The Smoothing Parameter in Music Kit Envelopes

by Julius O. Smith

NeXT Inc.
July 11, 1991

Introduction

Definition of an Envelope

A Music Kit "envelope" is a function of time specified by a list of "breakpoints". Each "breakpoint" is specified by two or three numbers. In the two-number case, you specify the time of the breakpoint and desired value of the envelope at that point. The envelope itself is a piecewise exponential function. The desired value at each breakpoint is approached exponentially from the previous breakpoint. Exponential functions are discussed below.

Definition of the Envelope Smoothing Parameter

The optional third parameter in a breakpoint specification is called "smoothness". It controls the relaxation time-constant used in approaching each "target value" in the envelope. The actual definition is the following:

Smoothness is defined as the number of repetitions of the time interval to the next breakpoint needed to reach that breakpoint's value.

Thus, the most natural choice of smoothness S is S=1. This means the envelope will just about reach the next envelope breakpoint value at the time specified for that breakpoint. Setting S=0 means the envelope will instantly jump to the next breakpoint. Setting S=2 means the envelope will about reach the next breakpoint value at two times the distance to the next breakpoint; this is "smoother" than S=1. It also means that using values of S larger than 1 means setting breakpoint target values which are larger (when going up) than you actually expect to reach. In fact, if there were no maximum number limit on the DSP, you could take S to infinity and get a piecewise linear envelope.

Exponential Envelopes

The Exponential Decay

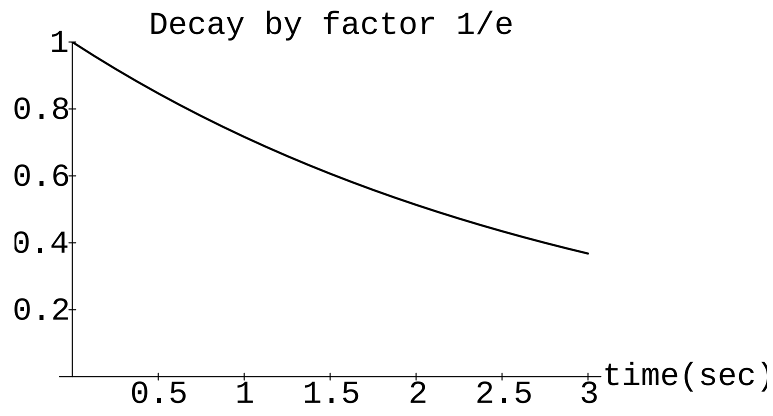
First lets look at an exponential decay. This is like the end of an envelope when it is decaying to zero.

```
Clear[f1,A,tau,t];   (* Clear any previous definitions *)
f1[t_] := A Exp[-t/tau];  (* tau = exponential time constant *
```

For t>0, f1[t] is the function used for the final decay of every envelope, where A is the value of the envelope when the decay starts (nominally the value of the last nonzero breakpoint)..

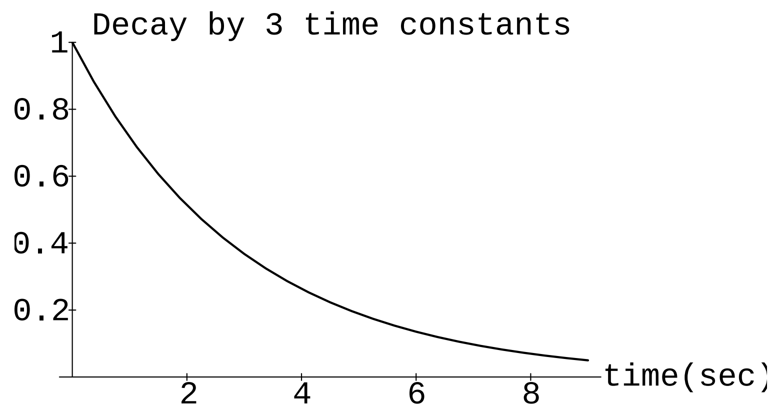
☐ Example Plots

```
A = 1;          (* A = amplitude (units arbitrary) *)
tau = 3;        (* time constant = 3 seconds *)
Plot[f1[t],{t,0,tau},
    PlotRange->{0,A},
    AxesLabel->{"time(sec)",""},
    PlotLabel->"Decay by factor 1/e"];
```



It takes several time constants to decay really close to zero:

```
Plot[f1[t],{t,0,3*tau},
    PlotRange->{0,A},
    AxesLabel->{"time(sec)",""},
    PlotLabel->"Decay by 3 time constants"];
```



■ T60

In audio, a reasonable definition of the end of an exponential envelope is when the final decay drops the level by 60 decibels (dB). This decay time for exponentials is called t60 in the reverberation literature. Since Amp_dB = 20 Log[10,Amp], this means we have the formula

☐ Derivation of t60

```
Clear[A,tau,t60];
-60 == 20 Log[10,f1[t60]/A]
```

```
                -(t60/tau)
       20 Log[E           ]
-60 == ----------------------
             Log[10]
```

We divide by A because we are interested in the decay factor, not the absolute final value. Solving the above equation for t60 can be done as follows. First divide both sides by 20, then raise 10 to the value on each side to get

```
0.001 == f1[t60]/A
```

```
           -(t60/tau)
0.001 == E
```

```
Solve[%,t60]
```

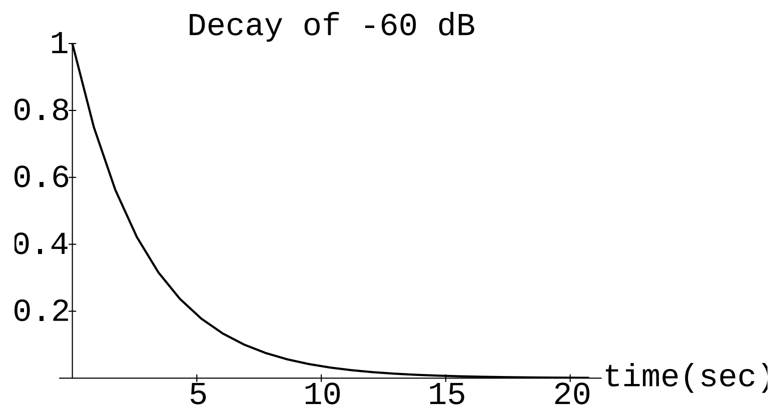
```
Solve::ifun:
   Warning: inverse functions are being used by Solve,
    so some solutions may not be found.
```

```
{{t60 -> 6.90776 tau}}
```

Thus, t60 is about 7 time constants, or 6.91 tau to 3 digits accuracy.

☐ Pictures of 60 dB Decay

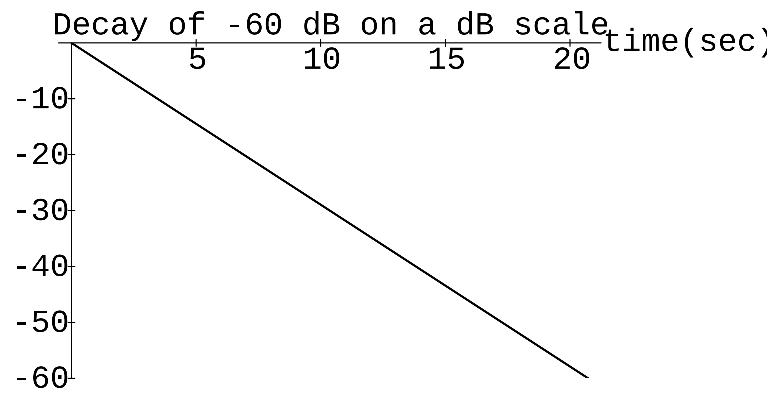
```
A = 1;
tau = 3;
t60 = 6.91 tau;
Plot[f1[t],{t,0,t60},
    PlotRange->{0,A},
    AxesLabel->{"time(sec)",""},
    PlotLabel->"Decay of -60 dB"];
```



Same thing on a dB scale:

```
Plot[20 Log[10,f1[t]],{t,0,t60},
    PlotRange->{-60,20 Log[10,A]},
    AxesLabel->{"time(sec)",""},
    PlotLabel->"Decay of -60 dB on a dB scale"];
```

```
General::dby0: Division by zero.
```



Exponentials which Approach a Non-Zero Asymptote

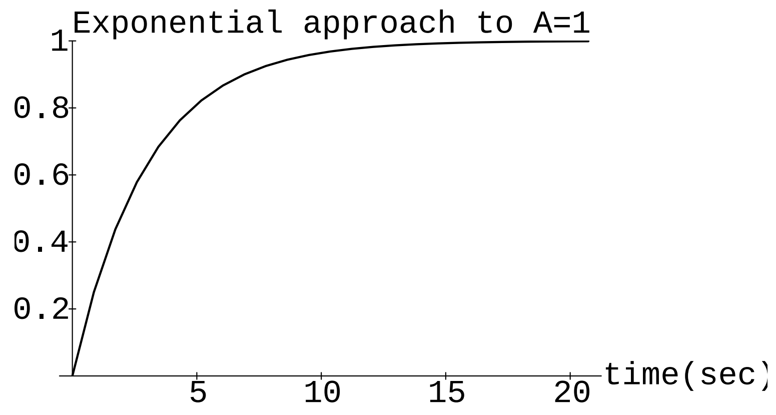
Note that the function

```
Clear[A,tau];
f2[t_] = A-f1[t]
```

```
        A
A - ---------
      t/tau
     E
```

approaches A starting from 0:

```
A=1;
tau=3;
Plot[f2[t],{t,0,t60},
    PlotRange->{0,A},
    AxesLabel->{"time(sec)",""},
    PlotLabel->"Exponential approach to A=1"];
```



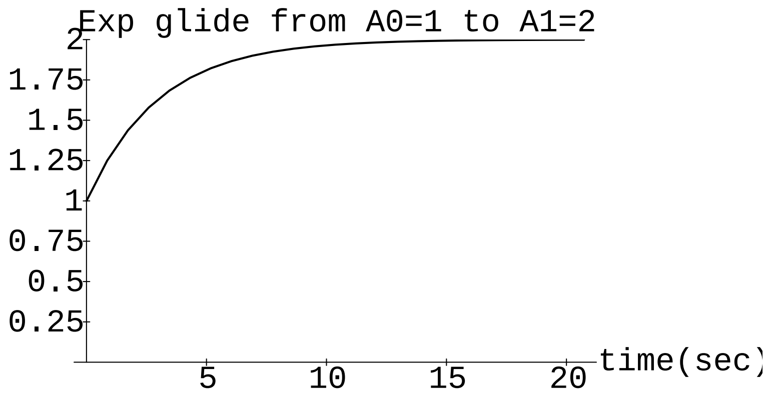
When approaching a constant breakpoint value A1 given an initial value A0 at time 0, the formula to use is like the one above, but adding in the initial value A0, and using A1-A0 (the amplitude difference) in place of A:

```
Clear[A,tau];
A = A1-A0;
f3[t_] = A0 + f2[t]
```

```
     -(t/tau)
2 - E
```

Let’s plot this for some sample values:

```
A0=1;
A1=2;
tau=3;
Plot[f3[t],{t,0,t60},
    PlotRange->{0,A1},
    AxesLabel->{"time(sec)",""},
    PlotLabel->"Exp glide from A0=1 to A1=2"];
```



A Music Kit envelope is put together by concatenating segments such as the above Actually, the segments are not exactly concatenated. The asymptote gets changed each breakpoint, so the envelope just starts approaching the new asymptote from wherever it happens to be.

Sampled Exponentials

Replacing t by n T

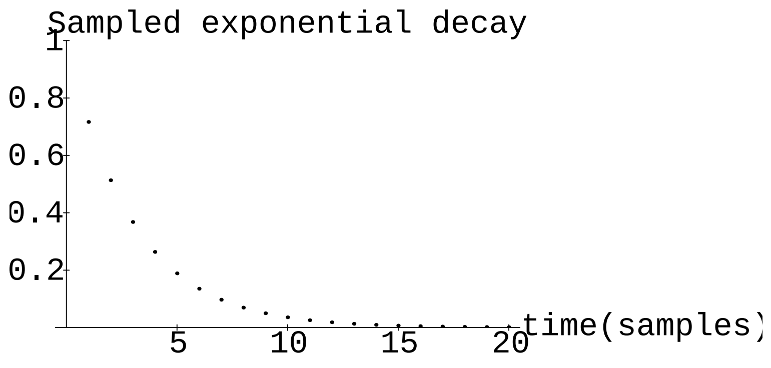
A digitally sampled exponential is obtained by replacing t with n T, where n is the sample number and T is the sampling interval in seconds. The sampling-rate is denoted fs. Thus,

```
T = 1/fs;      (* Corresponding sampling interval *)
```

```
Clear[A,tau,T,n];
t = n T;
f1d[n_] = f1[t]    (* Sampled exponential decay *)
```

```
    A
-------------
 (T n)/tau
E
```

```
Clear[t];
A=1;
tau=3;
T=1;
ListPlot[Table[f1d[n],{n,t60/T}],
    PlotRange->{0,A},
    AxesLabel->{"time(samples)",""},
    PlotLabel->"Sampled exponential decay"];
```



■ Sampled Exponentials are Geometric Sequences

Note that Exp[-n T / tau] can be written as Exp[-T/tau]^n.
Define

```
Clear[A,T,tau,g];
g = Exp[-T/tau];
```

Now our basic exponential looks like

```
f1d[n] = A g^n
```

```
    A
-------------
 (T n)/tau
E
```

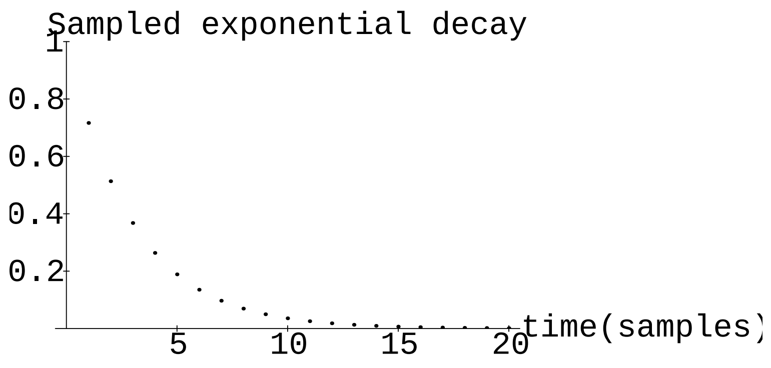
This is the function used by the AsympUG unit generator to compute the samples of an exponential. Note that it can be computed recursively via f1d[n] = g f1d[n-1]. The form f1d[n] = g f1d[n-1] + (1-g) A gives an exponential approach to the value A (show this!). This latter recursion is what the AsympUG DSP code actually does each sample, except that it uses the variable r = 1-g = 1-Exp[-T/tau], called the "rate". Thus, the recursion used by asymp.asm is

f1d[n] = (1-r) f1d[n-1] + r A

to approach the value A exponentially with time-constant tau, where r = 1-Exp[-T/tau]. The rate is so named because when it is at its maximum value of 1 (tau = 0), A is reached in one sample step, as substitution in the above recurrence immediately shows. When r is at its minimum value of 0 (tau = infinity), the exponential never gets to the asymptote A. The source code for AsympUG is in /usr/local/lib/dsp/ugsrc/asymp.asm.

Let’s prove its the same function as before:

```
A=1;
tau=3;
T=1;
ListPlot[Table[f1d[n],{n,t60/T}],
    PlotRange->{0,A},
    AxesLabel->{"time(samples)",""},
    PlotLabel->"Sampled exponential decay"];
```



The Music Kit Envelope Smoothness Parameter

T48

Instead of t60, the Music Kit uses "t48" which is the time it takes to decay -48 dB from the initial starting value (in the case of final decay toward zero).

```
t48taus = -N[Log[10^(-48/20)]] (* "t48" in time-constants *)
```

```
5.5262
```

Thus, 5.53 time-constants is about 48 dB of decay.

```
-20 Log[10, Exp[ - t48taus tau / tau]] (* test it out *)
```

```
48.
```

Relating Smoothness to Time Constant or Rate

Recall that smoothness is defined as the number of breakpoint intervals to reach the next breakpoint value. In this context, "reaching" the next breakpoint value is defined as traversing t48 seconds along the exponential which is approaching that breakpoint value. Let d denote the time between the current breakpoint and the next breakpoint. Then, for S=1, d must be t48. For S=2, we want d to be 1/2 of t48, and so on. Thus, the fundamental relation relating smoothness to exponential decay is

d S = t48

We found above that t48 was about 5.5 time constants. Thus,

d S = 5.5262 tau

(to 4 digits). Since tau is related to r by the formula r = 1-g = 1-Exp[-T/tau], we can solve for the AsympUG rate r in terms of S to get

r = 1 - Exp[-5.5262 / (d S)]

where d is the distance to the next breakpoint in seconds, and S is the smoothing parameter. Since the default value for S is 1, we see that increasing it to 2 corresponds to taking the square root of g so that r goes from 1-g to 1-Sqrt[g]. Since g is between 0 and 1, Sqrt[g] is always larger than g, so r (the "rate") is made smaller by increasing smoothness. This actually makes sense!

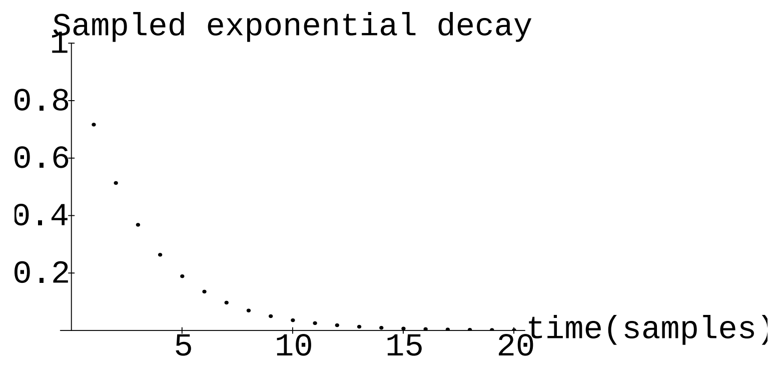
■ What AsympUG Does

As mentioned above, AsympUG generates an exponential envelope segment using the recursion

f1d[n] = g f1d[n-1] + r A

Let’s demonstrate this recursion for the following example:

```
A = 1;
tau=3;
T=1;
ListPlot[Table[f1d[n],{n,t60/T}],
    PlotRange->{0,A},
    AxesLabel->{"time(samples)",""},
    PlotLabel->"Sampled exponential decay"];
```

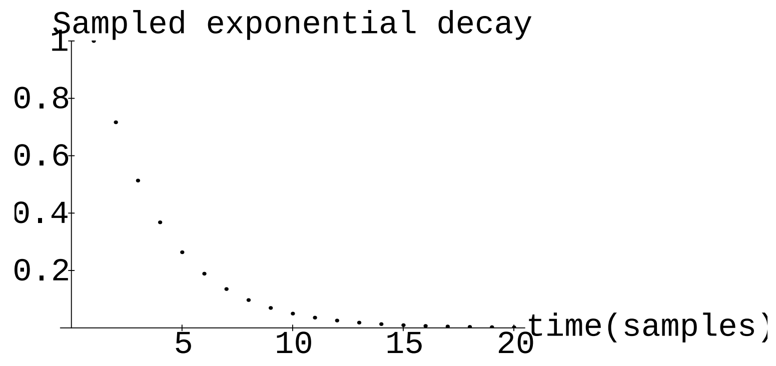


Now we'll compute the same samples using AsympUG's recursion:

```
g = N[Exp[-T/tau]];
r = 1-g
```

```
0.283469
```

```
npoints = Floor[t60/T];
env = Table[0,{npoints}];
Af = 0.0;   (* Approached amplitude [experiment!] *)
env[[1]]=1.0;   (* Initial segment amplitude *)
For[n=2,n<=npoints,n=n+1,
    env[[n]] = g env[[n-1]] + r Af];
ListPlot[env,
    PlotRange->{0,Max[env[[1]],Af]},
    AxesLabel->{"time(samples)",""},
    PlotLabel->"Sampled exponential decay"];
```


Sampled exponential decay
time(samples)