1. Math Functions

Overview of the MATH FUNCTIONS Menu

Calculus	**Complex**	**Constant**	**Hyperb.**	**List**
∂	ARG	e	ACOSH	CONCAT
∫	CONJ	i	ASINH	ΔLIST
TAYLOR	IM	MAXREAL	ATANH	MAKELIST
	RE	MINREAL	COSH	ΠLIST
		π	SINH	POS
			TANH	REVERSE
			ALOG	SIZE
			EXP	ΣLIST
			EXPM1	SORT
			LNP1	

Loop	**Matrix**	**Polynom.**	**Prob.**	**Real**
ITERATE	COLNORM	POLYCOEF	COMB	CEILING
RECURSE	COND	POLYEVAL	!	DEG→RAD
Σ	CROSS	POLYFORM	PERM	FLOOR
	DET	POLYROOT	RANDOM	FNROOT
	DOT		UTPC	FRAC
	EIGENVAL		UTPF	HMS→
EIGENVV		UTPN		→HMS
	IDENMAT		UTPT	INT
	INVERSE			MANT
	LQ			MAX
	LSQ			MIN
	LU			MOD
	MAKEMAT			%
	QR			%CHANGE
	RANK			%TOTAL
	ROWNORM			RAD→DEG
	RREF			ROUND
	SCHUR			SIGN
	SIZE			TRUNCATE
	SPECNORM			XPON
	SPECRAD			
	SVD			
	SW			
	TRACE			
	TRN			

Stat-Two	**Symbolic**	**Test**	**Trig**
PREDX	=	<	ACOT
PREDY	ISOLATE	≤	ACSC
	LINEAR?	= =	ASEC
	QUAD	≠	Col'
	QUOTE	>	CSC
	|	≥	SEC
		AND	
		IFTE	
		NOT	
		OR	
		XOR	
		OR	
		XOR	

Syntax Guide to MATH FUNCTIONS

Calculus

∂	∂<name>(<expr>)		
	x, expression	------>	*derivative of the expression with respect to variable x*
∫	∫(<R#>, <R#>, <expr>, <name>)		
	a, b, expression, x	------>	*integral from a to b of expression with respect to x*
TAYLOR	TAYLOR(<expr >, < name >, < N#>)		
	expression,x,n	----->	*Taylor polynomial about x = 0 of order n*

Complex

ARG	ARG(<C#>)		
	complex number	------>	*argument of complex number*
	[i.e., if the complex number z has polar form (r, t), then ARG(z= t]		
CONJ	CONJ(<C#>)		
	complex number	------>	*conjugate of complex number*
IM	*IM(<C#>)*		
	complex number	------>	*imaginary part*
RE	RE(<C#>)		
	complex number	------>	*real part*

Constant

(These take no arguments; they are special numbers with special names.)

e	the base of the natural log (rounded to 2.71828182846)
i	(0,1) the standard square root of –1
MAXREAL	the constant 9.9999999E499 *(largest positive real number that can be represented on the HP38G)*
MINREAL	the constant 1.0000000E499 *(smallest positive real number that can be represented on the HP38G)*
π	the constant n (rounded to 3.14159265359)

Hyperb

ACOSH ACOSH(<#>)

X ---------- > *inverse hyperbolic cosine of x*

ASINH ASINH<#>)

X ---------- > *inverse hyperbolic sine of x*

ATANH ATANH(<#>)

X --- ------ > *inverse hyperbolic tangent of x*

COSH COSH(<#>)

X ---------- > *hyperbolic cosine of x*

SINH SINH(< # >)

X ---------- > *hyperbolic sine of x*

TANH TANH(<#>)

X ----------> *hyperbolic tangent of x*

ALOG ALOG(<#>)

X ---------- > 10^x

NOTE: More accurate than the usual operation

EXP EXP(<#>)

X ----------> *e^x*

NOTE: More accurate than the usual operation e^x

EXPM1 EXPM1 (< # >) "exponential of # minus one"

X --------- > *exp (x) - 1*

NOTE: More accurate near x=0 than with usual operations.

LNP1 LNP1(< # >) logarithm of (# plus one)"

X --------- > *ln (x + 1)*

NOTE: More accurate near x = 0 than with usual operations.

List

CONCAT

CONCAT(< {} > ,< {} >) Concatenate two lists.

{X1,...,xn}, ---------- > *(x1, …xn, y1, …, ym)*

{y1, …, ym}

NOTE: Joins two lists end to end to make a new list.

ΔLIST

ΔLIST(<{}>) List of "first differences"

{x1,x2,...,xn} ---------- > *{x2-x1,x3-x2,...xn-x(n- 1,1)}*

ΠLIST

ΠLIST (< {} >) Product of elements

{x1, …,xn} ---------- > *x1 *x2 *…*xn*

ΣLIST

ΣLIST (< {} >) Sum of elements

{x1, …,xn} ---------- > *x1 + x2 …+ xn*

MAKELIST

MAKELIST(< expr >, < name >, < # >, < # >, < # >)

expression, variable name, starting value, end value, step size

---------- >

(expr| (name = start), expr| (name = start + step), ..., expr| (name = end)}

NOTE: Creates a list by evaluating the expression for each value of the variable name obtained by stepping from the starting value to the end value by the step size.

EXAMPLE:

MAKELIST(X^2,X,1,5,2) creates {1,9,25}

POS

POS(< {} >, < N# >) Position of value in list

{x1,...,xn}, y ---------- > *index value i such that xi = y,* or 0 if no such I exists

SIZE

SIZE(< {} >)

list ---------- > *size of the list (# of elements)*

SORT

SORT(< {} >)

list ---------- > *list sorted in increasing order*

Loop

ITERATE

ITERATE(< expr >, < name >, < R# >, < N# >)
expression, name, x0, n ---------- > *iterates the function expression in variable name n times starting with input x0*
EXAMPLE: ITERATE(X^2,X,5,3) results in 390625 because we have 5^2 = 25, 25^2 = 625, and 625^2 = 390625.

RECURSE

RECURSE(< name>, <expr>, <N#>, <N#>)
index variable, Nth term, first term, second term
---------- >
allows you to store a sequence definition from within a program or from *the HOME screen.*
EXAMPLE: RECURSE(N, N^2, 1, 4) STO> U1(N)

Σ

Σ(< name > = < N# >, < N# >, < expr >)
index n, starting index n1, ending index n2, expression for nth term
---------- >
summation of expression in index variable n from *n = n 1 to n = n2*

Matrix

COLNORM

COLNORM(<M#>)
matrix [[M]] ---------- > *column-norm of M*

COND

COND(<M#>)
matrix [M] ---------- > *condition number of M*

CROSS

CROSS(<V#>,<V#>)
vectors [v1], [v2} ---------- > *cross-product of v1 and v2*

DET

DET (<M#>)
matrix [M] ---------- > *determinant of M*

DOT

DOT *(<V#>,<V#>)*
vectors [v1], [v2] ---------- > *dot-product of v1 and v2*

EIGENVAL

EIGENVAL(<M#>)
matrix [[M]] ---------- > [eigenvalues] (vector of eigenvalues of M)

EIGENVV EIGENVV(<M#>)
matrix [[M]] ----------> *{[[eigen vectors]], [eigen values])*

NOTE: list of matrix of eigenvectors and a vector of eigenvalues

IDENMAT IDENMAT (<N#>)
Integer n ----------> *Creates an nxnidentity matrix*

INVERSE INVERSE (<M#>)
Matrix [[M]] ----------> *multiplicative inverse M^-1*

LQ LQ(< M# >)
matrix [[M]] ----------> *{ [[L]] [[Q]] [[P]] }*

LSQ LSQ*(<M#>,<M#>)*
matrices [[B]], [[A]] ----------> *matrix [[X]]*

Note: X is the least squares solution of A* X= B

LU LU(<M#>)
matrix [[M]] ---------- > *{ [[L]] [[U]] [[P]] }*

NOTE: result is list of three matrices L, U, and P, where P*L*=M and L is lower-triangular, U is upper-triangular with 1's on the main diagonal, and P is a permutation matrix

MAKEMAT MAKEMAT(<expr>, <n1#>, <n2#>)
expression in I and J, n1 (number of rows), n2 (number of columns)
--------->
matrix M(I,J) where the (I,J) entry is the value of the expression

QR QR<M#>)
matrix [[M]] ---------> *{ [[Q]] [[R]] [[P]] }*
NOTE: result is list of three matrices G, R, and P, where Q is orthogonal, R is triangular, and P is a permutation

RANK RANK(<M#>)
matrix [[M]] ---------- > *computed rank of M*

ROWNORM ROWNORM(<M#>)
matrix [[M]] ---------- > *row-norm of M*

RREF RREF(<M#>)
matrix [M] ---------> *reduced row-echelon form of M*

SCHUR SCHUR*(<M#>)*
matrix [[M]] ---------> *{ [[Q]] [[U]] }*

NOTE: result is list of two matrices where Q*U*Q^H = M

SIZE SIZE(<M#>)
matrix [[M]] -- ------- > *(n1, n2) (size of the matrix M)*

SPECNORM SPECNORM(<M#>)
matrix [[M]] ---------- > *spectral norm of M*

SPECRAD SPECRAD(<M#>)
matrix [M] ---------- > *spectral radius of M*

SVD SVD(< M# >)
matrix [[M]] ---------- > *{ [[U]] [[V]] [S] }*

NOTE: result is a list of two matrices and a vector where U^H * M *V^H = diag(S) and U * diag(S) * V = M

SVL SVL(<M#>)
matrix [[M]] ---------- > *vector [S]*

NOTE: Result is the vector [S] described in SVD as above

TRACE TRACE(<M#>)
matrix [[M]] ---------- > *trace of M*

TRN TRN(<M#>)
matrix [M] ---------- >*matrix transpose of M*

Polynom

POLYCOEF

POLYCOEFF(<V#>)
[Rn, …,R1] ----------> *[An, … A1, A0]*
vector of roots — *vector of coefficients*

NOTE: The r's are the roots of the polynomials aNx^N + A(N-1) * x6(N-1) + … = a1x = a0

POLYEVAL

POLYEVAL(<V#>,<#>)
[aN, …, a1, a0], r ----------> *aNr^N + … + A1R + A0*
vector of coefficients, real number — *value of polynomial*

POLYFORM

POLYFORM(<expr>, <name., <name> …,)
expression, x, y, … , ----------> *expression*

NOTE: Reslut is a polynomial in x whose coefficients are Polynomials in y whose coefficients are …

EXAMPLE: POLYFORM(X^2 * Y^2 + X^2 * Y,X)

Results in the polynomial (Y^2 + Y) * X^2

POLYROOT

POLYROOT(<V#>)
[aN, …, a0] ----------> *[rN, …, r1]*
vector of coefficients — *vector of roots*

NOTE: The r's are the roots of the polynomial aNx^N + A(N-1) * x^(N-1) + … + a1x + a0

Prob

COMB

COMB(<N#>, <N#>) — combinations
integers n, k ----------> *n choose k*

(n *(n-1) * …(n-k + 1))
(1 * 2 * … (k-1) * k

!

(<N#>)! — factorial
integer n ----------> *n!*
*(1 * 2 * … * n)*

PERM

PERM(<N#>, <N#>) — permutations
Integers n, k ----------> *n * (n - 1) * … (n – k + 1)*

RAND

RAND
No arguments ----------> *random # between 0 and 1*

Prob	*Continued*		
UTPC	UTPC(< N#>, <R#>) *integer N, real X*	---------- >	*upper-tail chi-square distrib. of order N evaluated at X*
UTPF	UTPF(< N#>, < N#>, <R#>) *integers N1, N2,*	---------- >	*upper-tail F-distrib. of real X order N1, N2, evaluated at X*
UTPN	UTPN(< R#>, <R#>, < R#>) *reals m, v, X*	---------- >	*upper-tail normal distrib. of mean m, variance v evaluated at X*
UTPT	UTPT(< N#>, <R#>) *Integer N, real X*	---------- >	*upper-tail student's T-distrib. N of order N evaluated at X*

Real			
CEILING	CEILING(<R#>) *real number x*	---------- >	*least integer* ^{3} *x*
DEG→RAD	DEG→RAD(<R#>) *real number x*	---------- >	*convert x degrees to corresponding radians*
FLOOR	FLOOR(< R# >) *real number x*	---------- >	*greatest integer* ^{2} to *x*
FNROOT	FNROOT(< expr >, < name >, < R# >) *expression, x,xO*	---------- >	*r such that exp /* (x = r) = 0. *Search is started near x0.*
FRAC	FRAC(<R#>) *real number x*	---------- >	*fractional part of x*
HMS→	HMS→(< expr >) *real number x*	---------- >	*converts hours/minutes/ seconds format to decimal hours*
→HMS	→HMS (< expr >) *real number x*	---------- >	*convert decimal hours to hours/minutes/seconds format.*

INT	INT(<R#>)		
	real number x	---------- >	*integer part of x*
	EXAMPLE:	INT(23.438) returns 23 as result	
MANT	MANT(<R#>)		
	real number x	---------- >	*mantissa of x*
MAX	MAX(<R#>, <R#>)		
	real numbers x, y	---------- >	*maximum of x, y*
MIN	MIN(<R#>, <R#>)		
	real numbers x, y	---------- >	*minimum of x, y*
MOD	MOD(<N#> MOD <N#>)		
	integers m, n	---------- >	*integer remainder when m is divided by n*
	EXAMPLE:	27 MOD 4 returns a result of 3	
%	%(<R#>, <R#>)		
	real numbers x, p	---------- >	*x * p * 0.01*
%CHANGE	%CHANGE(<R#>, <R#>)		
	real numbers x, y	---------- >	*100 * (y-x) / x*
%TOTAL	%TOTAL(<R#>, <R#>)		
	real numbers x, y	---------- >	*100 * y / x*
RAD→DEG	RAD→DEG (<R#>)		
	real number x	---------- >	*converts radians to degrees*
ROUND	ROUND (<R#>, <N#>)		
	real x, integer n	---------- >	*rounds x to n units*
SIGN	SIGN (<#>)		
	real or complex x	---------- >	*x / ABS(x) if x ≠ 0 0 if x = 0*
TRUNCATE	TRUNCATE (<R#>, <N#>)		
	real x, integer n	---------- >	*truncates x to n units*
XPON	XPON (<R#>)		
	Real number x	---------- >	*exponent of x when written In scientific notation.*

Stat-Two

PREDX

PREDX(< name >, < # >)
name, y
---------- >
predicted value for the "independent" variable of the indicated statistical dataset, given the "dependent" value

PREDY

PREDY(< name >, < # >)
name, x
---------- >
predicted value for the "dependent" variable of the indicated statistical dataset, given the "independent" value

Symbolic

=

This is an equational operator used to define equations like A + C = C/2, but it is not a predicate (like = = in Test).

ISOLATE

ISOLATE(< expr >, < name >)
expression or equation, name of a specific variable
--------- >
symbolic expression for specific variable in terms of the others

NOTE: An expression is interpreted as an equation with the expression on one side and 0 on the other side of the = .

LINEAR?

LINEAR?(<name>)
expression name --------- > *flag indicating whether the expression is linear*

QUAD

QUAD(< expr >, < name >)
quadratic expression or equation, name of a specific variable
--------- >
symbolic expression for the two complex roots

NOTE: An expression is interpreted as an equation with the expression on one side and 0 on the other side of the = .

QUOTE

QUOTE(<expr>)
expression --------- > *Suppress the expression eval.*

NOTE: Single quote marks also work as in '<expr>'

(WHERE)

│<expr> │ (<name> = < # >,<name> = < # >, …) "substitution"
expression x, p 1, y, p2,..,
--------- >
expression *evaluated where x =p 1, y =p2, etc.*

Test

<	*<R#> < <R#>*		
	a, b	---------- >	*1 if a < b, 0 otherwise*
≤	<R#> ≤ <R#>		
	a, b	---------- >	*1 if a≤ b, 0 otherwise*
= =	*<R#> = = <R#>*		
	a, b	----------- >	*1 if a = b, 0 otherwise*
≠	*<R#> ≠ <R#>*		
	a, b	---------- >	*1 if a ≠ b, 0 otherwise*
>	<R#> > <R#>		
	a, b	---------- >	*1 if a> b, 0 otherwise*
≥	<R#> ≥ <R#>		
	a, b	------- >	*1 if a≥ b, 0 otherwise*
AND	<R#> AND <R#>		
	a, b	---------- >	*1 if both a and b ≠0, 0 otherwise*
IFTE	IFTE(< predicate >, <true-clause >, < false-clause >)		
	a, expression 1, expression 2	---------- >	*expression 1 if a ≠ 0 expression 2 otherwise*
NOT	NOT <R#>		
	a	---------- >	*1 if a = 0, 0 otherwise*
OR	<R#>OR<R#>		
	a, b	---------- >	*1 if a or b a non-zero 0 otherwise*
NOR	<R#>XOR<R#> "exclusive or"		
	a, b	---------- >	*1 if a a ≠ 0 or b ≠ 0, but not both, 0 otherwise*

Trig

ACOT	*ACOT(<#>)*		
	x	---------- >	*inverse cotangent of x*
ACSC	*ACSC(<#>)*		
	x	---------- >	*inverse cosecant of x*
ASEC	ASEC(< # >)		
	x	---------- >	*inverse secant of x*
COT	*COT(< # >)*		
	x	--------- ~ >	*cotangent of x*
CSC	*CSC(< # >)*		
	x	---------- >	*cosecant of x*
SEC	*SEC(< # >)*		
	x	---------- >	*secant of x*

<u>Functions which are not in the MATH menu and not on the keyboard</u>:

NEG NEG(< # or V# or M# or {} or expr or grob >)

NOTE: same functionality as -x key.