

ELECTRONOTES 221

Newsletter of the Musical Engineering Group

1016 Hanshaw Road, Ithaca, New York 14850

Volume 23, Number 221

March 2014

RESONATORS AND FRIENDS - Prony with Noise

-by Bernie Hutchins

INTRODUCTION:

A teaching colleague of mine delighted in telling students that what he was about to present to them was nothing new because, in fact (so he claimed), – he only knew a few things and had to dress them up in new clothes and parade them out once again. All and all, we often do not appreciate that in education (formal or informal) we likely underestimate that value of recognizing a newly presented notion as "kind of like" something we have seen before.



The second-order digital network of Fig. 1 is very familiar, and we shall be referring to this as a "resonator" for reasons likely obvious to most persons reading here. It will be reviewed quite carefully, and then will serve as a test object for a number of topics that will follow.

REVIEWING THE RESONATOR

A digital network is much easier to analyze than a corresponding analog network. It is only a slight exaggeration that you could teach them to fifth-graders, at least as games. All you have are delays, multipliers, and summers. Tell the kids that the delays simply mean that you open doors all at once and everyone (every number) moves through. This shifting of data defines an "instance of time". Then you "update" any still empty locations (locations where no new number automatically walked in) and wait for the next declaration of a new instance of time. And so on. Our purpose here is certainly not to present material at a fifth-grade level, but here to point out that it <u>begins</u> very simple – exactly what you suppose.

We have, in the past [1] had numerous presentations of "Prony's Method" and related issues. Let us first quickly summarize the mathematics of the resonator. The "fifth-grade" analysis is to write down the "difference equation" (just an obvious equation) for the summer of Fig. 1:

$$y(n) = x(n) + a_1 y(n-1) + a_0 y(n-2)$$
(1)

Here we want to consider the "undriven" case where $x(n) \equiv 0$. Thus equation (1) becomes:

$$y(n) = a_1 y(n-1) + a_0 y(n-2)$$
⁽²⁾

This is unremarkable except we recognize that the three values of the y-sequence are successive values in time. Thus we do not get to choose y(n - 1) and y(n - 2) to be arbitrary value. But equation (2) does apply to <u>any three</u> consecutive values of the output sequence. Here we assume fixed values of a_1 and a_0 , [we could also assume networks of higher order (three, four, or more delays)]. We look to see if we can calculate unknown values of the a_k based on samples from the y-sequence y(n). So we might be saying that we know that a given sequence y(n) was produced by a particular network and decide to calculate the a_k . Or we might say that we don't actually know where y(n) came from, but what would the so-called auto-regressive (AR) coefficients a_k be if it were of a certain order. Thus determining the a_k from proper use of available y(n) is our goal. This we will get to shortly.

To get an alternative description of the network (meaning – frequency domain instead of the time-domain difference equation) you will need to z-transform the difference equation – which couldn't be simpler. You just use a z^{-1} as a delay operator (sometimes called a "lag operator"), and write variables as a function of z. Equation (1) becomes:

$$Y(z) = X(z) + a_1 Y(z) z^{-1} + a_0 Y(z) z^{-2}$$
(3)

which is a rather "mechanical" transformation, and probably less meaningful than the difference equation itself. This allows us to treat the network as a <u>digital filter</u> and write a "transfer function" H(z) as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 - a_1 z^{-1} - a_0 z^{-2}}$$
(4)

Notice that we had to put the input X(z) back in to get a meaningful transfer function. We can do a lot with this H(z) such as get the actual frequency response of the filter as:

$$H(z) = \sqrt{H(e^{j\omega})H(e^{-j\omega})}$$
(5)

Readers wishing more information on digital filters are invited to look at our full series [2].

For the moment, we are interested in the denominator of equation (4) from which we get the "poles" of the network. The poles are the values of z for which the denominator becomes zero. These are good indications of the nature of the frequency response, and are a direct indication of network stability. If the poles are inside a unit circle in the z-plane, we have stability.

So here we will want to look next, using a simple example, at how poles are found from the data (from the signal). We will be using the first part of Prony's method. [The second part is just the application of initial conditions.]

EXAMPLE – SOLVING INSTANCES OF THE DIFFERENCE EQUATION

First we need a signal, so first we choose some poles, generate a signal from them, and pretend that we don't know where the poles were to start with, and try to recover them. Our second-order network of Fig. 1 will support two poles. Let's place these poles at a radius r of 0.95 at angles of $\pm 10^{\circ}$. Thus at $p_{1,2} = 0.9356 \pm 0.1650j$. The denominator is thus $(z-p_1)(z-p_2) = (z^2 - 2zr \cos(10^{\circ}) + r^2) = z^2 - 1.8711 z + 0.9025$. From this we find $a_1 = 1.8711$ and $a_0 = -0.9025$. We can start with any initial state of Fig. 1. Choose y(n) as y(0)=1, with y(-1)=0 and y(-2)=0, and iterate equation (2). The result in this case is plotted in Fig. 2. The manner in which the difference equation (2) generates a new sample from the previous two is indicated by the three green points and the associated green summer (with path weights shown). Thus the three green points illustrate an instance of the difference equation, as do the three successive dark blue points, as do the three successive light blue points. In fact, all sequences of three successive points are an instance, and we will need only two of these instances to solve for a_0 and a_1 . That is – this is trivially just two equations in two unknowns.



So, for our calculation example, consider the use of the three green samples and the three dark blue samples. Writing these two difference equation instances in matrix from we have:

$$\begin{bmatrix} y(9) & y(10) \\ y(20) & y(21) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} y(11) \\ y(22) \end{bmatrix}$$
(6a)

and putting in the numbers we have:

$$\begin{bmatrix} 3.5743 & 3.2400 \\ -1.0322 & -1.2606 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 2.8367 \\ -1.4273 \end{bmatrix}$$
(6b)

which solve for

$$\begin{bmatrix} a_0\\a_1 \end{bmatrix} = \begin{bmatrix} -0.9025\\1.8711 \end{bmatrix}$$
(7)

which are indeed the coefficients we began with. Since there are three colored-in sets of successive samples in Fig. 2, we could have used any two of the three (or any other three consecutive samples). It is of course not uncommon that we use four successive samples for our calculation: the first three of these, and the last three of these. If we consider that the choice of initial conditions was arbitrary, we can see that it was not even necessary that the two instances be chosen from the same signal as long as both runs used have the same coefficients.

EN#221 (4)

Yet another case that will <u>suggest</u> a practical approach to dealing with random noise in the signal is that we can add instances. Here for example, we form our two equations by adding the green and the light blue samples:

$$\begin{bmatrix} y(9) + y(30) & y(10) + y(31) \\ y(20) & y(21) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} y(11) + y(32) \\ y(22) \end{bmatrix}$$
(8a)

and putting in the numbers we have:

$$\begin{bmatrix} 2.6274 & 2.4853 \\ -1.0322 & -1.2606 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 2.1789 \\ -1.4273 \end{bmatrix}$$
(8b)

which has the exact same solution of equation (7).

Alternative to adding the equations, we can include more equations and solve them by least squares, and again this may be useful in noisy cases. Here in the noiseless case, the pseudo-inverse contributes nothing to the result, <u>except it does not complain about the non-square matrix</u>. Keeping all three instances from Fig. 2 we have:

$$\begin{bmatrix} y(9) & y(10) \\ y(20) & y(21) \\ y(30) & y(31) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} y(11) \\ y(22) \\ y(32) \end{bmatrix}$$
(9a)

and putting in the numbers we have:

$$\begin{bmatrix} 3.5743 & 3.2400 \\ -1.0322 & -1.2606 \\ -0.9469 & -0.7548 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 2.8367 \\ -1.4273 \\ -0.5578 \end{bmatrix}$$
(9b)

Solving this with Matlab's *pinv* function (three equations in 2 unknowns) gives us, again, equation (7).

WHAT HAPPENS WITH NOISY DATA?

What we have done above is exponential modeling by using the first part of Prony's method, and we expected good results because the data was clean. In the case where there is noise, the results are not going to work as well of course. In fact, it may be the case that the results will become unusable.

There is little we can do with regard to the direct N equations in N unknowns calculation. It works perfectly with clean data but in general, does not like noise (gives wrong answers). We have seen two other methods which we might expect to help: summing instances (a classic ploy of reducing random noise through random cancellation) and overdetermination (keeping more equations). Both of these gave the perfectly correct solution to clean data, as we would have required. Here we will be looking at how these two methods help with the noise problem, and compare them to a fourth method of solution, the Yule-Walker calculation which is here presented in a simplified form.

WHAT DO ERRORS LOOK LIKE

Here we are trying to recover the feedback parameters from the data, and when there is no noise, a wide variety of calculation methods give a perfect answer. That is, we get back $a_0 = -0.9025$ and $a_1 = 1.8711$. We have errors when the input data is noisy, as we would expect. Qualitatively, recognizing the errors is a matter of observing the calculation results (such as plotting the results of many noisy runs). Quantitatively, this is a bit tricky at times, as errors may change with calculation method, and with the portions of the signal that are chosen.



Fig. 3 shows some examples. All of these use a direct calculation (two equations in two unknowns) with two instances, and the plots show the results for the coefficients for 100 runs. The top line is simply the nominal, no noise result. As we expect, adding noise causes the results to spread. The second line from the top shows a noise (random normal) of amplitude (standard deviation) 0.01 added to Fig. 2. Note the spread. Increasing the noise to 0.03 roughly increases the spread by a factor of 3. But there is another way the spread can

increase, and this is by taking the instances somewhat later. In the bottom line we have a noise going back to 0.01 with a spread roughly the same as the case of 0.03 above. The difference is that instead of taking the sample instances at [10,11,12] and [21,22,23] they are taken at [15,16,17] and at [48,49,50]. Clearly what is happening is that at the later sample times, the signal itself is smaller while the noise remains constant. Thus taking data from later on in this decaying case, we effectively have more noise, and more error in the coefficients. All this is important, as some methods (other than the two-equations calculations of Fig. 3) will involve different portions of the data and in general, data selected from a wide range, including the far end (in this case) where the data is effectively noisier. Here we are mainly illustrating methods and effects, and offer caution with regard to any hard conclusions.

ADDING INSTANCES

Based on Fig. 2 and equation (8a) we see that instances of the difference equation can be added, and the resulting equations still give exactly the same answer. Here we add the notion that when noise-containing signals are added (uncorrelated noise) we expect a general reduction in the noise due to accidental cancellations. Fig. 4 shows an initial example.



In Fig. 4 we get some evidence that summation of instances can be of <u>slight</u> value. The top line shows the case where the coefficients are calculated from instances [1,2,3] and [10,11,12]. That is, the original method. For the bottom plot, we have summed three instances for each of the two equations. Specifically we used [1,2,3]+[4,5,6]+[7,8,9] for one equation and [10,11,12]+[13,14,15]+[16,17,18] for the second equation. The improvement is real and noticeable but relatively slight. Note that these are all still relatively early samples, although we may be just starting to see the effect of the signal getting smaller.

Going further we consider what happens if we add a lot more instances. The ploy here is to "condense" nearly all the signal into one sequence of just four samples. Thus the first

number of the combined sequence is yy1=y(1)+y(2)+...y(95), the second number is yy2=y(2)+y(3)+...y(96), the third is yy3=y(3)+y(4)+...+y(97) and the fourth number we need is yy4=y(4)+y(5)+...+y(98). Note that this as a lot of data combined. Further, since we go nearly all the way through the input sequence, the end is going to be very noisy as the actual signal gets smaller. Finally, the sum of the whole center region from y(4) to y(95) is in all four numbers. For example (only an example), with a noise of 0.03 we found the equations to be:

$$\begin{bmatrix} yy1 & yy2 \\ yy2 & yy3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} yy3 \\ yy4 \end{bmatrix}$$
(10a)

$$\begin{bmatrix} 32.2323 & 31.2585 \\ 31.2585 & 29.4004 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 29.4004 \\ 26.7323 \end{bmatrix}$$
(10b)
$$\begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} -0.9770 \\ 1.9480 \end{bmatrix}$$
(10c)

We thus see that the results are not too good, and this would seem to be attributable to the decreasing signal/noise ratio. While the elements of the matrix are starting to get very similar, we likely have a way to go before the inverse is ill-conditioned. None the less, the slight improvement suggested in Fig. 4 does not persist. In Fig. 5 we see that essentially there is no improvement.



We suppose, as seems reasonable, that the decreasing signal/noise problem might be less if we look at a test signal that decays slower. This is apparently true. Changing the pole radius to 0.999 instead of 0.95 means that the spreads in error are reduced. (In general, we can't just insist on a less damped input signal – it is given to us.) At the same time, the errors tend to be asymmetric about the nominal values for reasons not apparent or investigated. As a general rule, results are ambiguous in this report, and we are longer on methodology of investigation than on conclusions.

OVER-DETERMINED

So we have looked at notions of using the extra instances of the difference equation by addition, and found them wanting. We didn't even get a guarantee of reduced effects of noise. It is certainly true that we get exactly the right answer using summation (in the noiseless case). In as much as we are throwing information away by summing equations, we might have anticipated a lack of success. But we can keep all the instances (all the equations) and still ask them to "participate' by solving an over-determined set. This is what we had in the equation (9) case. Nobody says we can only use one extra equation (three instead of two). Equation (11) shows a horrendous case of 95 equations in two unknowns corresponding to nearly all possible instances of the difference equation for a signal that is at least length 97.

y(y(1) 2)	y(2) y(3)		$\begin{bmatrix} y(3) \\ y(4) \end{bmatrix}$	
		•	$\left[\left[a_{1} \right] \right] =$	· ·	
y(9	94)	y(95)		y(96)	
L y(9) 5)	y(96)]	Ly(97)J	

(11)

Here the equations Ma=y have a matrix M [left-most in equation (11)] that is 95 rows by 2 columns, and the solution a=M⁻¹y requires the "pseudo-inverse" M⁻¹ which has 2 rows and 95 columns. This is obtained from Matlab's *pinv* function or as (M^tM)⁻¹M^t where M^t is the transpose. This finds a least squared error solution to all the equations. The surprise is perhaps that these 95 equations in 2 unknowns have a solution at all. That it should thereafter work better is perhaps something we should anticipate. (Summing equations as we did above did not retain the individual information.)

To perhaps be more clear, if we kept just the top two rows of equation (11) we would have the original two Prony equations, similar to equation (6), but using just the first four samples – the first three and the last three of the four. It would give the perfect answer in the noise-free case, but a faulty answer with the noise. In the noise-free case, the full equation (11), too, gives a perfect answer. It is the finding that equation (11) performs better with noise present. But something strange still happens as is seen in Fig. 6. In Fig. 6 we compare the use of just two equations (top line) and of using 95 equations (lower line). Clearly the use of all 95 equations is better, in that there is far less spread in the values for a_0 and a_1 . But, note that the answers are displaced from the correct ones, and that the errors are in the direction of low magnitudes. This seems to be consistent for different values of noise amplitude, and further investigation seems necessary and follows the Yule-Walker discussion just below.



YULE-WALKER

At this point, we want to compare the results we have above with the classical method of solving the problem known as "Yule-Walker" [3]. This method does not seem to evolve naturally through an intuitive solution process as we have used so far above. In addition, here we will be bypassing the derivation and just using the solution. At the same time, this is a "statistical" approach and carries notation and conceptual luggage with it.

By this I am suggesting that I, like probably many readers of our notes, am less than thrilled with the ideas of correlations and expected values. Indeed, the very notion of "a random signal" is hard to digest. Here I will write down "a random signal":

 $x = -0.9861 \quad 1.5199 \quad -2.4476 \quad 1.1554 \quad -0.7316 \quad -0.6488 \quad -0.4301$ (12)

which I got by resolving to just ask Matlab for 7 random samples: *randn(1,7)*. Yet many will insist and once we have chosen "a random signal" to be a specific example, instead of an unspecified abstraction, we have a deterministic system. If I ask about the mean of x and perhaps the standard deviation, I could look at the documentation of the *randn* function and learn that the mean is 0 and the standard deviation is 1. Yet if I apply the *mean* and *std* Matlab functions to the actual x of equation (12) I get -0.3670 and 1.3420. This is no surprise to anyone of course. The point is rather that, as we sometimes find explicitly stated, that when it comes to <u>estimating</u> parameters (means, standard deviations, auto-correlations, even "probability") we depart from random and are doing a deterministic calculation. Thus when we find comfort in being told that an "expected value" is "really" a mean, we are cautioned not to read too much into the result. But it is a way of moving forward, which we need.

The Yule-Walker approach is often depicted as a way of finding network coefficients which are usually considered to be the AR (Auto-Recursive) parameters of a sequence that results from a white-noise (thus flat) input. As such, a means of spectral estimation is available that is entirely equivalent to equation (5), which signal-processing engineers will immediately recognize. Consulting the usual derivation, you will find that the white-noise is made to "disappear" from the problem by forming the cross-correlation of the (input) noise with the output and simply pointing out that this term becomes zero. Fine. This is what we wanted because we are asking about an undriven case. Thus what remains is to solve a simplified version of the Yule-Walker equations. For the two coefficient network, this is:

$$\begin{bmatrix} r_{xx}(0) & r_{xx}(-1) \\ r_{xx}(1) & r_{xx}(0) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} r_{xx}(1) \\ r_{xx}(2) \end{bmatrix}$$
(13)

where the rxx are the autocorrelations of the signal being analyzed in terms of expected values:

$$r_{xx}(m-k) = E\{x(n-m)x(n-k)\}$$
(14)

which we agree to compute from actual data (estimate) by a mean:

$$r_{xx}(m) = \left(\frac{1}{N}\right)^{N-m-1} \sum_{n=0}^{N-m-1} x(n)x(n+m) \qquad m \ge 0$$
(15)

Equation (15) is really just our usual notion of calculating an autocorrelation (as is sometimes a step in spectral analysis) by offsetting, multiplying point-by-point, and summing, for as much of the sequence as we have available. [Note that equation (13) can be rewritten as:

$$\begin{bmatrix} r_{xx}(1) & r_{xx}(0) & r_{xx}(-1) \\ r_{xx}(2) & r_{xx}(1) & r_{xx}(0) \end{bmatrix} \begin{bmatrix} -1 \\ a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
(16)

where the (-1) put on the top of the a-column vector corresponds to the 1 in the denominator of equation (4).]

Equations (13) and (15) permit us to calculate the AR coefficients. Note the similarity to calculating the taps of an adaptive Wiener filter [4], a <u>very</u> useful comparison case. As we would always need to consider, we have to choose a range of y(n), the choice of N, over which we compute the r_{xx} , not unlike the choice of the number of overdetermined equations. The actual code use in this study, and for comparison cases is given in the Appendix at the end. We will see that the Yule-Walker solution has similar issues trading off the length of y analyzed and the effective decreasing signal/noise as y gets longer (as the signal decays).



We show now three figures (Fig. 7, Fig. 8, and Fig. 9) which expand on the initial finding of Fig. 6 and here also present and compare the results of the Yule-Walker to the overdetermined cases. In Fig. 7 we have in the top line the standard Prony (two equations in two unknowns). All of Fig. 7, Fig. 8, and Fig. 9 share this top line, and correspond to a noise level of 0.05, although the actual noise signal varies in the different figures. In Fig. 7, we have chosen (as



EN#221 (12)

would seem a wise first step) to use most of the y(n) data. Thus the middle line of Fig. 7 shows 95 equations, and the bottom line, corresponding to the Yule-Walker autocorrelation, computes length-95 correlations. The middle line corresponds well to Fig. 6 as it should. The bottom line, which is new, and is slightly wider in spread than the middle, although slightly less biased. Well, we suggested that using all the data might be a problem as the end gets very noisy, relatively.

Figure 8 looks like a vast improvement – <u>but it is really just a test</u>. What we have done here is to change the pole radius from 0.95 to 0.999, so that there is very little decay in 100 samples of y(n). This is only a test because we cannot in general change the input signal arbitrarily – it is a given signal to characterize. Now both the overdetermined and the autocorrelation results are improved and quite good. The purpose here was to test a hypothesis that it is the decay of signal amplitude that works against improvements expected from having longer lengths (more information) to work from. This seems to be true.



So we look to Fig. 9 to see if we can achieve a better result by using smaller sets of equations and shorter autocorrelations, and indeed, we can. Fig. 9 shows an example comparable to Fig. 7 except we use 30 equations instead of 95 (middle of Fig. 9) and 20 samples of y(n) instead of 95 (bottom of Fig. 9). No extensive study was made, although it seems that we get about a 2:1 improvement in both cases, relative to Fig. 7.

There is a fundamental idea here related to the old joke that the easiest way to improve a signal/noise ratio is to increase the signal – rather than to struggle to reduce noise. Accordingly we have found that it makes sense to extract parameters from a signal from a

region where the signal is largest. That is, we may get more information by using more of the signal, but it may work against us if it is dirtier. The noise itself is going to give us some wrong answers, and the errors might or might not be acceptable. We note for example, that in the case of the resonator, cases where a_0 comes out with a magnitude greater than 1 are unstable. First of all, whether or not an estimate is stable or not may not matter, as it doesn't make the <u>original</u> network that produced the signal unstable. In fact, successive estimates are going to vary about the nominal value – some underdamped, and some overdamped. If we are not attempting a reconstruction (perhaps we are just doing spectral estimation) we may not care, and allow things to average. A second point about using less input data: the computations are shorter, perhaps much much shorter.

REFERENCES

[1] [a, b, c] see for example:

(a) B. Hutchins, "Prediction, Deconvolution, and System Identification", *Electronotes*, Vol. 17, No. 179, June 1992;

(b) B. Hutchins, "Reader's Question", *Electronotes*, Vol. 21, No. 204, August 2004 <u>http://electronotes.netfirms.com/EN204.pdf</u>;

(c) B. Hutchins, Frequency Analysis/Resolution with Few Samples, *Electronotes* Application Note AN-365, April 15, 2006 (includes most recent program prony.m). <u>http://electronotes.netfirms.com/AN365.pdf</u>

[2] See our digital filter design series from 2001: <u>http://electronotes.netfirms.com/EN197.pdf</u>; <u>http://electronotes.netfirms.com/EN198.pdf</u>; <u>http://electronotes.netfirms.com/EN199.pdf</u>

[3] Many discussions get deeply into statistical terminology. Here I like an Internet video: <u>http://www.youtube.com/watch?v=PFyp4t16_xk</u> by Barry Van Veen, U. Wisconsin Madison. This is easy to watch, but is kind of a "blackboard" presentation. Barry has a whole series of videos that are excellent. Particularly useful if you are sort of familiar with the idea covered. You can then go on to personally investigating the topics with your own programming.

[4] I just updated the connection between LMS Algorithm and Wiener (which uses correlations) that are programmed in Matlab there. See B. Hutchins, "Wiener Alternative to LMS Algorithm," Electronotes Application Note AN-406, March 15, 2014 http://electronotes.netfirms.com/AN406.pdf

APPENDIX – Test Program

% prtest5.m

```
clear
for mm=1:100
   mm
% TEST PARAMETERS
r=0.999
an=0.05
eq = 95 % overdetermined equations
aut= 95 % points in autocorr
% Generate y
% Poles at r at 10 degrees
theta=10;
a0 = -r^{2};
a1= +2*r*cos(theta*2*pi/360);
y(1)=1;
y0=y(1);
ym1=0;
for n=2:100
 ym2=ym1;
 ym1=y0;
 y0=a0*ym2+a1*ym1;
 y(n)=y0;
end
% Add Noise ?
y=y+an*randn(1,100);
% Original Form
A11=y(1);
A21=y(2);
c1=y(3);
A12=y(2);
A22=y(3);
c2=y(4);
%disp('+ + +');
%disp('Direct Calculation Using Two Instances');
```

```
A=[A11 A21;A12 A22];
                        %
c=[c1 c2]';
                  %
a=inv(A)*c;
ae1=a(1);
aoriga0(mm)=ae1;
ae2=a(2);
aoriga1(mm)=ae2;
aorig=[a';[ae1 ae2]]';
% Overdetermined
%disp('* * *');
%disp('Overdetermined - Pseudo-Inverse');
A=[y(1:eq);y(2:eq+1)]'
c=y(3:eq+2)'
a=pinv(A)*c
ae1=a(1);
apinva0(mm)=ae1;
ae2=a(2);
apinva1(mm)=ae2;
apinv=[a';[ae1 ae2]]';
% autocorrelation
rxx0=0;
rxx1=0;
rxx2=0;
rxxm1=0;
             %
for k=2:aut
 rxx0 = rxx0+y(k)^*y(k);
 rxx1 = rxx1 + y(k)^*y(k+1);
 rxxm1 = rxxm1 + y(k)^{*}y(k-1);
 rxx2 = rxx2+y(k)*y(k+2);
end
R=[rxx0 rxxm1; rxx1 rxx0]
c=[rxx1 rxx2]'
a=inv(R)*[rxx1 rxx2]'
a=flipud(a);
ae1=a(1);
aautoa0(mm)=ae1;
ae2=a(2);
aautoa1(mm)=ae2;
aauto=[a';[ae1 ae2]]';
end
```

figure(1) % plot(aoriga0,3,'r*') hold on plot(aoriga1,3,'b*') % plot(apinva0,2,'r*') plot(apinva1,2,'b*') % plot(aautoa0,1,'r*') plot(aautoa1,1,'b*') plot([a0 a0],[-5 5],'r') plot([a1 a1],[-5 5],'b') hold off axis([-1.8 2.8 -0.5 4])

figure(1)

ELECTRONOTES Vol. 23, No. 221 March 2014

EN#221 (17)