Package: WBurt

Probability Distributions

by Warren Burt

Since ArtWonk has a probability module, in which you can specify just about any probability curve you want, why would one be interested in using probability distributions from the world of mathematics? Well, you might be able to draw any of these curves with the probability module, but you'd first have to know what these distributions were, in order to use them. Having them here, in function form, is a great way to learn about probability distributions. And besides, these probability distributions are a wonderful source of found objects. By playing with these, and adjusting their parameters, all sorts of surprising results occur.

Note also, that the output of these distributions can be treated with any of the other modules in ArtWonk to produce useful ranges of numbers. A few observations about useage follow:

1) You will want to multiply the output of any distribution to get values in a scale that you want. For example, if your function gives you values between 0 and 1, if you multiply that result by 60 and then put that result through an IMod module, also set to 60, and you apply that to pitch choice on a midi module, then voila! Pitch change occurring over a 5 octave range.

2) To get the inversion of any distribution that has outputs between 0 and 1, put the result through a SUB module, subtract 1, and then take the ABS of that. This will give you a resulting distribution that is symmetrical with the original. That is, if the distribution you're using has many more low values than high values, the distribution you get by inverting it will have many more high values than low values.

3) If you add or multiply or average any of these distributions together, you'll get more complicated results.

4) To get Poisson-like distributions, or Geometric distributions, put a bell-like curve distribution, or an exponential distribution through an INT module. This will produce discrete values for the result, which is the characteristic of Poisson and Geometric distributions.

The two sources for these functions are "Computer Music" by Charles Dodge and Tom Jerse, and "Appendix A: Compendium of Probability Distributions" to Regress +, a fine free mathematics program for the Mac. (http://www.causascientia.org/math_stat/Dists/Compendium.html). There are hundreds of other distributions out there, and if you get interested in this, you're encouraged to go out and harvest some of your own.

The demonstration patches for these distributions all use them to control pitches over a five octave range. This is just a quick way for you to hear the qualities of each of these distributions as a spray of pitches. (how many low or high pitches, etc.) You shouldn't let this emphasis on pitch inhibit you from using these distributions for controlling any and all aspects of sound, visuals or language.

And of course, these distributions can be used in any way you desire. Let's say that you have a distribution going through an INT module which is producing values 1,2,3,4 & 5. You can use these values any way you like. For example, you could have an array with 5 different midi key numbers in it. Each key could select a different sample. So you could use a probability distribution to select the midi key values from the array, which would then trigger off your (weighted) choice of 5 different samples. Or anything else. Durations, loudness levels, length of sections of a piece, tempi, etcetera.

Notes on the individual distributions follow. At the end of each section is the equation for the distribution as used in the function module in ArtWonk.

Beta Distribution (BetaDst.awf)

This is a unique distribution, with a shape unlike any other. Giving values between 0 and 1, it can emphasize the high and low values, with not too many results in the middle. There are two variables A & B, and several shapes.

When A & B > 1, it gives a bell-like Gaussian density.

When A = B = 1, it gives a degenerate case of the uniform density.

When A < 1 and B < 1, A controls the probability of values close to 0, while

B controls the probability of values close to 1.

In this case, you get the curve with the end values emphasized, and the middle values suppressed.

C.(Ran^(1/In2))

D.(Ran^(1/In3))

E.(C+D)

Out(1, (C/E))

Bradford Distribution (BradfordDst.awf)

This is a right-skewed distribution, which means that there are more low values than high values produced. There are a variety of shapes, controlled by a shape variable. This

distribution can also give values between any lower and upper value that you set. For normal useage, these values would be set between 0 and 1, but other ranges would be possible. There are 3 variables, A, B and C

A = the location, or start of the distribution. Set to 0 for values between 0 and 1.

B = the scale, or upper bound. Set to 1 for values between 0 and 1.

C = the shape. 5 = many more lows than highs; 1 = a few more lows than highs.

C can actually go very high >100 or more, and you'll get more and more skewed results.

Out(1, ((1/In4)*((In2*(In4+1)-In3+(In3-In2)*((In4+1)^Ran)))))

Bradford Mistake Distribution (BrdfdMstkDst.awf)

This is an attempt at the Bradford distribution, but I made a mistake with the ^Ran at the end of the equation, applying it to the wrong parentheses. It's included here because it shows that even a wrong equation can generate interesting results, and to encourage you to try out modifiying existing equations, or even making up your own equations for distributions. There are a variety of shapes, controlled by a shape variable. This distribution uses the lower and upper value that you set as a guide to range, but not quite. For normal useage, set variable A to 0 and variable B to 1,and hear what happens when you change variable C.

A = the location, or start of the distribution. Set to 0 for values between 0 and 1.

B = the scale, or upper bound. Set to 1 for values between 0 and 1.

C = the shape. If you set the value to 1, you'll just get repeating values of 0. If you set it to 0.9, you'll get values clustering around .98. If you set it to 1.1, you'll get values clustering around 1.02. As you increase or decrease the value, you'll hear the narrow band of values get wider. If you set the value to around 50, you'll get a result very similar to the (working version of the) Bradford distribution. In the BradfordMistakeDistribution patch, the IMOD is set to 60, and applied to control of pitch, so that with C = 0.9, you'll get a cluster of very high notes, and with C = 1.1, you'll get a cluster of very low notes.

Out(1, ((1/In4)*((In2*(In4+1)-In3+(In3-In2)*(In4+1))^Ran)))

Burr Distribution (BurrDst.awf)

The Burr Distribution is a very controllable distribution with a variety of shapes. It's somewhat right-skewed, which means that there will be more low values than high, but it can assume a variety of shapes based on the settings of the variables. It has 4 variables, A, B, C, and D.

A = location, the start of your distribution. 0 is a good value for this.

B = Scale, the "height" of the "hump" where you'll get the most frequently occuring value. 1 makes a higher "hump" than 2 does.

C and D = shape controls.

C = 2 = sharper initial rise of curve; C = 3 = shallower initial rise of curve.

D = 1 = sharper fall of curve; D = 2 shallower fall of curve.

This distribution produces values well above 1 (waaaay above 1), so you'll want to scale its output to get some useful values.

The Burr Distribution is a generalized distribution. It can assume the shapes of several other distributions.

With D = 1, it's often called the Fisk or LogLogistic distribution.

Out(1, (In2+(In3*(((1/(Ran^(1/In(5))))-1)^-(1/In4)))))

Cauchy Distribution (CauchyDst.awf)

The Cauchy Distribution produces bell-like curves which can be controlled as to their height and width. They always produce more values in the middle than at either of their ends. There are 2 variables, A and B.

A = Location - the center of the "hump." If 0, you'll get positive and negative values.

B = Scale. 1 = Moderately high. 0.5 = higher.

The Cauchy Distribution is sometimes also called the Lorentz distribution.

Out(1, (In2+(In3*Tan(pi*(Ran-(1/2))))))

Exponential Distribution (ExpDst.awf)

The Exponential Distribution produces numbers which have a much greater chance of occurring closer to 0. There's only one variable, A.

A = The "height" of the left end of the curve. The mean of the distribution is .69315/A. For example if A = 1, the mean will be .69315 and 99.9% of all results will fall below 6.9078. A can be as high as 20 or more, but best results seem to occur between 0.5 and about 8.

If you put the Exponential Distribution through an INT module, you'll get the Geometric Distribution.

Out(1, (-log(Ran)/In2))

Extreme LB Distribution (ExtrmLBDst.awf)

This is an Extreme Distribution - one that can have EXTREMELY high values,but it has a lower limit, or bound, set by A. There are three variables, A, B, and C.

A = Lowest value output. Can be 0, 1, or anything.

B = Scale, or "height" of the "hump." 0.5 is high and tight, 2 is lower and wider.

C = Shape. 2 = steep sides; 3 = shallower sides. C less than 1 will give VERY high values out. Be careful!

Out(1, In2+(In3*((-log(Ran))^-(1/In4))))

Gaussian (normal) Distribution

Do you want to be normal? Then this is the distribution for you! This is the classic "bell-shaped" curve beloved of pseudo-scientific thinkers everywhere. Now, you, too, can base your music on pseudo-science with this nifty distribution.

There are more values in the middle than at the ends. The middle is indicated by variable A, and the steepness of the slope by variable B.

A = location of middle of distribution.

B = steepness of slope of distribution. 0.1 = very narrow; 2= wider

C.(Ran+Ran+Ran+Ran+Ran+Ran+Ran+Ran+Ran+Ran+Ran+Ran)

Out(1, In3*(C-6)+In2)

Generalized Logistic Distribution (GnLgstcDst.awf)

This is a bell-like curve known as the Logistic Distribution, but with an additional parameter C, which enables you to set the skew of the bell-like curve.

C = 1, it's the normal logistic curve

C > 1 - it's right-skewed (higher values predominate)

C < 1 - it's left-skewed (lower values predominate)

Three variables, A, B, and C, with B & C > 0.

A = Centre of the "peak" of the hump.

B = height of the "peak" - 1 = lower; 0.5 = higher.

C = Skewedness of the shape. 1= symmetrical; >1=higher favored; <1 = lower favored.

Out(1, In2-(In3*(log((1/(Ran^(1/In4)))-1))))

Gumbel Distribution (GumbelDst.awf)

This is another left-skewed "hump" distribution with a controllable shape. It's an extreme value distribution. it can give very high values, because it's unbounded on the high side. It has two variables, A and B.

A = location - the "peak" of the "hump".

B = scale - 1 = higher; 2 = lower.

The Gumbel Distribution has many other names. It's also known as the LogWeibull, Gompertz, and the Fisher-Tippett distribution.

Out(1, In2-(In3*(log(-log(Ran)))))

Laplace Distribution (Bilateral Exponential) (LaplaceDst.awf)

This is a distribution which has a sharp curve, like the exponential distribution, but is symmetrical around a center point, A. The "height" of the center point is set by a variable B.

A = center point. If 0, you get both postive and negative values.

B = "height" of the "peak". 1= lower peak; 0.6 = higher peak.

This is known as the Laplace distribution, also as the double exponential distribution and the bilateral exponential distribution (called bilex in some programs).

DimLocal(2) ; make a local array to store -1 and 1 in

Store(0,-1) ; put -1 in the 0 position

Store(1,1) ; put 1 in the 1 position

C.(Fetch((2*Ran)>1)) ; put into variable C either a -1 or a 1

Out(1,In2-(C*(In3*log(Ran)))) ; C is the random sign in the equation

Linear Distributions (Lin2xDst.awf, Lin4xDst.awf, Lin6xDst.awf. LinInv2xDst.awf, LinInv4xDst.awf, LinInv6xDst.awf)

These are simple linear distributions which have a straight line slope. They consist of simply the lowest (for Linear) or highest (for LinearInverse) of N random numbers chosen simultaneously. In the Linear distribution, lower values have a greater chance of occurring than high values. In the Linear Inverse distribution, high values have a greater chance of occurring than low values. The number of random numbers in the function determines the steepness of the slope and the concentration on low or high values. These functions produce values between 0 and 1.

The demonstration patch for these functions allows you to hear the difference between them by moving a selection slider from left (extreme concentration on low values) to right (extreme concentration on high values.)

Any number of random numbers may be used to create you choice of slopes. You are encouraged to play around with these functions and develop your own slopes.

Out(1, Min(Ran,Ran,Ran,Ran,Ran,Ran))

Out(1, Max(Ran,Ran,Ran,Ran,Ran,Ran))

Pareto Distribution (ParetoDst.awf)

This distribution is always right skewed. That is, there will always be many more low values than high values. There are two variables, A and B.

A must be greater than 0. It's the lower limit of the distribution.

B must be greater than 0. It's the "height" of the low end of the graph. 1 = low. 2 = higher.

This distribution is sometimes used in economics as a measure of the probability of a particular income being above a given level. If you use this distribution, maybe your income will rise above that level! Then again, there might not be much probability of that.....:-)

Out(1, In2/(Ran^(1/In3)))

Reciprocal Distribution (RcprclDst.awf)

This distribution is always right skewed, and very steeply so. There will always be many many more low values than high with this. The shape of the curve is controlled by 2 variables, A and B.

A must be greater than 0. 1 is a reasonable value. It's the lower limit of output.

B must be greater than A. 60 is a reasonable value. It's the upper limit of output.

The Reciprocal distribution is often used to decribe 1/f noise.

C.(Ran)

Out(1, (In2^(1-C)*(In3^C)))

Triangle Distributions (Trngl2xDst.awf, Trngl4xDst.awf, Trngl6xDst.awf)

This is a distribution with straight slopes and a peak in the middle. That means that values in the middle will occur much more than values on either end. This version gives out values between 0 and 1. This function is simply the average of the sum of N random numbers. Here, we give you three flavors - the average of 2 random numbers, the average of 4 random numbers, and the average of 6 random numbers, so you can hear the different slopes and concentration on the center value that occurs. You are encouraged to make your own versions of these.

Out(1,(Ran+Ran+Ran+Ran+Ran+Ran)/6)

Weibull Distribution (WeibullDst.awf)

This is another useful distribution which can change shape based on its parameters. There are two variables, A and B.

A = the mean value. The "centre" of the distribution.

B = Shape. 0.5 = many more lows than highs. 1= sort of flat, but favoring lows. 3 = almost a bell-like curve.

With such a wide range of shapes, you can change B and get all sorts of results.

Out(1, In2*(log(1/(1-Ran))^(1/In3)))

Additional Functions

LEHMER Function.

The Lehmer Function is an equation used to generate pseudo random sequences in many computer programs, such as Fortran. But with values of A and B below 10, it can generate semi-repetitive patterns of interest. For example, is A+B < 1.0, then you get a single repeating value. If A+B = slightly more than 1, you'll get repeating patterns with some degree of randomness. As A + B get larger, you get closer and closer to a uniformly distributed pseudo-random sequence. The values in the demo patch for this (LehmerFunctionDemo.awp) will show you some of the possibilities.

BOREL Distribution.

This is not so much a distribution as a way of generating patterns. Over the long term, the distribution will be uniform, with an equal probability of any value occuring. But in the short term, the nature of the switching used will guarantee that you get patterns of ascending values clustering together.

The Borel Function and the Lehmer Function were written about by Charles Ames, in Leonardo Music Journal No. 2, 1992. His article also contains a number of other distributions you might want to program yourself.

CASE Statement [Macro]

This is a macro similar to the Probability Module, except here you can set your own probabilities in real time. Each of the four possibilities here has a setting from 0 to 100. These values are not absolute, but are proportional to each other. So, for example, if you had all four values set to 50 (or anything else), there would be an equal probability of any of the 4 possibilities occuring. If on the other hand, you had the four values set to 100 10 10 20, then the first possibility would occur 10 times as much as the 2nd and 3rd, and 5 times as much as the 4th possibility.

IFS 4 Variable

This is a crude 4 variable Iterated Function System for both graphics and sound. There are four variables. A single point, X is multiplied for both X and Y values, and transposed (moved) along both the X and Y axes. The values are divided by 1000, so 250 = .250 for example. There are four sets of possibilities (maps) in the separate columns. These are chosen between with the probability of any given column being selected being set by the probability numbers below the main grid. This module will repay the time spent experimenting with it. Some interesting graphics can be generated with it. There are many sets of rules for IFS fractals. Feel free to program your own. Peak and Frames "Chaos Under Control" is a good source of explanations for how these fractals work.

One Dimensional Attractors

There are a number of one dimensional attractor equations which generate interesting patterns. Three of these are the tent attractor, the sine attractor and the logistic equation. Each of these generates a series of numbers. The next member of the series is always generated by some mathematical operation on the current element of the series. In equation form it would look like this: nextX = currentX(changed in some way by an equation). So each current X is changed, and that value is the next member in the series. Then the next member is used as the current member, and a new next member is generated. This continues forever. For any equation like this, for each set of variables, there are 4 possible results. The equation will either, a) give a steady result, (ie, always the same answer), or b) go down to 0, or zoom away to infinity, or c) cycle around a series of the same or similar results, or d) give a series which is unpredictable to some degree.

For musical purposes, we're interested in the last 2 possibilities, cycling through a series, or having unpredicatable results. The tent attractor, the sine attractor, and the logistic attractor all do these with proper inputs. There are 2 variables in these equations, X (the number in our series), and S (the control number). For the tent attractor, if your X is between 0 and 1, and your S is between 1 and 2, you'll get a non-repeating series of numbers. However, the width of the range of those numbers, and whether they are fill the range, or have a hole in the middle of the range, will be determined by the number you choose for S. For values of S between 1 and 1.414 (the square root of 2) you'll get a series with a larger or smaller hole in the middle. This can be very useful in musical situations. For values of S between 1.414 and 2.0, you'll get non-repeating series of wider and wider range which have lots of interesting sudden small deviations in them, which sound very different from the output of a normal random generator. The sine attractor and the logistic attractor are more complex than the tent attractor. They have many areas where they have cycles of only a few repeating values, and other areas where the output sounds something like a random series. For the sine attractor, the area of interest is where S is between 2.5 and 3.14 (pi). The best results - ie, where you get a wide variety of repeating patterns and non-repeating series, happens when S is between 2.8 and 3.14 (pi). Similarly, with the logistic equation. The most useful area for that equation is when S is between 3.5 and 4, with the best results at the higher end of that range. The attached diagram (3Attractors.bmp) shows the "maps" of each equation - that is, what kind of series you get for each value of S. All three produce different kinds of patterns, and all three might be useful in a wide variety of musical settings.

Experimental Generators of Unpredictability

In Addition to the Probability Distributions, and the Three Attractors (tENT, Sine and Logistic), I've also enclosed two other generators of unpredictability that might amuse you. They are both very experimental (meaning that I've left a lot of inelegant operating things in them), and you should explore them and use them both carefully (finding ways of limiting their outputs to the ranges you want) and with abandon (full steam ahead captain! Ignore those danger warnings!)

The first of the two Experimental Generators of Unpredictability is a very funky 2 output version of the **Henon attractor.** It uses the Henon attractor equation in James Gleick's classic book on Chaos. The puck in the XY pad can be moved around, creating new X and Y values for the attractor to move from. Some of these will produce values out of range. You'll have to explore the output of this equation and it's input with the position of the puck to see how it works. There are more predictable versions of the Henon equation, but this one has it's own charm.

The second of the two Experimental Generators of Unpredictability is a very mutant **shift register feedback** patch. The "Burt Shift" fractal function is a much more controlled version of this. There are 3 parameters in this macro. The first is the number of bits to shift - it must be less than 31. The second is 2 for left rotate, 3 for right rotate. The 3rd is the number to XOR the feedback by. Try all sorts of values here. My hunch is that primes will work best.

The Demo patch for this has an interesting feature: Delay Step modules are used in order to set up a system where delayed values of the Shift Register Feedback patch are used to control different parameters. So in this patch

The SRFBK raw output controls pitch.

The SRFBK output delayed one step controls duration.

The SRFBK output delayed two steps controls velocity.

These sorts of delay line chains of values, graphed against themselves, are used in chaos investigations to reveal the structure of attractors. Here we use them to shape different aspects of a melody. However, you shouldn't be afraid to use them to control parameters for any sonic, visual or language process you can think of.

Package: WBurt

Additive Sequence Generators

by Warren Burt

An additive sequence is one in which a rule is applied for adding seed values. This rule is then repeated again and again, making an infinitely long series of numbers.

The most familiar additive sequence is the Fibonacci Sequence. The rule is that each element in the sequence is the sum of the two elements before it. Starting with the seed 1,1, the beginning of the sequence looks like this:

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610.....

Notice how each element of the sequence is the sum of the two elements before it. For example 144 = 89 + 55, 610 = 377 + 233, etc.

The rule for this sequence can be written as

A[n] = A[n-2] + A[n-1]

(Each number is the sum of the number 2 elements before it and the number 1 element before it. The [n] or [n–2] etc. indicates the position of the element in the sequence.)

If you use the seed 1, 1 with this rule, you get the classic Fibonacci series. If you use the seed 2, 1 with this rule, you get a series known as the Lucas numbers.

This package of sequence generators for ArtWonk allows you to have access to the following sequence rules.

A[n] = A[n-2] + A[n-1]

B[n] = B[n-3} + B[n-1]

C[n] = C[n-3] + C[n-2]

D[n] = D[n-4] + D[n-1]

E[n] = E[n-4] + E[n-3]

F[n] = F[n-5] + F[n-1]

G[n] = G[n-5] + G[n-2]

H[n] = H[n-5] + H[n-3]

I[n] = I[n-5] + I[n-4]

J[n] = J[n-6] + J[n-1]

K[n] = K[n-6] + K[n-5]

The Macros are named after these sequence names and their element places. So, for example, “AddSqA21" is the macro to use for rule A, A[n] = A[n-2] + A[n-1], while “AddSqH53" is the macro to use for rule H, etc.

Inputs:

Strobe: Input for Strobe to get the next element in the sequence.

Seed 1, 2, n: Inputs for the Seeds for the sequence. Each sequence has a different length seed, depending on how many elements back in the series the addition takes place. That is, “AddSqJ61" (rule J[n] = J[n-6] + J[n-1]) has a 6 element long seed, while rule A only has 2 seeds.

MOD: Input for Modulo input. To make the sequences manageable, this allows you to count the sequence modulo N, that is, when the value of the sequence reaches a certain limit, set by MOD, you do a modulo division on the sequence from that point on.

Output:

Out: The output of the chosen sequence.

Notes on the Seed Inputs:

Although simple seeds like 1, 1 (for A) or 1, 0, 0, 0, 0, 1 (for K) will produce simple sequences which start at low values, it should be emphasized that you can use ANY number for these inputs. Explore!

Note that since this uses a “Latest” module to select either this input or the internal feedback in the Macro, if you want to re-run the sequence with the same values, you’ll have to reload those values afresh into the Seed Inputs.

Notes on the MOD Input:

Again, you can use any value you like for the MOD input. Setting the MOD input to a very high value, like 5000, allows you to see the beginning of each sequence unchanged for quite a while. Setting the MOD to a low value, like 12, gives you repeating sequences or varying lengths.

For the seed 1, 1, here are the length of sequences you’ll get for MOD settings from 1 to 24, and 60 and 61. For the seed 2, 1, the sequence lengths appear to be the same for each MOD value. However, for MOD60, different seed values did indeed produce different length sequences. So clearly, there is much exploration to be done here, to see what combinations of Seeds, Rules, and MOD values produce sequences which are of interest to you. Have fun!

For seed 1, 1:

Modulo N	Seq. Length
1	0
2	0
3	8

4	6
5	20
6	24
7	16
8	12
9	24
10	60
11	10
12	24
13	28
14	48
15	40
16	24
17	36
18	24
19	18
20	60
21	16
22	30
23	48
24	24
60	120
61	60

[end]