Linear Prediction

- Original application to speech was for data compression
  - Factor of about 10
  - But data compression is of less interest for speech these days
- Lives on because the compression method works by approximating the speech's spectrum
- Used for
  - Finding formants
  - Estimating pitch

LInear Prediction

So what is it?

We will call a sequence α of numbers *linearly predictable* if each term α_n of the sequence can be expressed as the same linear combination (weighted sum) of previous terms. That is, if there are coefficients $c_1, c_2, \ldots, c_k$, for some number k, such that for all $n > k$

$$\alpha_n = c_1\alpha_{n-1} + c_2\alpha_{n-2} + \cdots + c_n\alpha_{n-k}$$

or

$$\alpha_n = \sum_{i=1}^{k} c_i\alpha_{n-i}$$

Linear Prediction

Recall that

$$\sum_{i=1}^{k} c_i \alpha_{n-i}$$

is the expression for the nth term of the convolution of sequences c and α

So the statement

α is linearly predictable with coefficients $c_1 \ldots c_k$

is equivalent to

For all n from $k+1$ up to $length(\alpha)$ the nth term of the convolution of α with $1, -c_1, \ldots, -c_k$ is equal to 0.

Linear Prediction

Let’s try it out.
Here is an octave function that constructs linearly predictable sequences

```
function predseq=linpred(startseq,coeffs,howmany)
    predseq = zeros(1,howmany);
    sl = length(startseq);
    cl = length(coeffs);
    if(sl < cl)
        error("Can\'t get started!");
    end
    predseq(1:sl) = startseq;
    cf = fliplr(coeffs);
    for j=sl+1:howmany
        predseq(j) = cf*predseq(j-cl:j-1)';
    end
end
```

Linear Prediction

```
> s1 = linpred([1 2],[2 -1],10)
s1 =

    1    2    3    4    5    6    7    8    9
10

> c1=conv(s1,[1 -2 1]);
> c1(3:10)
ans =

   0   0   0   0   0   0   0   0
```

Linear Prediction

```
> s2=linpred([0 1 4],[3 -3 1],11)
s2 =

     0       1       4       9      16      25      36      49
64      81     100

> c2 = conv(s2,[1 -3 3 -1]);
> c2(4:11)
ans =

   0   0   0   0   0   0   0   0
```

Linear Prediction

Ok, but what has this got to do with speech?

Well, we are going to see that

- With lots of hedging...
- Approximately...
- Within limits...
- Over a sufficiently brief interval...
- It can often be useful to pretend that ...

Speech is linearly predictable

Linear Prediction

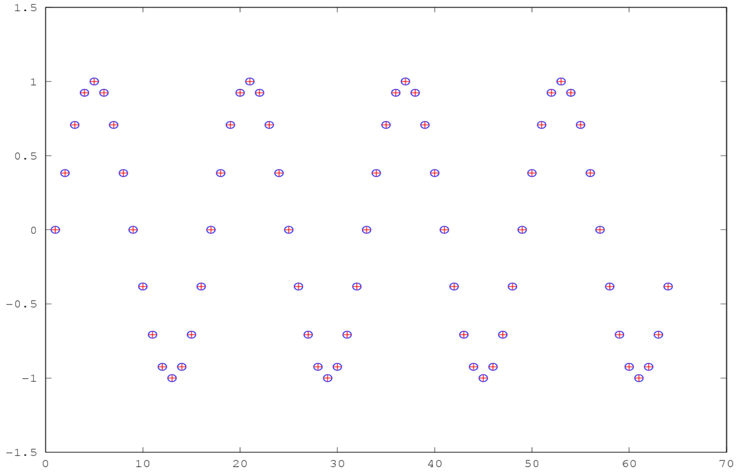
As a first step, note that sampled sinusoids are linearly predictable, because

$$sin((n+2)\times t) = 2\times cos(t)\times sin((n+1)\times t) - sin(n\times t)$$

From which it follows that a sinusoid with a period of p sample points can be linearly predicted with the coefficients $2cos(2\pi/p), -1$

Linear Prediction

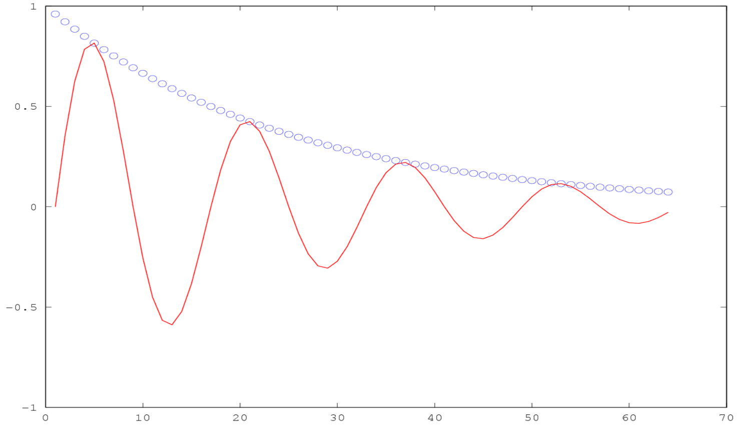
```
> s=linpred([0 sin(pi/8)],[2*cos(pi/8) -1],64);
> plot(s,'bo',sin((0:63)*pi/8),'r+');
```



Linear Prediction

An *exponentially damped* sinusoid is a sinusoid with the *nth* sample scaled down by d^n, where d is a "damping factor" ≤ 1.

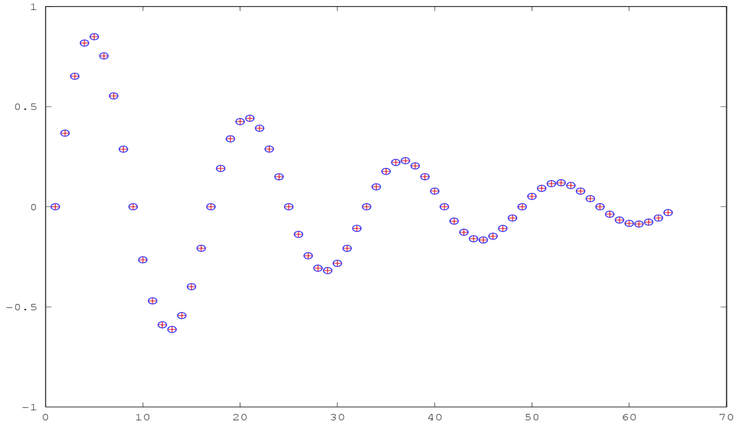
```
> edamp=0.96.^(0:63);
> plot(edamp,'bo',s.*edamp,'r);
```



Linear Prediction

For a damped sinusoid, the prediction coefficients are $2d \times cos(2\pi/p), -d^2$

```
> s2=linpred([0 0.96*sin(pi/8)],...
>            [2*0.96*cos(pi/8) -0.96^2],64);
> plot(s2,'r+',s.*edamp,'bo');
```

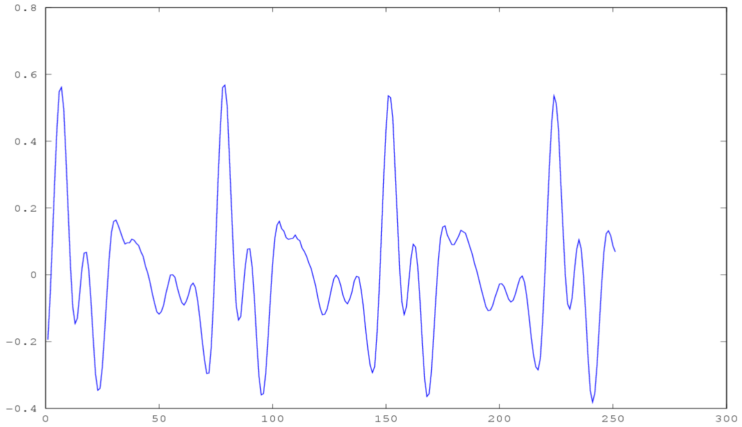


Linear Prediction

A quick look ahead

Here are several pitch periods of a vowel from the timit sentence sx133.wav.

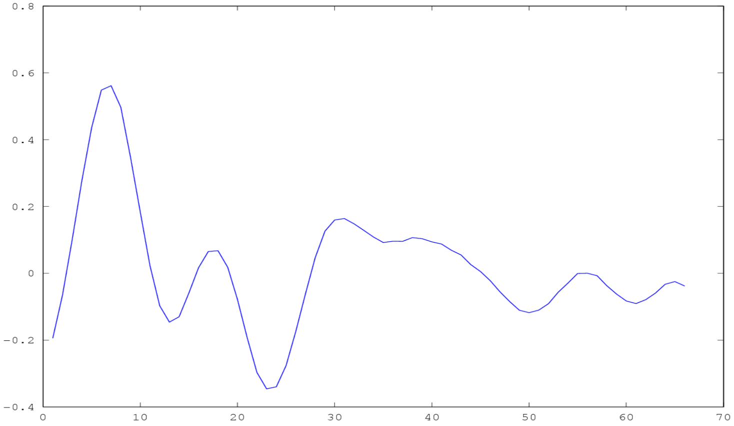
```
> plot(sx133(35450:35700))
```



Linear Prediction

Here is just the first one.

```
> plot(sx133(35450:35515))
```

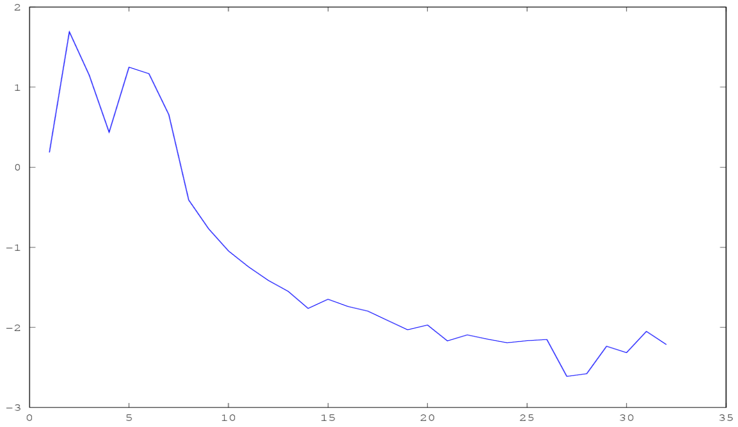


Not exactly a damped sinusoid, but maybe a sum of several?

Linear Prediction

Let's look at the spectrum of that pitch period.

```
> spec=log(abs(fft(sx133(35450:35515))));
> plot(spec(2:end/2))
```

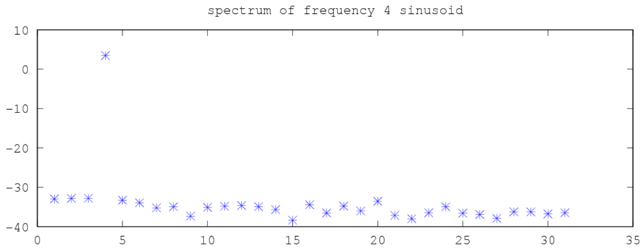


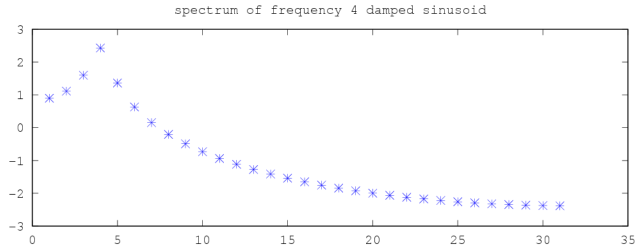
Linear Prediction

The spectrum of a sinusoid is a single spike in the position corresponding to the frequency of the sinusoid.
The spectrum of a damped sinusoid has a peak in the same position, but the peak is more spread out.

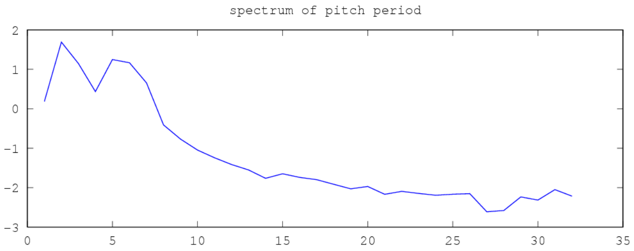
```
> spec=log(abs(fft(s)));
> dspec=log(abs(fft(s2)));
> subplot(2,1,1);plot(spec(2:32),'*');
> title("spectrum of frequency 4 sinusoid")
> subplot(2,1,2);plot(dspec(2:32),'*');
> title("spectrum of frequency 4 damped sinusoid")
```

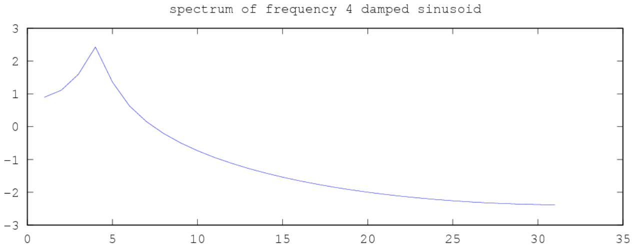
Linear Prediction





Linear Prediction





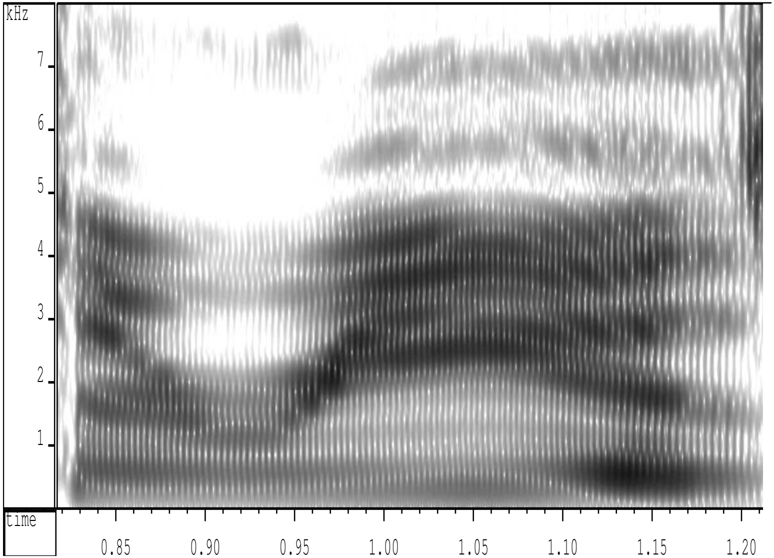
Linear Prediction

One way of viewing linear prediction is that it tries to approximate pitch periods as sums of damped sinusoids.

An alternative way is that it tries to approximate the spectrum as a sum of spectra of damped sinusoids.

For voiced speech, the largest damped sinusoids, or their spectra, correspond to *formants*, the high spectral energy regions that show up as as black bands on spectrograms.

Linear Prediction



Linear Prediction

We know that damped sinusoids are linearly predictable.

How about sums of them?

Back to the math...

Linear Prediction

Remember that convolution of sequences is isomorphic to multiplication of polynomials

$$
\begin{array}{ccc}
s_1 = a_0, a_1, \ldots, a_n & \Leftrightarrow & p_1 = a_0 + a_1 x + a_2 x^2, \ldots, a_n x^n \\
s_2 = b_0, b_1, \ldots, b_n & \Leftrightarrow & p_2 = b_0 + b_1 x + b_2 x^2, \ldots, b_n x^n \\
s_1 \otimes s_2 & \Leftrightarrow & p_1 \times p_2
\end{array}
$$

where $\otimes$ stands for convolution

Linear Prediction

The correspondence between sequences and polynomials is useful enough to have been given a name

The Z Transform

For historical reasons, the variable in the polynomial is written as z^{-1}.

$$\mathcal{Z}(a_0, a_1, \ldots, a_n) = a_0 + a_1 z^{-1} + \cdots + a_n z^{-n}$$

This doesn't make any difference to the coefficients that you get when you multiply two polynomials together.

So

$$\mathcal{Z}(s_1 \otimes s_2) = \mathcal{Z}(s_1) \times \mathcal{Z}(s_2)$$

or

$$\mathcal{Z}^{-1}(p_1) \otimes \mathcal{Z}^{-1}(p_2) = \mathcal{Z}^{-1}(p_1 \times p_2)$$

Linear Prediction

The Z transform applies to infinite sequences as well as finite ones.
An infinitely long polynomial is called a *power series*

$$\mathcal{Z}(a_0, a_1, a_2 \ldots) = a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots$$

Multiplication of power series works in the same way as multiplication of polynomials. That is, the coefficient of z^{-n} in a product is given by the formula for the *nth* term of the corresponding convolution.

Linear Prediction

Recall that we saw earlier that for a finite sequence α, the statement

α is linearly predictable with coefficients $c_1 \ldots c_k$

is equivalent to

For all n from $k+1$ up to $\mathit{length}(\alpha)$ the nth term of the convolution of α with $1, -c_1, \ldots, -c_k$ is equal to 0.

Linear Prediction

For infinite length α we can drop the restriction that $n \leq \textit{length}(\alpha)$ (because every n is less than $\textit{length}(\alpha)$) and say that

For all $n > k$ the nth term of the convolution of α with $1, -c_1, \ldots, -c_k$ is equal to 0.

Or in terms of polynomial multiplication

$$(\alpha_0 + \alpha_1 z^{-1} + \alpha_2 z^{-2} + \ldots) \times (1 - c_1 z^{-1} - \ldots - c_k z^{-k}) = Q$$

where Q is a polynomial with at most k terms

Linear Prediction

But, in polynomial multiplication and division

$$(\alpha_0 + \alpha_1 z^{-1} + \alpha_2 z^{-2} + \dots) \times (1 - c_1 z^{-1} - \dots - c_k z^{-k}) = Q$$

iff

$$(\alpha_0 + \alpha_1 z^{-1} + \alpha_2 z^{-2} + \dots) = \frac{Q}{1 - c_1 z^{-1} - \dots - c_k z^{-k}}$$

Linear Prediction

That is, the α_i can be computed by formal long division

For $\frac{1}{1-2x+x^2}$, we get

```
                 1 +2x    +3x^2    .....
                ______________________
1 - 2x + x^2)1 + 0x + 0x^2 ...
             1 - 2x +  x^2
             ____________
                 2x -  x^2
                 2x - 4x^2 + 2x^3
                 ----------------
                      3x^2 - 2x^3
                      3x^2 - 6x^3 + 3x^4
                             .....
```

Linear Prediction

Or, less laboriously, you could use the octave/matlab *filter* function

```
> filter(1,[1 -2 1],[1 0 0 0])
ans =

   1   2   3   4
```

Linear Prediction

Because (the inverse Z transform of) division by a fixed polynomial is in fact a linear shift-invariant filter.

Superposition: $\frac{c \times Q}{P} = c \times \frac{Q}{P}$, for any scalar c

Homogeneity: $\frac{Q + R}{P} = \frac{Q}{P} + \frac{R}{P}$

Shift invariance:

Shifting a sequence by by *n* places corresponds to multiplication of its Z transform by z^{-n}

$$\frac{z^{-n}Q}{P} = z^{-n}\frac{Q}{P}$$

Linear Prediction

The impulse response of such a filter will not in general be finite. The long division carries on indefinitely. Therefore these are called *infinite impulse response* (IIR) filters, as opposed to the finite impulse response (FIR) ones that we have seen previously.

In particular, a sinusoid is the impulse response of a filter with denominator polynomial $1 - 2cos(t)z^{-1} + z^{-2}$, suitably scaled and shifted, and a damped sinusoid is the impulse response of one with denominator $1 - 2dcos(t) + d^2z^{-2}$

Linear Prediction

Letting

$$P_1 = 1 - 2d_1 cos(t_1)z^{-1} + {d_1}^2 z^{-2}$$

and

$$P_2 = 1 - 2d_2 cos(t_2)z^{-1} + {d_2}^2 z^{-2}$$

we can see that the sum of the two damped sinusoids generated by expanding $\frac{1}{P_1}$ and $\frac{1}{P_2}$ respectively, can itself be generated by expanding $\frac{P_1 + P_2}{P_1 P_2}$.

Linear Prediction

Extending this reasoning, we can conclude that any sum of damped sinusoids can be generated by polynomial division and hence that any sum of damped sinusoids is linearly predictable.

Furthermore, by invoking the Fundamental Theorem of Algebra and partial fraction decomposition, we are going to be able to turn this around.

Linear Prediction

Theorem (Fundamental Theorem of Algebra)

Any polynomial P (of a variable x) with real number coefficients and constant term 1, can be factored as a product of terms of the forms $(1 + cx)$ *and* $(1 + bx + ax^2)$*, where a, b and c are real numbers and the* $(1 + bx + ax^2)$ *terms have no real number roots, i.e., can't be factored further into terms with real number coefficients.*

Linear Prediction

Theorem (Partial Fraction Decomposition)

If Q, P_1, P_2 are polynomials such that P_1 and P_2 have no common factors and the highest power in Q is less than the highest power in $P_1 P_2$, then

$$\frac{Q}{P_1 P_2} = \frac{Q_1}{P_1} + \frac{Q_2}{P_2}$$

where the highest powers in Q_1, Q_2 are less than the highest powers in P_1, P_2 respectively.

Linear Prediction

The conclusion is then that a power series generated as

$$\frac{1}{1 - c_1 z^{-1} - c_2 z^{-2} - \dots c_n z^{-n}}$$

where the factors satisfy the conditions for partial fraction decomposition (which turns out to be the case in practice for speech analysis) is a sum of series of the forms

$$\frac{A}{1 + bz^{-1}} \quad \textit{and} \quad \frac{A + Bz^{-1}}{1 + bz^{-1} + az^{-2}}$$

where the $1 + bz^{-1} + az^{-2}$ quadratics have no real number roots. The $\frac{A}{1 + bz^{-1}}$ series are exponentials. We shall assume that when they occur in speech analysis they are damped and we shall largely ignore them here.

Linear Prediction

What about the $\dfrac{A + Bz^{-1}}{1 + bz^{-1} + az^{-2}}$ series?

Well, the quadratic formula tells us that the (by assumption non-real) roots of $1 + bz^{-1} + az^{-2}$ are

$$\frac{-b \pm \sqrt{b^2 - 4a}}{2a}$$

If b^2 were $\geq 4a$, the roots would be real. Hence $b^2 < 4a$.

Linear Prediction

Let $d = \sqrt{a}$ and $c = \frac{b}{2d}$.
So b = 2dc.

Then

$$1 + bz^{-1} + az^{-2} = 1 + 2dcz^{-1} + d^2z^{-2}$$

Since $b^2 < 4a$

$$|b| < 2\sqrt{a} = 2d$$

and

$$|c| = |\frac{b}{2d}| < 1$$

Since every number between -1 and 1 is the cosine of some angle, we can write

$$c = -cos(\theta)$$

for some angle θ

Linear Prediction

Therefore

$$1 + bz^{-1} + az^{-2} = 1 - 2dcos(\theta)z^{-1} + d^2z^{-2}$$

for some angle θ and damping factor d, and $\dfrac{Az^{-1} + B}{1 + bz^{-1} + az^{-2}}$ is the Z transform of a sum of two damped sinusoids with period $2\pi/\theta$ and damping factor d.

Linear Prediction

The usual way of characterizing these damped sinusoids is not to consider the quadratic factors of the original polynomial $1 - c_1 z^{-1} \cdots - c_n z^{-n}$, but to look at the complex roots obtained by factoring the polynomial completely.
The quadratic formula tells us that the roots of

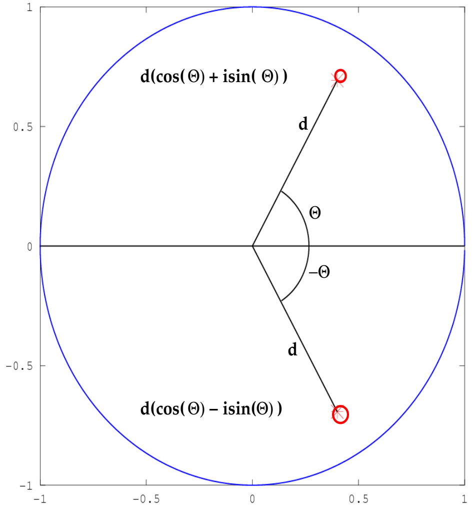
$$1 - 2dcos(\theta)z^{-1} + d^2 z^{-2}$$

are

$$z^{-1} = \frac{2dcos(\theta) \pm \sqrt{4d^2cos^2(\theta) - 4d^2}}{2d^2}$$
$$= \frac{cos(\theta) \pm \sqrt{cos^2(\theta) - 1}}{d} = \frac{cos(\theta) \pm isin(\theta)}{d}$$
$$= \frac{e^{\pm i\theta}}{d}$$

Linear Prediction

Since $1/e^{i\theta} = e^{-i\theta}$, the corresponding values for z itself are the complex conjugate pair $de^{\pm i\theta}$



Linear Prediction

A quotient $\dfrac{Q(z^{-1})}{P(z^{-1})}$ is undefined at points where the denominator $P(z^{-1})$ is equal to zero, i.e., at the roots of P. These are referred to as *poles* of the quotient. (The use of the term *pole* in mathematics derives from the history of map making.)

Linear Prediction

The angle of a pole determines the period of the corresponding damped sinusoid

A pole at angle θ corresponds to a sinusoid that repeats every $\frac{2\pi}{\theta}$ sample points.

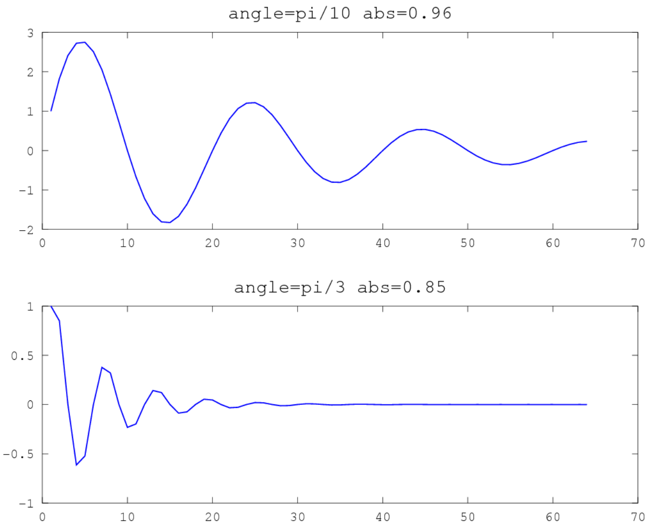
So the greater the angle, the higher the frequency of the sinusoid.

The distance of a pole from the origin determines the rate at which sinusoid dies away.

A pole near the origin gives rise to a sinusoid that dies away quickly.

A pole near the unit circle gives rise to a sinusoid that dies away slowly.

Linear Prediction



Linear Prediction

Note that there is nothing in the mathematics we have seen to prevent poles from lying outside the unit circle, that is, having a d value > 1. When this happens in speech analysis, it indicates a breakdown of the model of speech production on which linear predictive analysis is based. There are various methods of computing prediction coefficients to prevent this happening and of coping when it does. We won't go into them here, but you should remember that the model is not perfect.

Linear Prediction

What is the model?

The physics of tube resonators suggests that speech production can be idealized as passing either a series of isolated pulses (in voiced speech) or white noise (in frication) through a filter with an impulse response whose Z transform is of the form $\frac{1}{1 - c_1 z^{-1} - c_2 z^{-2} \cdots - c_n z^{-n}}$

In the case of pulses, most samples should be linearly predictable from previous ones, although not the pulses themselves, which come from outside the filter.

In the case of white noise, prediction should always be slightly wrong, but by about the same amount all the time.

Linear Prediction

Either way, we are looking for a set of coefficients that minimize the total error. That is what linear predictive analysis does. It looks for $c_1, c_2, \ldots c_n$ such that

$$s_k = c_1 s_{k-1} + c_2 s_{k-2} + \cdots + c_k s_0$$

$$s_{k+1} = c_1 s_k + c_2 s_{k-1} + \cdots + c_k s_1$$

$$\ldots$$

$$\ldots$$

$$s_N = c_1 s_{N-1} + c_2 s_{N-2} + \cdots + c_k s_{N-k}$$

Linear Prediction

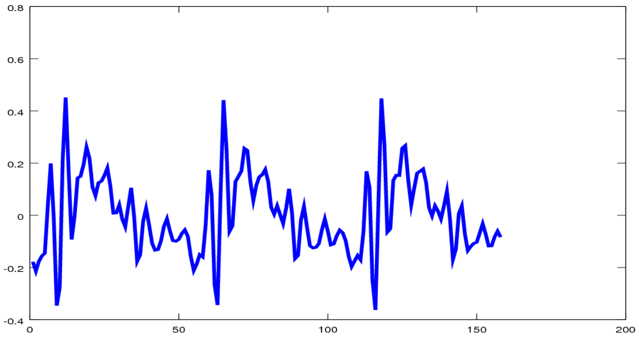
But these are N equations in k unknowns, where N is typically much larger than k and there is no perfect solution. So we make do with the c_j that give the best least squares fit.

Linear Prediction

Let’s try it on an example

Here are three pitch periods from the /i/ vowel in the “ri” syllable of “pizzeria” in SX133.wav.

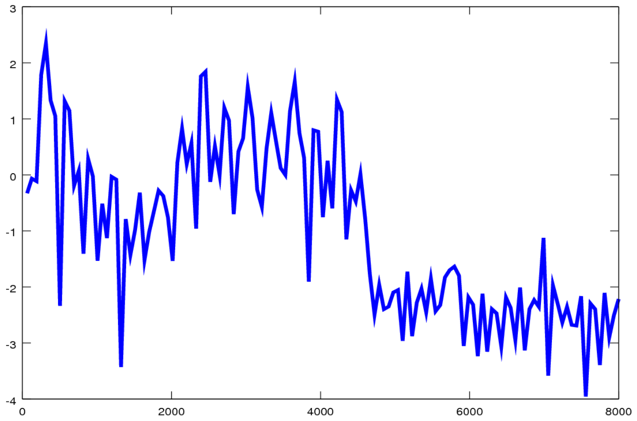
```
> pp3=sx133(16107:16243);
> plot(pp3);
```



Linear Prediction

And its spectrum

```
> spec=fft(pp3,256);
> plot((1:127)*8000/127,log(abs(spec(2:128))));
```



Linear Prediction

Now

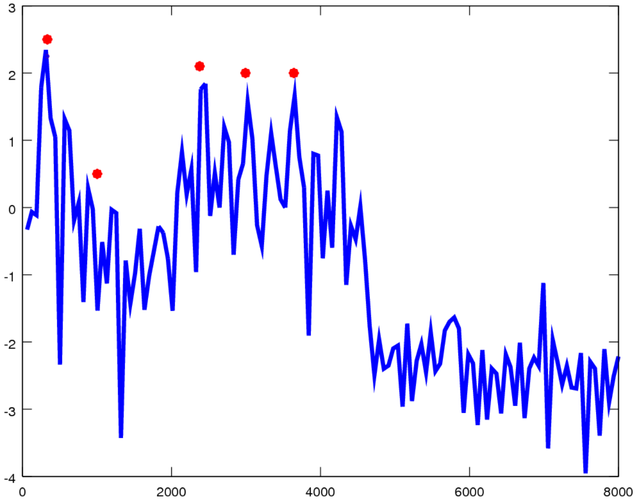
```
> k=18; %number of prediction coefficients
> m=zeros(length(pp3)-k,k);
> for j=0:length(pp3)-(k+1)
> m(j+1,:)=pp3(j+k:-1:j+1); % k samples
    before sample j, in reverse order
> end
> c=pinv(m)*pp3(k+1:end); % least squares
    solution to m*c=pp3(k+1:end)
> rts=roots([1; -c]);
```

Linear Prediction

```
> an=angle(rts);
> [an ind] = sort(an);
> rts=rts(ind); % sort roots in order of
    increasing angle
> an=an*8000/pi; %convert angles to frequencies
> pos=(an>0);
> an(pos)(1:5)' %first 5 positive roots
ans =
    338.21   1008.96   2386.85   3012.02   3660.20
> abs(rts(pos)(1:5))' % and their distance
    from the origin
ans =
    0.98134   0.83831   0.97737   0.93445   0.95862
```

Linear Prediction

Not all of the poles are formants, but the ones close to the unit circle are.



Linear Prediction

We might want to see not just pole locations, but the entire linear prediction spectrum, i.e., the spectrum of the filter's impulse response. That spectrum is obtained by a mathematical maneuver that at first encounter might look a bit like sleight of hand.

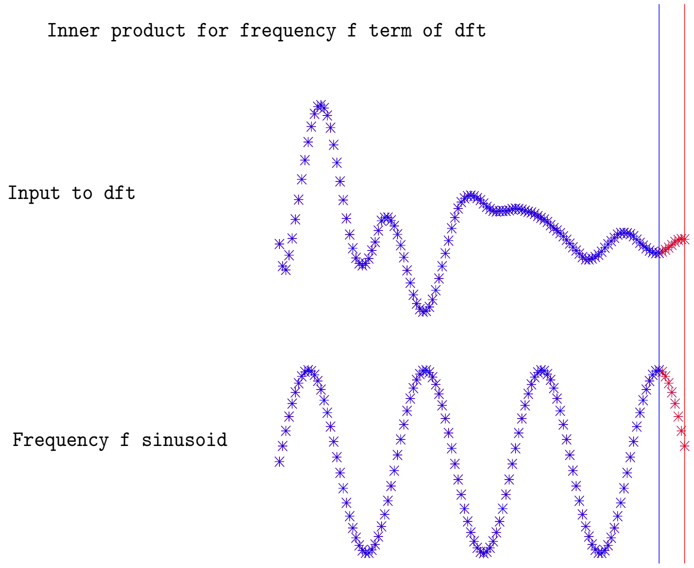
First of all, the impulse responses of interest are infinite, so we need to decide what we mean by the spectrum of an infinite sequence.

Well, for any given N, the dft of the first N elements of the sequence is defined for all frequencies up to $N/2$. If the dfts approach a limit at each frequency f as $N \to \infty$, it seems reasonable to adopt that limit for the value of the spectrum at f, for whatever range of frequencies f we are interested in.

Linear Prediction

It may be easiest to think about what is going on here in terms of the sine/cosine version of the dft. The frequency f term is computed by taking the inner product of the sequence with a sine and a cosine of frequency f.

Linear Prediction



Linear Prediction

As the damped input approaches 0, the difference in inner products for N and $N + k$ will get smaller. And of course this applies to the real and imaginary parts of the complex exponential version of the dft as well.

But wait...
What is this frequency f you are talking about?
There is no f in the definition we have seen of the dft.

Linear Prediction

The dft formula

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i(2\pi/N)kn}$$

gives us a spectral value for each number k from 0 to $N-1$.

But k is a frequency with respect to N. It is the number of times that the complex exponential $e^{-i(2\pi/N)kn}$ repeats over N samples. It is not a frequency in terms of samples per second, or dots per inch in a spatial domain.

Linear Prediction

If we have a speech wave sampled at 16 kHz (16000 samples per second) and we take the dft of 256 points, 2 kHz will be represented by the 32nd component of the dft, but if we take the dft of 128 sample points, 2 kHz will be represented by the 16th component.

What we want here is for the dft values corresponding to a given “absolute” frequency (really one relative to sampling frequency) to converge as we take longer and longer dfts.

Linear Prediction

In the dft, the kHz frequency f that we want is related to k, N and sf, the sampling frequency by

$$f = \frac{k}{N} sf \qquad or \qquad \frac{f}{sf} = \frac{k}{N}$$

So the dft formula can be rewritten

$$X_f = \sum_{n=0}^{N-1} x_n e^{-i(2\pi f/sf)n}$$

for f from 0 to $\frac{N-1}{N} sf$ in steps of $\frac{sf}{N}$

Linear Prediction

However, there is nothing to prevent us using the formula for X_f to obtain spectral values for “in between” frequencies f that are not multiples of $\frac{sf}{N}$, which is what we shall do.

That amounts to abandoning the dft for the discrete time/continuous frequency version of the fourier transform, but the two agree when f is chosen as $k\frac{sf}{N}$ for some k between 0 and $N-1$.

Now comes the sleight of hand...

Linear Prediction

Up to this point, we have been treating the powers of the variable z just as placeholders to help keep track of terms in convolution. But now we consider what happens if we substitute $e^{i(2\pi f/sf)}$ for z in the Z transform of the sequence $a_1, a_2, \ldots$

$$a_0 + a_1 e^{-i(2\pi f/sf)\times 1} + a_2 e^{-i(2\pi f/sf)\times 2} + \cdots = \sum_{n=0}^{\infty} a_n e^{-i(2\pi f/sf)n}$$

which we have just seen is the value of the spectrum of the sequence at frequency f (as long as the sum approaches a limit).

Linear Prediction

That's all very well, but how do we actually find the limit of the infinite series?

We use the fact that

$$\sum_{n=0}^{\infty} a_n z^{-n} = \frac{1}{1 - c_1 z^{-1} - c_2 z{-}2 \cdots - c_k z^{-k}}$$

if the series converges.
So to calculate the spectral value for a particular f we simply plug $e^{i(2\pi f/sf)}$ in for z in the denominator polynomial and divide 1 by the result!

(Having all of the poles inside the unit circle is a sufficient condition for the series to converge.)

Linear Prediction

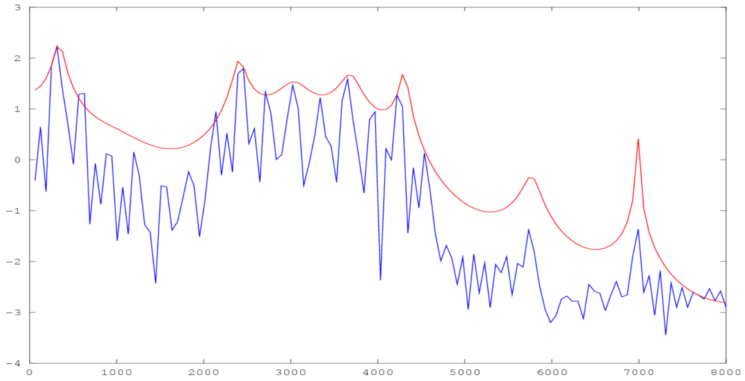
This can be done for whatever values of f we choose, but to display a spectrum, we would typically want to calculate the spectrum at a number of evenly spaced values, N multiples of some basic frequency f_0.

$$1-\sum_{n=1}^{k} c_n e^{-i(2\pi f_0/sf)}, 1-\sum_{n=1}^{k} c_n e^{-i(2\pi f_0/sf)\times 2}, \ldots, 1-\sum_{n=1}^{k} c_n e^{-i(2\pi f_0/sf)\times N}$$

But these are the terms of an N point dft of $1, -c_1, \ldots, -c_k$!

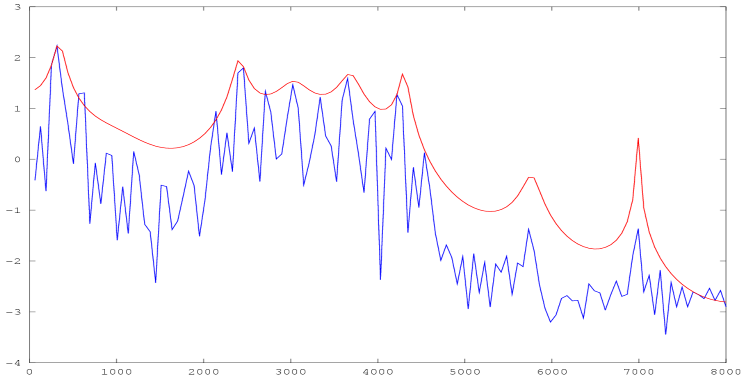
Linear Prediction

```
> lpspec=1./fft([1; -c],256);
> a=max(log(abs(spec)))/max(log(abs(lpspec)));
> %scale to same height
> plot((1:127)*8000/127,log(abs(spec(2:128))),...
> (1:127)*8000/127,a*log(abs(lpspec(2:128))),'r')
```



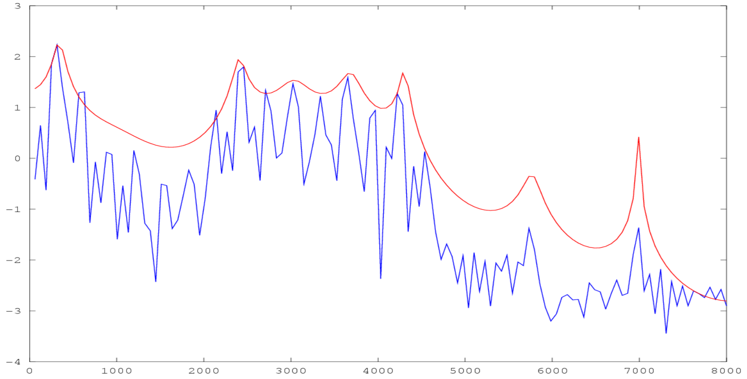
Linear Prediction

Our model of speech production says that the speech spectrum (in blue), is the result of filtering (convolving) some source with the filter whose spectrum is shown in red. We know that the dft of the convolution of two vectors is the pointwise product of the dfts of the vectors.



Linear Prediction

Since the display is actually of *logs* of spectra, the blue line should be the sum of the red with the log spectrum of the source.



How do we find the source?

Linear Prediction

Well convolving the source S with the filter whose impulse response is

$$\mathcal{Z}^{-1}\left(\frac{1}{1 - c_1 z^{-1}, \cdots - c_k z^{-k}}\right)$$

is supposed to yield the observed speech data D.

That is

$$D = conv(S, \mathcal{Z}^{-1}(\frac{1}{1 - c_1 z^{-1}, \cdots - c_k z^{-k}}))$$

or

$$\mathcal{Z}(D) = \mathcal{Z}(S) \times \frac{1}{1 - c_1 z^{-1}, \cdots - c_k z^{-k}}$$

Linear Prediction

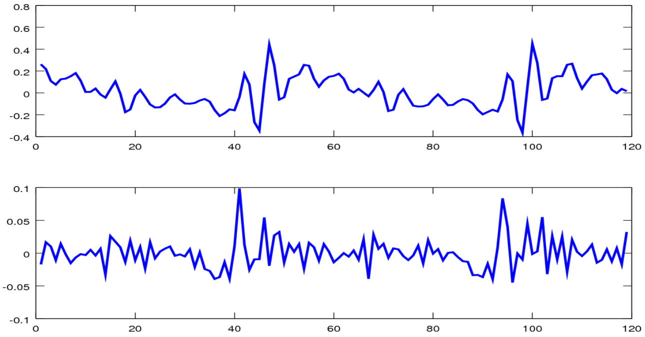
Multiplying both sides by $\dfrac{1}{1 - c_1 z^{-1}, \cdots - c_k z^{-k}}$ and taking inverse Z transforms, we see that S can be obtained by convolving D with $1 - c_1 z^{-1}, \cdots - c_k z^{-k}$.

Now recall that if a sequence is linearly predictable by $c_1, \ldots, c_k$, then the convolution of the sequence with $1 - c_1 \cdots - c_k$ will be 0 after the first k terms, and we computed a vector c that gave the best least squares fit to linearly predicting the segment pp3 = sx133(16107:16243).

Convolving with $[1; -c]$ is referred to as “inverse filtering” and the result is often referred to as the *error sequence* or *prediction residual*.

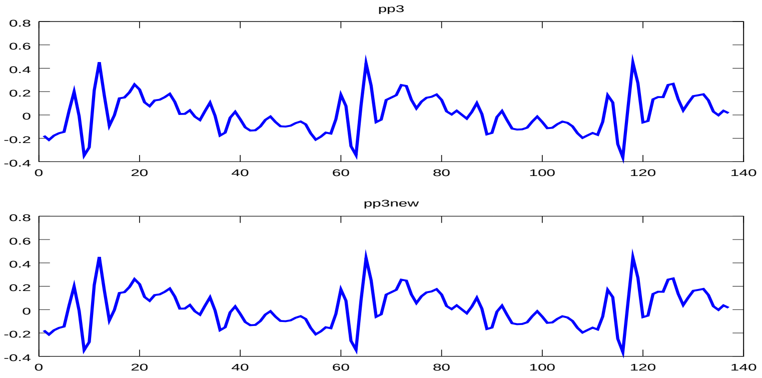
Linear Prediction

```
> resid=filter([1; -c],1,pp3);
> subplot(2,1,1);plot(pp3(19:end))
> subplot(2,1,2);plot(resid(19:end))
```



Linear Prediction

```
> pp3new=filter(1,[1;-c],resid);
> figure;subplot(2,1,1);plot(pp3);title("pp3")
> subplot(2,1,2);plot(pp3new);title("pp3new")
```



```
> max(abs(pp3-pp3new))
ans = 0
```

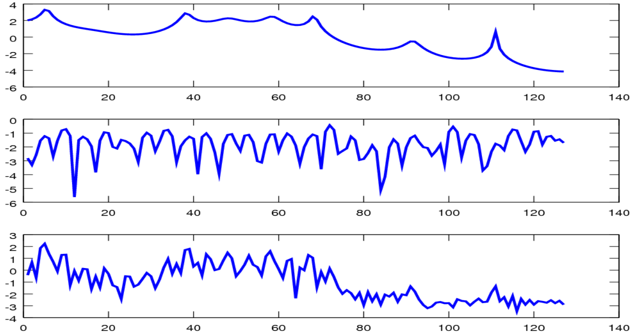
Linear Prediction

Note that this tells us nothing about speech, or about how successful our model of it is. It is a simple consequence of the fact that when you multiply and divide by the same polynomial, you get back to where you started.

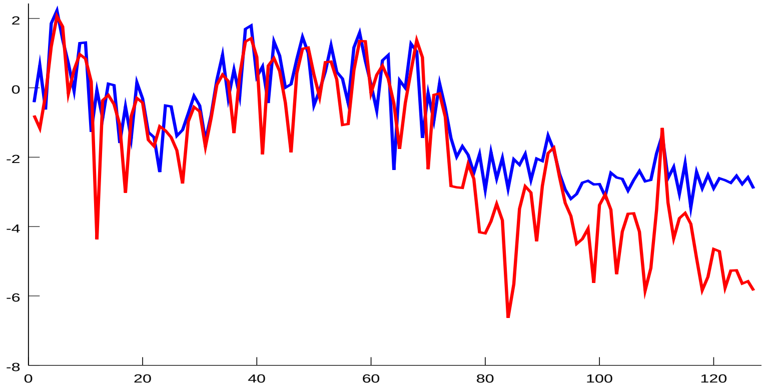
Reconstruction of the spectrum is not quite as perfect because of edge effects in convolution.

Linear Prediction

```
> residspec=fft(resid(19:end),256);
> figure;
> subplot(3,1,1);
    plot(plot(log(abs(1./fft(poly3,256)(2:128)))
> subplot(3,1,2);plot(log(abs(residspec)(2:128)))
> subplot(3,1,3);plot(log(abs(fft(pp3,256))(2:128))
```



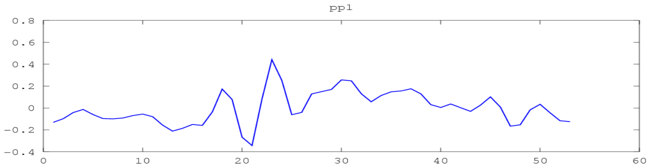
Linear Prediction

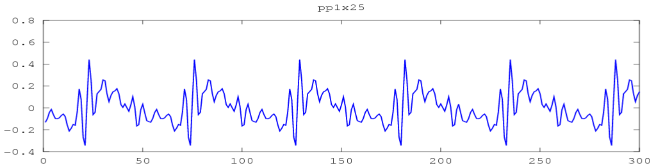


Artificial Speech

Waveform manipulation

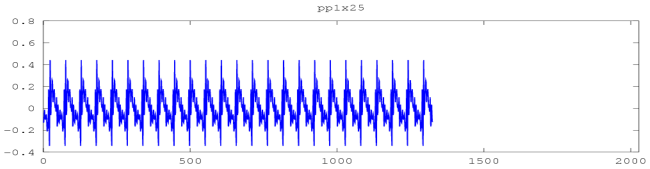
```
> pp1=pp3(43:95);
> pp1x25=repmat(pp1,25,1);
```

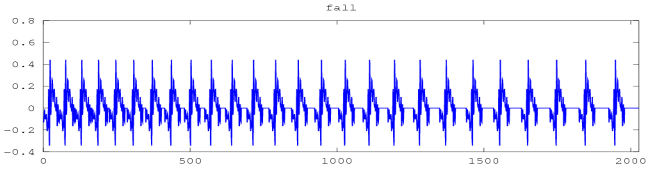




Artificial Speech

```
> fall=pp1;
> for j=1:25
> fall=[fall; pp1; zeros(2*j,1)];
> end
```

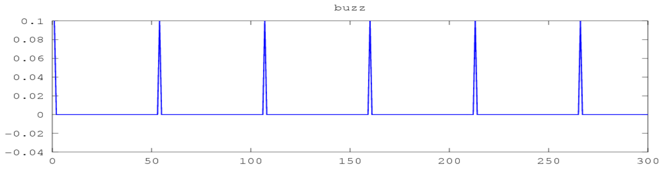


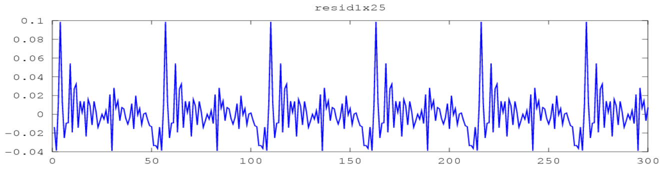


Manipulating Source and Filter

Create some artificial source . . .

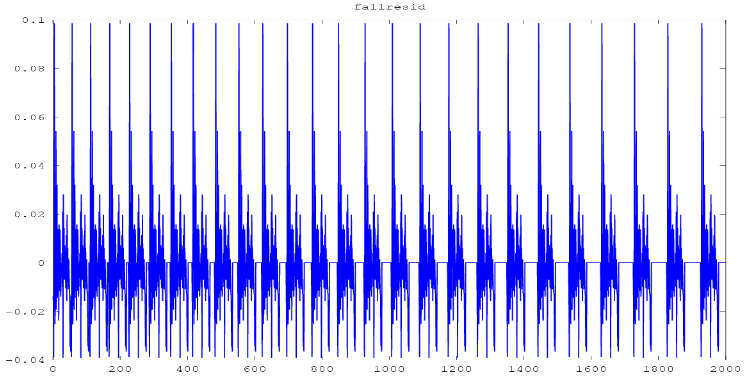
```
> buzz=repmat([1;zeros(52,1)],25,1);
> resid1=resid(43:95);
> resid1x25=repmat(resid1,25,1);
```





Manipulating Source and Filter

```
> fallresid=resid1;
> for j=1:25
> fallresid=[fallresid; resid1; zeros(2*j,1)];
> end
```

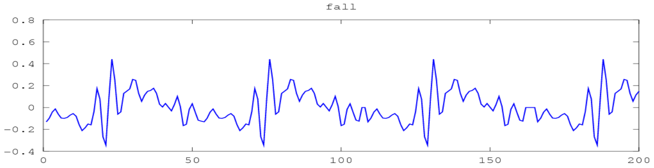


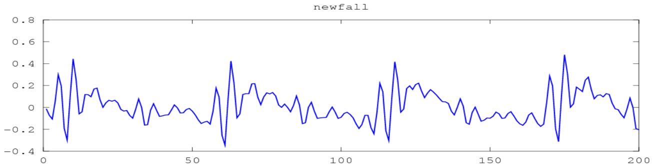
Manipulating Source and Filter

And filter it

- with the original filter

```
> newfall=filter(1,[1; -c],fallresid);
```





Manipulating Source and Filter

And filter it

- with a different filter

Recall

```
> real(angle(rts))'*8000/pi
ans =
 Columns 1 through 8:
   6933.75  -6933.75   5724.06  -5724.06   4272.88
-4272.88   3660.20
   -3660.20
 Columns 9 through 16:
   3012.02  -3012.02   2386.85 -2386.85 338.21
   -338.21    1008.96 -1008.96
 Columns 17 and 18:
   3918.37  -3918.37
```

Roots 11,12 correspond to F2 and roots 13,14 to F1

Manipulating Source and Filter

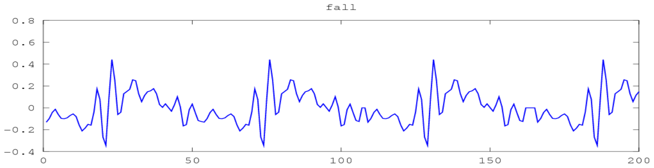
A new prediction polynomial based on F1=600, F2=1900

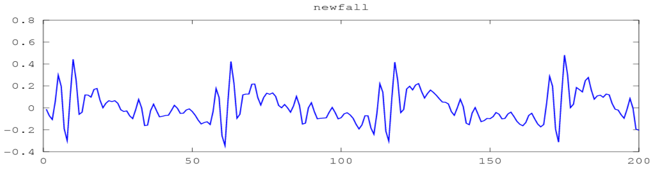
```
> a11=abs(rts(11))
a11 =  0.97737
> a13=abs(rts(13))
a13 =  0.98134

> newrts(11)=a11*(cos(1900*pi/8000)
>  +i*sin(1900*pi/8000));
> newrts(12)=newrts(11)';
> newrts(13)=a13*(cos(600*pi/8000)
>  +i*sin(250*pi/8000));
> newrts(14)=newrts(13)'
> newpoly=1;
> for j=1:length(newrts)
> newpoly=conv(newpoly,[1 -newrts(j)]);
> end
```

Manipulating Source and Filter

And filter with it

```
> newvowel=filter(1,newpoly,fallresid);
```





Manipulating Source and Filter

Make a diphthong by having the filter change over time

```
> % start with a longer residual
> fallresid=resid1;
> for j=1:50
> fallresid=[fallresid; res1; zeros(j,1)];
> end

> % create a series of values for F1 and F2
> f1s=linspace(600,250,length(fallresid));
> f2s=linspace(950,2500,length(fallresid));
>
> % start the diphthong with the first 18 frames
> % of newvowel
> diphth=zeros(length(fallresid)+18,1);
> diphth(1:18)=newvowel(1:18);
```

Manipulating Source and Filter

```
> newrts=rts;
>
> % compute the rest of the diphthong
> for j=1:length(fallresid)
> % make a new prediction polynomial for
> % each sample of the diphthong
>
> % first change the roots corresponding to
> % F1 and F2
> newrts(11)=a11*(cos(f2s(j)*pi/8000)
>    +i*sin(f2s(j)*pi/8000));
> newrts(12)=newrts(11)';
> newrts(13)=a13*(cos(f1s(j)*pi/8000)
>    +i*sin(f1s(j)*pi/8000));
> newrts(14)=newrts(13)';
```

Manipulating Source and Filter

```
> % multiply to make the polynomial
> newpoly=1;
> for k=1:length(newrts)
> newpoly=conv(newpoly,[1 -newrts(k)]);
> end;
>
> % predict the next sample point
> newcoeffs=-real(newpoly(2:end));
> diphth(j+18)=
>   newcoeffs*diphth(j+17:-1:j)+fallresid(j);
> end
```

Manipulating Source and Filter

```
> function wav = taper(wav);
>     % make the wave begin and end gradually
>     if(length(wav) < 320)
>         error("Input is too short!");
>     end
>     rampup = reshape((1:160),
>       size(wav(1:160)))/160;
>     rampdown = reshape((160:-1:1),
>       size(wav(1:160)))/160;
>     wav(1:160) .*= rampup;
>     wav(end-159:end) .*= rampdown;
> end
>
> diphth=taper(diphth);
```