File: /General/MLAB-Text/Papers/adaptive/adaptive.tex

Adaptive Delta Modulation

Gary D. Knott, Ph.D.
Civilized Software, Inc.
12109 Heritage Park Circle
Silver Spring MD 20906
Tel.: (301)-962-3711
email: csi@civilized.com
URL: www.civilized.com

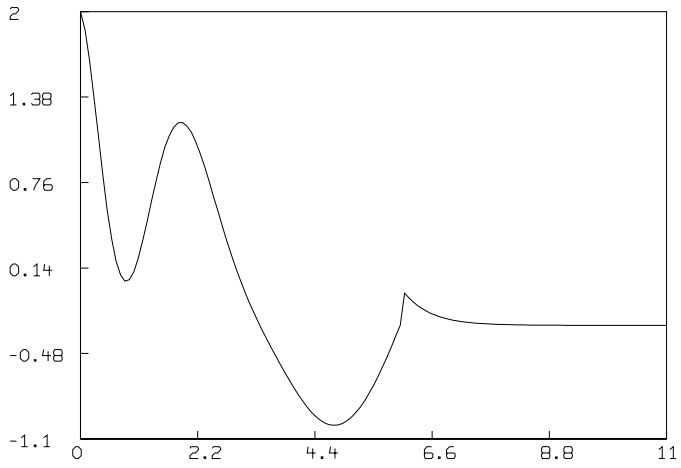
Given an analog signal $f(t)$, for $s_0 \leq t \leq s_1$, we may wish to transmit this curve to a remote site. For example, f may be a voice signal, or a continuous time series measurement reading of some transducer. Digital encoding generally allows data compression, permitting fewer bits per second to be transmitted, and error-correcting codes can be used as desired to further improve accuracy. We may transmit a train of rectangular pulses, using repeaters as needed, or we may use any of the various amplitude, frequency, or phase modulation methods for transmission as a band-limited oscillating signal. Digital encoding for data compression is also a useful device in storing digitally-encoded curves.

One common scheme is so-called delta-modulation encoding. This method can be used for sound and/or video digital radio transmissions and recordings as well as other types of signals. If the signal, f, is to be digitized for computational purposes, then delta-modulation encoding serves as a cheap and efficient analog-to-digital converter.

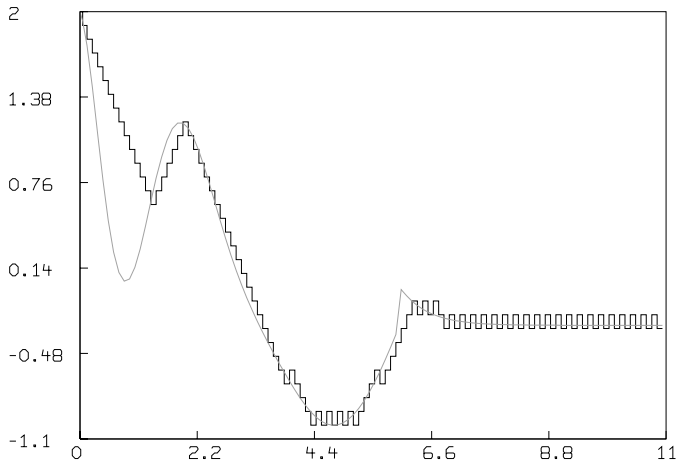
The basic idea is quite simple. Given the initial value, $h = f(s_0)$, a sampling interval, α, and an increment value, δ, we may interpret a string of bits b_1, b_2, ..., b_n to obtain a specification of estimated signal values $\hat{f}(t_1)$, $\hat{f}(t_2)$, ..., $\hat{f}(t_n)$, where $t_i = s_0 + i\alpha$ for $0 \leq i \leq n$, as follows. Each bit b_i indicates adding the constant δ if $b_i = 1$ and subtracting δ if $b_i = 0$, so that

$$\hat{f}(s_0 + k\alpha) = h + \sum_{i=1}^{k} (2b_i - 1)\delta.$$

For example, consider $f(t)$ =if $t \leq 6$ then $\sin(t) + 2\cos(3t)\exp(-t)$ else $-.27614(1 - \exp(-2(t-6)))$. A graph of f on the interval $[0, 11]$ is shown below



Choosing the parameter values: $s_0 = 0$, $s_1 = 11$, $\delta = .1$ and $\alpha = .1$, we have $h = 2$ and the binary digital sequence: 00000000000000111111000000000000000000100001010 01010101010101010101010101010101010101. This sequence specifies $\hat{f}$ as shown below, drawn with the traditional step-function interpolation which would be seen on an oscilloscope. The smooth curve is the true signal, and the stepped curve is the step-function form of $\hat{f}$. Thus the function $\hat{f}$ on the interval 0 to 11 is represented with 110 bits.



The encoding process computes the ith bit, b_i, by predicting $f(t_i)$ to be some value P_i, and then b_i is taken as one if $f(t_i) > P_i$ and zero otherwise. The choice of P_i used here is merely the previous estimate value $\hat{f}(t_{i-1})$.

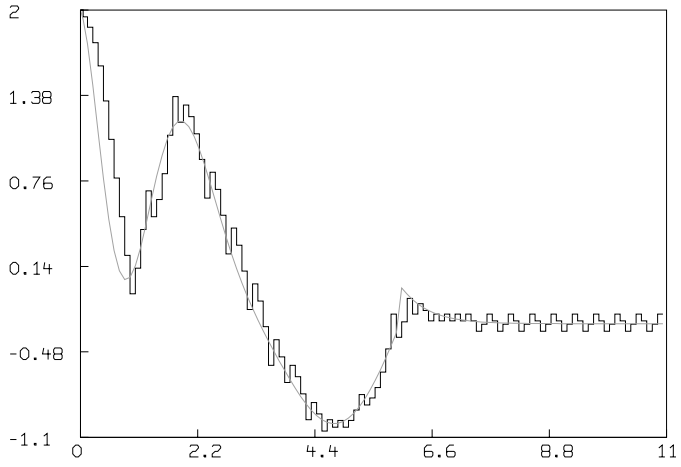
In general, to obtain a reasonable estimate for f, the sampling interval, α, must be such that $(s_1 - s_0)/\alpha$ is greater than the Nyquist frequency, which is twice the frequency of the highest-frequency component present in the signal f, and δ must be small enough to track high-frequency oscillations without undue shape distortion. A practical way to choose α and δ is to choose δ as the largest absolute error by which $\hat{f}$ may deviate from f, and then choose $\alpha = \delta/w$, where $w = \max_{s_0 \le t \le s_1} |df(t)/dt|$, the maximum absolute slope of f. In the previous picture, α is clearly too large for the chosen δ; the result is the large error in the initial part of the signal, called slope overload error, where the slope is too large in magnitude for $\delta = .1$ and $\alpha = .1$ to track the signal.

Even when the signal is being appropriately followed, the estimate oscillates about it. This so-called granular noise is unavoidable, although its size is controlled by δ. Note that the error characteristics of the estimator $\hat{f}$ are given by $|f(t) - \hat{f}(t)| < \delta$ for $s_0 \le t \le s_1$, assuming α is small enough. This is an absolute error criterion rather than a relative error criterion, and $\hat{f}$ behaves like a Chebychev approximation to f.

Note that a delta-modulation signal is very sensitive to transmission error. Changing a burst of a dozen bits or so during transmission can destroy the validity of the remaining bits. However, higher sampling rates mean

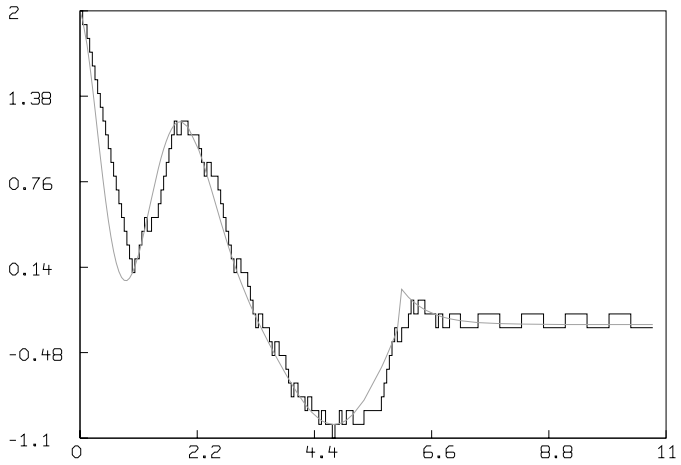
short burst errors are less harmful, and methods to periodically restart the delta-modulation process can be included in a practical transmission system. In general, delta-modulation is a very efficient way to encode a signal. It is not clear how to define the notion of the efficiency of an approximation (as opposed to an exact encoding) in a precise information-theoretic manner, but this is an intriguing direction for investigation.

We can elaborate on the basic idea of delta-modulation in several ways. First, it has been proposed that the increment δ can assume various values, depending upon the past tracking of the signal. If we output m ones or zeros in a row (indicating a region of large absolute slope), we can increase δ, replacing δ with $3\delta/2$ for example. If we output m alternating ones and zeros, we can then decrease δ, say to $2\delta/3$. The new value of δ is to apply to the current bit being output which forms the mth bit of the change-triggering pattern. This device is called adaptive delta-modulation. Changing δ, however, is not always an improvement. Indeed the signal may be such that a closely-tracking, but lagging, estimate becomes an oscillating estimate with greater absolute error when adaptive delta-modulation is employed. For example, for the signal shown above, with $\alpha = .1$ (too large), and δ varying within the limits .05 to .28, based on $m = 2$, so that two zeros or ones in a row increases δ, while two different values decreases δ, we obtain the approximation shown below.

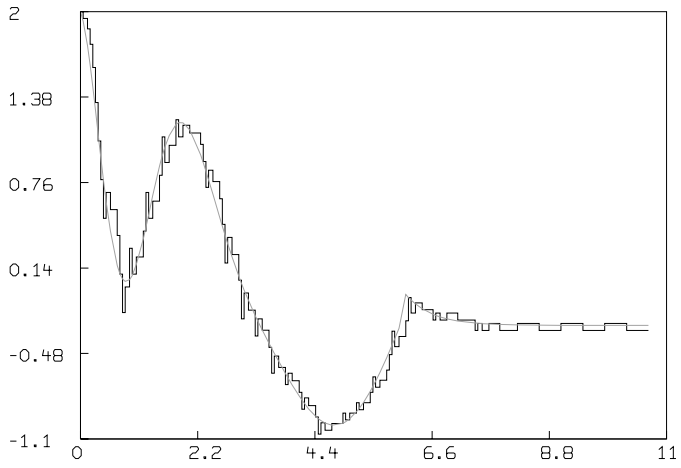


Another idea is to allow the sampling interval, α, to change. This is not

very useful for hardware, which is more conveniently designed to use a fixed clock rate, but for data-compression for digital storage purposes, varying α may allow fewer bits to be used to encode a given curve. We may increase α when the previous m bits have alternated in value, but when m ones or zeros in a row occur we reduce α to re-establish the fastest sampling rate. This permits large steps in slowly-varying regions, but it allows relatively large deviations in the estimate to occur at turning points where f changes from being flat to sharply rising or falling. Choosing $m = 2$ minimizes this effect, but it is still noticable. Lagging tracking at turning points is the major flaw in most delta-modulation schemes. The step-function estimate of our example signal is shown below where we have replaced α by 1.6α up to a limit of .41 whenever the previous two bits were the same and reset α to .05 otherwise. We have fixed $\delta = .1$ (which is too small to be completely reasonable for our range of α). Note we now have as our estimated points: $(s_0, f(s_0)), (t_1, \hat{f}(t_1)), ..., (t_n, \hat{f}(t_n))$for some irregular mesh$s_0 < t_1 < ... < t_n \leq s_1$.



If we allow δ and α to both vary as described above with δ in the interval $[.05, .28]$ and α in the interval $[.05, .41]$, we obtain the following approximation.



In order to compute the bit b_i which determines the point $(t_i, \hat{f}(t_i))$ when encoding the signal, f, we form an estimate of $f(t_i)$, called P_i, where P_i predicts $f(t_i)$ given the previous estimate points $(t_0, \hat{f}(t_0)), (t_1, \hat{f}(t_1)), ..., (t_{i-1}, \hat{f}(t_{i-1}))$. Then if P_i is less than $f(t_i)$ we output $b_i = 1$ and otherwise for $P_i \geq f(t_i)$, b_i is output as zero.

This same predictor must be used in decoding the bitstring, b, in order to compute $\hat{f}$; this is why P_i depends on $\hat{f}$-values, and not on f-values.

In the discussion above, we have used the simple predictor $P_i = \hat{f}(t_{i-1})$. Other predictor schemes are possible, and may provide better performance, allowing smaller δ and/or larger α values to be used. Of course other predictors do not necessarily have the property that $b_i = 1 \text{iff} \hat{f}(t_{i-1}) < f(t_i)$.

In general, then, our decoding calculation for obtaining $\hat{f}(t_i) \text{is} \hat{f}(t_i) = P_i + \delta_i(2b_i - 1) \text{for} 1 \leq i \leq n$, where δ_i is the particular value of the increment used when b_i is computed; δ_i is a function of $b_1, b_2, ..., b_{i-1}$.

We could choose P_i to be determined by extrapolation on the linear or quadratic curve passing through the previous two or three points of the estimate, but experimentation shows the error in this form of predictor to be worse than the simple constant form. A more promising idea is to use an extrapolation on a polynomial which fits the previous k estimate points in the least-squares (or better yet L^1-norm) sense. This works adequately although the predictor is slow to track f on turning intervals. A more elaborate filter predictor might be worth exploring, but weighted averages of the previous k points and weighted averages of the linear predictors based

on the k previous points taken two at a time also perform no better than the simple constant predictor. Thus finding a better predictor seems difficult and the constant form seems to be the best practical choice as well as the simplest.

If a good predictor is used, however, the adaptive variation of δ and α becomes less useful. Indeed, with a perfect predictor $P_i = f(t_i)$, our output is all zero bits, and α will foolishly stay at its minimum, while δ will stay at its maximum. The resulting $\hat{f}$ curve tracks below f with the maximum allowable error. Even using merely a good predictor means we should sharply decrease δ and perhaps increase α.

Algorithms for encoding and decoding an adaptive delta-modulation signal are presented below, with $m = 2$ and with the simple constant predictor function, $P_i = \hat{f}(t_{i-1})$, used above. The constants $C, \delta_{min}, \delta_{max}, g, \alpha_{min}$, and$\alpha_{max}$ are global parameters which may be "tuned" to the situation. We have used $c = 1.5, \delta_{min} = .05, \delta_{max} = .28, g = 1.6, \alpha_{min} = .05$, and$\alpha_{max} = .41$ in the example above.

```
ADMencode(s0, s1, d, a, f):
begin  real x1,x2,t,h,b;
real procedure P: [h <-h+(2b-1)d; return(h)];
b,x2.5; ts0; hf(s0);
send (t0, h,d, a) to ADMdecode; "This send starts the decoder."
while t <=  s1 do
      [tt+a; x1x2; if f(t) > P then b1 else b0;
       x2b; output(b);
       if x1=x2 then (aamin; dmin(cd,dmax))
       else (dmax(d/c,dmin); amin(ga,amax))
     ];
output(end-message); "This will stop the decoder."
end.
```

```
ADMdecode(t, h, d, a):
begin  real x1,x2,h,b;
real procedure P: [hh+(2b-1)d; return(h)];
b,x2.5;
repeat
     [output (t,P);
      "Initially, the output point (t,P) is the point (s0,(s0));
```

```
        then subsequent points on the -curve are output."
       receive a bit in b, on end-message, goto exit;
       x1x2; x2b; tt+a;
       if x1=x2 then (aamin; dmin(cd,dmax))
       else (dmax(d/c,dmin); amin(ga,amax))
      ];
exit: end.
```

There are many variations possible, based on different ranges for δ and α, and different formulas for changing them. In our example, we actually do about as well with even fewer bits than used above when we let δ assume just the values .1 and .2 and let α assume just the values .1 and .2. Another idea is to compute b by the test: if$f(t) > P - a(log(1 + |f'(t)|)/k)$then$b \leftarrow$ 1else$b \leftarrow 0$. This use of slope information can perhaps be marginally useful, even though it produces "lies" about the location of f. Some suggestions have been made that an "intelligent" encoder could hold m signal values in a memory, and carefully compute the best block of one or more output bits based on this "look-ahead". Perhaps just provisional output bits could be held, so that we would hold m bits in a local memory and output each bit based upon the $m - 1$ future bits which have been provisionally computed and stored, but it seems difficult to make a useful scheme out of this idea.

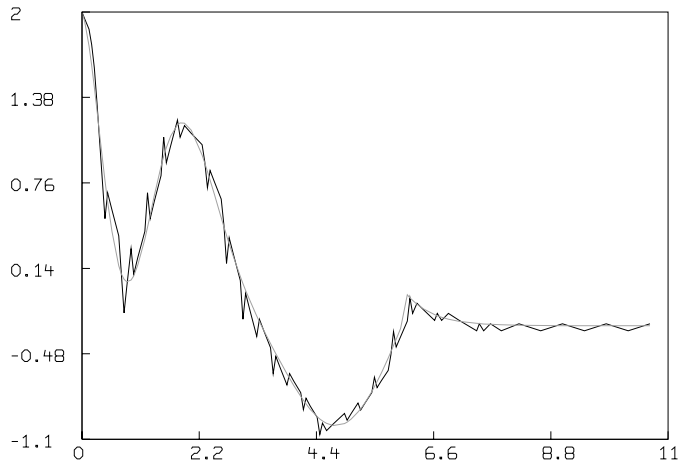
Also, when we use $m > 2$ to adaptively change δ and/or α, we could use a $2^m -$ bit decision tree to pick carefully-chosen δ and α modifications; this scheme does work well, but at a high cost in complexity.

All the pictures in this example were produced with MLAB. The precise statements needed to do such computer experiments with MLAB are shown below. The following text is a do-file, which when invoked with an appropriate MLAB do-command, is executed to produce matrices suitable for graphing.

```
FUNCTION F(T)=IF T<=6 THEN SIN(T)+2*COS(3*T)*EXP(-T) ELSE J-J*EXP(-2*(T-6));
J = -.27614;MINA = .05; MAXA = .41; MAXD = .28; MIND = .05;
T0 = 0; T1 = 11; D = MIND; A = MINA; G = 1.6; C = 1.5;
TYPE "SET T0,T1,A,D, ETC. AS DESIRED."; DO CON;

FUNCTION ADM(I)=
IF T+A<=T1 THEN (B_((PV_P(ME[I]_OLDP(ADM)))<=F(MT[I+1]_(T_T+A))))
 +0*(A_NEWA(X1_X2,X2_B))+0*(D_NEWD()) ELSE 1000-I;
```

```
FUNCTION OLDP(B)=PV+D*(2*B-1);
FUNCTION P(H1)=H1;

FUNCTION NEWA(X1,X2)=IF X1=X2 THEN MINA ELSE MIN(G*A,MAXA);
FUNCTION NEWD()=IF X1=X2 THEN MIN(D*C,MAXD) ELSE MAX(D/C,MIND);

X2 = 1; ADM = .5; T = T0; IF T1 <= T0 THEN TYPE ("null interval");  PV = F(T0);
"PRE-ALLOCATE THE ARRAYS MT, ME.";
MT = 0^^360; ME[360] = mt; MT[1] = T0;

MB = ADM ON 1:360;
N = MAXI(MB); IF N >=360 THEN TYPE "TOO MANY POINTS.";

ME(N) = OLDP(B); ME = ME ROW 1:N; MT = MT ROW 1:N; MB = MB ROW 1:(N-1);
SME = MESH(MT,MT)&'ROTATE(MESH(ME,ME),1); DELETE SME ROW (1,N+N);

ME = MT&'ME; MF = POINTS(F,MT);
"MB IS THE DELTA-MODULATION BIT-VECTOR, MF IS THE SAMPLED
SIGNAL POINTS, ME IS THE ESTIMATED SIGNAL POINTS, AND SME
IS THE STEP-FUNCTION ESTIMATE.";
```

It is worth observing that the step-function approximation form of drawing $\hat{f}$ is somewhat deceiving. A simple straight-line interpolation is a more reasonable presentation. For example, the (δ, α)-varying estimate shown above is seen again here using linear connecting segments.



Viewing a picture such as this suggests that we might estimate f more exactly if we compute the midpoints of the line-segments in the graph of f which cross the graph of $\hat{f}$. But this set of crossing points is only marginally useful when filtering is used. Generally the input, f, should be prefiltered to pass only frequencies below an appropriate cutoff point. In any event, the output points, $(t, \hat{f}(t))$, have both sampling and encoding error, and the output should be filtered to remove some of the noise. The filtering can be done with a distance-weighted smoothing transformation in software, or with an appropriate low-pass filter in hardware.

The smoothed variable δ and α estimate is shown below. A doubly-smoothed estimate would be even better in regions of slowly-changing slope.