ELECTRONOTES
1016 Hanshaw Road
Ithaca, NY 14850

APPLICATION NOTE NO. 419

Dec 26, 2014

FINDING CHEBYSHEV POLYNOMIALS

Should one happen to wake up some morning with an overwhelming desire to obtain some Chebyshev polynomials, you are likely in the position of having a moderately valid impression of what they are (equal ripple) and with two notions of how to get them: (1) look them up in the literature, or (2) write a program that iteratively converges on them. The first choice may leave you bedazzled in such things as recursion equations and much more than you needed for a start. As usual, Abramowitz and Stegun [1] comes to the rescue and will tabulate some actual values. At some point however, we want to use our computers to give answers to myriad "what if" type questions. In this case, polynomial fitting and plotting software is invaluable, allowing us to offer an unlimited source of examples to learn from.

FITTING POLYNOMIALS

We have experience fitting polynomials to data points (either a perfect fit of an Nth-order polynomial to N+1 points, or as a least square fit) and this is always fun and it has many applications to real problems such as filtering, rate-changing, and interpolation [2]. Let's briefly review it:

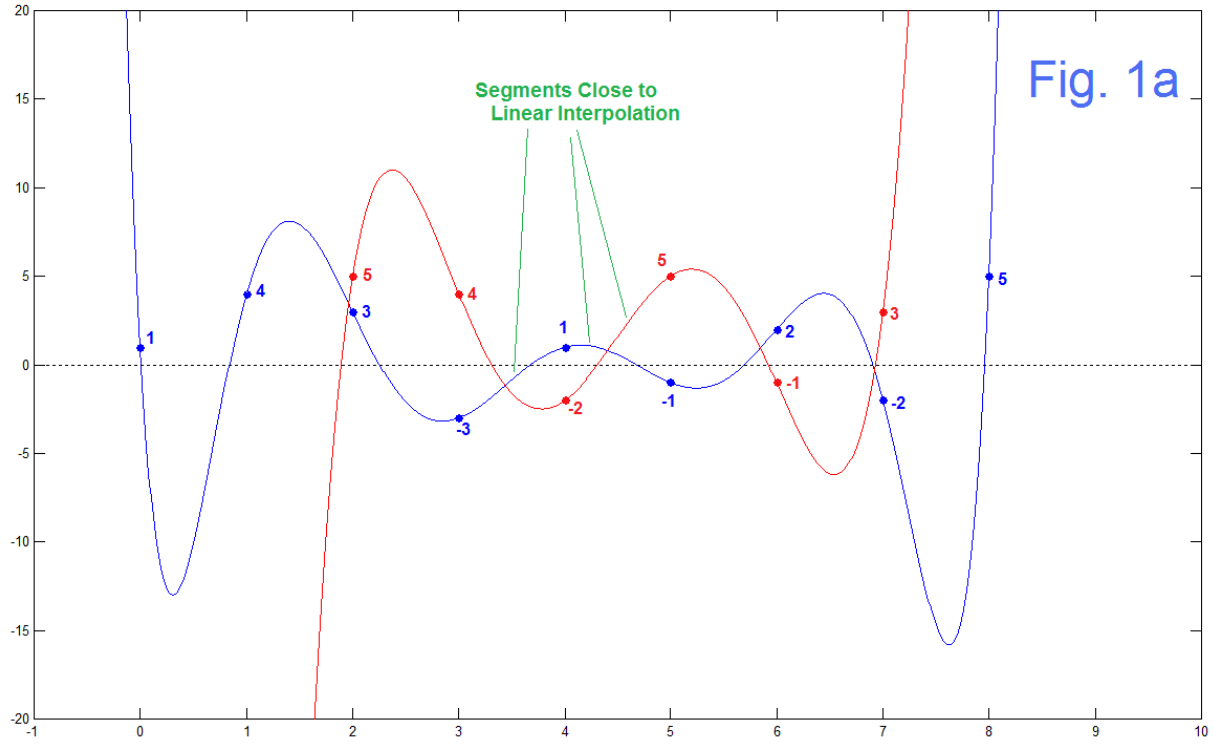
Fig. 1a and Fig. 1b are the same plots with different vertical scaling (to be discussed). These show two polynomial fits. The blue curve is an exact 8^{th}-order fit to the 9 blue points for horizontal integer points 0 to 8. The red curve is an exact 5^{th}-order fit to the 6 red points for horizontal integer points 2 to 7. The exact choices here are purely a matter of clarity of plotting. Here are the important observations:

(1) The polynomial curves fit exactly through the original points. The curve passes through the points, and in general the local min or max is not at the points (none here).

(2) The polynomial fits here were done using Matlab's ***polyfit*** and ***polyval***, although the calculations are easily hand-programmed (see below).

(3) The polynomials tend to ripple horizontally through the given points and then run off to +∞ or -∞. This is the purpose of rescaling in Fig. 1b to show more of the ends taking off. We have previously [3] noted that polynomials, being thus inherently vertical, are not that well suited to signals, which are horizontal.

(4) Indeed the polynomials are well-behaved close to their centers, and resemble linear interpolation there (red between 4 and 5, blue between 3 and 4 or 4 and 5) and would not be suitable interpolators on the ends.



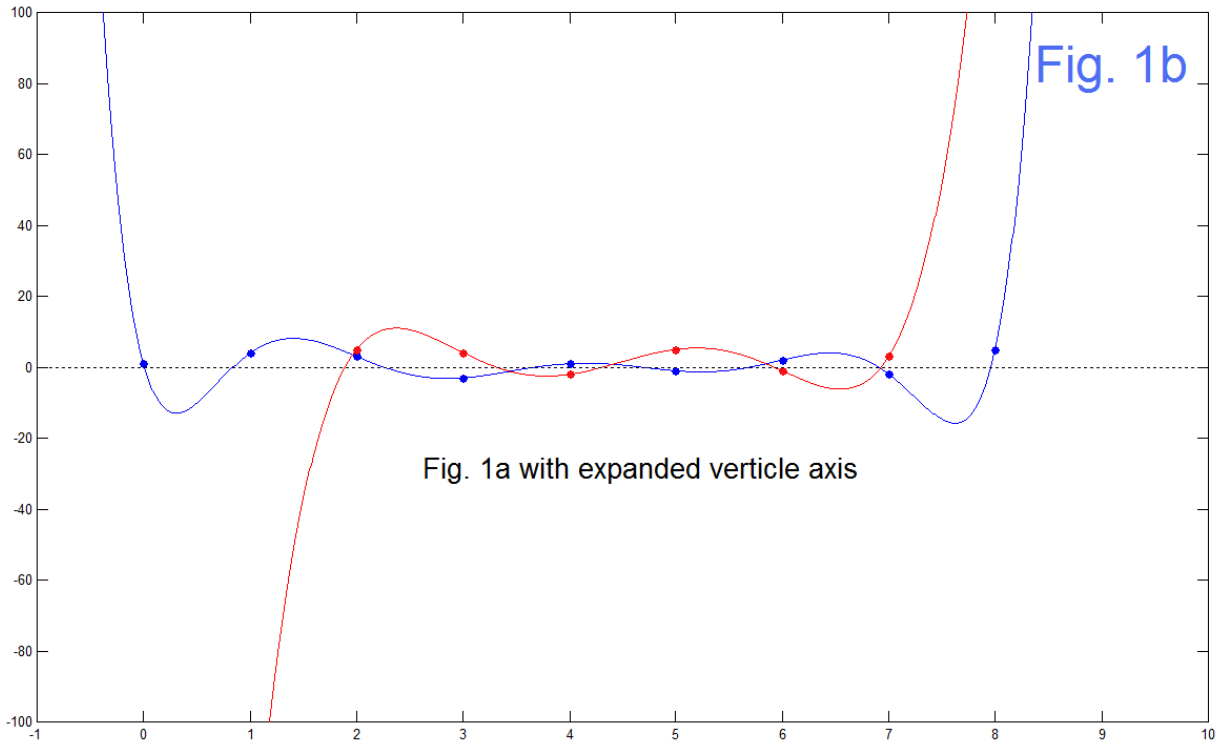


Fig. 1a with expanded verticle axis

THE HAND MATH

Just as a reminder, this is a math exercise of N+1 equations in N+1 unknowns based on just a polynomial equation:

$$P(t) = a_N t^N + a_{N-1} t^{N-1} + \ldots\ldots + a_1 t + a_o \qquad (1)$$

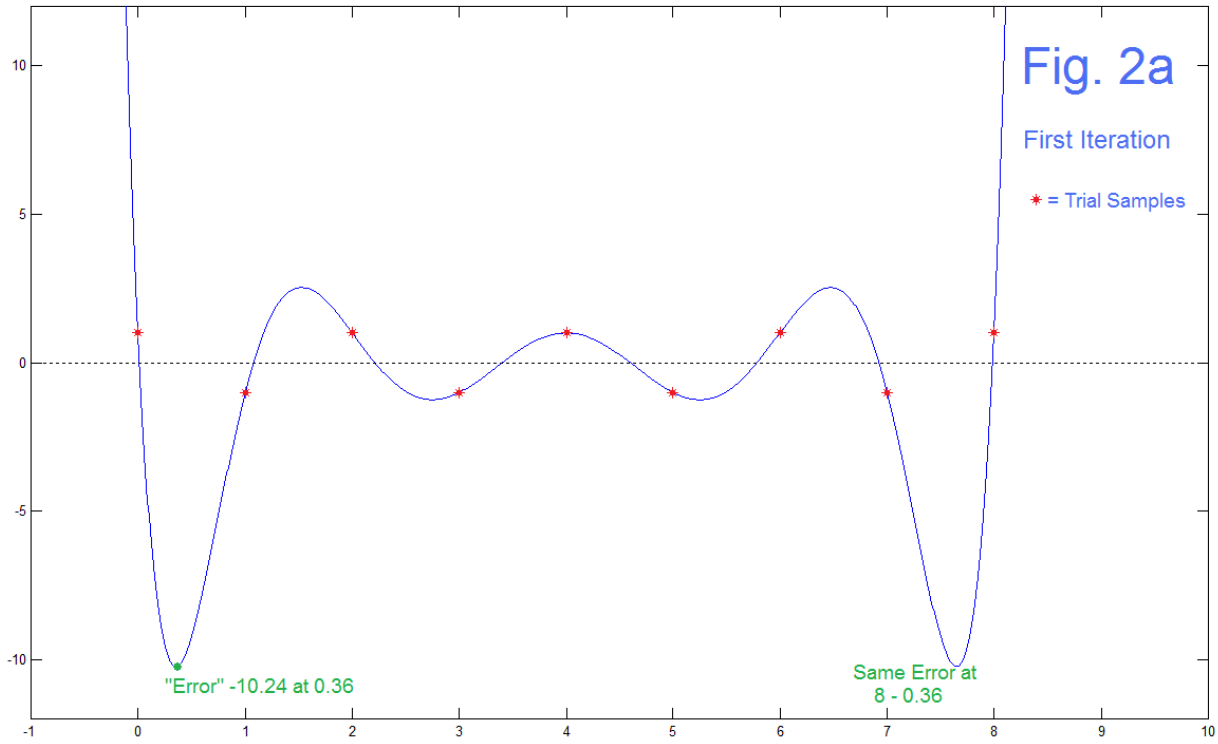
If we have N+1 samples of $P(t)$, we can solve for the N+1 values of a_n. The N+1 samples may be anywhere, not necessarily at integers. Once we have found the a_n, we can solve for $P(t)$ at any desired value of t, and indeed on a dense grid of t suitable for plotting what looks like a continuous function.

That's all there is to it.

CHEBYSHEV BY TRIAL AND ERROR

In engineering when we say "trial and error" we most often mean "iteration" and indeed, intelligent iteration with the specific notion of improvement with each new attempt, perhaps even with a mathematical assurance of some formal sense of a defined "optimal" result at the end.

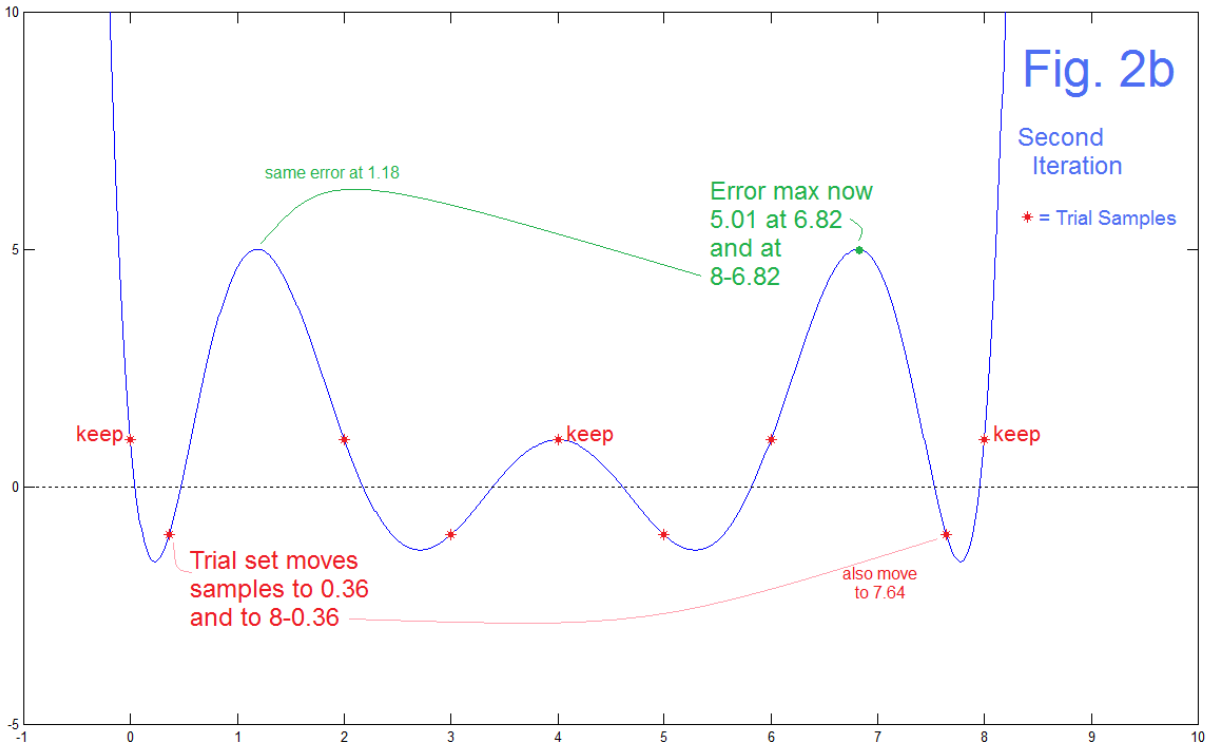
Accordingly we might well just try (to see what happens) a sequence of alternating +1's and -1's. Much as we used the sequence [1 4 3 -3 1 -1 2 -2 5] to generate the blue curve in Fig. 1a, we can consider a sequence [1 -1 1 -1 1 -1 1 -1 1] with the hope that it is a start toward an equal ripple. Fig. 2a shows the result.

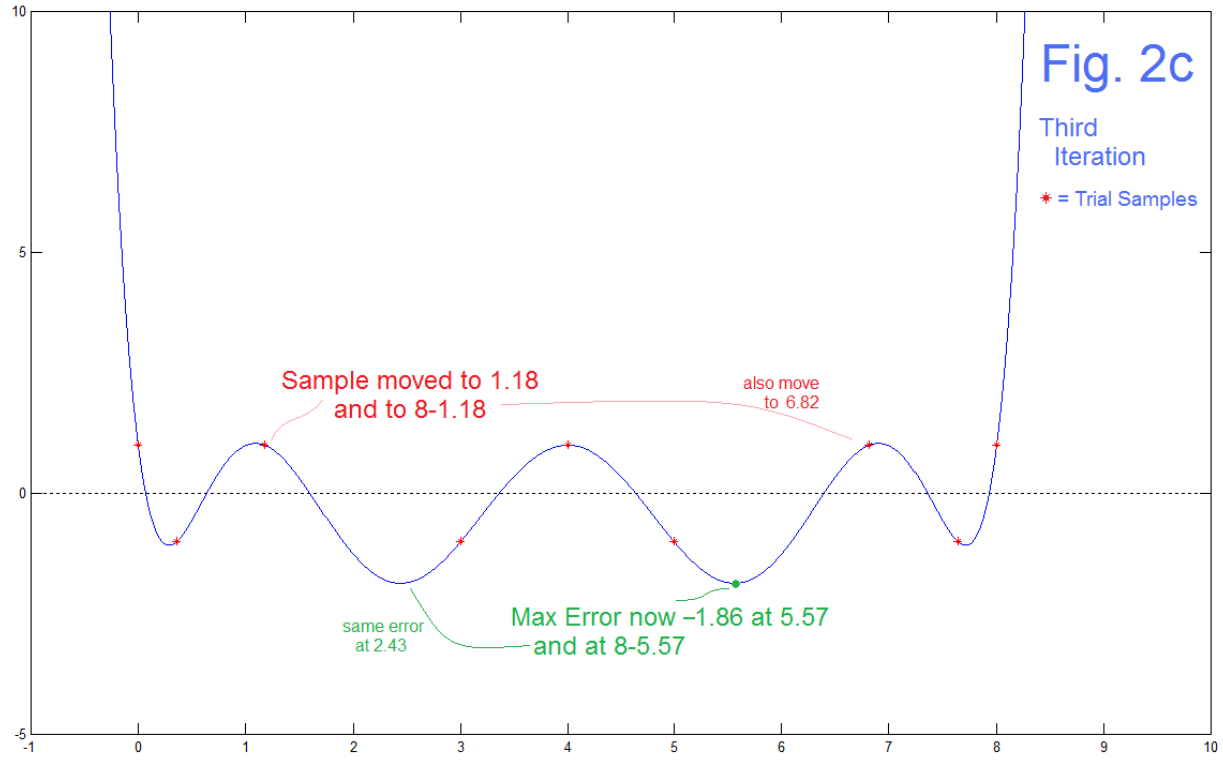


As we would have expected, the polynomial does go through the alternating series with no error on the defined points. If this were Chebyshev however, there would be no portion of the curve internal to the given samples where the value of the curve has magnitude greater than 1, and the local max/mins of the polynomial would all be equal. We note as well the symmetry which is expected on the basis of the generating samples. In addition, the two values on the ends, and the one in the middle (all equal to 1) are exactly right. So our concern is the regions of the curve that are outside the amplitude bounds of ±1. We have written code not just to solve for the polynomial and to plot it, but to also find the maximum absolute value between 0 and 8. This point is identified with the green dot, and occurs at about 0.36 with a value of -10.24, so this is clearly far outside the ±1 limits and we shall refer to this an "error". Note that due to the symmetry, the horizontal axis value of 8-0.36 = 7.64 should be the same error. Matlab either has a round-off difference or some sort of tie-breaker to give us just one answer. We are aware of the fact that there are two values. Our goal is the move positions on the horizontal axis to reduce the error, and to proceed iteratively.

Our second iteration, based on our experience with filter design [4] would be to move a defined sample to exactly the one where the error explodes. That is, we want to move a sample to 0.36 (with paired sample at 8-0.36). We do not want to move the ends, or the middle which are already where we want them to end up. Thus the sample at 1 is moved down to 0.36 and the sample at 7 moves up to 7.64. We rerun the program (Fig. 2b). Now the error is on the positive side and about half the magnitude, so we have an improvement. We did succeed in greatly reducing the large error in the vicinity of 0.36, as we intended.

In Iteration 3 (Fig. 2c) we now attack the new maximum by placing a sample at 1.18 and at 6.82 (8 – 1.18), while leaving the sample placed at 0.36 right where we moved it (this may eventually need further adjustment – that's the idea of getting a





convergence. In Fig. 2c, we see the improvement at 1.18 and at 6.82, and the max error now jumps to -1.86 at 2.43 and at 5.57. The error is smaller. As importantly, the error at 0.36 did not jump back up – it is in fact a bit better yet.

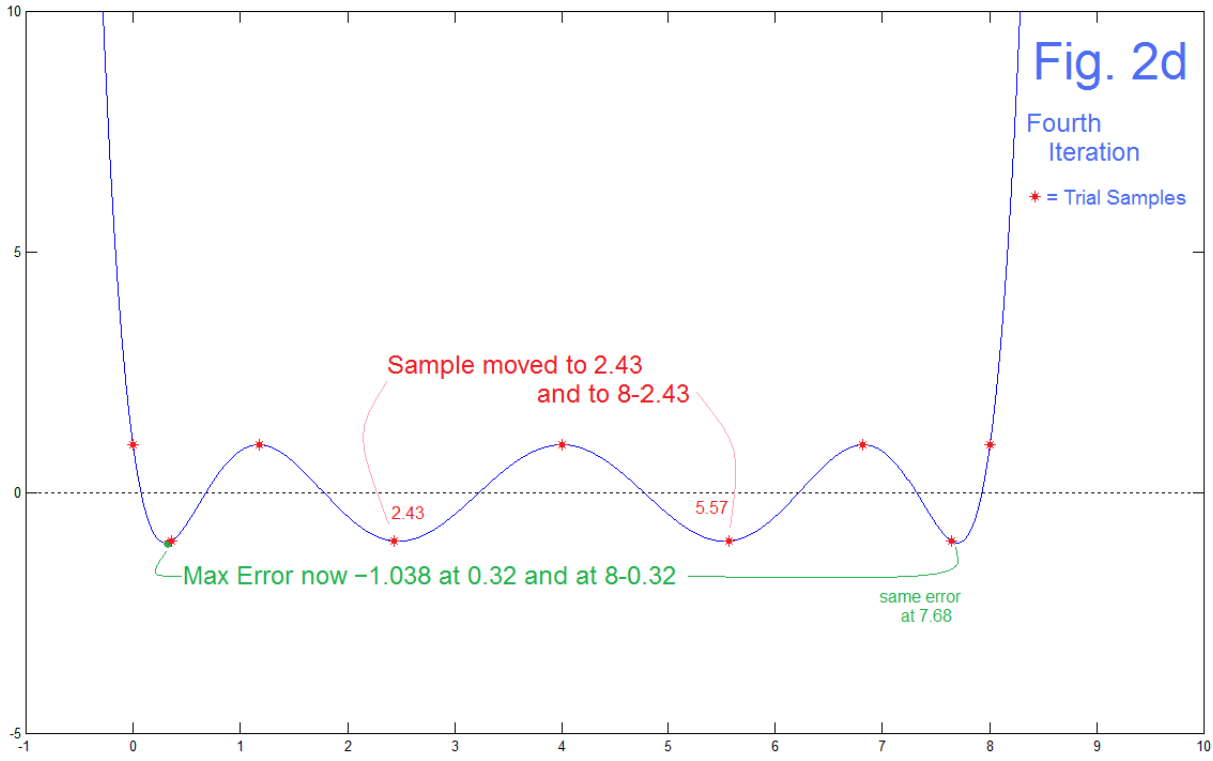
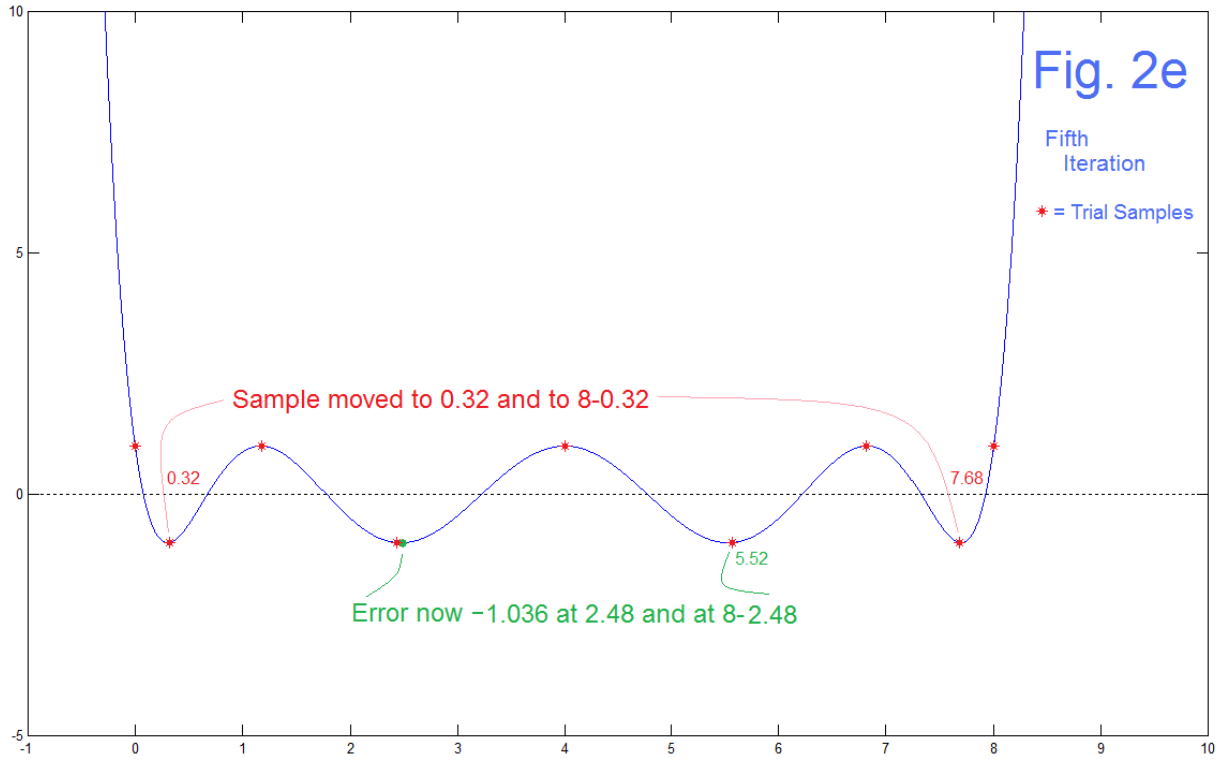
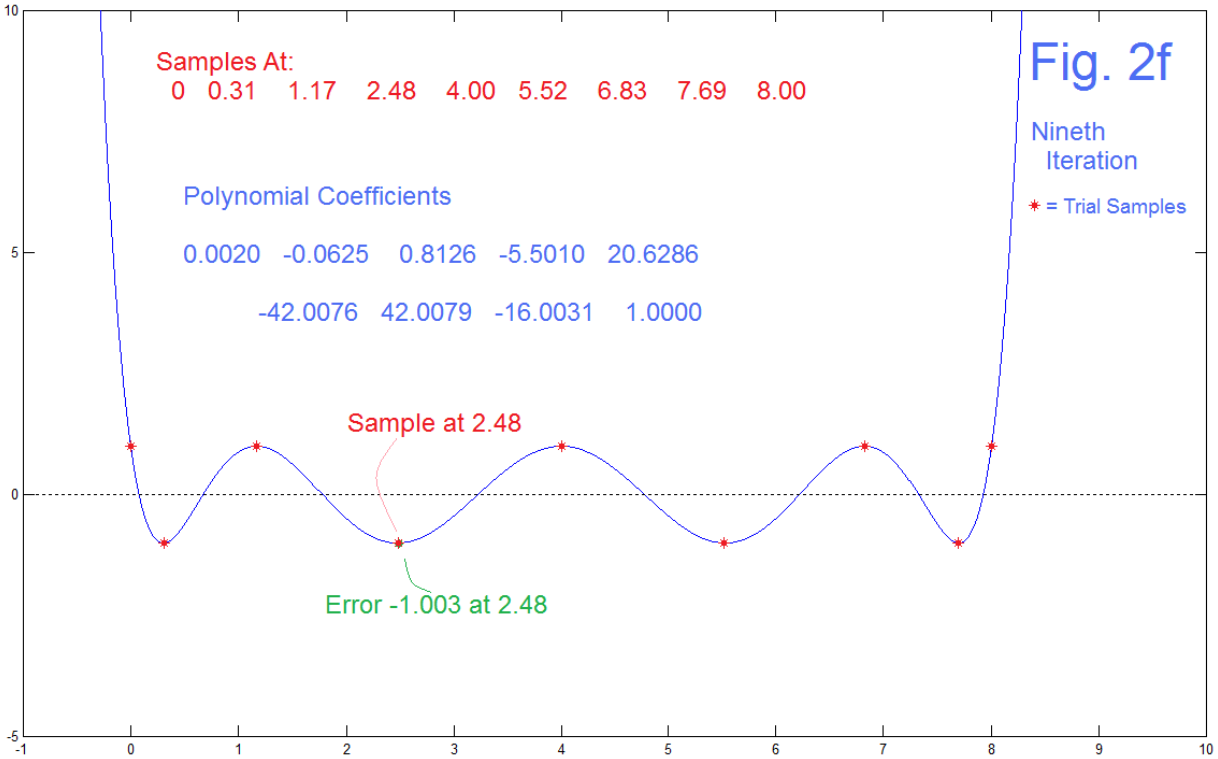


Fig. 2d shows Iteration 4, and we are getting close. The maximum error jumps back to the first internal bump, and wants to be moved from 0.36 to 0.32. Fig. 2f is Iteration 5 and we see the maximum error moves on to the third internal bump.





Skipping now a few additional iterations, we can stop at the 9th iteration of Fig. 2f where the error is now 0.3%. Unless we define the horizontal axis to more than two decimal places (which we are not attempting) the iterations alternate (are stuck). The result of Fig. 2f sure looks like the Cheybshev polynomial we were expecting. It is. Perhaps it is confusing that the polynomial coefficients (blue in Fig. 2f) don't ring any bells.

COMPARING COEFFICIENTS

If we really found the 8th-order Chebyshef polynomial in Fig. 2f, it should be close to:

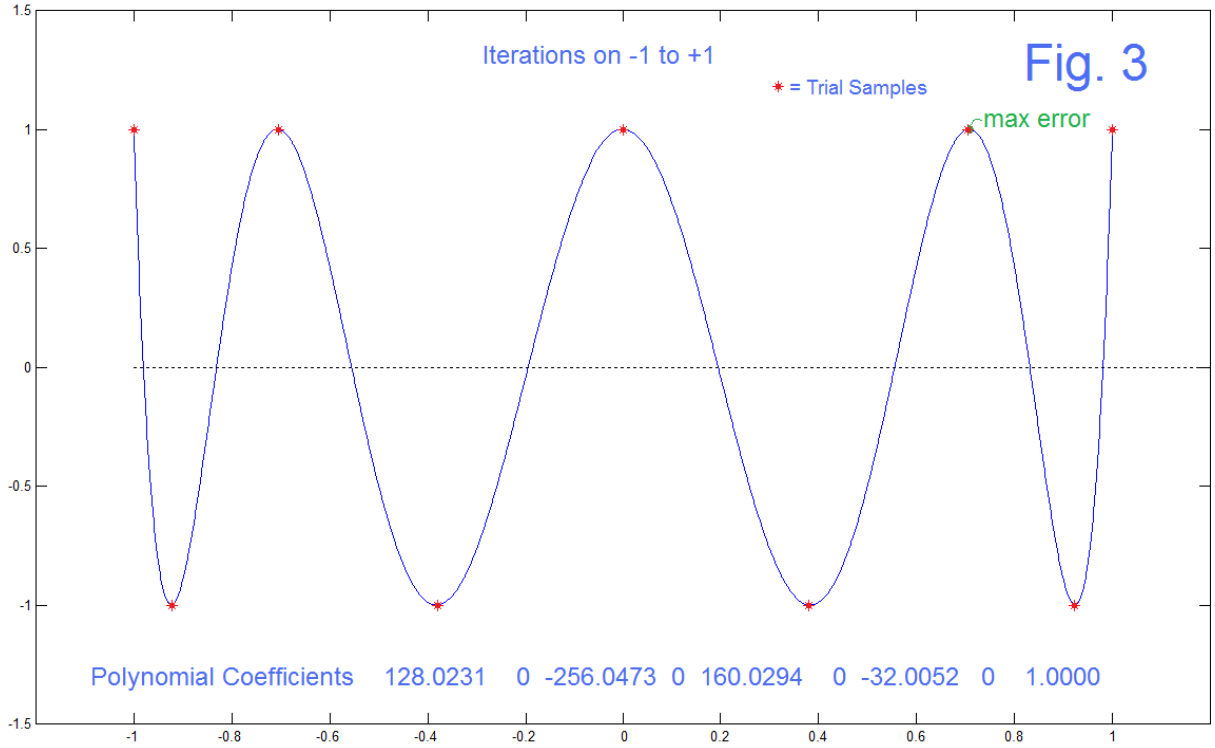
$$T_8(t) = 0.0020t^8 - 0.0625t^7 + 0.8126t^6 - 5.5010t^5 + 20.6286t^4 - 42.0076t^3 + 42.0079t^2 - 16.0031t + 1.0000 \quad (2)$$

So when we open the references [1] to see what T_8 should be, it's:

$$T_8(t) = 128t^8 - 256t^6 + 160t^4 - 32t^2 + 1 \quad (3)$$

which is "bell ringing" and simpler. If these are really the same, we should be able to get back and forth between them – the difference turning out to be the different interval over which the polynomial is defined. In the case of equation (3), and similar tabulations, the interval is t=-1 to t=+1. In our iterative fit, we assumed an interval t=0 to t=8. Sounds easy to reconcile?

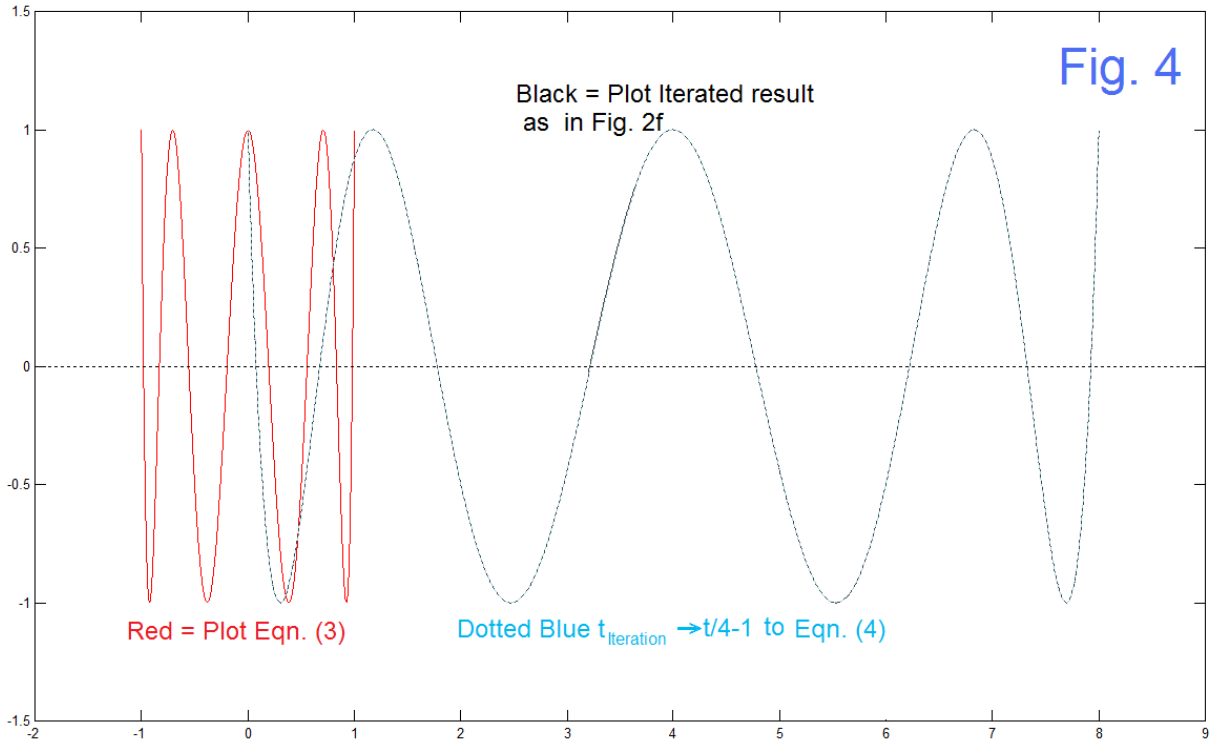
METHOD 1 Iterate in the Shifted Interval



In this case, we simply iterate as in the examples of Fig. 2, but now set the samples on the interval of -1 to +1. This is very easy to do, and following up on the original example, only takes about 5 minutes to reform the code and run it. The polynomial coefficients at convergence are printed on Fig. 3, and are a very good match to equation (3). This was easy.

METHOD 2 Do the Shift and Scale from Equation (3)

In Method 1 we showed that iteration on the interval -1 to +1 could lead to the textbook equation (3). Here we want to show that plotting equation (3) shifted (by +1) and scaled (by +4) will give the same result as the iteration on the original interval (0 to +8). The red curve in Fig. 4 is equation (3). The dotted light blue curve is obtained by taking the horizontal values 0 to 8, dividing by 4 (now 0 to +2) and subtracting 1 (now -1 to +1) and calculating with equation (3). These light blue results overplot the black curve from the original iterations on 0 to +8. This too was easy.



Method 3 – A Lot of Algebra

This is hard – tedious algebra that follows setting your mind to what needs to be done! Let's call t_1 the time for equation (3), the interval -1 to +1, and t_2 the time for the interval 0 to +8. Thus:

$$t_2 = 4(t_1 + 1) \qquad (4a)$$

or:

$$t_1 = \left(\frac{t_2}{4} - 1\right) \tag{4b}$$

It is convenient to use:

$$t_3 = \frac{t_4}{4} \tag{4c}$$

so that:

$$t_1 = t_3 - 1 \tag{4d}$$

Now the algebraic problem is in raising $(t_3 - 1)$ to the powers 8, 6, 4, and 2 as required by equation (3). Pascal's triangle (binomial expansion) to the rescue:

$$t_1^8 = t_3^8 - 8t_3^7 + 28t_3^6 - 56t_3^5 + 70t_3^4 - 56t_3^3 + 28t_3^2 - 8t_3 + 1 \tag{5a}$$

$$t_1^6 = t_3^6 - 6t_3^5 + 15t_3^4 - 20t_3^3 + 15t_3^2 - 6t_3 + 1 \tag{5b}$$

$$t_1^4 = t_3^4 - 4t_3^3 + 6t_3^2 - 4t_3 + 1 \tag{5c}$$

$$t_1^2 = t_3^2 - 2t + 1 \tag{5d}$$

$$t_1^0 = 1 \tag{5e}$$

Multiplying by the coefficients required by equation (3) and converting back to t_2 by dividing by the appropriate power of 4, we can obtain the terms:

$$\left(\frac{128}{4^8}\right)t_2^8 = 0.001953t_2^8 \tag{6a}$$

$$\left(\frac{-8 \times 128}{4^7}\right)t_2^7 = -0.0625t_2^7 \tag{6b}$$

$$\left(\frac{28 \times 128 - 256}{4^6}\right)t_2^6 = +0.8125t_2^6 \tag{6c}$$

$$\left(\frac{-56 \times 128 + 6 \times 256}{4^5}\right)t_2^5 = -5.5t_2^5 \tag{6d}$$

$$\left(\frac{70 \times 128 - 15 \times 256 + 160}{4^4}\right)t_2^4 = 20.625t_2^4 \tag{6e}$$

$$\left(\frac{-56 \times 128 + 20 \times 256 - 4 \times 160}{4^3}\right)t_2^3 = -42t_2^3 \tag{6f}$$

$$\left(\frac{28 \times 128 - 15 \times 256 + 6 \times 160 - 32}{4^2}\right) t_2^2 = 42 t_2^2 \qquad (6g)$$

$$\left(\frac{-8 \times 128 + 6 \times 256 - 4 \times 160 + 2 \times 32}{4}\right) t_2 = -16 t_2 \qquad (6h)$$

$$\left(\frac{128 - 256 + 160 - 32 + 1}{4^0}\right) = 1 \qquad (6i)$$

We note that these are very close to the values we obtained with iteration as printed on Fig. 2f. All we have really done here is to properly weight and add up terms of the same power along a general diagonal in equations (5a) through (5f).

DISCUSSION

The first part of this note, the polynomial fitting followed by iteration toward an equiripple-center region was done without any reference to equations and tables of Chebyshev polynomials. This is good because it is generally better to know how to obtain a result with minimal initial resources than it is to resort to canned data. We are relieved to see that this gives the same result as the text-book data. Further, we can imagine the development of a useful algorithm that performs the iterations automatically.

The discovery that the result of finding a polynomial from samples not centered and symmetric about zero is interesting, and understood in terms of even symmetry not requiring odd powers. Above our transformation of the scale and range indeed brought back all powers, odd as well as even. Thus the textbook data is a special case we need to be aware of. The even symmetry is of course attractive for filter designs.

REFERENCES

[1] Abramowitz, M., and I. A. Stegun, ***Handbook of Mathematical Functions***, Dover (1965), pg 795

[2] B. Hutchins, "Polynomial Fitting for Sample-rate Changing at Rational and Irrational FrequencyRatios," Electronotes Application Note No. 317, January 1992; http://electronotes.netfirms.com/AN317.PDF
"Interpolation, Decimation, and Prediction of Digital Signals," ***Electronotes***, Vol. 15, No. 164-167 (Special Issue F) July 1986.

[3] B. Hutchins, "Models – Good, And Bad: And The (Mis-)Use Of Engineering Ideas In Them," ***Electronotes***, Volume 22, Number 211 July 2012
http://electronotes.netfirms.com/EN211.pdf

[4] B. Hutchins, "Basic Elements of Digital Signal Processing: Filter Elements – Part 2" ***Electronotes***, Vol. 20, No. 198, June 2001, Section 3b
http://electronotes.netfirms.com/EN198.pdf

PROGRAM

Program here for complete documentation.

```
% an419

% Code below makes Fig. 1a and Fig. 1b of AN419
x=[0 1 2 3 4 5 6 7 8 ]
y = [1 4 3 -3 1 -1 2 -2 5 ]
p=polyfit(x,y,8)
xx=[-1:.01:9];
yy=polyval(p,xx);
x2=[2 3 4 5 6 7  ]
y2 = [5 4 -2 5 -1 3 ]
p2=polyfit(x2,y2,5)
yy2=polyval(p2,xx);
figure(1)
plot(xx,yy,'b')
hold on
plot(xx,yy2,'r')
plot(x,y,'ob')
plot(x2,y2,'or')
plot([-1 10],[0 0],'k:')
axis([-1 10 -100 100])
%axis([-1 10 -20 20])
hold off
figure(1)


% Code makes Fig. 2a
x=[0 1 2 3 4 5 6 7 8 ]
y = [1 -1 1 -1 1 -1 1 -1 1 ]
p=polyfit(x,y,8)
xx=[-1:.01:9];
yy=polyval(p,xx);
yyy=yy(101:800);
[yyymax1,ix1]=max(abs(yyy))
figure(2)
plot(xx,yy)
hold on
yyy(ix1)
ix1*.01
plot(ix1*0.01,yyy(ix1),'ro')
plot(x,y,'*r')
plot([-1 10],[0 0],'k:')
axis([-1 10 -12 12])
hold off
figure(2)

% Code made Fig. 2b through Fig. 2f
% x below is for last iteration
x=[ 0 .31 1.17 2.48 4 8-2.48 8-1.17 8-.31 8]
y = [1 -1 1 -1 1 -1 1 -1 1 ]
p=polyfit(x,y,8)
xx=[-1:.01:9];
yy=polyval(p,xx);
yyy=yy(101:800);
[yyymax1,ix1]=max(abs(yyy))
figure(3)
plot(xx,yy)
hold on
yyy(ix1)
ix1*.01
plot(ix1*0.01,yyy(ix1),'ro')
plot(x,y,'*r')
plot([-1 10],[0 0],'k:')
axis([-1 10 -5 10])
hold off
figure(3)
```

```
% not part of AN419  shows offset and scaling
figure(4)
t=-5:.01:10;
p8=[128 0 -256 0 160 0 -32 0 1];
yt=polyval(p8,t);
plot(t,yt)
hold on
plot(xx,yy,'r')
axis([-2 10 -2 2])
hold off
figure(4)

% calculating without polyval
t1=-1:.001:1;
t2=4*(t1+1);
P1=128*t1.^8-256*t1.^6+160*t1.^4-32*t1.^2 + 1;
P2=p(1)*t2.^8+p(2)*t2.^7+p(3)*t2.^6+p(4)*t2.^5+p(5)*t2.^4+p(6)*t2.^3+p(7)*t2.^2+p(8)*t2+p(9);
t4= (t2/4-1);
P2Ck = 128*t4.^8-256*t4.^6+160*t4.^4-32*t4.^2 + 1;
%  Fig. 4 of AN-419
figure(5)
plot(t1,P1,'r')
hold on
plot(t2,P2,'b')
plot(t2,P2Ck,'c:')
plot([-10 20],[0 0],'k:')
hold off
axis([-2 9 -1.5 1.5])
figure(5)


% Iteration to -1 to +1 range - Fig. 3 of AN-419
%x=[-1 -.75 -.50 -.25 0 .25 .50 .75 1]
x=[-1 -.923 -.705 -.38 0 .38 .705 .923 1]
y=[1 -1 1 -1 1 -1 1 -1 1 ]
p=polyfit(x,y,8)
xx=[-1:.001:1];
yy=polyval(p,xx);
[yyymax1,ix1]=max(abs(yy))
figure(6)
plot(xx,yy)
hold on
ix1*.001
plot(ix1*0.001-1,yy(ix1),'ro')
plot(x,y,'*r')
plot([-1 10],[0 0],'k:')
axis([-1.2 1.2 -1.5 1.5])
hold off
figure(6)
```