StatiCal

The Statistician's Calculator (release 1)

User Manual

By
John C. Pezzullo, PhD

Updated March 21, 2014

Table of Contents

Introduction

This manual describes the app "*StatiCal* – The Statistician's Calculator".

StatiCal is a full-function scientific calculator that also performs a variety of statistical calculations. Rather than listing them all here, I'd just ask you to take a look at the Table of Contents for this manual to get an idea of *StatiCal*'s capabilities

A few things about *StatiCal* as an smartphone / tablet app:

- *StatiCal* should run on all reasonably current versions of the device's operating system.
- *StatiCal* is a "lean and mean" app – the installed app is about ½ MB.
- *StatiCal* is completely free, and it will remain so as long as I'm around. It has no ads, banners, or other commercial add-ons. There are no "trial" or "limited edition" or "Pro" versions; there's only the **Full** version.
- *StatiCal* requests no special permissions to run – it does not modify the device's SD Card storage, does not access phone information, does not access the network, etc.
- *StatiCal* currently does not run on the iPhone, iPad, or any other iOS app.

History

I'm a semi-retired statistician, but I still do statistical consulting work.

Many years ago, I had a Palm PDA (personal digital assistant). I bought a program (called CASL) that let me develop my own apps (they weren't called apps back then) for the Palm operating system. One of these apps was a scientific calculator that could also calculate p-values for the Normal, Chi-square, Student t and Fisher F distribution, and their inverses. This proved to be very handy, and I used the Palm app as my main calculator until smart phones supplanted PDAs.

Unfortunately, CASL was never updated to work with the newer smart phones or tablets, and it's now a dead product. But I really missed my old calculator, and I searched around for an Android app that could do statistical calculations within the context of a general scientific calculator. Not finding any pre-existing apps fitting that description, I then looked around for an easy-to-use software environment to develop Android apps for the smart phone I was using.

After several false starts, I discovered "Basic for Android" (B4A), which was just what I needed. It had all the math and programming features needed to create a compact app, and used a programming language that was close enough to what I was already familiar with, so that I could actually learn it. So over the course of a month or so, I developed *StatiCal*. (This manual took another two months to write!) I was also able to port the application over to the Blackberry Playbook tablet.

StatiCal does everything my old Palm calculator did, and a lot more. I wanted to put into it every statistical calculation that was in any way amenable to a hand-held device. Obviously, this would not include multiple regression on data sets with hundreds of variables and thousands of cases, but I wanted it to handle anything where the data could be conveniently keyed into a hand-held device.

So I created *StatiCal* for myself. But now that it's a reasonably stable and reliable working app, I'm sharing it with the world. This is in keeping with the philosophy that has guided another of my pet projects – the *StatPages.org* web site, which provides free statistical software via online web-based calculators

If you like *StatiCal*, or if you think of other features that would be useful to have, let me know. If you don't like it, at least you have the consolation of knowing that you didn't sink any money into it.

General Operation of the Calculator

To understand how *StatiCal* works, you have to be aware of two things:

Reverse Polish Notation (RPN)

First and foremost, *StatiCal* is a "reverse-Polish-notation", or RPN calculator. That means that you enter numbers and operations differently from how you enter them into simple "algebraic" calculators. Usually, to add two numbers on a simple algebraic calculator, you might hit the keys this way:

5, +, 3, =

and you'd see the answer 8.

But an RPN calculator doesn't have an = key; and it has an Enter key that you don't find on simple calculators. For the above calculation, you hit the keys this way:

5, Enter, 3, +

and you'd see the answer 8, as before.

Notice that with an RPN calculator, you first give it a number, then tell it what you want it to do with that number.

The Stack

The second thing to be aware of is that *StatiCal* uses a *stack* to hold the results of calculations. *StatiCal's* stack is a storage mechanism that (in the current implementation of *StatiCal*) can hold up to 100 numerical quantities. Each quantity can be a positive or negative number with up to 16 significant digits. Numbers can range in magnitude from about 10^{-308} to 10^{+308}.

Numbers get "pushed" onto the stack when you enter them, or do calculations on them. A pushed number appears at the bottom of the three number fields at the top of the calculator. Whatever value that was in that field gets slid up one space to the middle field, and whatever value that was in the middle gets slid to the top field. The value that was in the top field (the third entry in the stack) gets pushed out of sight, but it's still in the stack (in the fourth position).

Normally, you place numbers on the stack, and then indicate the operation you want done on those numbers. Then the results of the operation are placed on the stack, where they are available for further calculations. Whenever you perform some operation, whether it's as simple as adding two numbers, or as complicated as performing an ANOVA on five groups of numbers, one rule is always the same:

*All the **input** to the operation (the "arguments") are removed ("popped") from the stack, and all the **results** of the operation are placed ("pushed") onto the stack.*

Stack Advantages

Hewlett-Packard has been making RPN calculators since the 1970's. They have a good web page that explains RPN:

http://h41111.www4.hp.com/calculators/uk/en/articles/rpn.html

I'll just summarize some of the main benefits of RPN calculators:

1. They don't have (and don't need) an = key; the result of any calculation is displayed immediately once you say what calculation to perform.
2. They don't have (and don't need) parenthesis keys. Complicated arithmetic formulas can be evaluated in a very simple and intuitive way (once you get the hang of RPN), and you can see the intermediate results of complicated expressions as you evaluate them.
3. You don't have to memorize any "operator priority" rules (like multiplication having a higher priority than addition.
4. RPN calculators are very consistent. Simple algebraic calculators are actually partly "stealth RPN" calculators, without telling you that, leading to conceptual inconsistencies. For example, to subtract a number, you first hit the subtract key then enter the number to be subtracted. But to evaluate square roots on a simple calculator, you first enter the number to be square-rooted, then hit the square root key. Not very consistent, is it.
5. RPN calculators are very "transparent" in their evaluation of complicated expressions, showing you the results of intermediate sub-expressions that go into the big expression. Algebraic calculators (those that let you enter complicated algebraic expressions to be evaluated) actually maintain a stack to hold intermediate results arising from parenthesized sub-expressions or operator hierarchy rules, but they hide it from you.
5. The RPN / Stack framework smoothly extends to handle statistical tests and functions that require several arguments and return several results.

Exiting the StatiCal app – Quitting vs. Suspending

As you already know, the standard Android user interface provides four standard buttons (either actual buttons, or on-screen icons): Menu, Home, Back, and Search. Two of these buttons can be used to exit *StatiCal* and return to the screen from which you had launched it (either one of your home screens, or the all-apps window). These two keys work in different ways:

The **Home** button **suspends** the *StatiCal* app, and saves its current status, so that the next time you launch *StatiCal*, it will come up in exactly the same state it was in when you left it. The stack and memory cells will still have their earlier values; it will show the panel you were using, the toggles and modifiers will be as you left them; etc. Use the Home button when you want to temporarily leave *StatiCal* to do something else, but plan to return to *StatiCal* to continue what you were working on.

The **Back** button **terminates** the *StatiCal* app, and doesn't save your work. The next time you launch *StatiCal*, you'll see the standard start-up configuration: the stack and memory cells will be zeroed out; the Num keypad panel will be showing; etc.

Keys and Panels

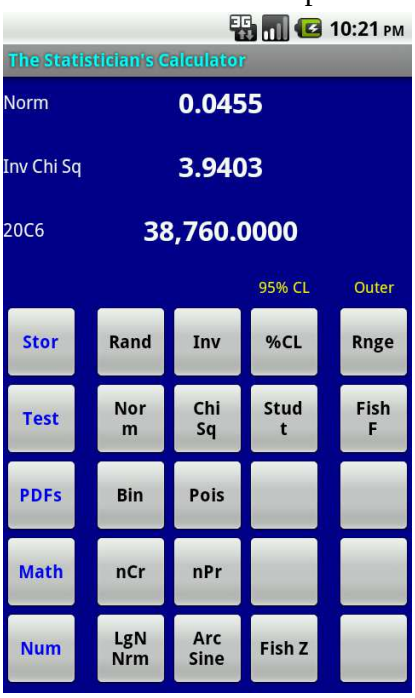
Scientific calculators can have lots of keys, because there are lots of scientific functions you might want to evaluate. Because I wanted *StatiCal* to be a full-function scientific calculator, and to also have statistical functions and tests, I realized that *StatiCal* would require 100 or more keys. Since smart phones normally use on-screen keyboards, cramming 100 different keys into a phone's window would result in tiny, closely-spaced keys. This might be workable for someone with tiny fingers, but not for my "man hands".

So I took a different approach. *StatiCal*'s keyboard layout looks very simple and uncluttered. There are only 25 keys on the screen, in a fairly spacious 5-by-5 layout. How do I get 100 keys into a 25-key screen? Simple – I define 5 different keypad screens, called *panels*, each containing 25 keys:

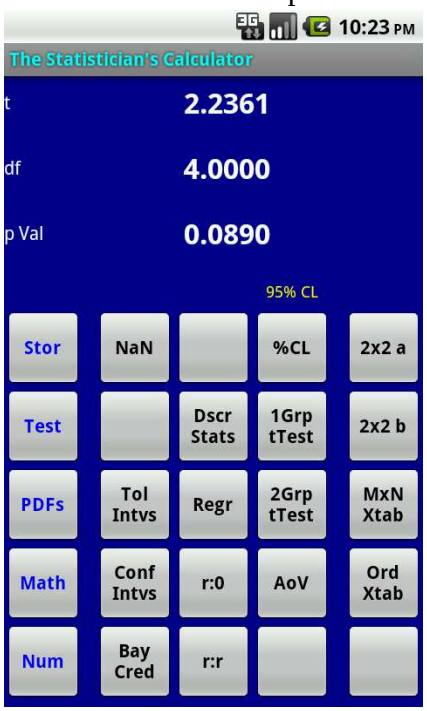
the **Num** Keypad Panel

the **Math** functions panel

the **PDFs** functions panel



the statistical **Test** panel



the **Stor** manipulations panel

You'll notice that the five left-most keys (with the blue letters) are the same on all five panels – they're the "panel-switching" keys, and tapping any of these keys instantly switches you from one panel to another.

Result Fields

StatiCal has, at the top of the screen, three fields where the results of data entry and calculations are displayed. These three fields show you the top three entries in the stack; these will be your most recently entered data, and/or the results of the most recent calculations on your data.

When you start up *StatiCal*, these fields are empty (because the stack is initially empty), and you might not even be aware that there are three fields there.

Field Labels

Each field has a label, in small type to the left of the field, that describes how that entry in the stack got there – whether it was a number you entered directly from the numeric keypad, or whether it was the result of some operation on data you had already entered.

Setting the Display Format

When you first launch *StatiCal*, it will show the first three stack positions. Since nothing is yet on the stack, these three positions will show the value 0.

Numbers can be displayed in two different formats: Fixed-point and Floating-point.

Fixed-point numbers will always show the same number of digits after the decimal point. You can control how many of these digits you want to see. You might want no fractional decimal digits at all (values would be rounded to nearest whole number); or you might want 2 decimal digits (good for money calculations); or you might want 6 decimal digits (for high-precision work).

But very large and very small numbers don't display well in fixed-point format. Very large numbers might spill over onto two or more lines, in a very unsightly and hard-to-read form. And very small numbers will have a lot of leading zeros after the decimal point.

Scientists and engineers use what's called "scientific notation" to easily represent very large or very small numbers in the format: x.xxxxxxxxxxxxxxxE±xx

Scientific notation will show as many significant digits as necessary (up to 16), with an exponent part if necessary.

You can control the appearance of the numbers by tapping on the number fields:

- To switch to scientific notation display, tap the middle number.
- To switch from scientific notation to fixed-point display, tap the upper or lower number.
- To increase the number of decimal places in fixed-point display, tap the upper number.
- To decrease the number of decimal places in fixed-point display, tap the lower number.

Toggle and Modifier Values

To keep the number of required keys down to a manageable level, most scientific calculators have several "toggle" keys to modify the way certain other keys work. And *StatiCal* is no exception.

For example, many scientific calculators have three trigonometric function keys: Sin, Cos, and Tan. They could have added three more keys for the inverse trig functions: ArcSin, ArcCos, and ArcTan, but it's easier to just have a single Arc key, so that if you tap the Arc key and then tap the Sin key, you'll get the ArcSin. (Some functions have a "shift" or "alternate" or "function" or some other kind of general toggle key, and then print the name of the alternate function in the space above the key.

Since we sometimes want trig functions to work in degrees, and other times in radians, most scientific calculators also have some way to toggle between degree and radian mode.

And since many scientific calculators provide hyperbolic sines, cosines and tangents as well as the more familiar circular trig functions, they also might provide a way to toggle between the these two families of functions.

Without this toggling scheme, scientific calculators would have to devote 24 keys to the circular and hyperbolic functions and their inverses, in degree and radian mode. But with toggling, only 6 keys are required – the Sin, Cos and Tan keys, an Inverse toggle, a Degrees/Radians toggle, and Circular/Hyperbolic toggle.

Getting Help

There are currently two sources of help for using *StatiCal*.

One is the *StatiCal Manual* you're reading now. Since free PDF-reader apps are readily available for most smart phones and tablets, you might want to store this PDF file on your device so that it will always be handy.

The other is a set of help messages that are built into *StatiCal*. If you "long-press" any key (hold it down for about a second), a box will pop up containing a concise description of what that key does, and how to use it (that is, what you have to place on the stack before you tap the key, what will be on the stack after you tap the key, and how the various toggles and modifiers affect the operation of that key.

Some Conventions Used in This Manual

For consistency, I've tried to adhere to certain notational conventions throughout this manual.

How Examples Are Shown

Almost every page of this manual contains specific examples of how the calculator works. I try to adhere to a standard format and notational convention for these examples.

First, examples will usually be shown in a table with two or three columns, like this:

All of the examples in this manual will use a consistent notation to indicate your keystrokes. For example, to enter the number 25, just tap the 2 key and then tap the 5 key, then tap the Enter key.

2, 5, Enter

To simplify this notation, I'll omit the commas between the individual digits of a single numeric entry:

25, Enter

To enter the quantity -25, tap the – key at the top of the keypad, then the 2 key, then the 5 key, then the Enter key:

-25, Enter

The x, y, and z designations

In the examples, and in the naming of many of the keys, you'll sometimes see the designations "x", "y", and "z". These stand for numbers you've just entered onto the stack, *in that order*.

So when you see something like "x^y", it means that if you first enter a 5 onto the stack, and then enter a 2 onto the stack, then hit the x^y key, it will evaluate 5^2 (which produces 25), not 2^5 (which would produce 32).

The Num Panel– Entering Numbers and Performing Simple Arithmetic

The Num Keypad panel is used to enter numbers into the calculator, and to perform the four simple arithmetic operations: addition, subtraction, multiplication and division. In effect, this panel is your basic "4-banger" calculator (but with RPN, of course).

You can instantly switch to the **Numeric Keypad** panel at any time by tapping the blue **Num** key in the left column of any panel.

The << key (Backspace)

The << key is used to erase the last keystroke when entering a number. You can think of it as a "backspace" key.

So the following sequence:

1, 2, 4, <<, 3, Enter

will place the value 123 on the stack.

You can hit << several times; each time will erase the rightmost character from the number being entered.

The << key is active only while a number is being entered into the calculator; that is, while the number appearing in the bottom field is shown in yellow. Once you hit Enter, the number turns white, and is no longer available for editing, and the << key will not have any effect.

The – key

The **–** key at the top of the keypad is used to enter a negative number, or a negative exponent in a scientific-notation number. So, to enter the number -25, or 1.23E-6, or -1.23E-6, you would use the upper-row – key at the beginning of the number and after the E.

Don't use this key to subtract two numbers; use the "**x-y**" key at the right of the keypad.

The E key and Scientific Notation

Scientists and engineers often have to work with very large or very small numbers, which would be very cumbersome to deal with in the usual notation.

To enter the quantity one million (which is 1000000, or $1x10^6$), you can do it directly:

1000000, Enter

or using scientific notation:

1E6, Enter

To enter the quantity one millionth, you can do it as:

0.000001, Enter

or as:

1E-6, Enter

The largest number *StatiCal* can deal with is 1.7976931348623157 E+308, and the smallest nln-zero number is 2.2250738585072014 E–308.

When is the Enter Key Necessary?

You must use the Enter key after entering a number only if your next action will be to enter another number.

So if you want to add 25 and 36, you would do it as follows:

25, Enter, 36, +

You could also have done it as:

25, Enter, 36, Enter, +

The first **Enter** is required, but the second **Enter** is optional. The first **Enter** is needed to separate the digits of the number 25 from the digits of the number 36, so that the calculator won't think you were trying to enter the number 2536. The second **Enter** is not needed because whenever you tap a mathematical operation or function key, *StatiCal* will automatically, in effect, tap the Enter key for you.

*The Simple Arithmetic Keys: x+y, x–y, x*y, x/y*

These keys implement the four basic arithmetic operations between the two most recently entered numbers on the stack. In each case, the result of the operation replaces the two original operands. What happens is this:

1. the two original numbers are removed from the stack;
2. the arithmetic operation is performed between them; and
3. the result is placed onto the stack.

So, for example:

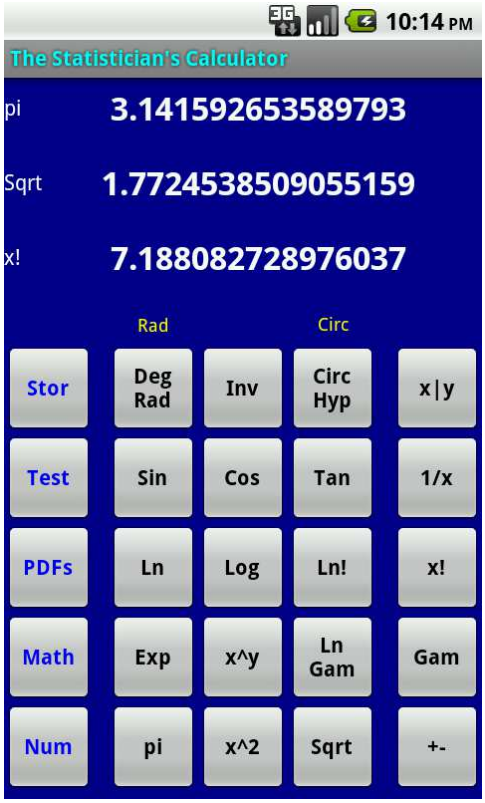
this key sequence	gives this result
5, Enter, 2, x+y	x+y 7
5, Enter, 2, x-y	x-y 3
5, Enter, 2, x*y	x*y 10
5, Enter, 2, x/y	x/y 2.5

Notice that at the left of each output field is an indication of what keystroke produced the result displayed in that field.

Note that this behavior (the result of the operation replacing the number or numbers that went into the operation) is applied consistently by *StatiCal* for all of its calculations, be they as simple as squaring a number or adding two numbers, or as complicated as calculating an ANOVA, where many numbers go into the calculation, and the calculation produces several results. The general rule is that **all inputs to the calculation will popped off of the stack, and all results will be pushed onto the stack**.

The Math Panel – Advanced Mathematical Calculations

The Math Panel implements the mathematical, logarithmic, exponential, and trigonometric functions found on most "scientific" calculators.



You can instantly switch to the **Mathematical Calculations** panel at any time by tapping the blue **Math** key in the left column of any panel.

In all of the examples that follow, I'll indicate the key that has to be tapped to switch over to the appropriate panel. If you're already at that panel, you don't have to tap that panel key, although there's no harm in doing so.

The Sqrt key: Square Root

The **Sqrt** key takes the square root of the number at the top of the stack (the bottom of the three displayed numbers). The square root replaces the original number (actually, the original number is popped off of the stack, and its square root is pushed onto the stack). The remainder of the stack is unaffected. So, for example:

this key sequence	gives this result
Num, 25, Math, Sqrt	Sqrt 5
Num, 0, Math, Sqrt	Sqrt 0
Num, -, Math, Sqrt	Sqrt NaN

The square root of zero is zero, and the square root of any negative number is the special value "NaN" (Not a Number). *StatiCal* does not work with complex numbers (those consisting of a real and an imaginary part). If you don't know what that last sentence means, don't worry about it; just remember that you cannot take the square root of a negative number.

The square root of Infinity is Infinity, and the square root of –Infinity is NaN. The square root of NaN is NaN.

The x^2 key: Squaring a Number

The **x^2** key simply squares the number at the top of the stack (the bottom of the three displayed numbers). That is, it multiplies that number by itself. The result replaces the original number on the top of the stack; the rest of the stack is unaffected..

The number to be squared can be any number, positive or negative, whole or fractional. The square will always be positive (negative times negative is positive).

If the number to be squared is greater than about 1.34E+154, the square will exceed the largest implemented floating-point number, and the result will be the special value "Infinity".

The square of Infinity is Infinity, and the square of –Infinity is also Infinity.

The square of NaN is NaN.

Squares and square roots "undo each other". That is, squaring 5 produces 25, and taking the square root of 25 produces 5 again. Similarly, the square root of 16 is 4, and squaring 4 produces 16 again. This is always true, but the results might be affected by round-off error in the calculator's floating-point arithmetic.

So, for example:

this key sequence	gives this result		because
Num, 1E200, Math, x^2	x^2	Infinity	1E400 is beyond the *StatiCal*'s floating-point range
Num, 1E-200, Math, x^2	x^2	0	1E-400 is beyond the *StatiCal*'s floating-point range

The x^y key: Raising a Number to a Power

The **x^y** key raises any base (x) to any power (y); that is, it generates x^y, often written as x^y or x**y in various computer languages.

Enter the base number (x), then the power number (y), then press the **x^y** key. The result x^y will replace x and y on the stack. So, to raise 4 to the third power:

this key sequence	gives this result	
Num, 4, Enter, 3, Math, x^y	x^y	64

The power can be positive or negative number, integer or fractional, and the base can be any **positive** number, integer or fractional, for any value of the power. So we have:

this key sequence	gives this result
Num, 1.7, Enter, -3.1, Math, x^y	x^y 0.19302

But the base can be **negative** if, *and only if*, the power is an **even integer**. So we have:

this key sequence	gives this result
Num, -1.7, Enter, 2, Math, x^y	x^y 2.89

If the base is negative and the power is anything other than an even integer, the calculator will return the value NaN, because in those cases the answer is not a real number (it is a *complex* number, with *real* and *imaginary* parts, and ***StatiCal*** does not work with complex numbers).

The x! key: Factorials

For a positive whole number, n, the factorial of n (written as n!) is the product of all the whole numbers from 1 to n, inclusive. So, 4! is equal to 1*2*3*4, or 24.

Zero factorial (0!) is defined as a special case to be equal to 1 (not 0 as you might expect).

The above definition doesn't make sense for fractional numbers, so the factorial definition can be "extended" to work for all real numbers (positive or negative, whole number or fractional). There are several ways this can be done, the most widely accepted way is to define it as the following integral:

$$n! = \int_0^\infty x^n e^{-x} dx$$

This gives exactly the same values as the simpler definition when n is a whole number, but also works for fractional n. This generalized definition is related to the definition of the "gamma function" (generally represented by Γ, the Greek capital gamma):

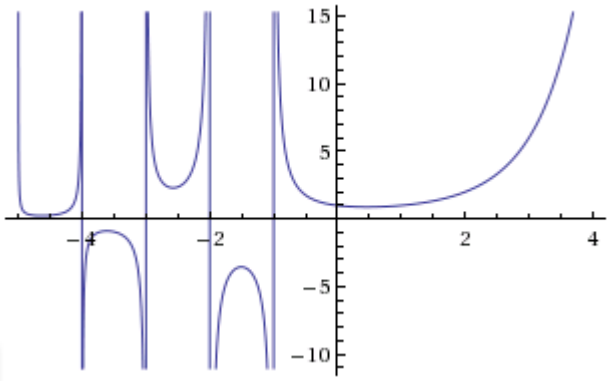
$$n! = \Gamma(n + 1), \text{ or } \Gamma(n) = (n - 1)!$$

The factorial function gets *very* large as n gets large:

- 4! = 24;
- 5! = 120;
- 10! = 3,628,800; and
- 100! = 9.3326×10^{157}.

For n larger than 170, n! exceeds the limit of the calculator's floating-point number system, so the calculator returns "Infinity" as the answer. (This limit might be different for different types of hardware or operating systems.)

For negative n, the factorial function behaves in a very peculiar fashion:



n! alternates between positive and negative values for negative n, and at negative whole numbers, it goes to infinity, in both the positive and negative directions. So for negative whole numbers, the calculator returns NaN, since n! has opposite signs on either side of the whole number, and one cannot say whether it's +∞ or –∞ at the whole number.

So, for example:

this key sequence	gives this result
Num, 1E200, Math, x!	x! Infinity

The Ln! key: Natural Logarithm of the Factorial of a number

We sometimes want to use factorials of very large numbers as intermediate expressions in a calculation where the final result of the calculation might not be very large. For example: 300! divided by 298! happens to equal 299 * 298, which is equal to the relatively small value 89,700. But if you try to calculate the value of (300! / 298!) in the obvious way (calculate 300!, calculate 298!, then divide), it won't work, because the intermediate values (300! and 298!) will both be calculated as Infinity, and Infinity / Infinity is undefined, and produces NaN:

For example

this key sequence	gives this result
Num, 300, Math, x!, Num, 298, Math, x!, Num, x/y	x/y NaN

In such cases, it is convenient to work in logarithms (or, as mathematicians like to say, "in the logarithmic domain"). For example, the rules of logarithms tell us that Ln(x/y) = Ln(x) – Ln(y), so:

$$\text{Ln}(300! / 298!) = \text{Ln}(300!) - \text{Ln}(298!)$$

therefore:

$$300! / 298! = \text{Exp}(\text{Ln}(300!) - \text{Ln}(298!))$$

Now Ln(300!) happens to equal 1414.90585, but you can't arrive at this value by entering 300, calculating the factorial, then taking the log, because calculating 300! in *StatiCal* (or just about any calculator) will give "Infinity" (or a "floating point overflow error), and the log of infinity is still infinity. Somehow, you have to be able to calculate Ln(300!) without having to calculate 300! first.

So *StatiCal* provides a **Ln!** key that calculates the natural log of x! by a special procedure that does not have to calculate x! first. So it will work for huge numbers.

For example:

this key sequence	gives this result
Num, 300, Math, Ln!	Ln! 1,414.9058
Num, 300, Math, Ln!, Num, 298, Math, Ln!, Num, -, Math, Exp	Exp 89,700

As another example, the natural log of 10,000! is easily calculated using *StatiCal*'s **Ln!** key as 82,108.92783681436, which agrees with the number provided by Wolfram Alpha.

The Gam key: Gamma Function

The Gamma function is related to the factorial function by the expression:

$$\text{Gamma}(x) = (x - 1)!$$

or:

$$x! = \text{Gamma}(x + 1)$$

So if we can calculate either one of these two functions (gamma or factorial), we can easily calculate the other, so I only needed to provide either factorial or gamma. But since people (like me) are very prone to forgetting which one is the +1 and which one is the -1 relationship, *StatiCal* provides both functions.

So, for example:

this key sequence	gives this result
Num, 4, Math, x!	x! 24
Num, 5, Math, Gam	Gam 24
Num, -2.5, Math, Gam	Gam -0.9453

The Ln Gam key: Natural Logarithm of the Gamma Function

The **Ln Gam** key is provided for the same reason that the **Ln!** key is provided – to be able to evaluate expressions involving gamma of very large numbers, in the logarithmic domain.

So, for example:

this key sequence	gives this result
Num, 1E50, Math, Gam	Gam Infinity
Num, 1E50, Math, Ln Gam	Ln Gam 1.14129…E52

The +- key: Reversing the Sign

The **+–** key reverses the sign of the number at the top of the stack – a positive number becomes negative, and a negative number becomes positive.

So, for example:

this key sequence	gives this result
Num, 25, Enter, Math, +-	+- -25
Num, -25, Enter, Math, +-	+- 25

The x|y key: Modulo (remainder after division)

The simple definition of the modulo function **x|y** (spoken as “x modulo y” is “the remainder when x is divided by y.

So, 13 modulo 5 is 3, because 13 / 5 = 2, with a remainder of 3.

Similarly, 5.4 modulo 1.38 is 1.26

It’s a little tricky when one or both of the two numbers are negative:

-13 modulo 5 is -3, as you might have expected.

13 modulo -5 is 3, not -3, as you might or might not have expected.

and -13 modulo -5 is -3, as you might or might not have expected.

Anything modulo 0 is NaN.

The 1/x key: Reciprocal of a number

The **1/x** key simply replaces the number at the top of the stack by its reciprocal.

The reciprocal of 0 is Infinity.

The reciprocal of Infinity is 0.

The reciprocal of –Infinity is -0, which is, for all practical purposes, treated the same as 0..

The reciprocal of NaN is NaN.

So, for example:

this key sequence	gives this result
Num, 5, Math, 1/x	1/x 0.2

The pi key: Entering the value of pi

The **pi** key pushes the precise value of pi (3.141592653589793) onto the stack. The other numbers on the stack push up to accommodate the new value.

So, for example:

this key sequence	gives this result
Math, pi	pi 3.1416

The Exp key: Exponentiating

The **Exp** key raises e to the power of the number at the top of the stack. (e is the number 2.71828…)

Exp of 0 is 1.

Exp of 1 is e (or 2.718281828459045)

Exp of Infinity is Infinity

Exp of –Infinity is 0.

Exp of NaN is NaN.

So, for example:

this key sequence	gives this result

The Ln key: Natural Logarithms

The **Ln** key replaces the number at the top of the stack with its natural (base e) logarithm.

The Ln of Infinity is Infinity.

The Ln of 0 is –Infinity.

The Ln of any negative number is NaN (*StatiCal* does not deal with complex numbers).

So, for example:

this key sequence	gives this result
Num, 100, Math, Ln	Ln 4.606

The Log key: Common Logarithms

The **Log** key replaces the number at the top of the stack with its common (base 10) logarithm.

So, for example:

this key sequence	gives this result
Num, 100, Math, Log	Log 2.0000

The Trigonometric Function keys (Sin, Cos, Tan)

The three trigonometric keys can perform 18 different kinds of calculations, depending on how three "toggles" have been set prior to tapping the trigonometric key.

In the simplest case (the way *StatiCal* starts up, before you do any special toggling), these three keys cause the number at the top of the stack to be replaced with the trigonometric sine, cosine, or tangent of the number. The number is considered to be the angle, expressed in radians.

If you want the cotangent, secant or cosecant, you have to remember your basic trigonometry – these are just the reciprocals of the tangent, cosine, or sine, respectively. So just press the 1/x key if necessary:

- For cotangent, press **Tan**, then **1/x**
- for secant, press **Cos**, then **1/x**
- for cosecant, press **Sin**, then **1/x**

The Deg/Rad toggle key: Degrees or Radians

Now things get interesting. What if you have the angles in degrees, rather than radians? That is, what if you want to calculate the Sin of 30 degrees?

Notice that there's a **Deg Rad** button on the top row with "Rad" in tiny yellow letters above it. This is a "toggle" (one of several that *StatiCal* uses). The yellow **Rad** means that the argument to the trig function will be interpreted as radians. If you press the **Deg Rad** button once, the yellow **Rad** turns into a yellow **Deg**, which means that from now on, *StatiCal* will be evaluating the trig functions for angles in degrees.

The **Deg Rad** toggle flips back and forth every time you press that key, and stays the same until you press it again.

So, for example:

this key sequence	gives this result	(comment)
Num, 30, Math, Deg Rad, Sin	SinD 0.5	Tapping the Deg Rad toggle once will change the toggle from its default Radians setting to the Degrees setting
Num 45, Math, Tan	TanD	The Deg Rad toggle will remain at Degrees until you toggle it to something else.

The Inv modifier key: Inverse Trigonometric Functions

Now, what about getting the inverse trigonometric functions? What if you want to know the angle whose sine is 0.5?

The inverse trigonometric functions give you the angle whose sine (or cosine or tangent) is the value you have specified. There are several conventions for how the inverse functions are named when speaking or writing them in a programming language:

The are usually spoken as “inverse sine”, or “arcsine”, and written as “InvSin”, ArcSin, or Asin, or Sin^{-1}, among several other ways. That last way (Sin^{-1}) can be confusing, because a superscript -1 usually means “raised to the power of -1”, which means the reciprocal, and the arcsine is definitely **not** the reciprocal of the sine (the cosecant is the reciprocal of the sine).

So, in common algebraic usage:

$Sin^{-1}(x)$ means the arcsine of x; but

$Sin(x)^{-1}$ means the reciprocal of the sine of x (that is, the cosecant of x); and

$Sin(x^{-1})$ means the sine of the reciprocal of x.

Confusing? Well fortunately, when using *StatiCal*, you don’t have to worry about this. I don’t use the $^{-1}$ notation; I use the **Inv** notation.

Here’s where the **Inv** key comes into play. The **Inv** key might appear to be a toggle key, just like the **Deg Rad** key, and indeed if you press **Inv** repeatedly, the yellow **Inv** indicator above it will go on and off. This is convenient in case you accidentally hit **Inv** when you didn’t want to – just hit it again to turn it off. But it’s important to note that while the **Deg Rad** toggle stays as you set it, until you change it, the **Inv** key is a “one-time” thing – as soon as you evaluate the inverse trig function, the **Inv** indicator goes off, and the next time you want an inverse trig function, you have to hit **Inv** again just prior to the trig function key. So I prefer to call the **Inv** key a “modifier”, rather than a “toggle”. In other words:

toggles are “persistent”, but modifiers are “one-time”

So, for example:

this key sequence	gives this result	(comment)
Num, 0.5, Math, Inv, Sin	ASin 0.5236	if the Deg/Rad toggle is Rad
Num, 0.5, Math, Deg Rad, Inv, Sin	ASinD 30	if the Deg/Rad toggle is Deg

The Circ Hyp modifier key: Circular or Hyperbolic functions

You might remember from high school that the ordinary trig functions we know and love so well are sometimes called the "circular" trig functions, and that there are also a set of strange beasts called the "hyperbolic" functions that have similar names as the circular functions.

StatiCal can calculate the hyperbolic functions by means of the **Circ Hyp** modifier key. If you press this key before pressing the **Sin**, **Cos**, or **Tan** key, you'll get the hyperbolic sine, cosine, or tangent function, respectively.

Like the **Inv** key, the **Circ Hyp** key is a modifier key, not a toggle key. That is, it's a "one-time" key – as soon as you calculate a hyperbolic function, the yellow **Hyp** indicator changes back to **Circ**. You have to hit **Hyp** every time you want a hyperbolic function.

You can get the inverse hyperbolic functions by pressing both the **Inv** modifier and the **Hyp** modifier (in either order) before pressing the **Sin**, **Cos**, or **Tan** key.

I'm not aware of hyperbolic functions being used with degree arguments, so when you use the **Hyp** key, the current status of the **Deg Rad** toggle is ignored. If people actually do use the hyperbolic functions with degree arguments, let me know and I'll change *StatiCal*'s behavior accordingly.

So, for example:

this key sequence	gives this result
Num, 1, Math, Hyp, Cos	CosH 1.5431

The PDFs Panel – Probability and Distribution Functions

StatiCal, as described so far in this manual, is (in my humble opinion) a fairly complete and respectable scientific calculator. But here's where it pulls ahead of the others (at least if you're a statistician).

You can switch to the **Probability Functions** panel at any time by tapping the blue **PDFs** key in the left column of any panel.

StatiCal can evaluate the most commonly-used statistical probability functions and their inverses:

- the **normal** (Gaussian) distribution (Norm);
- the **Chi-square** distribution (Chi Sq);
- the **Student t** distribution (Stud t);
- the **Fisher F** distribution (Fish F);
- the **Binomial** distribution (Bin); and
- the **Poisson** distribution (Pois).

It can evaluate the number of combinations and permutations of n things taken r at a time.

It can apply the transformations of mean and standard deviation between normal and log-normal distributions.

It can evaluate the arc-sine transformation for proportions, and the Fisher z-transformation for correlation coefficients.

All of these functions are invoked using keys in the PDFs panel:

The Rnge toggle Key: Setting the Range of Integration (tails of distribution)

Statisticians generally want probability *integrals*, which are integrals of the probability density functions over some range. *StatiCal* offers an easy way to select several commonly-used ranges for the probability integrals. You enter a single number (call it x) onto the stack, and then you tap the **Rnge** key repeatedly to toggle the yellow range indicator (right above the **Rnge** key) between four sets of integration limits.

For **symmetric** continuous distributions (like the Normal or Student t, the range toggles are defined this way:

Outer: the sum of the areas in the two symmetric tails of the distribution, that is: the integral from –infinity to –x, plus the integral from +x to +infinity;

Inner: the area between the two limits for a symmetric distribution, that is: the integral from –x to +x;

Left: the integral from –infinity to x;

Right: the integral from x to +infinity.

For **non-symmetric** continuous distributions (like the Chi-square or Fisher F), it doesn't make much sense to define **Outer** and **Inner** the way we did it above for symmetric distributions. For one thing, these functions take only positive values of x, so integrals from –infinity to –x are meaningless. Also, for most practical statistical work, the left tail is usually not very interesting – significance tests almost always utilize only the right tail area. So, at the present time, *StatiCal* defines the actions of the range toggles this way:

Outer: the integral from x to +infinity

Inner: the integral from 0 to x

Left: the integral from 0 to x

Right: the integral from x to +infinity

Note that for the non-symmetric distributions, **Outer** is the same as **Right** (either one gives the area of the right tail of the distribution), and **Inner** is the same as **Left** (either one gives the area "in the body" of the distribution (starting from x=0, and excluding the right tail). And I'd point out that Inner + Outer for the same value of x always equals 1.0, and Left + Right for the same value of x also always equals 1.0 .

For the discrete distributions (Binomial and Poisson), the "probability integrals" aren't really *integrals* in the calculus sense; they're the *sums* of the discrete terms of the distribution, so the range designations are handled in a slightly different way. See the sections in **Bin** and **Pois** for an explanation of how the **Rnge** toggle is interpreted for these functions.

The Inv Key: Inverse of Probability Functions, and Confidence Intervals

The **PDFs** panel has an **Inv** modifier key, just like the **Math** panel has, but in the **PDFs** panel it has two different interpretations:

For continuous PDFs, it refers to the inverse of a continuous distribution; that is, an **Inv** probability function calculates the value of the integration limit for which the integral equals the value that has been entered onto the stack.

The range of the integral is determined from the **Rnge** toggle in the same way as for the probability integrals.

So, for example:

this key sequence	gives this result	(comment)
Num, 0.05, PDFs, Inv, Norm	InvNorm 1.96	Assuming Rnge is set to Outer

For discrete PDFs, the **Inv** key generates exact confidence intervals (by the method of Clopper & Pearson for the binomial, and by the analogous method of Garwood for the Poisson). See the sections in **Bin** and **Pois** for an explanation of how the **Inv** modifier is interpreted for these functions.

The %CL Key: Setting the Confidence Level for confidence intervals

By default, *StatiCal* will evaluate 95% confidence limits, but you can change this by using the **%CL** key. You place a number on the stack, indicating the confidence level (expressed in percent) that you want, then hit the **%CL** key. The confidence level that you set will remain in effect until you change it, or until *StatiCal* is re-started.

The Norm key: the Normal Distribution

This key gives probability integrals and their inverses for the standard normal (Gaussian) distribution. The value of the calculation replaces the value of the input on the stack.

So, for example:

So, if you start up *StatiCal* in its default configuration, and carry out the following four example in succession (without doing other stuff in between), you get:

this key sequence	gives this result	(comment)
Num, 1.96, PDFs, Nrml	Nrml 0.05	Rnge defaults to **Outer**
Num, 1.96, PDFs, Rnge, Nrml	Nrml 0.95	Single tap on Rnge toggles to **Inner**
Num, 1.96, PDFs, Rnge, Nrml	Nrml 0.975	Another tap on Rnge toggles to **Left**
Num, 1.96, PDFs, Rnge, Nrml	Nrml 0.025	Another tap on Rnge toggles to **Right**

As the above example illustrates, the sum of the **Inner** + **Outer** integrals for any particular value of x always equals 1.0, and the sum of the **Left** + **Right** integrals for any particular value of x always equals 1.0 . Note that “Left” means “left integral”, that is, the area of everything to the left of x. It is not the “left tail” area unless x is given as a negative number.

Also, if you start up *StatiCal* in its default configuration, and carry out the following four example in succession (without doing other stuff in between), you get:

this key sequence	gives this result	(comment)
Num, 0.05, PDFs, Inv, Nrml	InvNrml 1.96	Rnge defaults to **Outer**
Num, 0.95, PDFs, Rnge, Inv Nrml	InvNrml 1.96	Single tap on Rnge toggles to **Inner**

StatiCal computes the normal distribution integrals using a formula developed by Graeme West, based on work by J Hart.

The Chi Sq key: the Chi-square Distribution

This key gives the probability integrals and inverses for the chi-square family of distributions. You must enter the value of chi-square, and also the degrees of freedom, in that order. You can also toggle between the four range indicators, if you want. Then tap the **Chi Sq** key. The values of chi-square and degrees of freedom will be popped off of the stack, and the value of the chi-square integral will replace them on the stack. That is, the two bottom fields in the calculator will be replaced by a single field, and the remaining stack contents will be slid down.

If you want the inverse of the chi-square probability value, you would enter the probability value, then the degrees of freedom, in that order. You can also toggle between the range indicators, if you want. Then tap the **Inv** key and the **Chi Sq** key, and the value of chi-square will replace the values of the probability and degrees of freedom on the stack.

So, for example: [examples to be provided]

this key sequence	gives this result

The Stud t key: the Student t Distribution

This key gives the probability integrals and inverses for the Student t family of distributions. You must enter the value of t, and also the degrees of freedom (n), in that order. You can also toggle between the four range indicators, if you want. Then tap the **Stud t** key. The values of t and n will be popped off of the stack, and the value of the t integral will replace them on the stack. That is, the two bottom fields in the calculator will be replaced by a single field, and the remaining stack contents will be slid down one position to fill the vacant slot.

If you want the inverse of the t probability value, you would enter the probability value, then the degrees of freedom (n), in that order. You can also toggle between the range indicators using the **Rnge** key, if you want. Then tap the **Inv** key and the **Stud t** key, and the value of t will replace the values of the probability and n on the stack.

So, for example:

this key sequence	gives this result

The Fish F key: The Fisher F Distribution

This key gives the probability integrals and inverses for the Fisher F family of distributions. You must enter the value of F, and also the degrees of freedom for the numerator (n1) and denominator (n2), in that order. You can also toggle between the four range indicators, if you want. Then tap the **Fish F** key. The values of F, n1, and n2 will be popped off of the stack, and the value of the F integral will replace them on the stack. That is, the three fields in the calculator will be replaced by a single field, and the remaining stack contents will be slid down two positions to fill the vacant slots.

If you want the inverse of the F probability value, you would enter the probability value, then the values of n1 and n2, in that order. You can also toggle between the range indicators, if you want. Then tap the **Inv** key and the **Fish F** key, and the value of F will replace the values of the probability, n1, and n2 on the stack.

So, for example: [examples to be provided]

this key sequence	gives this result

The Bin key: The Binomial Distribution and Confidence Intervals

The binomial distribution is defined as follows:

[formula to be provided]

The Bin key evaluates the sum of one or more terms of the binomial distribution. Before you tap **Bin**, you must enter values for N and x, and either one or two additional numbers (depending on how the **Rnge** toggle has been set) that indicate the range of x values in the sum you want.

The **Rnge** toggle works this way:

Outer: You enter two numbers after N and x (call the two numbers Lo and Hi). The sum will include terms from 0 to Lo (inclusive) and from Hi to N (inclusive).

Inner: You enter two numbers after N and x (call the two numbers Lo and Hi). The sum will include terms from Lo to Hi (inclusive).

Left: You enter one number after N and x (call the number Hi). The sum will include terms from 0 to Hi (inclusive).

Right: You enter one number after N and x (call the number Lo). The sum will include terms from Lo to N (inclusive).

So, for example: [examples to be provided]

this key sequence	gives this result

The **Inv** key modifies the action of the **Bin** key, but in a different way from how **Inv** modifies the four continuous probability functions. Strictly speaking, **Inv** doesn't produce the "inverse" of the binomial distribution. That is, it doesn't tell you what range of terms in the discrete binomial distribution sum up to a particular probability. Rather, it returns exact Clopper-Pearson confidence limits around an observed proportion, defined as x / N. The confidence level is determined by the value shown in yellow above the **%CL** key.

So, for example, if you flip a coin 10 times and get 7 heads (N=10, x=7), your observed proportion of heads is 0.7 (P=0.7). Then (assuming that the **%CL** has been set to show 95% CL):

this key sequence	gives this result
Num, 10, Enter, 7, PDFs, Inv, Bin	Hi CL 0.9333 Prop 0.7000 Lo CL 0.3475

To get confidence limits for other confidence levels, you would set the **%CL** value at some time before tapping the **Bin** key:

this key sequence	gives this result
Num, 10, Enter, 7, Enter, 90, PDFs, %CL, Inv, Bin	Hi CL 0.9127 Prop 0.7000 Lo CL 0.3934

The Pois key: The Poisson Distribution and Confidence Intervals

The Poisson distribution is defined as follows:

[formula to be provided]

The **Pois** key evaluates the sum of one or more terms of the Poisson distribution. Before you tap **Pois**, you must enter a value for M, and either one or two additional numbers (depending on how the **Rnge** toggle has been set) that indicate the range of x values in the sum you want.

The **Rnge** toggles work this way:

Outer: You enter two numbers after M (call the two numbers Lo and Hi). The sum will include terms from 0 to Lo (inclusive) and from Hi to N (inclusive).

Inner: You enter two numbers after M (call the two numbers Lo and Hi). The sum will include terms from Lo to Hi (inclusive).

Left: You enter one number after M (call the number Hi). The sum will include terms from 0 to Hi (inclusive).

Right: You enter one number after M (call the number Lo). The sum will include terms from Lo to N (inclusive).

So, for example: [examples to be provided]

this key sequence	gives this result

The **Inv** key modifies the action of the **Pois** key, but in a different way from how **Inv** modifies the four continuous probability functions. Strictly speaking, **Inv** doesn't produce the "inverse" of the Poisson distribution. That is, it doesn't tell you what range of terms in the discrete Poisson distribution sum up to a particular probability. Rather, it returns exact Garfield confidence limits around an event count, defined as N. The confidence level is determined by the value shown in yellow above the **%CL** key.

So, for example, if you observe the occurrence of 10 events (N=10) per unit of time (or space), your best guess of the true event rate is 10. Then (assuming that the **%CL** has been set to show 95% CL):

this key sequence	gives this result
Num, 10, PDFs, Inv, Pois	Hi CL 18.3904 Prop 10.0000 Lo CL 4.7954

To get confidence limits for other confidence levels, you would set the **%CL** value at some time before tapping the **Pois** key:

this key sequence	gives this result
Num, 10, Enter, 90, PDFs, %CL, Inv, Pois	Hi CL 16.9622 Prop 10.0000 Lo CL 5.4254

The nCr key: Combinations (binomial coefficient)

The **nCr** key gives the number of **combinations** of n objects taken r at a time (the number of different ways of drawing a sample of r objects from a collection of n distinct objects, where the sequence in which the objects are drawn **doesn't** matter).

The definition of nCr is: n! / (r! * (n – r)!)

nCr is sometimes read aloud as "from n choose r".

nCr is also referred to as the “binomial coefficient”.

For certain combinations of n and r, the factorials appearing in the above expression might exceed the device’s floating-point limits, but the final value of nCr might not be excessively large (due to partial cancellation of the large numerator and denominator). *StatiCal* uses logarithms when necessary to obtain nCr values for these extreme cases.

So, for example:

this key sequence	gives this result
Num, 20, Enter, 6, PDFs, nCr	20C6 38,760
Num, 1000, Enter, 957, PDFs, nCr	1000C957 6.622295559…E75

The nPr key: Permutations

The **nPr** key gives the number of **permutations** of n objects taken r at a time (the number of different ways of drawing a sample of r objects from a collection of n distinct objects, where the sequence in which the objects are drawn **does** matter).

The definition of nPr is: n! / (n – r)!

For certain combinations of n and r, the factorials appearing in the above expression might exceed the device’s floating-point limits, but the final value of nPr might not be excessively large (due to partial cancellation of the large numerator and denominator). *StatiCal* uses logarithms when necessary to obtain nPr values for these extreme cases.

So, for example:

this key sequence	gives this result
Num, 20, Enter, 6, PDFs, nPr	20C6 27,907,200
Num, 1000, Enter, 957, PDFs, nPr	1000P957 Infinity
Num, 100, Enter, 95, PDFs, nPr	100P95 7.77718…E155

Note that permutations tend to be a lot bigger than combinations! (in fact, r! times larger).

The LgN/Nrm and key: Transforming between Normal and Log-Normal Distributions

Many things in nature tend to have distributions that are very strongly right-skewed. The log-normal distribution very often describes such data nicely. Though the numbers may be very skewed, their logarithms are often nicely normally distributed.

Mathematically:

- if a set of numbers, X_i, are log-normally distributed,
  then their logarithms, $x_i = \mathrm{Ln}(X_i)$, are normally distributed.

And conversely:

- if a set of numbers, x_i are normally distributed,
  then their exponentials, $X_i = \text{Exp}(x_i)$ are log-normally distributed.

If we denote the mean and standard deviation of the X_i as M and S, and the mean and standard deviation of the x_i as m and s. For infinitely large populations, there are exact relationships between M,S and m,s:[formulas to be provided]

	Log-Normal to Normal	**Normal to Log-Normal**
Mean	m =	M =
Standard Deviation	s =	S =

The **LgN Norm** key transforms the M and S of a lognormal distribution into the m and s of the normal distribution that would result from log-transforming the numbers. (You don't actually work with the X_i or x_i numbers; you need only the mean and standard deviation.)

When preceded by the **Inv** toggle, the **LgN Norm** key will work in the opposite direction – transforming the m and s of a normally-distributed set of numbers into the M and S of the log-normal distribution that would result from exponentiating each of the x_i numbers.

Regardless of which direction you want to go, you would first enter the mean onto the stack, then the standard deviation.

So, for example:

this key sequence	**gives this result**
Num, 10, Enter, 8, PDFs, LgN Norm	m Ln 2.0552 sd Ln 0.7033
Num, 2, Enter, 0.7, PDFs, Inv, LgN Norm	m L-N 9.4404 sd L-N 7.5069

So if an infinitely large collection of log-normally distributed numbers had a mean of 10 and a SD of 8, then if you calculated the natural log of each number, those logs would be normally distributed with a mean of 2.0552, and a SD of 0.7033 .

And if an infinitely large collection of normally distributed numbers had a mean of 2 and a SD of 0.7, then if you exponentiated each number (raised e to the power of each number), those numbers would be log-normally distributed with a mean of 9.4404, and a SD of 7.5069 .

(An indication that you might be dealing with log-normally distributed data is that (1) the numbers represent something that cannot be zero or negative, and (2) the SD of the numbers is roughly as large as the mean of the numbers.)

The Arc Sine Key: The ArcSine transformation for Proportions

Proportions can arise from binomial sampling; that is, where you divide the number of occurrences of some kind event by the number of opportunities for that event to occur, these proportions tend not to be normally distributed. For one thing, they cannot fall outside of the bounds of 0.0 to 1.0.

The **Arc Sine** key transforms observed proportions into more normally distributed quantities, so that they can be used analyses that assume normal distributions. You would enter the observed proportion (as a fraction between 0 and 1) onto the stack, then go to the PDFs panel and tap the **Arc Sine** key.

The formula for the arc-sine transformation (which should actually be called the arc-sine-square-root transformation) is:

$$Y = \text{ArcSine}(\text{Sqrt}(p))$$

The transformed values will range from 0 (if the probability was 0) to pi/2 (if the probability was 1).

This transformation also tends to make the standard deviations of the samples more well-behaved (it's called a "variance-stabilizing" transformation). The transformed values will have a SD of 1 / (2 * Sqrt(n)), where n is the denominator of the fraction that produced the proportion.

When preceded by the **Inv** toggle, the **Arc Sine** key will perform the inverse transformation – turning a number between 0 and pi/2 into a probability between 0 and 1:

$$p = (\text{Sin}(y))^2$$

So, for example:

this key sequence	gives this result
Num, 0.5, PDFs, Arc Sin	ArcSine 0.7854
Num, 1.5, PDFs, Inv, Arc Sine	Prop 0.9950

The Fish z key: The Fisher z transformation for correlation coefficients

Correlation coefficients do not have a normally distributed sampling distribution. For one thing, they cannot have values outside of the range -1 to +1). But R.A. Fisher discovered a transformation (which he called the "z-transformation") that turned them into more nearly normally distributed numbers:

$$Z = 0.5 * \text{Ln}((1 + r) / (1 – r))$$

The transformed z numbers can lie anywhere between –infinity (when r = -1) and +infinity (when r = +1).

This transformation also has the interesting property that the standard deviation of the z numbers will always tend to be 1 /sqrt(N – 3), where N is the number of data points (x,y pairs) from which the r value was calculated. So this is another variance-stabilizing transformation.

So, for example:

this key sequence	gives this result	
Num, 0.6, PDFs, Fish Z	FishZ	0.6931
Num, -0.99, PDFs, Fish Z	FishZ	-2.6467
Num, 5, PDFs, Inv, Fish Z	r	0.9999092

[Need to finish this section.]

The Tests Panel – Statistical Tests

StatiCal's final *tour-de-force* is its ability to carry out several of the most commonly-used statistical hypothesis tests on summarized data (counts, means and SDs). It can also perform several tests (t-tests, ANOVAs, correlation tests and linear regression) on reasonably small sets of raw data (individual values). These are done using the keys on the Tests panel:

You can instantly switch to the **Statistical Tests** panel at any time by tapping the blue **Test** key in the left column of any panel.

The NaN key: Entering the "Not-a-Number" flag

We've already been introduced to the NaN (Not a Number) value, which (up to now) has been like an "error message" that appears when we ask *StatiCal* to do something it cannot do, like take the square root of a negative number.

But now NaN is actually going to serve a useful purpose, and we even have a key that lets us deliberately enter this **NaN** value onto the stack! Why would we ever want to do this?

Well, one of the things *StatiCal* can do is summarize a set of numbers. We can enter, say, 25 numbers onto the stack and ask *StatiCal* to summarize them (as count, mean, SD). This may be useful in itself, but is also very useful for doing subsequent analysis on these numbers (like doing a Student t test or ANOVA on two or more such sets of numbers). We can enter up to 100 numbers onto the stack (that's its current capacity), and we need to tell *StatiCal* how far down the

stack to look for the numbers. You might have other numbers sitting on the stack (perhaps from earlier calculations), which you want to save, and you don't want these other numbers to be included in the descriptive summarization.

So for that reason, there's a **NaN** key in the **Num** keypad to push this special flag onto the stack. We can hit **NaN** before we start entering our actual data values, so it will serve as a marker.

What if you've painstaking entered 50 numbers on the stack, and then realize, to your horror, that you forgot to hit the NaN key first? Don't panic, just read the .

The %CL key: Setting the Confidence Level

Several of the keys in the **Tests** panel calculate confidence limits of one type or another. By default, these will be 95% confidence intervals, but you can set any confidence level you'd like by entering that number onto the stack (as a percentage, not as a fraction) and then tapping the **%CL** key.

So, for example, if you'd like 90% confidence limits, you'd enter 90 from the **Num** keypad, then go to the **Tests** keypad and tap **%CL**. You'll see the indicator **90% CL** in small yellow type above the **%CL** button. This new confidence level will stay in effect until you change it, or until you terminate the running of *StatiCal*.

The 2x2 a key: p values for a 2-by-2 cross-tab table

Cross-tabs (contingency tables) are a very common way to summarize the relationship between two categorical variables. These tables can be tested for a significant association between the two variables, using a Chi-square or Fisher Exact test. Also, an astounding number of things can be calculated from the four numbers in the table, depending on what the row and column variables represent.

From the point of view of significance testing, it doesn't matter which variable is tabulated in rows and which is tabulated as columns, nor does it matter which category is the top row or the left column. But when calculating parameters, like relative risk, from the table, it **does** matter.

So we'll establish a convention that when entering cell counts from a 2x2 table into *StatiCal*, we'll enter them in the order: left-top, right-top, left-bottom, then right-bottom.

When testing for causality, we'll establish the following cell-count naming conventions:

	Outcome Occurs	**Outcome Does Not Occur**	**Total**
Causative Factor Present	a	b	a + b
Causative Factor Absent	c	d	c + d
Total	a + c	b + d	a + b + c + d

When using *StatiCal* to evaluate the significance of a 2x2 cross-tab, you must enter the cell counts for cells **a**, **b**, **c**, and **d**, *in that order*. Then you tap the **2x2 a** key to calculate the p-values.

For example: *Is there a significant association between smoking and emphysema?* If we were to study 60 people, we might get the following results:

	Developed Emphysema	**No Emphysema**	**Total**
History of Smoking	**22**	**10**	32
No Prior Smoking	**12**	**16**	28
Total	34	26	60

(These are **totally made-up numbers**, strictly for the sake of an example).

Entering these four cell-counts (we don't enter the row-totals or column totals), in the appropriate order, gives us this:

this key sequence	**gives this result**
Num, 22, Enter, 10, Enter, 12, Enter, 16, Test, 2x2 a	ChiSq 0.0435 Yates 0.0787 Fisher 0.0673

The simple, uncorrected Chi-square test gives a significant p-value (0.0435), for 2x2 tables, but this is not the best test for a 2x2 cross-tab. We should use the Yates-corrected chi-square test instead, which gives a non-significant p-value (0.0787). This may be disappointing, but it's closer to the exact p-value provided by the Fisher Exact test (0.0673).

In addition to providing these three p-values, the **2x2 a** key also does something "behind the scenes" that is very important. It save certain other computational results in a special "hidden" area (not on the stack, or in the M1 or M2 memory cells). These results are used by the **2x2 b** key to calculate the values and confidence intervals for various quantities derived from the 2x2 table (like odds ratios, kappa, etc.). See the following section for a description of all the things you can calculate from a 2x2 cross-tab.

The 2x2 b key: Other Quantities from a 2-by-2 cross-tab table

As mentioned in the previous section, an astounding number of things can be calculated from the four numbers in a 2x2 cross-tab table, depending on what the row and column variables represent:

- risk factors for unfavorable outcomes (odds ratio, relative risk, difference in proportions, absolute and relative reduction in risk, number needed to treat)
- measures of the effectiveness of a diagnostic criterion for some condition (sensitivity, specificity, pos & neg predictive values, pos & neg likelihood ratios, diagnostic and error odds ratios)
- measures of inter-rater reliability (% correct or consistent, mis-classification rate, kappa, Forbes' NMI)
- other measures of association (contingency coefficient, Cramer's phi coefficient, Yule's Q)

First, the four cell counts (a, b, c, and d) from the following table layout have to be entered into *StatiCal*, :

	Outcome Positive	**Outcome Negative**	**Total**
Causative Factor Present	a	b	a + b
Causative Factor Absent	c	d	c + d
Total	a + c	b + d	a + b + c + d

and the **2x2 a** button has to be tapped to produce the three p-values. Then any of the following parameters and their confidence intervals can be calculated by tapping the **2x2 b** key:

- Odds Ratio (OR) = (a/b)/(c/d)
- Relative Risk (RR) = (a/r1)/(c/r2)
- Kappa
- Overall Fraction Correct = (a+d)/t ; (often referred to simply as "Accuracy")
- Mis-classification Rate, = 1 - Overall Fraction Correct
- Sensitivity = a/c1; (use exact Binomial confidence intervals instead of these)
- Specificity = d/c2; (use exact Binomial confidence intervals instead of these)
- Positive Predictive Value (PPV) = a/r1
- Negative Predictive Value (NPV) = d/r2
- Difference in Proportions (DP) = a/r1 - c/r2
- Absolute Risk Reduction (ARR) = c/r2 - a/r1; which = - DP
- Number Needed to Treat (NNT) = 1 / absolute value of DP or ARR
- Relative Risk Reduction (RRR) = ARR/(c/r2)
- Positive Likelihood Ratio (+LR) = Sensitivity / (1 - Specificity)
- Negative Likelihood Ratio (-LR) = (1 - Sensitivity) / Specificity
- Diagnostic Odds Ratio = (Sensitivity/(1-Sensitivity))/((1-Specificity)/Specificity)
- Error Odds Ratio = (Sensitivity/(1-Sensitivity))/(Specificity/(1-Specificity))
- Youden's J = Sensitivity + Specificity - 1
- Number Needed to Diagnose (NND) = 1 / (Sensitivity + Specificity -1) = 1 / Youden's J
- Forbes' NMI Index
- Contingency Coefficient
- Adjusted Contingency Coefficient
- Phi Coefficient (= Cramer's Phi, and = Cohen's w Index, for 2x2 table)
- Yule's Q = (a*d-b*c)/(a*d+b*c) = (OR - 1) / (OR + 1)
- Equitable Threat Score = (a-e)/(a+b+c-e), where e = r1*c1/t
- Entropy H(r) = - ((r1/t)log2(r1/t) + (r2/t)log2(r2/t))
- Entropy H(c) = - ((c1/t)log2(c1/t) + (c2/t)log2(c2/t))
- Entropy H(r,c) = - ((a/t)log2(a/t) + (b/t)log2(b/t) + (c/t)log2(c/t) + (d/t)log2(d/t))
- Information shared by descriptors r and c:
  B = H(r) + H(c) - H(r,c)
- A = H(r,c) - H(r)
- C = H(r,c) - H(c)
- Similarity of descriptors r and c: S(r,c) = B / (A + B + C)
- Distance between r and c: D(r,c) = (A + C) / (A + B + C)
  (Don’t confuse these A, B, and C numbers with the a, b, and c cell counts; they’re quite different!)

In the smoking / emphysema example of the previous section, we might be interested in the relative risk, or perhaps the odds ratio, for smoking considered as a risk factor for emphysema. We would first enter our cell counts, then tap **2x2 a** (to calculate the p-values and, more importantly, the special info needed for the subsequent calculation of the various parameter values and their confidence intervals), then tap the **2x2 b** key. The program will bring up a list of all the things it can calculate from a 2x2 table. Tap the entry you want (for this example, say, the Relative Risk), and it will display that parameter and its confidence limits. Then tap **2x2 b** again, and this time select the Odds Ratio. Here's what you can expect:

this key sequence	gives this result
Num, 22, Enter, 10, Enter, 12, Enter, 16, Test, 2x2 a	ChiSq 0.0435 Yates 0.0787 Fisher 0.0673
2x2 b, then select Relative Risk from the popup list	Rel Rsk 1.604 Hi CL 2.719 Lo CL 0.956
2x2 b, then select Odds Ratio from the popup list	Odds Ratio 2.933 Hi CL 9.766 Lo CL 0.901

Some things to note about invoking the **2x2 b** calculaton:

1. You must run **2x2 a** on your data first. You cannot simply enter the four cell counts and then tap **2x2 b**. Remember: "**a**, then **b**".
2. You can run as many **2x2 b** taps as you want after you've run the **2x2 a**. You don't have to re-enter the cell counts for the **2x2 b** taps. Each time you run **2x2 b,** you select one of the parameters to be evaluated.

How the method works: Confidence intervals for the estimated parameters are computed by a very general method based on "constant chi-square boundaries", described in: *Statistical Methods for Rates and Proportions* (2nd Ed.) Section 5.6, by Joseph L. Fleiss (Pub: John Wiley & Sons, New York, 1981). This method is also described in *Numerical Recipes in C* (2nd Ed.) Section 15.6, by William H. Press et al. (Pub: Cambridge University Press, Cambridge UK, 1992).

This method calculates two "limiting tables" that represent how far away from your observed table, on either side, the "truth" could be, and still have your observations not be too unlikely (with "too unlikely" being quantified by the confidence level you want). Then, when you plug the numbers from these limiting tables into the formula for whatever parameter you want, they yield the confidence limits for that parameter.

You can get the contents of those two limiting tables by selecting the last entry in the popup list: **Limiting Tables**. You can use these tables if you're interested in some other quantity that's not in the list. Just get the tables and manually plug them into the formula for the parameter you're interested in.

The MxN Xtab key: Chi-square analysis of any size cross-tab table

The Fisher Exact test gets incredibly difficult for cross-tab tables larger than 2x2, and I won't attempt to perform *that* test on those tables. But cross-tab tables of *any* size can be analyzed quickly and easily by the Chi-square test.

To use the **MxN Xtab** key, you first enter the cell counts of the table. Enter them in "row order"; that is, enter all the numbers in the first row, then all the numbers in the second row, etc. Don't enter them column-wise, or you'll get the wrong answer.

After entering all the cell counts, you have to enter **two more numbers**, to tell *StatiCal* **how many rows** and **how many columns** the table has.

So, for example, Suppose you want to perform a chi-square test on the following 2-by-3 table, showing the dessert preferences of a sample of males and females:

	Pie	**Cake**	**Jello**
Male	15	20	23
Female	13	12	5

You would do it this way:

this key sequence	**gives this result**
Num, 15, Enter, 20, Enter, 23, Enter, 13, Enter, 12, Enter, 5, Enter, 2, Enter, 3, Test, MxN Xtab	Chi Sq 5.3456 df 2.0000 p Val 0.0690

So this table indicates that there is no significant association between the gender and the dessert preference (p = 0.069).

The Ord Xtab key: Analysis of a cross-tab of ordinal categories

The Chi-square and Fisher Exact tests assume that the rows and columns represent *nominal* categorical variables; that is, categories whose values don't fall into any natural, logical sequence, like the following:

- Cauc, AfrAm, Asian, Other
- Pie, Cake, Jello

The sequence is unimportant, and switching the rows around, or switching the columns around doesn't affect the p-value from either the Chi-square or Fisher Exact test.

But many categorical variables are *ordinal*; that is, they **do** fall into a logical sequence:

- Neonate, Infant, Child, Adolescent, Adult, Geriatric
- Extra Small, Small, Medium, Large, Extra Large
- Emaciated, Normal, Overweight, Obese, Morbidly Obese
- Absent, Slight, Moderate, Loaded
- Strongly Disagree, Disagree, No Opinion, Agree, Strongly Agree
- Cured, Improved, No Change, Worse, Died

The Chi-square and Fisher Exact tests are, by their very nature, not "on the lookout" for gradual, consistent trends across a range of ordinal categories, and so are not particularly efficient at detecting such trends. They deal very well with *nominal* categories, but not with *ordinal* categories.

Special tests have been designed for analyzing cross-tabulations of *ordinal* categorical variables.

(By the way, you can consider any dichotomous categorical variable, like Gender, to be ordinal.)

So let's take a look at a different example that happens to have the same cell-count numbers as the example of the previous section (on **MxN Xtab**), but representing something entirely different. Suppose the columns didn't represent dessert preferences, but rather the attitude of the subject toward "action adventure" movies, like "Cowboys vs. Aliens". The cross-tab would now look like this, with the same numbers, but different column labels:

	Dislike	**Neutral**	**Like**
Male	15	20	23
Female	13	12	5

Notice that the numbers in the top row steadily increase from left to right, while the numbers in the bottom row steadily decrease from left to right. (Males tend to lean more toward action-adventure movies than women do.) This gradual trend across columns was irrelevant in the previous example, because the sequencing of the columns was totally arbitrary. But now, with an ordinal column variable, this "steady trend" means something, and we want an analysis that is on the lookout for such a trend and can take advantage of it.

You would test this ordinal cross-tab this way:

this key sequence	**gives this result**
Num, 15, Enter, 20, Enter, 26, Enter, 13, Enter, 12, Enter, 5, Enter, 2, Enter, 3, Test, Ord Xtab	Conc 355.0000 Disc 835.0000 p Val 0.0241

So this table indicates a significant association between the gender and preference for action-adventure movies (p=0.0241). The regular Chi-square test (previous section) couldn't detect the gradual trends in the two rows, because the Chi-square test doesn't know left from right from middle columns (or top from bottom from middle rows). But the ordinal test is on the lookout for these progressive "directional" trends.

The Dscr Stats key: Basic Descriptive Statistics of a set of Numbers

The most common way to concisely summarize a set of numbers is by quoting the count, mean, and standard deviation (SD) of the numbers.

The **Descr** key calculates these three summary measures from a set of numbers that you've entered onto the stack. *StatiCal*'s stack currently can hold up to 100 numbers; if your sample is larger than that, use some other program (like Excel) to summarize them.

Before entering your numbers, you must first place a special flag, called "NaN" (for "not a number") onto the stack. The **Num** keypad has a **NaN** key for just this purpose.

So, for example, to calculate summary statistics on the six numbers: 18, 22, 20, 28, 19, and 30, you would do the following:

this key sequence	gives this result
Num, NaN, 18, Enter, 22, Enter, 20, Enter, 28, Enter, 19, Enter, 30, Enter, Test, Descr Stats	Mean 23.4000 SD 5.3666 N 5.0000

While this is nice in itself, the main utility of the **Descr Stats** key is as a precursor to using the **tTest** or **AoV** key. These keys want to see summary data (mean, SD, N) on the stack for each group. So if you have case-level data, you would use the **Descr Stats** key to summarize them first, for each group, then tap one of the the **tTest** (1Grp or 2Grp) keys or the **AoV** key.

The tTest keys: Student t tests

There are several Student t tests:

- one for a single set of numbers,
  testing whether the mean of those numbers is significantly different from zero
  (this is called the "single group" t-test).
- one for paired samples,
  testing whether the mean of the paired differences is significantly different from zero
  (this is called the "dependent samples" t test);
- one for unpaired samples, testing whether the difference in the two means is significantly different from zero (this is called the "independent samples" t-test);

The first two types are really equivalent, because the first step in doing a dependent-samples t-test is to calculate the paired differences, and then test them with the single-group t-test.

The last type (for independent samples) has two variants:

- one that assumes that the two samples come from populations with the same SD (this is called the "equal-variance" t-test, or the "classic" Student t-test);
- one that does not assume that the two population SDs are equal
  (this is called the "unequal-variance" or "equal variances not assumed" or "Welch" t-test).

StatiCal has two keys that can handle all of these different types of t-tests.

The **1Grp tTest** key performs a single-sample t-test. This key wants to see the mean, SD, and N on the stack, in that order. If you have individual values, you can use the **Descr Stats** key to calculate mean, SD and N, and it will put those three numbers onto the stack in the right order. Then you tap the **1Grp tTest** key to perform the test.

The **1Grp tTest** key is also used for the paired t test; you just set up the data differently. The mean and SD placed on the stack must be the mean and SD of the paired differences, and the N must be the number of pairs.

So, if you had pairs of numbers (x1, y1, x2, y2, x3, y3, etc), you would do the following:

- first tap the NaN key (to mark the start of your data),
- then enter x1, then tap Enter, then enter y1, then tap x-y (which will replace x1 and y1 by the difference),
- then do the same thing for x2 and y2, then for x3 and y3, etc.
- After entering all your paired differences this way, you'd tap Descr Stats to calculate the mean, SD and N values and put them on the stack,
- and finally tap the 1Grp tTest key to perform the test.

The **2Grp tTest** key performs two unpaired t-tests (equal-variance and unequal-variance). This key wants to see two sets of summary statistics on the stack (mean1, SD1, N1, mean2, SD2, N2), in that order. If you have individual observations for one or both groups, you can use **Descr Stats** to calculate the summary statistics for these groups and enter them onto the stack for you.

The **2Grp tTest** key places six numbers on the stack:

- the t value, degrees of freedom, and p-value for the unequal-variance (Welch) t test; and
- the t value, degrees of freedom, and p-value for the equal-variance (classic) t test.

The AoV key: 1-way Analysis of Variance

You can think of the 1-way Analysis of Variance (simply called AoV here) as an extension of the standard "unpaired" equal-variance Student t test to the situation where there are **3 or more** groups being compared. The AoV also works when there are only 2 groups, in which case it gives **exactly** the same p-value as the unpaired, equal-variance Student t test. So you can think of the t test as a special case of the AoV, for only 2 groups being compared.

Because the AoV is really an extension of the unpaired t-test, it wants to see, for each group, the summary statistics (mean, standard deviation, and count) for each group on the stack. This has to be followed by another entry on the stack telling it how many groups of summary statistics you placed on the stack: (m1, sd1, n1, m2, sd2, n2, …, mG, sdG, nG, G), in that order. Then you can tap the **AoV** key to perform the analysis of variance. This will place four numbers on the stack: the F value, the numerator and denominator degrees of freedom, and the p-value.

As with the unpaired t-test, if you have individual observations for one or more groups, you can use the **Descr Stats** key for these groups to calculate the summary statistics and enter them onto the stack for you.

The Regr key: Straight-line regression and correlation analysis of {x,y} data

- The **Regr** key performs a linear (straight-line) least-squares regression and correlation analysis on a set of x,y data.

Data is entered as pairs of numbers, representing the x and y values for each data point. You must first enter a **NaN** flag to mark the start of the data on the stack. Then you must place the paired data onto the stack, in the order: x1, y1, x2, y2, x3, y3, etc. Finally, you tap the **Regr** key to perform the analysis.

The **Regr** key calculates the following results and places them on the stack:

- the number of data points (number of {x,y} pairs of values);
- the slope of the fitted line;
- the intercept of the fitted line;
- the RMS Error, which is, loosely speaking, the average scatter of the points from the fitted line (actually, the standard deviation of the vertical distances of the points from the fitted line);
- the square of the Pearson correlation coefficient (r2);
- the Pearson correlation coefficient (r);

the p-value indicating the significance of the regression (also indicating the significance of the slope of the line).(“root-mean-square”) error of the fit, .

So, for example to calculate the correlation coefficient between the height and weight of four subjects:

Height (Inches)	Weight (Lbs)
69	170
62	134
72	210
64	180

you would do this:

this key sequence	gives this result
Num, NaN, 69, Enter, 170, Enter, 62, Enter, 134, Enter, 72, Enter, 210, Enter, 64, Enter, 180, Test, Regr	n 4.000 Slope 5.633 Intcp -202.534 RMS Err 21.858 r-sq 0.676 r 0.822 p Val 0.178

This means that the straight line: Weight = 5.633*Height – 202.534
fits the data, with an overall vertical scatter of about ± 22 lbs.

Note that although the correlation was quite high (r=0.822), it is not significantly different from zero (p=0.178), because of the small sample size

The r:0 key: Test the significance of a correlation coefficient

This key tests whether an observed correlation coefficient (r), based on N observations, is significantly different from zero. It also calculates the confidence interval around the observed r.

The test is based on the fact that the quantity:

$$t = \frac{r}{\sqrt{\left(1 - r^2\right) / \left(N - 2\right)}}$$

is distributed as Student t, with N-2 degrees of freedom.

To perform this test, go to the **Num** panel and enter r and N onto the stack, in that order, then switch to the **Test** panel and tap the **r:0** key.

So, for example, to test whether an r of 0.5 based on a sample of 30 observations is significantly different from 0:

this key sequence	gives this result
Num, 0.5, Enter, 30, Test, r:0	p Val 0.0049 Hi CL 0.7290 Lo CL 0.1704

So an r value of 0.5, based on 30 observations, is highly significantly different from 0 (with a 2-tailed p-value of 0.0049). The 95% confidence intervals around the observed r value of 0.5 goes from 0.1704 to 0.729. This could be reported as: r = 0.5, n=30, CI=0.17 to 0.73, p=0.005 .

The r:r key: Compare Two Correlation Coefficients

This key tests whether two observed correlation coefficient (r1 and r2), based on n1 and n2 observations, respectively, are significantly different from each other. It also calculates confidence limits around the difference between r1 and r2.

The test is based on the fact that Fisher Z transform (see that section of the manual) is approximately normally distributed, with a SD of 1 / Sqrt(n – 3), so the difference between two r values, r_1 and r_2, based on n_1 and n_2 observations, respectively, will be normally distributed with a mean of $r_1 – r_2$, and a SD of Sqrt($1/(n_1 – 3) + 1/(n_2 – 3)$).

To perform this test, go to the **Num** panel and enter r_1, n_1, r_2, and n_2 onto the stack, in that order, then switch to the **Test** panel and tap the **r:r** key.

So, for example, to test whether an r of 0.4 based on a sample of 25 observations is significantly different from an r of 0.6 based on a sample of 45 observations:

this key sequence	gives this result
Num, 0.4, Enter, 25, Enter, 0.6, Enter, 45, Test, r:r	p Val 0.3058 Hi CL 0.6558 Lo CL -0.2415

So an r value of 0.4, based on 25 observations, is not significantly different from an r value of 0.6 based on 45 observations (p = 0.306). The difference between the two r values is 0.20, with a 95% confidence intervals that goes from -0.24 to +0.66 .

The Tol Intvs key: Tolerance Intervals for Normally-distributed data

Loosely speaking, a tolerance interval for a measured quantity is the interval in which there is some "likelihood" (or, of which you feel a some "level of confidence") that a specified fraction of the population's values lie, based on a sample that you measured from this population.

Tolerance intervals have been widely used in statistical process control. The **Tol Intvs** key will calculate tolerance intervals for any specified population fraction, and for any specified level of confidence, from the mean and standard deviation of a finite sample, under the assumption that the population is normally distributed. One-sided (upper and lower) intervals, as well as the two-sided interval, are calculated.

Reference: *NIST/Sematech Engineering Statistics Handbook*, Section 7.2.6.3
http://www.itl.nist.gov/div898/handbook/prc/section2/prc263.htm

The example below is taken from one of the examples in this Handbook.

Note: The very last line on the Handbook page:
"The upper (one-sided) tolerance limit is therefore 97.07 + 1.8752*2.68 = 102.096."
is wrong. The standard deviation in their example is 0.0268, not 2.68, so the answer should be 97.12, not 102.096 .

The calculation of normal-based tolerance intervals requires the input of five numbers: The count, mean and SD of the original sample, the desired population fraction, and the “level of certainty” (which is put into the **%CL** field).

So, using the example in the Sematech Handbook, if I measure a sample consisting of 43 items, and get a mean value of 97.07, and a standard deviation of 0.0268, then I can be 99% certain that 90% of the population lies:

- **within** the interval from 97.0106 to 97.1294 (a 2-sided tolerance interval), or
- **below** the value 97.1203 (an upper 1-sided tolerance limit), or
- **above** the value 97.0197 (a lower 1-sided tolerance limit).

Here’s how you’d determine those tolerance limits with *StatiCal*:

this key sequence	gives this result
Num, 97.07, Enter, 0.0268, Enter, 43, Enter, 90, Enter, 99, Test, %CL, Tol Intvs then select 1-sided from the popup menu	H1 TL 97.1203 L1 TL 97.0197 H2 TL 97.1294 L2 TL 97.0106

Note that **Tol Intvls** puts **four** result numbers on the stack – the two 1-sided limits, and the pair of 2-sided limits. Only the bottom three of them will show (L1, H2, and L2), but you can slide the view window up to see the H1 value by tapping on the L1 field label.

Like many of the tests on this key panel, **Tol Intvls** can work with summary data (mean, standard deviation, and observation count) or with individual observed values (x1, x2, x3, …, xn). If have individual observations, you can enter these numbers onto the stack and then use the **Descr Stats** key to calculate the mean, sd, and n. The **Descr Stats** key will place these three summary values on the stack, in the correct order, for the **Tol Intvls** calculation; then all you have to add is the population fraction and the level of certainty (if it's different from what's showing above the **%CL** key).

By the way, there is an online web-based tolerance intervals calculator at:
http://StatPages.org/tolintvl.html

The Conf Intvs key: Confidence Intervals for Normal Data

For normally-distributed data, confidence limits are generally calculated as a value, plus and minus the standard error of that value multiplied by a number which is the critical value of Student t for the appropriate confidence level and degrees of freedom.

The simplest case is for a symmetric 2-sided 95% confidence interval around the observed mean (M) of a sample of N numbers with a standard deviation of SD:

$$\text{Conf. Limits} = M \pm t_{Crit} * SD/\text{sqrt}(N)$$

where t_{Crit} is the critical 2-tailed Student t value for p=0.05, with N-1 degrees of freedom.

The quantity SD/sqrt(N) is equal to the standard error of the mean.

So, for example, a sample of 10 numbers with a mean of 5 and a SD of 2 would have a symmetrical 2-sided 95% confidence interval going from 3.5693 to 6.4307 . This is because the critical 2-sided Student t value for p=0.05 and 9 degrees of freedom is 2.2622, and so we have:

$$\text{Conf. Limits} = 5 \pm 2.2622 * 2 / \text{sqrt}(10) = 5 \pm 1.4307 = 3.5693 \text{ to } 6.4307$$

This can be set up in *StatiCal* as follows:

this key sequence	gives this result
Num, 5, Enter, 2, Enter, 10, Enter, Test, Conf Intvs {From the popup dialog, select "Mean, SD, n (Desc)"}	Value 5.0000 Hi CL 6.4307 Lo CL 3.5693

Confidence intervals can arise in other contexts as well. In particular, regression programs usually produce a table that shows, for each variable appearing in the regression model, the value of the regression coefficient (or "parameter"), its standard error (SE), and perhaps a degrees-of-freedom (df) for that standard error. If the program does not show confidence limits for the coefficient, you would have to calculate them from this formula:

$$\text{Conf Limits} = \text{Parameter} \pm t_{Crit} * SE$$

If the regression table does not show the degrees of freedom, it can usually be calculated as the difference between the number of data points and the number of parameters appearing in the model (the "df = points minus parameters" rule). For example, a straight line has two adjustable

parameters (a slope and an intercept), so the SE for the slope and the SE for the intercept each have degrees of freedom equal to the number of data points minus 2.

Sometimes the actual number of data points is not given, so you might just want to assume that it is very large. The critical Student t value then is replaced by the critical normal z value.

There is one more complication. For some kinds of regression (like logistic, Poisson, or Cox Proportional Hazards (CPH) regression) the most useful results are obtained by raising e to the power of the regression coefficients. Exponentiating logistic regression coefficients gives odds ratios; exponentiating Poisson regression coefficients gives relative event rate ratios; and exponentiating CPH regression coefficients gives hazard ratios.

For these situations, the confidence limits around the regression coefficients would be calculated as described above, then the coefficient, and both confidence limits, would be exponentiated to get the confidence limits around the odds or rate or hazard ratio. For such situations, choose the option that ends in “(Ln)”. .

For example:

this key sequence	gives this result

The Bay Cred key: Bayesian Credibility Analysis from Odds Ratios

The **Bay Cred** key allows the credibility of a clinical trial finding to be assessed in the light of current knowledge. It is based on the groundbreaking work of Robert Matthews:

Matthews, R.A.J. 2001 *Methods for assessing the credibility of clinical trial outcomes*
Drug Information Journal vol 35 (4) 1469-1478
Available online at: ttp://bit.ly/CredibilityAnalysisPaper

Many clinical trials produce "statistically significant" evidence for the efficacy of some new therapy. However, statistical significance is a notoriously poor indicator of the credibility of a finding - that is, the extent to which it provides convincing evidence for efficacy - as it fails to take full account of the size of the trial, or of pre-existing insights.

Bayesian credibility analysis takes the outcome of a given clinical trial - as summarized by the published Odds Ratio (OR) together with its 95 % confidence interval (CI) - and uses a Bayesian method to determine the credibility of the finding in the light of what is already known. Specifically, it shows whether the new finding, when combined with existing knowledge, can be taken to have credibly demonstrated efficacy at the 95% confidence level. Such a finding is said to be credible at the 95% confidence level. (You can use any confidence level you prefer.)

For a given result to meet this standard, the prior evidence for efficacy must exceed a specific level; this level is captured by the Critical Odds Ratio (COR). The assessment of a given clinical trial result can then proceed as follows:

- If the stated 95% confidence interval excludes an OR of 1.00 (corresponding to no effect), the result can be deemed statistically significant at the 95% level;
- If ORs at least as impressive as that indicated by the COR can be justified by existing knowledge, the result can also be deemed credible at the 95% level

The COR produced by this calculator reflects both the effect-size and the statistical power of the clinical trial under study. Thus trials involving large numbers of patients and/or large effect sizes will typically produce undemanding CORs, which are easily justified on the basis of current knowledge. On the other hand, trials involving relatively few patients, and/or small effect sizes will - because they possess relatively little evidential weight - produce CORs that demand a substantial amount of prior evidence before the new result can be deemed credible.

So, for example:

this key sequence	gives this result

The Stor Keypad – Managing the Calculator's Storage

The fifth and final panel of keys may not be very impressive, but it's quite important. These keys don't do any real calculations (a few of them do adds and subtracts); instead, they deal with the management of *StatiCal*'s various storage locations – the *stack* and the two special *memory cells* (M1 and M2). They can clear these locations out, move numbers between them, juggle numbers around in the stack, and other similar kinds of operations.

You can instantly switch to the **Storage** panel at any time by tapping the blue **Stor** key in the left column of any panel.

The memory cells are useful in several kinds of situations. Sometimes, you may have a number that you know you're going to have to use over and over again during a set of calculations, and you don't want to have to enter it every time (it may have a lot of digits). Or, you may have the result of some calculations that you know you'll have to use at several places in subsequent calculations. It's very convenient to be able to stash this number someplace "safe" (at least safer than on the stack, where there's always a lot of action going on). There are keys on the Stor panel that let you move numbers between the stack and the memory cells. There are also keys to clear out (zero out) the contents of a memory cell. And there are keys that let you use these memory cells as "accumulators" to add (or subtract) a set of numbers.

There are also a few keys that are useful in dealing with the stack itself, like popping numbers off the stack, or swapping the positions of two numbers.

The Clr keys: Set a Memory Cell to Zero

The **M1 Clr** cell sets the M1 memory cell to zero. The stack is not altered.

The **M2 Clr** key does exactly the same thing, but with the M2 memory cell instead of the M1 cell.

The Sto keys: Store a number into a separate Memory Cell

The **M1 Sto** cell pops the most recently added number off of the stack and puts it into the M1 memory cell, replacing anything that might have been in M1 at the time. The number is removed from the stack, and the other numbers on the stack slide down one position to fill the gap.

The **M2 Sto** key does exactly the same thing, but with the M2 memory cell instead of the M1 cell.

The Rec keys: Recall a number from a Memory Cell and put it onto the stack

The **M1 Rec** key takes the number currently sitting in the M1 memory cell, and pushes it onto the stack, pushing the other stack contents up one position. The number also remains intact in the M1 cell, so it can be recalled again and again if you want.

The **M2 Rec** key does exactly the same thing, but with the M2 memory cell instead of the M1 cell.

The M + keys: Add a number from the stack onto a Memory Cell

The **M1+**key takes the most recently added number on the stack, and adds it to the contents of the M1 memory cell, putting the sum into M1, and popping the number off of the stack.

The **M2+** key does exactly the same thing, but with the M2 memory cell instead of the M1 cell.

The M – keys: Add a number from the stack onto a Memory Cell

The **M1-** key takes the most recently added number on the stack, and subtracting it from the contents of the M1 memory cell, putting the new value into M1, and popping the number off of the stack.

The **M2-** key does exactly the same thing, but with the M2 memory cell instead of the M1 cell.

The AC key: Clear out All Values from Stack and Memory Cell

The **AC** key works like the **All Clear** keys on most calculators – everything gets set to zero – all 100 elements of the stack, and both memory cells.

The Pop key: Pop the bottom number from the Stack

The **Pop** key will remove (pop) the most recently entered number from the stack, and slide the other numbers down to fill the gap.

What happens to the popped number? It gets annihilated – it vanishes, and is never seen or heard from again.

You will normally use this only when you no longer want the latest number in the stack.

The Dup key: Duplicate the bottom number on the stack

Pressing the **Dup** key will push a duplicate copy of the most recently added number onto the stack, pushing all the other numbers to make room for it. So if the top of the stack (the bottom of the three visible numbers) is 123.45, then tapping the **Dup** key will duplicate this number

So, for example:

If the Stack looks like this	Tapping Dup gives this result	Then tapping Pop gives this result
Sqrt 123 Exp 456 Enter 789	Exp 456 Enter 789 Dup 789	Sqrt 123 Exp 456 Enter 789

That 123 entry wasn't destroyed by the **Dup**; it was just slid to the 4th position (and beyond the first 3 visible entries). Popping the duplicated 789 causes the remaining entries to slide down one position, so the 123 comes back into view.

The Swap key: Swap the two bottom numbers on the Stack

The **Swap** key reverses the two top entries in the stack (the two bottom fields in the calculator).

Tapping **Swap** again will reverse the reversal, leaving things as they originally were.

So, for example:

If the Stack looks like this	Tapping Swap gives this result	Then tapping Swap again gives this result
Sqrt 123 Exp 456 Enter 789	Sqrt 123 Enter 789 Exp 456	Sqrt 123 Exp 456 Enter 789

Swap is useful if you've just gotten the result of a big calculation (suppose it comes out as the value 5), and you want to do one more operation on it, like, for example, raise 2 to the power of that result. You can enter 2 onto the stack, but then you can't tap the **x^y** key, because that would raise 5 to the power of 2 (that is, 5^2, which is equal to 25), rather than raise 2 to the power of 5 (that is, 2^5, which is equal to 32).

The easiest thing is to enter the 2, then hit the **Swap** button. Then you'll have the 2 and the 5 in the correct positions for doing the **x^y** operation.

The Roll Up and Roll Dn keys: Roll the stack, with wrap-around

The **Roll Up** key rolls the stack up one position. That is, every element in the stack moves up by one position. The number that was in the top of the stack wraps around to the bottom of the stack.

The **Roll Dn** key rolls the stack down one position. That is, every element in the stack moves down by one position. The number that was in the bottom of the stack wraps around to the top of the stack.

These operations roll only the "occupied" elements of the stack. So:

If the stack is empty, nothing is done when you press the **Roll Up** or **Roll Dn** key.

If there was only one number in the stack, nothing is changed (it rolls into its own position).

If there were two numbers in the stack, **Roll Up** and **Roll Dn** both behave exactly like **Swap**. That is, if you entered the two values 11 and 22, in that order, onto an empty stack, then after pressing either **Roll Up** or **Roll Dn**, the stack would have the two values 22 and 21.

If there are three numbers on the stack, those three numbers cycle around. That is, if you enter the values 11, 22, and 33 (in that order) onto an empty stack, then pressing Roll Up would produce 22, 33 and 11, while pressing Roll Dn would produce 33, 11, and 22.

The Roll Up and Roll Dn keys can be a great time-saver if you've entered a lot of numbers on to the stack, planning to use the Descr Stats key, and then realize that you forgot to mark the beginning of the data with a NaN. You don't have to clear out the stack and re-enter everything.

If the stack was empty before you started entering your data, just tap the **NaN** key; then go to the **Stor** panel and tap **Roll Dn** once, and the NaN will wrap around and be at the beginning of the data.

If there were already other numbers on the stack before you entered your data, it's a little more work, but still easier than re-entering everything. Just tap **Roll Dn** enough times to roll your data down until the first pre-existing number (that you don't want in the summary) is in the bottom field on the screen. (That will be one tap for every number you do want in the summary.) Then go to the **Num** panel and tap **NaN**. Then go back to the **Stor** panel and tap **Roll Up** that same number of times, so that your last desired number is in the bottom field. You will have inserted a NaN between the wanted and unwanted data, and you can now tap the **Descr Stats** key.