UAMS
UNIVERSITY OF ARKANSAS FOR MEDICAL SCIENCES
Department of Neurobiology and Developmental Sciences
Center for Translational Neuroscience
College of Medicine

Neuronal Signals - NBDS 5161
Session 8: Writing algorithms

Abdallah HAYAR

Lectures can be downloaded from
http://hayar.net/NBDS5161

Updated Tentative Schedule for Neuronal Signals (NBDS 5161)
One Credit–Hour, Summer 2010
Location: Biomedical Research Building II, 6th floor, conference room,
Time: 9:00 -10:20 am

Session	Day	Date	Topic	Instructor
1	Tue	6/1	Design of an electrophysiology setup	Hayar
2	Thu	6/3	Neural population recordings	Hayar
3	Thu	6/10	Single cell recordings	Hayar
4	Fri	6/11	Analyzing synaptic activity	Hayar
5	Mon	6/14	Data acquisition and analysis	Hayar
6	Wed	6/16	Analyzing and plotting data using OriginLab	Hayar
7	Fri	6/18	Detecting electrophysiological events	Hayar
8	Mon	6/21	Writing algorithms in OriginLab®	Hayar
9	Wed	6/23	Imaging neuronal activity	Hayar
10	Fri	6/25	Exam and students' survey - Laboratory demonstration	Hayar
11	Fri	7/9	Article presentation I: Electrophysiology	Hayar
12	Mon	7/12	Article presentation II: Imaging	Hayar
13	Wed	7/14	Exam and students' survey about the course	Hayar

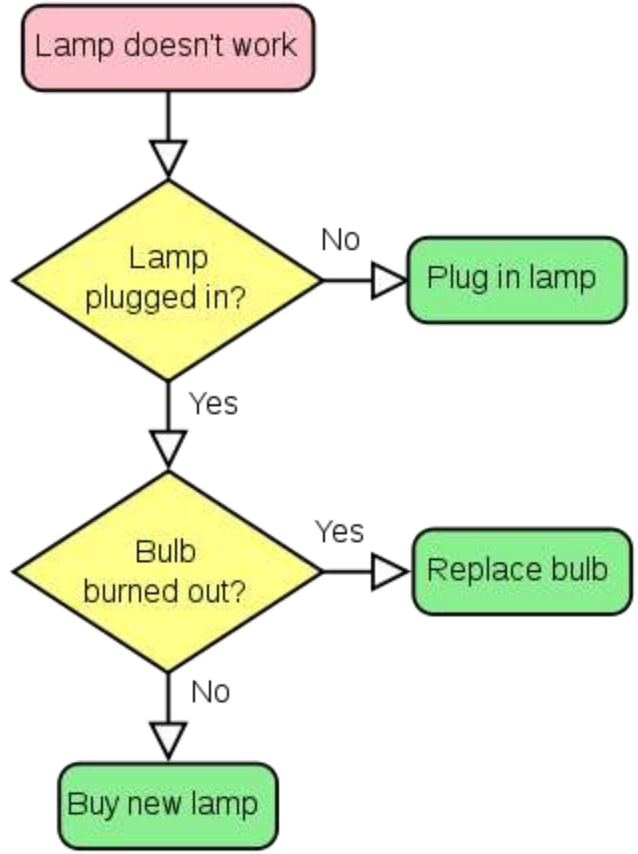
Student List

	Name	E-mail	Regular/Auditor	Department	Position
1	Simon, Christen	CSimon@uams.edu	Regular (form signed)	Neurobiology & Developmental Sciences	Graduate Neurobiology – Mentor: Dr. Garcia-Rill
2	Kezunovic, Nebojsa	NKezunovic@uams.edu	Regular (form signed)	Neurobiology & Developmental Sciences	Graduate Neurobiology – Mentor: Dr. Garcia-Rill
3	Hyde, James R	JRHyde@uams.edu	Regular (form signed)	Neurobiology & Developmental Sciences	Graduate Neurobiology – Mentor: Dr. Garcia-Rill
4	Yadlapalli, Krishnapraveen	KYadlapalli@uams.edu	Regular (form signed)	Pediatrics	Research Technologist – Mentor: Dr. Alchaer
5	Pathan, Asif	APATHAN@uams.edu	Regular (form signed)	Pharmacology & Toxicology	Graduate Pharmacology – Mentor: Dr. Rusch
6	Kharade, Sujay	SKHARADE@uams.edu	Regular (form signed)	Pharmacology & Toxicology	Graduate Pharmacology – 4th year - Mentor: Dr. Rusch
7	Howell, Matthew	MHOWELL2@uams.edu	Regular (form signed)	Pharmacology & Toxicology	Graduate Interdisciplinary Toxicology - 3rd year - Mentor: Dr. Gottschall
8	Beck, Paige B	PBBeck@uams.edu	Regular (form signed)	College of Medicine	Medical Student – 2nd Year - Mentor: Dr. Garcia-Rill
9	Atcherson, Samuel R	SRAtcherson@uams.edu	Auditor (form signed)	Audiology & Speech Pathology	Assistant Professor
10	Detweiler, Neil D	NDDETWEILER@uams.edu	Auditor (form not signed)	Pharmacology & Toxicology	Graduate Pharmacology – 1st year
11	Thakali, Keshari M	KMThakali@uams.edu	Unofficial auditor	Pharmacology & Toxicology	Postdoctoral Fellow – Mentor: Dr. Rusch
12	Boursoulian, Feras	FBoursoulian@uams.edu	Unofficial auditor	Neurobiology & Developmental Sciences	Postdoctoral Fellow – Mentor: Dr. Hayar
13	Steele, James S	JSSTEELE@uams.edu	Unofficial auditor	College of Medicine	Medical Student – 1st Year – Mentor: Dr. Hayar
14	Smith, Kristen M	KMSmith2@uams.edu	Unofficial auditor	Neurobiology & Developmental Sciences	Research Technologist – Mentor: Dr. Garcia-Rill
15	Gruenwald, Konstantin	kjoachimg@gmail.com	Unofficial auditor	Neurobiology & Developmental Sciences	High school Student – Mentor: Dr. Hayar
16	Rhee, Sung	RheeSung@uams.edu	Unofficial auditor	Pharmacology & Toxicology	Assistant Professor
17	Light, Kim E	LightKimE@uams.edu	Unofficial auditor	Pharmaceutical Sciences	Professor

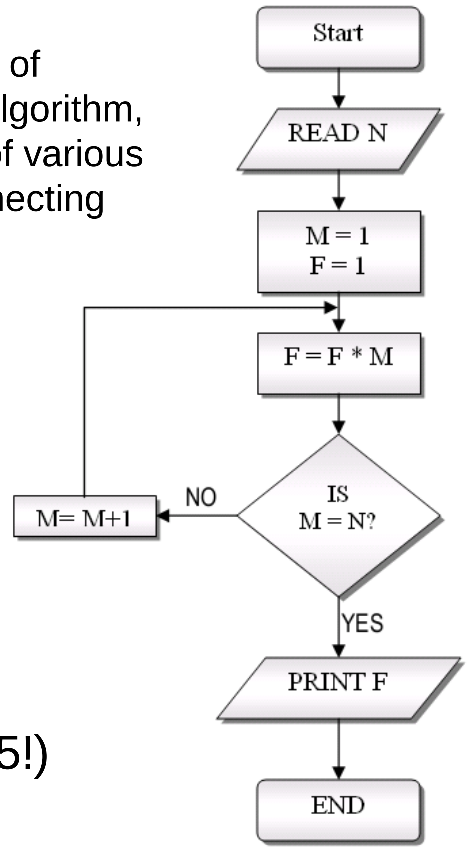
Algorithms and Flowchart

An 'algorithm' is an effective method for solving a problem expressed as a finite sequence of instructions. Algorithms are used for calculation, data processing, and many other fields.

Each algorithm is a list of well-defined instructions for completing a task. Starting from an initial state, the instructions describe a computation that proceeds through a well-defined series of successive states, eventually terminating in a final ending state.



A flowchart is a common type of diagram, that represents an algorithm, showing the steps as boxes of various kinds, and their order by connecting these with arrows

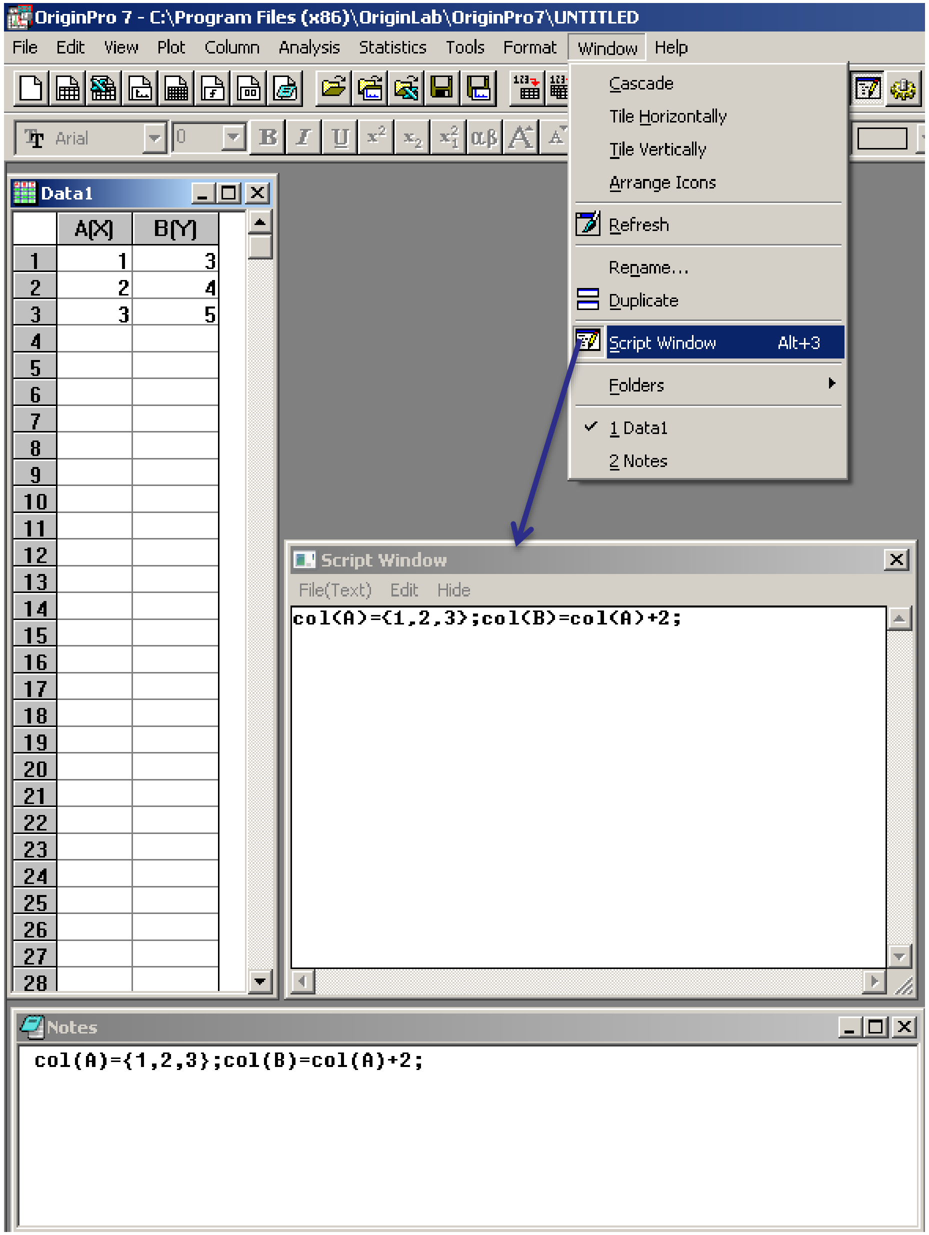


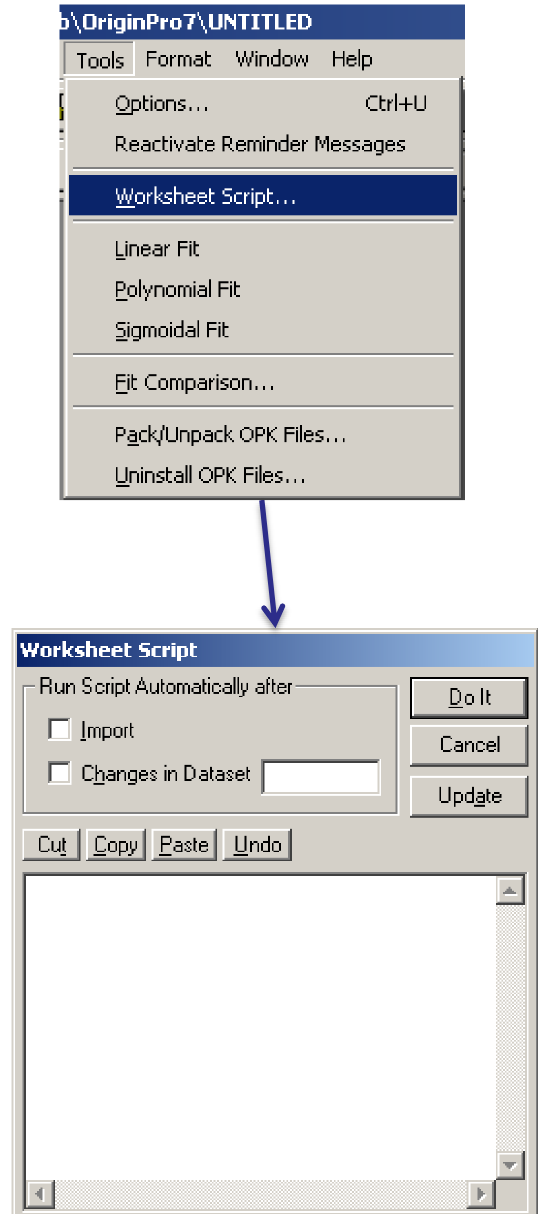
N!, is the product of all positive integers less than or equal to N

A simple flowchart for computing factorial N (5!)
5!=1*2*3*4*5 = 120

This is an algorithm that tries to figure out why the lamp doesn't turn on and tries to fix it using the steps. Flowcharts are often used to graphically represent algorithms.

Writing Scripts in Origin





Contrasting LabTalk and Origin C

	LabTalk	Origin C
Compiled?	No	Yes Programs can also be saved to disk in a pre-compiled form for faster recall.
Speed	An interpreted language. Relatively slow, especially in the case of loops requiring many iterations	A compiled language, so it is much faster (up to 20 times) than LabTalk. Especially well suited to computationally intensive operations and it is ideal for user-defined curve fitting functions.
Access to internal Origin objects	Yes Since LabTalk pre-dates Origin C, it provides somewhat better access than Origin C.	Yes Access is object-oriented. At present, it does not provide as much access to internal objects and properties as LabTalk. This should rapidly improve.
Case sensitive? (commands and variables)	No Variable a and variable A are considered to be the same variable.	Yes, Origin C is case sensitive. Variable A and variable a would be considered to be different variables. The same applies to function names.
Functions	No LabTalk has the ability to call sections in script files having .OGS extensions. This allows the passing of simple text arguments.	Yes Standard rules of C language apply for calling functions. This is much more convenient than calling script file sections in LabTalk.
Mode of execution	LabTalk scripts are usually organized by sections in .OGS script files. They can be called using the run.section() command from either Script window or from another LabTalk script. Also, LabTalk scripts can be typed directly to the Script window and executed from there. They can be associated with menu commands or toolbar buttons, or with buttons on various Origin windows (graphs, worksheets, etc.). LabTalk scripts can also be executed from Origin C function.	Origin C code is always organized in functions. Functions can be called from other Origin C functions in the standard way by passing arguments of different types. They can be called from the Script window, from LabTalk scripts, from menu commands and toolbars buttons, as well as from buttons on various Origin windows (graphs, worksheets, etc.).

	LabTalk	Origin C
Variable types	**Yes** Only numeric (double precision) and a limited number of string variables are supported. Variables representing internal Origin objects are not supported. It is possible to refer to various global objects, such as the active window, layer, etc.	**Yes** All standard C types are supported, as are pointers. Also, variables representing internal Origin objects are supported (access to those objects is object-oriented). All variables must be declared before being used.
Local variables	**No**	**Yes** Local variables in functions must be declared before being used, (as is standard in the C language).
Global variables	**Yes** All variables in LabTalk are global variables. These global variables are either numeric (do not have to be declared before being used since they are defined and space in memory is reserved for them on first use) or string (there are 26 LabTalk string variables, named %A, %B, etc. Some, such as %H, (contains the name of the active window), are reserved.	**Yes** All global variables must be declared outside functions before being used.
Multidimensional objects	**No**	**Yes** Origin C supports vector and matrix classes (and the associated classes Dataset and Matrix which provide access to Origin's internal datasets and matrices). These can be dereferenced using [ ] notation (vector v;…;. v[3] = …;) to access individual elements.
Collections	**No**	**Yes** Origin C supports various collections of internal Origin objects, such as the collection of all windows in a project, all columns in a worksheet, all data plots in a graph layer, etc. Collections allow for easy enumeration and access to the items being held in the collection.
Control structures	LabTalk supports C-like **if-else** and **switch** statements. It also supports C-like **for**-loop, as well as LabTalk-specific **repeat** and **loop** looping control structures.	It supports all C-style control structures (**if-else**, **switch**, **for**, **while**, **do-while**, **goto**). It also supports **foreach** loops which provide a simple way to enumerate all members of a collection.
Writing user-defined fitting functions	**Yes**	**Yes** Compiled Origin C functions greatly increase curve fitting speed.
Calling external functions (functions written in external DLLs)	**No**	**Yes** A function implemented in an external DLL (the function must be exported from the DLL in a standard way) can be called from Origin C. This enables use of proprietary routines written in standard Windows DLLs, to be used inside of Origin. This is no more difficult than calling another Origin C function.

A computer program in the form of a human-readable, computer programming language is called source code. Source code may be converted into an executable image by a compiler or executed immediately with the aid of an interpreter.

Either compiled or interpreted programs might be executed in a batch process without human interaction, but interpreted programs allow a user to type commands in an interactive session. In this case the programs are the separate commands, whose execution is chained together. When a language is used to give commands to a software application (such as a shell) it is called a scripting language.

Compiled computer programs are commonly referred to as executables, binary images, or simply as binaries — a reference to the binary file format used to store the executable code. Compilers are used to translate source code from a programming language into either object code or machine code.

Arithmetic Operators

<table>
<tr><th>Operator</th><th>Use</th></tr>
<tr><td>+</td><td>Addition</td></tr>
<tr><td>-</td><td>Subtraction</td></tr>
<tr><td>*</td><td>Multiplication</td></tr>
<tr><td>/</td><td>Division</td></tr>
<tr><td>^</td><td>Exponentiate (X^Y raises X to the Yth power)</td></tr>
<tr><td>&</td><td>Bitwise AND operator. Acts on the binary bits of a number.</td></tr>
<tr><td>|</td><td>Bitwise OR operator. Acts on the binary bits of a number.</td></tr>
</table>

10*5+3*2-10/5=;
10*5+3*2-10/5=54

5+6*2=
5+6*2=17

(5+6)*2=
(5+6)*2=22

2^16=;
2^16=65536

0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
10 & 11 = 10

0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1
10 | 00 = 10

Conditional and Loop Structures

	Loop	Repeat	For
Description	The loop is used when a single variable is being incremented with each successive loop.	The repeat loop is used when a set of actions must be repeated without any alterations.	The for loop is used for all other situations.
Syntax	loop (variable, startVal, endVal) {script};	repeat value {script};	for (expression1; expression2; expression3) {script};
Example 1: Count from 1 to 4	loop(X,1,4) {X=} X=1 X=2 X=3 X=4	X=1;repeat 4 {X=;X=X+1} X=1 X=2 X=3 X=4	for(X=1;X<=4;X++) {X=} X=1 X=2 X=3 X=4
Example 2: Calculate factorial N!	N=1;loop(X,1,4) {N=N*X;N=} N=1 N=2 N=6 N=24	N=1;X=1;repeat 4 {N=N*X;X=X+1;N=} N=1 N=2 N=6 N=24	for(X=1,N=1;X<=4;X++) {N=N*X;N=} N=1 N=2 N=6 N=24

Decision Structures

loop (X,1,8) { if (X <= 4) {X=} }
X=1
X=2
X=3
X=4

loop (X,1,8) { if (X > 4) {X=} }
X=5
X=6
X=7
X=8

loop (X,1,8) { if (X>3 && X<7) {X=} }
X=4
X=5
X=6

loop (X,1,8) { if (X/2 == int(X/2)) {X=} }
X=2
X=4
X=6
X=8

loop (X,1,8) { if (X/2 != int(X/2)) {X=} }
X=1
X=3
X=5
X=7

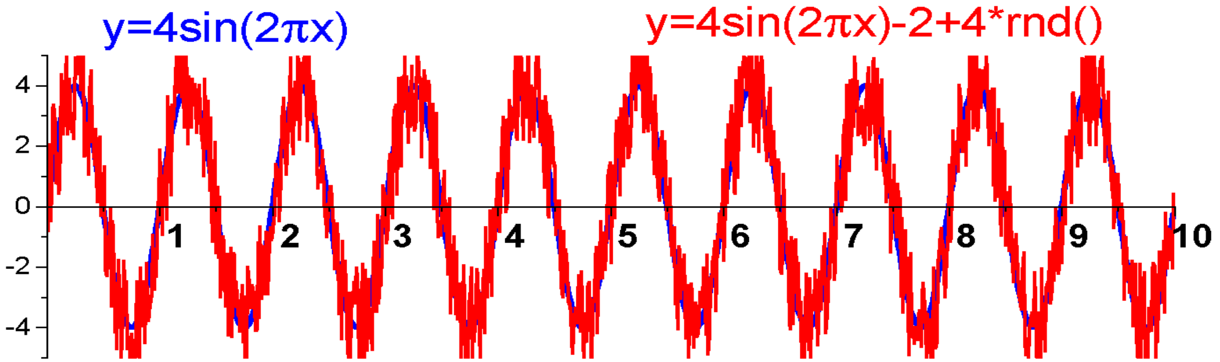
loop (X,1,4) { if (X/2 == int(X/2)) {type $(X) is even} else {type $(X) is odd} }
1 is odd
2 is even
3 is odd
4 is even

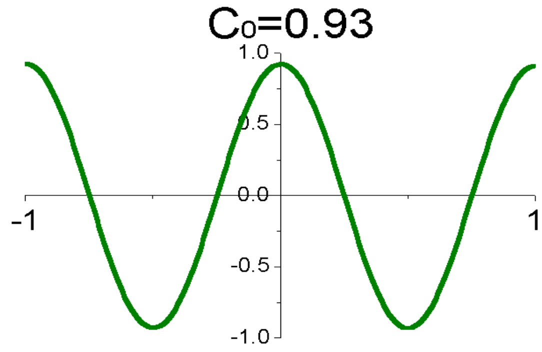
An expression involving logical or relational operators evaluates to either true (non-zero) or false (zero).

<table>
<tr><th colspan="2">Logical and Relational Operators</th></tr>
<tr><td>Operator</td><td>Use</td></tr>
<tr><td>></td><td>Greater than</td></tr>
<tr><td>>=</td><td>Greater than or equal to</td></tr>
<tr><td><</td><td>Less than</td></tr>
<tr><td><=</td><td>Less than or equal to</td></tr>
<tr><td>==</td><td>Equal to</td></tr>
<tr><td>!=</td><td>Not equal to</td></tr>
<tr><td>&&</td><td>And</td></tr>
<tr><td>||</td><td>Or</td></tr>
</table>

Mathematical Functions

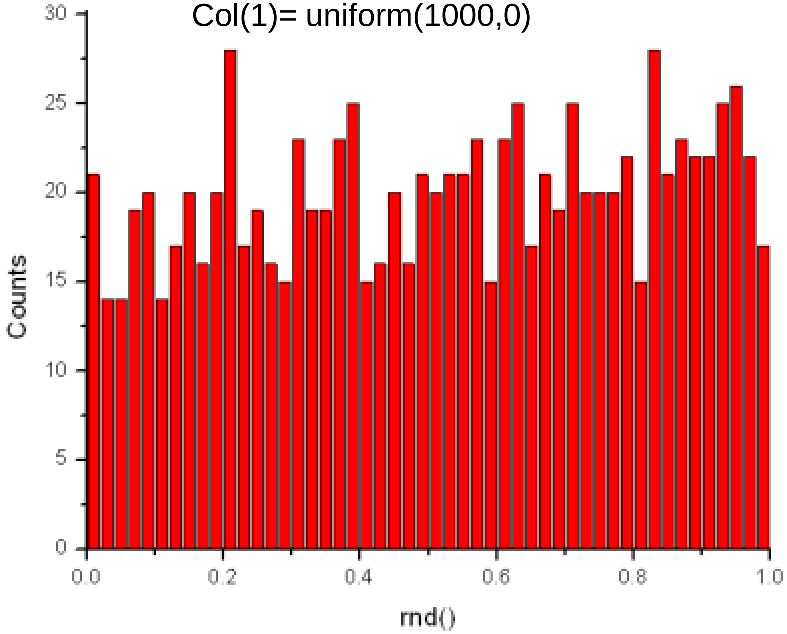
Name	Brief Description	Examples
Abs(x)	Returns the absolute value of x	abs(-5)= 5; abs(5)= 5; abs(0)= 0;
Cos(x)	Returns value of cosine for each value of the given x.	cos(0)= 1; cos(pi)= -1; cos(pi/2)= 0
Exp(x)	Returns the exponential value of x.	exp(1)= 2.718282; exp(0)= 1
Int(x)	Returns the truncated integer of x.	int(7.9) = 7; int(7.001)= 7; int(7.0)= 7
Ln(x)	Returns the natural logarithm value of x.	ln(1)= 0; ln(2.718282)= 1; ln(exp(1))= 1
Log(x)	Returns the base 10 logarithm value of x.	log(1)= 0; log (10)= 1; log (100)= 2
Mod(x, y)	Return the integer modulus (the remainder from division) of integer x divided by integer y; similar to: x- int(x/y)*y	mod(10,3)= 1; mod(11,3)= 2; mod(12,3)= 0;
Round(x, n)	Returns the value (or dataset) *x* to *n* decimal places.	round(9.124,2)= 9.12; round(9.124,2)= 9.13;
Sqrt(x)	Returns the square root of x; similar to: x^(1/2)	sqrt(9)= 3; sqrt(10)=3.162278
Rnd()	Return a value between 0 and 1 from a uniformly distributed sample.	rnd()=0.6933578; rnd()=0.240543
Grnd()	Returns a value from a normally (Gaussian) distributed sample,with zero mean and unit standard deviation.	


y=4sin(2πx)
y=4sin(2πx)-2+4*rnd()
4
2
0
-2
-4
1
2
3
4
5
6
7
8
9
10


C0=0.93
1.0
0.5
0.0
-0.5
-1.0
-1
1

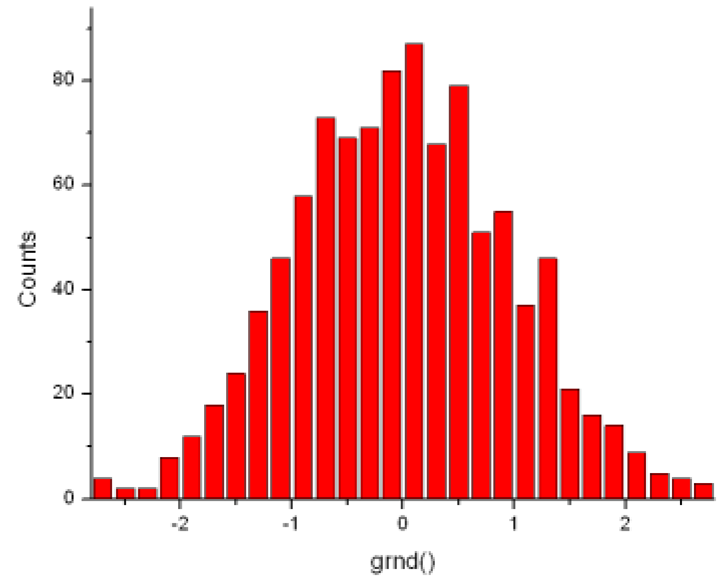
Uniformly distributed

loop(x,1,1000) {col(1)[x]=rnd()}
Col(1)= uniform(1000,0)



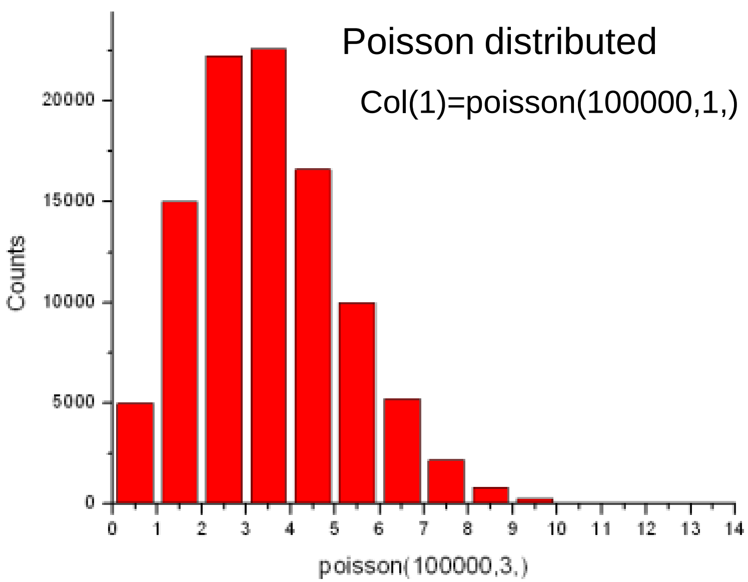
Normally (Gaussian) distributed

loop(x,1,1000) {col(1)[x]=grnd()};
Col(1)=normal(1000,0)



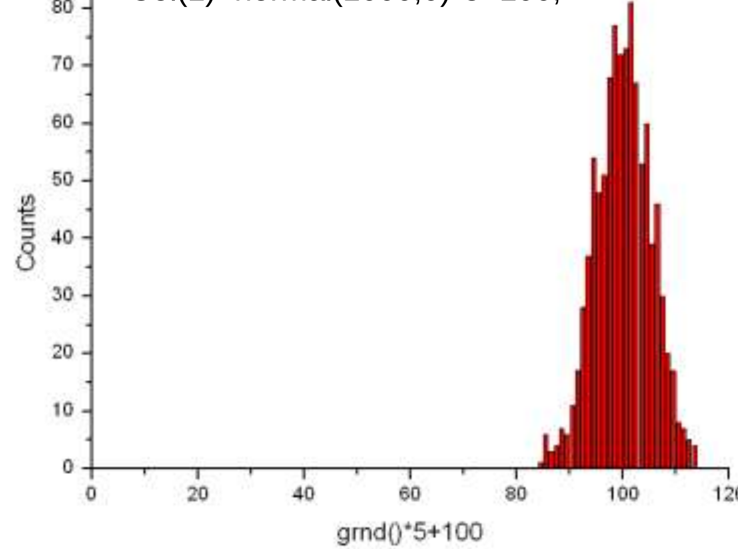
Poisson distributed

Col(1)=poisson(100000,1,)



Normally (Gaussian) distributed

Col(1)=normal(1000,0)*5+100;



Statistical Functions

Name	Brief Description	Examples
Data(x1, x2, inc)	Create a dataset with values ranging from x1–x2 with an increment, inc.	col(1)=data(0,1000,0.2)
Ave(dataset, n)	Breaks dataset into groups of size n, finds the average for each group, and returns a range containing these values.	col(2)=ave(col(1),5)
col(1)={a,b,c,d...}	Fills column 1 with data.	col(1)={1,3,4,7,10}
Sum(dataset)	Returns a range whose ***i*** th element is the sum of the first ***i*** elements of the dataset dataset.	col(2)=sum(col(1))
Diff(dataset)	Returns a dataset that contains the difference between adjacent elements in dataset.	col(3)=diff(col(1))
Histogram(dataset, inc, min, max)	Generates data bins from *dataset* in the specified range from *min* to *max*	col(4)=histogram(col(1),3,0,15)
sort(dataset)	Returns a dataset that contains *dataset*, sorted in ascending order.	col(5)=sort(col(3))
Xindex(x, dataset)	Returns the index number of the cell in the X dataset associated with *dataset*, where the cell value is closest to x.	

Data13

	A(X)	B(Y)
1	0	0.4
2	0.2	1.4
3	0.4	2.4
4	0.6	3.4
5	0.8	4.4
6	1	5.4
7	1.2	6.4
8	1.4	7.4
9	1.6	8.4
10	1.8	9.4
11	2	10.4

Data14

	A(X)	B(Y)	C(Y)	D(Y)	E(Y)
1	1	1	2	1	1
2	3	4	1	2	2
3	4	8	3	1	3
4	7	15	3	1	3
5	10	25		0	
6					
7					

col(1)={1,3,7,13}; sum (col(1)); sum.mean=; sum.total=; sum.min=; sum.max=; sum.sd=; sum.n=

SUM.MEAN=6
SUM.TOTAL=24
SUM.MIN=1
SUM.MAX=13
SUM.SD=5.291503
SUM.N=4

Data15

	A(X)	B(Y)
1	0	2
2	0.2	4
3	0.4	5
4	0.6	8
5	0.8	14
6	1	10
7	1.2	8
8	1.3	6
9	1.4	5
10	1.8	3
11		

limit col(2);limit.iMax=;limit.iMin=;limit.size=;limit.xMax=;limit.xMin=;limit.yMax=;limit.yMin=;

LIMIT.IMAX=5
LIMIT.IMIN=1
LIMIT.SIZE=10
LIMIT.XMAX=1.8
LIMIT.XMIN=0
LIMIT.YMAX=14
LIMIT.YMIN=2

Property	Description
limit.iMax	Corresponding index for maximum Y value
limit.iMin	Corresponding index for minimum Y value
limit.size	Total size (number of points) for dataset.
limit.xMax	Maximum X value.
limit.xMin	Minimum X value.
limit.yMax	Maximum Y value.
limit.yMin	Minimum Y value.

Data Access, Manipulation, and Calculation

Data10

	A(X)	B(Y)	C(Y)	D(Y)	E(Y)	F(Y)
1	1	5	–	0.5	5	2
2	3	8	–	1.5	8	4
3	5	12	–	2.5	12	6
4	7	–	10	3.5	–	8
5		20			20	10
6						12
7						
8						

Script example	Interpretation
Data10_A={1,3,5,7}	Fill Worksheet “Data10” column “A” with specific values
%(Data10,2)={5,8,12};	Fill Worksheet “Data10” column #2 with specific values
%(Data10,2,5)=20	Fill Worksheet “Data10” column #2 row# 5 with value 20
col(3)[4]=10	Fill column#3 row#4 with value 10
col(4)=col(A)/2	Column#4 = column “A” divided by 2
wcol(10/2)=col(2)	Column# 10/2=5 is filled with similar values as column#2
loop(x,1,6) {col(6)[x]=2*x}	Fill column#6 row# x with double the value of x

Truncate a waveform

To truncate data beyond a value of -45 in column 2 that contains 28800 points;

```
loop(i,1,28800){if(wcol(2)[i]>-45){col(2)[i]=-45}}
```

Appending traces in Origin

All traces from column 3 to 10 will be appended to column(2)

```
loop(x,3,10){copy -a col(%(x)) col(2)}
```

Transform bursts of spikes into single events

To keep the first spike in a burst and discard events that appear after in the same burst. Spikes that are preceded by a short interspike interval (<IBI) will be ignored.

Col(1) contains time of spike occurrence in sec;

IBI= minimun interspike interval in ms

Col (burst) will contain the time of occurrence of the first spike in a burst

```
IBI=75;for(i=1,j=0;col(1)[i+1];i++){if((col(1)[i+1]-col(1)[i])>IBI){j++;col(burst)[j]=col(1)[i+1]}};
```

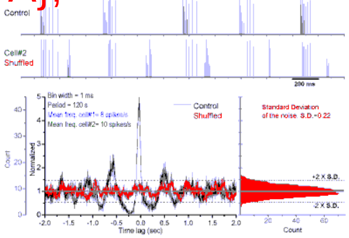
Shuffle time intervals

col(1) = time of events; col(2) = amplitude of events

col(3) = interevents intervals, shuffled interevents intervals, shuffled time of events

rnd()*1000 gives random numbers between 0 and 1000

```
col(3)=diff(col(1));limit col(3);
loop(i,1,limit.size) {A=col(3)[i]; R=rnd()*limit.size+1; col(3)[i]=col(3)[R]; col(3)[R]=A};
```



Measure the time a script will take to execute

```
sec -i;sec t;t=;for(x=1;x<5000;x++){y=x*x};sec t;t=;
```

T=0
T=0.25

Construct normalized interspike histograms for many columns

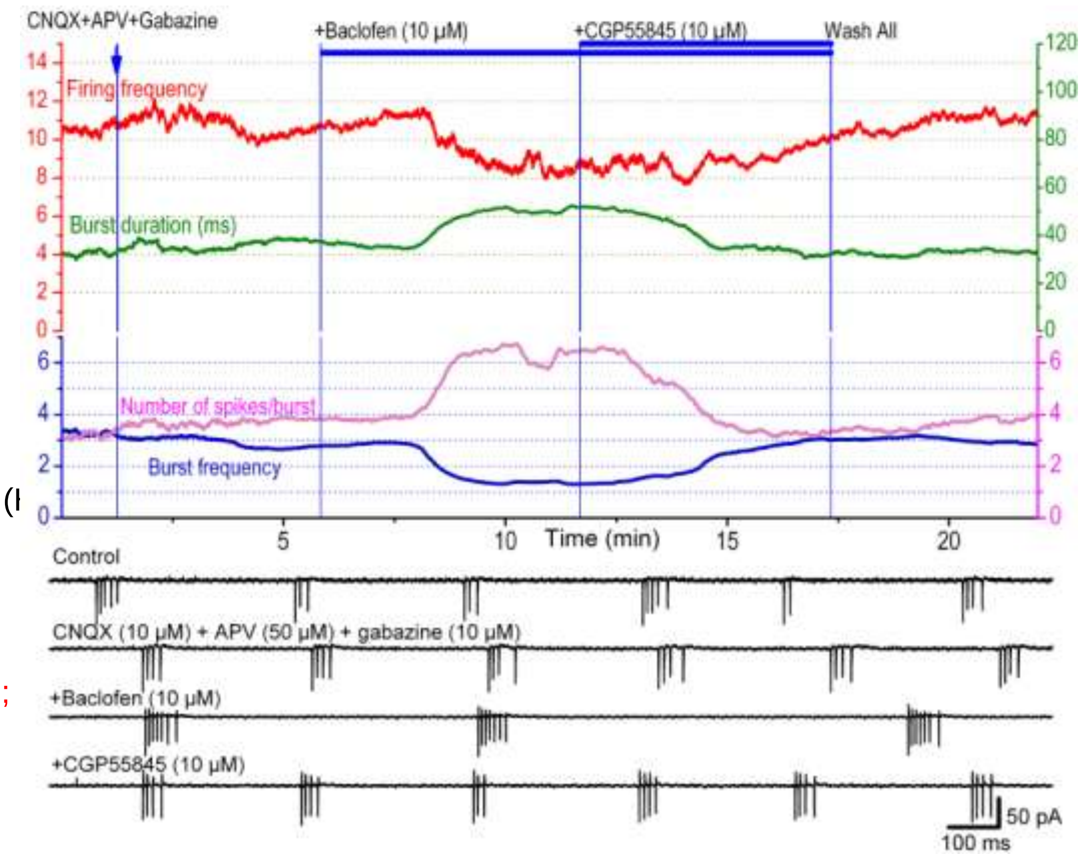
```
win -t data template A;win -a A;ClearWorksheet A;        //create worksheet named A; Activates worksheet A; ClearWorksheet
worksheet -n 2 B1;                                        //rename column2 as B1
loop(i,1,14){
worksheet -v B$(i);                                       //verify that a column B$(i) exists otherwise create it
%(A,i)=diff(%(Data1,i))};                                 //calculate intespike intervals for all columns

win -t data template B;win -a B;ClearWorksheet B;        //create worksheet named B; activates worksheet B; ClearWorksheet
worksheet -n 2 B1;                                        //rename column2 as B1
Bin=1;loop(i,1,14){
worksheet -v B$(i);                                       //verify that a column B$(i) exists otherwise create it
sum(diff(%(Data1,i)));%(B,1)=data(Bin/2,sum.max,Bin);//generate X axis column for histograms
%(B,i+1)=histogram(diff(%(Data1,i)),Bin,,sum.max)/sum.n}; //generate normalized interspike interval histograms
```

Calculate instantaneous spike and burst frequency, number of spikes/burst, burst duration

col(1) = time of spike occurrence (ms)
col(stime)= time of spike occurrence (min), X1 axis
col(ISI)= interspike interval (ms)
col(msfreq)= mean spike frequency (Hz)

col(burst)= time of burst ocurrence (ms)
col(btime)= time of burst ocurrence (min), X2 axis
col(IBI)= interburst interval (ms)
col(mbfreq)= mean burst frequency (Hz)
col(bd)=b= burst duration (ms)
col(mbd)=mean burst duration (ms)
col(spb)=n= number of spikes/ burst, n > 1
col(mspb)=mean number of spikes per burst (n)
col(mspbfreq)=mean spike frequency within a burst (I
col(ratio)=col(mspb)/col(mbfreq)




```
col(stime)=col(1)/60000;
col(ISI)=diff(col(1));col(msfreq)=col(ISI);
ave -n 60 col(msfreq);col(msfreq)=1000/col(msfreq);
S=75;n=1;j=1;b=0;for(i=1;col(1)[i+1];i++){
if(col(ISI)[i]>S){col(burst)[j]=col(1)[i+1];
col(bd)[j]=b;b=0;col(spb)[j]=n;n=1;j++} else {
n++;b+=col(ISI)[i]}};
col(btime)=col(burst)/60000;
col(IBI)=diff(col(burst));col(mbfreq)=col(IBI);
ave -n 60 col(mbfreq);col(mbfreq)=1000/col(mbfreq);
col(mbd)=col(bd);ave -n 60 col(mbd);
col(mspb)=col(spb);ave -n 60 col(mspb);
col(mspbfreq)=(1000/col(mbd))*(col(mspb)-1);
col(ratio)=col(mspb)/col(mbfreq);
window -a Graph1;layer1.x.rescalemargin=0;layer -s 1;layer -at;
```