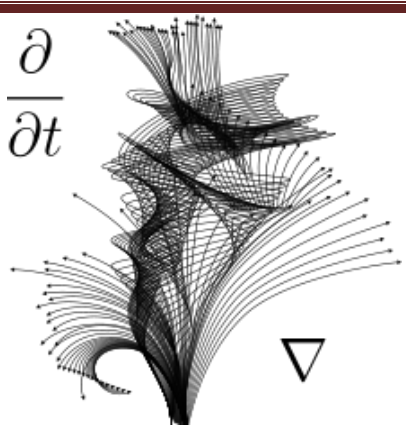
HP-41 Module: Areas, Series & Sums. Fractional Integro-Differentiation



Overview

This module includes a selection of functions and FOCAL routines mainly focused on Series and Sums field and other related subjects. For the most part the routines are taken from Jean-Marc Baillard extensive web site, although some others are takes from Poul Kaarup's collection as well. A few are already available in the SandMath Module –even if this version is a more portable implementation that suits itself better for Clonix/NoVRAM owners.

The initial section of the module covers the simple sums of integers and integer powers. This is followed by simple explicit sums for single, double, triple and multiple series; recursive term sums and Euler transformations. Examples are also provided in the FAT for quick familiarization. The second section includes a set of MCODE functions and FOCAL routines dealing with area calculations for several geometric figures, such as circles and triangles, as well as areas and diagonal lengths of cyclic and regular polygons.

The last section takes a relative large part of the FAT and deals with Fractional Integro-differentiation of a few Elemental and Special functions. This technique is based on the Hyper-Geometric function and consolidates an impressive set of capabilities under the same approach.Without further ado, see below the list of functions included in the module:

XROM	Function	Description	Input	Author
18,00	**-SERIES 1A**	Rounded Comparison	x,y in X,Y	*Ángel Martin*
18,01	**CHSYX**	Sign Change of Y by X: Y*(-1)^X	Value in Y, n in X	*Ángel Martin*
18,02	**NCR**	Combinations of N in sets of R	N in Y, R in X	*Ángel Martin*
18,03	**"#"**	Auxiliary for SOLVE	Under program control	*Ángel Martin*
18,04	**Σ0**	Sum of mantissa digits	argument, X	*Ángel Martin*
18,05	**Σ1**	Sum of first N integers	argument, X	*Poul Kaarup*
18,06	**Σ1/N**	Harmonic Numbers	argument, X	*Ángel Martin*
18,07	**Σ2**	Sum of Squares of Numbers	argument, X	*Poul Kaarup*
18,08	**Σ3**	Sum of Cubes of Numbers	argument, X	*Poul Kaarup*
18,09	**ΣN^X**	Generalized Faulhaber's Sum	N in Y, exponent In X	*Ángel Martin*
18,10	**"ΣUM"**	Single Series Sum (Explicit)	arguments in X, Y, ALPHA	*JM Baillard*
18,11	**"ΣΣUM"**	Double Series Sum (Explicit)	argument in X, ALPHA	*JM Baillard*
18,12	**"ΣΣΣUM"**	Triple Series Sum (Explicit)	arguments in Y,X, ALPHA	*JM Baillard*
18,13	**"ΣUME**	Euler Transformation	argument in X, Y, Z, ALPHA	*JM Baillard*
180,14	**"ΣUMR"**	Single Series Sum (Iterative)	arguments in X, Y, ALPHA	*JM Baillard*
18,15	**"NΣUM0"**	Multiple Series Sum (Explicit)	bbb.eee in X, data in registers	*JM Baillard*
18,16	**"NT"**	Example for NΣUM0	n/a	*Martin-Baillard*
18,17	**"T"**	Example for ΣUM	n/a	*Martin-Baillard*
18,18	**"TE"**	Example for ΣUME	n/a	*Martin-Baillard*

18,19	**"TR"**	Example for ΣUMR	n/a	*Martin-Baillard*
18,20	**"TT"**	Example for ΣΣUM	n/a	*Martin-Baillard*
18,21	**"TTT"**	Example for ΣΣΣUM	n/a	*Martin-Baillard*
18,22	**-AREAS_1B**	ALPHA Integer part	Argument in X	*Fritz Ferwerda*
18.23	**BRHM**	Brhamagupta formula	4 Sides in stack	*Ángel Martin*
18.24	**CIRCLE**	Circle through three points	Coordinates in R01-R06	*Ángel Martin*
18.25	**DIAG**	Diagonal formula	Used in CPLD	*JM Baillard*
18.26	**HERON**	Heron formula	Triangle sides in Z, Y, Z	*Ángel Martin*
18.27	**RPG1**	Regular Polygon from sides	# sides in Y, length in X	*Poul Kaarup*
18.28	**RPG2**	Regular polygon from circle	# sides in Y, radius in X	*Poul Kaarup*
18.29	**"3PNTS"**	Driver for CIRCLE/Areas	Prompts for coordinates	*Ángel Martin*
18.30	**"CCPA"**	Complex Cyclic Polygon Area	Parameters in stack and R00	*JM Baillard*
18.31	**"CCPA+"**	Driver for CCPA	Uses Newton Method	*Ángel Martin*
18.32	**"CPLA"**	Complex Cyclic Polygon Area	With all Sides known	*JM Baillard*
18.33	**"CPLA+"**	Driver for CPLA	Uses SOLVE	*Ángel Martin*
18.34	**"CPLD"**	Complex Cyclic Polyg. Diagonals	bbb.eee in X, data in Registers	*JM Baillard*
18.35	**"CPLD+**	Driver for CPLD	Prompts for data entry	*Ángel Martin*
18.36	**"STLA"**	Star Polygon Area	Parameters in Stack	*JM Baillard*
18.37	**"STLA+"**	Driver for STLA	Prompts for data entry	*Ángel Martin*
18.38	**-ITG/DIFF**	Generalized Hypergeometric	Data in Registers and Stack	*JM Baillard*
18.39	**1/GM**	Reciprocal of Gamma	argument in X	*JM Baillard*
18.40	**PSI**	Digamma Function	argument n X	*JM Baillard*
18.41	**"DAIRY"**	Airy Function Intg-diffm	μ in Y, x in X	*JM Baillard*
18.42	**"DEI"**	Exponential Intg. Intg-Diff.	μ in Y, x in X	*JM Baillard*
18.43	**"DERF"**	Error Function Intg.-Diff	μ in Y, x in X	*JM Baillard*
18.44	**"DCHI"**	Hyp Cos Integral Intg-Diff.	μ in Y, x in X	*JM Baillard*
18.45	**"DCH"**	Hyp. Cosine Intg-Diff.	μ in Y, x in X	*JM Baillard*
18.46	**"DCI"**	Cosine Integral Intg-Diff.	μ in Y, x in X	*JM Baillard*
18.47	**"DCOS"**	Cosine Intg-Diff	μ in Y, x in X	*JM Baillard*
18.48	**"DCX"**	Fresnel Cosine Integral Intg-Diff.	μ in Y, x in X	*JM Baillard*
18.49	**"DEXP"**	Exponential Intg-Diff.	μ in Y, x in X	*JM Baillard*
18.50	**"DHMT"**	Hermite Function Intg-Diff.	μ in Z, n in Y, x in X	*JM Baillard*
18.51	**"DINX"**	Bessel I function Intg-Diff.	μ in Z, n in Y, x in X	*JM Baillard*
18.51	**"DJNX"**	Bessel J function Intg-Diff.	μ in Z, n in Y, x in X	*JM Baillard*
18.53	**"DKNX"**	Mod. Bessel K function Intg-Diff.	μ in Z, n in Y, x in X	*JM Baillard*
18.54	**"DKUM"**	Kummer function intg-Diff.	μ in Y, x in X, a in R00, b in R01	*JM Baillard*
18.55	**"DLANX"**	Lagrange Function Intg-Diff.	μ in T, a in Z, n in Y, x in X	*JM Baillard*
18.56	**"DLN"**	Natural Log Intg-Fiff.	μ in Y, x in X	*JM Baillard*
18.57	**"DSB1"**	Spherical Bessel Funct. 1st. Kind	μ in Z, n in Y, x in X	*JM Baillard*
18.58	**"DSHI"**	Hyp. Sine Integral inth-Diff.	μ in Y, x in X	*JM Baillard*
18.59	**"DSH"**	Hyperbolic Sine Intg.Diff.	μ in Y, x in X	*JM Baillard*
18.60	**"DSI"**	Sine integral Intg-Diff.	μ in Y, x in X	*JM Baillard*
18.61	**"DSIN"**	Sine Intg-Diff	μ in Y, x in X	*JM Baillard*
18.62	**"DSX"**	Fresnel Sine Integral Intg-Diff.	μ in Y, x in X	*JM Baillard*
18.63	**"DYNX"**	Mod. Bessel Y function Intg-Diff.	μ in Z, n in Y, x in X	*JM Baillard*

1 –Sums and Series

The first section includes several functions to calculate sums of integer powers, as well as simple methods to sum series given their general term in explicit or recurrent form.

- **Σ0** is a small divertimento useful in pseudo-random numbers generation. It simply returns the sum of the mantissa digits of the argument – at light-blazing speed using just a few MCODE instructions. More about random numbers will be covered in the Probability/Stats section later on.

  Example: calculate the sum of all digits of the HP-41's rendition of pi:

  PI, XEQ "**Σ0**" => 40.000000000

- **Σ1/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 will be used in the calculation of the Kelvin functions 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, XEQ "Σ1/N" => 2.283333333
  25, XEQ "Σ1/N" => 3.815958178

- **Σ1, Σ2, Σ3** are implemented to calculate the sum of integer powers directly based on the corresponding formulas. The number of terms to sum is expected to be in the X- register. Functions calculate the linear sum using the triangular formula; the sum of squares using the pyramidal formulas; and the sum of cubes also using the pyramidal formulas.

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

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

  Example: Calculate the sum of the first 10 natural numbers and their squares and cubes:

  10, **Σ1** quickly returns: 55.00000000
  LASTX, **Σ2**" => 385.0000000
  LASTX, **Σ3** => 3,025.000000

- **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 (which are used in the functions described above) , 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 **Σ1/N**, albeit slower due to the additional complexity of the definition 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, XEQ "ΣN^X" => 55.00000000
10, ENTER^, 2, XEQ "ΣN^X" => 385.0000000

And using the convention B(1) = 0.5 the formula is:

$$S_m(n) = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k\, n^{m+1-k},$$

Which could be programmed using a few of the SandMath functions, albeit it would be considerably slower due to the impact of the Zeta algorithms (part of Bernoulli's) – kicking in for n>4.

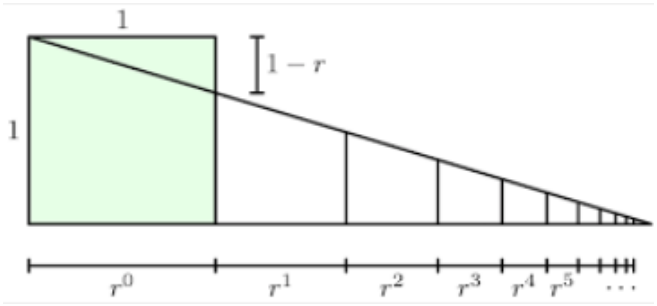
- **CHSYX** is related to the same subject, and in general relevant to the summation of alternating series – It can be regarded as an extension of **CHS** but dependent of the number in X. Its expression is:

  CHS(y,x)= y*(-1)^x, and thus changing the sign of Y when the number in X is odd.

- **NPR** calculates Permutations, defined as the number of possible different arrangements of N different items taken in quantities of R 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:

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

This last two functions will be used as subroutines in the series sums programs described next.



Sums of Series using their General Terms

The following programs allow you to obtain the value of simple, double and triple series for which the general term is known, either as explicit expression or as a recurrent form.

Simple Series.

ΣUM calculates a single series sum, that is:

$$S = (u_k + u_{k+1} + u_{k+2} + u_{k+3} + \ldots\ldots\ldots) = \Sigma\, u_n \quad \text{for } n >= k$$

defined by the general term $u_n = f(n)$ where f is a known (i,e, explicit) function.

A program which computes Un = f(n) is required as a subroutine. It is done assuming n is in X-register upon entry. The module includes the routine "**T**" as example for this case, with Un=1/n!

ALPHA, "TR", 1, ALPHA, XEQ "ΣUM" -> X = 1.718281830 = R01 (in 9 seconds)

ΣUMR calculates the same sum when Un is given as a recurrence expression with an initial known value, that is: u_k is given and $u_{n+1} = f(u_n;n)$ where f is a known function.

a program which computes un+1 = f(Un; n) is required as a subroutine. It is done assuming Un is in X-register and n is in Y-register upon entry. The module includes the routine "**TR**" as example for this case, with $u_1 = 1$ and $u_{n+1} = u_n/(n+1)$

ALPHA, "TR", ALPHA, 1, ENTER^, XEQ "ΣUMR" -> X = 1.718281830 = R01 (in 7 seconds)

Note that on both cases the initial index can also be zero, assuming that's compatible with the definition of Un, which adds more flexibility to the routine. In both cases the function needs to be programmed under a global label, and *its name is expected to be in the ALPHA register when the routines are called.*

STACK	INPUTS	OUTPUTS		STACK	INPUTS	OUTPUTS
Y	/	ΣUM		Y	k	ΣUMR
X	k	ΣUM		X	u_k	ΣUMR
ALPHA	F.NAME	F.NAME		ALPHA	F.NAME	F.NAME

Euler Transformation.

ΣUME calculates the same sum making use of the *Euler Transformation* to accelerate the convergence of alternating series: $S = u_0 - u_1 + u_2 - u_3 + \ldots\ldots + (-1)^n\ u_n + \ldots..$

The sum is re-written in function of the binomial coefficients, $C_n^p = n! / (p! (n-p)!)$ as follows:

$$S = a_0/2 + (C_1^1\, a_0 - C_1^0\, a_1)/2^2 + (C_2^2\, a_0 - C_2^1\, a_1 + C_2^0\, a_2)/2^3 + (C_3^3\, a_0 - C_3^2\, a_1 + C_3^1\, a_2 - C_3^0\, a_3)/2^4 + \ldots\ldots$$

This may produce superb acceleration but it can also fail.

A program which computes Un = |f(n)| is required as a subroutine, but *without the alternating sign.* It is done assuming n is in X-register upon entry. The module includes the routine "**TE**" as example for this case, with $f(n) = (n+1)^{-1/2}$

ALPHA, "TE", ALPHA, XEQ "ΣUME" -> X = 0.6048986431 = R01

For this particular example the error is -3 10-10 (!), and only 28 terms are calculated (taking about 6.5 minutes to converge). Without an acceleration method, more than 1,018 terms would be necessary to achieve the same accuracy... which execution time would be much greater than the age of the Universe.

STACK	INPUTS	OUTPUTS
X	/	ΣUME
ALPHA	F.NAME	F.NAME

Double and Triple Series.

ΣΣUM calculates a double series sum, that is: $S = \Sigma\Sigma (u_{n;m})$; for $n >= n_0$; $m >= m_0$

As before, the general term needs to be programmed under a separate subroutine using a global label. This assumes that "n" is in the X-register and "m" in the Y-register upon entry. The two initial indices are expected to be in the stack and the label name must be in the ALPHA register as well.

STACK	INPUTS	OUTPUTS
Y	m_0	ΣΣ
X	n_0	ΣΣ
ALPHA	F.NAME	F.NAME

The program uses data registers R00 to R05, which therefore should not be used in the definition of the general term. The module includes the routine "**TT**" as example for this case, with the expression: $f(n;m) = 1 / (n^n m!)$

ALPHA, "TT", ALPHA, 1, ENTER^, XEQ "ΣΣUM" -> X = 2.218793264 = R03 (in 1 min 18s)

ΣΣΣUM calculates a triple series sum, that is: $S = \Sigma\Sigma\Sigma (u_{n;m;p})$; for $n >= n_0$; $m >= m_0$; $p >= p_0$

As before, the general term needs to be programmed under a separate subroutine using a global label. This assumes that "n" is in the X-register "m" in the Y-register, and p is in the Z-register upon entry. The three initial indices are expected to be in the stack and the label name must be in the ALPHA register as well.

STACK	INPUTS	OUTPUTS
Z	p_0	/
Y	m_0	ΣΣΣ
X	n_0	ΣΣΣ
ALPHA	F.NAME	F.NAME

The program uses data registers R00 to R07, which therefore should not be used in the definition of the general term. The module includes the routine "**TTT**" as example for this case, with the expression: $f(n;m;p) = 1 / (n^n\ m!\ (p!)^2)$

ALPHA, "TTT", ALPHA, 1, ENTER^, ENTER^, XEQ "ΣΣΣUM" -> X = 2.839135243 = R04 (6 min 5s)

With an error ε =-7 E-9

Multiple Series

NΣUM0 calculates a multiple series sum, with k internal summations that all start in zero - that is:

$$S = \Sigma\Sigma...\Sigma\ [\ U(n_1;n_2;\ .....\ ;n_k)\ ]\ \ with\ \ n_1>= 0\ ;\ \ n_2>= 0\ ;\ .....\ ;\ \ n_k>= 0$$

Obviously now the number of indices will be in the X register, and there are no initial indices – which are assumed to be zero. This may need you to re-write the expression of the general term to make it compatible with this condition.

This is the most limiting requirement for this program, which is not suitable for cases that have mutual dependencies between the initial indexes.

Here too, the general term needs to be programmed under a separate subroutine using a global label, which needs to be entered in ALPHA. This assumes on entry that "n1" is in the R01 register, "n2" in the R02 register, "n3" in R03, and successively so until completing the number of variables.

STACK	INPUTS	OUTPUTS
X	k	NΣ0
ALPHA	F.NAME	/

Only the synthetic registers {M,N,O} are used by the program. The module includes the routine "**NT**" as example for this case, with the expression used in the triple aeample : $f(n;m;p) = 1 / (n^n\ m!\ (p!)^2)$.

We therefore need to change it to start at the zero indexes for the three variables, i.e. must make a change of arguments to reduce to the standard: n >= 0 ; m >= 0 ; p >= 0 by replacing n with (n+1) ; m with (m+1) ; p with (p+1): $f(n;m;p) = 1 / \{(n+1)^{(n+1)}\ (m+1)!\ [(p+1)!]^2 \}$

ALPHA, "NT", ALPHA, 3, XEQ "NΣUM0" -> X = 2.839135243 = R04 (8 min 39 s)

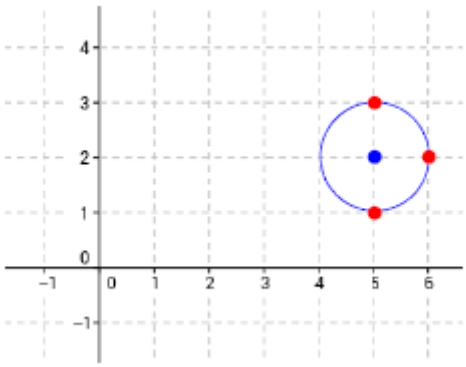
$$F(x,v) = \sum_{k=0}^{n} \sum_{m1=0}^{k} \sum_{m2=0}^{k} \alpha_{m1,m2,k}\, R^k y^{k-m1} (1-y)^{m1} z^{k-m2} (1-z)^{m2}$$

2 –Areas of Polygons

The second section of the module includes several MCODE functions for triangles, cyclic quadrilaterals and even non-regular polygons.

- **CIRCL** calculates the radius of a circle passing thru three data points, using the point x,y coordinates. The values are expected to be stored in R01-R07. Besides that, it'll also return in the Y-register the area of the circumscribed triangle defined by the three points.

  Example: Calculate the radius of the circle passing thru P(5,1), Q(6,2), and R(5,3)



The results are:

XEQ "CIRCL" => r=1,000000000,
X<>Y => A=1,000000000

The input sequence starts with the abscissa of P1 in R01.

Note that you can use the routines **IN** and **INPUT** available in the SandMath to populate the registers automatically.

- **HERON** calculates the area of a triangle knowing its three sides, using Heron's formula. Just enter the sides values in the stack, and execute the function. The result is stored in X, with the original side saved in LastX. The rest of the stack is unchanged.

  Let the triangle ABC with 3 known sides { a , b , c } and s = (a+b+c)/2 the semi-perimeter

  Heron's formula is: $\text{Area} = [\, s(s-a)(s-b)(s-c)\,]1/2$

  Example: a = 2, b = 3, c = 4
  Type: 2, ENTER^, 3, ENTER^, 4, XEQ "HERON" => Area = 2.904737510

  Note: the function **CIRCL** described above makes use of the HERON formula internally after it first calculates the triangle sides from the point coordinates.

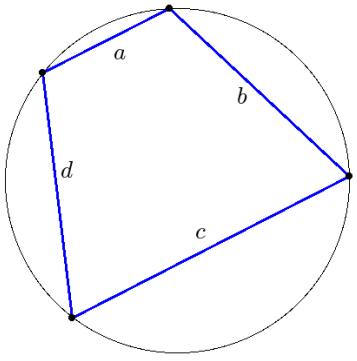
- **BRHM** is related to it, but the calculation for the area of the cyclic quadrilateral - using Brhamagupta's formula. Just enter the four values in the stack and execute the function. The result is stored in X, with the original side saved in LastX. The rest of the stack is unchanged.

Let a, b, c, and d be its sides lengths, and the semi-perimeter s = (a + b + c + d)/2 .The area A of the cyclic quadrilaterals:

$A = [(s-a).(s-b).(s-c).(s-d).]1/2$

Example: a = 4 , b = 5 , c = 6 , d = 7

Type: 4, ENTER^, 5, ENTER^, 6, ENTER^, 7,
XEQ "BRHM" => Area = 28.98275349



- **PG1** Calculates the area of a regular polygon when its side length is known. Input parameters are the number of sides in Y, and the side length in X. The result is left in the X register.

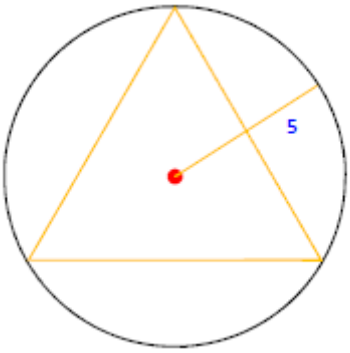
  Example: calculate the area of a triangle with side length a=5 m

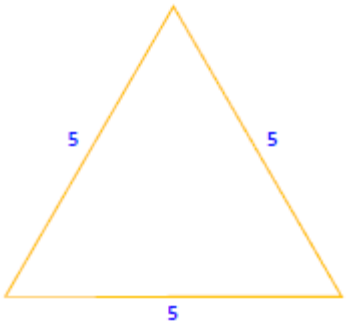
  3, ENTER^, 5, XEQ "PG1" -> 10.83 m^2

- **PG2** Performs the same calculation but using the radius of the circumscribed circle instead of the side length. Same order of parameters for input, with the number of sides in Y.

  Example: calculate the area of a triangle circumscribed in a circle with radius r=5 m

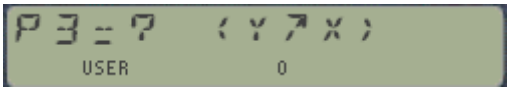
  3, ENTER^, 5, XEQ "PG2" -> 32.48 m^2





- Finally, the Routine "**3PNTS**" is a FOCAL driver for functions **CIRCLE**. You can use it to enter the coordinates of the three points into data registers R01-R06, presented as three screens with the prompts:

...


...


  Once this is accomplished the program offers you a choice for the value to calculate next, either the triangle area or the circle radius. You can Also press [E] to start over with a new set of three points.

  If used on the example listed above, it returns the following results:

and


Convex Cyclic Polygons.

And what about non-regular polygons, you may wonder? Well, those are the subject of the following set of FOCAL routines about to be described.

Programs **"CCPA"** and **"CPLA"** compute the area A and the circumradius R of a convex cyclic polygon assuming all the sides lengths are known. Moreover, we also assume that the *center* of the circumcircle is *inside* the polygon.

If$\{a_1 , a_2 , ... , a_n\}$ are the sides lengths and $\{ \mu_1 , \mu_2 , ... , \mu_n \}$are the corresponding central angles, we have to solve the system of (n+1) equations:

$$2.R \sin \mu_1/2 = a_1$$
$$2.R \sin \mu_2/2 = a_2$$
$$.....$$
$$2.R \sin \mu_n/2 = a_n$$
$$\mu_1 + \mu_2 + ... + \mu_n = 360°$$

Performing a few substitutions lead to an expression with the radius as single unknown, to be resolved iteratively using any root-finding method:

$$\text{asin}\ (a_1/2R) + \text{asin}\ (a_2/2R\ ) + ... + \text{asin}\ (a_n/2R) \ = \ 180°$$

After finding R, the Area is given by : $A = (R^2/2)\ (\text{Sin}\ a_1 + \text{Sin}\ a_2 + ... + \text{Sin}\ a_n)$

There are two versions included in the module – "**CPLA"** uses the SOLVE function in the Advantage and "**CCPA"** uses a built-in root finder based on Newton's method. Each one has advantages and shortcomings, as usual.

Drivers for Data Entry.

The routines expect the sides of the polynomial already stored in contiguous data registers, and the control word "bbb.eee" in the X register before you call the routine. For your convenience, a driver routine is also included that prompts for the side values and does the storing for you, Using "**CPLA+"** or "**CCPA+",** all you need to do is enter the values at each prompt, and once completed it'll direct the execution to the corresponding data engine downstream.

Example. Find the area of a convex cyclic polygon with sides: 4 , 5 , 6 , 7 , 8 , 9 , 10

XEQ "CPLA+"	"N=?"
7, R/S	"d1=?"
4, R/S	"d2=?"
5, R/S	"d3=?"
6, R/S	"d4=?"
7, R/S	"d5=?"
8, R/S	"d6=?"
9, R/S	"d7=?"
10, R/S	-> 174.6757940 the area, and
X<>Y	-> 8.143816980 the radius

Diagonal Lengths.

CPLD calculates the diagonal lengths of a convex cyclic polygon with its side lengths known. The method used involves solving a linear system of (n-3) equations with (n-3) unknowns, which is solved by successive approximations. The convergence is linear only – which results in relative longer execution times.

The iteration starts with all diagonals lengths = 0, which is very simplistic. The successive sums of the differences between 2 consecutive approximations (in absolute values) are displayed. They should tend to zero, however, the termination criterion may lead to an infinite loop. Since there are 319 registers at most, "CPLD" can find the diagonals lengths of a 159-gon – but the execution time will not be small without an emulator.

CPLD expects the number of sides in R00, and the sides of the polynomial already stored in contiguous data registers starting with R01 until Rn+1. Then you must provide the control words "bbb.eee" indicating the location of the data registers that store the side lengths.

STACK	INPUT	OUTPUTS
X	/	bbb.eee

For your convenience, a driver routine is also included that prompts for the side values and does the storing for you, Using "**CPLD+"** all you need to do is enter the values at each prompt, and once completed it'll direct the execution to CPLD downstream.

Example: Find the diagonals lengths of a convex cyclic hexagon with sides: 4 , 5 , 6 , 7 , 8 , 9

XEQ "CPLD+"	"N=?"
6, R/S	"a1=?"
4, R/S	"a2=?"
5, R/S	"a3=?"
6, R/S	"a4=?"
7, R/S	"a5=?"
8, R/S	"a6=?"
9, R/S	shows estimations… -> convergence
	d8=8.46278437
R/S	d9=12.12358502
R/S	d10=12.97690535

There are in fact n(n-3)/2 diagonals whose lengths may be obtained by "rotating" the sides lengths in registers R01 to R06 and we have similarly: n(n-3)/2 = 9 if n = 6

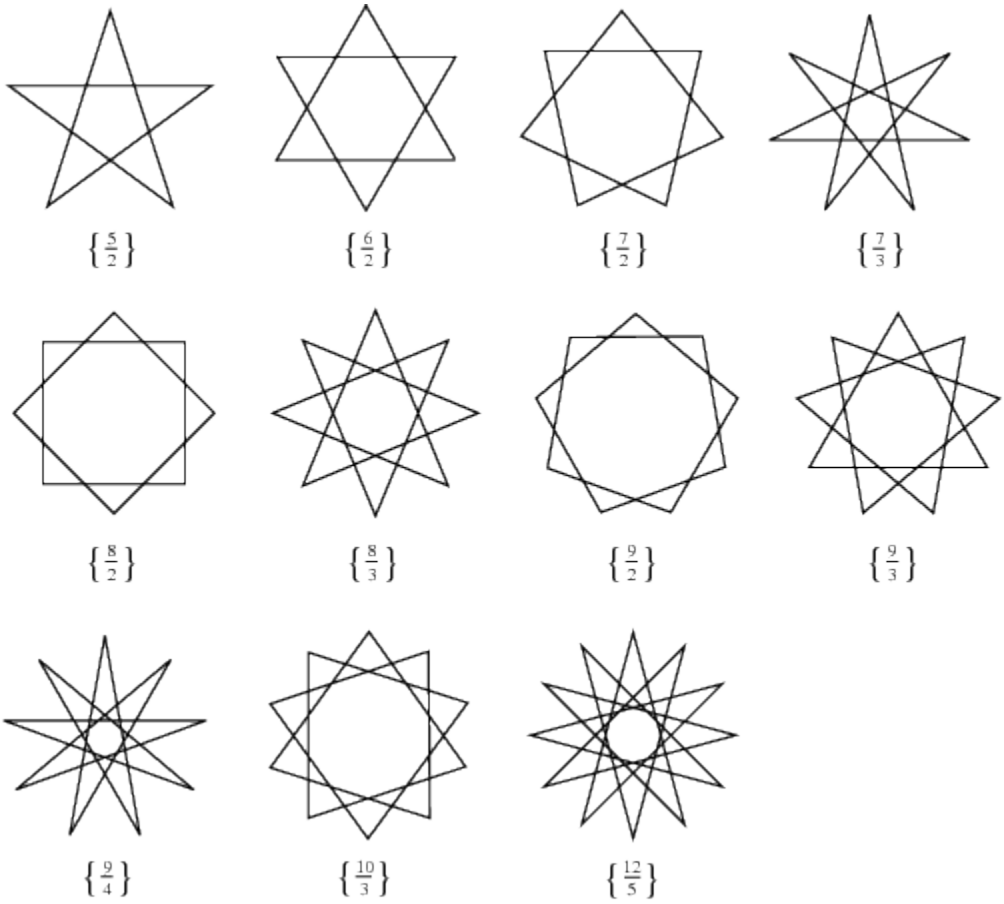
d_4 = 9.998827970	d_7 = 11.30861231
d_5 = 13.01214483	d_8 = 13.06010803
d_6 = 11.49035918	d_9 = 12.32872367

Finally, the MCODE function **DIAG** is used internally by **CPLD** to speed-up the calculations. It computes the following expression, assuming x , y , z , t are in registers X , Y , Z , T upon entry

$$\text{SQRT}\left[\{ x.y\,(z^2 + t^2) + z.t\,(x^2 + y^2) \} / (x.y + z.t) \right]$$

Regular Star Polygons

A star polygon {n/k}, with n,k positive integers, is a figure formed by connecting with straight lines every "k-th" point out of n regularly spaced points lying on a circumference. The number k is called the polygon density of the star polygon. Without loss of generality, take k<n/2. The star polygons were first systematically studied by Thomas Bradwardine.



If k=1, a regular polygon {n} is obtained. Special cases of {n/k} include {5/2} (the pentagram), {6/2} (the hexagram, or star of David), {8/2} (the star of Lakshmi), {8/3} (the octagram), {10/3} (the decagram), and {12/5} (the dodecagram).

"STLA" computes the area A, the perimeter P, the inradius r and the circumradius R of a regular star polygon { n / k } from its edge length a

Formulae:

$A = n R^2 \text{ Sin } (180°/n) \text{ Cos } (180° \text{ k/n}) / \text{ Cos } [180°(k-1)/n]$
$a = 2.R \text{ Sin } (180° \text{ k/n})$
$r = R \text{ Cos } (180° \text{ k/n})$
$P = 2.A / r$

The table on the left shows the input and output Required by the program – easy does it!

STACK	INPUTS	OUTPUTS
T	/	R
Z	a	r
Y	n	P
X	k < n/2	A

Examples:

- a = 1 , n = 5 , k = 2
  1 ENTER^, 5 ENTER^, 2 XEQ "STLA" -> A = 0.310270701
  RDN P = 3.819660113
  RDN r = 0.162459848
  RDN R = 0.525731112

- a = 1 , n = 10 , k = 3
  1 ENTER^, 10 ENTER^, 3, R/S -> A = 0.857567126
  RDN P = 4.721359547
  RDN r = 0.363271264
  RDN R = 0.618033989

- a = π , n = 41 , k = 13
  PI ENTER^, 41, ENTER^, 13, R/S -> A = 9.855571194
  RDN P = 19.37713086
  RDN r = 1.017237409
  RDN R = 1.871409374

This program works in all angular modes, however, DEG mode should be preferable.

If k = 1, we get the convex regular n-gon. For instance, with a = 1 , n = 5 , k = 1, "STLA" returns what corresponds to the regular pentagon.

A = 1.720477401
P = 5
r = 0.688190960
R = 0.850650808

Driver Program.

Here too you have a convenient driver program to guide you thru the data entry process: program "**STLA+**" will prompt for the input values and will present the results sequentially after the calculations are done.

, 

, 

, 

, etc…

3 –Fractional Integro-Differentiation

If μ is a real number (integer or fractional) and a function f is defined by a power series:

$$f(x) = \Sigma_{k=0,1,2,..}\{c_k\, x^k\}$$

then its fractionalintegro-differentiation may be computed by:

$$d^\mu f / dx^\mu = D^\mu f(x) = \Sigma_{k=0,1,2,....}\{ c_k\, [\, \Gamma(k+1) / \Gamma(k+1-\mu)\,]\, x^{k-\mu}\}$$

If the function may be expressed in terms of hypergeometric functions ${}_pF_q$, the following relation is very useful too:

$$D^\mu {}_pF_q\,(a_1, \ldots, a_p\,;\, b_1, .. b_q\,;\, x) = x^{-\mu}\Gamma(b_1)....\Gamma(b_q)\ {}_{p+1}\tilde{F}_{q+1}\,(1, a_1, ... , a_p\,;\, 1-\mu, b_1, ... b_q\,;\, x)$$

where Γ is Euler's Gamma function, and ${}_p\tilde{F}_q$ is the regularized generalized hypergeometric function. This function was already included in the SandMath module, and it is now copied here for convenience – tucked away under one of the section headers so that a dedicated FAT entry was not required.

As a trivial corollary, $D^0\ f(x) = f(x)$. Positive integer numbers (i.e. natural numbers) represent the successive derivatives of the function. Also if μ= -1, -2, -3, ... the results are the repeated integrals of the function f - usually those that vanish for x = 0.

Formulae and details.

Besides the special functions in the module there are several elementary functions included as well. Note that all program names begin with the 'D" letter, followed by the common acronym that designates the function's name.

As usual when a function is evaluated by a power series, the results are not very accurate for large arguments;they may even be meaningless ... unless all the terms have the same sign!

For data entry, μ is always to be entered first, then the order/index - if any - and finally, x in register X: (*)

Stack	Input#1	Input#2	Input#3	Result
T	/	/	μ	/
Z	/	μ	a	/
Y	μ	n	n	/
X	x	x	x	$(D^\mu f)\,(x)$

(*) as single exception, for Kummer's functions R01 and R02 must also be initialized first with the function parameters a & b.

*You're encouraged to visit JM's original page located at:*http://hp41programs.yolasite.com/fracintegrdiff.php

A few elementary functions.

- Hyperbolic Sine $D^{\mu}\ \mathrm{Sinh}\ x = 2^{\mu-1}\ \mathrm{sqrt}(\pi)\ x^{1-\mu}\ {}_1\tilde{F}_2\ (\ 1\ ;\ (2-\mu)/2\ ,\ (3-\mu)/2\ ;\ x^2/4\ )$
- Hyperbolic Cosine $D^{\mu}\ \mathrm{Cosh}\ x = (2/x)^{\mu}\ \mathrm{sqrt}(\pi)\ \ {}_1\tilde{F}_2\ (\ 1\ ;\ (1-\mu)/2\ ,\ (2-\mu)/2\ ;\ x^2/4\ )$
- Sine $D^{\mu}\ \mathrm{Sin}\ x = 2^{\mu-1}\ \mathrm{sqrt}(\pi)\ x^{1-\mu}\ {}_1\tilde{F}_2\ (\ 1\ ;\ (2-\mu)/2\ ,\ (3-\mu)/2\ ;\ -x^2/4\ )$
- Cosine $D^{\mu}\ \mathrm{Cos}\ x = (2/x)^{\mu}\ \mathrm{sqrt}(\pi)\ \ {}_1\tilde{F}_2\ (\ 1\ ;\ (1-\mu)/2\ ,\ (2-\mu)/2\ ;\ -x^2/4\ )$
- Exponential $D^{\mu}\ \mathrm{Exp}\ x = x^{-\mu}\ {}_1\tilde{F}_1\ (\ 1\ ;\ 1-\mu\ ;\ x\ )$
- Logarithm $D^{\mu}\ \mathrm{Ln}\ x = x^{-\mu}\ FC^{(\mu)}_{\log}\ (x)$

where $FC^{(\mu)}_{\log}\ (x) = (-1)^{\mu-1}\ (\mu-1)\ !$ if μ is a positive integer,
and $FC^{(\mu)}_{\log}\ (x) = [\ \mathrm{Ln}\ x - \mathrm{Psi}(1-\mu) - \gamma\ ]\ /\ \Gamma(1-\mu)$ otherwise .

Psi = Digamma Function; γ= Euler's constant =0.5772156649... and Γ=Gamma Function.

Examples:

- Hyperbolic Sine:
  3.14 ENTER^, 1.28 XEQ "DSH" -> $D^{3.14}\mathrm{Sinh}\ (\ 1.28\ ) = 1.999005451$
- Hyperbolic Cosine:
  3.14 ENTER^, 1.28 XEQ "DCH" -> $D^{3.14}\mathrm{Cosh}\ (\ 1.28\ ) = 1.502958219$
- Sine:
  3.14 ENTER^, 1.28 XEQ "DSIN" -> $D^{3.14}\mathrm{Sin}\ (1.28) = -0.019142092$
- Cosine:
  3.14 ENTER^, 1.28 XEQ "DCOS" -> $D^{3.14}\mathrm{Cos}\ (1.28) = 0.888787267$
- Logarithm:
  3.14 ENTER^, 1.28 XEQ "DLN" -> $D^{3.14}\mathrm{Ln}\ (1.28) = 1.138569850$
- Exponential:
  3.14 ENTER^, 1.28 XEQ "DEXP" -> $D^{3.14}\mathrm{Exp}\ (1.28) = 3.501963669$

A few Special Functions.

- Sine Integral $D^{\mu}\ Si\ x = 2^{\mu-2}\ \pi\ x^{1-\mu}\ {}_2\tilde{F}_3\ (\ 1/2\ ,\ 1\ ;\ 3/2\ ,\ (2-\mu)/2\ ,\ (3-\mu)/2\ ;\ -x^2/4\ )$

- Hyperbolic Sine Integral $D^{\mu}\ Shi\ x = 2^{\mu-2}\ \pi\ x^{1-\mu}\ {}_2\tilde{F}_3\ (\ 1/2\ ,\ 1\ ;\ 3/2\ ,\ (2-\mu)/2\ ,\ (3-\mu)/2\ ;\ x^2/4\ )$

- Cosine Integral
$D^{\mu}\ Ci\ x = [\ FC^{(\mu)}_{log}\ (x) + \gamma\ /\ \Gamma(1-\mu)\ ]\ x^{-\mu} - 2^{\mu-3}\ sqrt(\pi)\ x^{2-\mu}\ \ {}_2\tilde{F}_3\ (1,\ 1\ ;\ 2,\ (3-\mu)/2,\ (4-\mu)/2\ ;\ -x^2/4)$

- Hyperbolic Cosine Integral
$D^{\mu}\ Chi\ x = [\ FC^{(\mu)}_{log}\ (x) + \gamma\ /\ \Gamma(1-\mu)\ ]\ x^{-\mu} + 2^{\mu-3}\ sqrt(\pi)\ x^{2-\mu}\ {}_2\tilde{F}_3\ (1,\ 1\ ;\ 2,\ (3-\mu)/2,\ (4-\mu)/2\ ;\ x^2/4)$

- Exponential Integral
$D^{\mu}\ Ei\ x = [\ FC^{(\mu)}_{log}\ (x) + \gamma\ /\ \Gamma(1-\mu)\ ]\ x^{-\mu} + x^{1-\mu}\ \ {}_2\tilde{F}_2\ (1\ ,\ 1\ ;\ 2\ ,\ 2-\mu\ ;\ x\ )$

- Fresnel Cosine Integral
$D^{\mu}\ C(x) = 2^{2\mu-3/2}\pi^{3/2}\ x^{1-\mu}\ {}_3\tilde{F}_4\ [\ 1/4\ ,\ 3/4\ ,\ 1\ ;\ (2-\mu)/4\ ,\ (3-\mu)/4\ ,\ (4-\mu)/4\ ,\ (5-\mu)/4\ ;\ -\pi^2\ x^4/16\ ]$

- Fresnel Sine Integral
$D^{\mu}\ S(x) = 2^{2\mu-11/2}\pi^{5/2}\ x^{3-\mu}\ {}_3\tilde{F}_4\ [\ 3/4\ ,\ 1\ ,\ 5/4\ ;\ (4-\mu)/4\ ,\ (5-\mu)/4\ ,\ (6-\mu)/4\ ,\ (6-\mu)/4\ ;\ -(PI)^2\ x^4/16\ ]$

- Spherical Bessel Function - 1st kind
$D^{\mu}\ j_n\ (x) = 2^{\mu-2n-1}\pi\ \ x^{n-\mu}\ \ \Gamma(n+1)\ \ {}_2\tilde{F}_3\ [(n+1)/2\ ,\ (n+2)/2\ ;\ (n+1-\mu)/2\ ,\ (n+2-\mu)/2\ ,\ n+3/2\ ;\ -x^2/4]$

- Modified Bessel Function - 1st kind
$D^{\mu}\ I_n\ (x) = 2^{\mu-2n}\ sqrt(\pi)\ x^{n-\mu}\ \ \Gamma(n+1)\ \ {}_2\tilde{F}_3\ [(n+1)/2,\ (n+2)/2;\ (n+1-\mu)/2,\ (n+2-\mu)/2,\ n+1;\ x^2/4]$

- Bessel Function - 1st kind
$D^{\mu}\ J_n\ (x) = 2^{\mu-2n}\ sqrt(\pi)\ \ x^{n-\mu}\ \ \Gamma(n+1)\ \ {}_2\tilde{F}_3\ [(n+1)/2\ ,\ (n+2)/2\ ;\ (n+1-\mu)/2\ ,\ (n+2-\mu)/2\ ,\ n+1\ ;\ -x^2/4]$

- Modified Bessel Function - 2nd kind; where n is not an integer.
$D^{\mu}\ K_n\ (x) = 2^{\mu-2n-1}\pi^{3/2}\ \ x^{-\mu-n}\ csc(n.\pi)\ \{16^n\Gamma(1-n)\ \ {}_2\tilde{F}_3\ [(1-n)/2,\ (2-n)/2;\ (1-\mu-n)/2,\ (2-\mu-n)/2,\ 1-n;\ x^2/4]\ -\ x^{2n}\Gamma/(n+1)\ \ {}_2\tilde{F}_3\ [(n+1)/2,\ (n+2)/2;\ (n+1-\mu)/2,\ (n+2-\mu)/2,\ n+1;\ x^2/4]\}$

- Bessel Function - 2nd kind; where n is not an integer.
$D^{\mu}\ Y_n\ (x) = 2^{\mu-2n}\ (PI)^{1/2}\ \ x^{-\mu-n}\ csc(n.\pi)\ \{\ -16^n\Gamma(1-n)\ \ {}_2\tilde{F}_3\ [(1-n)/2\ ,\ (2-n)/2\ ;\ (1-\mu-n)/2,\ (2-\mu-n)/2,\ 1-n\ ;\ -x^2/4\ ]\ +\ x^{2n}\ Cos(n.\pi)\ \Gamma(n+1)\ \ {}_2\tilde{F}_3\ [(n+1)/2,\ (n+2)/2\ ;\ (n+1-\mu)/2,\ (n+2-\mu)/2,\ n+1\ ;\ -x^2/4]\}$

- Generalized Laguerre's Functions
$D^{\mu}\ L^a_n\ (x) = [\ \Gamma(n+a+1)/\Gamma(n+1)\ ]\ x^{-\mu}\ {}_2\tilde{F}_2\ (1,\ -n;\ a+1,\ 1-\mu\ ;\ x)$

- Airy Functions
$D^{\mu}\ Ai(x) = 3^{\mu-4/3}\ x^{-\mu}\ \{\ 3^{2/3}\Gamma(1/3)\ {}_2\tilde{F}_3\ [\ 1/3\ ,\ 1\ ;\ (1-\mu)/3\ ,\ (2-\mu)/3\ ,\ (3-\mu)/3\ ;\ x^3/9\ ]\ -$
$-\ x\ \Gamma(2/3)\ {}_2\tilde{F}_3\ [2/3\ ,\ 1\ ;\ (4-\mu)/3\ ,\ (2-\mu)/3\ ,\ (3-\mu)/3\ ;\ x^3/9\ ]\ \}$

$D^{\mu}\ Bi(x) = 3^{\mu-5/6}\ x^{-\mu}\ \{\ 3^{2/3}\Gamma(1/3)\ {}_2\tilde{F}_3\ [\ 1/3\ ,\ 1\ ;\ (1-\mu)/3\ ,\ (2-\mu)/3\ ,\ (3-\mu)/3\ ;\ x^3/9\ ]\ +$
$+\ x\ \Gamma(2/3)\ {}_2\tilde{F}_3\ [2/3\ ,\ 1\ ;\ (4-\mu)/3\ ,\ (2-\mu)/3\ ,\ (3-\mu)/3\ ;\ x^3/9\ ]\ \}$

- Error Function $D^{\mu}\ Erf\ (x) = 2^{\mu}\ x^{1-\mu}\ {}_2\tilde{F}_2\ [\ 1/2\ ,\ 1\ ;\ (2-\mu)/2\ ,\ (3-\mu)/2\ ;\ -x^2\ ]$

- Hermite Function

$D^{\mu} H_n(x) = [\ 2^{n+\mu}(\pi)\, x^{-\mu} / \Gamma((1-n)/2)\ ]\ {}_2\tilde{F}_2\ [\ 1\ ,\ -n/2\ ;\ (1-\mu)/2\ ,\ (2-\mu)/2\ ;\ x^2\ ] -$
$- [\ 2^{n+\mu}\pi\, x^{1-\mu} / \Gamma((-n)/2)\ ]\ {}_2\tilde{F}_2\ [\ 1\ ,\ (1-n)/2\ ;\ 1-\mu/2\ ,\ (3-\mu)/2\ ;\ x^2\ ]$

- Kummer's Function $D^{\mu} F(a;b;x) = x^{-\mu}\Gamma(b)\ {}_2\tilde{F}_2\ (\ 1\ ,\ a\ ;\ 1-\mu\ ,\ b\ ;\ x\ )$

Examples:

- Sine Integral
  3.14, ENTER^, 1.28, XEQ "DSI" -> $D^{3.14}\ Si\ (\ 1.28\ ) = -0.045395644$

- Hyperbolic Sine Integral
  3.14, ENTER^ , 1.28, XEQ "DSHI" -> $D^{3.14}\ Shi\ (\ 1.28\ ) = 0.576495211$

- Cosine Integral
  3.14, ENTER^, 1.28, XEQ "DCI" -> $D^{3.14}\ Ci\ (\ 1.28\ ) = 1.367323895$

- Hyperbolic Cosine Integral
  3.14, ENTER^, 1.28, XEQ "DCHI" -> $D^{3.14}\ Chi\ (\ 1.28\ ) = 1.405640394$

- Exponential Integral
  3.14, ENTER^, 1.28, XEQ "DEI" -> $D^{3.14}\ Ei\ (\ 1.28\ ) = 1.982135606$

- Fresnel Cosine Integral
  3.14, ENTER^, 1.28, XEQ "DCX" -> $D^{3.14}\ C\ (\ 1.28\ ) = 16.95612253$

- Fresnel Sine Integral
  3.14, ENTER^, 1.28, XEQ "DSX" -> $D^{3.14}\ S\ (\ 1.28\ ) = -11.20302776$

- Spherical Bessel Function - 1st kind
  3.14, ENTER^, 2.41, ENTER^, 1.28 XEQ "DSB1" -> $D^{3.14}\ j_{2.41}\ (\ 1.28\ ) = -0.064451622$

- Modified Bessel Function - 1st kind , n # -1 , -2 , -3 , ...
  3.14, ENTER^, 2.41, ENTER^, 1.28, XEQ "DINX" -> $D^{3.14}\ I_{2.41}\ (\ 1.28\ ) = 0.352247279$

- Bessel Function - 1st kind , n # -1 , -2 , -3 , ...
  3.14, ENTER^, 2.41, ENTER^, 1.28 XEQ "DJNX" -> $D^{3.14}\ J_{2.41}\ (\ 1.28\ ) = -0.150524582$

- Modified Bessel Function - 2nd kind - non-integer order
  3.14, ENTER^, 2.41, ENTER^, 1.28 XEQ "DKNX" -> $D^{3.14}\ K_{2.41}\ (\ 1.28\ ) = -38.98469314$

- Bessel Function - 2nd kind - non-integer order
  3.14, ENTER^, 2.41, ENTER^, 1.28, XEQ "DYNX" -> $D^{3.14}\ Y_{2.41}\ (\ 1.28\ ) = 25.49308580$

- Generalized Laguerre's Functions
  3.14, ENTER^, 1.76, ENTER^, 2.41, ENTER^, 1.28 XEQ "DLANX" ->
$D^{3.14}\ L^{1.76}_{2.41}\ (\ 1.28\ ) = -1.767203465$

- Airy Functions
  3.14 ENTER^, 1.28 XEQ "DAIRY" -> $D^{3.14}$ Ai (1.28) = -0.162004857
X<>Y $D^{3.14}$ Bi (1.28) = 3.432592624

- Error Function
  3.14 ENTER^, 1.28, XEQ "DERF" -> $D^{3.14}$ Erf (1.28) = 1.250557023

- Hermite Function
  3.14 ENTER^, 2.41, ENTER^, 1.28 XEQ "DHMT" -> $D^{3.14}$ $H_{2.41}$ (1.28) = 3.537707646

- Kummer's Functions With a = sqrt(2) & b = sqrt(3)
  2, SQRT, STO 01, 3, SQRT, STO 02
  3.14 ENTER^, 1.28, XEQ "DKUM" -> $D^{3.14}$ F ($2^{1/2}$; $3^{1/2}$; 1.28) = 2.075891500

End of the Manual.-