THE DESIGN OF DIGITAL FILTERS USING INTERACTIVE OPTIMIZATION

by

T. P. Lee, W. T. Nye and A. L. Tits

Memorandum No. UCB/ERL M81/69

15 September 1981

THE DESIGN OF DIGITAL FILTERS USING INTERACTIVE OPTIMIZATION

by

T. P. Lee, W. T. Nye and A. L. Tits

Memorandum No. UCB/ERL M81/69

15 September 1981

ELECTRONICS RESERACH LABORATORY

College of Engineering
University of California, Berkeley
94720

THE DESIGN OF DIGITAL FILTERS USING INTERACTIVE OPTIMIZATION[+]

by

T.P. Lee*, W.T. Nye* and A.L. Tits**

*Department of Electrical Engineering and Computer Science
and the Electronics Research Laboratory
University of California, Berkeley, CA 94720

**Electrical Engineering Department
University of Maryland, College Park, MD 20742

[+]This research was supported by National Science Foundation grants No. ECS-79-13148 and PFR-79-08261 (RANN) and the Air Force Office of Scientific Research (AFSC) United States Air Force under Contract No. F44620-76-C-0100, and by a grant from the Semiconductor Products Division of the Harris Corporation.

ABSTRACT

It is shown how modern optimization techniques can be used to design digital filters. The high flexibility of this approach makes tractable a large class of specifications, such as constraints on both magnitude and phase response or special stability requirement (running a better transient behavior). Efficient use of optimization requires a highly interactive computing environment, including application oriented graphics. As an example, the design of a low pass filter, performed on the Berkeley DELIGHT system, is discussed.

1. INTRODUCTION

The design of digital filters consists of approximation of desired magnitude and phase response specifications by a ratio of polynomials corresponding to a stable, causal transfer function.

Traditionally, designers have used classical techniques to approximate their desired response. As the knowledge of analog filter design is quite advanced, these techniques normally consist of the design of an analog filter transfer function and its conversion to a digital transfer function. The mapping from analog to discrete frequency domain can be accomplished by several commonly used methods such as the impulse invariant transformation or the bilinear transformation. These design techniques have been incorporated in the classical filter design package FILSYN [1].

Classical techniques for the design of infinite impulse response (IIR) filters are rigid in the design specifications allowed. The analog filters available for use by the designer each have definite characteristics. For example, the magnitude response of a butterworth filter is monotonic in both passband and stopband, Chebyshev filters exhibit an equiripple passband and monotonic stopband response, and elliptic filters are equiripple in both passband and stopband. Each of these filters is characterized by nonlinear phase response. Bessel filters are characterized by the property that the

group delay is maximally flat at the origin of the s-plane. This property is generally not preserved after digitization. The magnitude response of the Bessel filter tends toward Gaussian as the filter order is increased.

A much greater flexibility can be obtained if optimization techniques are utilized. Some attempts have been made in that direction in the past. Deczky [1a], for instance, designs recursive filters based on the minimum p-error criterion, using the Fletcher-Powell algorithm; he takes into account requirements on both amplitude and phase responses. In the approach described in this paper, we use a more general problem formulation, tractable by more recent, semi-infinite optimization algorithms. In addition to amplitude and phase response, any other "reasonable" requirement may be specified, such as special stability requirements or constraints on the size of some transfer function coefficients. Also, the degrees of numerator and denominator of the transfer function can be choosen at will; in particular, finite impulse response (FIR) filters are allowed.

More importantly, however, the aim of this paper is to show how filter design can be viewed as one of the numerous applications of a self-contained optimization and design system, such as DELIGHT [2]. In DELIGHT, any algorithm from an integrated library can be utilized without the need of any modification in the problem formulation provided by the application designer (an algorithm due to Gonzaza, Polak and Trahan [3] is chosen as an example); this total independence between problem

and algorithm is of great help. Other precious features of DELIGHT include the possibility to display graphical output in a desirable form and the outstanding interaction capabilities: one can interrupt, observe, diagnose, modify and restart a computation as it progresses, resulting in saving in both computer time and time needed to complete a design. All algorithms and problem formulations are coded in the interactive structured programming language RATTLE [4].

The balance of this paper is organized in the following manner. In section 2, after briefly discussing the optimization algorithm [3], we take the example of a lowpass IIR filter to show how a filter design problem can be formulated into a nonlinear program. Section 3 details the computational results obtained with the example described in section 2. Section 4 discusses the advantages of using an interactive system to perform the optimization. Finally the conclusions are set forth in section 5.

2. DESIGN USING DELIGHT

2.1 THE OPTIMIZATION ALGORITHM

The optimization algorithm employed in the filter design is of the combined phase 1-phase 2 feasible directions type. It is useful for solving optimization problems with functional inequality constraints. A brief description is given here. For a more detailed treatment, the reader is referred to [3]. // The algorithm solves problems of the form:

$$\min_x \{f^0(x) \mid f^j(x) \le 0, j=1,2,\ldots,p;\ \max_{\omega\in\Omega} \phi^j(x,\omega) \le 0, j=1,2,\ldots,m\} \quad (2.1)$$

where

$f^0: R^n \to R$ is the cost function

$f^j: R^n \to R$ $j=1,2,\ldots,p$ are inequality constraints

$\phi^j: R^n \times R \to R$ $j=1,2,\ldots,m$ are functional inequality constraints

$x \in R^n$ is the design vector

Ω is a compact interval of the real line

and where it is assumed that

(i) f^0, f^j $j=1,2,\ldots,p$ are continuously differentiable in x.

(ii) ϕ^j $j=1,2,\ldots,m$ are continuously differentiable in x and piecewise continuous in ω.

The algorithm also uses the function $\psi^+(x)$ defined as

$$\psi^+(x) = \max\{0, \psi(x)\}$$

with

$$\psi(x) \triangleq \max\{f^j(x), j=1,2,\ldots,p;\ \max_{\omega\in\Omega} \phi^j(x,\omega), j=1,2,\ldots,m\}.$$

The feasible region is defined as $\{x \mid \psi^+(x)=0\}$. A feasible

point is any point in the feasible region. Hence, roughly speaking, $\psi^{\dagger}(x)$ is a measure of the distance from point x to the feasible region.

To use the algorithm one need not start from a feasible point. The phase 1 portion of the algorithm, while trying to possibly decrease the cost function, computes a descent direction for the most violated constraints (including the most critical values of w in the functional constraints, $\phi^{j}(x,\omega), j=1,2,\ldots,m$), eventually forcing the iterate into the feasible region and hence driving $\psi^{\dagger}(x)$ to zero. Once this process is completed and a feasible point is found, the phase 2 portion of the algorithm computes a direction leading to a decrease in the cost function while maintaining feasibility.

The algorithm, as implemented, runs as a series of iterations. During each iteration a search direction is computed. Then a step is taken in this direction such that, in phase 1, $\psi^{\dagger}(x)$ is decreased and, in phase 2, the cost function $f^{\circ}(x)$ is decreased while maintaining feasibility ($\psi^{\dagger}(x)=0$). When running the algorithm in DELIGHT, the user has the option of specifying the number of steps (iterations) to be taken before the computation is automatically interrupted or to allow the computation to proceed continuously and explicitly interrupt it when

interaction seems desirable (key information about the algorithm is displayed at each iteration). After graphically displaying the current design and perhaps modifying constraints, the user may resume computation until he is satisfied with the filter design performance.

2.2 FORMULATION OF THE FILTER DESIGN PROBLEM AS A MATHEMATICAL PROGRAM

In this section, we show how digital filter specifications can be expressed in the form of equation (2.1).

Consider the design of a digital lowpass IIR filter having a passband cutoff frequency of ω_p radians, a stopband cutoff frequency of ω_a radians and linear (within a sector) phase. Our mathematical programming formulation of the specifications includes the following functional constraints:

(i) amplitude constraints

$$1-\varepsilon_p d \le \text{amplitude}(\omega) \le 1+\varepsilon_p d \qquad \forall \omega \in [0,\omega_p] \qquad (2.2)$$

$$0 \le \text{amplitude}(\omega) \le 1+\varepsilon_p d \qquad \forall \omega \in (\omega_p,\omega_a) \qquad (2.3)$$

$$0 \le \text{amplitude}(\omega) \le \varepsilon_a d \qquad \forall \omega \in [\omega_a,\pi] \qquad (2.4)$$

i.e., the magnitude is constrained to lie inside the region shown on fig. 1.

(ii) phase constraints

$$(m-\varepsilon_\phi d)\omega-\delta \leq \text{phase}(\omega) \leq (m+\varepsilon_\phi d)\omega+\delta \qquad \forall \omega\in[0,\omega_p] \quad (2.5)$$

i.e., the phase must lie inside the region shown on fig. 2.

Equations (2.2)-(2.5) can easily be expressed as functional inequality constraints in the form required in (2.1), with $\Omega=[0,\pi]$. For example, equation (2.5) can be written as

$$\phi^3(x,\omega)\leq 0 \qquad \forall \omega\in\Omega$$
$$\phi^4(x,\omega)\leq 0 \qquad \forall \omega\in\Omega$$

where

$$\phi^3(x,\omega)\triangleq -\text{phase}(\omega)+(m-\varepsilon_\phi d)\omega-\delta \qquad \text{for } \omega\in[0,\omega_p] \quad (2.6)$$
$$\triangleq -\infty \qquad \text{otherwise}$$

$$\phi^4(x,\omega)\triangleq \text{phase}(\omega)-(m+\varepsilon_\phi d)\omega-\delta \qquad \text{for } \omega\in[0,\omega_p] \quad (2.7)$$
$$\triangleq -\infty \qquad \text{otherwise}$$

Similarly, equations (2.2)-(2.4) can be expressed in the form $\phi^j(x,\omega)\leq 0$ $j=1,2$. ε_p, ε_a and ε_ϕ are constants selected by the user. m, the slope of the desired phase response is

unspecified. d, a measure of the allowable deviation in both magnitude and phase from ideal responses, is chosen as the cost function to be minimized. Finally, $\delta > 0$ ensures that the optimization algorithm performs properly, as explained in section 4.

The transfer function is expressed as a ratio of quadratics. This formulation, besides leading to a filter with low quantization noise [10], facilitates the use of the bilinear transformation and simplifies the expressions for conditions of stability. These conditions can be written, for each denominator quadratic, as (see [5])

$$1+a+b>0 \qquad (2.8)$$

$$1-a+b>0 \qquad (2.9)$$

$$1-b>0 \qquad (2.10)$$

where a and b are coefficients of the quadratic z^2+az+b. Stronger stability properties can be obtained if the poles of the digital filter transfer function are constrained to lie within a circle of radius $\rho \in (0,1)$. This is achieved by replacing (2.8)-(2.10) with

$$1+a/\rho+b/\rho^2\geq 0 \qquad (2.11)$$

$$1-a/\rho+b/\rho^2\geq 0 \qquad (2.12)$$

$$1-b/\rho^2\geq 0 \qquad (2.13)$$

Equations (2.8)-(2.13) correspond to ordinary inequality constraints f^i in formulation (2.1). For example, (2.11) can be written as

$$f^1(x)\triangleq -1-a/\rho-b/\rho^2\leq 0 \qquad (2.14)$$

The design parameter vector x has as components

(i) the constant factor multiplying the transfer function

(ii) the coefficients of the numerator and denominator quadratics

(iii) m, the unspecified slope of the desired phase response

(iv) d, the measure of allowable deviation of magnitude and phase from the ideal response, which is desired to be as small as possible.

Thus, the cost function to be minimized is $f^0(x)=d$. Our problem, which originally was a "min-max" problem (minimize the maximum deviation from the ideal response), has thus been transformed into a semi-infinite minimization problem through the introduction of the extra variable d.

2.3 THE RATTLE FORMULATION OF THE DESIGN SPECIFICATIONS

The RATTLE formulation of a filter design specification consists of a set of six files which include default parameter values and initial data (which can be modified interactively), functions and gradient (with respect to x) evaluation procedures as well as application oriented graphical output and miscellaneous procedures. A listing of these files is given in Appendix A. It should be stressed that, if gradient evaluation procedures had not been provided, DELIGHT would, by default, estimate them by finite differences (this feature is useful when a complex formula is used).

The first file, filterS (setup), defines filter parameters. The design vector x includes the transfer function multiplicative factor, coefficients of the numerator and denominator quadratics, the slope and the allowable deviation. For readability, the index of x corresponding to each entry has been given a name which is referred to when accessing an element of a related array. For example, if 'grad' is an array containing the gradient of some function with respect to the design vector, grad(Imult) will be the derivative with respect to the multiplicative factor and grad(Ia(2)), the derivative with respect to the

coefficient of the linear term of the second numerator quadratic.

Also, for easy interaction or input of data, mnemonic names have been equivalenced to each entry of x using the DELIGHT define mechanism [2]. For example,

Mult≡x(1)

A(i)≡x(2i)

Thus, the user may type "A(3)=2.5" rather than "X(6)=2.5".

File filterC (cost) contains two procedures. The first, cost, returns the value of the cost function. The other, gradcost, returns the gradient of the cost function. Since the cost is the last entry of x and no other entry depends on the cost, we have

$$\nabla \text{cost}(x)=(0,\ldots,0,1)' \qquad (2.15)$$

File filterI (inequality constraints) plays a similar role for the inequality constraints (eqns (2.8)-2.10) or (2.11)-(2.13)).

File filterF (functional inequalities) contains procedures computing the amplitude and phase responses of the digital filter, the functional inequalities and their gradients. The amplitude is computed in the following manner. For each w

$$\text{amplitude}(w)=A\left|\frac{\prod_{i=1}^{n}(z^2+a_i z+b_i)}{\prod_{i=1}^{m}(z^2+c_i z+d_i)}\right|_{z=e^{j\omega}} \qquad (2.16)$$

where A is the multiplicative factor and a_i, b_i, c_i, d_i are coefficients of the numerator and denominator quadratics. Similarly,

$$\text{phase}(w)=\angle\left.\frac{\prod_{i=1}^{n}(z^2+a_i z+b_i)}{\prod_{i=1}^{m}(z^2+c_i z+d_i)}\right|_{z=e^{j\omega}} \qquad (2.17)$$

Procedures fineq and gradfineq call amplitude(w) and phase(w) to compute various functional inequality constraints and their gradients (see eqns (2.2)-(2.7)).

File filterM (miscellaneous) contains procedures necessary to display the output in graphical form on the terminal screen, a procedure which performs the bilinear transformation on an analog transfer function given as a ratio of quadratics and a procedure to test the ability of FILSYN to design filters using some modified stability criteria, which is discussed in section 3 of this report.

An example of the graphical output is given in figure 1. The magnitude response is plotted over the interval $w\in[0,\pi]$. The phase response is plotted for $w\in[0,w_p]$. Also, the positions of the poles of the discrete transfer function are displayed. The dashed lines in the magnitude and phase plots show the functional inequality constraint boundaries. These boundaries can be modified to reflect

a desired response or to increase the speed of computation by varying combinations of ε_p, ε_a, ε_ϕ the cost function and the slope. As an example, in the figure, the lower constraint on the phase is violated. If left infeasible, optimization will eventually force the phase inside the constraint boundary. On the other hand, by increasing ε_ϕ one could make the phase feasible without disturbing the constraint boundaries on the magnitude response.

File filterD (data) provides the initial values of the design vector, x.

3. COMPUTATIONAL EXPERIMENTS

As previously noted, the optimization algorithm is sensitive to the initial choice of filter design parameters. Starting from randomly selected points would either increase the computing time necessary to converge to the solution or the algorithm would not converge at all to a global minimum, instead converging to a local, suboptimal solution. To avoid these pitfalls, the classical filter design program, FILSYN, was used to generate an analog design which was subsequently transformed, using the bilinear transformation, to a digital filter and used as the starting point for the optimization program. In all examples, the bilinear transformation was chosen to avoid the aliasing problems encountered with the use of impulse invariance. By prewarping the frequency scale, the desired initial frequency response could be preserved [6].

In the first example (fig. 4) a fourth order Chebyshev filter having a bandedge loss of 1db and a stopband loss of 17db was designed using FILSYN and used to initialize the optimization program. Optimization resulted in another fourth order chebyshev filter having a bandedge loss of .43db and a stopband loss of 16.7db. FILSYN required a fifth order filter to satisfy these criteria.

In example 2 (fig. 5) a fourth Butterworth filter having a 4db bandedge loss and a 13db stopband loss was designed

using FILSYN. At the price of slightly less linear phase and the appearance of ripple in the passband, a fourth order filter having a significantly narrower transition band resulted from optimization. // Elliptic filters are optimum in the sense that for a given order, no other filter has a narrower transition band. They are also characterized by a magnitude response which is equiripple in both the passband and the stopband [6], which best matches our specifications. Example 3 (fig. 6) illustrates an attempt to optimize the magnitude of a FILSYN designed elliptic filter having a .3db bandedge loss and a 23db stopband loss. The improvement observed was insignificant. A comparison demonstrated that FILSYN would design a fourth order filter meeting equivalent specifications. To demonstrate the versatility of the optimization system, an attempt to linearize the phase of the FILSYN designed elliptic filter was made. As displayed in figures 7 and 8, good success was made at linearizing the phase, but at the cost of a "hump" near the passband edge frequency in the magnitude response. These results demonstrate the ability of the optimization program to construct linear phase filters from an arbitrary initial design.

Pole placement within the unit circle is tied to the damping of transient response and hence improved stability. The smaller the circle containing the filter transfer

function poles, the faster frequency response transients settle out [7]. Hence it would be useful to be able to design filters using pole placement (i.e. magnitude from the origin) as a constraint. One might postulate that this could be done using FILSYN and applying the simple transformation $z \rightarrow \frac{z}{\rho}$, with $\rho \in (0,1)$, to the obtained transfer function. This hypothesis was tested, using a FILSYN designed elliptic filter, with the result displayed in figure 9 for $\rho = .8$. The shape of the transformed filter frequency response is affected and may, as in figure 9, result in a poor design. Hence this method is not useful. The optimization system allows the user to specify the radius of a circle within which the poles must lie after optimization. Figure 10 illustrates an example in which the modified stability criterion is used. Beginning from a FILSYN designed elliptic filter, the poles were constrained to lie within a circle of radius .8. The ability of the optimization system to modify both magnitude and phase as well as push the poles within the specified radius results in a greatly superior filter to that designed by FILSYN using the previously described technique.

4. ADVANTAGES OF INTERACTION

The advantages in using an interactive system are many. Without interactive capability, the amount of time necessary to solve the problem using batch mode would render the problem unmanageable.

The graphics capability is indispensable. It allows one to visually monitor the progress of the optimization, set constraints initially and vary them during the optimization process saving considerable computing time, and make visual comparisons between designs.

In the DELIGHT system, when modifying or adding to the program, the interactive system allows one to test single statements or subprograms without having to recompile the entire program resulting in a significant time savings. This is due to the one-pass nature of RATTLE compilation; there is no load/linkage phase. Program debugging is also simplified through the use of interaction. Program execution can be interrupted, procedures can be entered, values of locally and globally defined variables can be printed, diagnostic checks can be added and execution can be restarted once more in an effort to pinpoint errors. This is a vast improvement over non-interactive methods in which one must first wait for a run to be completed, add diagnostics, recompile,

relink and rerun the entire program, repeating this sequence perhaps several times to debug it. A specific instance in which interaction proved to be quite valuable in program debugging occurred when the functional inequality constraints were first tested. The phase constraints were originally specified as

$$(m-\varepsilon_\phi d)\omega \leq \text{phase}(\omega) \leq (m+\varepsilon_\phi d)\omega \quad \forall \omega \in [0,\omega_p] \qquad (4.1)$$

This resulted in a constraint region shown on fig. 11.

The algorithm was unable to solve the quadratic program used in finding a search direction. On the suspicion that the quadratic programming routine was being passed invalid gradient arguments, the interactive capabilities of DELIGHT were used and the procedure computing gradients of the functional inequality constraints was entered. The matrix of the gradients was checked and those entries corresponding to $\nabla\phi^3(x,0)$ and $\nabla\phi^4(x,0)$ were found to be zero. This prevents from obtaining a descent direction for these two constraints. Inspection showed that $\phi^3(x,0) \equiv \phi^4(x,0) \equiv 0$ for all x. Hence, the suspicion was confirmed. Subsequently, equation

(4.1) was corrected to its present form (eqn. (2.5)).

Another feature of DELIGHT, useful for our purpose, is that the entire progress, iteration and formulation, of the design can be saved so that the design may be resumed at a later time from exactly the same point.

5. CONCLUSIONS

We have shown how interactive optimization can be applied to the problem of digital filter design. A method has been described to formulate a digital filter design problem as a semi-infinite nonlinear program. We have stressed the importance of a highly interactive computing environment for efficiently solving such a nonlinear program. Examples have been presented demonstrating the substantially increased flexibility of the optimization system over classical design methods.

The price paid for high flexibility is a fairly large computing time (the examples presented typically require 30 minutes of CPU time on a VAX 11/780 running VM UNIX). However, the computing time necessary to complete the design could be significantly decreased if portions of the code where interaction is not essential, such as computation of amplitude and phase responses and of their gradients, were written in FORTRAN rather than RATTLE. Also, the algorithm is somewhat slow to converge. More recent algorithms (i.e.,[8]) should converge much more rapidly. Perhaps most importantly, the computation time is often not an overriding factor. A situation in which the same design is used to produce a large number of filters provides an example.

Although we have described in detail only the specific

case of an IIR lowpass filter, extension to other types of frequency response (highpass, bandpass, bandstop) is straightforward. Also, degrees of numerator and denominator can be changed at will; FIR filters are obtained as a particular case.

ACKNOWLEDGEMENT

The authors thank P. Dodd and Y. Wardi for their invaluable help and Professors E.I. Jury and J.C. Walrand and Dr. B. Gopinath for fruitful discussions.

REFERENCES

1. G. Szentirmai, "FILSYN- A General Purpose Filter Synthesis Program", Proc. IEEE, Vol.65, pp. 1443-1458, Oct. 1977

*1a.

2. W. Nye, E. Polak, A. Sangiovanni-Vincentelli and A. Tits, "DELIGHT: An Optimization-Based Computer-Aided Design System", presented at the IEEE International Symposium on Circuits and Systems (ISCAS), Chicago, Illinois 1981

3. C. Gonzaga, E. Polak and R. Trahan, "An Improved Algorithm for Optimization Problems With Functional Inequality Constraints", IEEE Transactions on Automatic Control, Vol.25, pp. 49-54, 1980

4. W. T. Nye, RATTLE/DELIGHT User's Manual, University of California, Berkeley 1980

5. A. Arapostathis and E. I. Jury, "Remarks on Redundancies in Stability Criteria and a Counterexample to Fuller's Conjecture", Int. J. on Control, Vol.29, no.6, pp. 1027-1034, 1979

6. L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, N.J. 1975

7. G. F. Franklin and J. D. Powell, Digital Control of Dynamic Systems, Addison Wesley 1980

8. E. Polak and A. Tits, "A Recursive Quadratic Programming Algorithm for Semi-Infinite Optimization Problems", ERL, University of California, Berkeley, UCB/ERL M80/50 Sept. 1980

9. G. Szentirmai, FILSYN, A General Purpose Filter Synthesis Program, User Manual June 1978

10. J. Szczupak and S. K. Mitra, "Recursive Digital Filters With Low Roundoff Noise", Circuit Theory and Applications, Vol.5, pp. 275-286, 1977

*1a. A.G. Deczky, "Synthesis of recursive filters using the minimum p-error criterion", IEEE Transactions on Audio and Electroacoustics, vol AU-20, no. 4, pp. 257-263, 1972.

APPENDIX A- LISTING OF THE FILTER DESIGN PROGRAM

```
#=========
# filterS - setup
#=========

# both Ndegree and Ddegree must be even as H(z) is
# a ratio of quadratics

Ndegree = 4
Ddegree = 4
Nparam = Ndegree+Ddegree+3
Nineq = (3/2)*Ddegree
Nfineq = 4
W0 = 0
Wc = PI

Imult=1
array Ia(Ndegree/2), Ib(Ndegree/2)
for i=1 to Ndegree/2  {
   Ia(i)=2*i
   Ib(i)=2*i+1
   }
array Ic(Ddegree/2), Id(Ddegree/2)
for i=1 to Ddegree/2  {
   Ic(i) = Ndegree + 2*i
   Id(i) = Ndegree + 2*i + 1
   }
Islope = Ndegree + Ddegree + 2
Ibest_bound  = Ndegree + Ddegree + 3


# numerator quadratic: z**2 + A(i)*z + B(i)
# denominator quadratic: z**2 + C(i)*z + D(i)

define (Mult,X(1))
define (A ( i ),X(i*2))
define (B ( i ),X(2*i+1))
define (C ( i ),X(Ndegree+2*i))
define (D ( i ),X(Ndegree+2*i+1))
define (Phase_slope,X(Ndegree+Ddegree+2))
define (Best_bound,X(Ndegree+Ddegree+3))

# edge of passband = Wp/(2*PI) Hz
# edge of stopband = Wa/(2*PI) Hz
# width of passband constraint region = 2*Eps_pass*Best_bound
# width of stopband constraint region = Eps_stop*Best_bound
# angle of phase constraint sector = Eps_slope*Best_bound
# distance of phase constraint boundaries from origin = Delphase
# stability constraint: |poles| <= Rho

Wp = .45*PI
Wa = .55*PI
Eps_pass = .05
Eps_stop = .1
Eps_slope = .2
Delphase = .1
Rho=1
```

```
#==========
# filterC - cost function
#==========

function cost (x)  {
   array x()
   import Ibest_bound
   return x(Ibest_bound)
   }

procedure gradcost (x,g)  {
   array g()
   import Ibest_bound
   matop g = array() of 0
   g(Ibest_bound) = 1
   }
```

```
#=========
# filterI - inequality constraints
#=========

procedure ineq(j,x)  {
   array x()
   k = (j - modrep(j,3))/3 + 1
   import Ic, Id, Rho
   rhosq = Rho**2
   case1 (modrep(j,3)==1) return -1-x(Ic(k))/Rho-x(Id(k))/rhosq
   case2 (modrep(j,3)==2) return -1+x(Ic(k))/Rho-x(Id(k))/rhosq
   case3 (modrep(j,3)==3) return -1+x(Id(k))/rhosq
   }


procedure gradineq(j,x,g)  {
   array x(), g()
   import Ic, Id, Rho
   k = (j - modrep(j,3))/3 + 1
   rhosq = Rho**2
   matop g = array() of 0
   case1 (modrep(j,3)==1)  {
      g(Ic(k)) = -1/Rho
      g(Id(k)) = -1/rhosq
      }
   case2 (modrep(j,3)==2)  {
      g(Ic(k)) = 1/Rho
      g(Id(k)) = -1/rhosq
      }
   case3 (modrep(j,3)==3)
      g(Id(k)) = 1/rhosq
   }
```

```
#==========
# filterF - functional inequality constraints
#==========

function amp (x, w)  {

   array x()
   import Ndegree, Ddegree
   import Ia, Ib, Ic, Id, Imult
   array y(3), xx(3)

   amptemp = x(Imult)
   xx(3) = 1
   y(3) = 1
   for i = 1 to Ndegree/2  {
      xx(2) = x(Ia(i))
      xx(1) = x(Ib(i))
      y(2) = x(Ic(i))
      y(1) = x(Id(i))
      amptemp = amptemp*tranfa(xx,2,y,2,w)
      }
   if (Ndegree < Ddegree) then  {
      for i = Ndegree/2+1 to Ddegree/2  {
         y(2) = x(Ic(i))
         y(1) = x(Id(i))
         amptemp = amptemp*tranfa(1,0,y,2,w)
         }
      }
   return amptemp
   }

function phase (x,w) {  # WARNING: the phase must be calculated
                        # in a loop starting with w=0 .

   array x()
   import Ndegree, Ddegree
   import Ia, Ib, Ic, Id
   array y(3), xx(3)

   if ( w == 0.0 ) then {
      ph = 0.0
      deltaph = 0.0
      }
   else  {
      phtemp = 0
      xx(3) = 1
      y(3) = 1
      for i = 1 to Ndegree/2  {
         xx(2) = x(Ia(i))
         xx(1) = x(Ib(i))
         y(2) = x(Ic(i))
         y(1) = x(Id(i))
         phtemp = phtemp+tranfp(xx,2,y,2,w)
         }
      for i = Ndegree/2+1 to Ddegree/2  {
         y(2) = x(Ic(i))
```

```
      y(1) = x(Id(i))
      phtemp = phtemp+tranfp(1,0,y,2,w)
      }
   ph = phtemp + deltaph

   if ( abs(ph+PI - phstore) < abs(ph - phstore) ) then  {
      ph = ph + PI
      deltaph = deltaph + PI
      }
   else  {
      if ( abs(ph-PI - phstore) < abs(ph - phstore) ) then  {
         ph = ph - PI
         deltaph = deltaph - PI
         }
      }
   }
phstore = ph
return ph
}


function fineq (j, x , w)  {
   array x()
   import Ndegree, Ddegree, Wp, Wa, Eps_pass, Eps_stop, Eps_slope, Delphase
   import Islope, Ibest_bound, Ib, Ic

   amplitude = amp (x,w)

   case1 (j==1)  {
      if (w<Wa) return (amplitude - (1+Eps_pass*x(Ibest_bound)) )
      if (w>=Wa) return (amplitude - Eps_stop*x(Ibest_bound))
      }

   case2 (j==2)  {
      if (w<=Wp) return ((1-Eps_pass*x(Ibest_bound)) - amplitude)
      if (w>Wp) return -sqrt(MAXREAL)
      }

   case3 (j==3)  {
      if (w<=Wp)
         return (phase(x,w)-(x(Islope)+Eps_slope*x(Ibest_bound))*w-Delphase)
      else return -sqrt(MAXREAL)
      }

   case4 (j==4)  {
      if(w<=Wp)
         return (-phase(x,w)+(x(Islope)-Eps_slope*x(Ibest_bound))*w-Delphase)
      else return -sqrt(MAXREAL)
      }
   }


procedure gradfineq (j,x,w,g)  {
   array x(), g()
   import Ndegree, Ddegree, Wp, Wa, Eps_pass, Eps_stop, Eps_slope
   import Ia, Ib, Ic, Id, Islope, Ibest_bound, Imult
```

```
array re_num(Ndegree/2), im_num(Ndegree/2)
array re_den(Ddegree/2), im_den(Ddegree/2)

for i = 1 to Ndegree/2  {
   re_num(i) = cos(2*w)+x(Ia(i))*cos(w)+x(Ib(i))
   im_num(i) = sin(2*w)+x(Ia(i))*sin(w)
   }

for i = 1 to Ddegree/2  {
   re_den(i) = cos(2*w) + x(Ic(i)) * cos(w) + x(Id(i))
   im_den(i) = sin(2*w) + x(Ic(i)) * sin(w)
   }

if (j==1 | j==2)  {

   amplitude = amp(x,w)

   for i = 1 to Ndegree/2  {
      numsq = re_num(i)**2+im_num(i)**2
      g(Ia(i)) = (amplitude**2)*2*(re_num(i)*cos(w)+im_num(i)*sin(w))/numsq
      g(Ib(i)) = (amplitude**2)*2*re_num(i)/numsq
      }
   for i = 1 to Ddegree/2  {
      densq = re_den(i)**2 + im_den(i)**2
      g(Ic(i)) = -2*(amplitude**2)*(re_den(i)*cos(w)+im_den(i)*sin(w))/densq
      g(Id(i)) = -2*(amplitude**2)*re_den(i) / densq
      }
   g(Islope) = 0
   matop g = (1/(2*amplitude)) * g
   g(Imult) = amplitude/x(Imult)

   if (j==2) then
      if (w<=Wp) matop g = (-1)*g
      else             matop g = array() of 0

   g (Ibest_bound) = -Eps_pass
   if (j==1 & w>Wp) g(Ibest_bound) = -Eps_stop
   if (j==2 & w>Wp) matop g = array() of 0

   }


if (j==3 | j==4)  {
   for i = 1 to Ndegree/2  {
   numsq = re_num(i)**2 + im_num(i)**2
   g(Ia(i)) = (re_num(i)*sin(w)-im_num(i)*cos(w)) / numsq
   g(Ib(i)) = -im_num(i) / numsq
   }
   for i = 1 to Ddegree/2  {
      densq = re_den(i)**2 + im_den(i)**2
      g(Ic(i)) = (im_den(i)*cos(w) - re_den(i)*sin(w))/densq
      g(Id(i)) = im_den(i) / densq
      }
   g(Islope) = -w
   g(Imult) = 0
   g(Ibest_bound) = -w*Eps_slope
```

```
    if (w>Wp) matop g = array() of 0

    if (j==4)  {
       matop g = (-1)*g
       g(Ibest_bound) = -g(Ibest_bound)
       }

    }
}
```

```
#=========
# filterM - miscellaneous for graphical output
#=========


define_viewport wamp 0 .55 1 1
define_world wamp 0 0 PI 1.5
define_viewport wphase 0 0 (Wp+Wa)/(2*PI) .5
define_world wphase 0 -5 (Wp+Wa)/2 0
define_viewport wpoles .6 0 .9 .5
define_world wpoles -1.25 -1.25 1.25 1.25

define (poles ; ''color='red' ,poles_(X,color))

Sizex = .02

procedure poles_(x,col)  {
   array x()
   import Ddegree, Sizex, Ndegree
   array repole(Ddegree), impole(Ddegree)
   for i = 1 to Ddegree/2  {
      c = C(i)
      d = D(i)
      square = c**2 - 4*d
      if ( square >= 0 )  {
         repole(2*i-1) = (-c + sqrt(square))/2
         impole(2*i-1) = 0
         repole(2*i) = (-c - sqrt(square))/2
         impole(2*i) = 0
         }
      else  {
         repole(2*i-1) = -c/2
         impole(2*i-1) = sqrt(-square)/2
         repole(2*i) = -c/2
         impole(2*i) = -sqrt(-square)/2
         }
      }
   window wpoles
      for i = 1 to Ddegree  {
         vector repole(i)-Sizex impole(i)-Sizex repole(i)+Sizex impole(i)+Sizex
         vector repole(i)-Sizex impole(i)+Sizex repole(i)+Sizex impole(i)-Sizex
         }
   }


define (frame,frame_())


procedure frame_  {

   window wamp
      color white
      box
   window wphase
      color white
      box
```

```
   window wpoles
      theta = 2*PI / 40
      for k = 0 to 40
         vector cos(theta*k) sin(theta*k) cos(theta*(k+1)) sin(theta*(k+1))
      vector 0 -1.25 0 1.25
      vector -1.25 0 1.25 0
   }

define (pfr ; ''color='red' ,pfr_(color))

if_NOTTHERE algo procedure algo q=0

procedure pfr_ (col) {

   import Ndegree, Ddegree, Wp, Wa, Wc, Eps_pass, Eps_stop, Eps_slope, Delphase
   access q from algo
   if q==0 q=20

   window wamp
      tol_pass = Best_bound * Eps_pass
      tol_stop = Best_bound * Eps_stop
      color yellow
      vector 0 1+tol_pass Wa 1+tol_pass
      vector 0 1-tol_pass Wp 1-tol_pass
      vector Wa tol_stop PI tol_stop
      array resp(q+1)
      for i=1 to q+1
         resp(i)=amp(X,(i-1)*PI/q)
      curvev resp col

   window wphase
      tol_slope = Best_bound * Eps_slope
      color yellow
      clip_vector 0 Delphase Wp (Phase_slope+tol_slope)*Wp+Delphase
      clip_vector 0 -Delphase Wp (Phase_slope-tol_slope)*Wp-Delphase
      array resp(q*Wp/Wc+1)
      for i=1 to arydim(resp)
         resp(i)=phase(X,(i-1)*PI/q)
      viewport 0 0 Wp/((Wp+Wa)/2) 1 relative to wphase
      curvev resp col

   poles col
   }



# the following variables must be input from the keyboard
# before calling "convert"

# mult = multiplicative factor of H(s) in db
# ncompzer = number of complex zero pairs
# ncomppol = number of complex pole pairs
# nozeros = 1 if H(s) has no finite zeros
# nozeros = 0 otherwise
# array renum contains the real parts of the complex zeros
# arrays imnum, reden, and imden are similarly defined
```

```
# arrays s1 and s2 contain real zeros to be transformed in
# pairs (ie. (s-s1(i))*(s-s2(i)) )
# arrays s3 and s4 are similarly defined for poles

create mult
create ncompzer
create ncomppol
create nozeros
array renum(ARB), imnum(ARB), reden(ARB), imden(ARB)
array s1(ARB), s2(ARB), s3(ARB), s4(ARB)

# procedure convert performs the bilinear transformation to
# a transfer function H(s) expressed as the ratio of quadratics

procedure convert {
   import mult, Ndegree, Ddegree
   import ncompzer, ncomppol, nozeros
   import renum, imnum, reden, imden
   import s1, s2, s3, s4
   Mult = 10**(mult/20)

   #perform bilinear transform on complex
   #zero pairs

   if (nozeros = 0) {
      for i = 1 to ncompzer {
         magsq = renum(i)**2 + imnum(i)**2
         temp = 1-2*renum(i)+magsq
         A(i) = -2*(1-magsq)/temp
         B(i) = (1+2*renum(i)+magsq)/temp
         Mult = Mult*temp
         }
      }

   #perform bilinear transform on complex
   #pole pairs

   for i = 1 to ncomppol {
      magsq = reden(i)**2 + imden(i)**2
      temp = 1-2*reden(i)+magsq
      C(i) = -2*(1-magsq)/temp
      D(i) = (1+2*reden(i)+magsq)/temp
      Mult = Mult/temp
      }

   #perform bilinear transform on real
   #numerator roots (in pairs)

   if (nozeros = 0) {
      for i = ncompzer+1 to Ndegree/2 {
         k = i-ncompzer
         temp = (1-s1(k))*(1-s2(k))
         A(i) = -2*(1-s1(k)*s2(k))/temp
         B(i) = (1+s1(k))*(1+s2(k))/temp
         Mult = Mult*temp
         }
```

```
   }

   #perform bilinear transform on real
   #denominator roots (in pairs)

   for i = ncomppol+1 to Ddegree/2  {
      k = i-ncomppol
      temp = (1-s3(k)) * (1-s4(k))
      C(i) = -2*(1-s3(k)*s4(k))/temp
      D(i) = (1+s3(k)) * (1+s4(k))/temp
      Mult = Mult/temp
      }

   if (nozeros = 1)  {
      for i = 1 to Ddegree/2  {
         A(i) = 2
         B(i) = 1
         }
      }
   }


# procedure map performs the transformation Rho*z1 = z
# to H(z) where |z|<=1 for stability

procedure map  {
   import Rho, Ndegree, Ddegree

   for i = 1 to Ndegree/2  {
      A(i) = A(i)/Rho
      B(i) = B(i)/Rho**2
      }

   for i = 1 to Ddegree/2  {
      C(i) = C(i)/Rho
      D(i) = D(i)/Rho**2
      }

   C1 = Ndegree-Ddegree
   Mult = Mult * (Rho**C1)
   }
```

```
#=========
# filterD - data file
#=========

# Elliptic filter designed using FILSYN
# for this example:
# Wp = .45*PI
# Wa = .55*PI
# passband loss = .3db
# stopband loss = 23db
# filter order = 4

X(1)= .1386
X(2)= .4189
X(3)= 1.000
X(4)= 1.494
X(5)= 1.000
X(6)=-.4599
X(7)= .2205
X(8)=-.1883
X(9)= .7830
X(10)=-2.000
X(11)= 1.900
```

APPENDIX B- LISTING OF THE OPTIMIZATION ALGORITHM

```
#==========
# Dgopotra - Direction finding subprocedure using the Gonzaga-Polak-Trahan
#========== phase1/phase2 method of feasible directions.

DESCRIPTION : Gonzaga-Polak-Trahan MFD (for semi-infinite problems)

parameter Gamma = 2.

procedure optfunc (x, eps, q, theta, h)  {

   array x(), h()
   import Gamma
   n = arydim(x)
   array jacobian(Nineq+Nfineq*(q+1)+1,n)
   array grad(n)

   evaluate grad = Gradcost(x)
   fill jacobian(1,) = grad'
   matop r = array(Nineq+Nfineq*(q+1)+1) of 0

   psiplus = maxviolq(x,q)
   r(1) = Gamma * psiplus

   k = 1

   for j=1 to Nineq
      if (Ineq(j,x) >= psiplus-eps) then  {
         k = k + 1
         evaluate grad = Gradineq(j,x)
         fill jacobian(k,) = grad'
         r(k) = psiplus - Ineq(j,x)      # (1)
         }


# computing eps-active left local maximizers

   for j=1 to Nfineq  {
      oldval = -MAXREAL
      for l=0 to q  {
         if ( Dfineq(j,x,l,q) > oldval )  {
            up = TRUE
            if ( Dfineq(j,x,l,q) >= psiplus-eps )  {
               lmax = l
               locmax = Dfineq(j,x,l,q)
               active = TRUE
               }
            else
               active = FALSE
            }
         if ( ( (Dfineq(j,x,l,q) <= oldval) | l==q ) & up )  {
            up = FALSE
            if active then  {
               k = k + 1
               evaluate grad = dgradfineq (j, x, lmax, q)
               fill jacobian(k,) = grad'
               r(k) = psiplus - Dfineq(j,x,lmax,q)
```

```
            }
          }
        oldval = Dfineq(j, x, l, q)
        }
    }

  clip e_jacobian = jacobian(1:k, )
  matop qq = e_jacobian * e_jacobian' ; matop qq = (1/2)*qq
  clip e_r = r(1:k)
  matop a = array(1, k) of 1
  matop b = array(1) of 1

  quadprog e_mu = argmin { x'*qq*x + e_r'*x | a*x>=b, x>=0} '

  matop h = e_jacobian' * e_mu ; matop h = (-1)*h

  theta = -!!h!!**2 /2  - <<e_r, e_mu>>

  }


# (1) if instead we leave r(k)=0, then, when X is feasible, we have r()=0
#     and, when all constraints are e_active, the quadratic program is
#     constant on an entire subspace and fails to be solved correctly.
```

```
#=========
# Saf12pm
#=========

DESCRIPTION : Armijo '+-' for phase 1-2 MFD with functional constraints.

#  Both positive and negative values of the exponent k are tried.
#  The first try is with the stepsize (k) used at previous iteration.

parameter Alpha = .5
parameter Beta = .5
create Delta
parameter Kmin = -5

function stepsize (x, h, eps, q)  {

   array x(), h()
   import Alpha, Beta, Delta, Kmin

   psiplus = maxviolq (x, q)

   kstart = k                 # will be 0 the first time the routine is called

   repeat  {
      update xnew = x + Beta**k * h
      psiplusnew = maxviolq (xnew, q)
      breakpt

      if (psiplus==0) then  {
         delta_cost = Cost(xnew) - Cost(x)
         if ((psiplusnew==0) &
             (delta_cost <= -Alpha * Beta**k * Delta * eps))  break
         }

      else  {
         if (psiplusnew==0) return Beta**k
         delta_psi = psiplusnew - psiplus
         if (delta_psi <= -Alpha * Beta**k * Delta * eps)  break
         }

      k = k + 1
      }
   forever

   if (k>kstart) return Beta**k

   k = kstart - 1

   repeat  {
      update xnew = x + Beta**k * h
      psiplusnew = maxviolq (xnew, q)
      breakpt

      if (psiplus==0) then  {
         delta_cost = Cost(xnew) - Cost(x)
         if ((psiplusnew!=0) |
```

```
            (delta_cost > -Alpha * Beta**k * Delta * eps))  break
         }

      else  {
         if (psiplusnew==0) return Beta**k
         delta_psi = psiplusnew - psiplus
         if (delta_psi > -Alpha * Beta**k * Delta * eps)  break
         }

      k = k - 1
      }
until (k < Kmin)

k = k + 1
return Beta**k
}
```

```
#========
# Cffdir
#========

DESCRIPTION : feasible directions main loop with functional constraints

parameter Eps0  = .1
parameter Epsmin = .001
parameter Delta = 1.
parameter Q0 = 10
parameter Mu1 = 1
parameter Mu2 = 1

procedure algo  {

   import Epsmin, Eps0, Delta, Q0,  Mu1, Mu2
   array h(Nparam)

   q = Q0

   repeat  {

      eps = Eps0

      repeat  {
         psiplus = maxviolq (X[Iter], q)
         interaction
         repeat  {
            evaluate theta, h = optfunc (X[Iter], eps, q)
            if (theta <= -Delta*eps) break
            eps = eps/2
            if ( (eps<Eps0*Mu1/q) & (psiplus<Mu2/q) )  {
               q = 2 * q
               if (q<=qmax) next 3  else  break 3
               }
            }
         forever
         lambda = stepsize (X[Iter], h, eps, q)
         update X[Iter+1] = X[Iter] + lambda * h
         Iter = Iter + 1
         }
      forever

      }
   forever

   return
   }
```

APPENDIX C- A SAMPLE FILSYN RUN

A listing of a sample FILSYN run is included here. For a detailed explanation of the meanings of zs, q, input and output terminations, and the options available when analysis is requested, the reader is referred to [9]. For our purposes, it will suffice to note that the responses shown are those necessary for the design of an analog filter suitable for transformation to a digital filter.

In the example shown, an analog elliptic filter was designed. This was then transformed, using the bilinear transformation procedure contained in the optimization system, to a digital filter. Elliptic filters are characterized by equiripple magnitude response in both the passband and stopband. It was required that in the discrete domain the passband cutoff frequency $w=.45\pi$ rads and the stopband cutoff frequency $w=.55\pi$ rads. FILSYN requests the analog frequency in hertz. Hence, it was necessary to prewarp the frequency scale. The following mapping was used.

$$\Omega=\tan(wT/2)$$

where

Ω=analog frequency in rads

w=discrete frequency in rads

T=sampling period (chosen=1)

Then the analog radian frequency was converted to hertz.

$$f(Hz)=\Omega/2\pi$$

Using trial and error to find the fourth order elliptic filter having the greatest stopband loss and smallest passband ripple, 23db and .3db were the respective values computed.

The optimization based filter design system contains a routine for performing the bilinear transformation. An analog transfer function of the form

$$H(s)=A\frac{\prod_{i=1}^{m}(s+z_i)(s+\bar{z}_i)}{\prod_{i=1}^{m}(s+p_i)(s+\bar{p}_i)}$$

is transformed to

$$H(z)=K\frac{\prod_{i=1}^{l}(z^2+a_iz+b_i)}{\prod_{i=1}^{l}(z^2+c_iz+d_i)} \qquad l=\begin{cases}\frac{n}{2} & n>m\\ \frac{m}{2} & m\geq n\end{cases}$$

FILSYN returns the real and imaginary parts of the numerator roots in rads/sec, the real and imaginary parts of the denominator roots in rads/sec and A, the multiplicative factor expressed in db. These values can be input directly to the bilinear transformation procedure and H(z) computed.

A SAMPLE FILSYN OUTPUT

```
 placer(p), filsyn(f), norton(n), digital(d), end(e),
 save-last-input(s) or modify-a-file(m)
f
 enter title
elliptic filter with prewarping
 wish to read file (y or n)
n
 filter type - lowpass: 1, highpass: 2, lin.-phase lowpass: 3, bandpass: 4
  1
 upper edge of the passband in hz
  0.13593000000000
 passband kind - max.-flat: 0, equal-ripple: 1, functional input: 2
  1
 what is the band edge loss in db
  0.30000000000000
 enter rs (-1,0 or 1). if don't care, enter 0
  -1
 monotonic(0), equal-minima(1) or arbitrary(2) stopband
  1
 wish to specify minimum required stopband loss (y or n)
y
 enter loss in db
   23.000000000000
 enter edge frequency of upper stopband in hz
  0.18635000000000
 enter input termination in ohms
   1.0000000000000
 enter output termination (0. indicates open or short)
  0.
 enter value of average q. if no predistortion, enter 0.
  0.
 is analysis required (y or n)
n
1general filter synthesis program

 elliptic filter with prewarping

   low-pass filter

      equal ripple pass band

         bandedge loss                              =  0.3000 db.

         upper passband edge frequency              =  1.3593000d-01 hz.

      equal minima stop band with edge frequency    =  1.8635000d-01 hz.

         required stop band loss                    =   23.00 db.

         multiplicity of zero at infinity           =  0

         number of finite transmission zero pairs   =  2

         overall filter degree                      =  4
```

```
        transmission zeros

              real part          imaginary part

           0.          d+00       1.9685158d-01
           0.          d+00       4.1826708d-01

        input termination                          =  1.0000000d+00 ohms

        output termination                         =  0.          d+00 ohms

        requested termination ratio                =  0.          d+00
 wish to see transfer function (y or n)
y
1   intermediate results

  elliptic filter with prewarping

     ******* transfer function polynomials *******

        numerator  roots in rad/sec

       real part              imaginary part


    0.                d+00    1.2368549402954 79d+00
    0.                d+00    2.6280495931458 12d+00

        denominator  roots in rad/sec

       real part              imaginary part


   -4.6383513494915 54d-01    4.8733219055741 37d-01
   -1.1008138302437 69d-01    8.9264150494552 25d-01

      gain factor needed for unity in-band gain    = -2.9504933d+01 db.
 realization - active: a, passive: p, digital: d, no synthesis: e
e
 placer(p), filsyn(f), norton(n), digital(d), end(e),
 save-last-input(s) or modify-a-file(m)
e
```

FIGURE CAPTIONS

Fig. 1. Amplitude constraints

Fig. 2. Phase constraints

Fig. 3. FILSYN designed elliptic filter

Fig. 4. Chebyshev filter

Fig. 5. Butterworth filter

Fig. 6. Elliptic filter

Fig. 7. Phase linearization on elliptic filter

Fig. 8. Phase linearization on elliptic filter with narrow transition band

Fig. 9. FILSYN design of an elliptic filter with modified stability requirement through scaling in the z-plane

Fig.10. Improved stability using optimization applied to a FILSYN designed elliptic filter

Fig.11. Inappropriate phase constraints

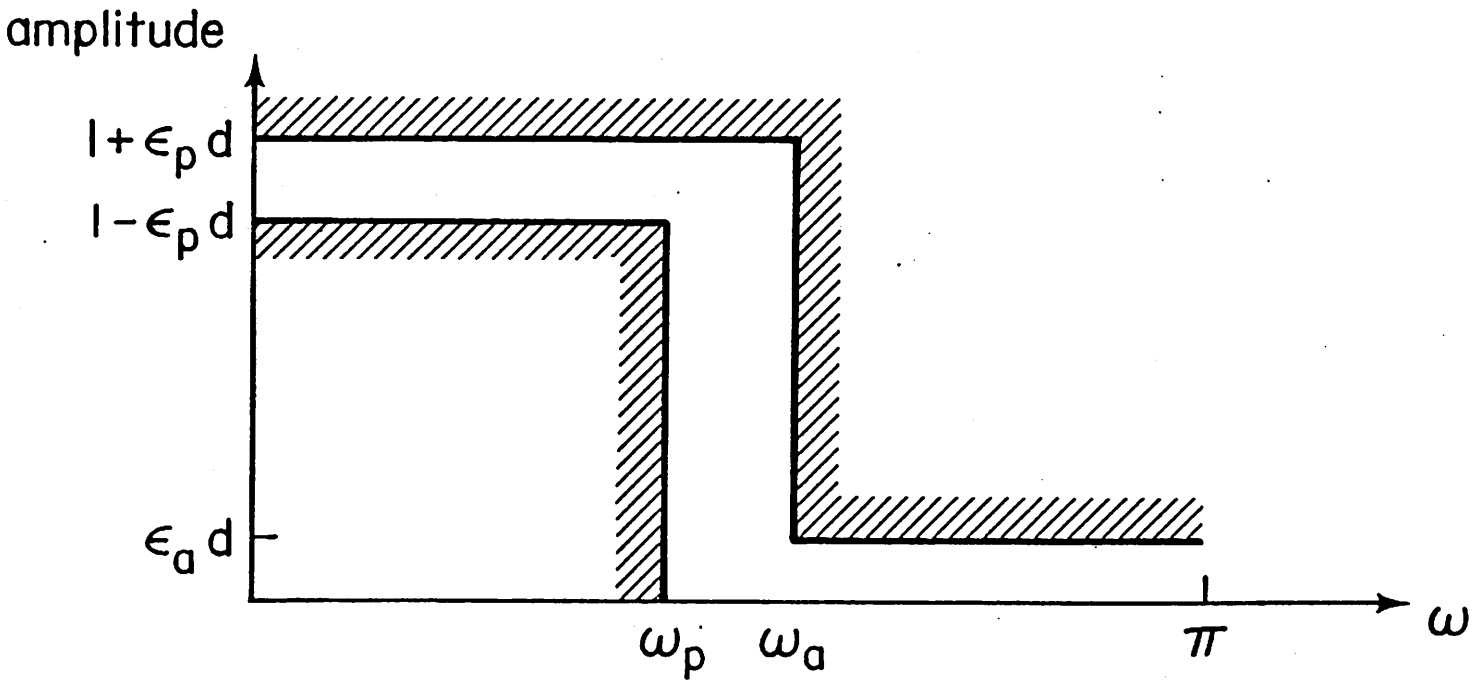

amplitude
$1+\epsilon_p d$
$1-\epsilon_p d$
$\epsilon_a d$
ω_p
ω_a
π
ω

fig. 1

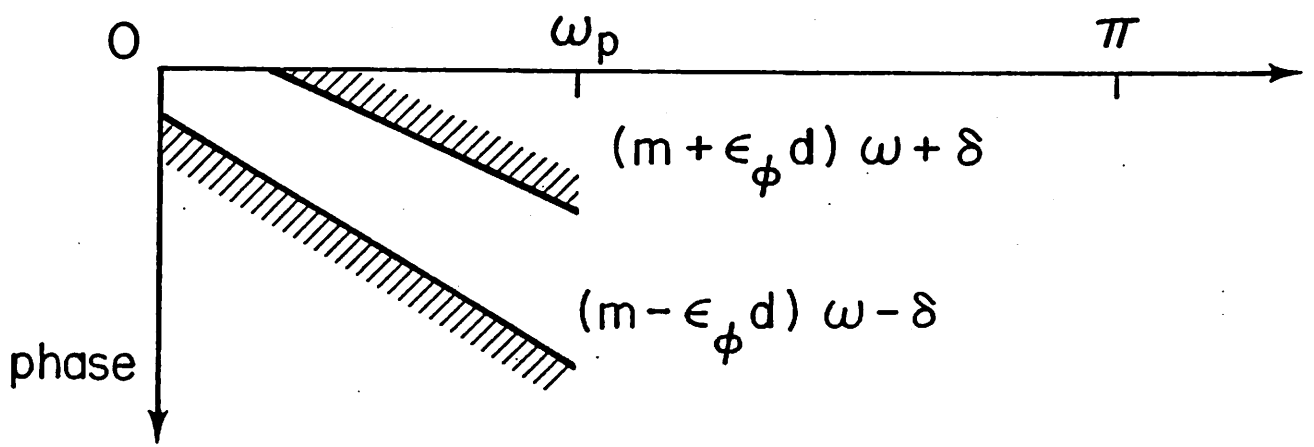
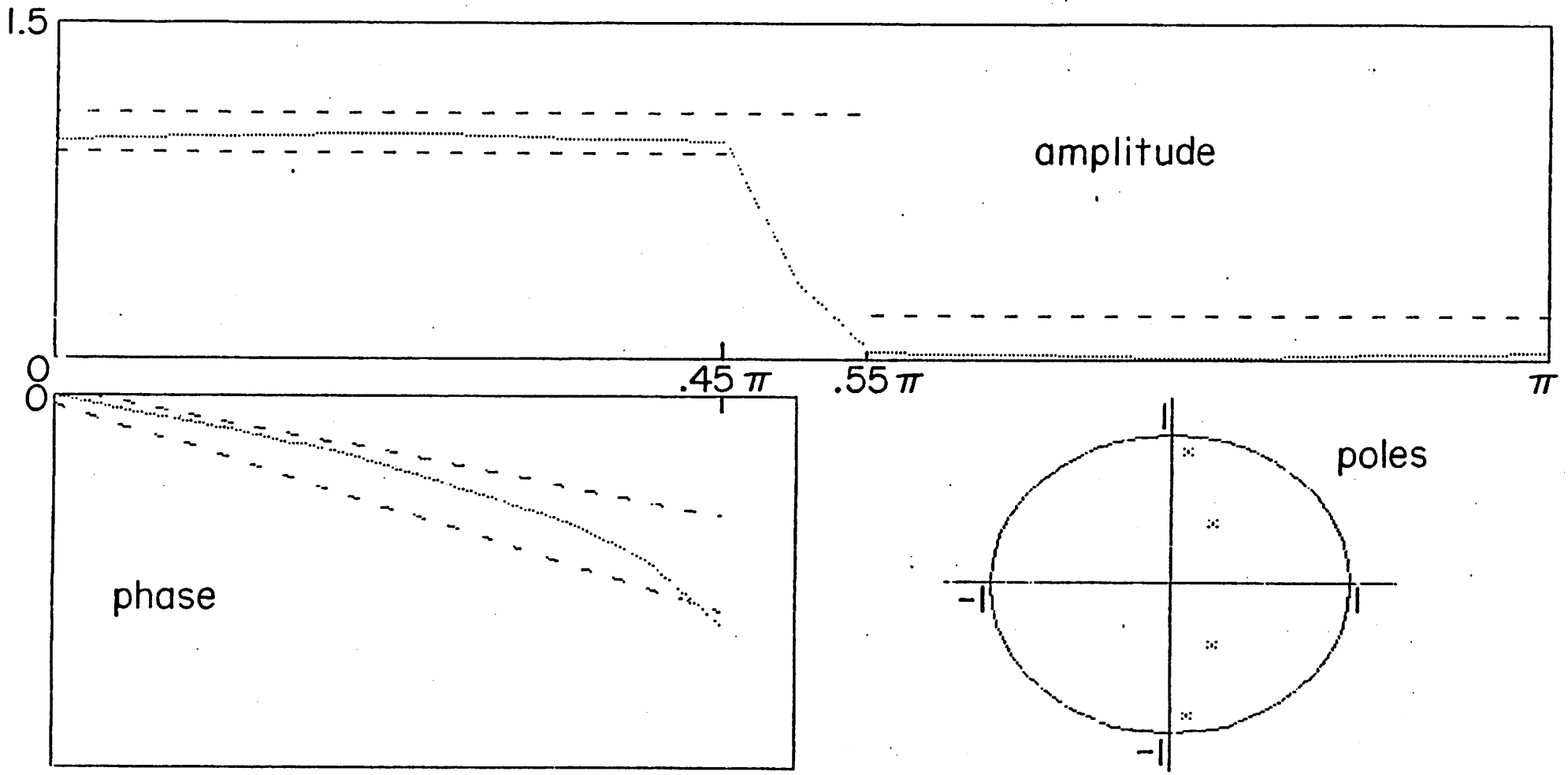

0
ω_p
π
$(m+\epsilon_\phi d)\,\omega+\delta$
$(m-\epsilon_\phi d)\,\omega-\delta$
phase

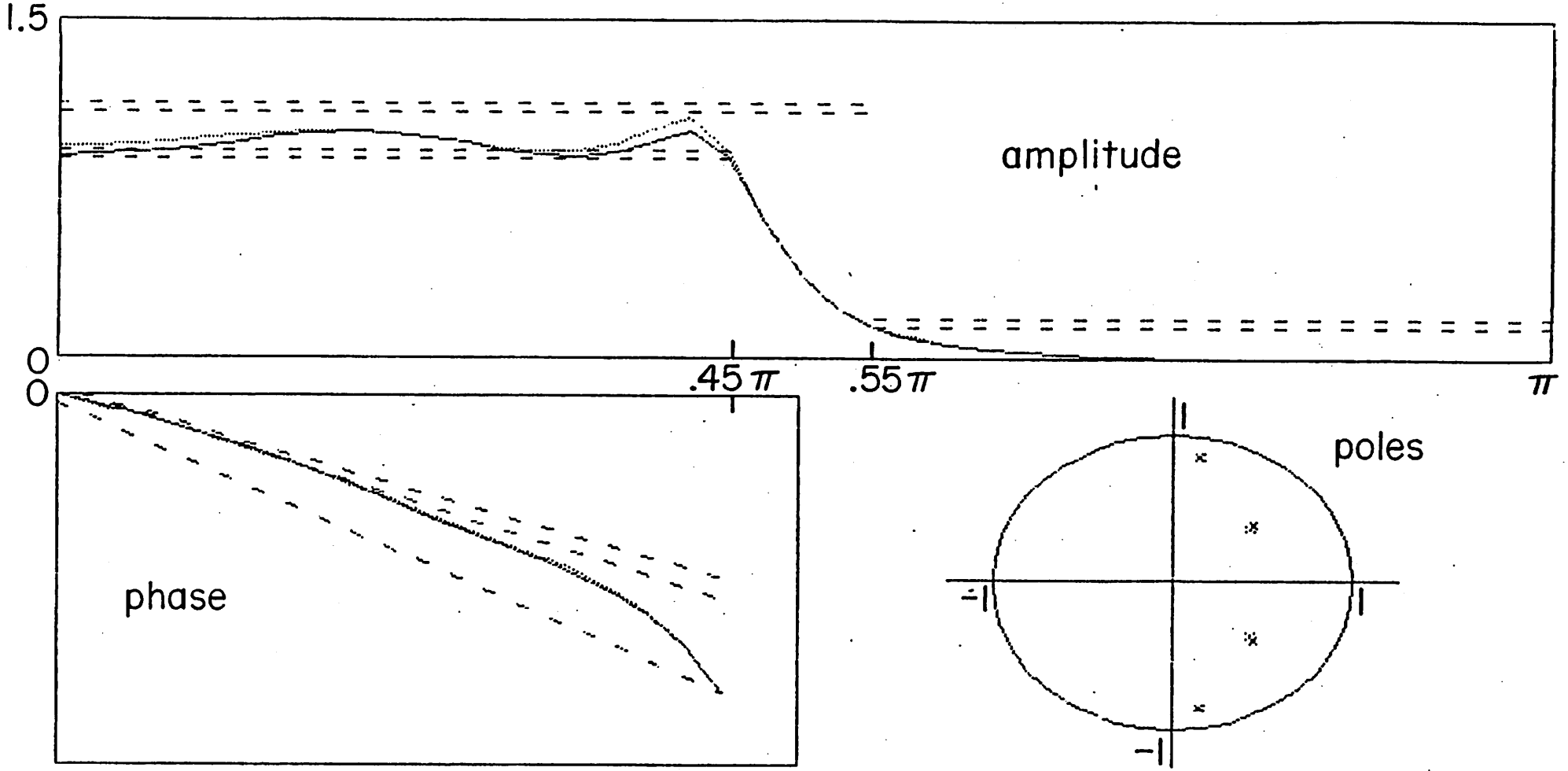
Fig. 2



$\epsilon_p = .03$

$\epsilon_a = .03$

$\epsilon_\phi = .25$



—— FILSYN Design

$\epsilon_p = .10$

$\epsilon_a = .15$

$\epsilon_\phi = .40$

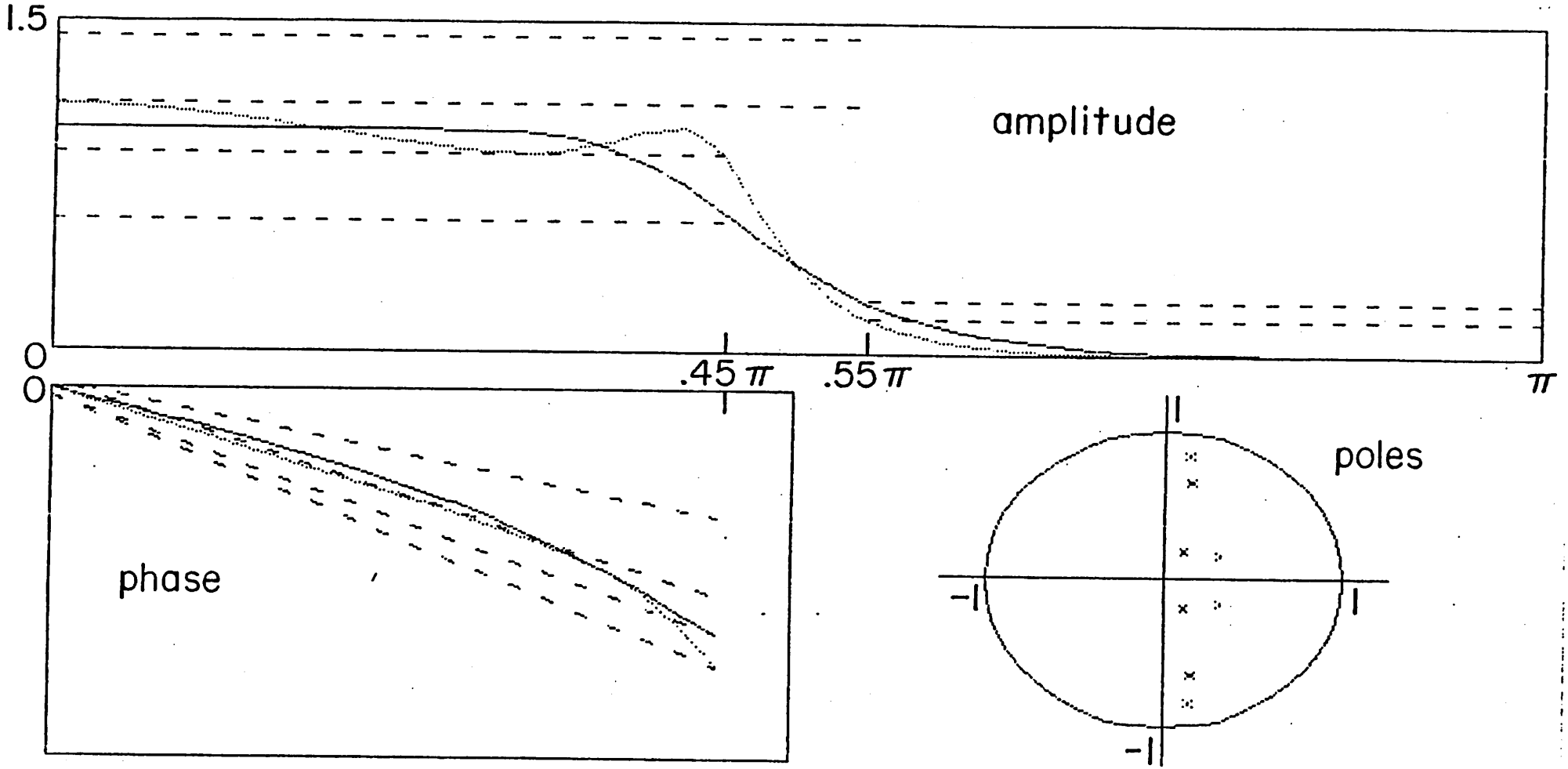
····· Optimized Design

$\epsilon_p = .09$

$\epsilon_a = .15$

$\epsilon_\phi = .40$

2.4



—— FILSYN Design

$\epsilon_p = .35$

$\epsilon_a = .20$

$\epsilon_\phi = .40$

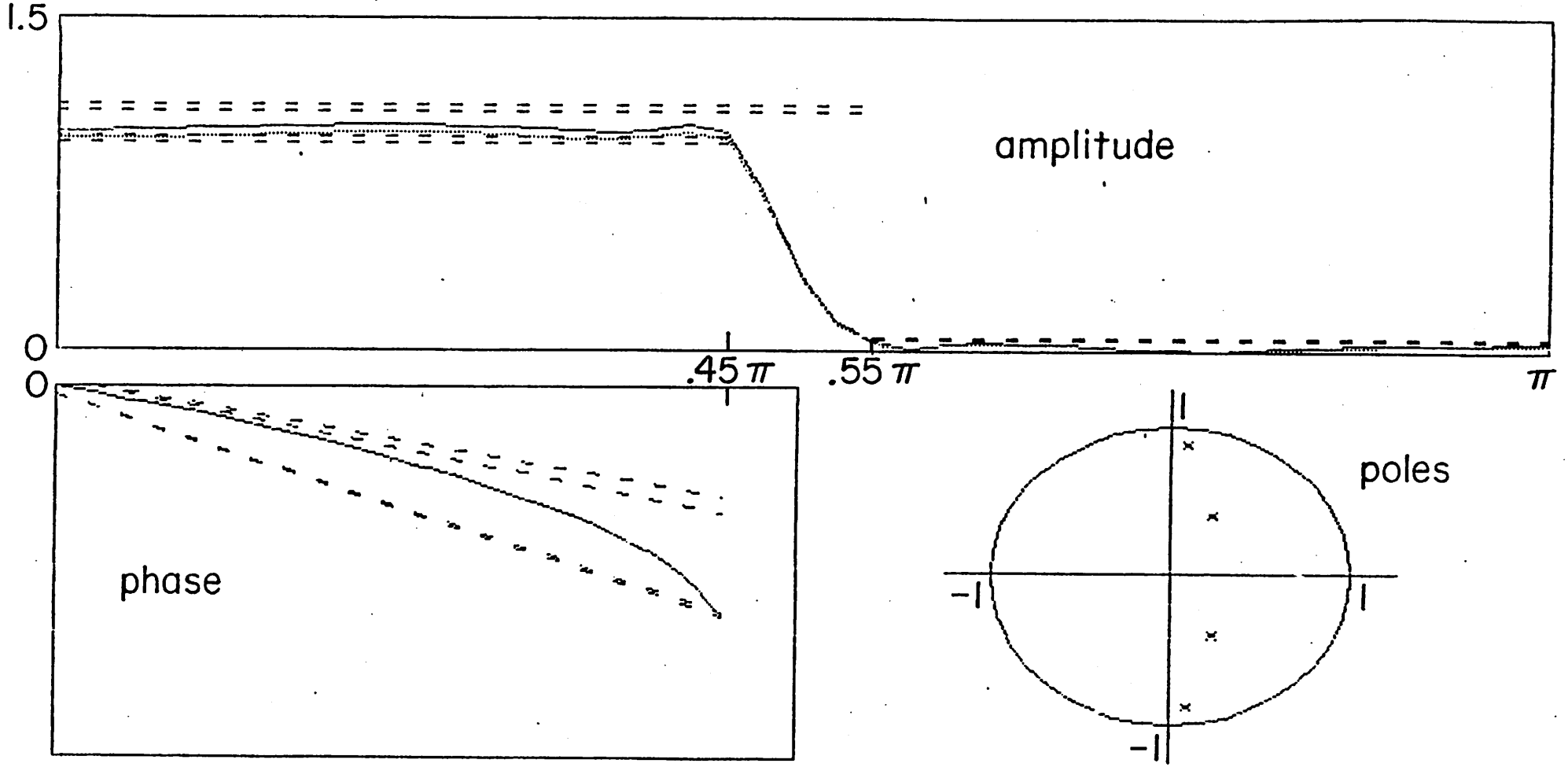
····· Optimized Design

$\epsilon_p = .25$

$\epsilon_a = .35$

$\epsilon_\phi = .60$

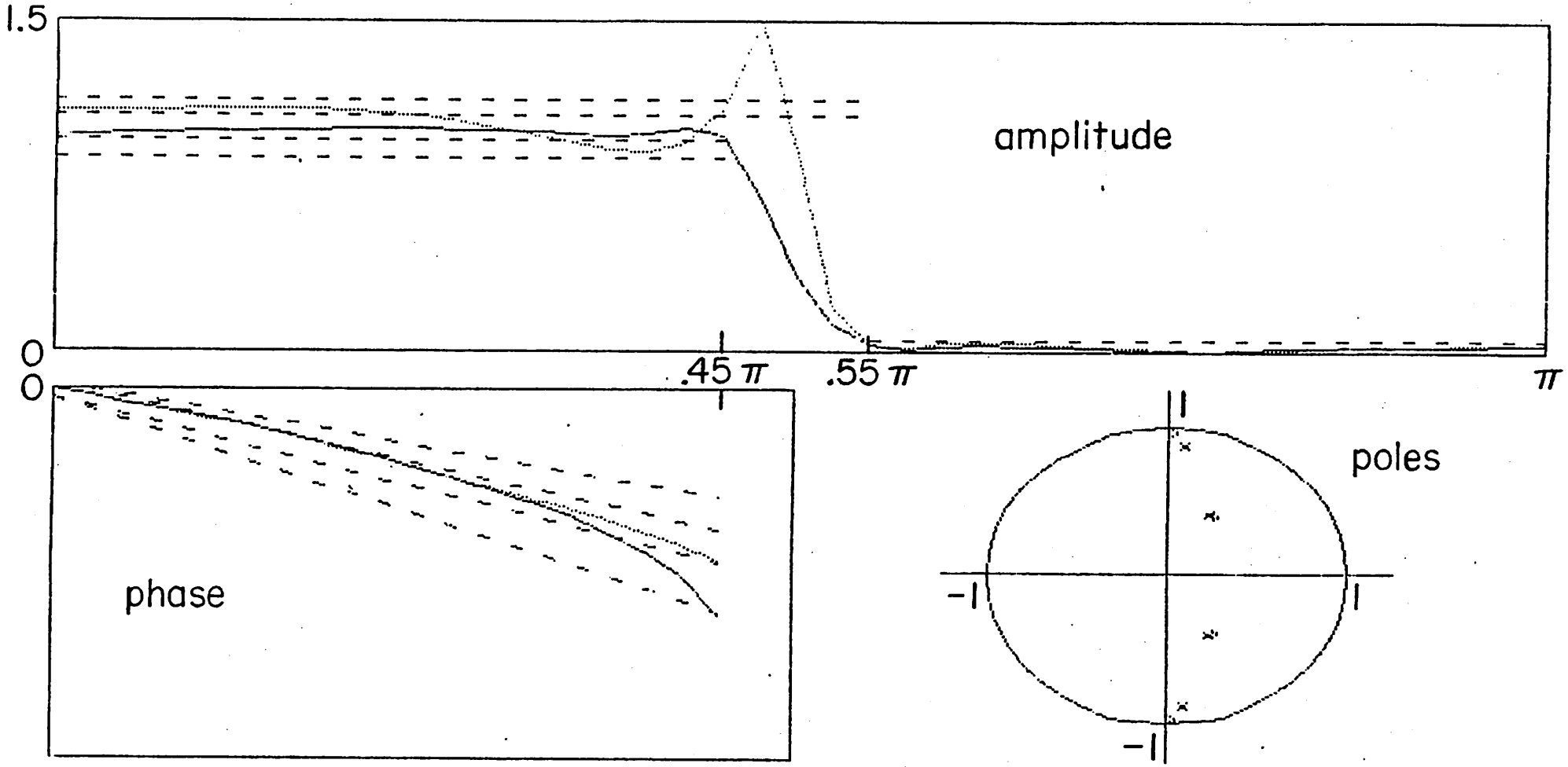
fig. 5



——— FILSYN Design
$\epsilon_p = .03$
$\epsilon_a = .03$
$\epsilon_\phi = .25$

····· Optimized Design
$\epsilon_p = .06$
$\epsilon_a = .03$
$\epsilon_\phi = .30$

fig. 6



—— FILSYN Design

$\epsilon_p = .03$

$\epsilon_a = .03$

$\epsilon_\phi = .25$

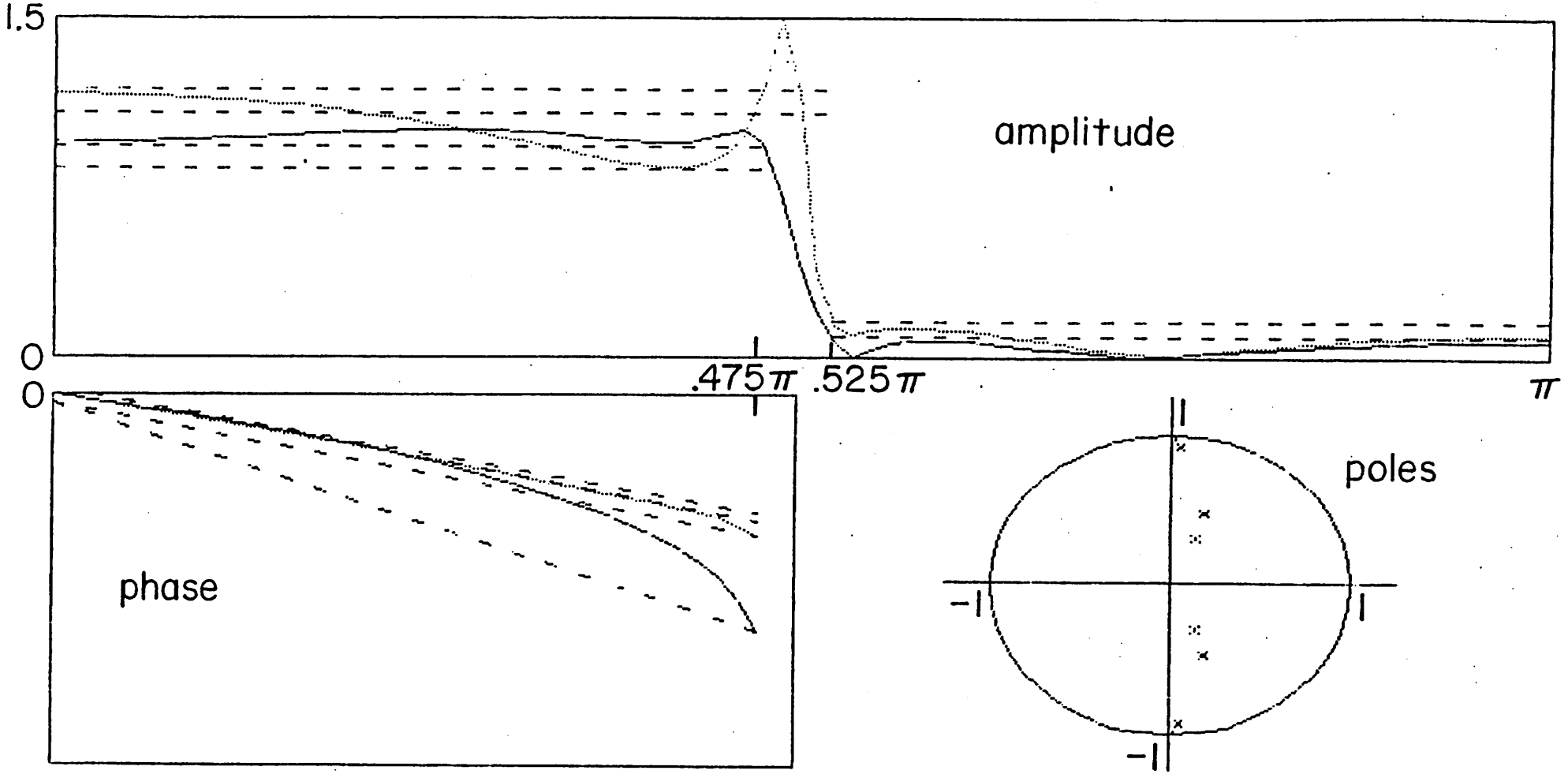
····· Optimized Design

$\epsilon_p = .15$

$\epsilon_a = .06$

$\epsilon_\phi = .10$

Fig. 7



——— FILSYN Design

$\epsilon_p = .04$

$\epsilon_a = .05$

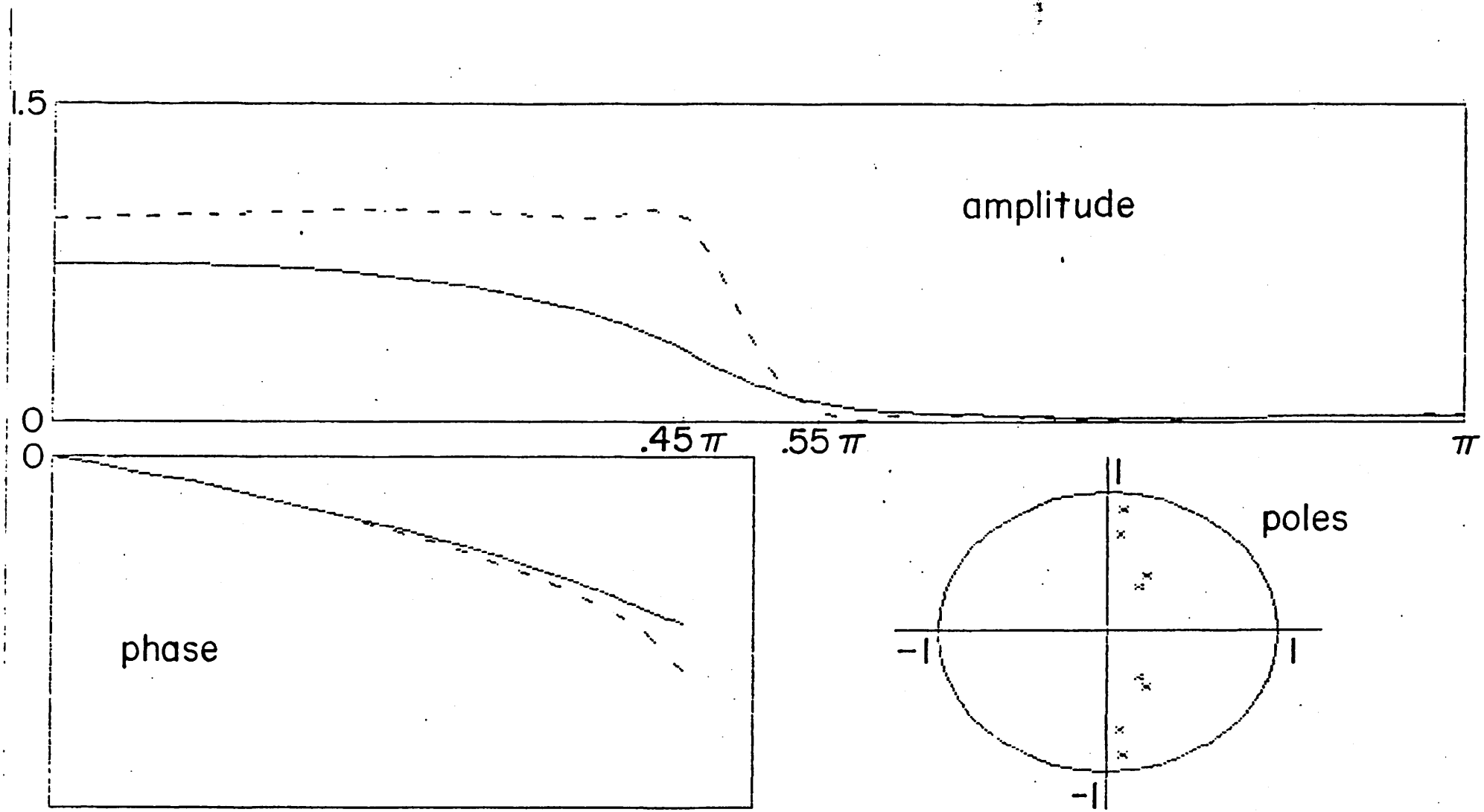
$\epsilon_\phi = .25$

····· Optimized Design

$\epsilon_p = .16$

$\epsilon_a = .14$

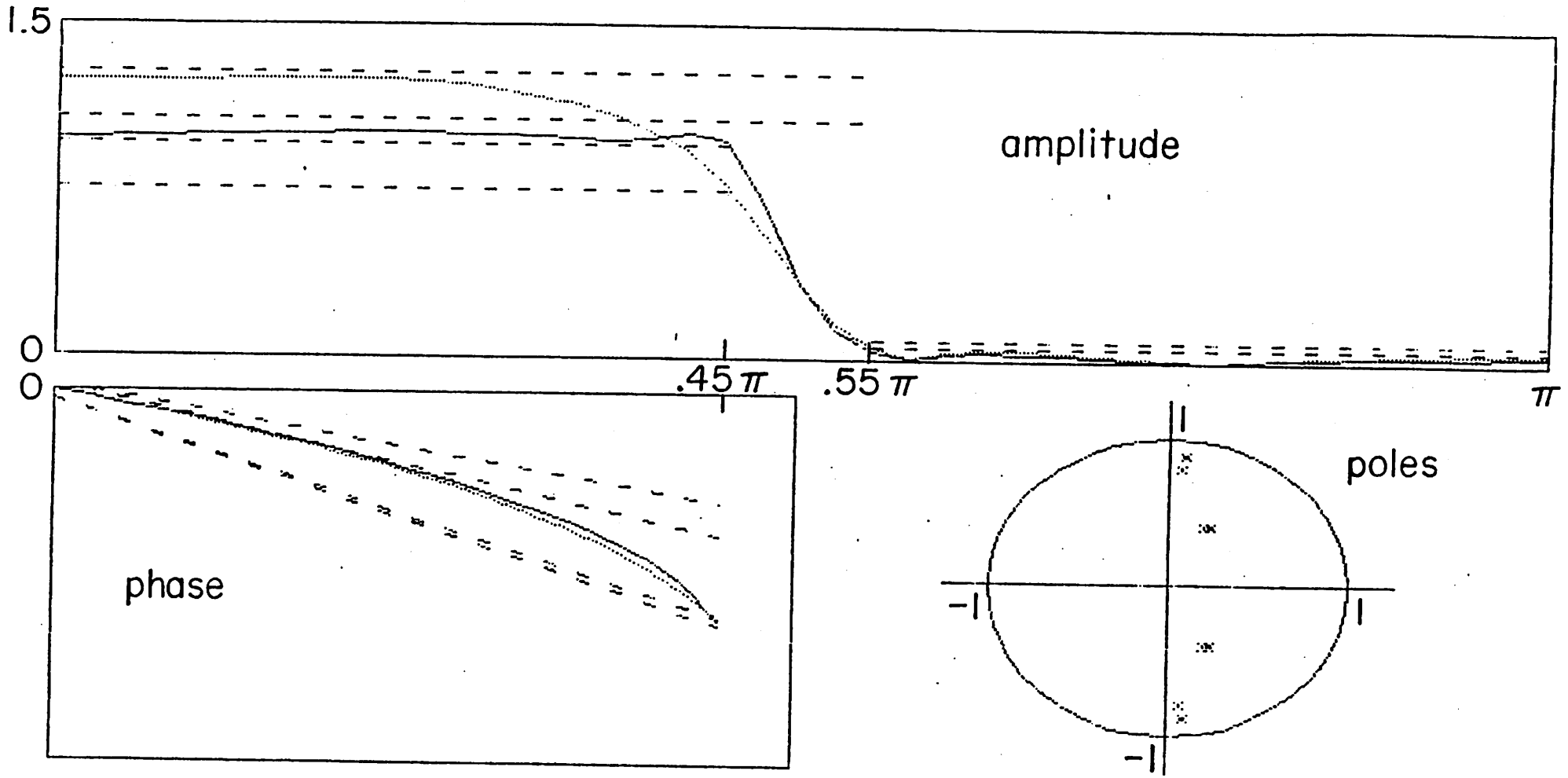
$\epsilon_\phi = .002$



---- FILSYN Design

—— Stability criteria applied to FILSYN Design

Fig. 7



—— FILSYN Design

$\epsilon_p = .03$

$\epsilon_a = .03$

$\epsilon_\phi = .25$

····· Optimized Design

$\epsilon_p = .22$

$\epsilon_a = .07$

$\epsilon_\phi = .30$

Fig. 10

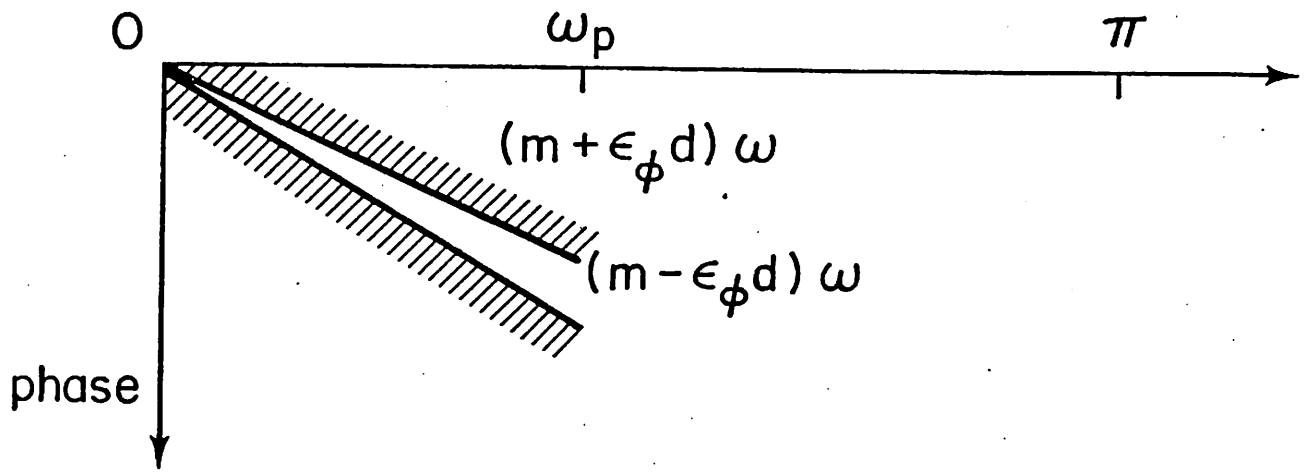


fig. 11