Phase Unwrapping by Factorization

KENNETH STEIGLITZ, FELLOW, IEEE, AND BRADLEY DICKINSON, MEMBER, IEEE

Abstract—An algorithm for the numerical factorization of very high degree but well-conditioned polynomials is developed. This is used to factor the z-transform of finite-length signals, and the zeros are used to calculate the unwrapped phase. The method has been tested on signals up to 512 points in length. A complete Fortran77 program is given for the case of a real-valued signal.

Two related analytical issues are treated. First, the interpretation of phase unwrapping as an interpolation problem is discussed. Second, an explanation is given for the observed numerical difficulties in the method of phase unwrapping using adaptive integration of the phase derivative. The trouble is due to the clustering of the zeros of high degree polynomials near the unit circle.

I. Introduction

In a number of signal processing applications, the task of *phase unwrapping*, namely, the determination of the argument of a signal's transform, must be performed. A widely used algorithm for this purpose, due to Tribolet [1], is based on adaptive integration of the phase derivative. The FFT algorithm is used to evaluate the phase derivative on a grid of points on the unit circle in the complex plane. The grid is sufficiently fine to (presumably) allow the correct integrated value to be obtained. However, there is no certainty that a finer subdivision will not change the answer. (An interesting new algorithm described by McGowan and Kuc [15] is based on a noniterative method, but its computational practicality on realistically large examples has not yet been established.)

The difficulty encountered by Tribolet's algorithm can be attributed to the presence of zeros in the signal transform near the unit circle, and we will argue that as the signal length increases, it becomes more and more likely that such zeros appear. In the context of evaluating the complex cepstrum of a signal, we proposed to use numerical factorization of the associated polynomial (essentially the z-transform) to evaluate unwrapped phase [2]. The computation of unwrapped phase may be performed easily and accurately given the roots of the polynomial, so our main effort is directed toward an appropriate algorithm for factoring high degree polynomials to the limit of machine accuracy. Ultimately, the success or failure of a phase unwrapping algorithm will depend on its ability to place a root inside or outside the unit circle, and we therefore attack directly the problem of determining a root's position.

Most literature dealing with numerical factorization of polynomials places great emphasis on the difficulties caused by ill conditioning. A principal source of poor conditioning is the clustering of roots, but this is far from the only possibility. Wilkinson [3, ch. 2] presents a comprehensive overview of the numerical analysis of polynomials. It has been our experience that the polynomials determined by the z-transforms of finite records of signals are almost always adequately conditioned. This agrees with the observation in [3, p. 47] that polynomials with complex zeros arising in practice generally have well-conditioned zeros. As another example, in their empirical study of techniques for finding the zeros of linear phase FIR digital filter transfer functions, Schmidt and Rabiner [4] found that class of polynomials to have well-conditioned zeros.

(Analogously, 18th and even 24th order linear prediction polynomials are routinely implemented in direct form without numerical difficulties even though recursive filters—especially narrow-band varieties—may require extraordinary coefficient accuracy in such implementations [5]. The linear prediction filters are "nice" in the sense that their poles are well spread out in angle in the z-plane.)

In this paper, we first develop a numerical factorization algorithm for high-degree polynomials and give some examples of its application to phase unwrapping. We then take up two related theoretical topics. An interpretation of phase unwrapping as an interpolation problem is presented, and we show that Tribolet's algorithm, based on a sufficiently fine grid, yields the same unwrapped phase function as one based on factorization. Finally, we give a theoretical explanation of the observed clustering of polynomial zeros near the unit circle in the z-plane.

II. Outline of the Algorithm

Our philosophy in constructing a factorization algorithm useful for our special purposes is as follows: We largely ignore the problems which usually absorb the attention of most root-solvers—the treatment of repeated roots and saddle points—on the grounds that severe instances of these problems are very rare and can be circumvented by perturbing the problem slightly. (For a discussion of the problems of testing root-finders see [13]. For examples of such programs, see its references, and [14].) On the other hand, we give very careful attention to over- and underflow problems, which are unavoidable when dealing with polynomials of degree 250 or 500, say. Our goal is to produce an algorithm that is not necessarily efficient, but is easy to use and very robust with respect to over- and underflow problems, and therefore can deal with very high degree polynomials.

Fig. 1 shows the an informal outline of the algorithm, which is based on the quadratically convergent Newton–Raphson method. The iteration kernel is therefore

Manuscript received October 2, 1981; revised April 1, 1982. This work was supported in part by NSF Grants ECS-7916292 and ENG7728523 and U.S. Army Research-Durham Grant DAAG29-79-C-0024.

The authors are with the Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ 08544.

INFORMAL DESCRIPTION OF PROGRAM DPROOT

```
initialize kount := 0
m := mm (=original degree of polynomial)
(beginning of main loop)
10: if m ≤ 0, we are done, return
    let z be a point on unit circle at angle kount
    ktry=0
    r1 := amin**(1/m)
    r2 := amax**(1/m)
    (m is current degree of polynomial)
    (beginning of inner loop)
    20: if |z| < r1 or |z| > r2, go to restart
        if ktry ≥ ktrym, go to restart
        ktry := ktry + 1
        if kount ≥ kmax, exit with error message
        kount := kount + 1
        p := f(z)
        if |p| > amax, go to restart
        pp := df(z)/dz
        if |pp| < amin, go to restart
        if |p| ≤ ε  go to deflate
        z := z - p/pp
        go to 20
    (end of innner loop)
(end of main loop)

300: restart
    let z be a point on the unit circle at angle kount
    ktry := 0
    go  to 20

500: deflate
    polish z in original polynomial to get root w
    stop polishing and deflate if w leaves original annulus
    stop polishing and deflate after kpolm tries
    go to restart if |p| > amax or |pp| < amin
    501: deflate current polynomial using z; m := m-1 or m-2
    go to 10
```

Fig. 1. Outline of the root-solver *subroutine dproot* in an informal language. The statement labels correspond to statement numbers in the Fortran77 program.

$$z \leftarrow z - f(z)/f'(z)$$

where f is the polynomial. As is standard practice in root solving, after a root z is found, we go back to the original polynomial and "polish" it, refining it by Newton-Raphson iterations using the *original* polynomial to get the declared root w. The current reduced-order polynomial is then deflated (by z and not w, more about this in the next section). The main structure of the program consists of two nested loops: The outer one counts the number of roots found, and the inner one searches for a root using the Newton-Raphson iteration on the current (deflated) polynomial.

The two main interuptions in the main loops are shown at the bottom of Fig. 1. The first is the *restart* section, which can be reached from several points in the algorithm. Whenever (1) we have reached a point where an over- or underflow condition is likely, or (2) the number of Newton-Raphson iterations in the inside iteration loop from the current starting point (*ktry*) exceeds a preset limit (*ktrym*), we restart the iteration from a new point on the unit circle. The new starting point is determined simply by

$$z \leftarrow e^{\sqrt{-1} * kount}$$

where *kount* is the total number of iterations so far. Thus, no starting point is ever repeated. The second interruption of the main nested loops occurs at convergence to a root, and is shown in the *deflate* section.

Synthetic division using complex arithmetic is used to evaluate the polynomial and its derivative, and double precision complex arithmetic is used throughout. When a complex root is found, its conjugate is also extracted. (The case of complex coefficients is discussed later.)

III. Numerical Considerations

It is clearly very important in dealing with high degree polynomials to keep close to the unit circle. We do this by first defining the "large" number amax, and the "small" number amin; these numbers should fit very easily in a computer word and not cause over- or underflow. In our case (64-bit double-precision words) we have used

$$\text{amax} = 2^{32}$$

and

$$\text{amin} = 1/\text{amax}.$$

We then demand that the current candidate root z be within the annulus defined by

$$\text{amin}^{1/m} \leqslant |z| \leqslant \text{amax}^{1/m}$$

where m is the *current* degree of the polynomial. Thus, the annulus expands as the polynomial decreases in degree, allowing the search for roots far from the unit circle only when the degree is low. The iterations during the polishing phase are done only within the *original* annulus, and we might therefore mistakenly reject valid roots far from the unit circle by restarting. To avoid this, we do not restart if the *root* leaves the annulus during polishing, but only if the magnitude of $f(z)$ or $f'(z)$ violates our tests.

We also require that the values of the polynomial and its derivative satisfy

$$|f| \leqslant \text{amax and } |f'| \geqslant \text{amin}.$$

By this means overflow is prevented during the Newton-Raphson iteration step in which f is divided by f'.

The reason why we deflate by the "unpolished" root z, and not by w, is really simple. As we deflate, the roots of our current polynomial will drift (hopefully slightly) from those of our original polynomial. The polished roots will therefore tend to differ more and more from the roots of the current polynomial, and deflating by the polished root will mean extracting a relatively incorrect root. On the other hand, if we do not drift too much, each root of the current polynomial will be close enough to a root of the original to be "captured" by it, and the second-order convergence of Newton-Raphson will allow the polishing to take place to full precision in a few iterations–usually 1 or 2 in our experience. For some further discussion of deflation and polishing, see [3, p. 55-67].

In iterative algorithms of this sort it is always necessary to specify the tolerance used to determine convergence, the variable ϵ. We have been careful to base the operation of the entire algorithm on the one parameter ϵ (possibly with some sacrifice in overall efficiency that might be obtained by "tuning" ϵ at different places in the algorithm). In general, if ϵ is too large, the roots found will simply not be accurate enough. On the other hand if ϵ is too small the algorithm will take too long to converge (or not converge at all). Thus there is usually a range of values of ϵ for which the algorithm will work well. Within this range the results seem to be quite insensitive to the particular value of ϵ, but the range will depend in general on the class of polynomials being worked on. For example, in the 511-degree example discussed below, the algorithm worked well in the range

$$10^{-8} \leqslant \epsilon \leqslant 10^{-13}$$

on a machine with 64-bit double-precision real numbers, the VAX 11/750.

Besides ϵ we also must specify the maximum number of Newton-Raphson iterations per start before restarting (*ktrym*), the maximum number of such iterations for polishing before the polishing attempt is abandoned (*kpolm*), and the total number of iterations before we quit entirely (*kmax*). We have been successful with the choices

$$kmax = Q \cdot m$$

$$ktrym = kpolm = Q$$

where m is the degree of the original polynomial, and $Q = 20$ for the program that factors polynomials with real coefficients, and $Q = 40$ for the complex polynomial program.

There are several ways of checking the operation of the program. First, we can simply substitute the value of each root back into the original polynomial and verify that this residual value is small. Extreme care must be exercised here in dealing with roots far from the unit circle, however. These roots have necessarily been found near the end of the algorithm, when the polynomial has been deflated to low degree, and substitution in the original polynomial of high degree will very often cause overflow. For this reason, the residual value calculation is skipped when the root lies outside the annulus corresponding to the actual, original, degree of the polynomial. The largest residual value among the roots tested is reported.

Another test of the program operation is to compare the principal value of the phase computed from the root positions with that obtained from the FFT. This test is not included in the programs given here, but has been tried in several cases of degree 127 with better than 10-place agreement.

Yet another test is of course to factor a polynomial with some or all known roots. For example, the polynomial

$$p(z) = z^n - 1$$

has well-conditioned zeros [3, p. 46], and is very easy to factor, the zeros being nicely spread out and on the unit circle. Many similar examples have been tried, including some with almost repeated roots. Such examples can easily provide cases in which our program fails; the cube of the polynomial above with $n = 100$, for example, should stop any root solver in its tracks. We repeat: Our goal is not to factor the most difficult polynomials, but only the average ones, which constitute those we are likely to meet as signals.

IV. Description of the Program

Fig. 2 shows Fortran77 source code for the real-coefficient version of the program. (Note that we compute the phase of a polynomial with ascending, positive powers of z.) The program consists of three parts: 1) A main program that calls SUBROUTINE FACTOR, and then computes the unwrapped phase from the returned root positions; 2) SUBROUTINE FACTOR that calls the actual root-finder DPROOT and then finds the residual values at the roots, the largest and smallest radii, and the smallest distance of a root from the unit circle; and 3) DPROOT, the root-finder itself. As mentioned above, all the arithmetic is carried out in double precision (real or complex). An effort has been made to write the program to be as portable as possible—for example, the code

$$z = d\cos(rkount) + jay * d\sin(rkount)$$

is used instead of using a complex exponential function. The functions DREAL and DCOMPLEX are not used.

Input/output always presents a portability problem. The particular version of the program given here was run under the UNIX operating system, and four files were used: File 0 for screen output during runs, file 1 for teletype input during runs, file 5 for standard input (usually piped from another program), and file 6 for standard output (usually piped to another program). For simplicity, all arrays are assumed complex, a luxury made possible by the fact that storage requirements are modest. Furthermore, the i/o is labeled. Input coefficients and output unwrapped phase are labeled with the character "#," and output radii (for making histograms of radii, for example) are labeled with a "*." The user will no doubt want to use his own i/o conventions.

V. Comments on the Case of Complex Coefficients

We mention here the changes that are necessary to allow the polynomial to have complex-valued coefficients. Obviously, we need to use real arrays for the coefficients in the one case, and complex arrays in the other. In the real case, zeros occur in conjugate pairs, and some code is necessary to distinguish between real and complex roots. This distinction is not necessary in the complex case, so in this sense the complex-coefficient version of the program is simpler. However, the phase function is odd-symmetric in the real-coefficient case, and not in the complex, so that the actual computation of the unwrapped phase from the zero locations is more complicated in the complex case, where the calculation can be broken down into 8 cases, depending on where the zero is with respect to the real axis and the unit circle.

VI. Examples

The first example is physically generated; it was provided to us by Dr. Michael A. Rodriguez [10]. The signal is a 128-point Hamming-windowed sample of an electroencephalogram resulting from a sudden visual stimulus. (For a discussion of delay estimation in visual evoked potentials using unwrapped phase, see [11].) The algorithm used 1503 iterations (11.8 per root), and 38 restarts with $\epsilon = 10^{-13}$. The unwrapped

```
c           factors real polynomial and finds phase function:
c              sigma coeff(i)*z**(i-1), i=1,k4
c              the coefficients are in ascending order, are not normalized
            implicit double precision (a-h, o-z)
c              signal length<=1024
            double complex z(1024),jay
            double precision y(1024),rootr(1023),rooti(1023)
            jay=(0.d0,1.d0)
c              open a file for tty input
            open(1,file='/dev/tty')
            pi=4.d0*datan2(1.d0,1.d0)
            tpi=2.d0*pi
            call input(k4,z)
            write(0,1600)k4
1600        format('executing phaser with k4= ',i4)
            do 1601 ii=1,k4
1601        y(ii)=z(ii)
c              find out if we want to suppress printing
            write(0,1667)
1667        format('type 0/return to suppress printing')
            read(1,1666)kk
1666        format(i1)
            kprint=1
            if(kk.eq.0)kprint=0
303         call factor(y,k4,rootr,rooti,kinsid,kprint)
            write(0,916)kinsid
916         format('there are',i5,'roots inside the unit circle')
c              get npts, number of phase points desired
            write(0,1615)
1615        format('type npts, number of phase points desired')
            read(1,1614)npts
1614        format(i4)
c              arg is phase of coeff(k4)
            partr=y(k4)
            parti=-jay*y(k4)
            arg=datan2(parti,partr)
            do 903 k=1,npts
c              z is point on unit circle
            real1=k-1
            real2=npts-1
            freq=pi*real1/real2
            zr=dcos(freq)
            zi=dsin(freq)
            sum=arg
            k4m=k4-1
            do 901 j=1,k4m
c              w is root
            wi=rooti(j)
            wr=rootr(j)
            r2=wi**2+wr**2
            phi=datan2(zi-wi,zr-wr)
            if(zr.ge.wr)goto901
            if(zi.lt.wi .and. r2.lt.1.d0)phi=phi+tpi
            if(zi.ge.wi .and. r2.gt.1.d0 .and. wi.gt.0.d0)phi=phi-tpi
901         sum=sum+phi
903         z(k)=sum
904         call output(npts,z)
            stop
            end
            subroutine factor(b,k4,rootr,rooti,kinsid,kprint)
c              sets up problem, calls dproot,
c              and checks residual values at roots
            implicit double precision (a-h,o-z)
            double complex z,res,jay
            double precision b(1),rootr(1),rooti(1),coe(1024)
            jay=(0.d0,1.d0)
            pi=4.d0*datan2(1.d0,1.d0)
            do 550 i=1,k4
550         coe(i)=b(i)
            k4m=k4-1
            call dproot(k4m,coe,rootr,rooti,kerr,kprint)
            write(0,600)kerr
600         format(' return from dproot with kerr=',i5)
            if(kerr.gt.0)stop
            kinsid=0
            resmax=0.d0
            rmax=0.d0
            rmin=2.d0**(32)
            dist=2.d0**(32)
c              mark radii with '*'
            write(6,1000)k4m
1000        format('*'/i4)
            amax=2.d0**(32)
            r2=amax**(1.d0/k4)
            do 701 j=1,k4m
            z=rootr(j)+jay*rooti(j)
            r=dsqrt(rootr(j)**2+rooti(j)**2)
            write(6,1001)r
1001        format(d20.10)
c              skip residue calculation if root is too big
            if(r.lt.r2)goto711
            write(0,712)r
712         format('skipping residue calculation at this root, r='d20.10)
            goto 713
711         res=b(k4)
            do 705 k=2,k4
705         res=res*z+b(k4-k+1)
            partr=res
            parti=-jay*res
            resmag=dsqrt(partr**2+parti**2)
            if(resmax.le.resmag)resmax=resmag
            if(kprint.eq.1)write(0,702)r,resmag
713         if(rmax.lt.r)rmax=r
            if(rmin.gt.r)rmin=r
            if(r.lt.1.d0)kinsid=kinsid+1
            distr=dabs(r-1.d0)
            if(dist.gt.distr)dist=distr
701         continue
702         format(' r= ',d20.10,' res= ',d20.10)
            write(0,703)resmax
            write(0,704)rmax,rmin,dist
703         format('resmax= ',d20.10)
704         format('rmax= ',d20.10/'rmin= ',d20.10/'dist=',d20.10)
            return
            end
            subroutine dproot(mm,a,rootr,rooti,kerr,kprint)
c              mm=degree of polynomial
c              a=coefficient array, lowest to highest degree
c              kprint=1 for full printing
c              kerr=0 is normal return
            implicit double precision (a-h,o-z)
            double complex b(1024),c(1024),p,pp,z,w,
            double complex bb(1024),cc(1024),jay
            double precision a(1),rootr(1),rooti(1)
            double precision save(1024)
            jay=(0.d0,1.d0)
            mmp=mm+1
            m=mm
            mp=mmp
            do 700 i=1,mp
700         save(i)=a(i)
c              kount is number of iterations so far
            kount=0
c              kmax is maximum total number of iterations allowed
            kmax =20*m
c              newst is number of re-starts
            newst=0
c              ktrym is number of attempted iterations before re-starting
            ktrym=20
c              kpolm is number of attempted iterations before polishing is stoppe(
            kpolm=20
c              amax is the largest number we allow
            amax=2.d0**(32)
            amin=1.d0/amax
c              rr1 and rr2 are radii within which we work for polishing
            rr1=amin**(1.d0/m)
            rr2=amax**(1.d0/m)
c              eps is a tolerance for convergence
            eps=1.d-13
            sqteps=dsqrt(eps)
c              main loop; m is current degree
10          if(m.le.0)goto200
c              new z, a point on the unit circle
            rkount=kount
            z=dcos(rkount)+jay*dsin(rkount)
            ktry=0
c              r1 and r2 are boundaries of an expanding annulus within which we w
            r1=amin**(1.d0/m)
            r2=amax**(1.d0/m)
c              inside loop
```

Fig. 2. Fortran77 source program for phase unwrapping. The parameter "eps" is fixed at 10^{-13}, but should be adjusted for a particular application.

```
20      partr=z
        parti=-jay*z
        size=dsqrt(partr**2+parti**2)
        if(size.lt.r1 .or. size.gt.r2)goto300
        if(ktry.ge.ktrym)goto300
        ktry=ktry+1
        if(kount.ge.kmax)goto400
        kount=kount+1
c           get value of polynomial at z, synthetic division
        b(mp)=a(mp)
        do 30 j=1,m
        k=m-j+1
30      b(k)=z*b(k+1)+a(k)
        p=b(1)
        partr=p
        parti=-jay*p
        if(dsqrt(partr**2+parti**2).gt.amax)goto300
c           get value of derivative at z, synthetic division
        c(mp)=b(mp)
        mdec=m-1
        do 60 j=1,mdec
        k=m-j+1
60      c(k)=z*c(k+1)+b(k)
        pp=c(2)
        partr=pp
        parti=-jay*pp
        if(dsqrt(partr**2+parti**2).lt.amin)goto300
c           test for convergence
        partr=p
        parti=-jay*p
        size=dsqrt(partr**2+parti**2)
        if(size.gt.eps)goto775
        nroot=mm-m+1
        if(kprint.eq.1)write(0,776)kount,nroot
776     format('kount=',i5,'  root no.=',i5)
        goto500
775     continue
        z=z-p/pp
        goto20
c           end of main loop

c           normal return
200     kerr=0
        goto600
c      new start
300     rkount=kount
        z=dcos(rkount)+jay*dsin(rkount)
        ktry=0
        newst=newst+1
        goto20

c           too many iterations
400     kerr=400
        goto600
c      root z located
c           polish z to get w
500     w=z
        kpol=0
510     partr=w
        parti=-jay*w
        size=dsqrt(partr**2+parti**2)
c           give up polishing if w is outside annulus
        if(size.lt.rr1 .or. size.gt.rr2)goto501
c           give up polishing if kpol>=kpolm
        if(kpol.ge.kpolm)goto501
        kpol=kpol+1
        if(kount.ge.kmax)goto400
        kount=kount+1
        bb(mmp)=save(mmp)
        do 530 j=1,mm
        k=mm-j+1
530     bb(k)=w*bb(k+1)+save(k)
        p=bb(1)
        partr=p
        parti=-jay*p
        if(dsqrt(partr**2+parti**2).gt.amax)goto300
        cc(mmp)=bb(mmp)
        mdec=mm-1
        do 560 j=1,mdec
        k=mm-j+1
560     cc(k)=w*oc(k+1)+bb(k)
        pp=cc(2)
        partr=pp
        parti=-jay*pp
        if(dsqrt(partr**2+parti**2).lt.amin)goto300
        partr=p
        parti=-jay*p
        size=dsqrt(partr**2+parti**2)
c           test for convergence of polishing
        if(size.le.eps)goto501
        w=w-p/pp
        goto510

c           deflate
501     b(mp)=a(mp)
        do 830 j=1,m
        k=m-j+1
830     b(k)=z*b(k+1)+a(k)
        p=b(1)
        rootr(m)=w
        rooti(m)=-jay*w
        m=m-1
        mp=mp-1
        parti=-jay*w
        if(dabs(parti).gt.sqteps)goto140
c           real root
        rooti(m+1)=0.d0
        do 100 j=1,mp
100     a(j)=b(j+1)
        goto10
c      complex root
140     partr=z
        parti=-jay*z
        z=partr-jay*parti
        c(mp)=b(mp+1)
        do 110 j=1,m
        k=m-j+1
110     c(k)=z*c(k+1)+b(k+1)
        rootr(m)=w
        rooti(m)=-(-jay*w)
        m=m-1
        mp=mp-1
        do 130 j=1,mp
130     a(j)=c(j+1)
        goto10
c           report and return
600     real1=kount
        real2=mm
        temp=real1/real2
        write(0,150)kount,temp
150     format(' kount=',i10,' kount/root=',f15.5)
        write(0,151)newst,kerr
151     format(' new starts=',i10,' kerr=',i10)
        return
        end


        subroutine input(nc,y)
        double complex y(1)
        character z,z1
c          #     z1        number of complex points, nc and y
        data z1/'#'/
c          default options
        nc=1
        y(1)=(0.d0,0.d0)
10      read(5,1000,end=50)z
1000    format(a1)
        if(z.eq.z1)goto1
        goto10
1       read(5,1001)nc
        read(5,1003)(y(ii),ii=1,nc)
        goto10
1001    format(i4)
1003    format(2d20.10)
50      return
        end


        subroutine output (nc,y)
        double complex y(1)
        write(6,1007)nc,(y(ii),ii=1,nc)
1007    format('#'/i4/(2d20.10))
        return
        end
```

Fig. 2. (*Continued.*)

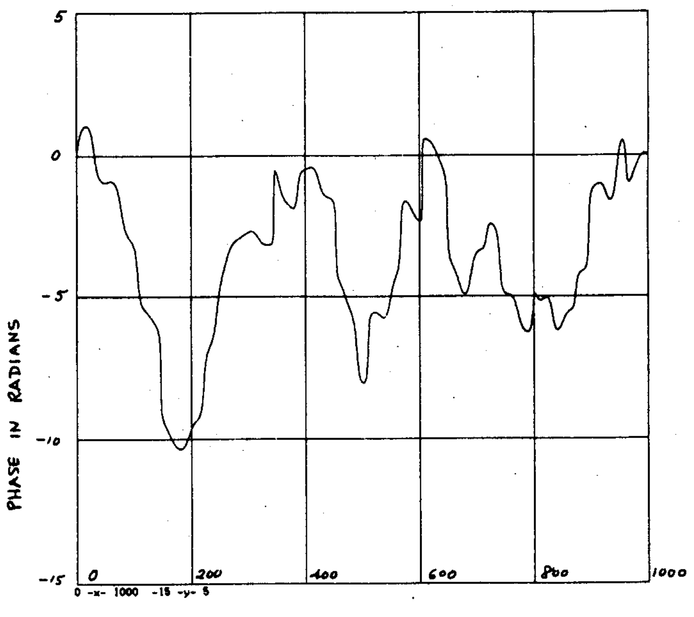

Fig. 3. Unwrapped phase for example 1, a 128-point electroencephalogram signal. The linear component has been removed.

phase function with the linear component removed is shown in Fig. 3. The apparent jumps are not malfunctions of the program; they are in fact caused by zeros near the unit circle, and can be correlated with the angular position of those roots. The smallest distance of a root from the unit circle is 0.98*e* -3.

We next generated a 512-point signal using the random number generator UNI [12] with seeds 12345 and 23456. The signal values were taken to be 2.0* [*ran* -0.5], and were therefore uniformly distributed between -1.0 and +1.0. The 511th order polynomial was factored on the VAX 11/750 in about 13 minutes, using 5737 iterations (11.2 per root) with $\epsilon = 10^{-13}$. (A problem of the same size took about 13 seconds on the IBM 3033, FORTH compiler.) There were 177 restarts, and the smallest distance of a root from the unit circle was 0.54*e* -5. A plot showing the position of the roots is shown in Fig. 4; the clustering at radius 1 is quite pronounced.

Finally, we factored the polynomial formed from the DFT of the $N = 128$-point complex-valued signal

$$f(k) = (1 + \cos(1.4 * k * 2\pi/N))\, e^{\sqrt{-1} * 2.5 * k * 2\pi/N},$$
$$\text{for } k = 0, \cdots, 127$$

using a version of the program modified for complex coefficients. If $F(i)$ is the DFT of $f(k)$, $f(k)$ is a polynomial whose coefficients are $F(i)/N$, and the phase of the DFT is in fact the phase of the complex signal $f(k)$ itself, as a function of the *time* variable k.

This signal is a complex exponential modulated by an envelope that actually becomes zero at some point—we might say it is a "fading" signal. (See [7], [8] for an example where such a phenomenon arises in ocean acoustics.) Fig. 5 shows the calculated unwrapped phase at the 128 time points (connected by straight lines); except for two features it is *precisely* linear. When the envelope approaches zero the complex-valued signal

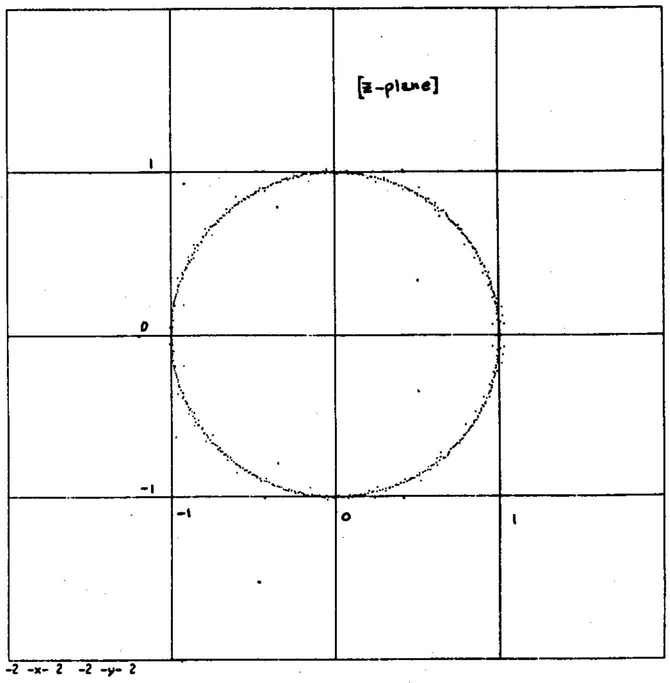

Fig. 4. The 511 roots for example 2, a 512-point signal.

approaches the origin in the complex plane, and the unwrapped phase suddenly changes rapidly in the vicinity of this point, as can be seen from the figure. What actually happens in this region is that the unwrapped phase increases by an extra 2π at several successive sample points, and this is caused by the interpolated continuous phase curve oscillating in the region near the origin and encircling the origin between sample points. The same phenomenon occurs near the end of the time interval, at which point the phase must be made to meet the phase at the

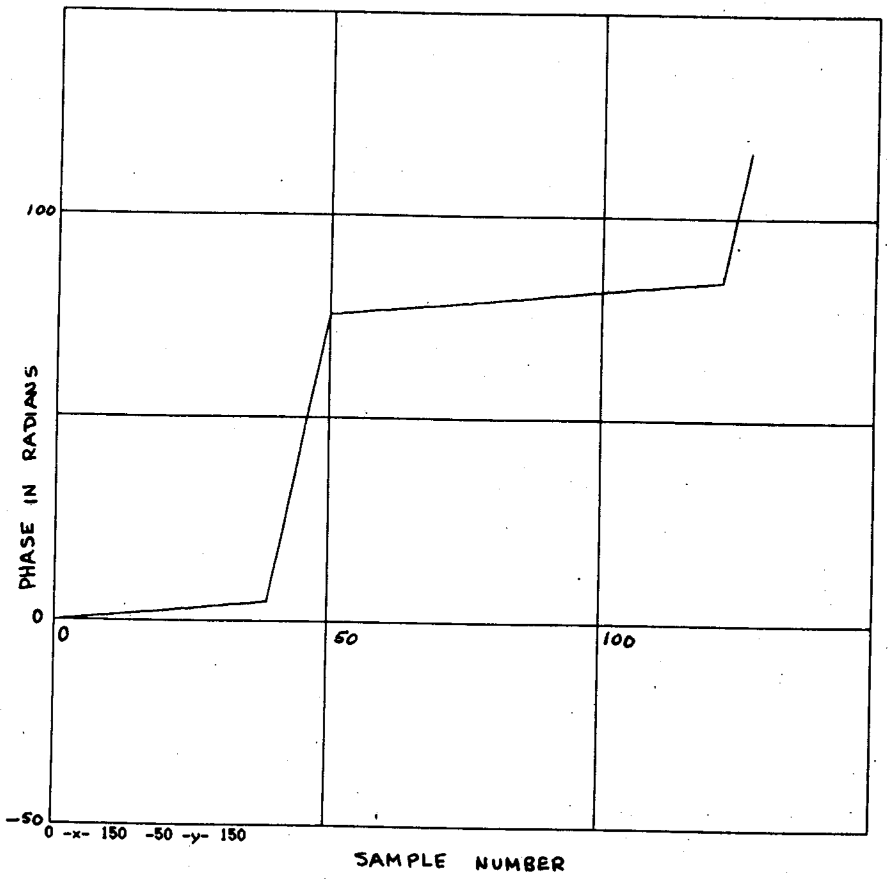

Fig. 5. Unwrapped phase for example 3, a complex signal modulated by a fading envelope.

beginning of the time interval in principal value. When the phase function is evaluated between the discrete time points, a continuous phase function results which ripples slightly between the points shown. The next section will take up the question of precisely what continuous function results from our phase calculation.

VII. Phase Unwrapping as Interpolation

In the definition of the phase unwrapping problem [6], a real signal $\{x_k, 0 < k < \infty\}$ having a z-transform $X(z)$ with no poles or zeros on the unit circle in the z-plane, is given. The function $\Theta(\omega) = arg\ (X(e^{j\omega}))$ may then be uniquely defined in such a way that $\Theta(\omega)$ is continuous and odd on $[-\pi, \pi]$. By *phase unwrapping* we mean the determination of $\Theta(\omega)$ from $\{x_k\}$.

For signals which are finite in extent, so that X is a polynominal in z^{-1}, the principal value of the phase function $\Theta(\omega)$ at any sufficiently fine uniform grid of points on $[-\pi,\pi]$ may be obtained from the DFT. However, the determination of the correct multiple of 2π to be added to the principal value in order to give the actual values of the phase of the signal at these points is of course not trivial.

By assuming that the z-transform of the signal is a polynomial, we are assuming that the phase of the signal at any point $z_0 = e^{j\omega_0}$ may be obtained from the zeros of the polynomial according to well-defined rules. In essence, we are assuming that the phase function is an *interpolant* of the known principal value function from a restricted class of functions: Namely the phase functions of finite signals.

It is clear that Tribolet's adaptive integration technique for phase unwrapping generates interpolants from this class. However, because it is not possible to bound the phase derivative without restricting the class even further, it is not possible to set an upper bound on the number of grid points on $[-\pi,\pi]$ which may be required to get correct phase functions. But we can say that for sufficiently fine grid size, the result of Tribolet's method will coincide with that obtained by factorization (ignoring numerical error).

VIII. Asymptotic Theory of Zero Distribution

By introducing a suitable mathematical model, we develop a theoretical justification of the empirically observed distribution of zeros of high order polynomials associated with finite signals. To avoid technicalities arising from existence of real roots of polynomials with real coefficients, we assume complex-valued signals and thus polynomials with complex coefficients.

As a tentative signal model, we may propose a sequence of identically distributed, independent random variables. With mild conditions on the underlying distribution function, the asymptotic distribution of the roots of the polynomial

$$p(z) = \sum_{l=0}^{n} a_l z^l$$

may be characterized as follows [9]. Let $N_n\ (\vartheta_1, \vartheta_2, p)$ be the number of zeros of $p(z)$ in the region

$$\{z | 0 < \vartheta_1 < \vartheta_2 < 2\pi, 1 - n^{-r} < |z| < 1 + n^{-r}, 0 < r < 1\}.$$

Then, with probability one

$$\lim_{n \to \infty} N_n(\vartheta_1, \vartheta_2, p)/n = (\vartheta_2 - \vartheta_1)/2\pi.$$

This is a very strong result: It indicates that the roots of a "random polynomial" tend to be evenly distributed in angle and tightly clustered near the unit circle as the degree of the polynomial increases. (Recall Fig. 4.) However, the independence condition corresponds to assuming a white-noise signal, which is unrealistic.

A more appealing signal model is the output signal of a fixed white-noise-excited FIR filter, which introduces the appropriate correlation or spectral structure. Since the filtering adds only a fixed (asymptotically negligible) number of zeros to the output's z-transform, we may still use the white-noise result on distribution of zeros to justify the empirically observed distribution illustrated by the examples presented earlier.

The theoretical result also suggests that the general approach to phase unwrapping taken by Tribolet will require increasingly fine grid sizes, and hence increasing FFT lengths, as the number of signal points grows, simply because the zeros of the polynomial are squeezed closer to the unit circle. Indeed, storage requirements seem to be a limiting consideration in phase unwrapping by adaptive integration using small computers.

IX. Conclusions

We have described a method for computing unwrapped phase that seems to be reliable and accurate. It is, however, fairly expensive on problems of realistic size. Experience has shown that the number of iterations per root is almost constant, independent of the signal length n, and each iteration takes time roughly proportional to n, so that we get a time complexity approximately $O(n^2)$. Thus, the method is not well-suited to on-line computation, or high-volume computation, at least with today's technology. On the other hand, the reliability and accuracy of the method may make it preferable to Tribolet's algorithm in certain applications.

We have also explained why the FFT approach necessarily runs into numerical difficulties: The zeros of a z-transform tend to cluster near the unit circle as the signal length becomes large. The phase unwrapping task seems to be inherently difficult for this reason, and it is unlikely that a really fast solution to the problem exists (comparable in speed, say, to the FFT).

Once a signal's transform has been factored, it may be possible to perform other operations besides phase unwrapping. For example, the minimum-phase version of the signal can be obtained directly by reflecting all zeros with magnitude larger than unity inside the unit circle. These ideas will be investigated in the future.

Finally, we ought to mention that the phase unwrapping problem for two-dimensional signals does not seem amenable to this approach, since the zeros of two-variable polynomials do not necessarily occur at isolated points.

Acknowledgment

The program used in [2] was written by W. H. Surber, and our program incorporates many of his ideas. The authors also thank M. A. Rodriguez, J. E. Brolley, L. Takács, R. Porter, and F. Baixas for helpful correspondence and interesting discussions about this problem.

References

[1] J. M. Tribolet, "A new phase unwrapping algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-26, pp. 170–177, 1977.

[2] K. Steiglitz and B. Dickinson, "Computation of the complex cepstrum by factorization of the Z-transform," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 1977, pp. 723–726.

[3] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*. Englewood Cliffs, NJ: Prentice-Hall, 1963.

[4] C. E. Schmidt and L. R. Rabiner, "A study of techniques for finding the zeros of linear phase FIR digital filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-25 pp. 96–98, 1977.

[5] J. F. Kaiser, "Some practical considerations in the realization of linear digital filters," in *Proc. 3rd Annu. Allerton Conf. Circuit and Syst. Theory*, 1965, pp. 621–633. Reprinted in *Digital Signal Processing* (L. R. Rabiner and C. M. Rader, Ed.), New York: IEEE Press, 1972.

[6] A. V. Oppenheim and R. Schafer, *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[7] R. P. Porter, "Acoustic probing of space-time scales in the ocean," in *Topics in Current Physics, vol. 8: Ocean Acoustics* (J. A. De Santo, Ed.), Berlin: Springer-Verlag, 1979, pp. 243–278.

[8] R. P. Porter and R. C. Spindel, "Low-frequency acoustic fluctuations and internal gravity waves in the ocean," *J. Acoust. Soc. Amer.*, vol. 61, no. 4, pp. 943–958, 1977.

[9] L. Arnold, "Über die Nullstellenverteilung zufälliger Polynome," *Math. Zeitschr.*, vol. 92, pp. 12–18, 1966.

[10] M. A. Rodriguez, personal communication.

[11] M. A. Rodriguez, R. H. Williams, and T. J. Carlow, "Signal delay waveform estimation using unwrapped phase averaging," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-29, pp. 508–513, 1981.

[12] *Programs for Digital Signal Processing*. Digital Signal Processing Committee of the IEEE Acoustics, Speech, Signal Processing Soc. (Eds.), New York: IEEE Press, 1979.

[13] M. A. Jenkins and J. F. Traub, "Principles for testing polynomial zerofinding programs," *ACM Trans. Math. Software*, vol. 1, no. 1, pp. 26–34, 1975.

[14] J. A. Grant and G. D. Hitchins, "Two algorithms for the solution of polynomial equations to limiting machine precision," *Computer J.*, vol. 18, no. 3, pp. 258–264, 1975.

[15] R. McGowan and R. Kuc, "A direct relation between a signal time series and its unwrapped phase: Theory, example, and program," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-30, pp. 719–726, 1982.

Kenneth Steiglitz (S'57–M'64–SM'79–F'81), for a photograph and biography, see p. 31 of the February 1982 issue of this Transactions.

Bradley Dickinson (S'70–M'74), for a photograph and biography, see p. 31 of the February 1982 issue of this Transactions.