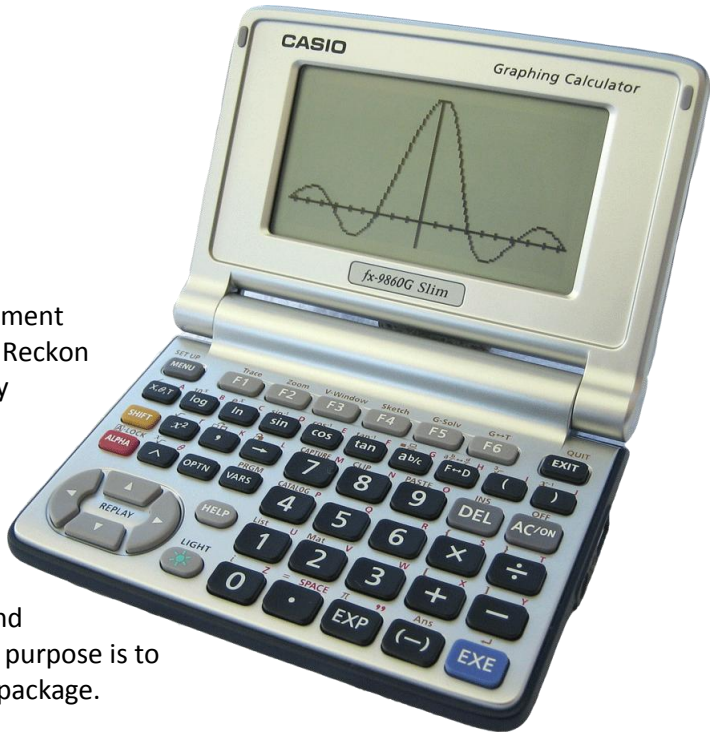
RECKON

V1.16



Introduction

Thank you for trying *Reckon*. Reckon is a replacement calculator for real calculators and small devices. Reckon has been written to be as portable as possible by limiting dependencies on specific hardware and subsystem features.

I have written Reckon to me *my* calculator. Each person wants something slightly different from a calculator and with any real calculator there are missing wanted functions and things which work in an awkward way. Reckon's purpose is to have all the features [i want] in one easy to use package.

Nevertheless, feature requirements change. The plan is to add features as and when they are needed and also to move reckon onto different hardware as it becomes possible. I am also hoping the feature set will be useful for others who might want to use it as their alternative calculator.

I am open to improvement suggestions and bug reports, rather than a huge wish list. I have my own long wish list too! It's likely that features from your wish list are also on mine. Some things are easy to add (like a new function and map it to a key), some are hard (e.g. add a CAS). Easy feature requests are likely to get done.

One of the biggest challenges is getting the right manner of operation. I am trying to avoid modes where possible, deep menus and prefix key sequences. I am hoping to hear good ideas from people that make for ease of use through operation rather than boilerplate modes and menus.

In Reckon, there is no complex mode, but there is an ***i*** key, so this trick is not necessary. Where possible, the functions automatically support complex operation, although currently you can only have real matrices.

A big area missing from the current release is display formatting. for example, the number of digits, ENG, SCI, FIX etc. These are usually modal. I am chewing over ideas for display formatting with the view to expand the concept from number formatting to also include HEX, OCT, BIN display, DMS, DATE calendar display etc.

There are many, many things to go in, several are half finished. Originally, i wanted to make a release only when a significant number of these were done. However, due to time constraints, i have decided to make an early first release because i think the features already implemented can be useful now (I'm already using reckon as my regular calculator) and other people have good ideas that might influence their implementation.

So there are things missing, but i have endeavoured to round up and fix bugs and annoyances from my list rather than chuck in half-finished features. I'm hoping there won't be any embarrassing defects in this version (like getting its sums wrong), but there are likely to be small misfeatures.

Enjoy,

Entry System

ALG and *RPN* entry modes are supported. If the prompt is "**>**" you are in ALG mode, if you have **0** at the bottom of the screen, you are in RPN mode.

In RPN mode, the bottom line is the top of the stack. One per line and the top line is level 8. RPN mode has an 8 level stack. Press **OPTN** to toggle between RPN and ALG mode. [*Todo: entry mode annunciator*]

Where it makes sense, the physical keys as labelled are used for their designated operation. So, for example, *sin* enters **sin(** in ALG mode and performs *sine* on stack X in RPN.

The Reckon expression parser is relaxed about final closing parentheses. For example **sin(1** will be accepted. Also **sin(sin(1** and for matrices **[[1,2][3,4**. This is a convenience feature. However, note that, **sin1** is not $\sin(1)$ but the variable **sin1**. **sin(1+** is not legal.

You can elide multiply in certain cases, immediately after a number and before a variable or function and before parentheses. Examples: **2sin(1**, **2X** and **2(sin(1)+1).** This can be useful in RPN. e.g. **2X** can be entered as part of usual number entry, also terms like **2π**. Pi is bound to the π symbol (**SHIFT EXP**) and also to the **Pi()** function.

The **SHIFT** key works in the usual way; press shift then another key for the shifted function. Do not hold down shift. [*Todo: shift annunciator*]

The **ALPHA** key works in the usual way; press alpha then another key for its alpha character. Press **SHIFT ALPHA** for *alpha lock*, disengage lock by pressing **ALPHA**, **EXE** or **AC**. **SHIFT ALPHA** whilst in alpha lock does not unlock (same as Casio). [*Todo: alpha annunciator. shift + alpha for lower case*]. Pressing **SHIFT** whilst in alpha, either immediately after the **ALPHA** key or in alpha lock allows the entry of lower case letters.

As well as on **SHIFT +**, **X** is mapped onto the **XθT** key. This works as a convenient way to enter the variable **X** without a shift key press.

Hexadecimal and binary integers can be entered by prefixing with **0X** (use the **X** key) and **0b** respectively. This is a bit experimental because only number input is supported, and there is no way to display an answer in hex or binary.

Pre-minus, i.e. **(-)**, is the same as operational minus, **–**, in ALG mode. There is no need for two keys. Note that unary minus binds tighter so **–x ^ 2** is $(-x)^2$ and not $-(x^2)$.

In RPN mode, pre-minus is like **+/–** during number entry. You begin entering a negative number with **(-)** and also for negative exponents. Pressing **–** in RPN always performs subtract, but **(-)** won't change the sign of a number after entry, because it will begin a new number entry.

The change sign function (negate) is bound to the fraction key **Ab/c**. This will **+/–** in RPN or **Neg()** in ALG. Do not use this during RPN number entry, use **(-)** to change sign.

Abs is bound to **SHIFT Ab/c**. Abs performs $|x|$ when *x* is real. For complex *z*, **Abs(z)** is the modulus and for matrix m, **Abs(m)** the determinant.

DEL performs a single character backspace for entry correction. **ALPHA DEL** is also **DEL** (currently).

SHIFT DEL (labelled **INS**) performs *X<>Y SWAP* in RPN.

The *arrow* key, **→**, which means assign-to in Casio, maps to **=** for convenience. In ALG you assign with expression like **A=2** (with arrow or with **=** they are the same). In RPN you can use arrow a bit like *STO*, e.g. 23 **EXE A →** , to store 23 in **A**.

F↔D is mapped to **Flt()** the *to-float* converter. In ALG you can convert a rational to floating point with **F↔D** e.g. **Flt(ans).** In RPN this is especially convenient for converting results (like large integers) to floats; just press **F↔D**. Pressing **F↔D** again or with a float attempts to convert the float back to a fraction.

Factorial, the **!** operator, is not marked on the keyboard. Reckon maps it to **SHIFT 3**.

For convenience **[[** is mapped to **SHIFT 2** (i.e. **MAT**).

i, $\sqrt{-1}$, is mapped to **SHIFT 0** (labelled ***i***). Note, currently complex ***i*** and lower-case **i** are the same character. This might change.

VARS is used by reckon for unit conversions (see section). For convenience **VARS** enters the **#** symbol and automatically engages alpha lock.

When operating in RPN, differences apply to number entry. The pre-minus **(-)** key acts to change the sign of the mantissa and then of the exponent after **EXP** is used.

Additionally, RPN mode supports *free-form* entry. This allows algebraic style syntax to be used to enter a single term within RPN mode. For example entering the formula, $2x^2 - x^4 + 1$ is easier as written than building the expression from RPN parts. Free form entry in RPN is initiated by the keys **(**, **[** or single *quote* (**SHIFT COMMA**).

Once initiated, the reminder of the current input works as if in ALG mode, until **EXE** or **AC** is hit. Furthermore, **DEL** works the same as ALG mode and there is no immediate input syntax checking.

Opening parentheses in RPN is useful, when you want to enter a subexpression that you wish to evaluate on **EXE**. The following example in RPN makes use of an ALG subexpression.

2 EXE 3 + (2 + 3 EXE –

Note that the free-form initiator key must be used at the start of expected RPN input. Square bracket is used as a free-form initiator to allow easy entry of vectors and matrices in RPN. e.g.

[2,3 EXE EXE +

Additionally single quote **'** is accepted as a free-form initiator. This is the quote operator and acts to prevent evaluation. It is useful for entering the name of a variable when that variable is already bound. e.g. as an expression or as argument to **Purge()**.

You can use quote with expressions involving numbers, but they will not be evaluated. e.g.

> '1+2
1+2

Using quote is handy for Plot() in RPN. e.g.

' SIN X EXE 0 EXE 2π PLOT

This gives an alternative to the usual RPN way, **X SIN 0 EXE 2π PLOT**

All trig operations work in radians, there is no DEG/RAD/GRAD mode. So, **sin(90)** finds the *sine* of 90 radians. If you want to enter an angular quantity, postfix the value with the degree symbol, **°**, (bound to **ALPHA ^** labelled θ). e.g. **sin(90°)** will result in $\sin 90\ degrees$.

Note: the *deg* symbol applies DMS, in the format DDD.MMSS to degrees, and then converts degrees to radians. So **sin(90.3045°)** calculates $\sin 90°30"45\ seconds$.

To go the other way say, for example, from *arcsine* back to DMS, use the *angle* operator (**SHIFT XθT** labelled ∠. e.g. **Asin(1)∠** is 90. Again the result is transformed from radians to degrees and then to DMS with the format DDD.MMSS. The same applies in RPN mode, except you will see the conversions happening immediately you press the conversion operations.

Other keys not used (currently):

EXIT, QUIT, CAPTURE, PASTE, PGRM, CATALOG, LIST, CR (SHIFT EXE). SETUP, F1, F2, F3, F6

Variables

Variable names can be more than one letter. Variables must start with a letter and thereafter alphanumeric. So **XX** is not **X*X**, but a different variable.

To assign a variable, use syntax **X=***value*. In RPN this can be thought of as *STO* by entering a variable name (e.g. **X**) then use arrow, →, as STO.

Whenever a variable is bound to a value, the variable will be replaced by the value during evaluation. Variables initially have no value and are unbound. Unbound variables are useful to represent formulae, e.g. as input to **Plot()** or **Solve()**.

It is possible to unbind a variable using **purge()**. The *purge* function is mapped to **SHIFT 8** (labelled **CLIP**). Unfortunately, calling **Purge(X)** when **X** is bound results in **Purge(***value-of-x***)** which is not what you want. To avoid this, you must *quote* **X** with the single quote operator **'X**. quote is bound to **SHIFT COMMA**.

Example:
```
> A=3
3
> A+1
4
> A=2A
6
Purge('A)
A
```

Note that expressions passed into **Solve()** and **Plot()** must involve a single unbound variable. All bound variables are replaced by their values; *Plot* and *Solve*, determine the relevant unbound variable as the intended term to plot against or solve for.

Variables used in vectors are expanded to their values at vector creation. The vector does not *remember* the variable used when creating it.

Example:
```
> A=1
1
> B=2
2
C=[A,B]
[1 2]
D=[C,C+2]
[[1 2]
 [3 4]]
1/D
[[-2 1]
 [1.5 -0.5]]
```

Numbers and Types

Reckon's evaluator performs automatic value type promotion. What that means is that a parameter of one type can be automatically converted to another type in order to continue evaluation.

When you enter a number like **3**, it starts life as an integer. But if you square root it, Reckon promotes the value to floating point in order to apply the square root operation. Before values wind up as floating point, calculations are in exact rationals.

Example:
```
> 69!  (shift 3 key)
171122452428141311372
468338881272839092270
544893520369393648040
```

923257279754140647424
000000000000000

What you get are all the digits of the integer result. How about 100!, this time you get the floating point answer, **9.332621544394415E157**

The latter is because Reckon limits integers from getting too big, when they do, they are automatically converted to floating point. You can always force a number to floating point with **F↔D**. Actually, integers are also rationals, so when you calculate **36/7** you get **3+1/7** rather than a floating point answer. This is also true of perfect squares, eg. **Sqrt(4/9)** gives **2/3**.

When a real number can't go, Reckon promotes it to a complex number. For example **sqrt(-2**) is not real, so a complex result is returned. All complex numbers are complex floating point, there are no complex fractions.

Floating point numbers in Reckon are just like those in other calculators. Reckon's floating point number system is internally stored in decimal (not binary IEEE754). This is also the case with most real world calculators. Binary floating point numbers lead to problems sometimes, for example, 0.1 is finite in decimal, but not in binary and can lead to rounding discrepancies.

Reckon uses 25 digits internally for all its scientific functions, but usually less are actually displayed on screen [*Todo: display style format*].

Reckon supports exponents of 4 digits, going up to 10^{9999}. Also the quantities **Inf** and **–Inf** are available. Note that **1/0** is **Inf**. **NaN**s are undefined results, e.g. **0/0** and 0^0 are **NaN**. These values as intermediate results do not halt evaluation.

Reckon supports some non-numerical types, including strings, e.g. **"hello"**, and array vectors, **[1 2 3]**. Currently curly braces are not used for anything.

Unit Conversions

Reckon uses an unusual method for unit conversions. At first, it might be a bit confusing but is very simple to use. Units are represented by their value in SI units and bound to specific variables.

All of the variables used for unit conversions are special and start with a **#**, otherwise they are same as normal variables. Most of the unit variables are 3 letters or less. To make it easier to enter these variables the **VAR** key enters the **#** sign and automatically engages *alpha lock*. You must disengage alpha lock manually, unless your next key is **EXE** in which case the lock is dropped automatically.

Example: The speed of light is **#C**

Enter **VARS C EXE** (no need to press **ALPHA**). You will see the value, **299792458**

This is the speed of light in m/s. i.e. in SI units.

By way of example, we will convert the speed of light into miles per hour. Miles per hour is the symbol **#MPH**. The rule for unit conversions is:

Multiply by the symbol for what you have and divide by the symbol for what you want.

Example: 3 times the speed of light in miles per hour:

3 VARS C ALPHA / VARS M P H EXE

Or in RPN: **3 VARS C EXE VARS M P H EXE /**

Note: the **ALPHA** is required to release the lock. Unfortunately the divide key otherwise has the alpha **T**. in RPN we can use **EXE** (i.e. enter) to release the lock for us.

Here is a list of the current symbols:

Symbol	Description	Units

#gg	Gravitational constant (big G)	$m^3\ Kg^{-1}\ s^{-2}$
#g	Acceleration due to gravity	m/s^2
#au	Astronomical Unit	m
#in	Inch	m
#ft	Foot	m
#yd	Yard	m
#mi	Mile	m
#nmi	Nautical Mile	m
#mph	Miles per hour	m/s
#c	Speed of light in vacuum	m/s
#yr	Gregorian calendar year	S
#oz	Ounce	Kg
#ozt	Troy ounce	Kg
#lb	Pound	Kg
#me	Electron mass	Kg
#mp	Proton mass	Kg
#mn	Neutron mass	Kg
#earth	Earth mass	Kg
#acre	International acre	m^2
#pt	UK pint	m^3
#foz	UK fluid ounce	m^3
#gal	UK gallon	m^3
#ptus	US pint	m^3
#fozus	US fluid ounce	m^3
#galus	US gallon	m^3
#atm	1 Atmosphere	N/m^2
#k	Bolzmann Constant	J/K
#e	Electron Charge	C
#na	Avagadro Number	mol^{-1}
#R	Gas constant	J/K/mol
#hp	Mechanical horsepower	W

Matrices

Reckon interprets vectors that look like matrices as matrices. The vector **[1 2 3]** is considered as a row, for example, it legal to multiply this row by a 3xN matrix. For matrix multiplication of **A*B**, the number of columns of **A** must match the number of rows of **B**.

The column vector of 1,2,3 looks like this: **[[1][2][3]]** and is very different. You can generate this column by transposing the vector **T([1 2 3])**. For RPN transpose is mapped to **SHIFT ARROW**

Note that currently you must separate elements of vectors or matrices by commas during the input process.

Currently, if Reckon does not see your input as a valid matrix it will treat it as a generic vector. This might cause confusion. For example the 1x1 matrix of **0** is the vector **[0]**, consequently **[[0]]** is not recognised as this object and will not operate arithmetically.

Once an object is recognised as a matrix, Reckon tries to treat it arithmetically as far as possible, If you add or multiply a vector or matrix by a scalar, the effect will be to apply that operation on each element of the vector. e.g. **[[1 2][3 4]] + 1** is **[[2 3][4 5]]**.

If you add vectors, corresponding elements will be added. eg **[1 2 3] + [4 5 6]** is **[5 7 9].** The same applies to matrices, eg **[[1 2][3 4]] + [[4 5][6 7]]** is **[[5 7][9 11]]**.

Multiplication of matrices works according to the usual rules:

[[1 2][3 4]] * [[4 5][6 7]] is

[[16 19]
[36 43]]

Reckon tries to offer division of matrices by defining A/B as B^{-1}*A. i.e., Reckon calculates the inverse of the matrix B and multiplies it in order to give the illusion of division[1].

for example **[[1 2][3 4]]/[[4 5][6 7]]** is

[[4 3]
[-3 -2]]

Other matrix operations including the determinant which is given by **Abs()**, and transpose which is **T()**.

The **Proot()** function finds the roots of a polynomial with real coefficients. For example,

$$5z^6 - 45z^5 + 225z^4 - 425z^3 + 170z^2 + 370z - 500$$

Proot takes a vector of coefficients where element *i* is the coefficient of z^i. *Note* the order of coefficients in the input vector.

eg.

> proot([-500,370,170,-425,225,-45,5])
[3-4i 3+4i 1-1i 1+1i 2 -1]

So the roots are $3 \mp 4i, 1 \pm i, 2\ and\ -1$

Graph Plotting

The **Plot()** function makes a graphical plot of a function. This is bound to the **F4** key. Plot expect three parameters, the function to plot, the min and max of the x-axis.

For example **Plot(Sin(X),0,2π).** will plot the graph of sin(x) for x in [0,2π].

To make things simpler, Reckon examines the expression given as the first parameter to **Plot()** to determine the independent variable; **X** in this example. Plot expects an expression with exactly *one* unbound variable. This variable is the one assumed to be the x-axis.

Example:

[1] Matrices form a ring and not a field, proper division does not really exist!

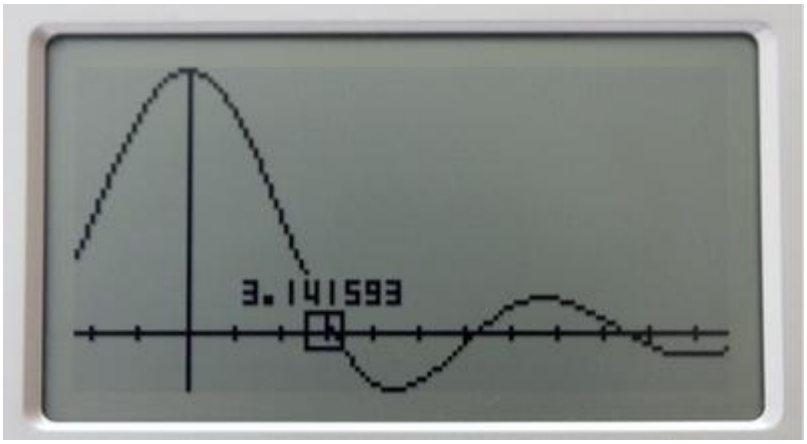
A = 2
B = 3
Plot(B*Sin(A*T),0,5)

Will plot 3Sin(2t) for t in [0,5] assuming T is not bound.

The plot function draw the plot and presents the graphical view. In this mode, the view may be examined before returning to the calculation engine. The 4-way navigator may be used to adjust the view. Left and right scroll the graph horizontally, whilst up and down zoom in and out. Zoom is much harder than scroll and consequently the graph may struggle to keep up. Presently, the new graph will smooth out and the view become accurate.

Example 1:

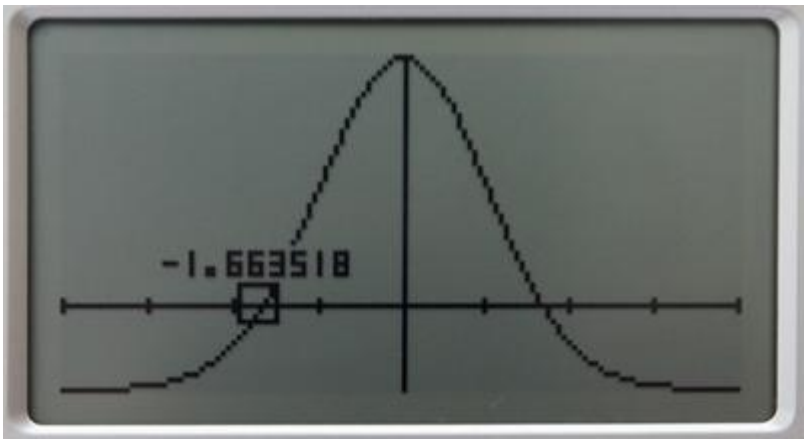
To plot $\frac{Sin(x)}{x}$ enter **Plot(Sin(X)/X,-7,-7).** Then scroll about. Here, the function has been scrolled left a bit. Pressing **EXE** at this point cycles through the *points of interest* (POI). These are the zeros and min/max values. When a box is drawn (like in the picture), this is a zero and the value of X for the root is shown. For min/max, the function value is shown (Y) instead of X (no box is drawn in this case).



Example 2:

Plot $\frac{e^{-(\frac{x^2}{2})}}{\sqrt{2\pi}} - 0.1$, enter **Plot(Exp(-X ^ 2/2)/Sqrt(2π)-0.1,-4,4)**

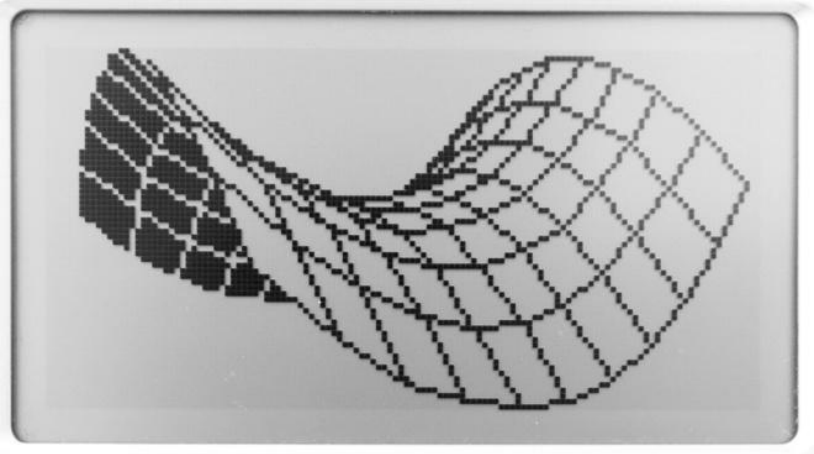
Again we can cycle through the POI with **EXE**. Note, that POI is only available *after* the plot is fully drawn.



It is also possible to plot 3D surfaces, using the **Plot3D** function (**F3**). Two independent variables are required, notionally X and Y. The range of X and Y may be supplied as parameters.

Example:

Plot3D(X ^ 2-Y ^ 2,-2,2,-2,2)



Quite often the range of X and Y is the same, so it is sometime easier to supply just one range, eg:

Plot3D(X ^ 2-Y ^ 2,-2,2)

By default Plot3D chooses a grid of 10x10 samples. It' s possible to change this by adding the sample mesh density as an additional parameter. However, increasing the number does not always produce clearer results. Indeed 16 is about maximum, otherwise there is no memory left! eg.

Plot3D(X ^ 2-Y ^ 2,-2,2,14)

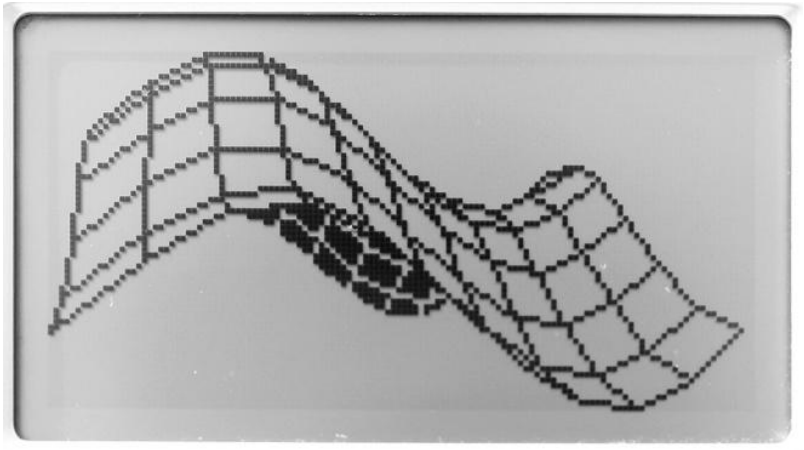
When the 3D surface is displayed, a number of control keys alter the view

Key	*Purpose*
Direction pad	Pan side to side and up and down
+	Zoom In
-	Zoom Out
AC	Quit
8, 5	Rotate parallel to the Z axis centred on view
7, 4	Rotate parallel to the X axis centred on view
9,6	Rotate parallel to the Y axis centred on view
0	Reset the view
1,2,3	Free spin about models local Z axis. Hit 1 to speed up left, 3 for right, 2 to stop spinning
(and)	Expand/Contract X & Y scale
.	Increase detail

Further examples

Plot3D(X ^ 2+Y ^ 2,-2,2)

Plot3D(Sin(x)-Cos(y),0,5)



Plot3D(Sin(X) ^ 2-Cos(Y) ^ 2,0,5)

Function Reference

Function	Description	Key Binding
Pi()	Pi. Also bound to π symbol	**SHIFT EXP**
'	Quote operator	**SHIFT COMMA**
!	Factorial operator, z!	**SHIFT 3**
^	Raise to power operator	
Factorial()	Factorial function	
Exp(z)	e^z	
Ln(z)	Natural log, $Ln(z)$	
Log(z)	Common log (base 10)	
ALog(z)	Common antilog, 10^z	
Sqrt(z)	Square root, $\sqrt{z}$	
Sin(z)	Sine of z radians	
Cos(z)	Cosine of z radians	
Tan(z)	Tangent of z radians	
ASin(z)	Arcsine of z radians	
ACos(z)	Arccosine of z radians	
ATan(z)	Arctangent of z radians	
Sinh(z)	Hyperbolic sine of z	
Cosh(z)	Hyperbolic cosine of z	
Tanh(z)	Hyperbolic tangent of z	
ASinh(z)	Hyperbolic arcsine of z	
ACosh(z)	Hyperbolic arccosine of z	
ATanh(z)	Hyperbolic arctangent of z	
Inv(z)	Invert, 1/x, 1/z, matrix inversion	**SHIFT)**
Neg(z)	-z, or negate vector terms	**Ab/c**
Ln1p(x)	$ln(1 + x)$, x real	
Exp1m(x)	$e^x - 1$, x real	
Flt(x)	Convert to float if possible	**F↔D**
Fac(n)	Find a factor of integer n. NB: Can take a long time (minutes). hold down DEL to abort. Either returns a factor of n or 1 if prime.	
Np(n)	Find the next prime, P > n	
Pp(n)	Find the previous prime, P < n (or 0 if none)	
T(m)	Transpose vector/matrix m	**SHIFT ARROW**
Abs(t)	$\|t\|$, complex modulus, matrix determinant.	**SHIFT Ab/c**
Sq(z)	Square function, z^2	**X2**
Cuberoot(z)	Cube root function, $\sqrt[3]{z}$	**SHIFT (**
Purge(v)	Purge variable v. use quote.	**SHIFT 8**
Atan2(x,y)	Arctan2 function, x & y real	

Nroot(z,n)	Nth root, $\sqrt[n]{z}$	**SHIFT ^**
DmsRad(x)	Convert x as DMS to Deg and then to RAD	**ALPHA ^**
RadDms(x)	Convert x from radians to Deg, then DMS	**SHIFT XθT**
SWAP	RPN only, swap stack X & T	**SHIFT DEL (INS)**
Plot()	Plot(expr, x-min, x-max)	**F4**
Plot3D()	Plot3D(expr, xy-min, xy-max) Plot *expr* with the range given for both X and Y independent variables Plot3D(expr, x-min, x-max, y-min, y-max)	**F3**
Solve()	Solve(expr, x-min, x-max)	**F5**
Proot(v)	Real and complex roots of polynomial whose coefficients are in vector **v**.	
Rat(x,eps)	Finds a rational approximation p/q to **x** within **eps**. |p/q - x| <= |eps|.	
Prime(n)	Determine whether n is prime (NB: slow)	
Mod(X,N)	Find X mod N	
Conj(z)	Conjugate complex number	
MJD(n)	Modified Julian day number for date yyyymmdd	
Date(n)	Date as integer yyyymmdd from modified julian day number	
Price(settlement, maturity, rate, yield, redemption, frequency)	Bond price eg. `price(20080215,20171115,5.75,6.5,100,2)` gives `94.635449207...` NB: same parameters as Excel.	
Mduration(settlement, maturity, coupon, yield, call, frequency)	Modified Macauley Duration for Bonds eg `mduration(20080101,20160101,8,9,100,2)` gives, `5.7356698139..` NB: same as Excel, except call parameter added.	

Features Planned

Many features are missing from this release. Major regrets are:

- Numerical integration.
- Polynomial CAS and Taylor series[2].
- Helpful mode and operational display annunciators.
- Built-in help.
- Various small, but specific functions
- Programming language.

[2] CAS features are not planned in general, except for polynomials.

Version Updates

The latest version of Reckon will be posted to http://www.voidware.com/reckon/

Please check for updates.