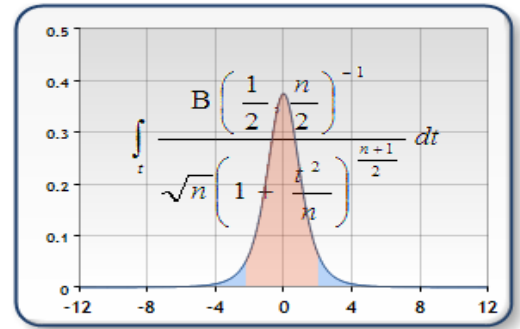


Extended Statistics HP-41 Module



Overview

Most of the functions in this ROM were included in the former version of the “Curve Fitting & Statistics” Module. The changes made to the Curve Fitting functionality in that module were also the perfect excuse to separate the statistics contents into its own ROM, dedicated to statistics and probability.

Some functions are also in the SandMath, a real Math powerhouse – but a few ones have been written anew; and yet more new functions have been added as result of the collaboration with Greg McClure on his GJM Module.

This module has three distinct sections. The first one includes the more fundamental functions covering sums, means, and other basic numeric calculations - such as linear regression, combinations/permutations, etc. Then it moves into the distributions section where you can find functions to calculate the density and probability functions for the most important distributions. Finally a third section includes the four Primality functions as well as other auxiliary functions and utilities related with the topic at hand.

Without further ado, let’s see the functions included in the module. Refer to the individual function descriptions later on for details on the syntax and use instructions – always a tricky think when it comes to the statistics topics.

XROM	Function	Description	Input	Author
06,00	-X STAT 1E	<i>Section Header</i>	<i>n/a</i>	--
06,01	%T	Total percentage	y,x in Y,X	<i>Poul Kaarup</i>
06,02	Σ0	Sum of mantissa digits	value in X	<i>Ángel Martin</i>
06,03	Σ1	Sum of N integers	N in X	<i>Poul Kaarup</i>
06,04	Σ1/N	Harmonic Number	n in X	<i>Ángel Martin</i>
06,05	Σ2	Sum of squares of N integers	N in X	<i>Poul Kaarup</i>
06,06	Σ3	Sum of cubes of N Integers	N in X	<i>Poul Kaarup</i>
06,07	ΣX^N	Geometric Sums	N in Y, X in X	<i>Ángel Martin</i>
06,08	ΣRG?	Stat Reg Location	none	<i>Ken Emery</i>
06,09	AGM	Arithmetic-GeometricMean	x,y in Stack	<i>Ángel Martin</i>
06,10	AMEAN	Arithmetic Mean	bbb.eee in X	<i>Ángel Martin</i>
06,11	CNK	Combinations	arguments in X,Y	<i>Ángel Martin</i>
06,12	CORR	LR Correlation	Σ+ Data Set	<i>JM Baillard</i>
06,13	COV	LR Covariance	Σ+ Data Set	<i>JM Baillard</i>
06,14	EVEN?	Tests for even value	value in X	<i>Ángel Martin</i>
06,15	GCD	Greater Common Divisor	arguments in X,Y	<i>Ángel Martin</i>
06,16	GHM	Geometric-Harmonic Mean	x,y in Stack	<i>Greg McClure</i>
06,17	GMEAN	Geometric Mean	bbb.eee in X	<i>Ángel Martin</i>
06,18	HMEAN	Harmonic Mean	bbb.eee in X	<i>Ángel Martin</i>
06,19	LCM	Least Common Multiple	arguments in X,Y	<i>Ángel Martin</i>
06,20	L1	Shows line equation in ALPHA	a,b in Stack	<i>Ángel Martin</i>
06,21	LR	Linear Regression	Σ+ Data Set	<i>JM Baillard</i>

Extended Statistics Module

06,22	LRX	LR X-Value	$\Sigma+$ Data Set, intercept	<i>Ángel Martin</i>
06,23	LRY	LR Y-Value	$\Sigma+$ Data Set, abscissa	<i>JM Baillard</i>
06,24	ODD?	Tests for odd value	value in X	<i>Ángel Martin</i>
06,25	PMEAN	Generalized Power Mean	p in Y; bbb.eee in X	<i>Ángel Martin</i>
06,26	PNK	Permutations	arguments in X,Y	<i>Ángel Martin</i>
06,27	RCLΣ	Stat Reg to Stack	none	<i>JM Baillard</i>
06,28	STLINE	Straight Line equation	(x1,y1) & (x2,y2) in stack	<i>Ángel Martin</i>
06,29	ST<>Σ	Swap Stack & Stat Registers	none	<i>Nelson F. Crowle</i>
06,30	-DISTRIBTN	<i>Section Header</i>	<i>n/a</i>	<i>--</i>
06,31	BIN	Binomial Distribution	Prompted by program	<i>Ángel Martin</i>
06,32	"BNP"	Binomial P(n,p,x)=k	n,x in Y, X	<i>JM Baillard</i>
06,33	"BNP+ "	Binomial P(n,p,x)<=k	n,k,x in stack	<i>JM Baillard</i>
06,34	C2CP	Chi-Square Probability Fnc.	n,x in Y, X	<i>JM Baillard</i>
06,35	C2DF	Chi-square Density Function	n,x in Y, X	<i>Ángel Martin</i>
06,36	CHI2	Chi^2 Distribution Driver	prompts for u value	<i>HP Co. (Stat Pac)</i>
06,37	"FDST"	Snedecor's F-Distribution Driver	menu: : "N1 N2 Q"	<i>HP Co. (Stat Pac)</i>
06,38	FSDF	Snedecor's F Density Function	Z: N1. Y: N2, X: Point	<i>Eugenio Úbeda</i>
06,39	"FSCP"	Snedecor's F Probability Fnc.	n,x in Y, X	<i>Eugenio Úbeda</i>
06,40	NRDF	Normal Density (m,s)	Z: Mean, Y: Sdev, X: Point	<i>Ángel Martin</i>
06,41	NRCP	Normal Probability (m,s)	Z: Mean, Y: Sdev, X: Point	<i>Ángel Martin</i>
06,42	POIS	Poisson Distribution	prompts for values	<i>Ángel Martin</i>
06,43	PSDF	Poisson Density Function	Y: Mean, X: #Events	<i>Ángel Martin</i>
06,44	"PX"	Standard Normal Probability	Argument in X	<i>Michael Faulhaber</i>
06,45	"PXX"	Probability between -x and x	Argument in X	<i>Michael Faulhaber</i>
06,46	QNTL	Quantile - inverse of NRCP	Argument in X	<i>Ángel Martin</i>
06,47	"TDIST"	T-Distribution Driver program	Menu driven	<i>HP Co. ??</i>
06,48	TSDF	Student's T Density Function	N in Y, Point in X	<i>Eugenio Úbeda</i>
06,49	TSCP	Student's T Probability Func.	n,x in Y, X	<i>Eugenio Úbeda</i>
06,50	"XP"	Inverse of PX	Argument in X	<i>Michael Faulhaber</i>
06,51	"XXP"	Inverse of PXX	Argument in X	<i>Michael Faulhaber</i>
06,52	-PRIMALITY	<i>Section Header</i>	<i>n/a</i>	<i>--</i>
06,53	1/GM	Inverse Gamma Function	Argument in X	<i>JM Baillard</i>
06,54	BETA	Beta Function	Arguments in X,Y	<i>JM Baillard</i>
06,55	D%	Difference Percent	x,y in Stack	<i>Ángel Martin</i>
06,56	ERF	Error Function	x in X	<i>Martin - Baillard</i>
06,57	GMF	Gamma Function	Argument in X	<i>JM Baillard</i>
06,58	NXTPRM	Next Prime	value in X	<i>PoulKaarup</i>
06,59	PFACT	Prime Divisor (factor)	Argument in X	<i>Peter Platzer</i>
06,60	PTWIN	Next Twin Primes	Argument in X	<i>Peter Platzer</i>
06,61	PRIME?	Prime Test	Argument in X	<i>Poul Kaarup</i>
06,62	RAND	Random Number (w/ Timer)	none	<i>JM Baillard</i>
06,63	"SLV"	Solve Subroutine	x1, x2 in X,Y, fname in Alpha	<i>PPC Members</i>

Note that this module was designed to be self-contained, i.e. there are no dependencies on any other one, not even the Library#4 (therefore never mind if you never heard of that one before). All required auxiliary functions for the FOCAL programs are included in the third section.

1 – Basic Statistical Functions.

Summation Functions. { $\Sigma 0$, $\Sigma 1$, $\Sigma 2$, $\Sigma 3$, $\Sigma 1/N$, ΣX^N }

Use them to calculate the resulting sum of a finite series of integer numbers (Triangular Numbers), their squares (Squared Pyramidal Numbers), cubes (Squared Triangular Numbers), or generalized (Faulhaber's) powers. Also included is the harmonic number $H(n)$, which sums the reciprocal of the numbers – i.e. a particular case of the generalized case with exponent -1.

The formulas for the first three cases are as follows:

$$T_n = \sum_{k=1}^n k = 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

$$P_n = \sum_{k=1}^n k^2 = \frac{n(n+1)(2n+1)}{6} \quad \sum_{k=1}^n k^3 = \left(\sum_{k=1}^n k \right)^2$$

- **$\Sigma 0$** sums the mantissa digits of the number in X. For example: PI, **$\Sigma 0$** => 40
- **$\Sigma 1, \Sigma 2, \Sigma 3$** calculate the first three cases using the explicit formulas – much faster than performing the actual summation even for short series.
- **$\Sigma 1/N$** calculates the Harmonic number of the argument in X; that is the sum of the reciprocals of the natural numbers (which excludes zero) lower and equal to n. It is used in the calculation of numerous special functions, like the Kelvin and the Bessel functions of the second kind, $K(n,x)$ and $Y(n,x)$.

$$H_n = \sum_{k=1}^n \frac{1}{k}$$

Example: calculate $H(5)$ and $H(25)$.

5, **$\Sigma 1/N$** => 2.283333333
 25, **$\Sigma 1/N$** => 3.815958178

- **ΣN^X** Calculates a generalized value of the Faulhaber's formula for integer values of x. – The few first integer values of x have explicit formulas for the result, but that's not the case for a general value, which can also be non-integer. Obviously for $x=-1$ this function returns identical results than the previous one, albeit slower due to the additional complexity of the term.

Example: Check the triangular ($x=1$) and pyramidal ($x=2$) formulas for $n=10$ – which are particular cases of the Faulhaber's Formula, involving Binomial coefficients and Bernoulli's numbers. See the link below for details: http://en.wikipedia.org/wiki/Faulhaber%27s_formula

10, ENTER ^, 1, **ΣN^X** => 55.00000000
 10, ENTER ^, 2, **ΣN^X** => 385.00000000

Single and Duplex Means (to an end).

In the means department there is a very complete selection of choices: arithmetic, geometric and harmonic means are calculated on a set of data registers controlled by the control word "bbb.eee" in X- i.e. beginning and end registers, *and *not* the statistical registers as defined by ΣREG !*. Also a generalized exponential mean is available using the same syntax.

The **AMEAN**, **GMEAN**, and **HMEAN** functions calculate the means of multiple values stored in data registers. Entering the control word describing the register set in X and executing AMEAN, GMEAN, or HMEAN will result in that mean being put into X (and the control word saved in LastX). So, for example, to get one of these means for values in registers 10 thru 15, put 10.015 in X and execute the appropriate mean function.

But there is more: The **PMEAN** function is also available for a generalized mean function. The power "p" is put into Y and the control word in X, and the Generalized Power Mean is calculated for the values pointed to by the control word. The PMEAN formula is:

$$M_p(x_1, \dots, x_n) = \left(\frac{1}{n} \sum_{i=1}^n x_i^p \right)^{\frac{1}{p}}$$

For p=0 this would normally lead to a problem. However the limit for this expression as p → 0 yields the Geometric Mean, so when p=0, the GMEAN function code is used.

From the above formula you can see that p=1 yields the Arithmetic mean, and p=-1 yields the Harmonic mean. However fractional and other negative values can be used, and you will notice that as p becomes infinite (positive), the mean tends to be the MAX value of the numbers. As p becomes infinite (negative), the mean tends to be the MIN value of the numbers.

With the exception of the AMEAN program, all values used in the registers must be non-zero positive values. Otherwise a DATA ERROR will occur.

Let's move now to the duplex means on a pair of numbers placed in X and Y registers: the Arithmetic-Geometric mean **AGM** and the Harmonic-Geometric mean **HGM**. An interesting definition of the mean of two values occurs when combining Arithmetic, Geometric, and Harmonic means.

- The Arithmetic-Geometric mean is a special value, defined as the common limit of A=AMean(A,B) and B=GMean(A,B) repeated until A-B = 0.
- The Geometric-Harmonic mean is defined as the limit of A=GMean(A,B) and B=HMean(A,B) repeated until A-B = 0.

AGM calculates the Arithmetic-Geometric Mean, whilst **GHM** calculates the Geometric-Harmonic mean.

As an interesting note, AM(A,B) ≥ AGM(A,B) ≥ GM(A,B) ≥ GHM(A,B) ≥ HM(A,B).

What happened to the Arithmetic-Harmonic mean? That is simply the Geometric mean in disguise, thus no need for such function. Finally, note that taking p=0.5 in the PMEAN function (on two registers) will NOT yield the AGM (and -0.5 will NOT yield the GHM) unless, of course, the register values are identical! It is not that simple to get those values, and the power value required changes depending on the two values chosen for AGM or GHM.

Extended Statistics Module

Combinations and Permutations – two must-have classics.

Nowadays it would be unconceivable to release a calculator without this pair in the function set – but back in 1979 when the 41 was designed things were a little different. So here they are, finally and for the record.

- **PNK** calculates Permutations, defined as the number of possible different arrangements of N different items taken in quantities of K items at a time. No item occurs more than once in an arrangement, and different orders of the same R items in an arrangement are counted separately. The formula is:

$$\frac{n!}{(n - k)!}$$

- **CNK** calculates Combinations, defined as the number of possible sets of N different items taken in quantities of K items at a time. No item occurs more than once in a set, and different orders of the same R items in a set are not counted separately. The formula is:

$$\frac{n!}{k!(n - k)!}$$

The general operation includes the following enhanced features:

- Gets the integer part of the input values, forcing them to be positive.
- Checks that neither one is Zero, and that $n > k$
- Uses the minimum of $\{k, (n-k)\}$ to expedite the calculation time
- Checks the Out of Range condition at every multiplication, so if it occurs it is determined as soon as possible
- The chain of multiplication proceeds right-to-left, with the largest quotients first.
- The algorithm works within the numeric range of the 41. Example: $CNK(335, 167)$ is calculated without problems.
- It doesn't perform any rounding on the results. Partial divisions are done to calculate **CNK**, as opposed to calculating first **PNK** and dividing it by $k!$

Provision is made for those cases where $n=0$ and $k=0$, returning zero and one as respective results. This avoids DATA ERROR situations in running programs, and is consistent with the functions definitions for those singularities.

Note as well that there is no final rounding made to the result. This was the subject of heated debates in the HP Museum forum, with some good arguments for a final rounding to ensure that the result is an integer. The SandMath implementation however does not perform such final "conditioning", as the algorithm used seems to always return an integer already. Pls. Report examples of non-conformance if you run into them.

Example: Calculate the number of sets from a sample of 335 objects taken in quantities of 167:

Type: 335, ENTER ^, 167, XEQ "CNK" -> 3,0443587 99

Example: How many different arrangements are possible of five pictures, which can be hung on the wall three at a time:

Type: 5, ENTER ^, 3, XEQ "PNK" -> 60,00000000

The execution time for these functions may last several seconds, depending on the magnitude of the inputs. The display will show "RUNNING..." during this time.

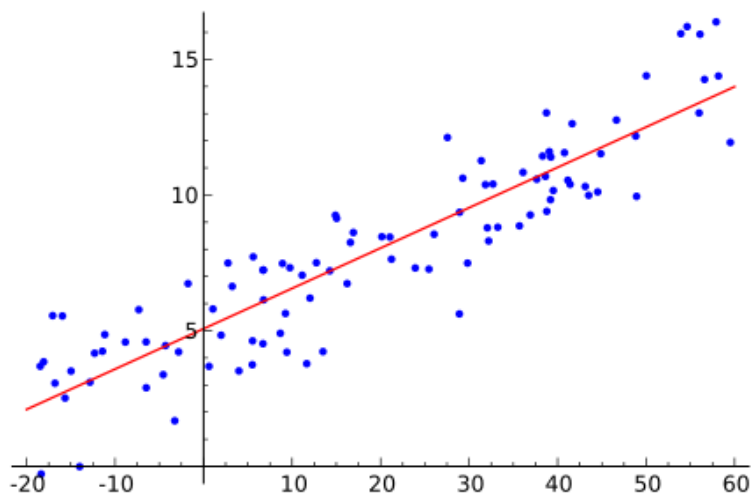
Extended Statistics Module

Linear Regression – Let's not digress.

The following four functions deal with the Linear Regression, the simplest type of the curve fitting approximations for a set of data points. They complement the native set, which basically consists of just **MEAN** and **SDEV**.

	Function	Description	Author
[ΣΣ]	CORR	Correlation Coefficient of an X,Y sample	JM Baillard
[ΣΣ]	COV	Covariance of an X,Y sample	JM Baillard
[ΣΣ]	LR	Linear Regression of an X,Y sample	JM Baillard
	LRX	X-Value for a Y point	Ángel Martín
[ΣΣ]	LRY	Y- value for an X point	JM Baillard

Linear regression is a statistical method for finding a straight line that best fits a set of two or more data pairs, thus providing a relationship between two variables. Using the well-known method of least squares, **LR** will calculate the slope A and Y-intercept B of the linear equation: $Y = Ax + B$.



Results are placed in Y and X registers respectively. When executed in RUN mode the display will show the straight-line equation, similar to the **STLINE** function described before.

Example: find the y-intercept and slope of the linear approximation for the data set given below:

X	0	20	40	60	80
Y	4.63	5.78	6.61	7.21	7.78

Assuming all data pairs values have been entered using Y-value, ENTER^, X-value, **Σ+**; we type: XEQ "LR" -> 0,038650000 and X<>Y -> 4,856000000 producing the following output in FIX 2:



As to the remaining functions, **COV** calculates the sample covariance. **CORR** returns the correlation coefficient, and **LRY** the linear estimate for the function at the given point.

For the same sample still in the calculator's memory, we obtain the values:

$$\text{Covariance} = 38.65; \quad \text{CORR} = 0.987954828; \quad \text{LRY} = 4.894184454 \text{ (using Corr value as X)}$$

Ratios and other Tools.

- **%T** and **D%** are miniature functions to calculate the percent of a number relative to another one (its reference), and the delta percent between the numbers in Y(reference) and X(new value). The formulas are:

$$\%T(y,x) = 100 x / y ; \quad D\% = 100 (x-y) / x$$

Example: the relative percent of 4 over 25 is 16%.- You type: 25, ENTER^, 4, XEQ "**%T**"

Example: the delta percent of a change from 85 to 75 is -11,765%

- **GCD** and **LCM** are fundamental functions also inexplicably absent in the original function set. They are short and sweet, and certainly not complex to calculate. The algorithms for these functions are based on the PPC routines **GC** and **LM** – conveniently modified to get the most out of MCODE environment.

If a and b aren't both zero, the greatest common divisor of a and b can be computed by using least common multiple (lcm) of a and b:

$$\gcd(a, b) = \frac{a \cdot b}{\text{lcm}(a, b)}$$

Examples: $\text{GCD}(13,17) = 1$ (primes), $\text{GCD}(12,18) = 6$; $\text{GCD}(15,33) = 3$

Examples: $\text{LCM}(13,17) = 221$; $\text{LCM}(12,18) = 36$; $\text{LCM}(15,33) = 165$

- **ST<>Σ** exchanges the contents of five statistical registers and the stack (including L). Use it as a convenient method to review their values when knowing their actual location is not required.
- **RCLΣ** recalls the contents of five statistical registers to the stack (including L). It is therefore just one half of the previous function.
- **ΣRG?** returns to the X register the current pointer to the statistics registers block. It is identical to the function ΣREG? in the 41CX.
- **ODD?** and **EVEN?** are simple tests to check whether the number in X is odd or even. The answer is YES / NO, and in program mode the following line is skipped if the test is false. The implementation is based on the MOD function, using $\text{MOD}(x,2) = 0$ as criteria for evenness.
- **RAND** is a random number generator that uses the current time as seed. If the Time Module is not present the function uses the value in X as seed. The result value is within 0 and 1.
- **STLINE** calculates the slope "m" and 0-intercept "p" of a straight line that passes through the points (x1, y1) and (x2, y2) stored in the stack registers. The equation is: $Y = m X + p$
- **L1** puts the straight line equation in the Alpha register, using the values in X- and Y- as slope and 0-intercept for the line. This function is used internally in the Linear Regression and **STLINE** functions.

Extended Statistics Module

Primes and other Relatives.

- **PRIME?** Is a primality test function, which will return YES/NO in run mode and skip a line if false in a running program.
- **NXTPRM** Is a fast and accurate method to obtain the next prime following the value in X (thus not itself in case it already is a prime number).
- **PTWIN** is similar to NXTPRM, only that it'll find the next Twin primes following the value in X. Those are two consecutive prime numbers p_1 and p_2 , such that $p_2 = (p_1 + 2)$.
- **PFACT** Is a fast and accurate method to obtain prime divisors of the value in X. This function can be used repeatedly to obtain the complete prime factor decomposition, see the short FOCAL program example provided below.

Line	Instruction	Line	Instruction
01	LBL "PFCTR"	13	X=Y?
02	CLA	14	GTO 05
03	AINT	15	" -*"
04	" - ="	16	/
05	LBL 00	17	GTO 00
06	ENTER^	18	LBL 05
07	PFACT	19	AVIEW
08	AINT	20	END

As an exercise, improve this program so that the repeated factors are grouped in an exponent, as opposed to in the single fashion listing in ALPHA. You can refer to the SandMath and SandMatrix modules for more sophisticated Prime Factorization programs.

2 – Statistical Distributions.

Moving on to the next section, the following pages describe the different statistical distribution included in the module. Some will have individual functions to perform the calculations independently, whilst other programs are of the “driver” type – providing a micro environment to do calculations related to a given distribution.

It is important to note the naming conventions for these functions. In general there are three types:

- Distribution density function – use “**DF**” as last two letters of function name
- Distribution Cumulative Probability – use “**CP**” as last two letters of function name
- Distribution “driver” program - some include the word “DIST” or “DST” in their names

The following table shows a summary of the available functions by type:

Distribution	Density Function	Cumulative Probability	Driver
Normal	NRDF	NRCP	--
T-Student	TSDF	TSCP	TDIST
F-Snedecor	FSDF	FSCP	FDIST
Chi-Squared	C2DF	C2CP	CHI2
Poisson	PSDF	--	POIS
Binomial	BNP	BNP+	BIN

While some of these functions are implemented as MCODE routines, others are a combination of some module functions in short FOCAL “wrappers” – which appear as MCODE functions in the module’s FAT and in your own FOCAL code as program steps. Note however that they likely utilize data registers and don’t always follow the “argument to LastX” rule due to their hybrid nature.

A number of auxiliary functions are provided to facilitate the calculation of the distributions listed above. The more important ones are the error function $\text{erf}(x)$ – implemented in MCODE as a particular case of the hypergeometric function ${}_2F_2$, as well as Euler’s Gamma and Beta functions (and inverse Gamma as well for convenience). These last two use a continuous fractions method for the calculation – slightly faster than the Lanczos formula and for the most part with comparable accuracy.

Unfortunately there wasn’t enough room left in the module to also include the Incomplete Gamma and Incomplete Beta functions, therefore some approximation formulas are used for the cumulative probability of the T- and F- distributions. Note however that these two functions are included in the SandMath Module in case you need them.

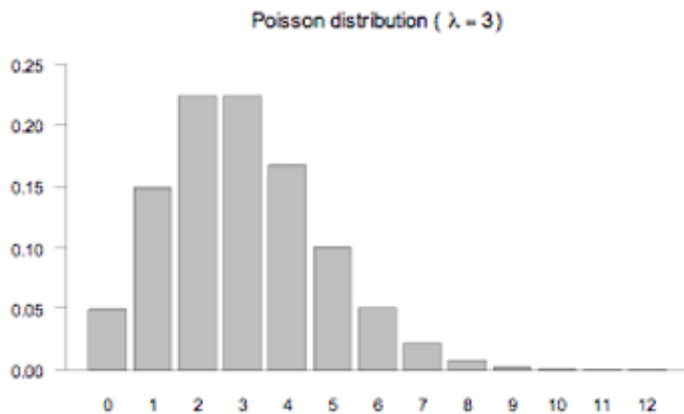
Extended Statistics Module

Poisson Standard Distribution. { PSD , POIS }

PSD is the Statistical function that calculates the Poisson Standard Distribution. In probability theory and statistics, the Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time and/or space if these events occur with a known average rate and independently of the time since the last event

A discrete stochastic variable X is said to have a Poisson distribution with parameter $\lambda > 0$, if for $k = 0, 1, 2, \dots$ the probability mass function of X is given by:

$$f(k; \lambda) = \Pr(X = k) = \frac{\lambda^k e^{-\lambda}}{k!},$$



Its inputs are k and λ in stack registers X and Y. PSDF's result is the probability corresponding to the inputs.

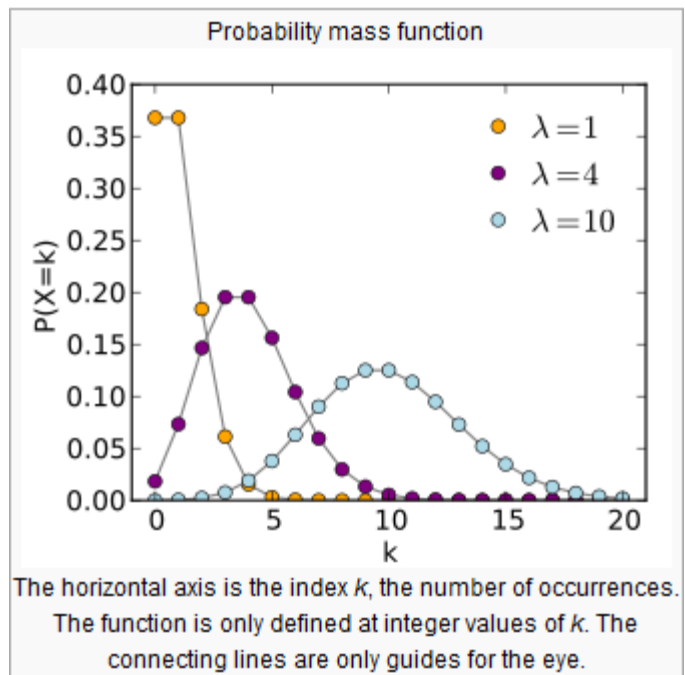
Example 1.-

Calculate the probability mass function for a Poisson distribution with parameters: $\lambda=4$, $k=5$

4, ENTER^, 5, XEQ "PSD"
Returns: 0.156293452

Example 2: do the same for $\lambda=10$ and $k=10$

10, ENTER^, XEQ "PSD"
Returns: 0.125110036



Extended Statistics Module

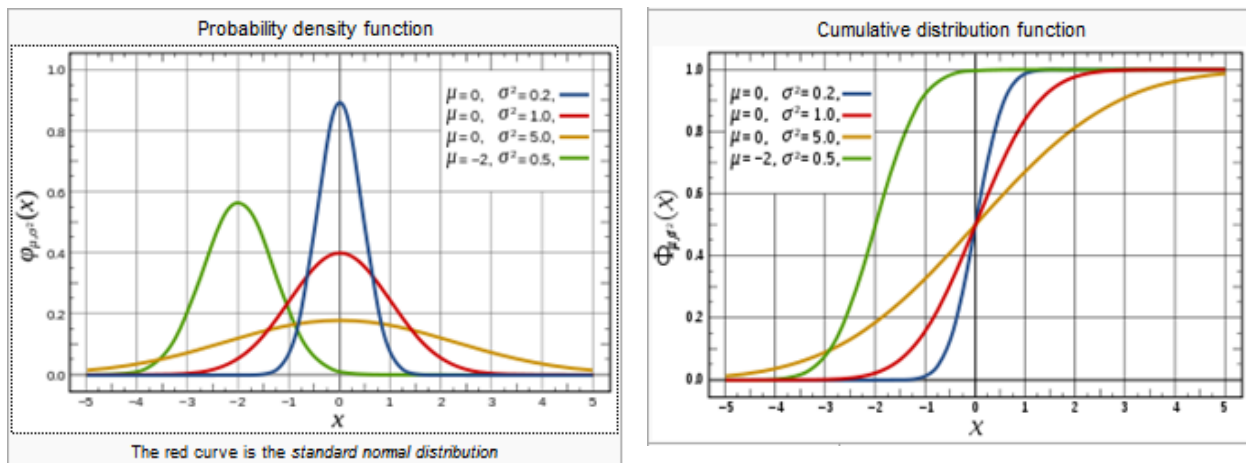
Normal Distribution Functions. { **NRDF** , **NRCP** }

In probability theory, the normal (or Gaussian) distribution is a continuous probability distribution that has a bell-shaped probability density function, known as the Gaussian function or informally as the bell curve:

$$f(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

The parameter μ is the mean or expectation (location of the peak) and σ^2 is the variance. σ is known as the standard deviation. The distribution with $\mu = 0$ and $\sigma^2 = 1$ is called the standard normal distribution or the unit normal distribution

NRDF expects the mean and standard deviation in the Z and Y stack registers, as well as the argument x in the X register. Upon completion x will be saved in LASTx, and $f(\mu, \sigma, x)$ will be placed in X. It has an all-MCODE implementation, using 13-digit routines for increased accuracy.



The figures above show both the density functions as well as the cumulative probability function for several cases. The Error function **ERF** can be used to calculate the **NRCP** – no need to apply brute force and use **NRDF** in an **INTEG**-like scenario, much longer to obtain or course. The relation to use is:

$$F(x; \mu, \sigma^2) = \Phi\left(\frac{x - \mu}{\sigma}\right) = \frac{1}{2} \left[1 + \operatorname{erf}\left(\frac{x - \mu}{\sigma\sqrt{2}}\right) \right], \quad x \in \mathbb{R}.$$

Example program: The routine below calculates CPF. Enter μ , σ , and x values in the stack.

```

01 LBL "NRCP"      08 /
02 RCL Z           09 ERF
03 -              11 INCX
04 X<>Y           12 2
05 /              13 /
06 2              14 END
07 SQRT
    
```

Extended Statistics Module

Cumulative Probability Function and its Inverse. { **PX** , **PXX** , **XP** , **XXP** , **QNTL** }

The following routines perform the calculation of the cumulative probability and its inverse, i.e. to obtain the value x that yields a given value for the probability. The inverse functions are based on the Newton or Halley's methods to iterate for the solution.

As a convenience shortcut, functions **PXX** and **XXP** are provided to obtain the CENTRAL probability within a certain zone of the arguments $[-x, x]$ – and the inverse value of X to yield said probability. The expression to obtain this is given by the formula: $PXX = 2 P(x) - 1$

Note that all these functions *assume a Standard Normal distribution*, with zero mean and standard deviation equal to 1, that is: $\sigma=1$ and $\mu=0$

Newton's method is implemented as per the PPC ROM routine **SLV** – also contained in this module. Refer to the PPC ROM manual for details on its operation if used independently from these functions.

Halley's method uses the following expression to calculate the successive approximations to the root:

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2[f'(x_n)]^2 - f(x_n)f''(x_n)}$$

where our function in this case is $f(x) = [\text{NRCP}(x) - \text{Value}]$, thus we take advantage of the fact:

$$\begin{aligned} f'(x) &= \text{NRDF} \text{ and} \\ f''(x) &= -k f'(x); \end{aligned}$$

and therefore the general expression above gets simplified considerably since $f'(x)$ is in both numerator and denominator and thus cancelled out.

Examples. Which argument yields a probability of 75% for a Standard Normal distribution?

- a) Using **XP**: 0.75, XEQ"**XP**" -> 0,674489750
- b) Using **QNTL**: 0.75, XEQ"**QNTL**" -> 0,674489750

What is the cumulative probability for the argument obtained in the previous example?

Type: 0, ENTER ^, 1, RCL Z, XEQ "**NRCP**", -> 0,750000000

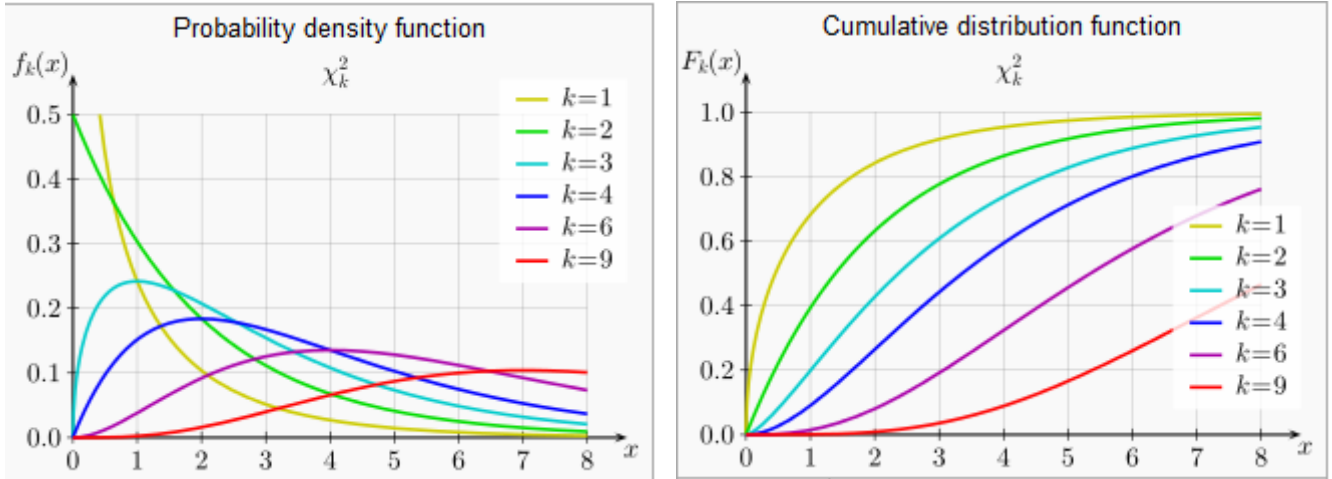
The accuracy is quite good, also holding up well across the entire range of values for both functions – thanks to the iterative methods employed. Execution speed is slightly faster for **XP** than for **QNTL**, but this one is more accurate for arguments in the vicinity of 1.

Note: By comparison with these functions, the SandMath module uses the inverse Error Function **IERF** to calculate the inverse probabilities. That method is much faster and slightly more accurate – but there wasn't enough room left in this module to implement it.

Extended Statistics Module

Chi-Squared Distribution { **C2DF** , **C2CP** , **CHI2** }

In probability theory and statistics, the chi-squared distribution (also chi-square or χ^2 -distribution) with k degrees of freedom is the distribution of a sum of the squares of k independent standard normal random variables. It is a special case of the gamma distribution and is one of the most widely used probability distributions in inferential statistics, e.g., in hypothesis testing or in construction of confidence intervals. When it is being distinguished from the more general non-central chi-squared distribution, this distribution is sometimes called the central chi-squared distribution.



There are three functions available in the XMSTAT module dealing with this distribution.

- This first two **C2DF** and **C2CP** are independent implementations for the density and probability functions respectively, using the values in Y (degrees of freedom) and X (argument). The result is placed in X but no LastX is done.
- The third and last **CHI2** is a driver routine to calculate both the density function and the cumulative probability. After prompting for the parameter μ it presents the simple menu of choices shown below:



Simply enter the argument and press **[A]** for the Probability distribution function (same as **CHDF**), or **[B]** for the cumulative distribution function $P(x)$. See the formulas below:

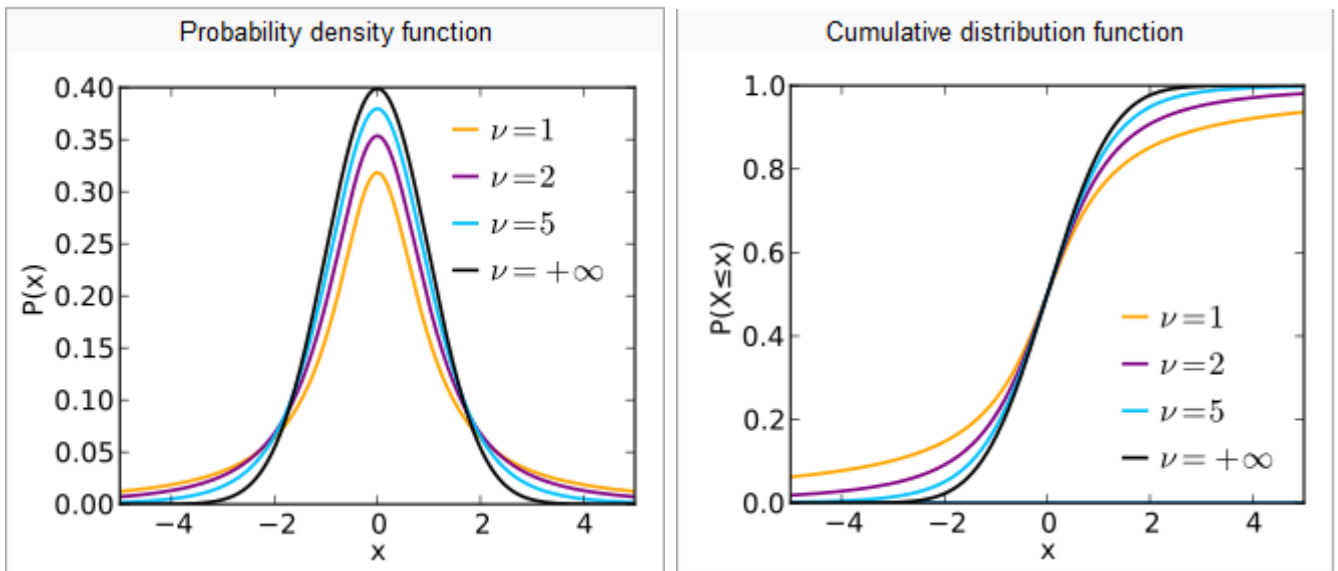
PDF	$\frac{1}{2^{\frac{k}{2}} \Gamma\left(\frac{k}{2}\right)} x^{\frac{k}{2}-1} e^{-\frac{x}{2}}$	CDF	$\frac{1}{\Gamma\left(\frac{k}{2}\right)} \gamma\left(\frac{k}{2}, \frac{x}{2}\right)$
------------	---	------------	--

Examples: CHDF(2, 0) = 0.5 and CHCP (2,1) => P=0.393469341

Note that for the value combination $m=2$ and $X=0$ the built-in power function Y^X will return DATA ERROR. This has been overcome in CHDF (but not in the driver program) by using an alternative version of Y^X that returns $0^0=1$.

Student's T-Distribution { **TSDF** , **TSCP** , **TDST** }

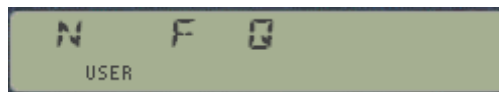
In probability and statistics, Student's t-distribution (or simply the t-distribution) is any member of a family of continuous probability distributions that arises when estimating the mean of a normally distributed population in situations where the sample size is small and population standard deviation is unknown. It was developed by William Sealy Gosset under the pseudonym Student. Whereas a normal distribution describes a full population, t-distributions describe samples drawn from a full population; accordingly, the t-distribution for each sample size is different, and the larger the sample, the more the distribution resembles a normal distribution.



There are three functions available in the XMSTAT module dealing with this distribution.

- This first two are independent functions to calculate the density function (**TSDF**) and the CENTRAL probability (**TSCP**) – between $[-x, x]$ - independently, taking the input parameters from Y (degrees of freedom) and X (argument). The result is placed in X but no LastX is done.
- The third one (**TDIST**) is also a driver program in the same guise as those seen previously. The menu presents the three usual choices:

Degrees of freedom in [A],
Density function in [B], and
Cumulative Probability in [C].



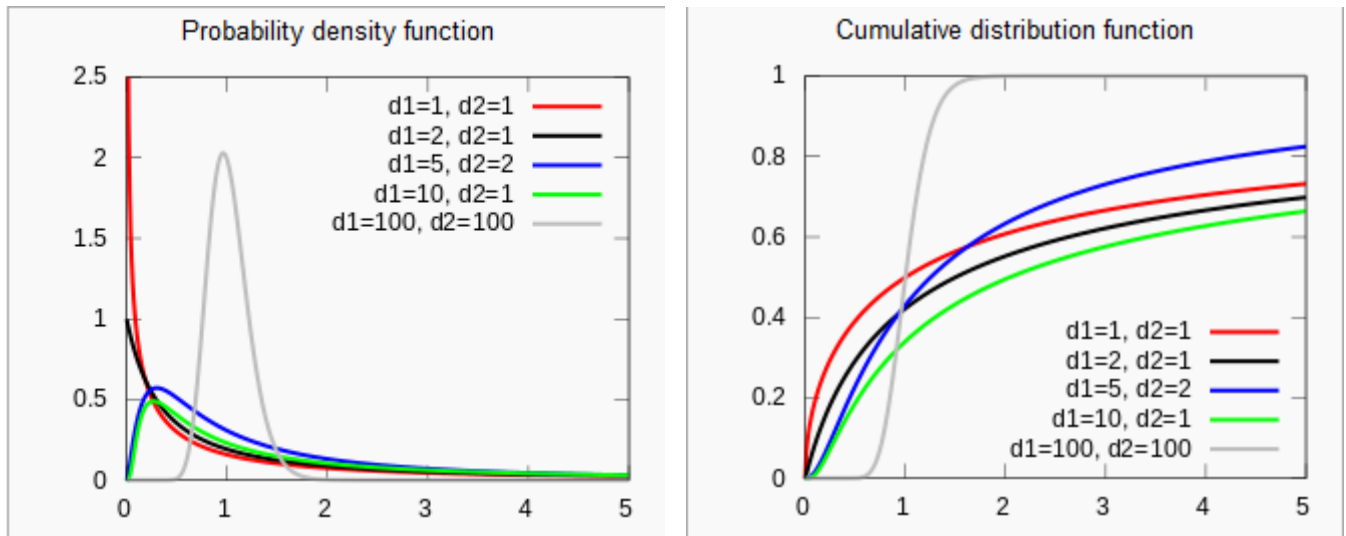
TSDF and **TSCP** are hybrid MCODE/Focal functions that don't make any use data registers, but they use the ALPHA registers and the stack. **TDIST** on the other hand is data-register hungry, you need to have SIZE 12 or higher for this routine to work correctly.

Examples: **TSDF**(4, 1) = 0.214662526; and **TSCP**(4, 1) = 0.626099034

Extended Statistics Module

Snedecor's F-Distribution { **F**SDF , **F**SCP , **F**DST }

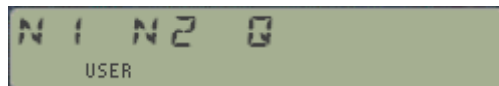
The F-distribution, also known as Snedecor's F distribution or the Fisher–Snedecor distribution (after Ronald Fisher and George W. Snedecor) is, in probability theory and statistics, a continuous probability distribution. If a random variable X has an F-distribution with parameters n_1 and n_2 , we write $X \sim F(n_1, n_2)$. The parameters n_1 and n_2 are positive integers, but the distribution is well-defined for positive real values of these parameters.



There are three functions available in the XMSTAT module dealing with this distribution.

- This first two are independent functions to calculate the density function (**F**SDF) and the TAILING probability (**F**SCP) – between $[x, \infty[$ - independently, taking the input parameters from Z, Y (degrees of freedom) and X (argument). The result is placed in X but no LastX is done.
- The third one (**F**DST) is also a driver program in the same guise as those seen previously. The menu presents the three usual choices:

Parameter n_1 in **[A]**,
Parameter n_2 in **[B]**, and
Cumulative Probability in **[C]**.



FSDF and **F**SCP are hybrid MCODE/Focal functions that don't make any use data registers, but they use the ALPHA registers and the stack. For **F**DST however, you need to have SIZE 08 or higher for this routine to work correctly.

Examples: **F**SDF(7, 6, 4.3) = 0.022012383, and **F**SCP(7, 6, 4.3) = 0.047640800

Note: The implementation for the Cumulative Probability does not use the incomplete beta function but trigonometric approximations. It is therefore faster but not so accurate for small values of the argument x .

Binomial Distribution { **BNP** , **BNP+** , **BIN** }

In probability theory and statistics, the binomial distribution with parameters n and p is the discrete probability distribution of the number of successes in a sequence of n independent yes/no experiments, *each of which yields success with probability p* . A success/failure experiment is also called a Bernoulli experiment or Bernoulli trial; when $n = 1$, the binomial distribution is a Bernoulli distribution. The binomial distribution is the basis for the popular binomial test of statistical significance.

Probability Single-Event function

In general, if the random variable X follows the binomial distribution with parameters $n \in \mathbb{N}$ and $p \in [0,1]$, we write $X \sim B(n, p)$. The probability of getting exactly k successes in n trials is given by the probability mass function:

$$f(k; n, p) = \Pr(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

Cumulative Probability function.

The cumulative Probability function can be expressed as:

$$F(k; n, p) = \Pr(X \leq k) = \sum_{i=0}^{\lfloor k \rfloor} \binom{n}{i} p^i (1 - p)^{n-i}$$

where $\lfloor k \rfloor$ is the "floor" under k , i.e. the greatest integer less than or equal to k .

There are three functions available in the XMSTAT module dealing with this distribution.

- This first two are independent functions to calculate the density function (**BNP**) for a single event and the cumulative probability (**BNP+**) for all previous events – in both cases taking the input parameters from Z , Y (n and p) and X (single or multiple event). The result is placed in X but no LastX is done. These routines use data registers R00 to R03.
- The third one (**BIN**) is also a driver program but contrary to the previous ones, this will prompt sequentially for the parameters and the number of events – to calculate the same value as BNP (i.e. non-cumulative case).

Note that to obtain a cumulative probability we could simply add all the single-event probabilities for those arguments less than or equal to argument. The implementation of **BNP+** however uses a different approach, which is faster than the simple addition of individual factors.

Examples: calculate $\text{BNP}(100, 1/6, 15)$ as single-event and also cumulative:

$\text{BNP}(100, 1/6, 15) = 0.100236634$; and $\text{BNP+}(100, 1/6, 15) = 0.387657550$

Extended Statistics Module

Appendix. Some FOCAL Program Listings

1. Binary Distribution, by JM Baillard.

01	LBL "BNP+"	09	STO 02	17	LASTX	25	*	
02	STO 00	10	Y^X	18	LBL 01	26	RCL 03	33 GTO
03	SIGN	11	SIGN	19	LASTX	27	1	01
04	X<>Y	12	RCL	20	RCL	28	+	34 SIGN
05	STO 01	00		01		29	STO	35 LBL 02
06	-	13	X=0?	21	*	03		36 LASTX
07	ST/ 01	14	GTO 02	22	RCL 02	30	/	37 END
08	X<>Y	15	CLX	23	RCL 03	31	+	
		16	STO 03	24	-	32	DSE 00	

2. Chi-Squared Distribution, by JM Baillard

01	LBL "CH2"	11	/	21	STO M	31	2	
02	STO N	12	E^X	22	1	32	+	41 *
03	X<>Y	13	ST* Y	23	RCL	33	STO T	42 RCL N
04	STO O	14	X<> L	0		34	/	43 SIGN
05	2	15	RCL	24	1	35	STO T	44 X<>
06	ST+ Y	0		25	ENTER^	36	X<>Y	0
07	/	16	2	26	LBL 01	37	ST+ Y	45 X<>Y
08	GAMMA	17	/	27	X<> T	38	X#Y?	46 CLA
09	RCL N	18	Y^X	28	RCL N	39	GTO	47 END
10	2	19	X<>Y	29	*	01		
		20	/	30	R^	40	RCL M	

3. Normal Distribution; by Michael Faulhaber

01*	LBL "XXP"	25	ENTER^	49	FS?C 18	73*	LBL 09	97	X#0?
02	X<0?	26	ENTER^	50	CHS	74	RCLFLAG	98	GTO 01
03	LOG	27	STO O	51	"GI"	75	STO 10	99	RCL 10
04	1	28	1308 E-6	52	ASTO 06	76	FIX 7	100	STOFLAG
05	+	29	*	53	E-3	77	LASTX	101	RCL N
06	2	30	,189269	54	X<>Y	78	ENTER^	102	2
07	/	31	+	55	SF 10	79	X^2	103	/
08*	LBL "XP"	32	*	56	XROM "SV"	80	STO N	104	CHS
09	X<=0?	33	1,432788	57	CF 10	81	SIGN	105	E^X
10	LOG	34	+	58	RTN	82	STO O	106	PI
11	STO 11	35	*	59*	LBL "PXX"	83	X<>Y	107	ST+ X
12	,5	36	1	60	ABS	84	STO M	108	SQRT
13	X>Y?	37	+	61	SF 18	85	ABS	109	/
14	SF 18	38	X<> O	62*	LBL "PX"	86*	LBL 01	110	RCL M
15	SIGN	39	,010328	63	ABS	87	LASTX	111	*
16	X<>Y	40	*	64	6	88	RCL N	112	ENTER^
17	X>Y?	41	,802853	65	X>Y?	89	RCL O	113	FC?C 18
18	SIGN	42	+	66	GTO 09	90	2	114	,5
19	FC? 18	43	*	67	LASTX	91	+	115	+
20	-	44	2,515517	68	SIGN	92	STO O	116	FC? 10
21	1/X	45	+	69	X<0?	93	/	117	RTN
22	X^2	46	RCL O	70	CLX	94	*	118	RCL 11
23	LN	47	/	71	CF 18	95	ST+ M	119	-
24	SQRT	48	-	72	RTN	96	RND	120	END

XROM	Name	Description	Input	Output	Type	Source	Author
06,00	-XM STAT 1E	<i>Section Header</i>	n/a	n/a	MCODE	This Project	
06,01	%T	Total percentage	y,x in Y,X	100 * x/y	MCODE	SandMath Project	Ángel Martin
06,02	Σ0	Sum of mantissa digits	value in X		MCODE	PK Collection	Ángel Martin
06,03	Σ1	Sum of N integers	N in X		MCODE	PK Collection	Poul Kaarup
06,04	Σ1/N	Harmonic Number	n in X	result in X	MCODE	SandMath Project	Ángel Martin
06,05	Σ2	Sum of squares of N integers	N in X	Σn; n=1,2..N	MCODE	PK Collection	Poul Kaarup
06,06	Σ3	Sum of cubes of N integers	N in X	Σn^2; n=1,2..N	MCODE	PK Collection	Poul Kaarup
06,07	ΣX^N	Geometric Sums	N in Y, X in X	Σn^3; n=1,2.. N	MCODE	SandMath Project	Ángel Martin
06,08	ΣRG?	Stat Reg Location	none	Σ(k^N); k=1,2...N	MCODE	MCODE for Beginners	Ken Emery
06,09	AGM	Arithmetic-Geometric Mean	x,y in Stack	Stat Registers Location	MCODE	SandMath Project	Ángel Martin
06,10	AMEAN	Arithmetic Mean	bbb.eee in X	result in X	MCODE	This Project	Ángel Martin
06,11	CNK	Combinations	arguments in X,Y	Result in X	MCODE	SandMath Project	Ángel Martin
06,12	CORR	LR Correlation	Σ+ Data Set	Correlation coefficient	MCODE	hp41 Programs	JM Baillard
06,13	COV	LR Covariance	Σ+ Data Set	Sample Covariance in X	MCODE	hp41 Programs	JM Baillard
06,14	EVEN?	Tests for even value	value in X	Yes/No, skips if false	MCODE	SandMath Project	Ángel Martin
06,15	GCD	Greater Common Divisor	arguments in X,Y	Result in X	MCODE	SandMath Project	Ángel Martin
06,16	GHM	Geometric-Harmonic Mean	x,y in Stack	result in X	MCODE	GJM ROM	Greg McClure
06,17	GMEAN	Geometric Mean	bbb.eee in X	result in X	MCODE	This Project	Ángel Martin
06,18	HMEAN	Harmonic Mean	bbb.eee in X	result in X	MCODE	This Project	Ángel Martin
06,19	LCM	Least Common Multiple	arguments in X,Y	Result in X	MCODE	SandMath Project	Ángel Martin
06,20	L1	Shows line equation	x,y in Stack	result in Alpha	MCODE	This Project	Ángel Martin
06,21	LR	Linear Regression	Σ+ Data Set	a in Y, b in X	MCODE	hp41 Programs	JM Baillard
06,22	LRX	LR X-Value	Σ+ Data Set, intercept	Y-value in X	MCODE	This Project	Ángel Martin
06,23	LR Y	LR Y-Value	Σ+ Data Set, abscissa	X-value in X	MCODE	hp41 Programs	Martin - Baillard
06,24	ODD?	Tests for odd value	value in X	Yes/No, skips if false	MCODE	SandMath Project	Ángel Martin
06,25	PMEAN	Generalized Power Mean	p in Y; bbb.eee in X	result in X	MCODE	This Project	Ángel Martin
06,26	PNK	Permutations	arguments in X,Y	Result in X	MCODE	SandMath Project	Ángel Martin
06,27	STLINE	Straight Line equation	(x1,y1) & (x2,y2) in stack	a,b, in stack, shows eqt.	MCODE	SandMath Project	Ángel Martin
06,28	RCLΣ	Stat Reg to Stack	none	Stat Regs values	MCODE	hp41 Programs	JM Baillard
06,29	ST<->Σ	Swap Stack & Stat Reg	none	Stack and Stat swapped	MCODE	NFC ROM	Nelson F. Crowle
06,30	-DISTRIBTN	<i>Section Header</i>	n/a	n/a	MCODE	This Project	
06,31	BIN	Binomial Distribution	Prompted by program	P(X<k; n,p)	FOCAL	Program Collection	Ángel Martin
06,32	"BNP"	Binomial P(n,p,x)=k	n,x in Y, X	P(X=k; n,p)	FOCAL	hp41 Programs	JM Baillard
06,33	"BNP+"	Binomial P(n,p,x)<k	n,k,x in stack	P(X<k; n,p)	FOCAL	hp41 Programs	JM Baillard
06,34	C2CP	Chi-square Probability (n, x)	Arguments in X,Y	Result in X	FOCAL	hp41 Programs	JM Baillard
06,35	C2DF	Chi-square Density (n, x)	n,x in Y, X	Result in X	FOCAL	This Project	Ángel Martin
06,36	CH2	Chi^2 Distribution	prompts for v value	f(v,x), and P(v,x)	FOCAL	STAT Pac	HP Co.

	"FDST"	Snedecor's F-Distribution	menu: : "N1 N2 Q"	Prompts result	FOCAL	STAT Pac	HP Co.
06,37	FDST	Snedecor's F Density Function	Z: N1, Y: N2, X: Point	Result in X	FOCAL	CTE Collection	Eugenio Úbeda
06,38	FSCP	Snedecor's F Probability Fnc.	n,x in Y, X	Result in X	FOCAL	CTE Collection	Eugenio Úbeda
06,39	NRDF	Normal Density (m,s)	Z: Mean, Y: Sdev, X: Point	Result in X	MCODE	This Project	Ángel Martín
06,40	NRCP	Normal Probability (m,s)	Z: Mean, Y: Sdev, X: Point	Result in X	HYBRID	SandMath Project	Ángel Martín
06,41	POIS	Poisson Distribution	prompts for values	Result in X	FOCAL	This Project	Ángel Martín
06,42	PSDF	Poisson Density Function	Y: Mean, X: #Events	Result in X	MCODE	This Project	Ángel Martín
06,43	"PX"	Standard Normal Probability	Argument in X	Result in X	FOCAL	Author's Collection	Michael Faulhaber
06,44	"PXX"	Probability between -x and x	Argument in X	Result in X	FOCAL	Author's Collection	Michael Faulhaber
06,45	QNTL	Quantile - inverse of NRCP	Argument in X	Result in X	FOCAL	SandMath Project	Ángel Martín
06,46	"TDST"	Students' T-Distribution	Prompts choices	results in X	FOCAL	STAT Pac	HP Co.
06,47	TSDF	Student's T Density Function	N in Y, Point in X	result in X	FOCAL	CTE Collection	Eugenio Úbeda
06,48	"TSCP"	Student's T Probability Func.	n,x in Y, X	result in X	FOCAL	CTE Collection	Eugenio Úbeda
06,49	"XP"	Inverse of PX	Argument in X	Result in X	FOCAL	Author's Collection	Michael Faulhaber
06,50	"XXP"	Inverse of PXX	Argument in X	Result in X	FOCAL	Author's Collection	Michael Faulhaber
06,51	-PRIMALITY	Calculates Y^X	x,y in Stack	y^x, (0^0=1)	MCODE	SandMath Project	Ángel Martín
06,52	1/GM	Inverse Gamma Function	Argument in X	Result in X	MCODE	SandMath Project	JM Baillard
06,53	BETA	Beta Function	Arguments in X,Y	Result in X	HYBRID	hp41 Programs	JM Baillard
06,54	D%	Difference Percent	x,y in Stack	(y-x)/x % in X and Alpha	MCODE	SandMath Project	Ángel Martín
06,55	ERF	Error Function	x in X	erf(x) in X	MCODE	iERF Module	Martin - Baillard
06,56	GMF	Gamma Function	Argument in X	Result in X	MCODE	hp41 Programs	JM Baillard
06,57	NXTPRM	Next Prime	value in X	result in X	MCODE	PK Collection	Poul Kaarup
06,58	PFACT	Prime Factor	Argument in X	prime divisor in X	MCODE	This Project	Peter Platzer
06,59	PRIME?	Prime Test	Argument in X	Yes/No Test.	MCODE	PK Collection	Poul Kaarup
06,60	PTWIN	Prime Twins	Argument in X	next twin prime in X	MCODE	This Project	Peter Platzer
06,61	RAND	Random Number (w/ Timer)	none	random number in X	MCODE	hp41 Programs	JM Baillard
06,62	"SLV"	Solve Subroutine	x1, x2 in X,Y, fname in Alpha	root value	FOCAL	PPC ROM	PPC Members