Professional Electronics
for Automotive and Motorsport

6 Repton Close | Basildon
Essex | SS13 1LE | United Kingdom
+44 (0) 1268 904124
info@liferacing.com
www.liferacing.com

Maths Functions
User Manual

Document revision:
V2.0

Document Release Date:
2019-04-30

Document Author:
MH

Contents

1 INTRODUCTION 2
2 MATH LANGUAGE VERSION 2
3 WRITING AN EXPRESSION 3
3.1 OPERATORS 4
3.2 DEFINITIONS AND REFERENCES 4
3.3 NUMERICAL ANALYSIS 5
3.4 TRIGONOMETRY 6
3.5 MATHMATICAL OPERATORS 7
3.6 SESSION TIMING 8
3.7 DATA ANALYSIS 9
3.8 FILTERS 11
4 MATHS REPORTS 15
4.1 REPORT FORMAT 15
4.2 STATISTICAL FUNCTIONS 16
4.3 EXAMPLE 17
5 DOCUMENT REVISION HISTORY 18

© Life Racing Ltd

1 Introduction

Math can be used to manipulate monitoring items on several Life Racing applications. It can be used to process downloaded track data in LifeView or used on live data with LifeMon. Life Racing displays can also function in a similar way to LifeMon when math items are created in LifeDash (Dash math functions are currently limited). There is a large number of mathematical functions, capable of a wide range of tasks, resulting in a flexible and powerful tool.

2 Math Language Version

From LifeView v2.15.166 the default maths language has changed. For backwards compatibility, where a maths language version is not specified, v1 is used. To specify a language version, begin an expression with “MathLanguageVersion(x,y);”

Language version 2 is required for maths reports, expression blocks and changes how IF statements are evaluated. Previously both THEN and ELSE clauses were evaluated at each sample whereas in v2, only the winning side is evaluated. An IF statement also no longer requires an ELSE clause where no action is to be taken. Maths blocks can be used in any instance where a single expression is expected. For example:

```
A = if(rpm>6000,1+2,5+6);
B = if(rpm>6000,3+4,7+8);
```

Can now become:

```
If(rpm>6000),{A = 1+2; B = 3+4;},{A = 5+6; B = 7+8;});
```

Or the more easily read:

```
If(rpm>6000),
{
  A = 1+2;
  B = 3+4;
},
{
  A = 5+6;
  B = 7+8;
});
```

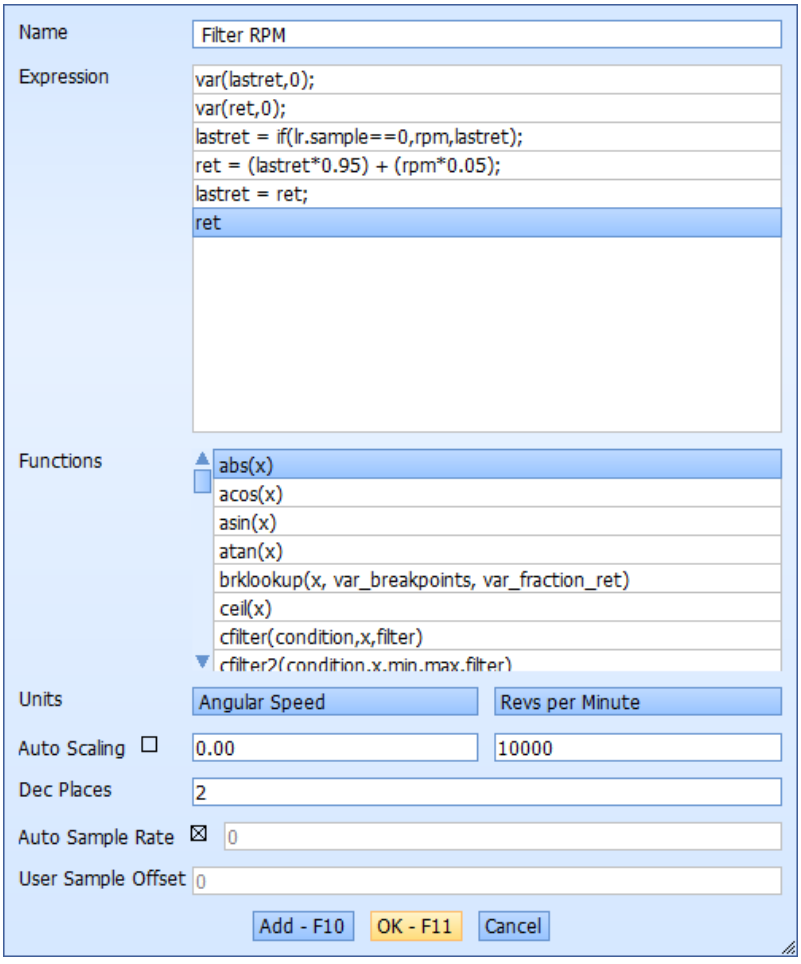
As well as adding flexibility, because the condition is only evaluated once and only one outcome is executed, this version results in a faster and more efficient expression.

© Life Racing Ltd

3 Writing an Expression

Math expressions can vary in complexity from simple multiplications to multidimensional lookup tables. Care must be taken to correctly write expressions to avoid errors, particularly as they become more complicated. Expressions can consist of several lines and several functions. A new line will continue from the previous line. To mark the end of a particular function, use a semi-colon (;). The final function does not require this as this will be taken as the final value.
Any text following a double forward slash (//) is considered a comment and is not used in the expression. This is required for each line of comment. e.g.

//maths channel for calculating brake balance
*bpf/(bpf+bpr)*100 //percentage*



All useable functions (to date of document) are described in this document in the following format:

Name
Description
Example

3.1 Operators

Several shorthand standard operators can be used within expressions and include the following:

(	Open Bracket	*	Multiply	&&	Logical AND	=	Assignment
)	Close Bracket	/	Divide	||	Logical OR	*=	Multiplication assignment
[	Open Array	%	Modulus (remainder)	!	Logical NOT	/=	Division assignment
]	Close Array	+	Add	&	Bitwise AND	%=	Modulus assignment
{	Open Block	-	Subtract	|	Bitwise OR	+=	Addition assignment
}	Close Block	<	Less than	^	Bitwise XOR	-=	Subtraction assignment
,	Comma	<=	Less than or equal to	~	Bitwise NOT	&=	Bitwise AND assignment
“”	String	>	More than	<<	Bitwise Left Shift	|=	Bitwise OR assignment
?	Then	>=	More than or equal to	>>	Bitwise Right Shift	^=	Bitwise XOR assignment
:	Else	==	Equal to			<<=	Bitwise Left Shift assignment
pi	π, 3.14159265…	!=	Not equal to			>>=	Bitwise Right Shift assignment

3.2 Definitions and References

lr. Functions (LifeView only)
lr. functions are fixed variables that return values for:

Current sample number.
lr.sample
Time into session in seconds.
lr.time
The time taken for current sample.
lr.sampleSize

const(name,value)
Define a constant and its initial value.
const(a,3)

var(name,value)
Declare variable and initial constant value.
var(x,0)
Multi-dimensional variables allow entire tables of data to be defined. When recalling, note that entry numbers begin at 0. See brklookup for how multiple dimensional arrays can be used effectively.

```
var(x[5],1,2,3,4,5);
x[3]            =4
var(x[5,2],
0,1,2,3,4,
5,6,7,8,9);
x[3,1]          =8
```

undefined()
Allows blank data.
undefined() *=###*

isdefined(x)
Returns 1 if x is defined. Returns 0 if x is set as undefined or there is a gap in the data. Useful for flagging sensor errors.
lam1(x)

math("math fn name")
Use result from other math function.
math(acceleration)/9.81

previous(initial_const_value)
Returns the previous expression result. Initial value must be specified.
previous(0)

previous(initial_const_value,sample)
Returns the previous nth result from the expression.
previous(0,5)

previousx(x,initial_const_value)
Returns the previous value of channel x.
previous(rpm,0)

previousx(x,initial_const_value,sample)
Returns the previous nth value of channel x.
previous(rpm,0,10)

itemvalueat(monitem, absolute time)
Returns monitoring item value at the specified time. Time is defined in seconds.
itemvalueat(rpm,500)

3.3 Numerical Analysis

abs(x)
Absolute value (magnitude) of x. Useful for inverting negative values.
abs(-10) *=10*

sign(x)
Sign value of x. returns 1 or -1 depending on sign.
sign(ignbase1)

round(x)
Round to nearest integer.
round(1.2) *=1.0*
round(1.8) *=2.0*

© Life Racing Ltd

floor(x)
Round down to nearest integer.
floor(1.2) *=1.0*
floor(1.8) *=1.0*

ceil(x)
Round up to nearest integer.
ceil(1.2) *=2.0*
ceil(1.8) *=2.0*

fmod(x,y)
Remainder of x/y.
fmod(22,6) *=4*

min(x,y,...)
Returns the lowest value of bracketed numbers or items.
min(tps1A,tps1B)

max(x,y,...)
Returns the highest value of bracketed numbers or items.
max(tps1A,tps1B)

duration(x)
Counts the duration in seconds that x has been true. Resets to zero when x is not true.
duration(rpm>6000)

3.4 Trigonometry

sin(x)
Sine of x where x is in radians.
sin(pi) *=0*

cos(x)
Cosine of x where x is in radians.
cos(pi) *=-1*

tan(x)
Tangent of x where x is in radians.
tan(pi) *=0*

asin(x)
Arcsine of x (reverse sine) in radians range -pi/2 – pi/2.
asin(1) *=pi/2*

© Life Racing Ltd

acos(x)
Arccosine of x (reverse cosine) in radians range 0 – pi.
acos(1) *=0*

atan(x)
Arctangent of x (reverse tangent) in radians range -pi/2 – pi/2.
atan(1) *=pi/4*

3.5 Mathmatical Operators

exp(x)
Exponential value of x, e^x.
exp(2) *=e^2=7.39*

log(x)
Log of x with base 10.
log(500) *=2.7*

pow(x,y)
Raise x to the power of y.
pow(2,3) *=8*

sqrt(x)
Square root of x.
sqrt(9) *=3*

derivative(x)
Differentiate x by time. Only function to use interpolated values by default.
derivative(vehicleSpeed) *=acceleration*

integral(x)
Integrate x by time.
integral(vehicleSpeed) *=distance*

© Life Racing Ltd

integral(x,condition)
Integrate x when conditions are true. Resets to 0 when condition is not met.
integral(vehicleSpeed,laptime()>0) =lap distance

The integral function is useful for counting durations as the integral of 1 will return a running clock in seconds. This can be used differently to the duration function as it has an independent reset condition.

e.g.
```
var(vResetCond,0);
var(vMeasCond,0);
var(vTimeOn,0);
var(vTimeTotal,0);

vMeasCond=tps1>95;
vResetCond=lapTime<0.1;

vTimeOn=integral(vMeasCond,!vResetCond);
vTimeTotal=integral(1,!vResetCond);

vTimeOn/vTimeTotal
```

interpolatedvalue(x)
Use interpolated items in expression rather than last sample as is default.
interpolatedvalue(rpm)

if(condition,then,else)
Classic if statement. Maths language version 2 does not require an “else” clause.
if(rpm>200,1,0)

3.6 Session Timing

Note that these items are calculated in LifeView and may not correspond to the logged ECU items. e.g. lapTime is a logged item from the ECU, laptime( ) is a calculated value.

time( )
Current log time in seconds.
time()

ctime( )
Comparative time of current lap to fastest lap. Requires distance channel.
ctime()

ctime(lap)
Comparative time of current lap to a defined other in the current session. Requires distance channel.
ctime(3)

© Life Racing Ltd

ctime(session,lap)
Comparative lap time to another in specified session when more than one file is loaded. Sessions start at 0. Requires distance channel.
ctime(1,3)

laptime( )
Time into current lap.
laptime()

laptime(session)
Time into current lap in specified session when more than one file is loaded.
laptime(3)

laptime(session,lap)
Total duration of specific lap time.
laptime(1,3)

fastestlap(session)
Identity of fastest lap in specified session. Sessions start at 0.
fastestlap(1)

fastestlapsession( )
Identity of session with fastest lap.
fastestlapsession()

getlap( )
Returns current lap number.
getlap()

Can be used with *laptime( )* to get complete lap time for current lap:
laptime(0,getlap( ))

getbeacontimes(lap, var_starttime, var_endtime)
Sets two variables as lap start and end time.
var(x,0);
var(y,0);
getbeacontimes(2,x,y)

3.7 Data Analysis

extrapolate(channel, x1,y1, x2,y2 ...)
Find an estimated value of y from x (channel) from xy pairs. x values higher or lower than the list will continue the trend at that point.
extrapolate(1.3,0,0,1,2,2,4,3,6,4,8)==2.6
extrapolate(5,0,0,1,2,2,4,3,6,4,8)==10

© Life Racing Ltd

interpolate(channel, x1,y1, x2,y2, ...)
Find an estimated value of y for 'channel' from xy pairs. Item values higher or lower than the list are cut.
interpolate(1.3,0,0,1,2,2,4,3,6,4,8)==2.6
interpolate(5,0,0,1,2,2,4,3,6,4,8)==8

As functions are not ended on a line but by the symbol **;** arrays can be written across several lines for clarity:

```
interpolate(1.3,
0,0,
1,2,
2,4,
3,6,
4,8);
```

lookup(x, y0,y1,y2...)
Returns the xth value in a single dimensional array defined by y0,y1,y2 ect.
lookup(3, 0,2000,3000,4000,5000)==4000

brklookup(x, var_breakpoints, var_fraction_ret)
Returns the lower extrapolated value and fraction to next breakpoint in a single dimensional array.
var(xbrk[5],0,2000,3000,4000,5000);
var(frac,0);
brklookup(2800,xbrk,frac)==1, frac==0.8

In the below example, brklookup is used to find the interpolated value of a 3 dimensional table defined by 2 single dimensional arrays as axis breakpoints and a third array as the map content. The lookup function applied to the breakpoint arrays gives a position in the content table to reference. The 4 nearest points can then be used to find an interpolated value based on the fraction return.

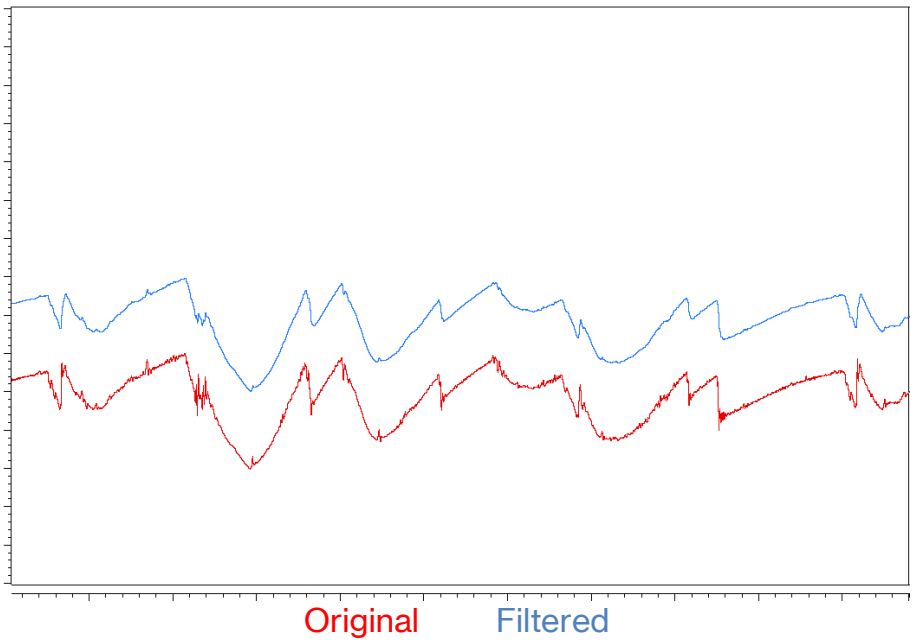
```
var(xbrks[13], 200,400,600,800,1000,1200,1400,1600,1800,2000,2200,2400,2600);
var(ybrks[14], 700,1000,1500,2000,2500,3000,3500,4000,4500,5000,5500,6000,6500,7000);
var(map[13,14],
0.4,    1.324,  2.118,  2.663,  3.207,  4.752,  6.402,  8.379,  10.715, 13.217, 15.37,  17.76,  20.385,
0.226,  1.116,  2.07,   2.639,  3.207,  4.725,  6.479,  8.451,  10.863, 13.438, 15.595, 17.988, 20.618,
0.135,  0.969,  1.99,   2.599,  3.207,  4.678,  6.608,  8.57,   11.11,  13.806, 15.969, 18.369, 21.006,
0.045,  0.891,  1.838,  2.47,   3.074,  4.631,  6.736,  8.69,   11.357, 14.173, 16.342, 18.749, 21.394,
0.041,  0.995,  1.939,  2.308,  3.603,  5.111,  6.865,  8.809,  11.604, 14.541, 16.716, 19.13,  21.782,
0.036,  1.091,  2.062,  2.726,  3.84,   5.542,  7.271,  9.401,  12.23,  14.909, 17.09,  19.477, 22.07,
0,      0.688,  1.977,  3.093,  4.604,  6.159,  7.974,  9.912,  11.847, 14.038, 16.472, 18.905, 21.906,
0,      0.798,  1.897,  2.795,  4.368,  5.577,  7.616,  9.763,  11.821, 14.195, 16.812, 19.428, 22.131,
0,      1.007,  2.015,  3.023,  4.555,  5.879,  7.871,  10.075, 12.37,  14.204, 16.494, 19.194, 21.857,
0,      1.123,  2.245,  3.368,  4.961,  6.366,  8.218,  10.61,  12.377, 14.145, 16.107, 18.568, 21.176,
0,      1.144,  2.287,  3.69,   5.214,  6.841,  8.684,  10.68,  12.776, 14.729, 16.582, 18.635, 20.881,
0,      1.165,  2.329,  3.843,  5.619,  7.197,  9.139,  11.354, 13.683, 15.599, 17.516, 19.432, 21.72,
0,      1.186,  2.37,   3.912,  5.7,    7.367,  9.724,  12.171, 14.234, 16.297, 18.403, 20.431, 22.599,
0,      1.207,  2.412,  3.981,  5.781,  7.329,  9.519,  11.982, 14.108, 16.235, 18.398, 20.269,22.452);
```

© Life Racing Ltd

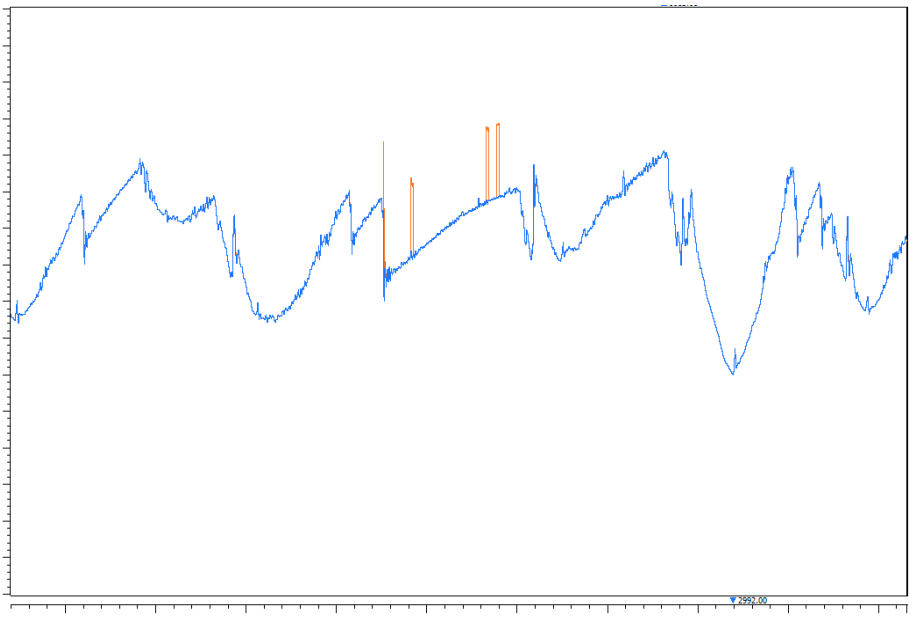
```
var(x,0);
var(xfraq,0);
var(y,0);
var(yfraq,0);
x = brklookup(map1, xbrks, xfraq);
y = brklookup(rpm, ybrks, yfraq);

var(x1y1,0);
var(x2y1,0);
var(x1y2,0);
var(x2y2,0);
x1y1 = map[x,y];
x2y1 = map[x+1,y];
x1y2 = map[x,y+1];
x2y2 = map[x+1,y+1];

var(xy1,0);
var(xy2,0);
xy1 = (x1y1+((x2y1-x1y1)*xfraq));
xy2 = (x1y2+((x2y2-x1y2)*xfraq));

var(xy,0);
xy = xy1+((xy2-xy1)*yfraq);

xy
```

3.8 Filters

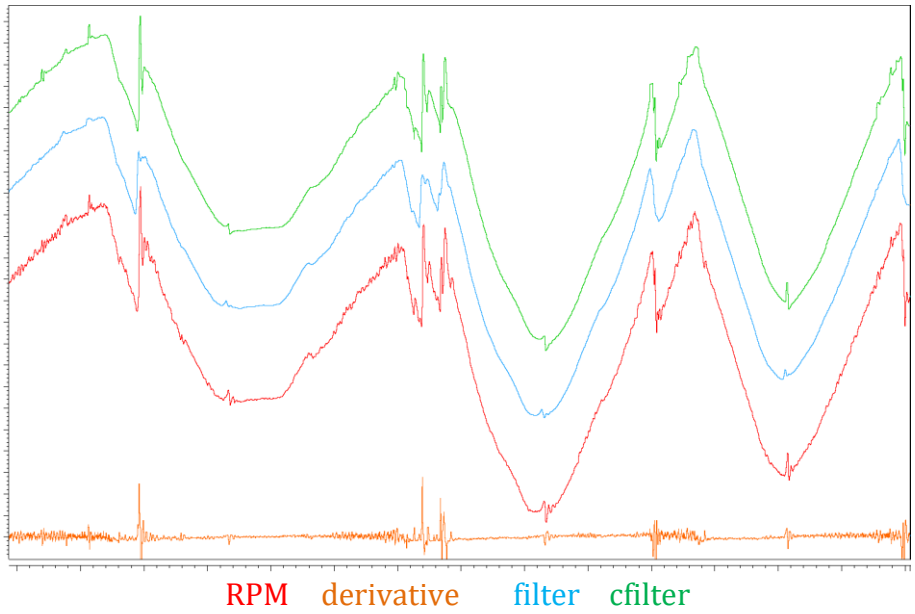
filter(x,filter)
Smooths the x trace at a magnitude of filter (between 0 and 1).
filter(rpm,0.9)



nfilter(x,delta)
Noise filter of x where delta is max difference allowed between samples. Useful for removing spikes in data.
nfilter(rpm,500)



cfilter(condition,x,filter)
Smooths the x trace when condition is true at a magnitude of filter (between 0 and 1).
cfilter(abs(derivative(rpm))<2800,rpm,0.9)



cfilter2(condition,x,min,max,filter)
Smooths the x trace when conditions are met and only between the specified min and max values at a magnitude of filter (between 0 and 1).
cfilter2(abs(derivative(rpm))<2800,rpm,0,6000,0.9)

© Life Racing Ltd

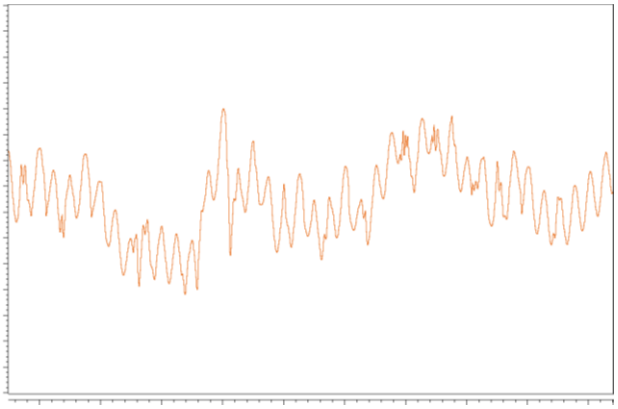
Running average functions are primarily used as noise filters. By setting the number of samples as the period of the noise frequency, the noise should cancel itself out. If the noise frequency changes a varying number of samples can be used. This should be a function of the cause of noise. Use the delta to work out the frequency and number of samples across the period making sure the sample rate is the same for the data and runavg filter. Running min/max functions are used in exactly the same way but return the lowest or highest values instead of the average.

runmin(x,num_samples)
Returns minimum value of x over the last n samples.

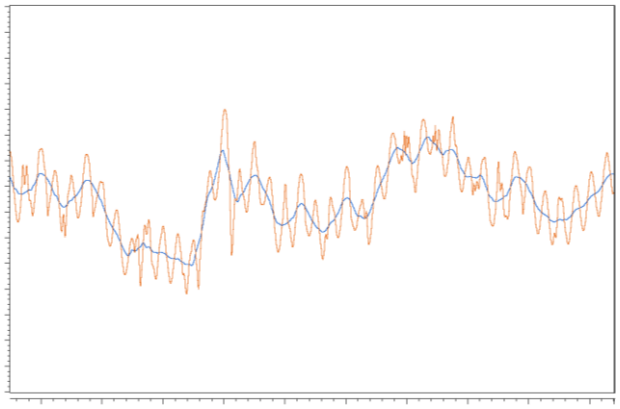
runmax(x,num_samples)
Returns maximum value of x over the last n samples.

runavg(x,num_samples)
Running average of x using n number of samples.

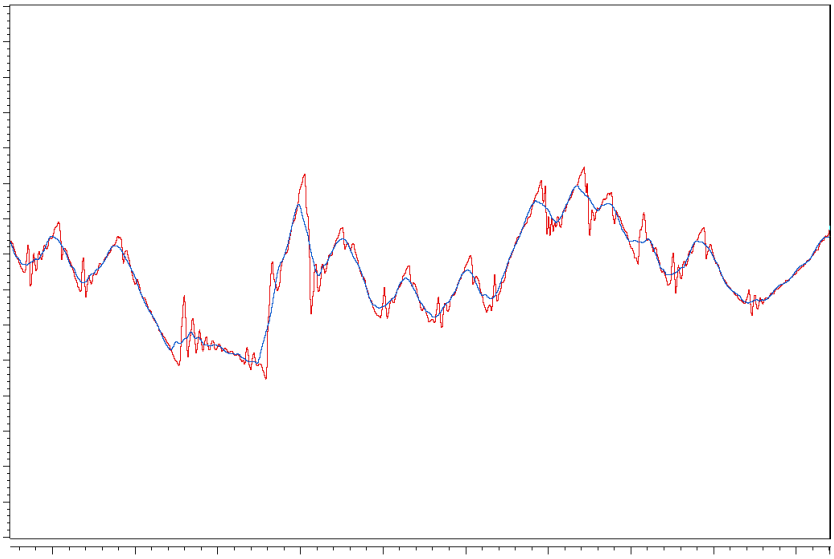
Constant frequency, measured delta 0.63 seconds at 100Hz, 63 samples
runavg(orange,63)



data with noise



runavg filter over noisy data



runavg filter over original

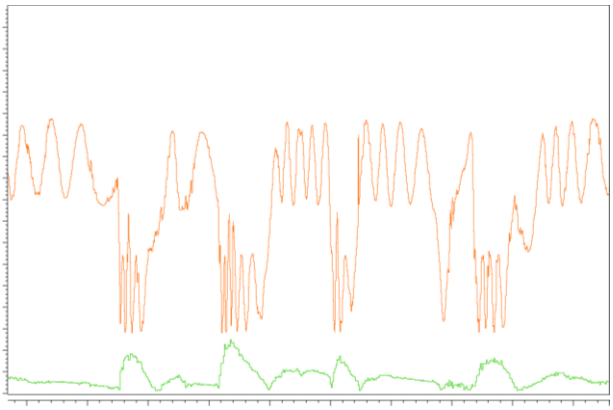
© Life Racing Ltd

runmin(x,max_samples,var_num_samples)
Running minimum of x where the number of samples can vary.

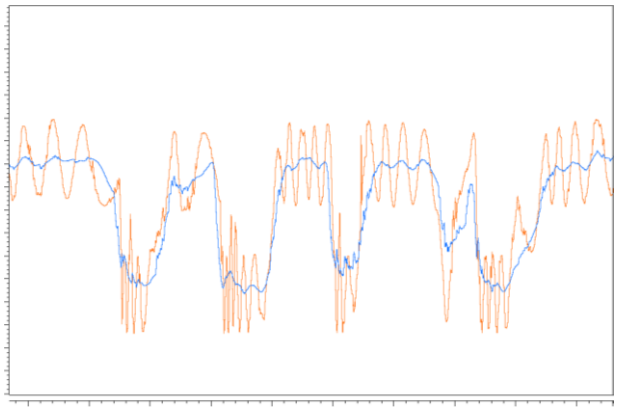
runmax(x,max_samples,var_num_samples)
Running maximum of x where the number of samples can vary.

runavg(x,max_samples,var_num_samples)
Running average of x where the number of samples can vary.

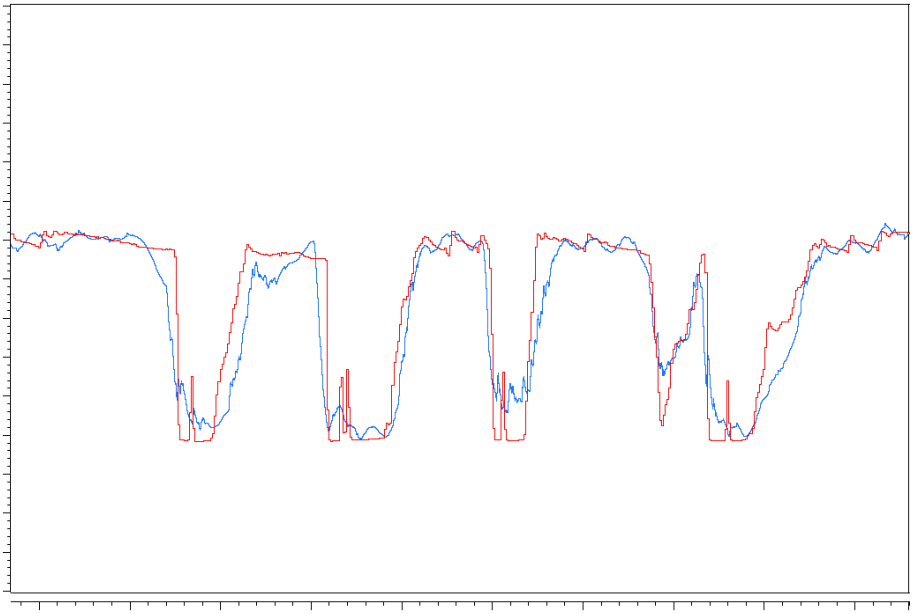
Variable frequency found to be dependent on *green*.
Largest period: delta=2.485s at 100Hz, 250 samples.
Random, average size period: delta=1.298s at 100Hz, 130 samples.
green = 16.95 at midpoint.
130x16.95=2203.5
filter(runavg(orange,250,2200/green),0.95)



data with noise and frequency dependant



runavg filter over noisy data

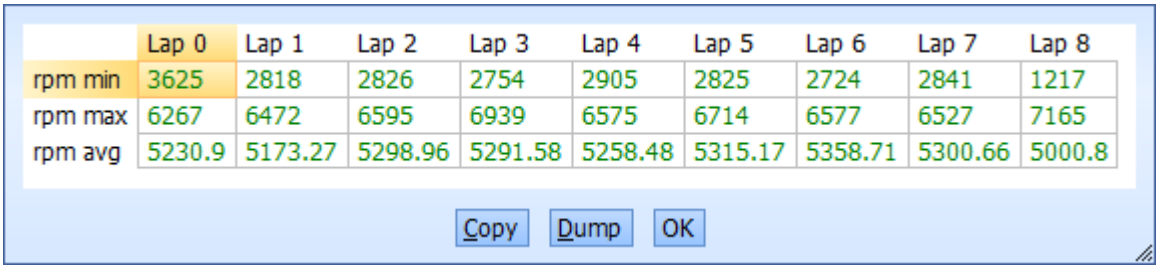


runavg filter over original

© Life Racing Ltd

4 Maths Reports

Maths reports will generate information in a tabular format in a similar way to the native lap reports. These however can be customised to display any information generated by maths. These make useful quick glance comparisons or can be exported for further analysis. Maths reports must use Maths Language Version 2.



	Lap 0	Lap 1	Lap 2	Lap 3	Lap 4	Lap 5	Lap 6	Lap 7	Lap 8
rpm min	3625	2818	2826	2754	2905	2825	2724	2841	1217
rpm max	6267	6472	6595	6939	6575	6714	6577	6527	7165
rpm avg	5230.9	5173.27	5298.96	5291.58	5258.48	5315.17	5358.71	5300.66	5000.8

4.1 Report format

If any of the following functions are used in a math channel then a report will be generated rather than a trace in a graph.

reportLeft(y, formatString ...)
Define y axis heading at position “y”.
reportLeft(0,”rpm min”);
reportLeft(1,”rpm max”);
reportLeft(2,”rpm avg”);

reportTop(x, formatString ...)
Define x axis heading at position “x”.
reportTop(lastLapNo,"Lap %u",lastLapNo);

reportData(x, y, value)
Define table content in position “x,y”.
reportData(lastLapNo,0,rpmMin);
reportData(lastLapNo,1,rpmMax);
reportData(lastLapNo,2,rpmAvg);

Note that the table starts at position zero. Rows and columns must be increased in size by one element at a time. Format specifiers may be used to add variable values to row and column labels. For each specifier, an additional parameter must be added. In this way, multiple specifiers can be used with each parameter applied in order of specifier appearance. For example:

reportLeft(row,"Lap %u : Corner %u",lastLapNo,cornerNo);

© Life Racing Ltd

Supported specifiers are listed below. To use a percentage sign as a percentage sign, it needs to be typed twice i.e. %%.

Format specifiers
%d = signed integer
%u = unsigned integer
%g = floating point number
%x = hex number
%c = asci character

4.2 Statistical functions

To assist in writing reports, several built in statistical functions have been added. These can also be used in standard expressions.

stat(variableName, stat type)
Declares statistic variable for MIN, MAX, AVG, STDDEV and LINEFIT stat types
stat(rpmAvg,AVG);

statUpdate(variableName, value)
Update stat variable from sample value
statUpdate(rpmAvg,rpm)

statUpdate(variableName, valueX, valueY)
Update stat variable from two sample values for LINEFIT stat type
statUpdate(driveRatio,drivenSpeed,rpm)

statReset(variableName)
reset stat variable, typically done at beginning of each lap
if(lapNo!=lastLapNo,statReset(rpmAvg);

statValue(variableName, altResult)
Return alternative statistical value for the STDDEV and LINEFIT stat types.

STDDEV
altResult==0 gives standard deviation - default
altResult==1 gives coefficient of variation

LINEFIT
altResult==0 gives M value (y=mx+c) - default
altResult==1 gives C value (y=mx+c)
altResult==2 gives correlation coefficient

reportData(4,lastLapNo,statValue(rpmSpd_C,1));

Calling a stat variable without this function will return the default result.

© Life Racing Ltd

4.3 Example

```
MathLanguageVersion(2,0);
var(lapNo,0);
var(lastLapNo,0);

stat(rpmMin,MIN);
stat(rpmMax,MAX);
stat(rpmAvg1,AVG);
stat(rpmStdDev,STDDEV);
stat(rpmSpd_C,LINEFIT);

if(lr.sample==0,
{
  //add top header first time round
  reportTop(0,"rpm min");
  reportTop(1,"rpm max");
  reportTop(2,"rpm avg");
  reportTop(3,"rpm StDv");
  reportTop(4,"rpmSpd C");
}); //no else clause

lapNo = getlap();
if(lapNo!=lastLapNo || lr.lastSample,
{
  //add new left row header and report previous laps statistics
  reportLeft(lastLapNo,"Lap %u",lastLapNo);
  reportData(0,lastLapNo,rpmMin);
  reportData(1,lastLapNo,rpmMax);
  reportData(2,lastLapNo,rpmAvg1);
  reportData(3,lastLapNo,rpmStdDev);
  reportData(4,lastLapNo,statValue(rpmSpd_C,1));

  //reset stat variables
  statReset(rpmMin);
  statReset(rpmMax);
  statReset(rpmAvg1);
  statReset(rpmStdDev);
  statReset(rpmSpd_C);
}); //no else clause

//update stat variables for this sample
statUpdate(rpmMin,rpm);
statUpdate(rpmMax,rpm);
statUpdate(rpmAvg1,rpm);
statUpdate(rpmStdDev,rpm);
statUpdate(rpmSpd_C,rpm,vehicleSpeed);
lastLapNo = lapNo;
```

© Life Racing Ltd

5 Document Revision History

2015-08-12	MH V1.0	Initial public release
2017-02-15	MH V2.0	Updated to new format More examples and explanations added.
2019-04-25	MH V2.1	Added new language version Added comments Added assignment operators Added “Duration” and “interpolatedValue” functions New brklookup example for more complicated arrays Added Reports

© Life Racing Ltd