SIMON FRASER UNIVERSITY
School of Engineering Science

ENSC 428 Data Communications

Assignment 5 **Semester 01-1**

1. Linear Equalizer

(a) Because three data bits are represented in each matched filter samples, we first have to determine which one we are trying to detect. Recall that the impulse response is

$$x_0 = 0.1 \qquad x_1 = 1 \qquad x_2 = -0.3$$

so that the samples are $y_k = \sum_{i=0}^{2} a_{k-i} \cdot x_i + \nu_k$

The strongest sample in $x(k)$ is x_1, and it exceeds the sum of absolute values of the other samples, so we'll use it to pick off the data. That is, we use y_k to determine a_{k-1}.

To determine BER, note that the noise-free value of the samples (i.e., the expected values over the noise ensemble) depends on the data, so there are 8 possible values. By symmetry, we can concern ourselves only with $a_{k-1}=1$, reducing the noise-free values to the following:

a_k a_{k-1} a_{k-2} y_k

$$\begin{bmatrix} -1 & 1 & -1 & 1.2 \\ -1 & 1 & 1 & 0.6 \\ 1 & 1 & -1 & 1.4 \\ 1 & 1 & 1 & 0.8 \end{bmatrix}$$

where $y_k = 0.1 \cdot a_k + a_{k-1} - 0.3 \cdot y_{k-2}$ (no noise)

The BER depends on the data pattern. Averaging over those patterns, we have

$$P_b = \frac{1}{4} \cdot \left(Q\left(\frac{1.2}{\sigma_\nu}\right) + Q\left(\frac{0.6}{\sigma_\nu}\right) + Q\left(\frac{1.4}{\sigma_\nu}\right) + Q\left(\frac{0.8}{\sigma_\nu}\right) \right)$$

Although the average sample value is 1, just as it would be without ISI, the Q function is strongly nonlinear, and the weak values do more harm than the strong values do good. So we lose performance if there is ISI.

(b) One remedy is a simple linear equalizer; i.e., an FIR filter. With 5 coefficients, its impulse response is w_j, j=0..4 and its output at time k is

$$\sum_{j=0}^{4} w_k \cdot y_{k-j} = \mathbf{w}^T \cdot \mathbf{y} \qquad \text{where} \qquad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_k \\ y_{k-1} \\ y_{k-2} \\ y_{k-3} \\ y_{k-4} \end{bmatrix}$$

Now we need a compact expression for the components of **y**. Since the oldes component, y_{k-4}, depends on a_{k-4} to a_{k-6}, **y** depends on 7 successive data values.

$$\begin{bmatrix} y_k \\ y_{k-1} \\ y_{k-2} \\ y_{k-3} \\ y_{k-4} \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & x_2 & 0 & 0 & 0 & 0 \\ 0 & x_0 & x_1 & x_2 & 0 & 0 & 0 \\ 0 & 0 & x_0 & x_1 & x_2 & 0 & 0 \\ 0 & 0 & 0 & x_0 & x_1 & x_2 & 0 \\ 0 & 0 & 0 & 0 & x_0 & x_1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} a_k \\ a_{k-1} \\ a_{k-2} \\ a_{k-3} \\ a_{k-4} \\ a_{k-5} \\ a_{k-6} \end{bmatrix} + \begin{bmatrix} \nu_k \\ \nu_{k-1} \\ \nu_{k-2} \\ \nu_{k-3} \\ \nu_{k-4} \end{bmatrix} \qquad \text{or} \qquad \mathbf{y} = \mathbf{X} \cdot \mathbf{a} + n$$

Which data symbol will we try to estimate with this equalizer? Since y_{k-2} is in the centre, following the same logic as in part (a) says we should estimate a_{k-3}. No surprise - it's in the centre of the **a** vector.

By now, you should not need a rederivation of the normal equations, but I'll throw it in, anyway. You form the estimate

$$a'_{k-3} = \mathbf{w}^T \cdot \mathbf{y} \qquad \text{with error} \qquad e = a'_{k-3} - a_{k-3} = \mathbf{w}^T \cdot \mathbf{y} - a_{k-3}$$

You want to minimize, by choice of **w**, the error variance

$$\sigma_e^2 = E\left(e^2\right)$$

Taking the gradient and setting it to (vector) **0** gives

$$\tilde{N}_{\mathbf{w}}\left(\sigma_e^2\right)=E\left(2\cdot\tilde{N}_{\mathbf{w}}(e)\cdot e\right)=2\cdot E(\mathbf{y}\cdot e)=2\cdot E\left(\mathbf{y}\cdot\mathbf{y}^T\cdot\mathbf{w}-\mathbf{y}\cdot a_{k-3}\right)=\mathbf{0}$$

Using $\mathbf{R_y}=E\left(\mathbf{y}\cdot\mathbf{y}^T\right)$ $\mathbf{p}=E\left(\mathbf{y}\cdot a_{k-3}\right)$ we have the normal equations $\mathbf{R_y}\cdot\mathbf{w}=\mathbf{p}$

Assuming i.i.d. data and noise samples uncorrelated with each other or the data, we determine

$$\mathbf{R_y}=E\left[(\mathbf{X}\cdot\mathbf{a}+n)\cdot\left(\mathbf{a}^T\cdot\mathbf{X}^T+n^T\right)\right]=\mathbf{X}\cdot\mathbf{X}^T+\sigma_v^2\cdot\mathbf{I_5}$$ (5x5 identity matrix)

Expansion gives a bulky 5x5 matrix, of which the first 3 columns are

$$\begin{bmatrix}
\left(x_0\right)^2+\left(x_1\right)^2+\left(x_2\right)^2+\sigma_v^2 & x_1\cdot x_0+x_2\cdot x_1 & x_2\cdot x_0 \\
x_1\cdot x_0+x_2\cdot x_1 & \left(x_0\right)^2+\left(x_1\right)^2+\left(x_2\right)^2+\sigma_v^2 & x_1\cdot x_0+x_2\cdot x_1 \\
x_2\cdot x_0 & x_1\cdot x_0+x_2\cdot x_1 & \left(x_0\right)^2+\left(x_1\right)^2+\left(x_2\right)^2+\sigma, \\
0 & x_2\cdot x_0 & x_1\cdot x_0+x_2\cdot x_1 \\
0 & 0 & x_2\cdot x_0
\end{bmatrix}$$

and the last 2 are

$$\begin{bmatrix}
0 & 0 \\
x_2\cdot x_0 & 0 \\
x_1\cdot x_0+x_2\cdot x_1 & x_2\cdot x_0 \\
\left(x_0\right)^2+\left(x_1\right)^2+\left(x_2\right)^2+\sigma_v^2 & x_1\cdot x_0+x_2\cdot x_1 \\
x_1\cdot x_0+x_2\cdot x_1 & \left(x_0\right)^2+\left(x_1\right)^2+\left(x_2\right)^2+\sigma_v^2
\end{bmatrix}$$

Substitution of numerical values gives

$$\mathbf{R}_{\mathbf{y}} = \begin{bmatrix} 1.1+\sigma_v^2 & -0.2 & -0.03 & 0 & 0 \\ -0.2 & 1.1+\sigma_v^2 & -0.2 & -0.03 & 0 \\ -0.03 & -0.2 & 1.1+\sigma_v^2 & -0.2 & -0.03 \\ 0 & -0.03 & -0.2 & 1.1+\sigma_v^2 & -0.2 \\ 0 & 0 & -0.03 & -0.2 & 1.1+\sigma_v^2 \end{bmatrix}$$

Similarly, the **p** vector is given by

$$\mathbf{p} = E\left(\mathbf{y}\cdot a_{k-3}\right) = \mathbf{X}\cdot\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ x_2 \\ x_1 \\ x_0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -0.3 \\ 1 \\ 0.1 \\ 0 \end{bmatrix}$$

With a value for the noise variance, we could solve $\mathbf{R}_{\mathbf{y}}\cdot\mathbf{w}=\mathbf{p}$ and obtain the equalizer coefficients that minimize the MSE in estimating a_{k-3}. To determine the resulting improvement, we would:

* calculate the new noise variance as

$$\sigma^2 = \sigma_v^2\cdot\mathbf{w}^T\cdot\mathbf{w}$$

* calculate the signal amplitude for the $2^6=64$ combinations of the 6 interfering bits (they interfere a lot less than in part (a)!)

* average the 64 *Q* function values.

2. Carrier Recovery

There are many ways to handle this problem, and I don't care which way you did it (unless it's hopelessly bad). In these solutions, I have provided:

* the decision directed feedforward approach;
* the decision directed loop;
* the statistically optimum (but computationally intensive) solution.

The first step is to read the data arrays. They are pasted here, and you can scroll up or down with the mouse to see the full array. The lines below convert them to complex format.

temp :=

	0	1
0	2.33	0.12
1	-1.08	-1.43
2	1.27	1.81
3	1.37	-0.6
4	-1.83	-0.58
5	0.35	1.67
6	0.65	1.31

matched filter output samples

temp2 :=

	0	1
0	0	0
1	0	-1
2	-1	0
3	0	-1
4	-1	0
5	0	-1
6	1	0

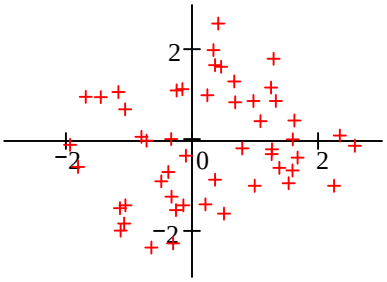
transmitted data (don't use!)

$y := temp^{<0>} + j \cdot temp^{<1>}$

$a := temp2^{<0>} + j \cdot temp2^{<1>}$

$N_{sym} := length(y) - 1$ $n0 := 0 .. N_{sym}$ $n1 := 1 .. N_{sym}$

A first look at the data suggests that we have a challenge ahead. It seems to be all over the place.



Matched Filter Output Samples

No matter which way we approach the problem, some procedures will be useful:

$$\text{decn}(x) := \exp\left[j \cdot \frac{\pi}{2} \cdot \left(\text{floor}\left(\frac{2}{\pi} \cdot \arg\left(x \cdot e^{j \cdot \frac{\pi}{4}} \right) \right) \right) \right]$$

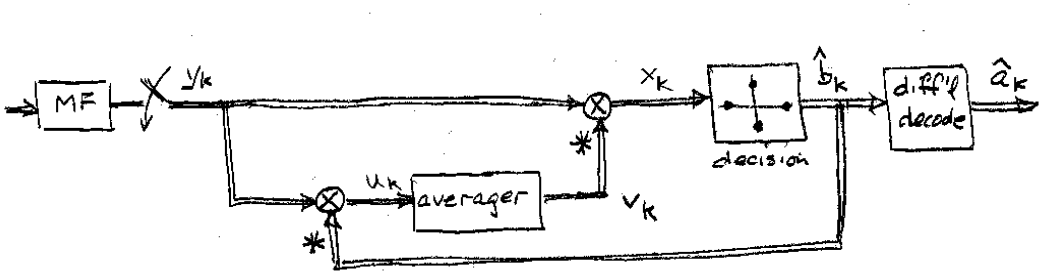
This decision function returns the constellation point nearest the complex input *x* (there are many ways to write this function).

$$\text{reduce}(\theta) := \arg\left(e^{j \cdot \theta} \cdot \overline{\text{decn}\left(e^{j \cdot \theta} \right)} \right)$$

This one reduces an arbitrary angle θ to the range -π/4 to π/4, reflecting the four-fold ambiguity.

The Feedforward Approach

The feedforward approach was outlined in the lecture notes and is sketched below.



The memory of the tracker is the averager. Many ways to average, but the simplest is a recursive averager - a first order lowpass, described variously by

$v_n = (1 - \alpha) \cdot v_{n-1} + \alpha \cdot u_n$ (v_n is the ouput, u_n is the input)

$$V(z) = \frac{\alpha \cdot z}{z - (1 - \alpha)} \cdot U(z) \qquad h_n = \alpha \cdot (1 - \alpha)^n$$

dc gain = 1, time constant is about 1/α samples

The tracker operation is captured in the following code. One of the input parameters is the starting value of the averager state. It returns an array with its first column equal to the averager output and second column equal to the *b* decisions.

$$\text{track}(y,\alpha,\text{vstart}) := \left|\begin{array}{l} \text{bhat}_0 \leftarrow \text{decn}\left(y_0 \cdot \overline{\text{vstart}}\right) \\ v_0 \leftarrow (1-\alpha)\cdot \text{vstart} + \alpha \cdot y_0 \cdot \overline{\text{bhat}_0} \\ \text{for } n \in 1..N_{\text{sym}} \\ \quad \left|\begin{array}{l} \text{bhat}_n \leftarrow \text{decn}\left(y_n \cdot \overline{v_{n-1}}\right) \\ v_n \leftarrow (1-\alpha)\cdot v_{n-1} + \alpha \cdot y_n \cdot \overline{\text{bhat}_n} \end{array}\right. \\ \text{augment}(v,\text{bhat}) \end{array}\right.$$

Now we can use it on the samples. Since we don't know anything about the phase to begin, we'll try the following:

$\text{vstart} := 0.001$ $\alpha := 0.04$ (try different values and watch the results)

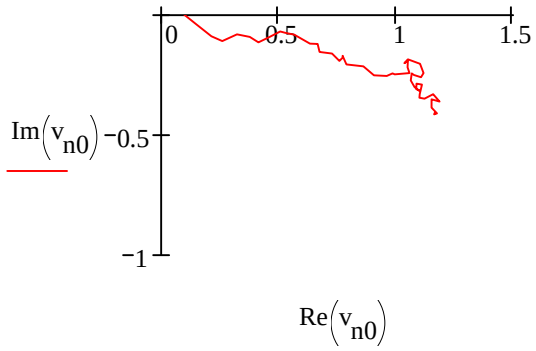
$\text{est} := \text{track}(y,\alpha,\text{vstart})$

$v := \text{est}^{\langle 0 \rangle}$ $\text{bhat} := \text{est}^{\langle 1 \rangle}$ separate the returned values

$\text{ahat}_{n1} := \text{bhat}_{n1} \cdot \overline{\text{bhat}_{n1-1}}$ differentially decode

$e := \overrightarrow{\left[(\text{ahat} - a)\cdot \overline{a}\right]}$ find the errors (overhead arrow means "do it to every component")

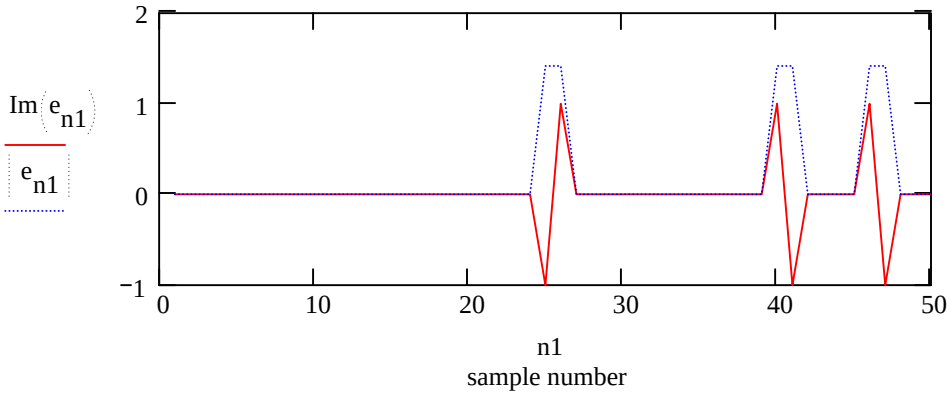
The results are shown below.



Trajectory of Averager Output

This seems like a pretty reliable estimate. It is growing in magnitude because of the small starting value, but magnitude doesn't matter. The phase seems reasonably consistent.

$\alpha = 0.04$ $\quad vstart = 1 \cdot 10^{-3}$



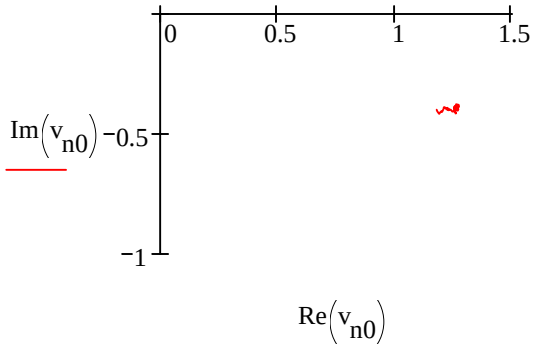
Error Occurrences

Errors occur in pairs, because an error in *bhat* affects two successive values of *ahat.* The decisions are listed in the Appendix.

Can we improve the tracker with a second pass, starting it where the first pass left off? Let's try:

$\alpha := 0.01$ $\quad est := track(y, \alpha, v_{N\,sym})$ $\quad v := est^{\langle 0 \rangle}$ $\quad bhat := est^{\langle 1 \rangle}$

$ahat_{n1} := bhat_{n1} \cdot \overline{bhat_{n1-1}}$ $\quad e := \overline{(ahat - a) \cdot a}$

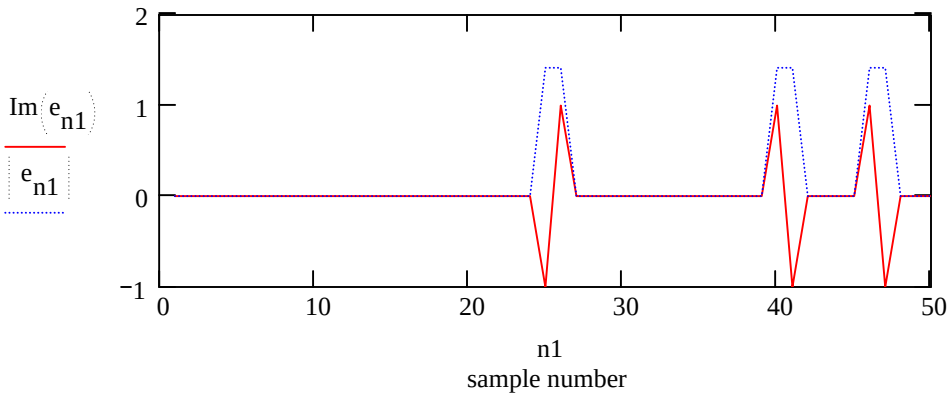


The estimate *v* is nicely stable.

$\alpha = 0.01$

$\arg\left(v_{N\,sym}\right) = -0.317$

Trajectory of Averager Output



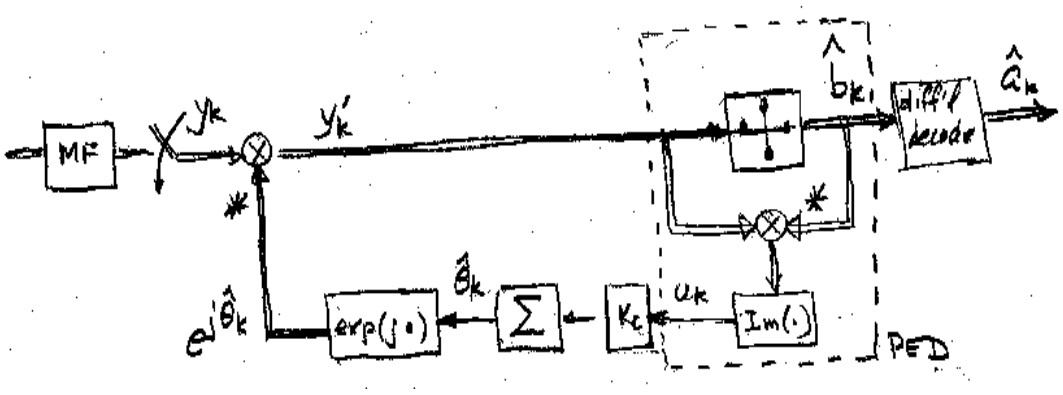
Error Occurrences

Too bad it didn't make any difference to the number of errors - but remember that even perfectly coherent detection produces errors.

The Tracking Loop

Our next version is a loop. Again, it comes directly from the lecture notes. The sketch below shows that its memory is in the accumulator and the phase error detector (PED) delivers the imaginary part of *y*' after derotation by the decision *bhat.* The rationale is that the mean PED output should be zero if θ*hat* equals the true rotation θ, and that θ*hat* should be adjusted in proportion to this mean value. Since we don't have the mean value, we'll use the instantaneous value and rely on the loop time constant for the averaging.

In contrast to the feedforward loop, the PLL is strongly nonlinear through the polar to rectangular conversion implicit in the exp(j *) function. A linearized approximate analysis shows that it operates as a first order averager in θ*hat*, with time constant in samples closely equal to $(K_c A_y)^{-1}$, where A_y is the average value of $|y|$. It requires more computation than feedforward.



The loop operation is captured in the following code. One of the input parameters is the starting value of the phase estimate θ*hat* stored in the accumulator. The procedure returns an array with its first column equal to the estimated phase and second column equal to the estimated phase and second column equal to the *b* decisions.

$$\begin{aligned}
&\text{loop}(y, K_c, \theta\text{hatstart}) := \\
&\quad y' \leftarrow y_0 \cdot \exp(-j \cdot \theta\text{hatstart}) \\
&\quad \text{bhat}_0 \leftarrow \text{decn}(y') \\
&\quad \theta\text{hat}_0 \leftarrow \theta\text{hatstart} + K_c \cdot \text{Im}\left(y' \cdot \overline{\text{bhat}_0}\right) \\
&\quad \text{for } n \in 1 .. N_{sym} \\
&\qquad y' \leftarrow y_n \cdot \exp\left(-j \cdot \theta\text{hat}_{n-1}\right) \\
&\qquad \text{bhat}_n \leftarrow \text{decn}(y') \\
&\qquad \theta\text{hat}_n \leftarrow \theta\text{hat}_{n-1} + K_c \cdot \text{Im}\left(y' \cdot \overline{\text{bhat}_n}\right) \\
&\quad \text{augment}(\theta\text{hat}, \text{bhat})
\end{aligned}$$

Now we can use it on the samples. Since we don't know anything about the phase to begin, we'll try the following:

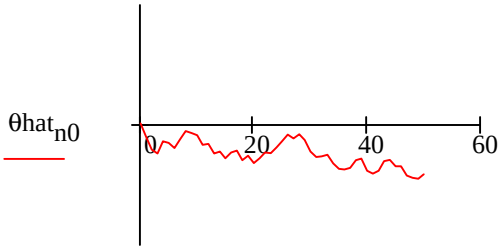
$\theta\text{hatstart} := 0$ $K_c := 0.1$ (try different values and watch the results)

$est := loop(y, K_c, \theta hatstart)$

$\theta hat := est^{<0>}$ $bhat := est^{<1>}$ separate the returned values

$ahat_{n1} := bhat_{n1} \cdot \overline{bhat_{n1-1}}$ differentially decode

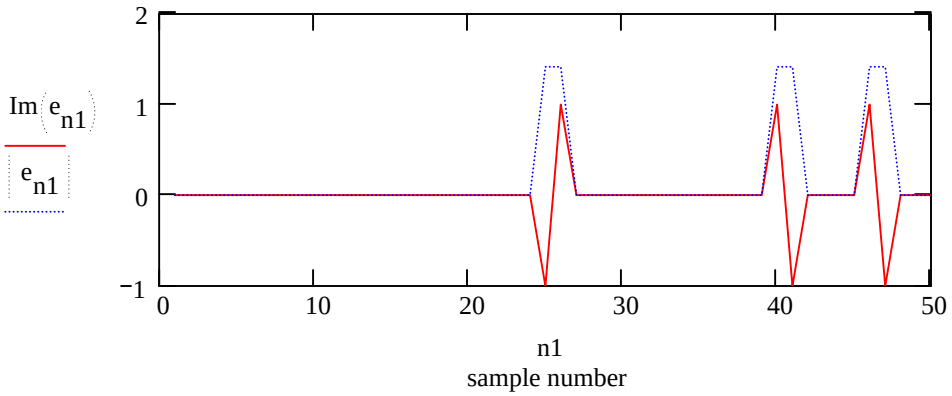
$e := \overrightarrow{\left[(ahat - a) \cdot \overline{a}\right]}$ find the errors (overhead arrow means "do it to every component")



Trajectory of Phase Estimate

The phase is creeping toward some value, but increasing K_c to speed it up also gives the estimate more jitter.

$K_c = 0.1$ $\theta hatstart = 0$



Error Occurrences

I wonder if a second pass, initialized with the phase estimate inherited from the end of the first pass, would improve things. Reduce the time constant, too. Try it.

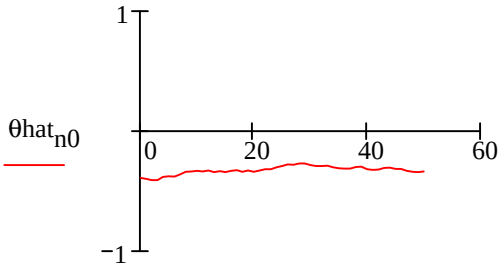
$\theta hatstart := \theta hat_{N_{sym}}$ $K_c := 0.02$ (try different values and watch the results)

$est := loop(y, K_c, \theta hatstart)$

$\theta hat := est^{<0>}$ $bhat := est^{<1>}$ separate the returned values

$ahat_{n1} := bhat_{n1} \cdot \overline{bhat_{n1-1}}$ differentially decode

$e := \overrightarrow{\left[(ahat - a) \cdot \overline{a}\right]}$ find the errors (overhead arrow means "do it to every component")

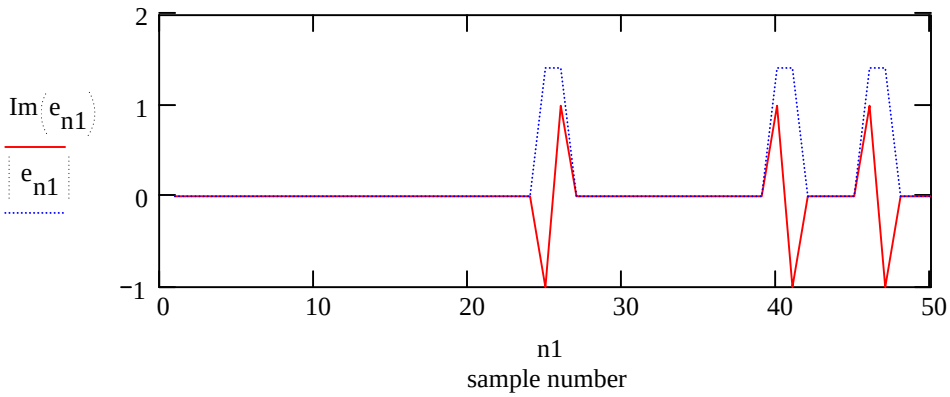


n0

Trajectory of Phase Estimate

It's pretty stable, but K_c is smaller than in first pass. We can't make it too small (i.e., time constant too large), or we don't get much benefit from the second pass.

$K_c = 0.02$ $\theta hatstart = -0.411$



n1
sample number

Error Occurrences

The data decisions are listed in the Appendix.

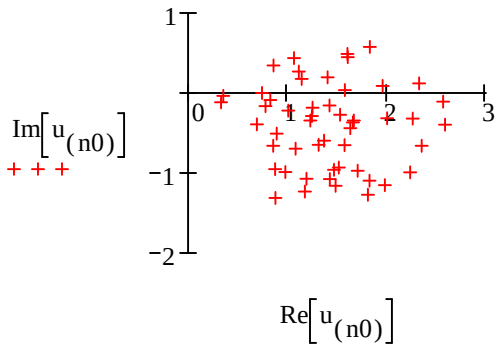
Comparison of the final values of the feedforward estimator and the loop shows they both work.

feedforward	loop
$\arg\left(v_{N_{sym}}\right) = -0.317$	$\theta hat_{N_{sym}} = -0.339$

but it's easier to fool the loop if θ initially puts the constellation points on decision boundaries.

Finally, the SNR estimate. First, get rid of the data modulation:

$u := \overrightarrow{\left(y \cdot \overline{bhat}\right)}$ overhead arrow means "do it to every component", so *u* is an array of derotated samples, shown below.



This represents the constellation point that is nominally on the positive real axis, rotated by θ and buried in noise.

The average value (centroid) is $\mu_u := mean(u)$ $\mu_u = 1.416 - 0.413i$

and its length is $\sqrt{2 \cdot E_s}$ so $E_s := \frac{1}{2} \cdot \left| \mu_u \right|^2$ $E_s = 1.088$

The real part and the imaginary part of u both have variance N_o, so

$$N_o := \frac{1}{2} \cdot \frac{1}{N_{sym} + 1} \left[\sum_{i=0}^{N_{sym}} \left| u_i - \mu_u \right|^2 \right] \qquad N_o = 0.271$$

$\frac{E_s}{N_o} = 4.018$ $E_b := 0.5 \cdot E_s$ $\frac{E_b}{N_o} = 2.009$

The Statistically Optimum Solution

Both feedforward and the loop have the appearance of being *ad hoc*, approximate solutions. They are. Now let's see what the optimum solution is for a block of *N* symbols. Refer to page Section 7.5 of the lecture notes.

We have a length-*N* vector of samples **r** and a large number of candidate signal vectors $\mathbf{s}_i$, i=1..M, each corresponding to a different vector **b**. Large does mean *large:* for QPSK, $M=4^N$. We have the basic pdf

$$p_{\mathbf{r}|\mathbf{s}_i,\theta}(\mathbf{r}\,|\,\mathbf{s}_i,\theta) = p_{\mathbf{n}}(\mathbf{r}-e^{j\theta}\mathbf{s}_i) = \frac{1}{(2\pi N_o)^N}\exp\left(-\frac{\left(\mathbf{r}-e^{j\theta}\mathbf{s}_i\right)^H\left(\mathbf{r}-e^{j\theta}\mathbf{s}_i\right)}{2N_o}\right)$$

where superscript *H* denotes Hermitian transpose (conjugate transpose). Expanding the exponent reveals factors that do not affect the decision

$$p_{\mathbf{r}|\mathbf{s}_i,\theta}(\mathbf{r}\,|\,\mathbf{s}_i,\theta) = \frac{1}{(2\pi N_o)^N}\exp\left(-\frac{\mathbf{r}^H\mathbf{r}+2NE_s}{2N_o}\right)\exp\left(\frac{\mathrm{Re}\left[e^{-j\theta}\mathbf{s}_i{}^H\mathbf{r}\right]}{N_o}\right)$$

since $\mathbf{s_i}^{H}\cdot\mathbf{s_i} = 2\cdot N\cdot E_s$

We don't know θ, so we consider it a nuisance parameter, and form the marginal density by integrating it out:

$$p_{\mathbf{r}|\mathbf{s}_i}(\mathbf{r}\,|\,\mathbf{s}_i) = \int_0^{2\pi} p_{\mathbf{r},\theta|\mathbf{s}_i}(\mathbf{r},\theta\,|\,\mathbf{s}_i)d\theta = \int_0^{2\pi} p_{\mathbf{r}|\mathbf{s}_i,\theta}(\mathbf{r}\,|\,\mathbf{s}_i,\theta)\,p_\theta(\theta)d\theta = \frac{1}{2\pi}\int_0^{2\pi} p_{\mathbf{r}|\mathbf{s}_i,\theta}(\mathbf{r}\,|\,\mathbf{s}_i,\theta)d\theta$$

Substituting for the conditional density in the last equality above, ignoring the factors that do not affect the decision and integrating as in Section 7.5, we obtain

$$\hat{i} = \arg\max_i\left(I_0\left(|\,\mathbf{s}_I^H\mathbf{r}\,|\right)\right) = \arg\max_i\left(\left|\mathbf{s}_I^H\mathbf{r}\right|^2\right)$$

This means that we should correlate the N-sample received vector with every possible transmitted sequence and select the correlation with the largest magnitude (or magnitude squared). This could be a *lot* of work for long sequences. However, for shorter blocks, such as 4 to 6 symbols, the method received a brief flurry of interest as *multisymbol differential detection* in the context of wireless communication.

APPENDIX: DATA DECISIONS

The feedforward and loop structures produce the same sequence of decisions. The array is broken into a first half and a second half so they fit on one page.

ahatfirst := submatrix(ahat, 0, 25, 0, 0) ahatsecond := submatrix(ahat, 26, 50, 0, 0)

ahatfirst =

	0
0	0
1	-1i
2	-1
3	-1i
4	-1
5	-1i
6	1
7	-1i
8	-1i
9	-1
10	-1
11	1i
12	-1i
13	-1
14	1
15	-1i
16	1i
17	1i
18	1
19	-1i
20	-1
21	1
22	1i
23	-1i
24	1i
25	-1i

ahatsecond =

	0
0	1i
1	-1i
2	1
3	1
4	1
5	-1
6	1i
7	1
8	-1
9	-1
10	-1
11	1
12	1i
13	-1i
14	1i
15	1i
16	-1i
17	1i
18	-1
19	-1i
20	1
21	-1
22	1i
23	1i
24	1i