

# **WL-Calc 1.1**

## **Wesley Loewer's Collection of Calculus Programs**

### **for the HP-49g/49g+**

July 2006

The programs included in WL-Calc are small but useful programs that I wrote for use in teaching Calculus. They have been very helpful to me and perhaps others may benefit from them as well. There are numerous other programs available which have similar functionality, however, the programs included WL-Calc have the following important advantages

- They follow the same syntax as similar functions built-in to the 49g/49g+.
- They take advantage of certain commands that were not available on the 48 series which makes them run a bit faster.
- They can be used in either RPN or Algebraic Mode
- All of the programs are written in pure UserRPL. They have been tested on a real 49g+ and a 49g emulator (Emu48). They should probably work fine on a real 49g and 50g.

The programs can be divided into several categories.

Numerical Root Solvers: NEWTON, BISECT

Numerical Integrators: RIEMANN, TRAP, SIMPSON, SIMPS38

Graphical Integrators: GrInt, MonteCarlo

Arc Length Calculators: ArcLen, EstArc

Error Functions: ERF, ERFC

## **Installation**

To install, just copy the WLCalc.hp directory to your calculator or emulator.

Some of the 49g/49g+'s built-in commands come in two versions (DERIV and DERVX for instance), one which requires you to specify the independent variable, and another in which the independent variable is assumed to be the system default independent variable. Most of the commands in WL-Calc also have two such versions: NEWTON and NEWTX, TRAP and TRAPX, etc.

These 'X' versions of the commands are simply tiny wrapper programs which call the full program with default arguments. If you find that having two or more versions of each command makes for a cluttered directory, you can safely delete any of the commands that end in 'X'. The full command will run fine without them. However, you cannot delete the full versions and use just the 'X' versions as the latter depend on the former.

## Description of Commands

### Numerical Root Solvers: NEWTON, NEWTX, BISECT, BSCTX

These commands are useful for demonstrating Newton's Method and the Method of Bisection for solving equations. The built-in ROOT command is certainly more powerful for solving equations, but these are useful for teaching and learning the principles of calculus.

#### NEWTON

The NEWTON command has the same syntax as the built-in ROOT command described by the following stack diagram:

Level 3	Level 2	Level 1	→	Level 1
expression	variable	guess	→	root

For example, to find the root of SIN(X) near 3.0, the following stack diagram would be used.

Level 3	Level 2	Level 1	→	Level 1
'SIN(X)'	'X'	3.0	→	3.14159265359

For those unfamiliar with stack diagrams, the stack diagram above means that you enter the values shown below on the left, then execute the NEWTON command, which results in the values shown below on the right.

```

RAD XYZ DEC R= 'X'
{HOME WLCALC}  USR
7:
6:
5:
4:
3: SIN(X)
2: 'X'
1: 3.
NEWTX|NEWTO|

```

→

```

RAD XYZ DEC R= 'X'
{HOME WLCALC}  USR
7:
6:
5:
4:
3:
2: 3.14159265359
1:
NEWTX|NEWTO|

```

For those who prefer to use **Algebraic Mode** instead of **RPN Mode**, the stack diagrams shown also describe the order of the arguments for algebraic entry. For example, to use the NEWTON command shown above, you would enter:

**NEWTON(SIN(X),X,3.)**

All of the programs in WL-Calc can be used in either RPN or Algebraic Mode.

```

RAD XYZ HEX R= 'X'
{HOME WLCALC}  USR  ALG
: NEWTON(SIN(X),X,3.)
3.14159265359
HEX ■ DEC ■ OCT ■ BIN ■ R←B ■ B←R

```

## NEWTX

The NEWTX command is similar to NEWTON, but assumes the variable is the default independent variable, usually 'X'.

Level 2	Level 1	→	Level 1
expression	guess	→	root

Example:

Level 2	Level 1	→	Level 1
'SIN(X)'	3.0	→	3.14159265359

## BISECT

The BISECT command is similar to the NEWTON command, except it takes two initial guesses: the left bound and the right bound. The function evaluated at the left and right guesses must have different signs (one positive and the other negative). The program simply evaluates the function at these guesses and their midpoint, then repeatedly divides the interval in half until it gets sufficiently close to the root. This is not a very efficient method, but is important in developing the concept of a limit.

Level 4	Level 3	Level 2	Level 1	→	Level 1
expression	variable	left bound	right bound	→	root

Example:

Level 4	Level 3	Level 2	Level 1	→	Level 1
'SIN(X)'	'X'	3.0	4.0	→	3.14159265358

## BSCTX

The BSCTX command is similar to the BISECT command, but assumes the variable is the default independent variable, usually 'X'.

Level 3	Level 2	Level 1	→	Level 1
expression	left bound	right bound	→	root

Example:

Level 3	Level 2	Level 1	→	Level 1
'SIN(X)'	3.0	4.0	→	3.14159265358

## Numerical Integrators: RIEMANN, TRAP, SIMPSON, SIMPS38

For comparison, the built-in definite integration,  $\int$ , has the following stack diagram:

Level 4	Level 3	Level 2	Level 1	→	Level 1
lower limit	upper limit	integrand	variable	→	integral

Example:

Level 4	Level 3	Level 2	Level 1	→	Level 1
0	$\pi$	'SIN(X)'	'X'	→	2.

## RIEMANN

The RIEMANN command for calculating Riemann Sums adds two more arguments: one indicating the number of steps to use, and another indicating whether the function should be evaluated on the left edge, the right edge, or the midpoint.

Level 6	Level 5	Level 4	Level 3	Level 2	Level 1	→	Level 1
lower limit	upper limit	integrand	variable	steps	lmr	→	integral

where lmr indicates whether the sum is evaluated on the left, middle, or right of the section (Left = 0., Middle = 1., Right = 2.).

Example: Integrate SIN(X) from 0 to  $\pi$  using 24 increments evaluated on the right.

Level 6	Level 5	Level 4	Level 3	Level 2	Level 1	→	Level 1
0	$\pi$	'SIN(X)'	'X'	24	2	→	1.99714339581

## RIEMLX RIEMMX RIEMRX

These three commands are shortcuts for Riemann Sums evaluated on the left edge, midpoint, and right edge respectively, assuming the variable is the default independent variable, usually 'X'.

Level 4	Level 3	Level 2	Level 1	→	Level 1
lower limit	upper limit	integrand	steps	→	integral

Example:

Level 4	Level 3	Level 2	Level 1	→	Level 1
0	$\pi$	'SIN(X)'	24	→	1.99714339581

### TRAP SIMPSON SIMP38

These three commands find numerical definite integrals by the Trapezoid Rule, Simpson's Rule, and Simpson's 3/8 Rule. All three have the same syntax. For Simpson's Rule, the number of steps must be even; and for Simpson's 3/8 Rule, 'steps' must be a multiple of 3. If these criteria are not met, 'steps' is rounded up to the next multiple of 2 or 3.

Level 5	Level 4	Level 3	Level 2	Level 1	→	Level 1
lower limit	upper limit	integrand	variable	steps	→	integral

Example: Integrate SIN(X) from 0 to  $\pi$  using 24 increments.

Level 5	Level 4	Level 3	Level 2	Level 1	→	Level 1
0	$\pi$	'SIN(X)'	'X'	24	→	2.00000326887

### TRAPX SIMPX SMP38X

These three commands are shortcuts to TRAP, SIMPSON, SIMP38, but assume the variable is the default independent variable, usually 'X'.

Level 4	Level 3	Level 2	Level 1	→	Level 1
lower limit	upper limit	integrand	steps	→	integral

Example:

Level 4	Level 3	Level 2	Level 1	→	Level 1
0	$\pi$	'SIN(X)'	24	→	2.00000326887

## Graphical Integrators: GrInt, MonteCarlo

### GrInt

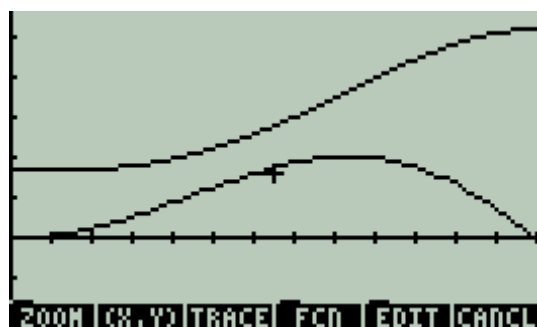
GrInt is a Graphical Integrator which graphs the indefinite integral and approximates the definite integral of a function. A more technical description will follow, but an example will help explain how this command works.

Example:

Say you want to graph the integral of 'X·SIN(X)' from 0 to  $\pi$  with an integration constant of 1.5. First go to the Y= window and make X·SIN(X) the only function listed. Then go to the WIN editor and indicate the integration limits by setting the H-View values to be 0 to 3.14159 and the V-View to be -2 to 5.



Press ERASE then DRAW to see what the original function looks like. Then exit out of the graph screen back to the stack, enter an integration constant 1.5, and run GrInt. You should see the graph shown below. When you exit back to the stack, the approximate definite integral (area under the curve) is shown.



You do not actually have to graph the original function — GrInt can graph just the integral.

And now for a few more technical details:

- GrInt reads the EQ and PPAR variables to determine the graphing properties.
- The function to be graphed does not really have to be only function, but it does have to be the *first* function. From the Y= screen, press NXT to get to the MOVE↑ and MOVE↓ commands, if necessary, to make the desired function the first one.
- It is possible to set the integration limits different from the H-View settings by using the Indep Low and High values for the integration limits.
- The Step size is used when graphing the integral.
- The graph can be connected or points only as specified in Plot Setup (System Flag 31)

- The independent variable does not have to be 'X'.
- The “integration constant” simply specifies the  $y$  value on the left edge of the graph. If the bottom of the stack (Level 1) does not contain a number, then the graph starts at  $y=0$ .

The stack diagram is

Level 1	→	Level 1
[integration constant]	→	approximate area under curve

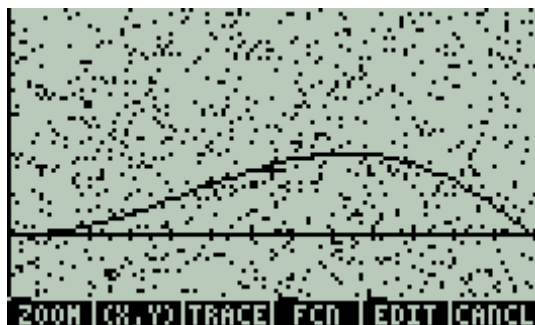
where the optional integration constant is assumed to be zero if not present or if it is not a number.

## MonteCarlo

This command gives a graphical demonstration of the Monte Carlo method of determining the area under a curve. This method is not very practical or even accurate, but it is a clever concept and can be fun to see in practice. The idea is to pick random points on the screen and count how many hit within the area under the curve. The ratio of “hits”：“total points” should be approximately the same as the ratio “area under curve”：“total area in window”. The more points you plot, the more accurate it will be.

Example:

Enter the same function and window settings as in the GrInt example above. On the stack, enter the number of points you want to plot, say 1000. Then run MonteCarlo. After it has finished plotting point, the approximate area is returned on the stack.



The stack diagram is

Level 1	→	Level 1
number of points	→	approximate area under curve

Example:

Level 1	→	Level 1
1000.	→	3.25468724

## Arc Length Calculators: ArcLen, EstArc

These two functions calculate the arc length of a function between two points.

### ArcLen

ArcLen generate the integral that calculates the arc length of a given function. If the calculator is in Exact Mode, the integral will be left in integral form if the integral cannot be solved symbolically. If the calculator is in Approximate mode, the integral will be evaluated numerically.

Level 4	Level 3	Level 2	Level 1	→	Level 1
lower limit	upper limit	expression	variable	→	length

Examples:

Level 4	Level 3	Level 2	Level 1	→	Level 1
0	1	$X^2$	'X'	→	'-((LN(-2+ $\sqrt{5}$ )-2* $\sqrt{5}$ )/4)'

Level 4	Level 3	Level 2	Level 1	→	Level 1
0.0	1.0	$X^2$	'X'	→	1.47894285755

### ArcLX

Same as ArcLen, but assumes the variable is the default independent variable, usually 'X'.

Level 3	Level 2	Level 1	→	Level 1
lower limit	upper limit	expression	→	length

Examples:

Level 3	Level 2	Level 1	→	Level 1
0	1	$X^2$	→	'-((LN(-2+ $\sqrt{5}$ )-2* $\sqrt{5}$ )/4)'

Level 3	Level 2	Level 1	→	Level 1
0.0	1.0	$X^2$	→	1.47894285755



**EstArc**

Estimates arc length by dividing the function into small increments and using a linear approximation (distance formula) to estimate the length of each increment.

Level 5	Level 4	Level 3	Level 2	Level 1	➔	Level 1
lower limit	upper limit	expression	variable	steps	➔	length

Example:

Level 5	Level 4	Level 3	Level 2	Level 1	➔	Level 1
0	1	$X^2$	'X'	40	➔	1.4788962724

**EstAX**

Estimates arc length assuming the variable is the default independent variable, usually 'X'.

Level 4	Level 3	Level 2	Level 1	➔	Level 1
lower limit	upper limit	expression	steps	➔	length

Example:

Level 4	Level 3	Level 2	Level 1	➔	Level 1
0	1	$X^2$	40	➔	1.4788962724

### Error Functions: ERF, ERFC

The Error Function and Complementary Error function are used to evaluate certain integrals. They are defined to be:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$
$$\text{erfc}(x) = 1 - \text{erf}(x)$$

**ERF**  
**ERFC**

Level 1	→	Level 1
number	→	result

ERF Example:

Level 1	→	Level 1
0.75	→	0.711155633654

ERFC Example:

Level 1	→	Level 1
0.75	→	0.288844366346