://modular.math.washington.edu/projects/168/kevin\\_mcgown/bernproj.pdf](http://modular.math.washington.edu/projects/168/kevin_mcgown/bernproj.pdf) .

I found that for computing the denominator  $d = \text{Prod}(p-1|n)[p]$  it is a lot faster to factor  $n$  into its prime factors  $n = p_1^{e_1} * p_2^{e_2} * \dots * p_m^{e_m}$ , then generate each divisor  $d_i$  of  $n$  from this and include  $p = d_i + 1$  in  $\text{Prod}(p-1|n)[p]$  if  $p$  is a prime. This method is hinted to on page 5 of [bernproj.pdf](#).

I get the prime factors by using: "Prime Factorization by ECM, Elliptic Curve Method, from UBASIC program emc.ub, Prime Factorization by ECM, 1987-1990 by Yuji KIDA."

$BernG(X)$  = Generalized Bernoulli number  $B(X)$ :

This is the analytic continuation of the Bernoulli number. Good for all values of  $x$ . If  $x \geq 0$  and  $x$  is an integer this is the same as  $Bern(x)$ , otherwise

$$BernG(x) = -x * Zeta(1-x).$$

The exception to this is  $BernG(1) = 0.5$  were  $Bern(1) = -0.5$ .

$BernN(X)$  = Numerator of  $B(X)$ :

This function returns the numerator of the Bernoulli number  $B(x)$ , a rational number reduced to its lowest terms. If  $x < 0$ , 0 is returned. If  $x > 10,000,000$ , 0 is returned and an error message is generated. If  $x$  is not an integer, the integer portion of  $x$  is used. This uses GenBern to generate and save all Bernoulli number upto  $Bern(x)$  if they are not already saved.

$BernNL(X)$  = Numerator of Large Bernoulli number  $B(Int(X))$

This function returns the numerator of the Bernoulli number  $B(x)$ , a rational number reduced to its lowest terms. If  $x < 0$ , 0 is returned. If  $x$  is not an integer, the integer portion of  $x$  is used.  $BernNL(0) = 1$ ,  $BernNL(1) = -1$ ,  $BernNL(\text{odd number} > 1) = 0$ , otherwise  $BernNL(n)$  is computed as follows:

$k = 2*n! / (2*Pi)^n$   
 $d = BernDL(n)$   
 $m = \text{ceiling}((k * d)^{1/(n-1)})$   
 $z = \text{Product of } (1 - p^{(-n)})^{-1} \text{ for all prime } p \leq m$   
 $BernNL(n) = k*d*z$  Rounded to the nearest odd integer.  
If  $n$  is divisible by 4,  $BernNL(n) = -BernNL(n)$ .

If  $m \geq 2^{53}$ , 0 is returned and an error message is generated.

See: [http://modular.math.washington.edu/projects/168/kevin\\_mcgown/bernproj.pdf](http://modular.math.washington.edu/projects/168/kevin_mcgown/bernproj.pdf) .

Beta(X) = Dirichlet beta function:

Dirichlet beta function of  $x > 0$  is defined by the infinite series  $1 - 1/3^x + 1/5^x - 1/7^x + \dots$ . Also  $\text{Beta}(x) = \text{Lerch}(-1, x, 1/2) / 2^x$ .

For  $x \geq 0.5$ ,  $\text{Beta}(x)$  is computed by

$$\text{Beta}(x) = 1 - 1/3^x + 1/5^x - \dots - 1/(J - 2)^x + 1/(2*(J)^x) + x/(2*(J)^{(x+1)}) - x*(x+1)*(x+2)/(6*(J)^{(x+3)}) + \dots,$$

with the  $k$ -th appended term being  $x*(x+1)*\dots*(x+k-2)*2^k * (2^k - 1) * \text{Bern}(k) / (2*k!*(J^{(k+x-1)}))$ .  $\text{Bern}(k)$  is a Bernoulli number and  $J$  is a large number of the form  $4n + 1$ . This is from "An Atlas Of Functions" by Spanier, J. and Oldham, K. B. 1987, equation 3:3:7.

For  $x < 0.5$ , the reflection formula is used: let  $y = 1-x$ , then

$$\text{Beta}(x) = (2/\text{Pi})^y * \text{Sin}(\text{Pi}*y/2) * \text{Gam}(y) * \text{Beta}(y).$$

$\text{Beta}(0) = 1/2$ .  $\text{Beta}(1) = \text{Pi}/4$ .  $\text{Beta}(2) = G = \text{Catalan's constant} = 0.91596,55941,77219\dots$ .  $\text{Beta}(x) = 0$  for all negative odd integers.

$z = \text{Beta}(x)$  // Dirichlet beta function for  $x \geq 0.5$

```
{
  if (x == 0) return 1/2;
  if ((x < 0) && (x is odd)) return 0;
  sum = 0; // Beta(x) = Sum ...
  n = (int)((decimal-digits-desired / 300.0) * decimal-digits-desired + 5.0);
  if (n < 300) n = 300;
  j = 4 * n + 1;
  sign = -1; // (-1)^(k-1)
  for (int i = 3; i < j; i += 2)
  {
    sum += sign * (i ^ -x);
    sign = -sign;
  }
  sum += (j ^ -x) / 2;
  twoToK = 4;
  d = 4 * (j ^ (x + 1)); // denominator 2*k!*(J^(k+x-1)) with k = 2
  xK = x; // xK = x + k - 2
  xKP = xK; // xKP = x * (x+1) * (x+2) * ... * (x+k-2)
  GenBernI(limBern); // generate Bernoulli numbers
  for (int k = 2; ; k += 2)
  {
    term = (twoToK - 1) * twoToK * xKP;
    xK++;
    xKP *= xK;
    if (k > 2)
    {
      term *= xK;
      xK++;
      xKP += xK;
    }
    term *= Bern(k) / d; // Bern(k) = k-th Bernoulli number
    sumN = sum + term; // sumN = new sum
    if (sumN == sum) break;
    sum = sumN;
    twoToK *= 4; // twoToK = 2^k
    d *= j * j * (k+1) * (k+2);
  }
  sum += 1;
  return sum;
}
```



}

BetaC(X, Y) = The complete beta function:

The complete beta function  $BetaC(x, y) = BetaC(y, x) = \Gamma(x) * \Gamma(y) / \Gamma(x+y)$ . Also  $BetaC(x, y) = 1 / (x * Bino(x+y-1, y-1))$ .  $\Gamma(n)$  is infinite if  $n$  is an integer  $\leq 0$ , but  $BetaC(x, y)$  may have a finite value even when  $\Gamma(x)$ ,  $\Gamma(y)$  or  $\Gamma(x+y)$  are infinite.

Let  $gx = \Gamma(x)$ ,  $gy = \Gamma(y)$ ,  $gs = \Gamma(x+y)$ , and  $B = BetaC(x, y)$ . If  $gs$  is the only one of the three that is infinite,  $B = 0$ . If  $gx$  and  $gy$  are both infinite, or one is infinite without  $gs$  being infinite,  $B$  is infinite. If exactly one, say  $gy$ , of  $gx$  and  $gy$  is infinite and  $gs$  is also infinite,  $B$  has a finite value  $B = \Gamma(x) * \Gamma(1 - (x+y)) / \Gamma(1 - y)$ . The sign of  $B$  is negative iff exactly one of the integers  $y$  and  $x+y$  is odd.

If none of  $gx$ ,  $gy$ ,  $gs$  are infinite, the equation  $BetaC(x, y) = 1 / (x * Bino(x+y-1, y-1))$  is used, where  $Bino(x, y)$  is the generalized binomial coefficient.

Bino(X, Y) = Binomial coefficient (X, Y), generalized:

The binomial coefficient is generalized so  $x$  and  $y$  can be any real number, though some values will be infinite.  $Bino(x, y) = \Gamma(x+1) / (\Gamma(y+1) * \Gamma(x-y+1))$ .

The following is true if  $y$  is an integer: If  $y < 0$ ,  $Bino(x, y) = 0$ .  $Bino(x, 0) = 1$ .  $Bino(x, 1) = x$ . If  $x$  is also an integer:  $Bino(x, y)$  is the binomial coefficient of  $x$  things taken  $y$  at a time. If  $y \geq 0$  and  $x < 0$ ,  $Bino(x, y) = (-1)^y * Bino(y-x-1, y)$ . If  $y > x$ ,  $Bino(x, y) = 0$ . If  $y > x/2$ ,  $Bino(x, y) = Bino(x, x-y)$  is a better way to compute  $Bino(x, y)$ .

If  $y$  is a small integer  $> 0$ , depending on the precision desired, for any  $x$  left after using the logic of the previous paragraph, the fastest way to compute  $Bino(x, y)$  is

$$Bino(x, y) = x*(x-1)/2*(x-2)/3* \dots *(x-y+1)/y.$$

For example  $Bino(-4.5, 3) = (-1)^3 * Bino(3 + 4.5 - 1, 3) = -Bino(6.5, 3) = -(6.5)*(5.5)/2*(4.5)/3 = -26.8125$  exactly.

If none of the above applies,  $Bino(x, y) = \Gamma(x+1) / (\Gamma(y+1) * \Gamma(x-y+1))$  is used, but if  $x-y$  is an integer  $< 0$ ,  $Bino(x, y) = 0$ . For example  $Bino(5.5, 6.5) = 0$ , while  $Bino(-1, 1.5) = \Gamma(0)/(\Gamma(2.5) * \Gamma(-1.5))$  is infinite.

BinoS(X, Y) = Binomial coefficient (X, Y) by standard method:

Same as  $Bino(X)$  except does not use binary splitting.

Ceiling(X) = Least integer  $\geq X$ :

$Ceiling(2.1) = 3.0$ .  $Ceiling(-2.9) = -2.0$ .  $Ceiling(3.0) = 3.0$ .

CharPoly(M) = Characteristic polynomial of square matrix or scalar  $M$ :

The characteristic polynomial is the polynomial left-hand side of the characteristic equation:

$$\text{Det}(M - x * I) = 0$$

Where M is a square matrix and I is the identity matrix of the same size. The roots of the characteristic polynomial are the eigenvalues of the matrix.

See: <http://mathworld.wolfram.com/CharacteristicPolynomial.html> from MathWorld.

See: [http://perso.orange.fr/jean-pierre.moreau/Cplus/carpoll\\_cpp.txt](http://perso.orange.fr/jean-pierre.moreau/Cplus/carpoll_cpp.txt) from Jean-Pierre Moreau's Home Page.

If the argument is a scalar s, CharPoly(s) = x - s = {1; -s}

For scalars the characteristic polynomial is the unique monic polynomial with real coefficients and least power whose value at s is equals to zero. For example: CharPoly(1) = x - 1 = {1; -1}.

Cheby(X) = Chebyshev polynomial of Int(X.r)

The Chebyshev polynomials of the first kind are defined by the recurrence relation:

$$\text{Cheby}(0) = 1 = \{1\}$$

$$\text{Cheby}(1) = x = \{1; 0\}$$

$$\text{Cheby}(n+1) = 2*x*\text{Cheby}(n) - \text{Cheby}(n-1) = \{2; 0\}*\text{Cheby}(n) - \text{Cheby}(n-1).$$

The first few Chebyshev polynomials of the first kind are:

$$\text{Cheby}(0) = 1 = \{1\}$$

$$\text{Cheby}(1) = x = \{1; 0\}$$

$$\text{Cheby}(2) = 2*x^2 - 1 = \{2; 0; -1\}$$

$$\text{Cheby}(3) = 4*x^3 - 3*x = \{4; 0; -3; 0\}$$

$$\text{Cheby}(4) = 8*x^4 - 8*x^2 + 1 = \{8; 0; -8; 0; 1\}$$

$$\text{Cheby}(5) = 16*x^5 - 20*x^3 + 5*x = \{16; 0; -20; 0; 5; 0\}$$

$$\text{Cheby}(6) = 32*x^6 - 48*x^4 + 18*x^2 - 1 = \{32; 0; -48; 0; 18; 0; -1\}$$

$$\text{Cheby}(7) = 64*x^7 - 112*x^5 + 56*x^3 - 7*x = \{64; 0; -112; 0; 56; 0; -7; 0\}$$

$$\text{Cheby}(8) = 128*x^8 - 256*x^6 + 160*x^4 - 32*x^2 + 1 = \{128; 0; -256; 0; 160; 0; -32; 0; 1\}$$

$$\text{Cheby}(9) = 256*x^9 - 576*x^7 + 432*x^5 - 120*x^3 + 9*x = \{256; 0; -576; 0; 432; 0; -120; 0; 9; 0\}$$

$$\text{Cheby}(-n) = \text{Cheby}(n).$$

See:

<http://mathworld.wolfram.com/ChebyshevPolynomialoftheFirstKind.html>

[http://en.wikipedia.org/wiki/Chebyshev\\_polynomials](http://en.wikipedia.org/wiki/Chebyshev_polynomials)

Chin(X, Y) = add to Chinese remainder problem z == X mod Y:

This uses the Chinese remainder theorem (CRT) to solve

$$z == x1 \text{ mod } y1$$

$$z == x2 \text{ mod } y2$$

for z, where x1 is the value of X1 and y1 is the value of Y1 on the named number list, x2 is the x and y2 is the y value being input. X1 on the list is set to the solution z and Y1 is set to the new modulus = y1\*y2/g. A solution exists if g = GCD of y1 and y2 is equal to one or (y1 - y2) mod g = 0. If no solution exists, z, X1, and Y1 are set to zero. The signs of y1 and y2 are not used.

Note, z == x mod y is to be read z is congruent to x modulo y. This means that z and x have the same remainder when divided by y. This is the same as (z - x) is divisible by y, or (z - x) mod y = 0. Since y > 0, the mod operator here is the

remainder function.  $x \bmod y = r$  such that  $0 \leq r < y$ , and  $x = q * y + r$ , and  $q$  the integer quotient of  $x / y$ .

Chin1(X, Y) = Initialize Chinese remainder with X1, Y1:

Y1 on the named number list is set to  $|y|$  and X1 to  $x \bmod Y1$ . The value X1 is returned. To solve the Chinese remainder problem

$$z == x_i \bmod y_i \text{ for } i = 1 \text{ to } n$$

use Chin1(x1, y1) to enter the first congruence and then use Chin(xi, yi) to enter each of the congruences with  $i > 1$ . The answer  $z$  is returned and stored on the list as X1. Y1 on the list is set to the new modulus. All  $z == X1 \bmod Y1$  are solutions, X1 is just the smallest nonnegative one.

For example, an input of

$$Z = \text{Chin1}(2, 3) \text{ Chin}(3, 5) \text{ Chin}(2, 7)$$

will give  $Z = 23$ ,  $X1 = 23$ , and  $Y1 = 105$ . So the answers are  $23 + 105*n$ ;  $n = 0, 1, 2, \dots$

CompM(P) = Companion matrix of polynomial P:

First P is converted to a monic polynomial. Then the companion matrix to a monic polynomial

$$a(x) = \{1; a(n-1); \dots, a(1); a(0)\}$$

is the  $n$  by  $n$  square matrix

$$A = \begin{bmatrix} 0 & 0 & \dots & 0 & -\alpha_0 \\ 1 & 0 & \dots & 0 & -\alpha_1 \\ 0 & 1 & \dots & 0 & -\alpha_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & -\alpha_{n-1} \end{bmatrix}$$

with ones on the subdiagonal and the last column given by the coefficients of  $a(x)$ ,  $\{-a(0); -a(1); -a(2); \dots; -a(n-1)\}$ . The characteristic polynomial of the companion matrix is the same monic polynomial.

See: <http://mathworld.wolfram.com/CompanionMatrix.html> from MathWorld.

Concat(X, Y) = Concatenate the columns of matrix X and Y:

Returns a matrix with the columns of X followed by the columns of Y.

ConcatR(X, Y) = Concatenate the rows of matrix X and Y:

Returns a matrix with the rows of X followed by the rows of Y.

ContFrac(X) = Continued fraction expansion of  $x$ :

Computes the terms of the continued fraction that represents the real number  $x$ . This is computed as an  $n$  element row vector of integers  $\{b_0, b_1, b_2, \dots, b_{(n-1)}\}$ , where  $x = b_0 + 1/(b_1 + 1/(b_2 + 1/(b_3 + \dots + 1/b_{(n-1)})))$ .

$\text{ContFrac}(X, Y)$  = Continued fraction expansion of  $x$  with  $y$  terms:

This is the same as  $\text{ContFrac}(X)$  except at most  $y$  terms/elements are computed.

$\text{ContFracV}(X)$  = Value of continued fraction  $x$ :

This is the inverse of the  $\text{ContFrac}()$  function. The row (or column) vector  $x$  is interpreted as a continued fraction and converted to a real value.

$\text{Cos}(X)$  = CoSine( $X$ ):

Trigonometric CoSine function, error if  $|x|$  is very large.

$\text{Cosh}(X)$  = HyperbolicCoSine( $X$ ):

Hyperbolic CoSine function.

$\text{Cot}(X)$  = CoTangent( $X$ ):

$\text{Cot}(X) = 1/\text{Tan}(X)$ .  $\text{Cot}(X) = 0$  if  $X = \text{Pi}/2 \pm n*\text{Pi}$ . Error if  $\text{Tan}(X) = 0$ .

$\text{Coth}(X)$  = HyperbolicCoTangent( $X$ ):

$\text{Coth}(X) = 1/\text{Tanh}(X)$ . Error if  $X = 0$ .

$\text{Csc}(X)$  = CoSecant( $X$ ):

$\text{Csc}(X) = 1/\text{Sin}(X)$ . Error if  $\text{Sin}(X) = 0$ .

$\text{Csch}(X)$  = HyperbolicCoSecant( $X$ ):

$\text{Csch}(X) = 1/\text{Sinh}(X)$ . Error if  $X = 0$ .

$\text{CuRt}(X)$  = CubeRoot( $X$ ) =  $X^{(1/3)}$ :

The cube root function. The sign of  $\text{CuRt}(x)$ , and  $x$  are the same. For example  $\text{CuRt}(-27) = -3$ .

$\text{Decomp}(M)$  = The LU decomposition of matrix  $M$ :

Returns the LU decomposition of matrix  $M$ .  $L*U = M$ .  $U$  is an upper triangular matrix and  $L$  is a lower triangular matrix with its diagonal elements all equal to one. The returned matrix is the sum of  $U$  and  $L$  minus  $L$ 's diagonal elements.

See: <http://www.library.cornell.edu/nr/bookcpdf/c2-3.pdf> from Numerical Recipes Home Page.

$\text{Del}(M, C)$  = Delete column  $C$  of matrix  $M$ :

Returns matrix M with the C-th column deleted. If the last column is deleted, the result will be the scalar zero.

`DelR(M, R)` = Delete row R of matrix M

Returns matrix M with the R-th row deleted. If the last row is deleted, the result will be the scalar zero.

`Deriv(P)` = Derivative of polynomial P:

Since the derivative of  $5x^4 + 4x^3 + 3x^2 + 2x + 1 = 20x^3 + 12x^2 + 6x + 2$ , `Deriv({5; 4; 3; 2; 1}) = {20; 12; 6; 2}`.

`Det(M)` = Determinate of a matrix M:

Computes the determinant, a scalar, of the square matrix M. The determinant is zero iff the matrix is singular.

One way to compute `Det(A)` of a square matrix A:

```
If (A.m == 1)
  Det(A) = a11
else If (A.m == 2)
  Det(A) = a11 * a22 - a12 * a21
else
{
  compute Decomp(A) = The LU decomposition of matrix A
  Det(A) = the product of the diagonal elements of Decomp(A)
}
```

See: <http://mathworld.wolfram.com/Determinant.html> from MathWorld.

`Diagonal(V)` = Diagonal matrix from a column or row vector V:

Returns a diagonal matrix with its diagonal elements equal to the column or row vector V and the off diagonal elements equal to zero.

`Dig(X, Y)` = Number of base Y digits in X,  $Y \geq 2$ :

For example, `Dig(5, 2) = 3` ( $5 = 101$  base 2). The signs of X and Y are not used. If  $|Y| < 2$ , Y is changed to 10 and a message is generated. Y can be larger than 36. If  $X == 0$ , the answer is 0. If  $Y == X$ , the answer is 2. If  $Y > X$ , the answer is 1.

`DigD(X)` = Number of decimal digits in X:

For example, `DigD(427) = 3`. If  $X == 0$ , the answer is 0.

`Dilog(X)` = Dilogarithm(X):

The `Dilog(x)` function is equal to `Polylog(2, x)` which is equal to  $x * \text{Lerch}(x, 2, 1)$ . See the `Polylog(S, X)` and `Lerch(X, S, A)` functions.

$$\text{Dilog}(x) = \sum_{k=1, 2, \dots} [x^k / k^2],$$

where  $|x| \leq 1$ . The real function is defined for all  $x \leq 1$ .

When  $x < -1$ ,  $\text{dilog}(1-y) + \text{dilog}(1-1/y) = -(\ln(y))^2 / 2$  is used in the form

$\text{dilog}(x) = -\text{dilog}(1-y) - (\ln(y))^2 / 2$ , where  $x = 1-1/y$ ,  $y = 1/(1-x)$ .

$\text{DivInt}(X, Y) = \text{Floor}(X/Y)$ , integer divide:

Integer divide,  $\text{DivInt}(X, Y) = x \setminus y = \text{Floor}(x/y)$ . An error message is given if  $y = 0$ .

$\text{DivRem}(X, Y) = \text{Floor}(X/Y)$  and set  $\text{Re}$  to remainder:

Integer divide,  $q = \text{DivInt}(X, Y) = x \setminus y = \text{Floor}(x/y)$ . The remainder is also computed and added to the list as an item names  $\text{Re}$ . An error message is given if  $y = 0$ .  $x = q * y + \text{Re}$ . The remainder always has the same sign as  $y$  or equal to 0,  $0 \leq |\text{Re}| < |y|$ . If  $\text{Re}$  is on top of the list, an error message is displayed: "Cannot set same location to both quotient and remainder, continuing...".

$\text{E1}(X) = \text{Exponential integral function one}$ :

$\text{E1}(x) = \int_1^{\infty} [e^{-t*x} / t] dt = \int_x^{\infty} [e^{-t} / t] dt$ .

$\text{E1}(x)$  is related to  $\text{Ei}(x)$  by  $\text{E1}(x) = -\text{Ei}(-x)$ .

For very large positive  $x$ , the asymptotic expansion of  $\text{Ei}(x)$  is used:

$\text{Ei}(x) \sim \text{Exp}(x) * (1/x + 1/x^2 + 2/x^3 + \dots + k!/x^{(k+1)} + \dots)$

It is used if  $x \geq \text{lim} = 10 + \text{digits-desired} * \ln(10)$ . If  $x < \text{lim}$ , the series expansion of  $\text{Ei}(x)$  is used:

$\text{Ei}(x) = \ln(|x|) + \gamma + x + x^2/4 + x^3/18 + \dots + x^k/(n*n!) + \dots$ ,  
where  $\gamma$  is Euler's constant = EulerC = 0.57721,56649,01532...

When  $1 < x < \text{lim}$ , this series expansion is subject to large relative errors. In this case, extra digits are added to the calculations to maintain the final desired precision. The number of decimal digits added is more =  $8 * (1 + x / 9.22)$ . This process can be monitored if  $\text{Diag}$  is turned on.

$\text{Ei}(X) = \text{Exponential integral Ei}(x) = -\text{E1}(-x)$ :

See  $\text{E1}(x)$  above.

$\text{EqQ}(Q) = \text{Quaternion equivalent to quaternion } Q$ :

For every rotational quaternion i.e. a normalized quaternion, there is another quaternion that performs an equivalent rotation, but in a different way. This equivalent quaternion of  $Q$  is  $-Q = -1*Q$ .

$\text{Erf}(X) = \text{Error function}$ :

By definition,  $\text{Erf}(x) = 2/\text{Sqrt}(\text{Pi}) * \int_0^x [\text{Exp}(-t^2)] dt$ .

$$\text{Erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

If  $|x| \geq 10^{-8}$ ,  $\text{Erf}(x) = 1 - \text{ErfC}(x)$ , else

$$\text{Erf}(x) = 2x/\text{Sqrt}(\text{Pi}) * \text{Sum}[(-x^2)^j / (j!(2*j + 1))], j \geq 0.$$

$$\text{Erf}(x) = \frac{2x}{\sqrt{\pi}} \left[ 1 - \frac{x^2}{3} + \frac{x^4}{10} - \dots \right] = \frac{2x}{\sqrt{\pi}} \sum_{j=0}^{\infty} \frac{(-x^2)^j}{j!(2j+1)}.$$

This sum is needed for small  $x$  to minimize the relative error in  $\text{Erf}(x)$ .

$\text{ErfC}(X)$  = Complementary error function:

By definition,  $\text{ErfC}(x) = 1 - \text{Erf}(x)$ . If  $x \geq 0$ ,  $\text{ErfC}(x) = \text{GamQ}(1/2, x^2)$ , where  $\text{GamQ}(x, y)$  is the upper regularized incomplete gamma function (see the  $\text{GamQ}(X, Y)$  function). If  $x < 0$ ,  $\text{ErfC}(x) = 2 - \text{ErfC}(-x)$ .

$\text{Eta}(X)$  = Dirichlet eta function:

Dirichlet eta function of  $x > 0$ , is defined by the series  $1 - 1/2^x + 1/3^x - 1/4^x + \dots$ .

For of  $x \geq -4$ ,  $\text{Eta}(x)$  is computed by

$$\text{Sum}\{k=1 \text{ to } n\} [(-1)^{(k+1)} * k^{-x}] + 2^{(-n)} * \text{Sum}\{k=n+1 \text{ to } 2*n\} [((-1)^{(k+1)} * k^{-x}) * \text{Sum}\{j=0 \text{ to } 2*n-k\} [\text{Bino}(n, j)]]$$

where  $n = (\text{int})(5 + 1.3 * \text{decimal-digits-desired}) + 5$  and  $\text{Bino}(n, j)$  is the binomial coefficient.

For negative  $x < -4$ , let  $y = 1-x$ , then

$$\text{Eta}(x) = 2 * (2*\text{Pi})^{(-y)} * \text{Sin}(\text{Pi}*x/2) * \text{Gam}(y) * \text{Eta}(y) * (1-2^y)/(1-2^x)$$

$\text{Eta}(0) = 1/2$ .  $\text{Eta}(1) = \text{Ln}(2)$ .  $\text{Eta}(2) = \text{Pi}^2 / 12$ .  $\text{Eta}(3) = 3*\text{Zeta}(3)/4$ .  $\text{Eta}(4) = 7*\text{Pi}^4 / 720$ .  $\text{Eta}(x) = 0$  for all negative even integers.

$\text{Eta} = \text{Eta}(x)$  // Dirichlet eta function for  $x \geq -4$

```
{
  if (x == 0) return 1/2;
  if ((x < 0) && (x is even)) return 0;
  n = (int)(5 + 1.3 * decimal-digits-desired) + 5;
  sum = 0;
  sign = -1;          // (-1)^(k-1)
  c = 1;              // c = Bino(n, j)
  e = 1;              // e = Sum Bino(n, j)
  m = n;              // m = n - j + 1
  j = 1;
  for (k = n+n; k > n; k--)
  {
    sum += sign * e * (k ^ -x);
    c = (c * m) / j;
  }
}
```

```

    e += c;
    m--;
    j++;
    sign = -sign;
}
sum = sum / (2 ^ n);
for (k = n; k > 0; k--)
{
    sum += sign * (k ^ -x);
    sign = -sign;
}
return sum;
}

```

Euler(X) = Euler number E(X):

This function returns the value of the Euler number E(x), an integer. If x is less than zero, zero is returned. If x is not an integer, the integer portion of x is used. Euler(0) = 1. Euler(1) = 0. Euler(2\*n+1) = 0 for n >= 0. If x > 4.05997,83000,01705,888E+18, an alarm message is displayed. If x < 150 + 0.55 \* decimal-digits, this is the same as the EulerN(X) command. If x is larger than this, the equation Euler(n) = (-1)^(n/2) \* 2 \* Gam(n+1) \* Beta(n+1) / (Pi/2)^(n+1) is used.

EulerG(X) = Generalized Euler number E(X):

This is the analytic continuation of the Euler number. Good for all values of x. If x >= 0 and x is an integer this is the same as Euler(x), otherwise

$$\text{EulerG}(x) = 2 * \text{Beta}(-x).$$

EulerN(X) = Numerator of E(X) = E(X):

This function returns the value of the Euler number E(x), an integer. If x < 0, 0 is returned. If x > 10,000,000, 0 is returned and an error message is generated. If x is not an integer, the integer portion of x is used. This differs from the Euler(X) command in that it generates and save all Euler number upto Euler(x) if they are not already saved. The Euler(X) command only generates and saves the Euler numbers if x is relatively small.

Exp(X) = eToThePower(X):

Evaluates to e raised to the x power, where e is the base of the natural logarithms. The item Ln10 = Ln(10) is put on the list when needed by the exponential functions. If Ln10 is not on the list, the file Ln10.XPN is read-in and Ln10 added to the list. If the file Ln10.XPN is not found, Ln10 is computed. The Ln10.XPN file delivered with XMCalc has 10144 decimal digits.

ExpL(X) = eToThePower(X) - 1:

Evaluates to one less than e raised to the x power, where e is the base of the natural logarithms. This function is needed when an expression contains Exp(X) - 1 and X can take on small values. ExpL(X) is accurate for small X.

Extract(M, C) = Extract column C from matrix M:

Returns a column vector equal to the C-th column of M.



ExtractR(M, R) = Extract row R from matrix M:

Returns a row vector equal to the R-th row of M.

Fac(X) = Factorial of Int(X):

Factorial function =  $1 * 2 * 3 * \dots * x$ . Only the integer portion of x is used in the calculation. If  $n < 0$ , zero is returned with an error message. If  $x > 4.05997,83000,01705,888E+18$ , zero is returned with the error message: "Number too large for factorial function". If x is not too large but  $x > 2000$ ,  $n! = \text{Gam}(n + 1)$  is used. See the Gam(X) function.

Fac2(X, Y) = Fac(Int(X)) / Fac(Int(Y)) by binary splitting:

For the binary splitting method, define the product

$$\text{Pr}(a, b) = (b+1)*(b+2)* \dots * (a-1)*a = \text{Fac}(a) / \text{Fac}(b).$$

This is computed by

$$\text{Pr}(a, b) = \text{Pr}(a, (a+b)/2) * \text{Pr}((a+b)/2, b),$$

where the two terms in the product are computed recursively in the same way until a-b is small:

```
d = a-b; m = (a+b)/2 // integer divide
If (d) > 3, Pr(a, b) = Pr(a, m) * Pr(m, b);
Else if (d) == 0, Pr(a, b) = 1;
Else if (d) == 1, Pr(a, b) = a;
Else if (d) == 2, Pr(a, b) = a*(a-1);
Else if (d) == 3, Pr(a, b) = a*(a-1)*(a-2);
Else Pr(a, b) = 0.
```

FacM(X) = (Factorial of Int(X)) Mod FMB:

The factorial function with modulo arithmetic. The modulo process is performed after each multiply to prevent the intermediate results from becoming large. If FMB is not on the list, it is added to the list with a value of zero. If FMB is zero, the modulo is not performed. See the % primitive.

FacM2(X, Y) = Fac(Int(X)) / Fac(Int(Y)) Mod FMB by binary splitting:

Same as Fac2(X, Y), but with modulo arithmetic like FacM(X). Except, Cannot compute FacM2(X, Y) if FMB is not an integer.

FacMS(X) = (Factorial of Int(X)) Mod FMB by standard method:

Same as FacM(X) except does not use binary splitting.

FacS(X) = Factorial of Int(X) by standard method:

Same as Fac(X) except does not use binary splitting.

Fib(X) = FibonacciNumber(X), Fib(0) = 0:

Fib(N) returns the N'th Fibonacci number.  $Fib(0) = 0$ ,  $Fib(1) = 1$ ,  $Fib(n+2) = Fib(n) + Fib(n+1)$  for all  $n$ . If  $n$  is odd,  $Fib(-n) = Fib(n)$ . If  $n$  is even,  $Fib(-n) = -Fib(n)$ . For  $n \geq 0$ ,  $Fib(n) = Round(\Phi^n / \sqrt{5})$ , where  $\Phi = (1 + \sqrt{5})/2 =$  the Golden Ratio. If  $x$  is not an integer, the generalized Fibonacci number is computed:

$$Fib(x) = (\Phi^x - \cos(x\pi) / \Phi^x) / \sqrt{5}.$$

See: <http://www.geocities.com/hjsmithh/Fibonacci/FibWhat.html>

Floor(X) = Greatest integer  $\leq X$ :

Floor(2.9) = 2.0. Floor(-2.1) = -3.0. Floor(3.0) = 3.0.

Frac(X) = FractionalPart(X).

Gam(X) = GammaFunction(X) = (X-1)!::

If  $|x| > +4.05997,83000,01705,88908,81E+18$ , an error message is displayed: "Number too large for gamma function". Otherwise, for  $x \geq -5$ , the asymptotic formula with Bernoulli numbers is used to compute this. If  $x$  is an integer  $\leq$  zero, an alarm is displayed. For non-integer negative  $x < -5$ , the reflection formula is used:  $Gam(x) = \pi / (Gam(y) * \sin(\pi * y))$ , where  $y = 1 - x$ . For large  $x$  ( $x > n$ ),

$$\ln(Gam(x)) \approx (x - 0.5) * \ln(x) - x + \ln(2\pi)/2 + \sum_{k=1,2,\dots} [Bern(2*k) / (2*k*(2*k-1)*x^(2*k-1))],$$

where  $Bern(2*k)$  are Bernoulli numbers. For small  $x$ ,

$$Gam(x) = Gam(n + x) / x / (x+1) / (x+2) / \dots / (n-1+x).$$

This is based on  $Gam(x) = Gam(x+1) / x$ .  $n$  is computed by a heuristic,  $n = (digits^{1.5}) / 13.0 + 1$ , where  $digits$  is the current decimal digits in a computed mantissa. Actually for very small integer the factorial function is used:  $Gam(n) = (n-1)! = 1*2*3*4*\dots*(n-1)$ . If  $x$  is an integer  $\leq$  zero, an alarm is displayed.

Gam1(X) = 16-digit GammaFunction(X):

Gamma function of  $x$ . If  $x$  is a positive integer,  $Gam1(x) = (x-1)!$ . Otherwise  $Gam1(x)$  is only good to 15 or 16 decimal digits. If  $x$  is an integer  $\leq$  zero, an alarm is displayed.

GamL(A, X) = Lower incomplete gamma function:

$GamL(a, x)$  is the lower incomplete gamma function with parameter  $a$  and argument  $x$ . It is defined as the integral from 0 to  $x$  of  $Exp(-t) * t^{(a-1)} dt$  ( $a > 0$ ).

If  $x < a + 1$  or  $x < 100$ ,  $GamL(a, x) = Exp(-x) * \sum[x^n * Gam(a)/Gam(a+1+n)]$  is used, summed for  $n = 0$  to when the sum stops changing.

If  $x \geq a + 1$  and  $x \geq 100$ ,  $GamL(a, x) = Gam(a) - GamU(a, x)$ , where  $GamU$  is the upper incomplete gamma function. See the  $GamU(X, Y)$  function.

Cannot take  $GamL(a, x)$  if  $a$  is an integer  $\leq$  zero, except  $GamL(0, 0) = 0$ .

GamP(A, X) = lower regularized incomplete gamma function:

GamP(a, x) = GamL(a, x) / Gam(a). Cannot take GamP(a, 0) if a is an integer < zero. GamP(0, 0) = 0.0.

GamQ(A, X) = Upper regularized incomplete gamma function:

GamQ(a, x) = GamU(a, x) / Gam(a). Cannot take GamQ(a, 0) if a is an integer < zero. GamQ(0, 0) = 1.0.

GamU(A, X) = Upper incomplete gamma function:

GamU(a, x) is the upper incomplete gamma function with parameter a and argument x. It is defined as the integral from x to infinity of  $\text{Exp}(-t) * t^{(a-1)} dt$  (a > 0).

If  $x < a + 1$  or  $x < 100$ ,  $\text{GamU}(a, x) = \text{Gam}(a) - \text{GamL}(a, x)$ , where GamL is the lower incomplete gamma function. See the GamL(X, Y) function.

If  $x \geq a + 1$  and  $x \geq 100$ ,  $\text{GamU}(a, x) = \text{Exp}(-x) * x^a * (1/x + (a-1)/1 + 1/x + (2-a)/1 + 2/x + \dots)$  is used, summed evaluated until the value stops changing. The notation  $(1/x + (a-1)/1 + 1/x + (2-a)/1 + 2/x + \dots)$  represents a continued fraction development for GamU(a, x) that converges for  $x > 0$ .

Cannot take GamU(a, 0) if a is an integer  $\leq$  zero.

GCD(X, Y) = Greatest Common Divisor:

Greatest common divisor function. Uses the oldest algorithm in the book, Euclid's algorithm (see Euclid's Elements, Book 7, Propositions 1 and 2). Both the integer and fractional parts of the absolute values of x and y are used in the computation. For example, GCD(11.5, -16.1) is 2.3.

GCDe(X, Y) = Extended GCD(X, Y) = X\*X1 + Y\*Y1:

The extended GCD returns  $g = \text{GCD}(x, y) \geq 0$  and also computes x1 and y1 such that  $x*x1 + y*y1 = g$ . This is useful for computing modular multiplicative inverses. The signs of x and y are used in the computation. For example, the GCDe of 12 and -18 is 6 with x1 = -1 and y1 = -1 since  $-1*12 + -1*-18 = 6$ . The GCDe(-99, 78) = 3 with x1 = 11 and y1 = 14 since  $11*-99 + 14*78 = 3$ . The named numbers "X1" = x1 and "Y1" = y1 are added to the named number list.

Get(M, R, C) = Returns element at row R, column C of matrix M

R and C are scalars and their integer part is used. M is a matrix with at least R rows and C columns.

Hess(M) = Return upper Hessenberg matrix, same eigenvalues:

```
#include <math.h>
#define SWAP(g, h) {y = (g); (g) = (h); (h) = y;}
void hess(float **a, int n)
// Reduction to Hessenberg form by the elimination method. The real,
// nonsymmetric matrix a[1..n][1..n] is replaced by an upper Hessenberg matrix
// with identical eigenvalues. Recommended, but not required, is that this
// routine be preceded by balance. On output, the Hessenberg matrix is in
```

```

// elements a[i][j] with i <= j + 1. Elements with i > j+1 are zero.
{
  int m, j, i;
  float y, x;
  for (m = 2; m < n; m++) // m is called r + 1 in the text.
  {
    X = 0.0;
    i = m;
    for (j = m; j <= n; j++) // Find the pivot.
    {
      if (fabs(a[j][m - 1]) > fabs(x))
      {
        x = a[j][m - 1];
        i = j;
      }
    }
    if (i != m) // Interchange rows and columns.
    {
      for (j = m - 1; j <= n; j++)
        SWAP(a[i][j], a[m][j])
      for (j = 1; j <= n; ++j)
        SWAP(a[j][i], a[j][m])
    }
    if (x != 0.0) // Carry out the elimination.
    {
      for (I = m + 1; I <= n; i++)
      {
        if ((y = a[i][m - 1]) != 0.0)
        {
          y /= x;
          a[i][m-1] = y;
          for (j = m; j <= n; j++)
            a[i][j] -= y * a[m][j];
          for (j = 1; j <= n; j++)
            a[j][m] += y * a[j][i];
        }
      }
    }
  }
  for (j = 1; j <= n - 2; j++)
    for (i = j + 2; i <= n; i++)
      a[i, j] = 0.0;
}

```

See: <http://www.library.cornell.edu/nr/bookcpdf/c11-5.pdf>

Hilbert(X) = Equals an X by X Hilbert matrix:

Hilbert(X) = H, an X by X matrix with  $H_{ij} = 1 / (i + j - 1)$ .

Example: Hilbert(3) = {1, 1/2, 1/3; 1/2, 1/3, 1/4; 1/3, 1/4, 1/5}.

See: <http://mathworld.wolfram.com/HilbertMatrix.html> from MathWorld.

Image(M) = Returns a basis for the image/range of matrix M:

Returns a list of column vectors forming a vector space basis for the range space of M. This uses the RREF(M) Reduced Row Echelon Form of matrix M to find the pivotal columns of M and returns those columns of M. A vector is in the range space of M iff it is a linear combinations of the returned column vectors.

See: <http://www.math.lsa.umich.edu/~hochster/419/ker.im.html> FINDING A BASIS FOR THE KERNEL OR IMAGE -- from Mel Hochster, University of Michigan.

Insert(M, V, C) = Insert column vector V at column C of matrix M:

Return matrix M with column vector V at column C.

InsertR(M, V, R) = Insert row vector V at row R of matrix M:

Return matrix M with row vector V at row R.

Int(X) = IntegerPart(X):

Integer part function. If  $x \geq 0$ , Int(x) is the largest integer less than or equal to x. If  $x < 0$ , Int(x) = -Int(-x). See the Floor(X) function.

Integ(P) = Formal integral of polynomial P:

For example, since the formal integral of  $20x^3 + 12x^2 + 6x + 2 = 5x^4 + 4x^3 + 3x^2 + 2x$ , Integ({20; 12; 6; 2}) = {5; 4; 3; 2; 0}.

Inv(X) = 1 / X:

Inverse or reciprocal function, 1.0 divided by X, error if X = 0. X may be a square matrix.

One way to invert a square n by n matrix X is to do a LU decomposition of matrix X and then do n LU back substitutions to solve for each column of the inverse. If one of the diagonal elements of U is zero, the matrix X is singular and cannot be inverted.

See: <http://www.library.cornell.edu/nr/bookcpdf/c2-3.pdf> LU Decomposition and Its Applications -- From Numerical Recipes Home Page.

See: <http://mathworld.wolfram.com/MatrixInverse.html> Matrix Inverse -- from MathWorld

Inv(X, Y) = Z = Inverse of X Mod Y,  $X*Z == 1 \text{ Mod } Y$ :

Returns z ( $0 < z < y$ ) such that  $x*z == 1 \text{ Mod } y$  or 0 (False) if no inverse exists, i.e. if x and y are not relatively prime. The sign of y is not used.

InvQ(Q) = Inverse quaternion of quaternion Q

Returns the conjugate of the quaternion Q, its inverse if it is normalized.

InvQ({a; b; c; d}) = {-a; -b; -c; d}.

IsDiag(M) = Returns 1 if M is a diagonal matrix, else 0:

If M is a square matrix with all off diagonal elements equal to zero, the result is 1, else 0. If M is a scalar, 0 is returned.

IsFib(X) = 1 (True) if X is a Fibonacci number, else 0:

Set equal to 1 (True) if  $|x|$  is a Fibonacci number, else it is set to 0 (False). True iff  $x = 0$  or  $z = |x|$  is an integer and the closed interval  $[\Phi z - 1/z, \Phi z + 1/z]$  contains a positive integer. Extra testing is performed if  $x < 0$  and  $|x|$  is a Fibonacci number. 144, 233, and -144 are Fibonacci number, but -233 is not. For  $x < 0$  to be a Fibonacci number,  $|x|$  must be a Fibonacci number and  $\text{AFib}(x)$  must be even.

$\text{IsFib2}(X) = \text{IsFib}(X)$  by second method:

Same as  $\text{IsFib}(X)$  except true iff  $x = 0$  or  $z = |x|$  is an integer and  $5x^2 + 4$  or  $5x^2 - 4$  is a perfect square.

$\text{IsSq}(X) = 1$  (True) if  $X$  is a square, else 0:

Set equal to 1 (True) if  $x$  is a perfect square integer, else it is set to 0 (False). This is computed by factoring  $x$  and examining the powers of the prime factors.  $x$  is a square iff all of the powers are even. The first prime found with an odd power causes the factoring to stop, and an answer of 0 (False) is given. It may be faster to use the  $\text{SqrtRem}(X)$  function and check the remainder in  $\text{Re}$  for zero if the number is hard to factor.

$\text{Kron}(X, Y) = \text{Kronecker-Legendre symbol } X \text{ over } Y$ :

The Kronecker symbol is an extension of the Jacobi symbol to all integers. It is variously written as  $(x/y)$  or  $(x|y)$  or  $(x \text{ vinculum } y)$ . Some values are:

	y =:	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9
x = -9:		0	-	+	0	-	-	0	-	-	0	+	+	0	+	+	0	-	+	0
x = -8:		-	0	+	0	+	0	-	0	-	0	+	0	+	0	-	0	-	0	+
x = -7:		-	-	0	+	+	-	+	-	-	0	+	+	-	+	-	-	0	+	+
x = -6:		0	0	-	0	-	0	0	0	-	0	+	0	0	0	+	0	+	0	0
x = -5:		-	+	-	+	0	-	-	+	-	0	+	-	+	+	0	-	+	-	+
x = -4:		-	0	+	0	-	0	+	0	-	0	+	0	-	0	+	0	-	0	+
x = -3:		0	+	-	0	+	-	0	+	-	0	+	-	0	+	-	0	+	-	0
x = -2:		-	0	+	0	+	0	-	0	-	0	+	0	+	0	-	0	-	0	+
x = -1:		-	-	+	+	-	-	+	-	-	+	+	+	-	+	+	-	-	+	+
x = 0:		0	0	0	0	0	0	0	0	0	+	0	+	0	0	0	0	0	0	0
x = 1:		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
x = 2:		+	0	+	0	-	0	-	0	+	0	+	0	-	0	-	0	+	0	+
x = 3:		0	-	-	0	-	+	0	-	+	0	+	-	0	+	-	0	-	-	0
x = 4:		+	0	+	0	+	0	+	0	+	0	+	0	+	0	+	0	+	0	+
x = 5:		+	-	-	+	0	+	-	-	+	0	+	-	-	+	0	+	-	-	+
x = 6:		0	0	-	0	+	0	0	0	+	0	+	0	0	0	+	0	-	0	0
x = 7:		+	+	0	+	-	+	+	+	+	+	+	+	+	+	-	+	0	+	+
x = 8:		+	0	+	0	-	0	-	0	+	0	+	0	-	0	-	0	+	0	+
x = 9:		0	+	+	0	+	+	0	+	+	0	+	+	0	+	+	0	+	+	0

where + is for +1 and - is for -1. For example  $\text{Kron}(3, 7) = -1$ ,  $\text{Kron}(7, 3) = +1$ . This function is used internally by the Adleman function for prime number testing, and is part of the prime factorization by Elliptic Curve Method (ECM).

$\text{Lam}(X) = \text{Dirichlet lambda function}$ :

The Dirichlet lambda function of  $x > 1$  is defined by the infinite series  $1 + 1/3^x + 1/5^x + 1/7^x + \dots$ . It is evaluated by first computing the Zeta function and then using the identity:

$$\text{Lam}(x) = \text{Zeta}(x) * (1 - 2^{-x})$$

which is good for all  $x$  except  $x = 1$  where  $\text{Lam}(x)$  is infinite.  $\text{Lam}(0) = 0$ .  
 $\text{Lam}(2) = \text{Pi}^2 / 8$ .  $\text{Lam}(x) = 0$  for all negative even integers.

$\text{LCM}(X, Y) = \text{Least Common Multiple } X, Y:$

Least common multiple function.  $\text{LCM}(x, y) = x * y / \text{GCD}(x, y)$ . Both the integer and fractional parts of  $x$  and  $y$  are used in the computation. For example,  $\text{LCM}(11.5, -16.1)$  is  $11.5 * (-16.1) / 2.3 = -80.5$ .

$\text{Legen}(X) = \text{Legendre polynomial of order Int}(X):$

The general form of a Legendre polynomial of order  $n$  is given by the sum:

$$\text{Legen}(n) = P(n) = \text{Sum } \{m=0 \text{ to } M\} [(-1)^m * (2^n - 2*m)! / (2^n * m! * (n - m)! * (n - 2*m)!)] * x^{(n-2*m)}$$

where  $M = n/2$  or  $(n-1)/2$ , whichever is an integer.

$P(n)$  has  $n+1$  terms  $\{p(n); p(n-1); \dots; p(1); p(0)\}$ .

```
p(n) = Fac(n+n) / (2^n * Fac(n)^2);
for (m = n - 2; m >= 0; m -= 2)
  p(m) = -p(m+2) * (m+2)*(m+1) / ((n-m)*(n+m+1));
for (m = n - 1; m >= 0; m -= 2)
  p(m) = 0;
```

The first few Legendre polynomials are:

```
P(0) = 1 = {1}
P(1) = x = {1; 0}
P(2) = (3*x^2 - 1)/2 = {1.5; 0; -0.5}
P(3) = (5*x^3 - 3*x)/2 = {2.5; 0; -1.5; 0}
P(4) = (35*x^4 - 30*x^2 + 3)/8 = {4.375; 0; -3.75; 0; +0.375}
P(5) = (63*x^5 - 70*x^3 + 15*x)/8 = {7.875; 0; -8.75; 0; +1.875; 0}
P(6) = (231*x^6 - 315*x^4 + 105*x^2 - 5)/16 = {14.4375; 0; -19.6875; 0; +6.5625; 0; -0.3125}
```

See:

<http://mathworld.wolfram.com/LegendrePolynomial.html>  
<http://hyperphysics.phy-astr.gsu.edu/hbase/math/legend.html>  
<http://gershwin.ens.fr/vdaniel/Doc-Locale/Cours-Mirrored/Methodes-Maths/white/math/s8/s8legd/s8legd.html>

$\text{Lerch}(X, S, A) = \text{LerchPhi}(X, S, A)$ , preferred method:

The  $\text{Lerch}(x, s, a)$  function or Lerch transcendent is

$$\text{Lerch}(x, s, a) = \text{Sum } \{k = 0, 1, \dots\} [x^k / |a + k|^s],$$

where any term with  $a + k = 0$  is excluded and  $|x| \leq 1$ .

When  $a = 1$ ,  $\text{Polylog}(s, x) = x * \text{Lerch}(x, s, 1)$ .

When  $s = 2$  and  $a = 1$ ,  $\text{Dilog}(x) = \text{Polylog}(2, x) = x * \text{Lerch}(x, 2, 1)$ .

When  $x = -1$  and  $a = 1/2$ ,  $\text{Beta}(s) = 2^{-s} * \text{Lerch}(-1, s, 1/2)$ .

When  $x > 0.5$ , the convergence acceleration technique called Combined Nonlinear-Condensation Transformation (CNCT) is used. When  $x < -0.5$ , the delta transform is used to speed convergence.

This function is actually defined by analytic continuation for values less than -1 and the method used to speed convergence will give a result at reduced accuracy for these values. For example:

```
Command: Lerch(-5, 2, 1)
```

```
Lerch: No convergence within the maximum number of iterations = 320
Lerch: Results at iteration 78 had a relative error approximately =
2.57506,44951,20027E-41 (16) [16]
Lerch: Iteration = 178
```

```
X = 5.49855,82521,21616,58005,11750,30752,53728,89904,73097,61E-1 (48) [56]
```

The correct answer is

```
X = 5.49855,82521,21616,58005,11750,30752,53728,89941,25006,55E-1,
```

that is gotten by increasing the precision to 80 decimal digits.

Lerch1(X, S, A) = 16-digit LerchPhi(X, S, A):

Same as Lerch(X, S, A) except that all operations are done in double precision to give a 16-digit result. This function can also get results at a reduced precision for  $x < -1$ .

Lerch2(X, S, A) = LerchPhi(X, S, A) by simple sum:

Same as Lerch(X, S, A) except that convergence acceleration is not used. This function cannot get results  $|x| > 1$ .

LerchT(X, S, A) = LerchPhiT(X, S, A), traditional:

The LerchPhiT(x, s, a) function or Lerch transcendent is

$$\text{PhiT}(x, s, a) = \text{Sum } \{k = 0, 1, \dots\} [x^k / (a + k)^s],$$

where  $|x| \leq 1$  and  $a \neq 0, -1, -2, \dots$

When  $a = 1$ ,  $\text{Polylog}(s, x) = x * \text{LerchT}(x, s, 1) = x * \text{Lerch}(x, s, 1)$ .

When  $s = 2$  and  $a = 1$ ,  $\text{Dilog}(x) = \text{Polylog}(2, x) = x * \text{LerchT}(x, 2, 1)$ .

When  $x = -1$  and  $a = 1/2$ ,  $= \text{Beta}(s) = 2^{-s} * \text{LerchT}(-1, s, 1/2)$ .

When  $x > 0.5$ , the convergence acceleration technique called Combined Nonlinear-Condensation Transformation (CNCT) is used. When  $x < -0.5$ , the delta transform is used to speed convergence. See:

<http://aksenov.freeshell.org/lerchphi/source/lerchphi.c> and  
<http://www.mpi-hd.mpg.de/personalhomes/ulj/jentschura/Documents/lphidoc.pdf> .

For example,  $\text{Lerch}(-0.5, 3, -4.5) = 2.32478,91995,91076,87837E-1$  and  
 $\text{LerchT}(-0.5, 3, -4.5) = -7.24071,26460,61940,35439E-1$ . The Lerch value agrees with Mathematica but is not the traditional value. See:

<http://functions.wolfram.com/webMathematica/FunctionEvaluation.jsp?name=LerchPhi&ptype=0&z=-0.5&s=3&a=-4.5&digits=21> and  
<http://mathworld.wolfram.com/LerchTranscendent.html> .

This function can also get results at a reduced precision for  $x < -1$ .



LerchT1(X, S, A) = 16-digit LerchPhiT(X, S, A):

Same as LerchT(X, S, A) except that all operations are done in double precision to give a 16-digit result. This function can also get results at a reduced precision for  $x < -1$ .

LerchT2(X, S, A) = LerchPhiT(X, S, A) by simple sum:

Same as LerchT(X, S, A) except that convergence acceleration is not used. This function cannot get results  $|x| > 1$ .

Li(X) = Logarithmic integral = Ei(Ln(X)):

Li(x) approximates the prime counting function Pi(x). It is computed using the Exponential integral Li(x) = Ei(Ln(x)). See the Ri(x) function for a better approximation of Pi(x).

Ln(X) = NaturalLog(X):

Evaluates to the Log base e of x, where e is the base of the natural logarithms, error if  $x \leq 0$ .

LnL(X) = NaturalLog(X + 1):

Evaluates to the Log base e of (x + 1), where e is the base of the natural logarithms, error if  $x \leq -1$ . This function is needed when an expression contains Ln(x + 1) and x can take on a value near zero. LnL(x) is accurate for values of x near zero.

Log(X) = LogBase10(X):

Evaluates to the Log base 10 of x, error if  $x \leq 0$ .  $\text{Log}(x) = \text{Ln}(x) / \text{Ln}(10)$ .

Lop(X) = ReducePrecision(X):

This function evaluates to X with its least significant super-digit removed. If rounding is turned on, the removed super-digit is used to round into the new least significant super-digit. In XMCalc numbers are normalized from both ends. If a calculation results in a number with some trailing zero super-digits, these super-digits are removed by reducing the count of the number of super-digits in the mantissa, and memory is reallocated. The Lop function can result in many super-digits being removed if removing one super-digit results in many trailing zeros.

Mag(X, Y) = SqRt(Sq(X), Sq(Y)):

Magnitude of (x, y) function. Used to find the Polar coordinates radius coordinate of the Cartesian coordinates (x, y).

Mat(R, C) = Return a zero matrix with R rows and C columns:

R and C are scalars and their integer part is used. All R \* C elements of the returned matrix are set to zero. If R or C < 1, the scalar zero is returned.

MatId(X) = Equals an X by X identity matrix:

Returns an identity matrix with X columns and X rows. The integer part of X is used and must be greater than zero.

Max(X, Y) = Greater of X and Y:

If  $x \geq y$ , the result is x, else the result is y.

MeanC(X, Y) = Common mean of X and Y = AGM(X, Y):

If  $x = 0$  or  $y = 0$ , MeanC(x, y) = 0. The common mean of two numbers  $G(0) > 0$ ,  $A(0) > 0$  is gotten by computing the geometric mean  $G(i+1) = \text{Sqrt}(G(i) * A(i))$  and the arithmetic mean  $A(i+1) = (G(i) + A(i)) / 2$ . As i increases,  $G(i) < A(i)$  converge to the common mean of  $G(0)$  and  $A(0)$ . The signs of x and y are not used by this function.

MEq(X) = Mersenne equation =  $2^X - 1$ :

If X is prime, MEq(X) is a Mersenne number. If MEq(X) is prime, MEq(X) is a Mersenne prime. See <http://www.geocities.com/hjsmithh/Perfect/Mersenne.html> .

Min(X, Y) = Lesser of X and Y:

If  $x \geq y$ , the result is y, else the result is x.

Mod(X, Y) =  $X - (\text{Floor}(X/Y) * Y)$ :

Modulo function.  $z = \text{Mod}(x, y) = x - (\text{Floor}(x/y) * y)$ . See the  $A = X \% Y$  operator.

Mord(A, N) = Multiplicative order of base A (mod N) or 0:

The multiplicative order function Mord(a, n) is the minimum positive integer e for which  $a^e \equiv 1 \pmod{n}$ , or zero if no e exists. If a or n is less than 2, zero is returned.

```
e = Mord(a, n) // e = Multiplicative order
// multiplicative order of base a (mod n) or zero if it does not exist. From
Henri Cohen's book,
// A Course in Computational Algebraic Number Theory, Springer, 1996, Algorithm
1.4.3, page 25.
```

```
{
  e = 0
  if (a <= 1 or n <= 1) return;
  if (GCD(a, n) != 1) return;
  h = Phi(n); // Euler's Totient Function
  factor h by ECM (Elliptic Curve Method)
  // h = Prod(pi^vi) i = 1, ..., k
  e = h;
  for (int i = 1; i <= k; i++)
  {
    e = e / pi^vi;
    g1 = a^e mod n
    while (g1 != 1)
    {
      g1 = g1^pi mod n
```

```

    e = e * pi
  }
}
}

```

MPG(X) = The X'th Mersenne Prime Generator, MPG(1) = 3:

If  $1 \leq X \leq 46$ ,  $m = \text{MPG}(X)$  is the X'th Mersenne prime.

MPP(X) = Mersenne Prime Power, MPP(1) = 2:

If  $1 \leq X \leq 46$ ,  $p = \text{MPP}(X)$  is the power that makes  $2^p - 1$  the X'th Mersenne prime. MPP(1) = 2, MPP(46) = 43112609. MEq(MPP(X) is the X'th Mersenne prime. PEq(MPP(X) is the X'th perfect number.

MPrime(X) = 1 (True) if  $2^X - 1$  is a Mersenne Prime else 0:

This uses the Lucas-Lehmer-Test to determine if  $2^x - 1$  is a Mersenne Prime.

```

boolean b = MPrime(x) // this is the Lucas-Lehmer-Test
{
  if (x == 2) return true;
  if (x < 2 || x is not prime) return false;
  m = 2^x - 1;
  u = 4;
  for (i = 3; i <= x; i++)
  {
    u = (u*u - 2) mod m;
  }
  if (u == 0)
    return true;
  else
    return false
}

```

MToQ(M) = Quaternion equivalent of 3 by 3 matrix M:

Meant to converts the rotation matrix M to a normalized quaternion. If M is not a rotation matrix, results can vary.

A rotation matrix M is a proper orthogonal matrix ( $\text{Det}(M) = 1$ , not -1) and  $M^* \text{Tran}(M) = \text{Identity matrix}$ . It can be used to rotate vectors in 3D space.

MToR(M) = Rotation vector from 3 by 3 matrix M:

Meant to converts the rotation matrix to a normalized rotation vector. If M is not a rotation matrix, results can vary. See the MToQ(M) function.

A rotation matrix or a normalized quaternion can be converted to a 4D Rotation vector R containing its rotation axis direction cosines (Eigen vector) and its rotation angle (Eigen angle)  $R = \{a; b; c; d\}$ .  $\{a; b; c\}$  is the Eigen vector and d is the Eigen angle.

Mu(X) = Moebius Mu(X) function:

The Moebius or Möbius  $\mu(n)$  function is a number theoretic function defined by:

$\text{Mu}(n) = 0$  if  $n$  has one or more repeated prime factors  
 $\text{Mu}(n) = 1$  if  $n = 1$   
 $\text{Mu}(n) = (-1)^k$  if  $n$  is the product of  $k$  distinct primes  
 $\text{Mu}(n) = \text{Mu}(-n)$  ( $\text{Mu}(n)$  is not normally defined for  $n < 1$ )  
 $\text{Mu}(0) = 0$

so  $\text{Mu}(n) = 0$  iff  $n$  is not square-free. If  $n$  is square-free, the sign of  $\text{Mu}(n)$  tell whether there is even or odd number of distinct prime factors (+1 for even, -1 for odd). For  $n = 0, 1, 2, \dots$  the first few values are 0, 1, -1, -1, 0, -1, 1, -1, 0, 0, 1, -1, 0, ... .

$\text{MulQ}(X, Y) =$  Quaternion multiply,  $X * Y$ :

Standard quaternion multiply,  $X * Y$ .

$\text{NormC}(X, M, S) =$  Normal Cumulative distribution function (cdf):

Computes the cumulative distribution function (cdf) of the normal distribution with mean  $M$  and standard deviation  $S$ . This is defined as the integral  $\{-\infty, x\} [\exp(-((t-m)^2)/(2*s^2))/(s*\text{Sqrt}(2*\text{Pi})] dt$ . It is computed by:

$$\text{NormC}(x, m, s) = (1/2) * \text{ErfC}((m - x)/(s*\text{Sqrt}(2))).$$

The ErfC function is used to give accurate results when  $x \ll m$  where NormC is very small.

If  $s$  is zero, the answer is zero if  $x < m$ , one if  $x > m$ , and 0.5 if  $x = m$ .

If  $s < 0$ , the complement is given,  $\text{NormC}(x, m, -s) = 1 - \text{NormC}(x, m, s)$ .

$\text{NormQ}(Q) =$  Quaternion  $Q$  normalized:

Returns quaternion  $Q / \text{NormV}(Q)$ .

$\text{NormR}(R) =$  Rotation vector  $R$  normalized:

Returns Rotation vector  $R$  with  $R1^2 + R2^2 + R3^2 = 1$  and  $R4$  between 0 and  $2*\text{Pi}$ . See the  $\text{MToR}(M)$  function.

$\text{NormS}(X) =$  Standard Normal Cumulative distribution function:

Computes the cumulative distribution function (cdf) of the normal distribution with mean 0 and standard deviation 1. This is defined as the integral  $\{-\infty, X\} [\exp(-t^2/2)/\text{Sqrt}(2*\text{Pi})] dt$ . It is computed by:

$$\text{NormS}(x) = \text{NormC}(x, 0, 1).$$

$\text{NormV}(V) =$  Norm of vector  $V$ , its length:

Return  $\text{Sqrt}(v1^2 + v2^2 + v3^2 + \dots)$ .  $V$  can be a row or column vector.

$P(X) =$  The  $X$ 'th prime:

$P(n)$  is the  $n$ 'th prime.  $P(1) = 2, P(2) = 3, \dots$ .  $P(0)$  is defined to be 0. This is the inverse of the function  $\text{Pi}(x)$ . In other words,  $P(n)$  is the smallest  $x$  for which  $\text{Pi}(x) = n$ .

Pascal(X) = Equals an X by X Pascal triangle,  $P_{ij} = \text{Bino}(i,j)$ :

The upper triangle above the diagonal will be all zeros.

PEq(X) = Perfect equation =  $(2^X - 1) * 2^{(X-1)}$ :

If MEq(X) is prime, MEq(X) is a Mersenne prime and PEq(X) is an even perfect number. All even perfect numbers are of this form. See <http://www.geocities.com/hjsmithh/Perfect/Mersenne.html> .

PGT(X) = First prime > X:

What more can I say.  $PGT(0) = PGT(1) = 2$ ,  $PGT(2) = 3$ ,  $PGT(3) = 5$ , ... . If  $x < 0$  its sign is ignored.  $PGT(-\text{Abs}(x)) = PGT(\text{Abs}(x))$ . The Adleman function is used to determine if a number is prime.

Phi(X) = Euler's totient function:

Phi, Sig (sigma) and Tau are number theoretic functions.  $\Phi(n)$  = number of positive integers not exceeding n and relatively prime to n. For example,  $\Phi(60) = 16$ . The 16 integers are 1, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 49, 53, and 59.  $\Phi(0)$  is defined to be 0.  $\Phi(1) = 1$ .  $\Phi(2) = 1$ .  $\Phi(3) = 2$ .

The classic formula for Phi is:

$$\Phi(n) = n * (1-1/P_1) * \dots * (1-1/P_r)$$

but for computation the following formula is used

$$\Phi(n) = ((P_1-1)*P_1^{(A_1-1)} * \dots * (P_r-1)*P_r^{(A_r-1)})$$

For this document the following notation is used. An integral number  $n > 1$  can be written as  $n = P_1^{A_1} * P_2^{A_2} * \dots * P_r^{A_r}$  where the  $P_i$ 's are the various different prime factors,  $A_i$  the number of times  $P_i$  occurs in the prime factorization and r the number of prime factors.

PhiL(X, A) = Legendre's formula:

PhiL(x, a) = Legendre's formula i.e., the number of integers in [1, x] not divisible by any of the first a primes. This function is used by the prime counting functions  $\Pi(x)$ ,  $\Pi_L(x)$ ,  $\Pi_{L1}(x)$ , and  $\Pi_M(x)$ . For values of A larger than about 13000, this function is prone to a Run-time error of "System.StackOverflowException was thrown", but this will not crash the program.

Pi(X) = Number of primes  $\leq X$  by sieve or Lehmer:

This is the prime counting function, for every real  $x \geq 0$ ,  $\Pi(x)$  is the number of primes p such that  $p \leq x$ . For example  $\Pi(5) = \Pi(6) = 3$ . The 3 primes are 2, 3, and 5. For small x, the sieve of Eratosthenes is used, for large x, Lehmer's formula is used. This is the method to use unless you are testing  $\Pi_{L1}(x)$  (slowest),  $\Pi_M(x)$  (slow), or  $\Pi_L(x)$  (fastest for large x).

PiL(X) = number of primes  $\leq X$  by Lehmer's formula:

This is the prime counting function using Lehmer's formula. This is the fastest of the three functions  $\text{PiL1}(x)$ ,  $\text{PiM}(x)$ , and  $\text{PiL}(x)$ .

$\text{PiL1}(X)$  = number of primes  $\leq X$  by Legendre's formula:

This is the prime counting function using Legendre's formula.

$\text{PiM}(X)$  = number of primes  $\leq X$  by Meissel's formula:

This is the prime counting function using Meissel's formula.

$\text{PLT}(X)$  = Largest prime  $< X$ :

Finds the next prime  $P < x$ . By definition,  $\text{PLT}(0) = \text{PLT}(1) = \text{PLT}(2) = 0$  (False). The sign of the input  $x$  is ignored. The running time of this and the other commands that require prime factoring can be quite long if a number having large factors is encountered. The Adleman function is used to determine if a number is prime.

$\text{PNG}(X)$  = The  $X$ 'th Perfect Number Generator,  $\text{PNG}(1) = 6$ :

If  $1 \leq X \leq 46$ ,  $p = \text{PNG}(X)$  is the  $X$ 'th perfect number.

$\text{PolAdd}(X, Y)$  = Polynomial  $X + Y$ :

For example,  $\text{PolAdd}(\{4; 3; 2; 1\}, \{5; 4; 2\}) = \{4; 8; 6; 3\}$ . It adds from the bottom up and the size does not have to be the same.

$\text{PolDisc}(P)$  = Discriminant of polynomial  $P$ :

The discriminant of the quadratic  $\{a; b; c\}$  is  $b^2 - 4*a*c$ .  
The discriminant of the cubic  $\{a; b; c; d\}$  is  $b^2*c^2 - 4*b^3*d - 4*a*c^3 + 18*a*b*c*d - 27*a^2*d^2$ .

See: <http://en.wikipedia.org/wiki/Discriminant> from Wikipedia encyclopedia  
and: <http://mathworld.wolfram.com/PolynomialDiscriminant.html> from MathWorld.

$\text{PolDiv}(X, Y)$  = Polynomial  $X / Y$ , "Re" = remainder:

See: <http://faculty.ed.umuc.edu/~swalsh/Math%20Articles/Polynomial.html> ... and Divide Polynomials -- from Shelley Walsh, University of Maryland.

$\text{PolEval}(P, X)$  = Polynomial  $P$  evaluated at  $X$ :

$P$  is a column vector representing a polynomial and  $X$  is a scalar. If  $P = \{p_n; p_{(n-1)}; \dots; p_2; p_1; p_0\}$ ,  $\text{PolEval}(P, X) = ((\dots((p_n*X + p_{(n-1)})*X + p_{(n-2)})*X + \dots + p_2)*X + p_1)*X + p_0$ .

$\text{PolMod}(X, Y)$  = Returns "Re" = remainder of polynomial  $X / Y$ :

See  $\text{PolDiv}(X, Y)$ . Example:  $\text{PolDiv}(\{1; 2; 3; 4; 5; 6\}, \{1; 2; 3\}) = \{1; 0; 0; 4\}$  with  $\text{Re} = \{-3; -6\}$ .

PolMonic(P) = Monic polynomial of P, leading coeff. = 1 or {0}:

Divides the polynomial P by its leading coefficient. If P = {0} it is left unchanged.

PolMul(X, Y) = Polynomial X \* Y:

Example: PolMul({1; 2; 3; 4}, {3; 2; 1}) = {3; 8; 14; 20; 11; 4}. It is a convolution, a multiply without carry.

See: <http://faculty.ed.umuc.edu/~swalsh/Math%20Articles/Polynomial.html> ... Multiply, and Divide Polynomials -- from Shelley Walsh, University of Maryland.

PolNorm(P) = Norm of polynomial P, leading coeff. not 0 or {0}:

If the leading coefficient of P is zero, it is reduced by one degree until the leading coefficient is not zero or until it is equal to {0}.

PolRecip(P) = Reciprocal polynomial of P,  $x^{\deg(P)} * P(1/x)$  :

The coefficients are reversed from most to least. For example {1; 2; 3; 4} becomes {4; 3; 2; 1}.

PolSturm(P) = Number of real roots in real polynomial P:

P is a polynomial with real coefficients and must be square free for the function to work. An alarm is given if P is not square free.

```
P = PolMonic(P)
a = 0
for (int r = 1; r <= P.m; r++)
    a = a + Abs(p[r])
return PolSturmI(p, -a, a)
```

PolSturmI(P, X, Y) = Number of real roots in P in interval (X, Y) :

Same as PolSturm(P), but only checks for roots between X and Y. P is a polynomial with real coefficients and must be square free for the function to work. **Roots at X or Y may or may not be counted.** An alarm is given if P is not square free or X and Y are not scalars.

See: <http://www.math.niu.edu/~rusin/known-math/94/sturm.seq> from The Mathematical Atlas

See: <http://www.math.niu.edu/~rusin/known-math/96/sturm> from The Mathematical Atlas

PolSub(X, Y) = Polynomial X - Y:

For example, PolSub({4; 3; 2; 1}, {5; 4; 2}) = {4; -2; -2; -1}. It subtracts from the bottom up and the size does not have to be the same.

See: <http://faculty.ed.umuc.edu/~swalsh/Math%20Articles/Polynomial.html> .

Polylog(S, X) = Polylogarithm(S, X):

The Polylog(s, x) function is equal to  $x * \text{Lerch}(x, s, 1)$ . See the Lerch(X, S, A) function. When s = 2, this is the Dilog(x) function.

Pow(X, Y) = X to the power Y:

The Exponential function. This function operates differently depending on whether y is an exact integer. If y is an exact integer, the peasants' method is used in which up to  $2 * \text{Log base 2 of } y$  multiplies of powers of x are done to compute the result. If y is not an exact integer, the result is computed by  $\text{Exp}(y * \text{Ln}(x))$ . An error message is generated in two cases: 1) x is  $< 0$  and y is not an integer. 2)  $x = 0$  and y is  $< 0$ . If  $x = 0$  and  $y = 0$ , an answer of 1.0 will be given.

I have speeded up  $x=a^b$  when  $a=10$  or  $a=-10$  and b is an integer, especially for large precision. Also works for Pow(a, b) but not PowM(a, b)

PowM(X, Y) = (X to the power Y) Mod FMB:

The Exponential function with modulo arithmetic. This function operates differently depending on whether y and FMB are an exact integer. If y and FMB are exact integers, the peasants' method is used in which up to  $2 * \text{Log base 2 of } y$  multiplies of powers of x are done to compute the result. The Modulo process is performed after each multiply to prevent the intermediate results from becoming large.

If y and FMB are not both an exact integer, the result is computed by  $\text{Exp}(y * \text{Ln}(|x|)) \text{ Mod FMB}$ . If FMB is not on the list, it is added to the list with a value of zero. If FMB is zero, the Modulo is not performed. An error message is generated in two cases: 1) x is  $< 0$  and y is not an integer. 2)  $x = 0$  and y is  $< 0$ . If  $x = 0$  and  $y = 0$ , an answer of 1.0 will be given. See the % primitive.

Prime(X) = 1 (True) if X is Prime else 0:

If x is a prime number then Prime(x) = 1 (True). If x is not prime, Prime(x) = 0 (False).

Primo(X) = Primorial function, product of all primes  $\leq X$ :

The primorial function is a cross between the prime counting function and the factorial function. For every real  $x \geq 0$ , Primo(x) is the product of all primes p such that  $p \leq x$ . For example  $\text{Primo}(5) = \text{Primo}(6) = 2*3*5 = 30$ .  $\text{Primo}(0) = \text{Primo}(1) = 1$ . The primorial function is undefined for  $x < 0$ .

PrimR(X) = Largest and smallest primitive root of X or 0:

The number of primitive roots (pr's) that x has is displayed. If x does not have any pr's, zero is returned. If x has only one pr, it is returned. If x has 2 or more pr's, the largest and smallest are displayed and the smallest is returned. The named number "Re" is set to the largest primitive root or 0 if x does not have any pr's. For example:

Command: X = PrimR(41)

The prime number 41  
has 16 primitive roots, the largest is  $35 = n - 6$ , the smallest is 6

X = 6

Command: Re=



Re = 35

Command: X = PrimR(137842)

The non-prime number  $1,37842 = 2^1 * 41^3$   
has 26240 primitive roots, the largest is  $1,37835 = n - 7$ , the smallest is 7

X = 7

Command: Re=

Re = 1,37835

PrimRP(X) = Largest and smallest prime primitive root of X:

The number of primitive roots (pr's) that x has is displayed. If x does not have any prime pr's, zero is returned. If x has only one prime pr, it is returned. If x has 2 or more prime pr's, the largest and smallest prime pr are displayed and the smallest is returned. The named number "Re" is set to the largest prime primitive root or 0 if x does not have any prime pr's.

PrimRQ(X, Y) = 1 (True) if X is a primitive root of Y, else 0:

This function answers the question: Is x a primitive root of |y|?

PrinQ(Q) = Principal equivalent quaternion of quaternion Q:

Returns Q or -Q with last non-zero element > 0, or {0; 0; 0; 0} if all elements are zero.

Psi(X) = DigammaFunction(X):

Psi(x) = logarithmic derivative of the gamma function  $d/dx(\ln(\Gamma(x))) = \Gamma'(x) / \Gamma(x)$ . For  $x \geq -5$ , the asymptotic formula with Bernoulli numbers is used to compute this. If x is an integer  $\leq$  zero, an alarm is displayed. For non-integer negative  $x < -5$ , the reflection formula is used:

$$\Psi(x) = \Psi(y) + \pi / (\tan(\pi * y)), \text{ where } y = 1 - x.$$

For large x,

$$\Psi(x) \approx \ln(x) - 1/2*x - \sum_{k=1,2,\dots} [\text{Bern}(2*k) / (2*k*x^{(2*k)})],$$

where Bern(2\*k) are Bernoulli numbers. For small x,

$$\Psi(x) = \Psi(n + x) - 1/(n-1+x) - 1/(n-2+x) - \dots - 1/(1+x) - 1/x.$$

This is based on  $\Psi(x) = \Psi(x+1) - 1/x$ . n is computed by a heuristic,  $n = \max((\text{int})((\text{digits}^{1.5}) / 13.0 + 1 - x), 0)$ , where digits is the current decimal digits in a computed mantissa.

QToM(Q) = Matrix equivalent of quaternion Q:

Meant to convert a normalized quaternion to a rotation matrix. If Q is not a normalized quaternion, results can vary.

QToR(Q) = Rotation vector equivalent of quaternion Q:

Meant to convert a normalized quaternion to a normalized rotation vector. If Q is not a normalized quaternion, results can vary. See the MToR(M) function.

QuadR(a, b, c) = Roots of  $a*x^2 + b*x + c = 0$ , sets X1 and X2:

Computes the two real roots (if they exist) of the quadratic equation  $a*x^2 + b*x + c = 0$ . The items X1 and X2 are put on the list, if not already there, and are set to the two roots. The root X2 is returned. This is done by:

```
z = QuadR(a, b, c) // Roots of  $a*x^2 + b*x + c = 0$ 
{
  if (a == 0 && b == 0)
  {
    if (c == 0)
      Write("QuadR: No unique solutions");
    else
      Write("QuadR: No solutions");
    return 0;
  }
  if (a == 0)
  {
    X1 = X2 = -c/b;
    return X2;
  }
  MultiF d = new MultiF();
  d = b^2 - 4*a*c;
  if (d < 0)
  {
    Write("QuadR: No real solutions");
    return 0;
  }
  d = d^(1/2);
  if (b < 0)
    d = b - d;
  else
    d += b;
  if (d == 0)
  {
    X1 = X2 = 0;
  }
  else
  {
    d *= -0.5;
    X1 = d/a; // X1 = (-b - (b^2 - 4*a*c)^(1/2)) / (2 * a)
    X2 = c/d; // X2 = (2 * c) / (-b - (b^2 - 4*a*c)^(1/2))
  }
  return X2;
}
```

Rank(M) = Rank of matrix M:

The number of independent rows or columns of M. A nonsingular n by n matrix has a rank of n. This uses RREF(M) to compute the Reduced Row Echelon Form of matrix M and returns the number of non-zero rows.

See: <http://mathworld.wolfram.com/MatrixRank.html> from MathWorld.

Rev(X) = Digit Reversal of X base 10:

The decimal digits of  $x$  are reversed MSD to LSD.  $\text{Rev}(1234) = 4321$ .

$\text{Rev}(X, Y) = \text{Digit Reversal of } X \text{ base } Y$ :

The base- $y$  digits of  $x$  are reversed MSD to LSD.  $\text{Rev}(1234, 5) = 2746$  because  $1234 = 14414$  (base 5) and  $41441$  (base 5) = 2746. If  $y < 2$ , base 10 is used.  $\text{Rev}(X, 0) = \text{Rev}(X, 10) = \text{Rev}(X)$ .

$\text{Ri}(X) = \text{Riemann prime counting function}$ :

$\text{Ri}(x)$  approximates the prime counting function  $\text{Pi}(x)$ . It is computed using the Gram series  $\text{Ri}(x) = 1 + \text{Sum}\{n = 1, \dots\} (\ln(x)^n / (n*n!*Zeta(n+1)))$ .

$\text{RInt}(X, Y) = \text{RandomInteger between Int}(X) \text{ and Int}(Y) \text{ inclusive}$ :

Generates a random integer  $z$ ,  $x \leq z \leq y$  or  $y \leq z \leq x$  if  $x > y$ . Repeat this function,  $F4$ , to generate a sequence of random integers. Only the integer part of  $x$  and  $y$  are used. You should not use  $\text{RN}$  in the command, like  $\text{RN}=\text{RInt}(\text{RN}, \text{RN})$ .

$\text{RN}(X) = \text{RandomNumber}(X=\text{Seed or matrix})$ :

Random number function.  $\text{RN}(X)$  evaluates to a random number between zero and 1.0. This number will never have more than 35 decimal digits. Theoretically the random number generator will cycle after  $10^{**} 35$  numbers, but the earth will not last that long. The fractional part of the argument of the function is taken as the seed of the random number generator. For a consecutive set of random numbers, the argument  $x$  should be the previous random number generated. The items  $\text{RNA}$ ,  $\text{RNC}$ , and  $\text{RN}$  are put on the list by the random number function. The equation used is:  $\text{RN}(x) = (\text{RNA} * x * 10^{35} + \text{RNC}) \text{ mod } (10^{35}) / 10^{35}$ , where  $\text{RNA}$  and  $\text{RNC}$  are 35 digit integers.  $\text{RN}$  is put on the list so it will be used as the seed the next time the  $\text{N}$  command is used.

$\text{RNA} = 18436248305725075346374291920765341,$   
 $\text{RNC} = 86346479732047945672382544639625443.$

If  $X$  is a matrix, the scalar "RN" on the list is used as the first seed and the result is a matrix the same size as  $X$  with each element the next generated random number.

$\text{RotateM}(M, V) = 3\text{-D vector } V \text{ rotated by matrix } M$ :

Meant to rotate a 3-D vector by a rotation matrix. If  $M$  is not a rotation matrix, the result is  $M * V$ . See the  $\text{MToR}(M)$  function.

$\text{RotateQ}(Q, V) = 3\text{-D vector } V \text{ rotated by quaternion } Q$ :

Meant to rotate a 3-D vector by a normalized quaternion. If  $Q$  is not a normalized quaternion, results can vary.

$\text{Round}(X) = \text{Integer nearest to } X$ :

$\text{Round}(2.5) = 3.0$ .  $\text{Round}(-2.5) = -3.0$ .  $\text{Round}(3.0) = 3.0$ .

$\text{RPYM}(R, P, Y) = \text{Rotation matrix, } R = \text{roll, } P = \text{pitch, } Y = \text{yaw}$ :

Returns rotation matrix that does a roll then a pitch and then a yaw.

```
rollMatrix = {1, 0, 0; 0, cos(R), sin(R); 0, -sin(R), cos(R)}  
pitchMatrix = {cos(P), 0, -sin(P); 0, 1, 0; sin(P), 0, cos(P)}  
yawMatrix = {cos(Y), sin(Y), 0; 0, -sin(Y), cos(Y), 0; 0, 0, 1}  
Rotation matrix = yawMatrix * pitchMatrix * rollMatrix.
```

RREF(M) = Return Reduced Row Echelon Form of matrix M:

See: <http://mathworld.wolfram.com/EchelonForm.html> Definition of Reduced Row Echelon Form from MathWorld

See:

[http://www.krellinst.org/AiS/textbook/unit2/example\\_projects/starter/math/matrix/gauss.html](http://www.krellinst.org/AiS/textbook/unit2/example_projects/starter/math/matrix/gauss.html) Gauss - Jordan Elimination from KRELL Institute

See: [http://en.wikipedia.org/wiki/Gaussian\\_elimination](http://en.wikipedia.org/wiki/Gaussian_elimination)

Gaussian elimination Pseudocode from Wikipedia

RREFa(M, V) = RREF solution U to augmented matrix [M,V],  $M*U=V$ :

Uses RREF of matrix M augmented with V to compute U such that  $M * U = V$ . V must have the same number of rows as M and can have 1 or more columns. If V has one column and M is a non-singular square matrix, this produces the same result as Solve(M, V).

RToM(R) = 3 by 3 matrix from rotation vector R:

Meant to convert a normalized rotation vector to a rotation matrix. If R is not a normalized rotation vector, results can vary. See the MToR(M) function.

RToQ(R) = Quaternion from rotation vector R:

Meant to convert a normalized rotation vector to a normalized quaternion. If R is not a normalized rotation vector, results can vary. See the MToR(M) function.

Sec(X) = Secant(X):

$Sec(X) = 1/Cos(X)$ . Error if  $Cos(X) = 0$ .

Sech(X) = HyperbolicSecant(X):

$Sech(X) = 1/Cosh(X)$ . Cosh(X) is never zero.

Set(M, R, C, Y) = Return M with element at row R, column C set to Y:

R and C are scalars and their integer part is used. Y is also a scalar. M is a matrix with at least R rows and C columns.  $Z = Set(M, R, C, Y)$  will not change M.

SetAll(M, Y) = Return matrix M with all element set to Y

Y is a scalar and M is a matrix.  $Z = SetAll(M, Y)$  will not change M.

Sig(X) = Sum of divisors of X:

Sig(n) = sum of the positive integral divisors of n. Sig is short for sigma the 18'th letter of the Greek alphabet. Sig(0) = 0, Sig(1) = 1, Sig(2) = 3.

The formula for this is:

$$\text{Sig}(n) = ((P_1^{(A_1+1)} - 1) / (P_1 - 1)) * \dots * ((P_r^{(A_r+1)} - 1) / (P_r - 1))$$

Sig0(X) = Sum of divisors of X (-X):

Sig0(n) = sum of the positive integral divisors of n, not including n. Sig0(n) = Sig(n) - n. Sig(n) is the classical number theoretic function and Sig0 is handy for check perfect numbers. A number n is a perfect number iff  $n > 0$  and Sig0(n) = n. Sig0(0) = Sig0(1) = 0, Sig0(2) = 1, ..., Sig0(6) = 6. Six is the first perfect number.

Sign(X) = 0 if X=0, else = X / |X|:

Sign(x) = -1.0 if x < 0, = 0.0 if x = 0, = +1.0 if x > 0.

Sin(X) = Sine(X):

Trigonometric Sine function, error if |x| is very large.

Sinh(X) = HyperbolicSine(X):

Hyperbolic Sine function.

Size(X) = {r, c} where r = rows in X, c = columns in X:

Size(X) is a two element row vector {r, c} with r = the number of rows in X and c = the number of columns in X. If X is a scalar, the result is zero.

Solve(M, V) = Solve for vector U where square matrix M \* U = V:

Uses the LU decomposition of M to solve the classical problem of n linear equations and n unknowns.

See: <http://www.library.cornell.edu/nr/bookcpdf/c2-3.pdf> LU Decomposition and Its Applications from Numerical Recipes Home Page.

Solve(X, Y, N) = Solve for z, X \* z == Y Mod N:

Returns z equal to the minimum solution of the modular linear equation  $x * z == y \text{ Mod } n$  or zero if no solution exists. A solution exists iff GCD(x, n) is a divisor of y. If there is more than one solution, all solutions are displayed. The absolute value of N is used. For example,

Command: z = solve(35, 10, 20)

Solution 5: 14  
Solution 4: 10  
Solution 3: 6  
Solution 2: 2  
Solution 1: 18

$z = 2$

Command:  $z = \text{solve}(35, 1, 20$

Solution: No solution

$z = 0$  (False)

SolveH(M) = Solve for vectors  $U \neq 0$  where matrix  $M * U = 0$ :

Solves the homogeneous equation  $M * U = 0$  for all non trivial solutions by returning a list of column vectors forming a [vector space basis](#) for the null space of M. If no solutions exist, the scalar zero is returned. This uses the Singular Value Decomposition SVD of M. This function is also called the kernel of M. The vector U is a solution iff it is a linear combinations of the returned column vectors.

See: <http://www.math.lsa.umich.edu/~hochster/419/ker.im.html> FINDING A BASIS FOR THE KERNEL OR IMAGE from Mel Hochster, University of Michigan

SolveS(M, V) = Solve for minimum vector U where matrix  $M * U = V$ :

Uses the Singular Value Decomposition SVD method and can solve linear equations with a singular matrix M. There can be more, the same, or fewer equations than unknowns. If M is singular, the solution with the smallest length is given. The output also contains information like:

**Singular, Rank = 2, Residual = 0.0 (False)**

The residual will be the length of the vector  $r = M * U - V$ . The solution minimizes this residue. If the residual is not zero, this is a least square solution. The (False) just says there is no residual.

Sord(A, N) = Multiplicative suborder of base A (mod N) or 0:

The multiplicative suborder function Sord(a, n) is the minimum positive integer e for which  $a^e \equiv \pm 1 \pmod{n}$ , or zero if no e exists. If A or N is less than 2, zero is returned.

```
e = Sord(a, n) // e = Multiplicative Suborder
// multiplicative suborder of base a (mod n) or zero if it does not exist.
// From Stephen Wolfram, Algebraic Properties of Cellular Automata (1984).
// Sord = Mord or Mord/2, Mord/2 iff  $a^{(Mord/2)} \pmod{n} = -1$ 
{
  e = Mord(a, n)
  if ((e is odd) or e == 0), return
  e = e/2
  g1 =  $a^e \pmod{n}$ 
  if (g1 != (n-1)) e = 2*e
}
```

SortC(X) = Sort each column of matrix X in numerical order:

If x is a scalar or a row vector, it is left unchanged.

SortCS(X, Y, Z) = Sort each column of matrix X, row Y thru Z:

Same as SortC(X) except only rows Y through Z are included in the sort. If Y is a matrix or less than 1, it considered equal to 1. If Y is larger than the number of rows in X, it is considered equal to the number of rows. If Z is a matrix or larger than the number of rows in X, it considered equal to the number of rows. If Z is less than 1, it is considered equal to 1. If  $Z < Y$ , rows Z through Y are in the sort.

SortR(X) = Sort each row of matrix X in numerical order:

If x is a scalar or a column vector, it is left unchanged.

SortRS(X, Y, Z) = Sort each row of matrix X, column Y thru Z:

Same as SortR(X) except only columns Y through Z are included in the sort. If Y is a matrix or less than 1, it considered equal to 1. If Y is larger than the number of columns in X, it is considered equal to the number of columns. If Z is a matrix or larger than the number of columns in X, it considered equal to the number of columns. If Z is less than 1, it is considered equal to 1. If  $Z < Y$ , columns Z through Y are in the sort.

Sq(X) = X Squared:

The square function, x times x.

SqFree(X) = 1 (True) if X is a squarefree, else 0:

Set equal to 1 (True) if x is squarefree, else it is set to 0 (False). This is computed by factoring x and examining the powers of the prime factors. x is squarefree iff all of the powers are One. The first prime found with a power > 1 causes the factoring to stop, and an answer of 0 (False) is given.

SqRt(X) = SquareRoot(X):

The positive square root function, error if  $x < 0$ .

SqRtRem(X) = Floor(SquareRoot(X)) and set Re to remainder:

Z = Integer square root = Floor(SquareRoot(X)). Re =  $x - z^2$ . If Re is on top of the list, an error message is displayed:

**"Cannot set same location to both SqRt and remainder, continuing..."**

SumD(X, Y) = Sum of base Y digits in X,  $Y \geq 2$ :

For example,  $\text{SumD}(5, 2) = 1 + 0 + 1 = 2$  ( $5 = 101$  base 2). The signs of X and Y are not used. If  $|Y| < 2$ , Y is changed to 10 and a message is generated. Y can be larger than 36. If  $Y == X$ , the answer is 1. If  $Y > X$ , the answer is X.

SumDD(X) = Sum of decimal digits in X:

For example,  $\text{SumDD}(427) = 4 + 2 + 7 = 13$ .

SVD(M) = Singular Value Decomposition,  $M = u * w * v^T$ :

Singular Value Decomposition,  $M = U * W * \text{Tran}(V)$ .  $W$  is a diagonal matrix and is returned as a row vector. The number of rows in  $M$  should be  $\geq$  the number of columns in  $M$ , if smaller,  $M$  will be filled with zero rows. This code is adapted from Numerical Recipes in C. This function also stores  $U$  in "MatU",  $W$  in "MatW", and  $V$  in "MatV" on the list. The elements of  $W$  will always be non-negative and sorted in ascending order.

See: <http://www.library.cornell.edu/nr/bookcpdf/c2-6.pdf> Numerical Recipes Books On-Line.

$\text{Tan}(X) = \text{Tangent}(X)$  :

Trigonometric Tangent function, error if  $|x|$  is very large. It is also an error if  $x$  is equivalent to plus or minus 90 degrees.

$\text{Tanh}(X) = \text{HyperbolicTangent}(X)$  :

Hyperbolic Tangent function.

$\text{Tau}(X) = \text{Number of divisors of } X$  :

$\text{Tau}(n)$  = number of positive integral divisors of  $n$ .  $\text{Tau}(0) = 0$ ,  $\text{Tau}(1) = 1$ . The formula for  $\text{Tau}$  is:

$$\text{Tau}(n) = (A_1+1) * \dots * (A_k+1)$$

When the command  $\text{AllD}(X)$ ,  $\text{IsSq}(X)$ ,  $\text{Mu}(X)$ ,  $\text{PFA}(X)$ ,  $\text{PFE}$ ,  $\text{Phi}(X)$ ,  $\text{Sig}(X)$ ,  $\text{Sig0}(X)$ ,  $\text{SqFree}(X)$ , or  $\text{Tau}(X)$  completes the factorization of  $x$ , this factorization is saved. When one of these commands is executed with  $x = 0$  and there is a saved factorization, the saved factors and exponents are used to complete the command. This can save a lot of time when more than one of these commands are needed for the same number. To see the saved factorization, try  $\text{PFE}(0)$ .

Executing one of these commands with  $x = 1$  will delete the saved factorization. The commands  $\text{Prime}(X)$ ,  $\text{PFB}(X)$ ,  $\text{PGT}(X)$ , and  $\text{PLT}(X)$  will also delete any saved factorization. The prime related commands  $\text{Pi}(X)$ ,  $\text{P}(X)$ , and  $\text{PTab}(X)$  have no effect on the saved factorization.

$\text{ToDeg}(X) = \text{RadiansToDegrees}(X)$  :

Converts radians to degrees. Evaluates to  $x$  multiplied by  $180/\text{Pi}$ .

$\text{ToRad}(X) = \text{DegreesToRadians}(X)$  :

Converts degrees to radians. Evaluates to  $x$  multiplied by  $\text{Pi}/180$ .

$\text{Trace}(M) = \text{Sum of elements on the principal diagonal of } M$  :

$\text{Trace}(M) = m_{11} + m_{22} + m_{33} + \dots$  .  $M$  must be a matrix but it does not need to be square.

$\text{Tran}(M) = \text{Transpose of matrix } M$  :

If  $M$  is an  $r$  by  $c$  matrix, the result will be a  $c$  by  $r$  matrix with the  $j$ ,  $i$  element equal to the  $i$ ,  $j$  element of  $M$ .



Vander(V) = Vandermonde's matrix from vector V:

Let V be a row or column vector with elements v1, v2, v3, ..., vn. Vandermonde's matrix of V is

$$\begin{Bmatrix} 1, & v_1, & v_1^2, & \dots, & v_1^{(n-1)}; \\ 1, & v_2, & v_2^2, & \dots, & v_2^{(n-1)}; \\ \dots & & & & \\ 1, & v_n, & v_n^2, & \dots, & v_n^{(n-1)} \end{Bmatrix}$$

It is nonsingular if  $v_i \neq v_j$  for all  $i$  and  $j \leq n$ .

Zeta(X) = Riemann zeta function:

The Riemann zeta function of  $x > 1$  is defined by the infinite series  $1 + 1/2^x + 1/3^x + 1/4^x + \dots$ . It is evaluated by first computing the eta function and then using the identity:

$$\text{Zeta}(x) = \text{Eta}(x) * (2^x)/(2^x - 2)$$

which is good for all  $x$  except  $x = 1$  where  $\text{Zeta}(x)$  is infinite.  $\text{Zeta}(0) = -1/2$ .  $\text{Zeta}(2) = \text{Pi}^2/6$ .  $\text{Zeta}(3) =$  the zeta number  $= Z = 1.20205,69031,59594\dots$ .  $\text{Zeta}(x) = 0$  for all negative even integers.

ZetaH(S, A) = Hurwitz zeta function:

The Hurwitz zeta function of  $s > 1$  is defined by the infinite series  $1/a^s + 1/(a+1)^s + 1/(a+2)^s + 1/(a+3)^s + \dots$ .

It is evaluated by  $\text{ZetaH}(s, a) = \text{Lerch}(1, s, a)$ .

$\text{ZetaH}(s, 1) = \text{Zeta}(s)$ , the Riemann zeta function.

{M11, M12, ...; ..., Mrc} = Enter a matrix with r rows and c columns:

For example, {1, 2; 3, 4; 5, 6} will enter the matrix

$$\begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix}$$

The closing } is not required iff it would be the last character on the command line. This function can be used in expressions like any other function. The first row sets the column size. If some rows are shorter than this, they are entered as zero.

-X = Negative of X,  $0 - X$ :

Negative inverse of x. The -, +, and ! functions do not require the parentheses so they also can be considered as unary or monadic operators.

+X = Positive of X,  $0 + X$ :

The identity operator,  $+x = x$ .

!X = Not X,  $0 \rightarrow 1$  else 0:

Logical Not operator. Not Not x (!!x) will leave 0 alone and will change all other values to 1 (True).

Constants -

Cat = Catalan's constant  $G = 0.91596,55941,77219\dots$ , see the Cat procedure.

Ee = Exp(1) = e = 2.71828,18284,59045\dots, see the Ee procedure.

EulerC = Euler's constant  $\gamma = -\Psi(1) = 0.57721,56649,01532\dots$ , see the EulerC procedure.

Ln10 = The natural log of 10 = 2.30258,50929,94045\dots, see the Ln10 procedure.

Phi = Golden Ratio =  $(1 + \text{Sqrt}(5)) / 2 = 1.61803,39887,49894\dots$ , see the Phi procedure.

PhiP = Phi prime =  $(1 - \text{Sqrt}(5)) / 2 = -0.61803,39887,49894\dots$ , see the PhiP procedure.

Pi = Archimedes' Constant  $\pi = 3.14159,26535,89793\dots$ , see the Pi procedure.

Qi = Quaternion i = {1; 0; 0; 0}, see the Qi procedure.

Qj = Quaternion j = {0; 1; 0; 0}, see the Qj procedure.

Qk = Quaternion k = {0; 0; 1; 0}, see the Qk procedure.

QId = Quaternion I = {0; 0; 0; 1}, identity quaternion, see the QId procedure.

Ubiq = The ubiquitous constant  $U = 0.84721,30847,93979\dots$ , see the Ubiq procedure.

Function keys -

The function keys described here or used on the Run form.

F1 => Display Help form, same as the "?" primitive.

F2 => Totally Quit/end the program, same as the "Q" primitive:

If a calculation is running, indicated by the word **\*\*Running\*\*** displayed on the Run form, when F2 is pressed, it is treated as an Abort Calc. request.

F3 => Restore previous input command:

The F3 key normally will restore the command input text box to the value of the previously executed command input. After the B, K, or X command is executed, this key will restore the command line with the learned line. If the learned line changes, when it executes, the previous value of the learned line will be restored by this key.

F4 => Restore previous input command and accept:

The F4 key is the same as F3 except that the previous command is executed without the Enter key being required.

F5 => Get Status of calculation:

Press the F5 key while a computation is being performed and a message like "Integer Multiply: 50 Percent done" will be displayed in the output text box. This is the same as the "Status of Calc." command button.

F6 => Display Configuration form:

Press the F6 key and the Configuration form will appear. F6 is the same as the "Configuration" command button.

F7 => Display Restore Input History form:

Press the F7 key and the Restore Input History form will appear. F7 is the same as the "Restore Input" command button.

F8 => Accept input and Calculate:

This will cause a non-empty contents of the input text box on the Run form to be accepted as command input. Pressing the enter/return key when the cursor is at the end of the command input will accomplish the same thing. F8 is the same as the "Calculate" command button.

F9 => Toggle Logging to Log file on/off:

If the "Logging screen to log file mode is off it will be turned on, if on it will be turned off. F9 is the same as the "Logging is On/Off" command button and the same as the "H" primitive.

F11 => Clear output text box:

This clears the output text box. F11 is the same as the "Clear Output" command button.

F12 => Pause:

This causes the program to suspend all operations until the operator clicks the "OK" button to resume. This frees up the computer if the processor is needed for a priority task. The timing function provided by the Diag command is also suspended so it will continue to give good timing information. F12 is the same as the "Pause" command button and the XMCalc Pause command

Ctrl+F2 => Restart (\$).

Ctrl+F9 => Clear Log File (ClearLog).

Ctrl+F11 => Clear input text box:

This clears the input text box so you can start typing a new command. The box is also cleared when a command is accepted for execution, so this key is not needed in that case. Ctrl+F11 is the same as the "Clear Input" command button. In Windows the F10 key is used to activate the file menu button in the upper left hand corner of the form so F10 is no longer used in this program.

Ctrl+S => Save All (Save).

Ctrl+O => Restore All (Restore).

ESC => Clear Run form Input Text Box:

If a calculation is not currently running, the escape key on the RUN form clears the input text box. On the Startup, Help, About, Restore Input History, Configuration, and Change Disk Directory form, the escape key exits the form.

ESC => Interrupt/Abort a long process:

The ESC key from the Run form while a calculation is running is the same as the "Abort Calc" command button. It can be used after the program has been asked to perform a task that is taking longer than the operator is willing to wait. Press the ESC key once and the message box with the message:

```
*** INTERRUPT: INTERRUPT: To Continue select Ignore
    To Abort Computation select Abort
    To Set SoftAbort flag select Retry
```

will appear.

If Ignore is selected, the interrupt is ignored.

If Abort is selected or Space bar, Enter, or the "A" key is pressed, the message:

```
Computation aborted by operator!
```

will appear, and if auto display is on, the value of the currently active item will be displayed. If the ESC key is pressed during the display of a value, the same messages will appear, but if Abort is selected, it is displayed with fewer digits. Pressing ESC and aborting during execution of the Z command causes all item on the list not yet displayed to be displayed with fewer digits.

If Retry is selected, the message:

```
SoftAbort flag set by operator!
```

will appear. When the SoftAbort flag is set, the variable SoftAbort is put on the list and is set to a value of 1. Its purpose is to allow the operator to flag an XMCalc Code file that it should gracefully terminate its operation. This flag has no other purpose. The SoftAbort item remains =1 until a code file or the operator changes it.

There is another variable that the calculator can add to the list. The variable muErr is put on the list and is set to a value of 1 (True) when a multi-precision error occurs. Its purpose is to allow a code file to detect when it has caused an error and adjust or terminate.

Page Up key => Display previous newer command:

The page up and page down keys can be used on the Run form to move newer and older previous commands into the input text box for execution.

Page Down key => Display previous older command:

The page up and page down keys can be used on the Run form to move newer and older previous commands into the input text box for execution.

Commands used in XMCalc code files:

The following commands are primarily for use in XMCalc code files, but can be used from the operator command input text box.

If Command:

The If command is the first word of an If statement. The syntax of the If statement is:

```
If {expression} Then {statements} Else {statements}
```

The expression following the If is evaluated and if it is True, i.e., not zero, all statements on the same line following the next Else are deleted and execution continues with the statements following the Then. If the expression evaluates to zero (False), all statements following the expression up to the next Else are deleted and execution continues with the statements following the next Else. The Then key word is optional, the Then {statements} is optional and the Else {statements} is optional.

The equivalent of a case statement can be constructed for example like:

```
If A=1 B=3 Else If A=2 B=5 Else If A=3 B=7 Else B=0  
This is equivalent to:
```

```
B=0 If (1 <= A) & (A <= 3) Then B=2*A+1
```

GoTo Command:

The GoTo {label} command will skip all statements following the GoTo until {label}: is found and then start executing the statements following the {label}:. If the GoTo command is in an XMCalc code file, lines of input also will be skipped until the {label}: is found or until an end-of-file. It is not an error if the {label}: is not found, but a GoTo end-of-file or end-of-line will be performed in this case.

GoUpTo Command:

The GoUpTo {label} command will skip all statements following the start of the current line until {label}: is found and then start executing the statements following the {label}:. If the {label}: is not found on the current line and the GoUpTo command is in an XMCalc code file, the file will be reset to the first line of the file and lines of input will be skipped until the {label}: is found

or until an end-of-file. It is not an error if the {label}: is not found, but a GoTo end-of-file or end-of-line will be performed in this case.

#### Labels:

A label is a name followed by a colon (:). When encountered as a command, a label is a no-op. When searching for where to go from a GoTo {label} or from a GoUpTo {label} command, the {label}: is used to determine where to restart execution. If duplicate labels are on a command line or in a code file, the first one encountered is the one that is effective. Labels are alphanumeric with the first character alphabetic, have all characters significant but not case sensitive. Other non-delimiter characters can be used in labels, but this is not recommended. The delimiter characters are:

, ; < = > + - ! | \* / & : ( ) ^ @ # % \ ' "

#### Continuation lines:

Continuation lines are indicated by the last non-blank character of the line being a + or - character. A + says, this line is to be continued by adding the next line, but a blank character should be included between them if it is needed to separate fields. A - says, this line is to be continued by adding the next line, but no extra blank characters should be included between them.

#### Batch Commands (Echo, @Echo, Pause, and Rem) -

##### Echo Command:

Normally, commands from an XMCalc code file are displayed on the screen as they are executed. This can be turned off by the Echo off command and turned on by the Echo on command. If something other than on or off follow the word Echo, it is considered a message and is output to the screen.

##### @Echo Command:

The @Echo command is the same as the Echo command except that, if it is the first command on a line, it is executed before the command line is echoed to the screen. Thus, an @Echo off at the beginning of a line will do an Echo off without the command being echoed to the screen.

##### Pause Command:

The pause command will output the following message to a Message Box and wait for operator to resume:

XMCalc is Paused. OK to resume?

The escape key, space bar, enter/return key, or selecting "OK" will clear the pause action.

##### Rem Command:

The syntax of the Rem command is Rem {remark}. It is a no-op command and everything on the line following the word Rem is skipped.

// Command:

The // command is the same as the Rem command except that it does not need any spaces or delimiters before or after it.

Command buttons -

There are 13 command buttons on the Run form:

Abort Calc.: Same as ESC key.  
Calculate: Same as F8 key.  
Clear Input: Same as Ctrl+F11 key.  
Clear Log File: Clears the log file, same as the ClearLog command and Ctrl+F9.  
Clear Output: Same as F11 key.  
Configuration: Same as F6 key.  
Logging Is On/Off: Same as F9 key.  
Pause: Same as F12 key or Pause command.  
Quit: Quit the Run form and go back to the Startup form.  
Restore All: Restore Configuration, History, & List, same as Restore command and Ctrl+O.  
Restore Input: Same as F7 key.  
Save All: Save Configuration, History, & List, same as Save command and Ctrl+S.  
Status of Calc.: Same as F5 key.

If the Quit button, the File menu Exit, or the form's Close button is selected when a calculation is running, indicated by **\*\*Running\*\*** being displayed on the Run form, the calculation is automatically aborted. The message "When **\*\*Running\*\***, select Quit twice to quit" is displayed in the text output text box and a second quit request is required to quit.

The Run form's Close button, Close menu item and Exit menu item will totally end the program without stopping at the Startup form.

The Run form has a File, a Restore, and a Help menu button. The File menu has Pause, Restart, and Exit. The Restore menu has Restore previous input command and Accept previous input command. The Help menu has Help Form, On Top and About...

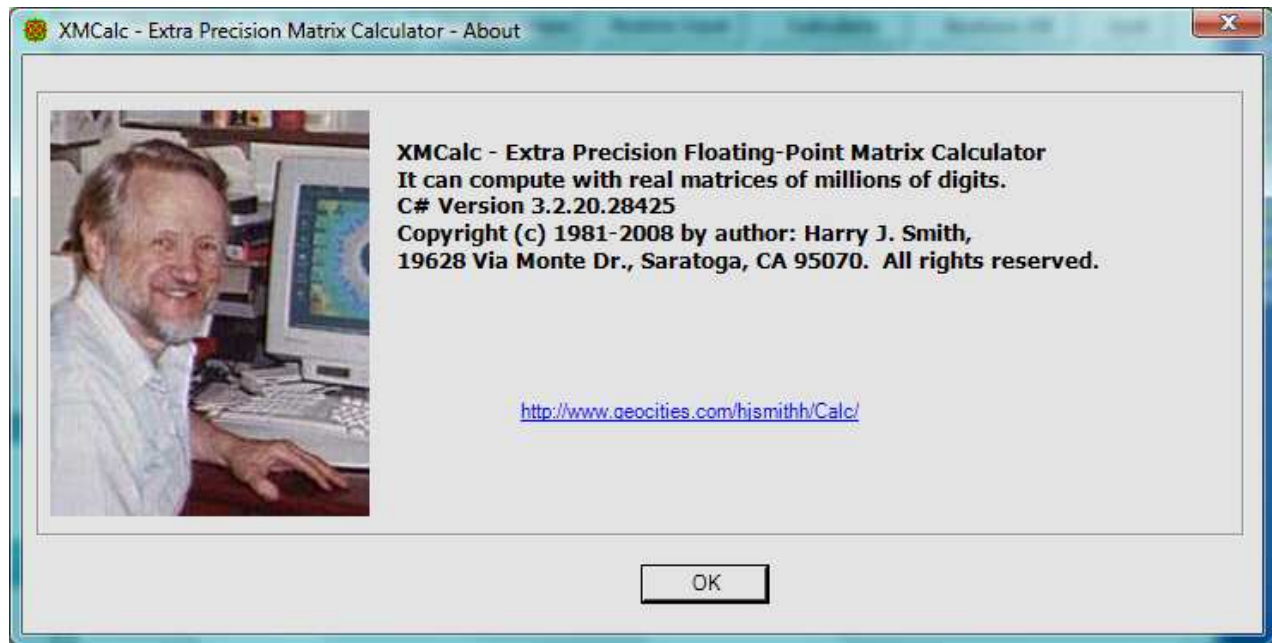
Pause is the same as the Pause button.  
Restart is the same as the \$ command.  
Exit is the same as the Quit button.

Restore previous input command is the same as function key F3.  
Accept previous input command is the same as function key F4.

Help Form is the same as the ? primitive and causes the Help form to be displayed.

On Top is a check box that, when checked, will keep the Run form on top of other windows.

About will bring up a dialog box with Application Title, Version, Application Description and URL to obtain the latest version of the program:



Click the OK button or hit the escape button to remove this box.

#### Transcendental Function Evaluation -

All transcendental functions,  $\text{Sin}(x)$ ,  $\text{Cos}(x)$ ,  $\text{Tan}(x)$ ,  $\text{Cot}(x)$ ,  $\text{Sec}(x)$ ,  $\text{Csc}(x)$ ,  $\text{ASin}(x)$ ,  $\text{ACos}(x)$ ,  $\text{ATan}(x)$ ,  $\text{ATan2}(y, x)$ ,  $\text{ACot}(x)$ ,  $\text{ASec}(x)$ ,  $\text{ACsc}(x)$ ,  $\text{Exp}(x)$ ,  $\text{ExpL}(x)$ ,  $\text{Pow}(x, y)$ ,  $\text{Ln}(x)$ ,  $\text{LnL}(x)$ ,  $\text{LogB}(b, x)$ ,  $\text{Log}(x)$ ,  $\text{Sinh}(x)$ ,  $\text{Cosh}(x)$ ,  $\text{Tanh}(x)$ ,  $\text{Coth}(x)$ ,  $\text{Sech}(x)$ ,  $\text{Csch}(x)$ ,  $\text{ASinh}(x)$ ,  $\text{ACosh}(x)$ ,  $\text{ATanh}(x)$ ,  $\text{ACoth}(x)$ ,  $\text{ASech}(x)$ , and  $\text{ACsch}(x)$ , when they are evaluated, ends up using one of the four basic transcendental functions,  $\text{Sin}(x)$ ,  $\text{ATan}(x)$ ,  $\text{ExpL}(x)$ , and  $\text{LnL}(x)$ . The methods used by these four functions are quite similar: 1) For  $F(x)$ , reduce the given argument  $x$  to a related argument  $f$ . 2) Further reduce  $f$ ,  $NN$  times in a recursive loop to produce an argument  $g$  much smaller than  $f$ . 3) Evaluate the Taylor series for the argument  $g$ . 4) Reconstruct  $F(f)$  from  $F(g)$  by a recursive process executed  $NN$  times. 5) Reconstruct the desired function value  $F(x)$  from  $F(f)$ .

The number  $NN$  in steps 2) and 4) is computed by a heuristic equation of the form  $NN = a + b * \text{SqRt}(M)$  where  $a$  and  $b$  are constants and  $M$  is the current max decimal digits in a mantissa. The best value of  $NN$  is the value that produces the smallest total execution time. After step 4) a best value of  $NN$  is computed and output by estimating a value of  $NN$  that would have made the running time of step 3) equal the sum of the running time of steps 2) and 4).  $\text{Best } NN = NN * \text{SqRt}(T3 / (T2 + T4))$ . Where  $T_n$  is the time to execute step  $n$ ). This equation is based on  $T2$  and  $T4$  being proportional to  $NN$  and  $T3$  being inversely proportional to  $NN$ .

If the operator wants to control the value on  $NN$ , he can enter a value on the list for item  $\text{MSinNN}$ ,  $\text{MATanNN}$ ,  $\text{MExpLNN}$ ,  $\text{MLnLNN}$  to control the value used for  $NN$  in the  $\text{Sin}(X)$ ,  $\text{ATan}(X)$ ,  $\text{ExpL}(X)$ , and  $\text{LnL}(X)$  functions respectively.

The recursive method used to reduce the argument for  $\text{Sin}(X)$  is based on the equation:  $\text{Sin}(X) = \text{Sin}(X/3) * (3 - 4 * \text{Sq}(\text{Sin}(X/3)))$ . In step 2)  $f$  is divided by 3,  $NN$  times to produce  $g$ . In step 4) the recursion:  $S = S * (3 - 4 * \text{Sq}(S))$ , is performed  $NN$  times, where  $S$  is initially the value of  $\text{Sin}(g)$  produced in step 3) and the final value is  $\text{Sin}(f)$ .

The recursive method used to reduce the argument for  $\text{ATan}(X)$  is based on the equation:  $\text{Tan}(X/2) = \text{Tan}(X) / (1 + \text{SqRt}(1 + \text{Sq}(\text{Tan}(X))))$ . In step 2) the recursion:



$T = T / (1 + \text{Sqrt}(1 + \text{Sq}(T)))$ , is performed NN times, where T is initially the value of f from step 1) and the final value of T is the value of g for step 3). In step 4) the angle value,  $A = \text{ATan}(g)$ , produced in step 3) is multiplied by 2, NN times to produce  $\text{ATan}(f)$ .

The recursive method used to reduce the argument for  $\text{ExpL}(X)$  is based on the equation:  $\text{Exp}(x) = \text{Sq}(\text{Exp}(x/2))$ . In step 2) f is divided by 2, NN times to produce g. In step 4) the recursion:  $a = a * (2 + a)$ , is performed NN times, where a is initially the value of  $\text{ExpL}(g)$  produced in step 3) and the final value is  $\text{ExpL}(f)$ . The recursion  $a = a * (2 + a)$  is equivalent to, but more accurate than, the recursion  $e = \text{Sq}(e)$ , where  $e = a + 1$ ; The recursive method used to reduce the argument for  $\text{LnL}(y)$  is based on the equation:  $\text{Ln}(X) = 2 * \text{Ln}(\text{Sqrt}(x))$ . In step 2) the recursion:  $y = y / (1 + \text{Sqrt}(1 + y))$ , is performed NN times, where y is initially the value of f from step 1) and the final value of y is the value of g for step 3). In step 4) the log value  $L = \text{LnL}(g)$  produced in step 3) is multiplied by 2, NN times to produce  $\text{LnL}(f)$ . The recursion  $y = y / (1 + \text{Sqrt}(1 + y))$  is equivalent to, but more accurate than, the recursion  $x = \text{Sqrt}(x)$ , where  $y = x - 1$ ;

If the diagnostic mode is on, the values computed for NN in the four subroutines MSin, MATan, MExpL, and MLnL are displayed, for example, as:

```
MExpL: NN = 22.299
MExpL: NN = 21
Best   NN = 21.935 +/- 1.633
```

In this example the MExpL subroutine estimated NN to be 22.299. A value of NN = 21 was actually used (this is not 22 because the number being worked on was less than 3, the base number used to generate the heuristic equation). Based on the actual timing of the run, the best value for NN is computed to be 21.935. Due to the uncertainty of the timing, the Best NN could be off by + or - 1.633.

The Taylor series used for  $\text{Sin}(x)$  is:

$$\text{Sin}(x) = x - x^3 / 3! + x^5 / 5! \dots$$

The Taylor series used for  $\text{ATan}(x)$  is:

$$\text{ATan}(x) = x - x^3 / 3 + x^5 / 5 \dots$$

The Taylor series used for  $\text{ExpL}(x)$  is:

$$\text{ExpL}(x) = x + x^2 / 2! + x^3 / 3! \dots$$

The Taylor series used for  $\text{LnL}(Y)$  is:

$$\begin{aligned} \text{LnL}(y) &= \text{Ln}(1+y) = \text{Ln}((1+z)/(1-z)) = 2 * (z + z^3/3 + z^5/5 \dots) \\ \text{Where } x &= 1+y = (1+z) / (1-z), \\ y &= x-1 = 2 * z / (1-z), \\ z &= (x-1) / (x+1) = y / (2+y). \end{aligned}$$

Other equations used to produce the transcendental functions:

$$\text{Cos}(x) = \text{Sin}(x + \text{Pi}/2).$$

$\text{Tan}(x) = \text{Sin}(x) / \text{Sqrt}(1 - \text{Sq}(\text{Sin}(x)))$ , and change sign of  $\text{Tan}(x)$  if in 2nd or 3rd quadrant, but error if x is equivalent to plus or minus 90 degrees.

$$\text{ASin}(S) = \text{ATan2}(S, \text{Sqrt}(1 - \text{Sq}(S))), \text{ but error if } |S| > 1.$$

$$\text{ACos}(C) = \text{ATan2}(\text{Sqrt}(1 - \text{Sq}(C)), C), \text{ but error if } |C| > 1.$$

$\text{Log}(x) = \text{Ln}(x) / \text{Ln}(10)$ , but error if  $x \leq 0$ .

For  $x \geq 0.1$ ,  $\text{Sinh}(x) = (y - 1/y) / 2$ , where  $y = \text{Exp}(x)$ ,  
for  $x < 0.1$ ,  $\text{Sinh}(x) = y / (2 * \text{Sqrt}(y+1))$ ,  
where  $y = \text{ExpL}(2*x)$ , and  $\text{Sinh}(-x) = -\text{Sinh}(x)$ .

$\text{Cosh}(x) = (y + 1/y) / 2$ , where  $y = \text{Exp}(|x|)$ .

$\text{Tanh}(x) = y / (y + 2)$ , where  $y = \text{ExpL}(2 * x)$ ,  
and  $\text{Tanh}(-x) = -\text{Tanh}(x)$ .

For  $x \geq 0.1$ ,  $\text{ASinh}(x) = \text{Ln}(x + \text{Sqrt}(1 + \text{Sq}(x)))$ ,  
for  $x < 0.1$ ,  $\text{ASinh}(x) = \text{LnL}(x + \text{Sq}(x) / \text{Sqrt}(1 + \text{Sq}(x)))$ ,  
and  $\text{ASinh}(-x) = -\text{ASinh}(x)$ .

$\text{ACosh}(x) = \text{Ln}(x + \text{Sqrt}(\text{Sq}(x) - 1))$ , but error if  $x < 1$ .

$\text{ATanh}(x) = \text{LnL}(2 * x / (1 - X))$ , but error if  $|x| \geq 1$ ,  
and  $\text{ATanh}(-x) = -\text{ATanh}(x)$ .

Notes about applying commands to matrices:

Most commands like  $\text{Sin}(X)$  applied to a matrix  $X$  will produce a matrix with the same dimensions as  $X$  with the elements equal to the command applied to the corresponding elements of  $X$ . If the command takes 2 or 3 arguments, if one argument is a matrix, the other arguments must also be matrices of the same size. This will produce a matrix with the same dimensions as  $X$  with the elements equal to the command applied to the corresponding elements of arguments. This also applies to the infix operators  $^$ ,  $@$ ,  $\#$ ;  $\%$ ,  $\backslash$ ,  $\&$ ,  $|$ ,  $<$ ,  $>$ ,  $=$ ,  $==$ ,  $<=$ ,  $!=$ ,  $<>$ ,  $>=$ , and  $--$ .

Exceptions are the true matrix operations.

$0 \Rightarrow x = 1/x$  Matrix inverse of a square matrix  
 $A = X * Y \Rightarrow$  Matrix multiply. Can multiply a scalar and a matrix.  
 $A = X / Y \Rightarrow X * (1/Y)$ . Can divide a scalar and a matrix.  
 $\text{Inv}(X) = 1 / X \Rightarrow$  Matrix inverse of a square matrix.

Error reports -

There are many different error reports that are a result of directly or indirectly requesting an operation that cannot be performed (or conceivably an error in the XMCalc program). Another type of error is the syntax error, where a command cannot be interpreted.

The computation error reports are:

$\text{ACos}(x)$  error:  $|x| > 1.0$   
 $\text{ACosh}(x)$  error:  $x < 1.0$   
Addition overflow, continuing... (M={m})  
 $\text{ASin}(x)$  error:  $|x| > 1.0$   
 $\text{ATanh}(x)$  error:  $|x| \geq 1.0$   
 $B^P$  overflow, continuing...  
BernN:  $n > 10000000$ , too large for Bernoulli number  
 $\text{BetaC}(x, y)$  error: BetaC is infinite  
BinoD: numbers too large for binomial coefficient  
BinoN: numbers too large for binomial coefficient  
Cannot compute  $\text{FacM2}(X, Y)$  if FMB is not an integer  
Cannot divide by zero, continuing...  
Cannot divide by zero, continuing...

Cannot generate random numbers with digits in mantissa < 35  
Cannot raise negative number to a non-integer power  
Cannot raise zero to a negative power  
Cannot set an element of a scalar  
Cannot set FMB with a Mod FMB function, FMB cleared  
Cannot set same location to both quotient and remainder, continuing...  
Cannot set same location to both square root and remainder, continuing...  
Cannot take digamma of integer <= zero  
Cannot take factorial of number < zero  
Cannot take gamma of integer <= zero  
Cannot take GamL(a, x) if a is an integer <= zero, except GamL(0, 0) = 0  
Cannot take GamP(a, 0) if a is an integer < zero  
Cannot take GamQ(a, 0) if a is an integer < zero  
Cannot take GamU(0, 0)  
Cannot take GamU(a, 0) if a is an integer <= zero  
Cannot take lambda(x) for x = one  
Cannot take square root of negative number, continuing...  
Cannot take Zeta(x) for x = one  
Cannot use a matrix as an exponent  
Cannot use a matrix as a scalar  
Cannot use a scalar as a matrix  
E1(x) error: x = 0  
Ei(x) error: x = 0  
Element outside of matrix  
Error in Ln10, FMC = {fmc}  
Error in Pi, FMC = {fmc}  
Error: Expanded precision left on  
Error: FMC > basic FMC  
EulerN: n > 10000000, too large for Euler number  
Exp overflow, continuing...  
Exp underflow, continuing...  
FHT cannot recover from failure/probable error in FHT multiply  
FHT Max Error = {max}, probable error in FHT multiply!!!!!!  
FHT overflow, should not happen but it did!!!!!!  
FHT Recovered from failure/probable error in FHT multiply  
FHT Sum-of-Digits failed, error in FHT multiply!!!!!!  
Floating divide overflow, continuing...  
Floating multiply overflow, continuing...  
Floating Norm overflow, continuing...  
Floating SetTo overflow, continuing...  
Floating shift left overflow, continuing...  
Floating shift right underflow, continuing...  
Floating Value overflow, continuing...  
GamLS: a too large  
GamLS: x <= 0  
GenBern: n > 10000000, too large for Bernoulli number  
GenBernI: Bernoulli n too large  
GenEuler: n > 10000000, too large for Euler number  
GenEulerI: Euler n too large  
Get1 overflow, continuing...  
GetD overflow, continuing...  
Input number overflow, continuing... (M={m})  
Li(1) = -infinity  
Li(x) undefined for x < 0  
LnL: Cannot take Ln of X < 0, continuing...  
LnL: Cannot take Ln of zero, continuing...  
Matrices are incompatible  
Matrix is singular  
Mersenne Prime index out of supported range [1, 46]  
Multiplication overflow, continuing... (N={n} > M={m})  
No convergence in {maxIt} SVD iterations  
Not all vectors found  
Number too large for Bernoulli number

```

Number too large for Digamma function
Number too large for Euler number
Number too large for factorial function
Number too large for gamma function
One digit multiply overflow, continuing... (M={m})
PolSturm failed: Polynomial is not square free
QuadR: Cannot set X1 or X2
QuadR: No real solutions
QuadR: No solutions
QuadR: No unique solutions
Ri(x) error: x <= 0
SetTo overflow, continuing... (N={n} > M={m})
Signed Integer shift left overflow, continuing...
Sin: Angle too large
Tan(x) error: x equivalent to +/- 90 Deg.
This is not a column vector
This is not a quaternion
This is not a rotation matrix
This is not a square matrix
This is not a vector
Unexpected error in Cube Root function
Unsigned subtraction error, continuing...
X too big to add, continuing... (N={n} > M={m})

```

Some of these error messages are due to internal checks and should never be seen by the user.

The syntax errors are:

```

( expected: {procedure}||{string}
Directory name expected: CHDIR(||{string}
Error in function's 2nd argument: {name}(...,||{string}
Error in function's 3rd argument: {name}(...,||{string}
Error in function's argument: {name}(|{string}
Exponent expected: ^{sign}||{string}
Expression expected: (||{string}
Expression expected: (IF ||{string}
Expression expected: {name}(|{string}
Factor expected: {op}||{string}
File name expected: {procedure}(|{string}
Simple Expression expected: {op}||{string}
Term expected: {op}||{string}
Unknown function: |{name}({string}
Unknown operation, Command line discarded: ||{string}

```

The vertical bar | always shows the start of the string of characters that cannot be interpreted.

I/O error messages are:

```

Cannot open file "{filename}" for input
Cannot open file "{filename}" for output
File not found
Read error file "{filename}"

```

Other Informational messages:

```

{function name}: 2nd and 3rd argument assumed = 0.0
{function name}: 3rd argument assumed = 0.0
{function name}: Only two arguments used

```

```

{function name}: Second argument assumed = 0.0
{function name}: Second argument ignored
{n} Commands read from History file, {m} total
{n} Commands written to History file
(Done {n} of {m}, {g} to go)
Aborting file input...
Aborting file write...
Bernoulli number storage cleared
Command history cleared
Command line was: {command}
Computation aborted by operator!
Continuing...
Directory name = {directory}
Directory changed to {directory}
Directory not changed {directory}
Euler number storage cleared
Factorial function aborted by operator, {n} multiplies to go
Gamma function aborted by operator, {n} multiplies to go
FHT Radix = {b} ldn = {n} Max Error = {e} DT = {t} sec. x.N = {n} ...
File "{filename}" closed
File "{filename}" opened for reading
File "{filename}" opened for writing
File is corrupted: "{filename}"
Full name = "{directory}\filename}"
{function}: {error message}
Generating Bernoulli numbers upto B({x})
Generating Euler numbers upto E({x})
Have Bernoulli numbers upto B({n})
Have Euler numbers upto E({n})
Home directory is {directory}
I/O operation aborted by operator!
Inverse: No inverse
Label skipped = {label}:
Log file "{filename}" Cleared {date} {time}
Log file "{filename}" Closed {date} {time}
Log file "{filename}" Opened for Append {date} {time}
Matrix is singular
Max Bernoulli number to store = B({m})
Max Euler number to store = E({m})
Max commands in history changed from {old max} to {new max}
Max commands in history not changed from {max}
MersenneP: u{i} = {n}
Normal multiply, baseM = {b} DT = {t} sec. x.N = {n} y.N = ...
Not in current directory {dir}
Not in home directory {dir}
OpenAppend: {error message}
OpenRead: Could not find file '{directory}\filename}'.
OpenRead: {error message}
OpenWrite: {error message}
Path not found!
PolSturmI: PolEval(p, a) = 0.0
PolSturmI: PolEval(p, b) = 0.0
Power = {p}
Priority is {priority}
Reading Commands from History file - Begin ... Aborted
Reading Commands from History file - Begin ... End
Reading Help File: "{filename}"
Reading number file "{filename}" No named number found
Reading number file "{filename}" containing {item name}
Rewinding code file
Running code file "{filename}"
{Singular, }Rank = {rank}, Residual = {residual}
SoftAbort flag set by operator!

```

```

SortC: No elements were swapped
SortC: Some elements were swapped
SortR: No elements were swapped
SortR: Some elements were swapped
T = x.xx DT = x.xx sec. End of execution
T = x.xx DT = x.xx sec. Start execution
Using the Chudnovsky brothers' binary splitting algorithm to compute Pi
When **Running**, select Quit twice to quit
Writing all Commands to History file - Begin ... End
Writing file "{filename}"
Writing file "{filename}" with {item name}
ys < qs in SqRtRemFastSI()

```

Status/Diagnostic messages -

```

BaseI = {base-in} base ten, Max digit = {x}
BaseO = {base-out} base ten, Max digit = {x}
BetaNF: (Done {n} of {m}, {g} to go)
BetaNF: n = {n}
BinoFMC: FMC = {fmc}
BinoN: (Done {n} of {m}, {g} to go)
CheckNMax: Max Bernoulli number too large
CheckNMax: Max Euler number too large
Converting number to output base: xx Percent done
E1N: added {more} more digits
E1N: k = {k}
E1N: lost {lost} digits
EtaNF: (Done {n} of {m}, {g} to go)
FacFMC: FMC = {fmc}
FacNF: (Done {n} of {m}, {g} to go)
FHT: {xx} Percent done
FHT0: {xx} Percent done
GamNF1: (Done {n} of {m}, {g} to go)
GamLS: n = {n}
GamUC: n = {n}
GenBernI: (Done {n} of {m}, {g} to go)
GenEulerI: (Done {n} of {m}, {g} to go)
Integer Divide: xx Percent done
Integer Multiply: xx Percent done
Integer Square Root: xx Percent done
Lerch{n}: Cannot compute LerchPhi(x, s, a) for |x| > 1
Lerch{n}: Cannot compute LerchPhi(x, s, a) for x > 1
Lerch{n}: Iteration = {i}
Lerch{n}: LerchPhi is not defined for integer a <= 0
Lerch{n}: No convergence within the maximum number of iterations = {i}
Lerch{n}: Overflow in Aj()
Lerch{n}: Pow() is not defined for a < 0 and s not integer
Lerch{n}: Results at iteration {i} had a relative error approx. = {r}
Lerch{n}: Underflow in remainder estimate omega in LerchPhi()
Lerch{n}1: Accuracy reduced {acc}
Lerch{n}1: Denominator reached infinity
LimFac: LimFac = {limFac}
MATanN: expand (Done {n} of {m}, {g} to go)
MATanN: reduce (Done {n} of {m}, {g} to go)
MeanCN: Iteration = {i}
MExpLN: expand (Done {n} of {m}, {g} to go)
MExpLN: reduce (Done {n} of {m}, {g} to go)
MLnLN: expand (Done {n} of {m}, {g} to go)
MLnLN: reduce (Done {n} of {m}, {g} to go)
MSinN: expand (Done {n} of {m}, {g} to go)
MSinN: reduce (Done {n} of {m}, {g} to go)
MuAbort exit GenBern
MuAbort exit GenEuler

```

```

PiCh: {pct}% complete, DT = {t} sec.
PsiNF: k = {k}
SATanN: series (Done {n} of {m}, {g} to go)
SExpLN: series (Done {n} of {m}, {g} to go)
SlnLN: series (Done {n} of {m}, {g} to go)
SSinN: series (Done {n} of {m}, {g} to go)
Used Bernoulli number B({n})
Used Euler number E({n})
Went to {label}:

```

Prime number status and diagnostic messages -

```

(Testing {n})
{a}^{b} mod {m} = {c}
{d} is a big factor of {n}
{n} / {d} MethodX
{n} / {d} MoreDiv
{n} / {d} SieveMethod", Tot
{n} ^ 1
{n} has no small factors
{n} is a hard nut to crack
a > 2147483647 too large for Legendre's formula
Completely factored
Completely factored (Exponents are in decimal)
DelX=X-Root(N)={n}
EC Method with limit (B1 = {b1} B2 = {b2}) Curve 1 2 ...
ECM: (Done {n} of {m}, {g} to go)
Enter FactorN
Enter LargeFactor
Enter MoreDiv, Try the division method some more
Enter PowerMod
Enter ReportFactor
Enter SieveMethod
Enter SmallFactors
Error, cannot store more than 2000 prime factors
Error, factors do not check
Exit FactorN
Exit FactorN from LargeFactor
Exit FactorN from ReportFactor
Exit LargeFactor
Exit PowerMod
Exit ReportFactor
Exit SmallFactors
FnAdleman: I quit! Too big.
FnAdleman: Test5_P
Generating prime number table with {n} primes
GetExSI: Specified argument was out of the range of valid values.
I={i} J={j} MATCH={True/False}
In FactorN loop
Is x a prime number?:
IsSq(X) function:
KS({I})={m}
Large factor {f} is prime
Large factor is not prime
LargeFactor exit FactorN
Moebius Mu(X) function:
MoreDiv exit FactorN
MuAbort exit MoreDiv
N is not prime
N is prime
N is too big
N= {n} Start= {s}
Normal exit MoreDiv

```

```

{Number} already factored with {n} prime(s)
Number not completely factored
Numbers got too large for prime number generator
P({n}): P({i}) = {p}
P(n) = The n'th prime:
PhiLEx: n = {n}, a = {a}
PhiLInt: n = {n}, a = {a}
PhiLSI: n = {n}, a = {a}, depth = {d}
Prime factor algorithm A:
Prime factor algorithm B:
Prime Greater Than:
Prime Less Than:
Prime Pi Function:
Primo({n}): P({i}) = {p}
PrimePiL: (Done {n} of {m}, {g} to go)
PrimePiM: (Done {n} of {m}, {g} to go)
Prime Phi function = Number of integers <= n relatively prime to x:
PrimRQ function, is x a primitive root of y?:
Prime Sig function = Sum of the divisors of x:
Prime Sig0 function = Sum of the divisors of x (-x):
Prime Tau function = Number of divisors of x:
Primorial Function:
Quick exit SmallFactors
Report={True/False} NF={nf} Po={po} Factor={f}
ReportFactor exit FactorN
ReportFactor exit MoreDiv
ReportFactor exit SieveMethod
ReportFactor exit SmallFactors
Should exit LargeFactor
SieveMethod exit FactorN
SmallFactors exit FactorN
SqFree(X) function:
Square Max P = {n}
SqRt={x} Rem={r}
Starting to factor
Stored Number not squarefree
Wait, filling the {n} entry prime table
Y=Root(X*X - N)={root} Rem={rem}

```

The (DONE {n} of {m}, {g} to go) message can be displayed when the Abort Calc. button is selected. It helps you decide if you really want to abort. {n} is the number of iterations of an innermost loop. {g} is the number of iterations left to go. For example, for the factorial function, {n} is the number of multiplications done and {g} is the number left to do.

Other forms that can be displayed to help in using the program are the Startup, Help, Restore Input History, Configuration, and the Change Disk Directory forms. On all forms except the Run form, pressing the ESC key will remove the form.

Startup form -

The Startup form is shown above and has two parameters that can be set before the calculator starts accepting commands:

"Max Digits in a Number" can be set to a value from 1 to 134218400. XMCalc will change this to the next multiple of 8 greater than or equal to 56. The M and SetMax(X) commands will not be able to set Max digits in a number greater than this. The nominal starting value is 134218400.



"Max Commands in History" can be set to a value from 1 to 100000. XMCalc will allocate an array of this length to save command history. The nominal starting value is 1000 and it can be changed by the SetC(X) command

These two values are tested as they are edited. Spaces on the left or right are removed so you will not see them. These numbers are displayed in **bold** face font iff they are acceptable values. If either value is unacceptable, the Run button is disabled.

A "Default Settings" button is provided to reset the two parameters to their original values.

The Run button is used to bring up the Run form. It can be used after a Run form exits to reinitialize XMCalc and start running again. The Exit button will terminate the program.

Help form -

The Help form was shown and partially described above. It has a File menu button that has a menu with Print Setup, Print..., Print Preview and Exit. The Print Setup will bring up the Windows printer setup dialog box so you can select the printer to use. The Print... will start the printing process to start printing the help file. This will normally bring up the printers dialog box. The Print Preview does just that, it can also be used for printing. The Exit does the same thing as pressing the escape key; the Help form is removed.

This form also has a "F3 Find:" button, a "F2 Up" button and a small text box for entering a string of characters to find. Pressing F3 is the same as clicking the "F3 Find:" button, the text in the small text box is searched for. Pressing F2 is the same as clicking the "F2 Up" button, same as F3 except the search is done from the current location towards the top of the large text box.

If during a find the text is not found, the sound from NotFound.wav is heard. If it is found, but only by starting over from the other end of the text, the sound from Wrap.wav is heard. This sound process uses the system file winmm.dll. If that file cannot be found, there will be no sound.

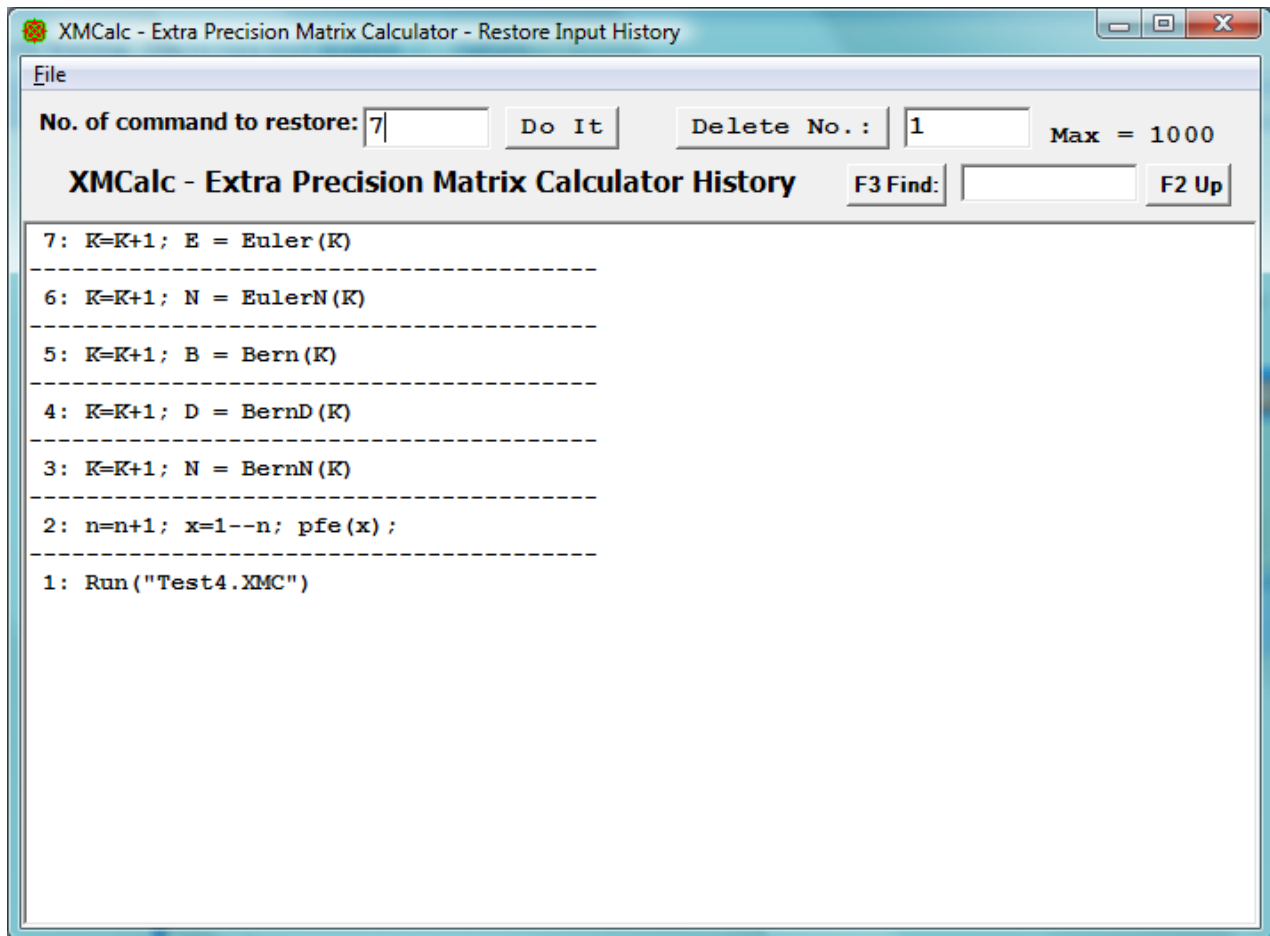
A short beep is also heard when the text is found. The beep is 277Hz = middle C# for a normal search and 554Hz one octave above for a search up. The beep process uses the system file kernel32.dll. If that file cannot be found, there will be no beep.

Text can be copied from the large text box to the small text box by selecting the text with the left mouse button and pressing Ctrl-F.

The Help form is unique in that more than one copy can be brought up at the same time. This may or not be useful, but it illustrates a technique in Windows programming.

Restore Input History form -

The Restore Input History form is displayed when function key F7 is pressed or the "Restore Input" command button is selected on the Run form:



The history of up to "Max Commands in History" previous operator entries are saved and can be retrieved by this form

While this form is up and the cursor is in the output text box, use the Up, Down, PgUp, and PgDn keys and the keypad + and - keys to select an entry and then use the Enter key or space bar to accept it. The command accepted will be put into the Command entry text box on the Run form, and there it can be edited before it is executed. The ESC key will remove the History form without changing the Command input text box.

The Ins key will toggle the locked status of the entry containing the cursor and move the cursor to the next higher command. When an entry is locked it cannot be deleted or scrolled off the bottom of the list. A # will be displayed to the left of a locked entry.

The Del key will delete the entry containing the cursor, if it is not locked, and move to the next higher command. The command numbers in the two command number entry text boxes are updated as commands are locked, unlocked, deleted or scrolled to by the + and - keypad keys. The - key is up and + is down due to layout of the keypad, - above +. Only the keypad + and - keys are used here.

The mouse can also be used. A left mouse click will select a command and a right mouse click will accept it.

The "Number of command to restore:" text box and the "Do It" command button can be used to select a command to be restored to the command input text box. The "Delete Command No." command button and its text box can be used to delete a given command by number. These controls act in the same way. Its text box can be edited and a click on the command button will accept the entry. A space at the right of the text or a Return/Enter key will also accept it.

When the Delete Command No.: button is selected the command referenced is deleted if it is not locked. If locked the command number field will be increased to point to the next command or a set to 1 if at max number.

The Restore input History form has a file menu with Clear History, Restore history, Save History, and Exit.

Selecting Clear History will remove all commands saved in memory. This is the same as the ClearHist command.

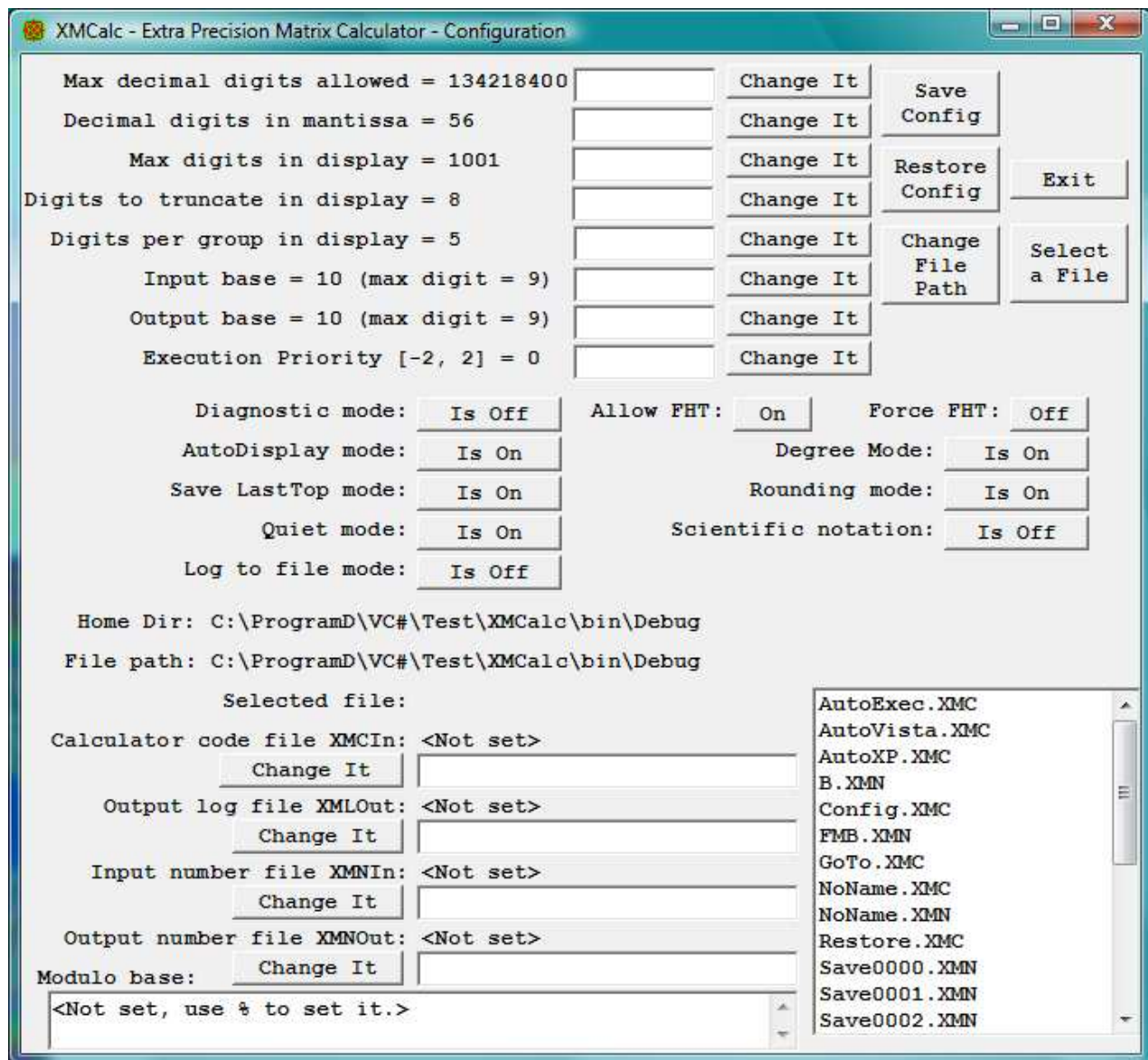
Selecting Restore History will read in the file XMCalcHist.txt and add its commands to the ones in memory. This is the same as the [ command. Duplicate commands are deleted as the new copy is entered, but the lock state will be as set on the copy of the command in memory.

Selecting Save History will write all of the current command history to file XMCalcHist.txt. This is the same as the ] command. If a command history file already exists, it will be renamed XMCalcHist.Bak.

Selecting Exit does the expected, same as the escape key.

Configuration form -

The Configuration form is displayed when function key F6 is pressed or the "Configuration" command button is selected on the Run form:



This form allows you to change:

- Max Decimal digits allowed, same as the SetMax(X) command.
- Digits in mantissa, same as the M command.
- Max digits in display, same as the SetD(X) command.
- Digits to truncate in display, same as the T command.
- Digits per group in display, same as the G command.
- Input base = xx (max digit = x), same as the BaseI(X) command
- Output base = xx (max digit = x), same as the BaseO(X) command
- Execution Priority [-2, 2] = {p}, same as the Pri(X) command
- Diagnostic mode, a combination of Diag(X) and Time commands.
- AutoDisplay mode, same as the A command.
- Save LastTop mode, same as the SaveTop(X) command.
- Quiet mode, same as Quiet(X) command.
- Log to file mode, same as the LogScreen(X) command.
- Degree mode, same as the D and E commands.
- Rounding mode, same as the U and V commands.
- Scientific notation mode, same as the ScientificN(X) command.
- Name of Calculator code file XMCIn, same as the XMCIn(F) command.
- Name of Output Log file XMLOut, same as the XMLOut(F) command.
- Name of input number file XMNIn, same as the XMNIn(F) command.
- Name of output number file XMNOut, same as the XMNOut(F) command.

It also shows the Home Directory, File path, Modulo Base, and the name of the log file if the LogScreen mode is on. The Browse For Folder form can be brought up by selecting the "Change File Path" command button, and the Select a file form can be brought up by selecting "Select a File" command button. The Save Config button is the same as the > command to save the configuration to file Config.XMC. The Restore Config button is the same as the < command to restore the configuration.

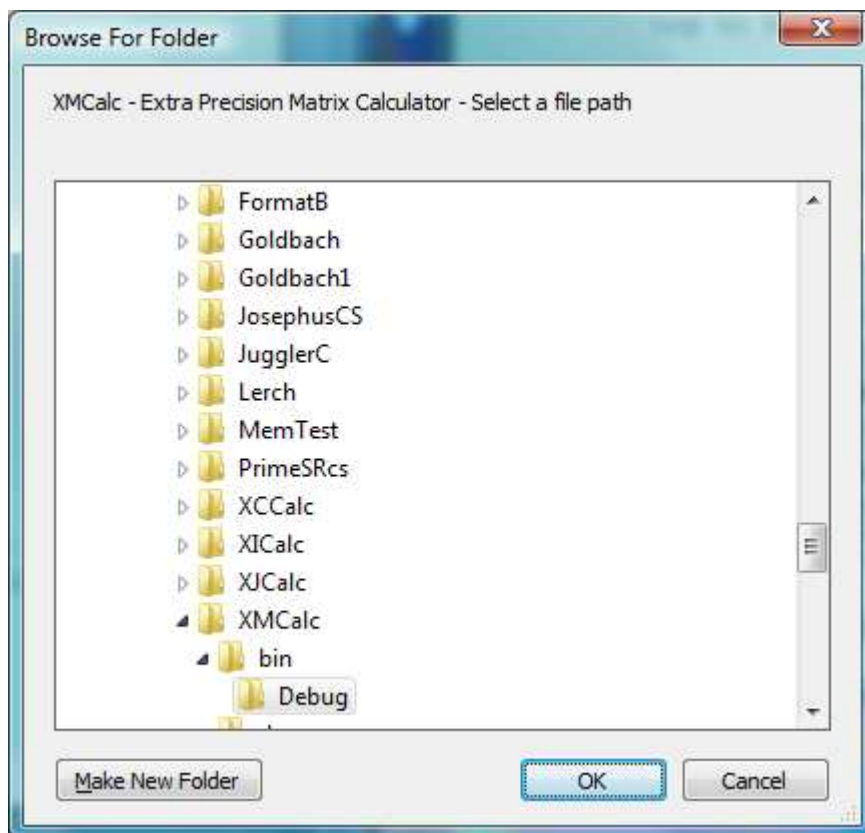
All twelve "Change It" buttons act in the same way. Clicking it when its text box is empty will bring up the current value. It can be edited, and a second click with the text box not empty will accept the entry. A space at the right or a Return/Enter key will also accept it.

The ten modes can be changed/toggled by clicking the corresponding button. If a mode is on, the button will say "Is On", then clicking it will turn the mode off and the button will change to "Is Off". If a mode is off, the button will say "Is Off", then clicking it will turn the mode on and the button will change to "Is On". The Diagnostic mode differs, it has four states, Is Off, Time, Diag, and Debug.

The Log to file mode and Name of output log file XMLOut interact in that if the file name is changed while a log file is open, the file name of the open file will not change until the log file is closed and reopened.

Browse For Folder form -

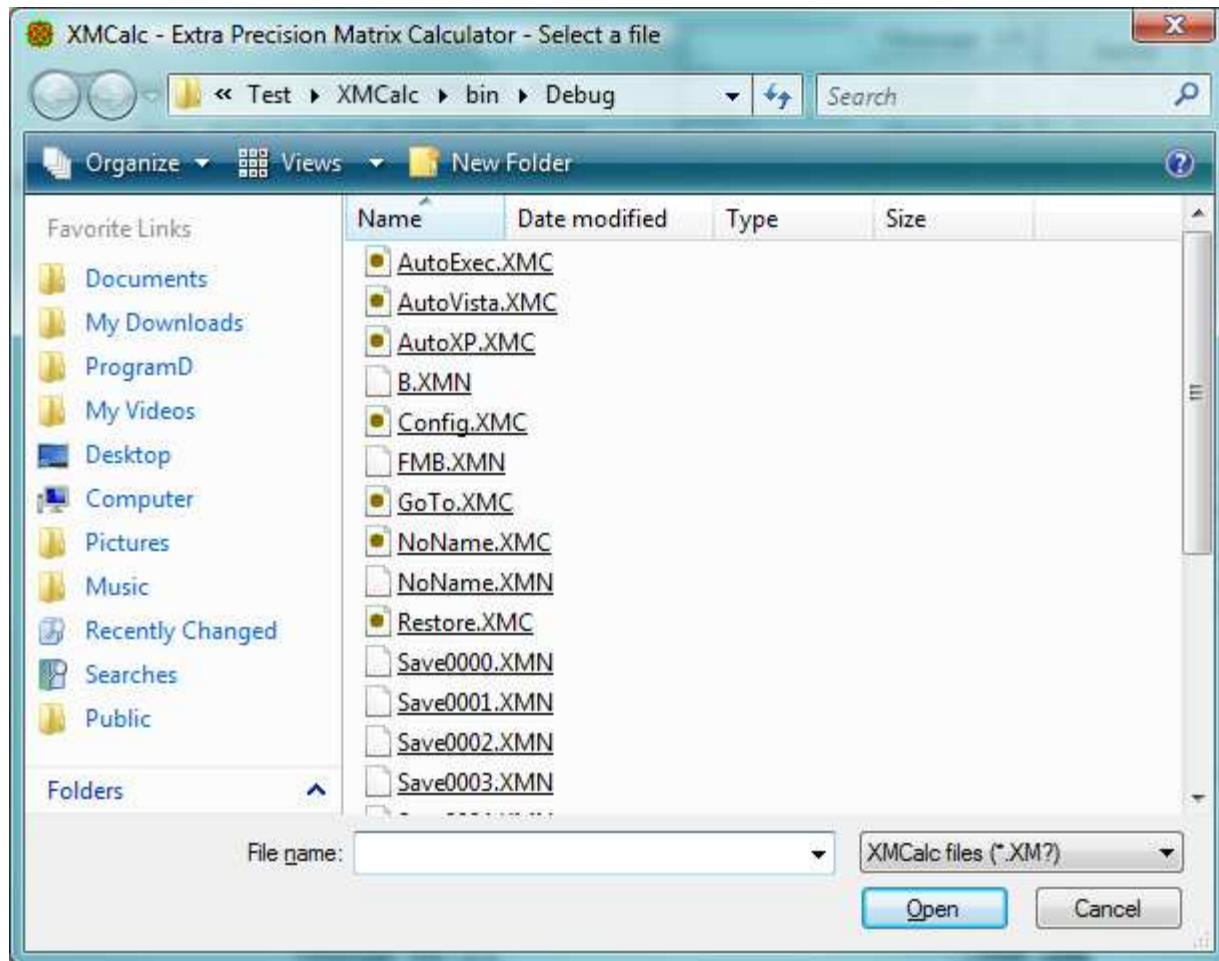
The Browse For Folder form is displayed when the "Change File Path" command button is selected on the Configuration form:



To change the current directory to be used by the H, I, J, W, ReadN(F), and Run(F) commands, use this form to select a path. This form is resizable as is the Run, Help, History, and Select a file forms.

Select a File form -

The Select a file form is displayed when the "Select a File" command button is selected on the Configuration form:



This provides a way to browse and select a file. If a file is select, it can then be used to set a file name on the Configuration form by clicking one of the file name "Change It" buttons. The first click enters the file name in the text box and a second click will accept it. This form can also be used to change the file path.

The list box in the lower right hand corner of the Configuration form is a list of the files in the currently selected file path. The wild card selection for these files (\*.XM?, \*.txt, or \*.\* ) is the same as last used on the Select a file form.

The distribution files can be downloaded from my website:

<http://www.geocities.com/hjsmithh/>

in the [Files to Download](#) section

<http://www.geocities.com/hjsmithh/download.html#XMCalc>.

When you install the program using the distribution file XMCalc32?.exe or XMCalc32?.zip, a folder is created with 4 subfolders:

```
├── XMCalc 3.2
│   └── doc
```

```
|  ┌─run
|  │─src
|  └─test
```

The main folder has two files sseexec.dat and SSEun.dat. These are needed to be able to uninstall the program

The doc subfolder has the following 4 files:

```
00.txt      (A short list of features)
Fixes.txt   (A list of features and fixes added to each version)
XMCalc.doc  (Microsoft Word file)
XMCalc.txt  (Plain ASCII text copy of the .doc file without graphics)
```

There is also available on my website a copy of the .doc file in an Adobe Acrobat Reader file XMCalc.pdf.

The run subfolder has the following 11 files

```
00.txt
AutoExec.XMC
AutoVista.XMC
AutoXP.XMC
Ln10.XPN
NotFound.wav
Pi.XPN
Wrap.wav
XMCalc.exe
XMCalc.exe.config
XMCalcHelp.txt
```

The .exe file is executed from there.

The src subfolder has all the source files needed for development:

```
00Note.txt
AboutForm.cs
AboutForm.resx
app.config
AssemblyInfo.cs
Calc.cs
Common.cs
ConfigForm.cs
ConfigForm.resx
COPYING.txt
cs.ico
FHTMult.cs
Global.cs
harry.jpg
HelpForm.cs
HelpForm.resx
HistoryForm.cs
HistoryForm.resx
MultiFD.cs
MultiID.cs
MultiMD.cs
MultiPi.cs
RunForm.cs
RunForm.resx
StartupForm.cs
StartupForm.resx
XIMult.cs
XMCalc.csproj
```

XMCalc.sln  
XMCalc.suo  
XMCalc.csproj.user  
XMCallst.cs  
XPCallst.cs

The test subfolder has the files I use for testing the program. They are:

00.txt  
AutoExec.XMC  
AutoVista.XMC  
AutoXP.XMC  
Config.XMC  
FMB.XMN  
NoName.XMC  
NoName.XMN  
NotFound.wav  
Restore.XMC  
Save0000.XMN  
Save0001.XMN  
Test.Bat  
Test1.XMC  
Test1.XML  
Test2.XMC  
Test2.XML  
Test3.XMC  
Test3.XML  
Test4.XMC  
Test4.XML  
Test5.XMC  
Test5.XML  
WhatForTst.txt  
Wrap.wav  
XMCalc.exe  
XMCalc.exe.config  
XMCalcHelp.txt  
XMCalcHist.Bak  
XMCalcHist.txt  
XMInfo.txt

The program can generate the following files:

Config.XMC  
FMB.XMN  
NoName.XMC  
NoName.XML  
NoName.XMN  
Restore.XMC  
Savexxxx.XMN  
XMCalcHist.Bak  
XMCalcHist.txt

The end -

Report any errors by sending me a letter, an e-mail or call me at my home phone.

-Harry

Harry J. Smith



19628 Via Monte Dr.  
Saratoga, CA 95070-4522, USA

Home Phone: 1 408 741-0406  
E-mail: [hjsmithh@sbcglobal.net](mailto:hjsmithh@sbcglobal.net)  
Website: <http://www.geocities.com/hjsmithh/>