Report R-225

Summary of

TREATMENT OF DIGITAL CONTROL SYSTEMS AND NUMERICAL PROCESSES IN THE FREQUENCY DOMAIN

(Sc.D. Thesis in Electrical Engineering, M.I.T., 1951, by J.M. Salzer)

Edited by
John W. Craig, Jr.

DIGITAL COMPUTER LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Cambridge 39, Massachusetts

July 1, 1953

Report R-225

FOREWORD

This report is a condensation of the doctoral thesis of John M. Salzer. The condensation was made so that the work could receive wider circulation than was possible with the original document which is quite long. For those needing greater detail than the condensation provides, the complete report is available on loan from the Digital Computer Laboratory or can be obtained on Microfilm from the M.I.T. Library.

Signed: John W Craig, Jr.
John W. Craig, Jr.

Approved: William K Linvill
W.K. Linvill

May 25, 1953

Approved: Jay W Forrester
J.W. Forrester

ABSTRACT

This thesis develops methods of frequency analysis and synthesis of digital computer programs describable in the form of a linear difference equation with constant coefficients.

The mainspring of this investigation was the need for dealing with control systems consisting of both analog and digital filters. Most conventional control systems consist of analog units and operate on continuous data, but digital computers use sampled data. A uniform treatment of the two types of data is essential in the analysis of control systems incorporating a digital computer. The conventional method of treating systems operating on only continuous data uses Fourier or Laplace transformation; that is, transformation to the frequency domain. The conventional method of treating digital programs is numerical analysis, which deals almost exclusively in the domain of the independent variable; that is, the time domain. By exploiting and further developing those areas of numerical analysis to which frequency-transformation techniques were applied, the thesis points the way to a common language of dealing with a mixed-data system.

If data are sampled at equal intervals of time (a practical feature), description of a linear computer program always reduces to a difference equation. It is possible to describe such a program by a transfer function in the frequency domain in a manner analogous to the conventional description of analog filters. Whereas components using continuous data have transfer functions which are rational functions of the complex frequency

variable s, those of a digital program are rational functions of $z = e^{-sT}$, where e is the Naperian base and T is the constant interval of sampling.

Having described the digital computer with its program by a transfer function, one may apply all the techniques of complex-variable and transform theory to deal with digital filters. Theorems on realizability, stability and other properties of programs are developed, and the amplitude, phase and locus of a program are defined. The adaptation of the methods of analog filters to digital ones is direct, although the necessary modifications are often significant.

The synthesis of computer programs can be conducted along lines employed in the synthesis of networks. First, the desired frequency characteristics of the program are stated; next, a rational function of $z = e^{-sT}$ is found which approximates the desired characteristics for real frequencies, $s = j\omega$; finally, the program is realized on basis of the approximating transfer function. For facilitating the approximation basic entities or blocks of programs are analysed and methods are shown by which such programming units can be combined to obtain the frequency characteristics of the complete program. Various methods of program realization, that is, programming, are developed and compared on the basis of time and storage requirements, and criteria are developed to permit the choice of the optimum programming procedure by considering the mere form of the program transfer function.

Numerous examples of program analysis and synthesis are shown, and one example of synthesizing a program for the compensation of a control system is worked out. The latter example shows that the frequency analysis of a complete hybrid system can be undertaken along the conventional lines and that digital compensation of a control system is possible.

The application of the methods of the thesis to various problems in numerical analysis is also shown. The problems of convergence (stability) and of truncation errors (approximation) can be analyzed in the frequency domain effectively. The study of convergence by conformal mapping is related to the usual methods, and a novel way of estimating truncation error is shown provided only that the function to which the numerical process is applied can be described by its frequency spectrum.

TABLE OF CONTENTS

	Page
FOREWORD	ii
ABSTRACT	iii
INTRODUCTION	1
CHAPTER I. DESCRIPTION OF THE SAMPLING PROCESS	5
1.1 Sampling a Continuous Function	5
1.2 Equivalent Mathematical Model of Ideal Sampling -- Impulse Modulation	6
1.3 Use of Impulse Modulated Functions in the Analysis of Linear Digital Computer Programs	8
1.4 Laplace Transforms of Impulse Modulated Functions	9
CHAPTER II. TRANSFER FUNCTION OF COMPUTER PROGRAMS -- REALIZABILITY AND STABILITY	18
2.1 Transfer Function of Linear, Real-Time Digital Computer Programs	18
2.2 Stability of Programs	21
2.3 Loci of Q(s)	26
CHAPTER III. ANALYSIS AND SYNTHESIS OF LINEAR, DIGITAL COMPUTER PROGRAMS IN THE FREQUENCY DOMAIN	31
3.1 Responses of Programs at Real Frequencies	31
3.2 Analysis of Building Blocks of Transfer Functions	31
3.3 Realization of Programs from their Transfer Functions	39
3.4 Synthesis of Programs in the Frequency Domain	64

TABLE OF CONTENTS

(Continued)

	Page
CHAPTER IV. FREQUENCY ANALYSIS OF SOME NUMERICAL INTEGRATION FORMULAS	70
4.1 Numerical Integration	70
4.2 Comparison of Numerical Integration Formulas	75
APPENDICES	
A. Proof that the Locus of $Q(j\omega)$ Crosses the Real Axis Either Normally or Tangentially at $\omega = 0$ and $\pm \infty$	78
B. Coding of Direct Regression Programs	81
C. Coding of Cascaded Programs	89
D. Coding of Parallel Programs	93

INTRODUCTION

The use of digital computers in control systems is now coming into the fore. Unlike most conventional control systems involving analog units which operate on continuous data, a control system employing a digital computer of the present-day type must use sampled data in the part of the system involving the digital computer. Hence, some parts of this system use continuous data and others, sampled data. The Fourier and Laplace transform methods of analyzing continuous-data control systems is well-known and developed, but the conventional treatment of digital computer programs is by numerical analysis or in the time domain. Therefore, in order to apply the methods of frequency analysis to control systems involving digital computers (mixed-data systems), the sampled data part of the system must be described in the frequency domain. Some work along these lines has been done but it must be further developed.

An analog system is a physical model of a set of differential equations; whereas, a digital system is a physical model of a set of difference equations. Operational and transform methods have been applied to difference equations for some time. In 1942 Gardner and Barnes[1] presented a comprehensive and systematic treatment of the solution of linear difference equations with constant coefficients by the Laplace transform method. However, they do not deal with stability and errors

---

[1] Gardner and Barnes, Transients in Linear Systems, John Wiley and Sons New York, 1942, Chapter IX.

which are important in control applications. The control point of view is stressed in Tustin's[1] work on time sequences. In 1949 and 1950 Tustin's method was further developed by Madwed[2], who shows the relations of his aspects of stability, but they do not analyze the errors associated with their approximations.

In the meantime, Hurewicz[3] pioneered the analysis of pulsed filters in the frequency domain, developed stability criteria, and showed several examples of choosing parameters. It should be noted, however, that Hurewicz's filters are only simple units such as differentiators and lead networks, which are incapable of performing involved computations as a computer can. Also, Hurewicz evaluates the output of a pulsed filter at the sampling instants only. The behavior of the filter between pulses remains a separate problem, and no ready method is presented to investigate the whole question in the frequency domain.

W. K. Linvill[4] shows that sampling a continuous function is equivalent to the modulation of a series of unit impulses by the function. The result is a new time function which can be thought of as being applied to the sampled data part of the system. Furthermore, this new time function has a Laplace transform; thus a frequency-domain analysis is possible. Linvill shows that reconversion from discontinuous to continuous

---

1 Tustin, A Method of Analysing the Behavior of Linear Systems in Terms of Time Series, J.I.EE. Vol. 94, Part 2A,#1, pp. 130 - 142.

2 Madwed, Number Series Method of Solving Linear and Non-Linear Differential Equations, SC.D. thesis in Mechanical Engineering, MIT.

3 Hurewicz, Filters and Servo Systems with Pulsed Data, Chapter 5 of James, Nichols and Phillips, Theory of Servomechanisms.

4 Linvill, W.K., Analysis and Design of Sampled-Data Control Systems, Digital Computer Laboratory, MIT, Report R-170.

data is a filtering process and also shows what happens when the loop is closed on a mixed data system. He is concerned only with the effect of sampling on the system and does not consider the influence of digital computer operations on the system.

This report is a summary of the work done by Salzer[1]. His results permit the analysis of linear digital computer programs in the frequency domain; i.e., the operation of a digital computer program is described by a transfer function. Thus the field is opened for the complete analysis and synthesis, wholly in the frequency domain, of control systems employing digital computers.

From the frequency-domain point of view, conditions governing the realizability of program transfer functions are developed, the problem of stability is studied, and conditions to insure stability are given. Three methods of realization of programs from their transfer functions are presented, and the time and storage requirements of each are studied. An elementary example of transfer function synthesis is given. As in the case of network theory, the analysis of a computer program in the frequency domain is straightforward with a unique result, but the synthesis of a transfer function has many alternate realizations. Also as in network theory, the characteristics of the transfer function to be realized may not be given directly in a form leading to immediate realization but an intermediate approximation problem may need to be solved. The background for solving the approximation problem has been set up in that conditions of physical

---

[1] Salzer, J.M., Treatment of Digital Control Systems and Numerical Processes in the Frequency Domain, SC.D. thesis in Electrical Engineering

realizability have been derived and methods of realization of all realizable transfer functions have been obtained. While some work has been done directly on the approximation problem, much remains to be done in this respect.

The function of this report is to provide a concise picture of the frequency analysis of digital control systems and numerical processes. The first chapter describes the processes of sampling and desampling continuous functions and indicates that sampling is analogous to impulse modulation while desampling is analagous to ripple filtering in demodulation. Thinking of sampling as impulse modulation allows one to relate the sampled to the continuous function in either the frequency domain or the time domain. Furthermore, thinking of sampled functions as impulse modulated functions allows one to characterize linear computer operations on the sampled functions by transfer functions.

Chapter II derives the conditions of physical realizability for computer-program transfer functions, discusses stability conditions on these transfer functions, and presents procedures for plotting transfer loci.

Chapter III deals with techniques for realization of transfer functions with some attention to the approximation problem, while Chapter IV deals with frequency analysis of some numerical integration formulas.

CHAPTER I

DESCRIPTION OF THE SAMPLING PROCESS

1.1 Sampling a Continuous Function

A digital computer operates on numbers that represent samples of continuous signals taken at discrete instants of time. The time interval, T, between samples is a constant as shown in Figure 1.1, page 7 . In this case, the input to the computer is the sampled function, $\bar{i}$ (t). The computer senses the amplitude of each of these pulses (as a number) and operates on the number.

The purpose of this chapter is to describe the sampling process, to characterize it mathematically, to evaluate how well a continuous signal may be represented by its samples, and to show how and under what conditions a continuous signal may be recovered from its samples.

The mathematical model of the sampling process which will be derived later is very similar to actual physical processes. For example, assume that i (t) is the voltage across a pair of terminals of some network. How might it be sampled? The voltage may be sampled by connecting a condenser across the terminals, allowing a current flow to build up a charge on the condenser until the condenser voltage is equal to the terminal voltage, and then disconnecting the condenser. In order that the condenser voltage be equal to the terminal voltage at some instant of time, the sampling time should be as small as possible. It may be made very small, but not zero. The total charge on the condenser is the integral of the current

flowing over the time required to take the sample. Thus, as the sampling time decreases, the current intensity must increase. Physically this is how sampling might be done. Ideally, however, we wish to take the sample instantaneously or in zero time. Therefore, for ideal sampling in the above example the current flow must be infinite for zero time at each sampling instant. Thus, in the ideal case the charging current is an impulse whose area equals the amount of charge required to build up the condenser to the sampled value. Physically, ideal sampling is not possible, but the idea permits us to set up a model of sampling that can be treated mathematically.

1.2 Equivalent Mathematical Model of Ideal Sampling - Impulse Modulation

The ideal situation in the above example is to transfer to the plates of the condenser a portion of charge in zero time, or to "hit" the condenser with an impulse of current. The same end can be obtained if we modulate the voltage waveform with an infinite series of unit impulses separated by equal intervals, T, as shown in Figure 2. The area of any one of the modulated impulses equals the value of the input function at the corresponding instant of time. Thus, impulse modulation is analogous to the process of sampling. The samples of Figure 1.1[1] have finite height, zero width, and zero area; therefore, the sampled function does not have a Laplace transform. The impulses of Figure 1.2[1] have infinite height, zero width,

---

[1] The bar (–) over i(t) indicates the sampled functions.

[2] The circumflex (~) over i(t) indicates the impulse - modulated function.

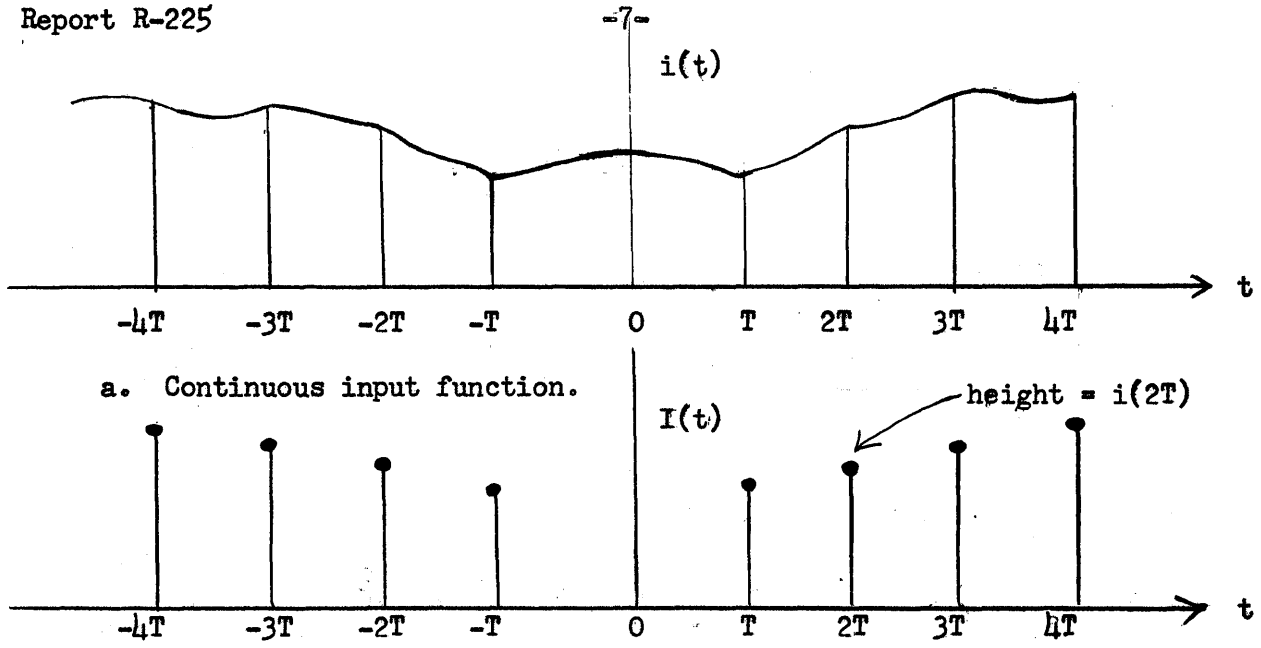
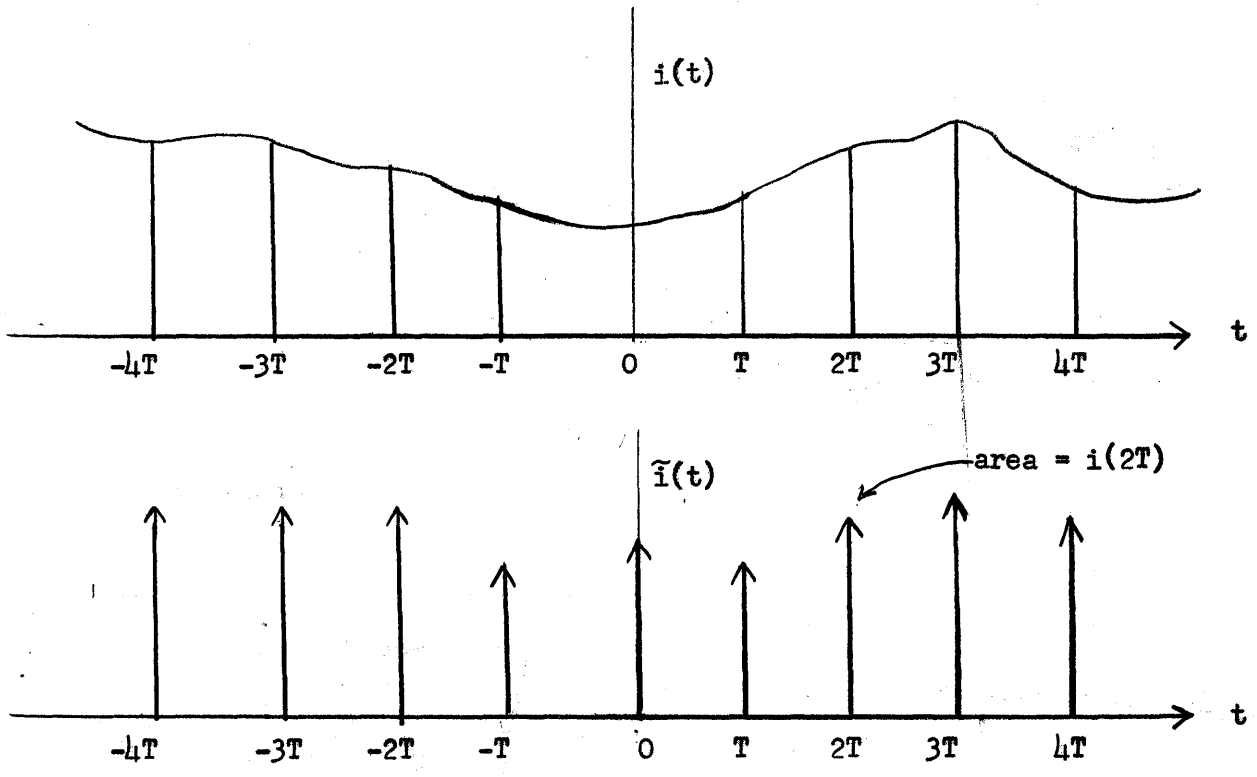


Figure 1.1 Relation between continuous and sampled functions



a. Continuous input function

Figure 1.2 Relation between continuous and impulse modulated functions.

but non-zero area; therefore, the impulse-modulated function does have a Laplace transform, which is why this mathematical model has been set up.

1.3 Use of Impulse Modulated Functions in the Analysis of Linear Digital Computer Programs

A digital computer operates on numbers that occur at discrete instants of time, i.e. it operates on samples of a continuous function. In the previous section it was shown that for the ideal case, sampling is equivalent to impulse modulation. If we think of the computer as "sensing" the amplitude of samples, we may just as easily think of it as "sensing" the area of impulses. With this extension or mathematical model, we may analyze computer programs by describing the input to the computer as impulses instead of samples. Since a sample does not have a Laplace transform, while an impulse does, the advantage of this extension is immediately obvious. In this mathematical model, both input and output are treated as impulses, and both have Laplace transforms. In conventional (continuous-data) systems, the transfer function is the ratio of the transform of the output to that of the input. Since both input and output of computer programs (when treated as impulse-modulated functions) have transforms, we may define the transfer function of a linear computer program as the ratio of the transform of its output to the transform of its input. In order to carry out this analysis, we must have a knowledge of some of the properties of impulse-modulated functions, or impulsed functions. The remainder of this chapter is devoted to a discussion of some of these more useful properties.

1.4 Laplace Transforms of Impulse-Modulated Functions[1]

Our analysis of computer programs is restricted to the cases in which the time interval between samples is a constant, T. Thus, the impulsed function can be expressed as the product of a continuous input function and an infinite string of unit impulses, the interval between impulses being T.

As the following derivation will show, the process of impulse modulation may be readily described in the frequency domain. Essentially, since the string of unit impulses (which is the carrier) has all harmonics of equal amplitude, the impulse modulated wave has an infinite number of side-bands rather than just the two which are present for a sinusoidal carrier. The method of the derivation is to make a Fourier analysis of the carrier and to associate each side-band of the impulse modulated wave with a Fourier component of the carrier. Let i(t) be the continuous input function and $\sum_{k=-\infty}^{\infty} \mu(t - kT)$ be the infinite string of unit impulses. [$\mu(x)$ = unit impulse occurring at $x = 0$.] Then the impulse-modulated input function is,

$$\tilde{i}(t) = i(t) \sum_{k=-\infty}^{\infty} \mu(t - kT). \qquad (1\text{-}1)$$

To find the Laplace transform of (1-1) let us first find the complex Fourier series of the string of unit impulses.

$$\sum_{k=-\infty}^{\infty} \mu(t - kT) = \sum_{m=-\infty}^{\infty} c_m \, e^{jm\Omega t} \qquad (1\text{-}2)$$

---

[1] A more complete derivation and discussion of the transforms of impulse modulated functions is given in Reference 2, page 3.

In (1-2), $\Omega = \frac{2\pi}{T}$. The c'_m s are the complex Fourier coefficients.

Solving for c_m in the usual manner we have,

$$c_m = \frac{1}{T} \int_{-T/2}^{T/2} \left[\sum_{k=-\infty}^{\infty} \mu(t - kT) \right] e^{-jm\Omega t} \, dt. \qquad (1\text{-}3)$$

By writing out a few terms of the series, (1-3) becomes,

$$c_m = \frac{1}{T} \int_{-T/2}^{T/2} \left[\ldots\ldots\ldots + \mu(t - T) + \mu(t) + \mu(t + T) \, x \ldots . \right] e^{-jm\Omega t} \, dt \qquad (1\text{-}4)$$

Within the range of the integral, the only term inside the bracket of the integrand that is non-zero is the term, $\mu(t)$. Thus (1-4) becomes,

$$c_m = \frac{1}{T} \int_{-T/2}^{T/2} \mu(t) \; e^{-jm\Omega t} \, dt. \qquad (1\text{-}5)$$

Because of the unit impulse in the integrand, the value of the integral is just $e^{-jm\Omega t}$ evaluated at $t = 0$, which is unity. Therefore,

$$c_m = \frac{1}{T} \qquad (1\text{-}6)$$

and the Fourier series of a string of unit impulses is,

$$\sum_{k=-\infty}^{\infty} \mu(t - kT) = \frac{1}{T} \sum_{m=-\infty}^{\infty} e^{jm\Omega t} \qquad (1\text{-}7)$$

Then the impulsed function becomes,

$$\tilde{i}(t) = \frac{i(t)}{T} \sum_{m=-\infty}^{\infty} e^{jm\Omega t} \qquad (1\text{-}8)$$

Now take the Laplace transform of the above equation.

$$\tilde{I}(s) = L\left[\tilde{i}(t)\right] = L\left[\frac{i(t)}{T} \sum_{m=-\infty}^{\infty} e^{jm\Omega t}\right] \qquad (1\text{-}9)$$

The indicated summation can be done after the transformation is made.

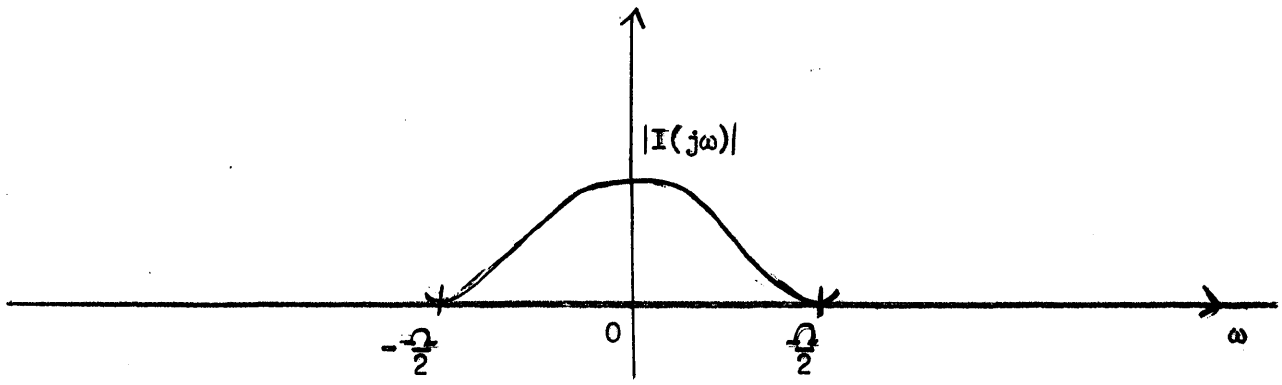
$$\tilde{I}(s) = \frac{1}{T} \sum_{m=-\infty}^{\infty} L\left[i(t)\ e^{jm\Omega t}\right] \qquad (1\text{-}10)$$

A fundamental theorem in Laplace transform theory leads directly to the following result:

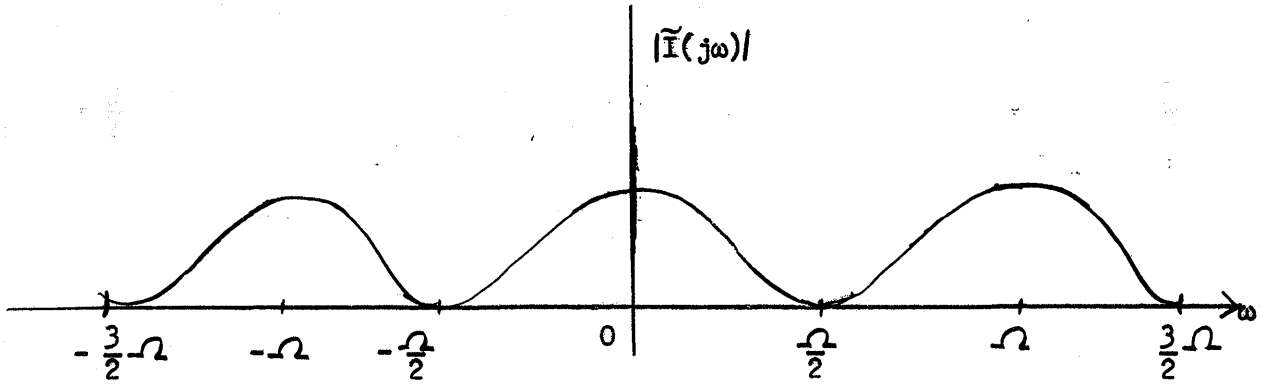
$$\tilde{I}(s) = \frac{1}{T} \sum_{m=-\infty}^{\infty} I(s + jm\Omega) \qquad (1\text{-}11)$$

Thus we see that the Laplace transform of an impulsed function is periodic having a repetition interval of $j\Omega$.

An important fact about $\tilde{I}(s)$ should be observed from (1-11). It is that there is a unique correspondence between $I(s)$ and $\tilde{I}(s)$ if and only if the frequency spectrum of $i(t)$, the continuous time function, lies in the range, $-\frac{\Omega}{2} < \omega < \frac{\Omega}{2}$. If the spectrum of $i(t)$ lies outside this range, $\tilde{I}(s)$ will specify the spectrum (in the range $-\frac{\Omega}{2} < \omega < \frac{\Omega}{2}$) of a continuous time function, but this time function will differ from the



A. Specturm of i(t)



B. Spectrum of $\tilde{i}(t)$

Figure 1.3 Unique Correspondence Between I(s) and $\tilde{I}(s)$

original time function. Thus, there is a limitation of bandwidth caused by sampling. Figure 1.3 illustrates the case of unique correspondence, and Figure 1.4, the case in which the spectrum of i(t) is too wide.

As given by (1-11), $\tilde{I}(s)$ consists of an infinite number of terms; however, an infinite series is difficult to handle, and it is desirable to have a closed form expression for $\tilde{I}(s)$. This can be obtained from the partial fraction expansion of I(s). Consider a typical term, $\frac{K_i}{s - s_i}$, of the partial fraction expansion of I(s). Referring to (1-11) we see that corresponding to this typical term, $\tilde{I}(s)$ will have a typical series of terms of the form,

$$\frac{K_i}{T} \sum_{k=-\infty}^{\infty} \frac{1}{s - s_i + jk\Omega} .$$

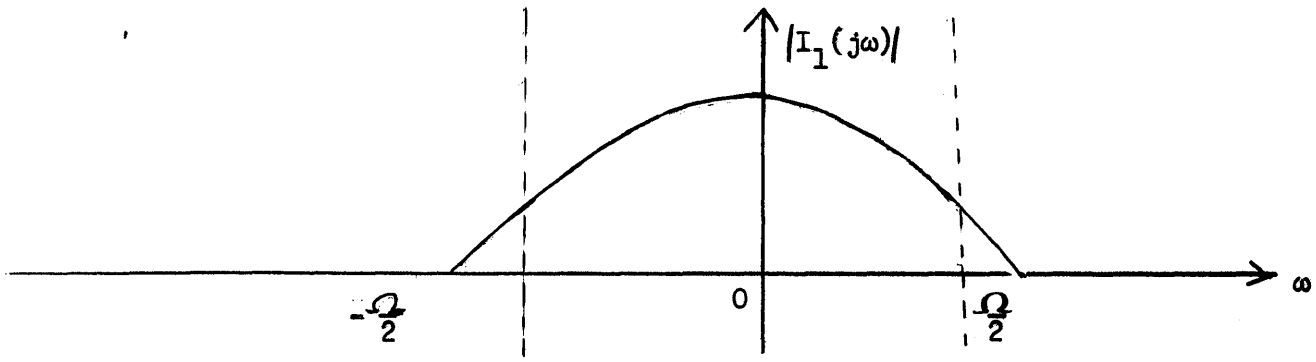
Thus we see that the pole at $s = s_i$ is repeated an infinite number of times at intervals of $j\Omega$, the line through these poles being parallel to the imaginary axis in the s - plane.

A closed form equivalent of the above typical series can be obtained by a change of variable in the following equation.[1]

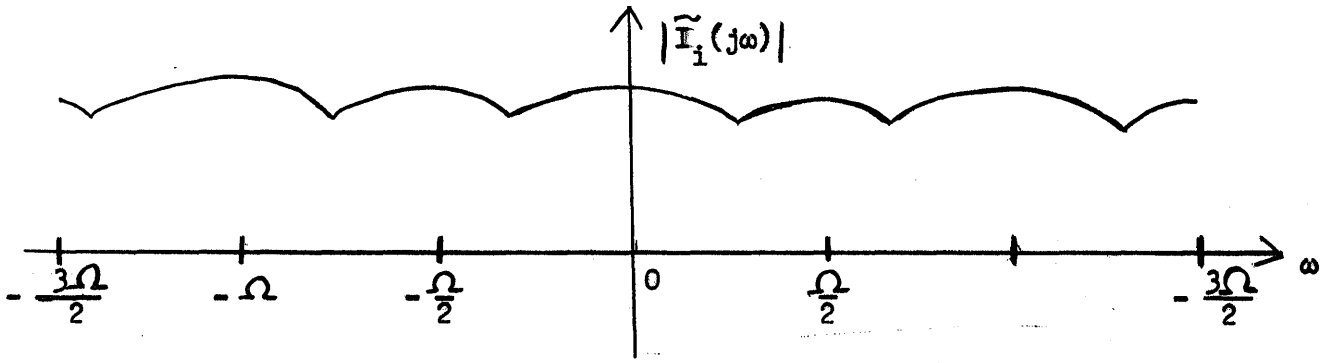
$$\pi z \cot \pi z = 1 + 2 z^2 \sum_{n=1}^{\infty} \frac{1}{z^2 - n^2} \qquad (1\text{-}12)$$

---

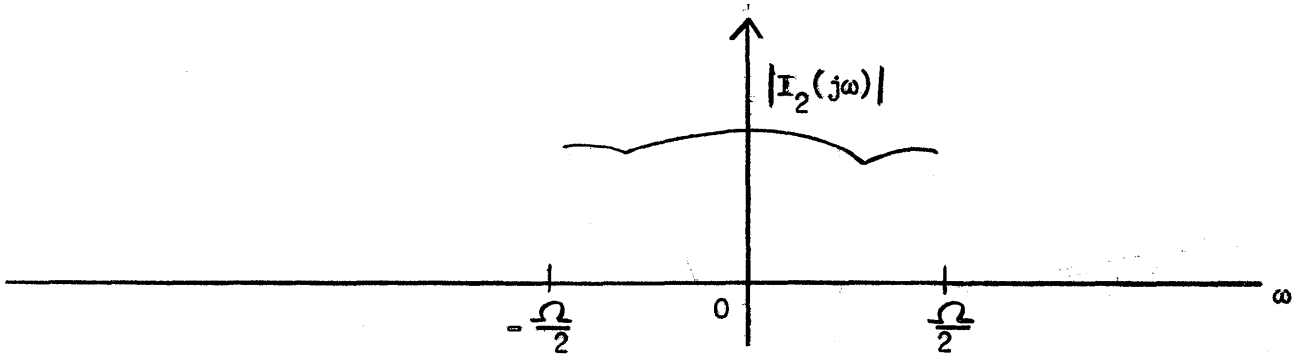
[1] Knopp, "Theory and Application of Infinite Series", New York, 1948, p. 419



A. Spectrum of $i_1(t)$ whose spectrum is too wide for the sampling rate.



B. Spectrum of $\tilde{i}_1(t)$.



C. Spectrum of $i_2(t)$ that would produce the same sampled function as (B).

Figure 1.4 Illustration of Bandwidth Limitation Caused by Sampling.

Divide each side of (1-12) by z, and make the change of variable, $z = j\frac{\omega}{\Omega}$.

$$\pi \cot j\frac{\pi\omega}{\Omega} = \frac{\Omega}{j\omega} - j2\frac{\omega}{\Omega}\sum_{n=1}^{\infty}\frac{1}{\frac{\omega^2}{\Omega^2}+n^2} \qquad (1\text{-}13)$$

Multiply both sides of (1-13) by j/Ω and obtain,

$$j\frac{\pi}{\Omega}\cot j\frac{\pi\omega}{\Omega} = \frac{\pi}{\Omega}\coth\frac{\pi\omega}{\Omega} = \frac{1}{\omega} + \sum_{n=1}^{\infty}\frac{2\omega}{\omega^2 + n^2\Omega^2} \qquad (1\text{-}14)$$

The infinite series of poles of $\tilde{I}(s)$ corresponding to a pole of $I(s)$ at s_i can be put into a form that is identical to the right-hand member of (1-14) as follows: separate the term for $k = 0$.

$$\sum_{k=-\infty}^{\infty}\frac{1}{s - s_i + jk\Omega} = \frac{1}{s - s_i} + \sum_{k=1}^{\infty}\left[\frac{1}{s-s_i + jk\Omega} + \frac{1}{s-s_i - jk\Omega}\right] \qquad (1\text{-}15)$$

Combine the two terms in the summation.

$$\sum_{k=-\infty}^{\infty}\frac{1}{s - s_i + jk\Omega} = \frac{1}{s - s_i} + \sum_{k=1}^{\infty}\frac{2(s - s_i)}{(s - s_i)^2 + k^2\Omega^2} \qquad (1\text{-}16)$$

A comparison of (1-14) and (1-16) shows that,

$$\sum_{k=-\infty}^{\infty}\frac{1}{s - s_i + jk\Omega} = \frac{\pi}{\Omega}\coth\frac{\pi(s - s_i)}{\Omega} \qquad (1\text{-}17)$$

Thus we have the following closed form equivalent of the typical series of $I(s)$,

$$\frac{K_i}{T}\sum_{k=-\infty}^{\infty}\frac{1}{s - s_i + jk\Omega} = \frac{K_i}{2}\coth\frac{T}{2}(s - s_i), \qquad (1\text{-}18)$$

for a pole of I(s) at $s = s_i$. Therefore, corresponding to the partial fraction expansions of I(s), we have the following series for $\tilde{I}(s)$.

$$\tilde{I}(s) = \frac{1}{2} \sum_{i=1}^{n} K_i \coth \frac{T}{2} (s - s_i) ,$$

where "n" is the total number of poles of I(s), taking into account multiple poles.

Let us now investigate the limitations on the positions of the poles of I(s) due to sampling. Consider an infinite strip of width Ω in the s-plane and parallel to the real axis as shown in Figure 1.5. Assume that all the poles of I(s) lie within this strip and in the left half plane (LHP). Thus, $\tilde{I}(s)$ has

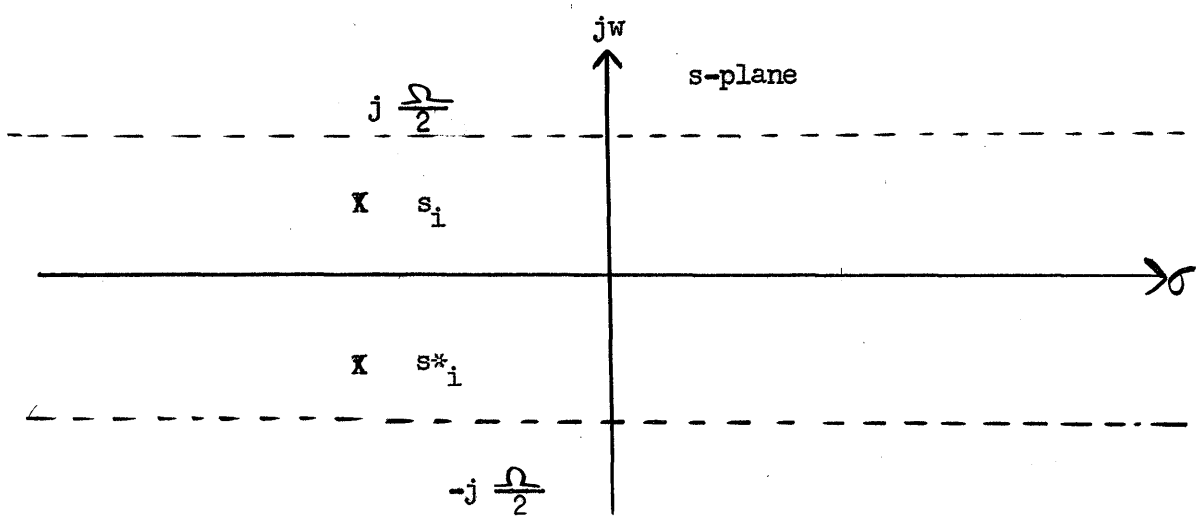


Figure 1.5 Infinite Strip Containing Poles of I(s)

poles at these points plus poles at points shifted from the s_i's and s^*_i's (* means conjugate) by the distance $\pm$ jk Ω. Since I(s) has poles only in the strips being considered, there is a one-to-one correspondence between the poles of I(s) and $\tilde{I}(s)$ that lie in the same strip. However, if I(s) had poles outside this strip, there no longer would be this one-to-one correspondence.

CHAPTER II

TRANSFER FUNCTION OF COMPUTER PROGRAMS - REALIZABILITY AND STABILITY

Using the properties of impulse-modulated functions given in Chapter I, we are now ready to investigate transfer functions of computer programs. Our interest in program transfer functions is much more than academic. The transfer function describes the program completely and with it we can analyze and synthesize control systems employing digital computers by conventional frequency domain methods.

In this chapter a linear digital computer program is defined in terms of the mathematical model of sampling set up in Chapter I, its transfer function is derived, and methods for determining the realizability and stability of transfer functions are given. Several examples of stability determination are also presented.

2.1 Transfer Function of Linear, Real-Time, Digital Computer Program

As pointed out in Chapter I, the input to a digital computer may be assumed to be an impulsed function, for purposes of mathematical analysis. A linear program of a digital computer operating in real time is one in which the present output is a linear function of the present and past inputs and the past outputs. The general form of this relation is,

$$\tilde{o}(t) = \sum_{k=0}^{m} a_k \tilde{i}(t - kT) - \sum_{k=1}^{n} b_k \tilde{o}(t - kT), \qquad (2\text{-}1)$$

in which all a_k's and b_k's are real, and T is the time between samples. The time required for the computation must be less than T if each calculation is to be completed before the next input arrives.

Taking the Laplace transform of (2-1) yields,

$$\tilde{O}(s) = \tilde{I}(s) \sum_{k=0}^{m} a_k e^{-ksT} - \tilde{O}(s) \sum_{k=1}^{m} b_k e^{-ksT}. \qquad (2\text{-}2)$$

As in continuous data systems, we will define the transfer function of a computer program as the ratio of the transform of the output to that of the input. Let W(s) be the transfer function of a computer program; then,

$$W(s) = \frac{\tilde{O}(s)}{\tilde{I}(s)}. \qquad (2\text{-}3)$$

Solving for $\tilde{O}(s)/\tilde{I}(s)$ from (2-2) we obtain,

$$W(s) = \frac{\tilde{O}(s)}{\tilde{I}(s)} = \frac{\sum_{k=0}^{m} a_k e^{-ksT}}{1 + \sum_{k=1}^{n} b_k e^{-ksT}} \qquad (2\text{-}4)$$

as the transfer function of a linear, real-time, digital computer program. With the understanding that $b_o = 1$, (2-4) becomes,

$$W(s) = \frac{\sum_{k=0}^{m} a_k e^{-ksT}}{\sum_{k=0}^{n} b_k e^{-ksT}}. \qquad (2\text{-}5)$$

The inverse steps from (2-5) to (2-1) are unique; therefore, (2-5) is the general form of the transfer function of a realizable, linear, digital computer program. Thus, to be realizable, the transfer function of a linear, digital computer program must be expressible as the ratio of two polynomials in e^{-sT}. The criteria for stability will be discussed in a later section.

It has already been shown that the Laplace transform, $\tilde{I}(s)$, of the impulsed input function is periodic of period $\frac{\Omega}{}$, as seen in (1-11). By showing that $W(s)$ is also periodic with the same period, we can prove that $\tilde{O}(s)$ is also periodic of period Ω. A typical term of either numerator or denominator of $W(s)$ contains e^{-ksT}. For $s \longrightarrow s + jm\Omega$ (m is a positive or negative integer), the typical term becomes,

$$e^{-k(s + jm\Omega)T} = e^{-ksT} e^{-jkm\Omega T}.$$

As $T\Omega = 2\pi$ and k and m are integers, the second factor is,

$$e^{-jkm\Omega T} = e^{-j2\pi km} = 1.$$

Hence, $e^{-k(s + jm\Omega)T} = e^{-ksT}$.

Therefore, the terms of the numerator and denominator of $W(s)$ are periodic of period Ω, and so is $W(s)$. In equation form this means, $W(s) = W(s + jm\Omega)$, for m a positive or negative integer. The product of two periodic functions is also periodic. Since $\tilde{O}(s) = W(s)\,\tilde{I}(s)$, $\tilde{O}(s)$ is also periodic of period Ω, as indeed it should because the computer output is also sampled.

Since all the coefficients of (2-5) are real, it is readily seen that $W(s) = W(s^*)^*$, in which the asterisk means conjugate. For real frequencies this becomes $W(j\omega) = W(-j\omega)^*$. This fact together with the periodicity of $W(s)$ tells us that $W(s)$ is completely specified for all s if it is defined over the range, $0 \leq \omega \leq \frac{\Omega}{2}$.

Summary: In order to be realizable, the transfer function of a linear digital computer program must be expressible as the ratio of polynomials in e^{-sT}. $W(s)$ is periodic of period Ω; i.e., $W(s) = W(s + jm\Omega)$. Specification of $W(s)$ over the range, $0 \leq \omega \leq \frac{\Omega}{2}$, completely determines $W(s)$.

2.2 Stability of Programs

We have expressed the transfer function of computer programs as a function of the complex frequency "s"; therefore, the same methods of investigating stability as used in network analysis and servomechanisms are applicable. The general necessary and sufficient criterion for stability of a unit is that its transfer function have no poles in the right half s-plane (RHP) or multiple poles on the jω-axis. In network analysis the frequency-domain method used to study stability is to map a contour enclosing the right half of the s-plane (the contour is usually the jω-axis and an infinite semicircle) into the W-plane. Because of the transcendental nature of the transfer function of a realizable computer program, the mapping contour in the s-plane must be modified.

As we have shown before, the transfer function of the computer program is,

$$W(s) = \frac{P(s)}{Q(s)} = \frac{\sum_{k=0}^{m} a_k e^{-ksT}}{\sum_{k=0}^{n} b_k e^{-ksT}} \qquad (2\text{-}6)$$

in which P(s) is the numerator and Q(s), the denominator of W(s); and it is assumed that P(s) and Q(s) have no common factor. Both P(s) and Q(s) are entire transcendental functions having as their only singularity an essential singularity at infinity.† Hence, we see that the only singularities of W(s) in the finite s-plane are poles, and these poles occur at the zeros of Q(s). Our stability criterion is that there be no poles of W(s) in the RHP and only simple poles on the imaginary axis. Therefore, in order for

† For a further discussion of entire transcendental functions, consult Knopp, "Theory of Functions," or Guillemin, "The Mathematics of Circuit Analysis."

the program to be stable, Q(s) must have no zeros in the RHP and only simple zeros on the jω-axis. To investigate the possibility of Q(s) having zeros in the RHP or on the imaginary axis, we may take advantage of the periodicity of Q(s). In proving that W(s) is periodic, it was shown that e^{-ksT} is periodic property. Therefore, if Q(s) has a zero in the RHP, it must have one in the semi-inifinite strip shown in Fig. 2.1.

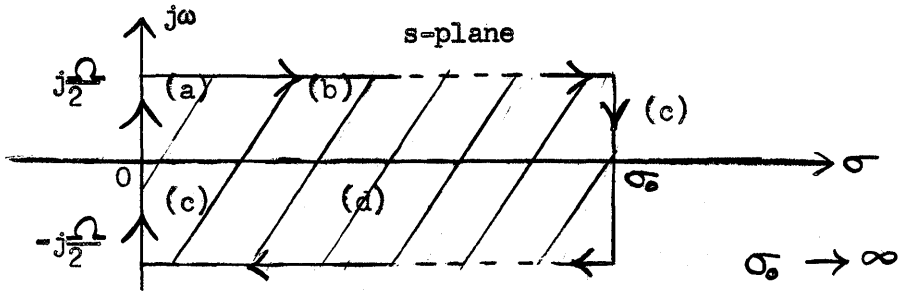


Figure 2.1 Semi-inifinite strip of s-plane that must have a zero of Q(s) if Q(s) has any zeros in the RHP.

Consider the map of the contour of Fig. 2.1 into the e^{-sT} plane. Let us begin the path at the origin in the s-plane and encircle the strip in a clockwise direction, corresponding to increasing frequency. It is readily understood that corresponding path and enclosed region in the e^{-sT} plane is shown in Fig. 2.2. The origin of the s-plane maps into the point (1,0) in the e^{-sT} plane. The corresponding sections of the path are marked by small letters on both contours. In Fig. 2.2 we see that the paths (b) and (d) cancel leaving the annular ring as the region conformal to the strip of the s-plane that is under consideration. As σ_o (of Fig. 2.1) approaches ∞, the radius of the circular path (c) in Fig. 2.2 approaches zero. Thus, the conformal map of the indicated strip consists of two separate contours: one, a unit circle centered at the origin

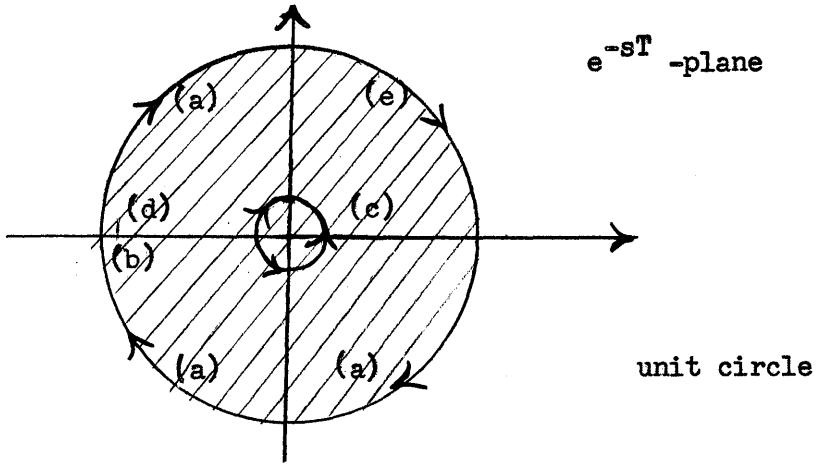


Figure 2.2 Conformal map of the semi-inifinite strip of Fig. 2.1 into the e^{-st} plane.

and the other an infinitesimally small circle that excludes the origin in this particular case. Only a slight extension of the foregoing procedure is required to determine the map of powers of e^{-sT}. The map of e^{-ksT} will appear like that of e^{-sT} except that each of the two separate paths will be traversed "k" times; the region excluded by the infinitesimally small circle will be that at the origin. Thus we see that the map of this semi-infinite strip of the s-plane is effective in handling the essential singularity of Q(s) at ∞.

Now, consider the conformal map of the semi-infinite strip of Fig. 2.1 into the Q-plane. Remembering that $Q(s) = 1+b_1 e^{-sT} + b_2 e^{-2sT} + \ldots + b_n e^{-nsT}$, we see that the map of this strip into the Q-plane will exclude the point (1,0), (the map of each term except the first excludes the origin). This eliminates the need for mapping path (c). Moreover, since the paths (b) and (d) cancel, we need to plot only the paths (a) and (e). In other words the only part of the s-plane contour that we need to plot in order to determine the locus of Q(s) is the part of the contour that lies on the imaginary axis. This contour in the Q-plane will encircle the origin z-N times in the counterclockwise direction, where

z is the number of zeros and N is the number of poles of Q(s) (taking into account their multiplicity) in this strip of the s-plane. It has already been pointed out that Q(s) is an entire transcendental function and, therefore, has no poles in this strip. So, N = 0, and the contour in the Q-plane will encircle the origin Z times (clockwise is to be understood). The condition for stability of W(s) states that Q(s) must not have any zeros in the RHP or any multiple order zeros on the imaginary axis. Therefore, the map in the Q-plane must not enclose the origin; Z must be zero. If Q(s) has zeros on the imaginary axis, the Q-plane locus will pass through the origin. In this case, we must determine the order of the zero. The following method can be used: Assume that the locus in the Q-plane passes through the origin for $s = j\omega_i$. Then Q(s) must contain the factor $(e^{-sT} - e^{-j\omega_i T})^n$ where n is the order of the zero. Divide Q(s) by $(e^{-sT} - e^{-j\omega_i T})^2$. If there is no remainder, the zero is of higher order than the first and the program will be unstable.

In addition to determining the stability of programs, conformal maps give an indication of the degree of stability or instability and an approximate value of the frequency at which the program is or may become unstable. The amount by which the locus in the Q-plane misses encircling the origin gives a measure of the stability of the program. The farther the locus is from the origin, the more stable or convergent the program. The frequency corresponding to the point on the Q-plane locus nearest the origin is approximately the frequency at which the program is or may become unstable, or at which it will oscillate in a damped fashion.

In addition to expressing the program transfer function as a function of "s" we may also write it as a function of e^{-sT}. Make the change of variable, $z = e^{-sT}$. Then we may define a new function,

$$V(z) = \frac{N(z)}{D(z)} = \frac{\sum_{k=0}^{m} a_k z^k}{\sum_{k=0}^{n} b_k z^k} \tag{2-7}$$

$$V(z) = \frac{a_o + a_1 z + a_2 z^2 + \cdots\cdots\cdots + a_m z^m}{1 + b_1 z + b_2 z^2 + \cdots\cdots\cdots + b_n z^n} \tag{2-8}$$

It is readily seen that the right half of the s-plane maps into the inside of a unit circle centered at the origin in the z-plane. The imaginary axis of the s-plane becomes the unit circle in the z-plane (see Fig. 2.3).

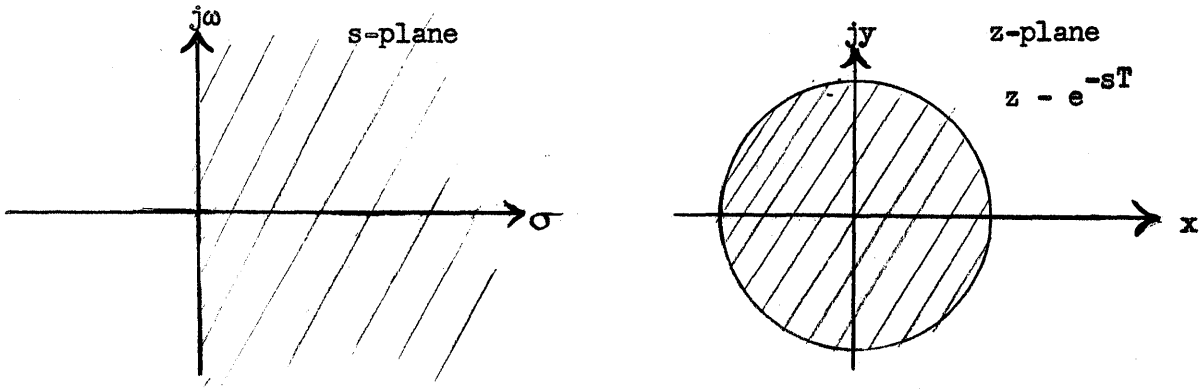


Figure 2.3 Map of right half of s-plane into z-plane

Therefore, if the program is to be stable, all the zeros of D(z) must lie outside the unit circle except that single order zeros may occur on the unit circle. In other words, the magnitude of the roots of D(z) must be greater than or equal to unity, and the roots of unity magnitude must be simple.

Summary: To test for the stability of a program, map the semi-infinite strip of Fig. 2.1 into the Q-plane [Q(s) is the denominator of the program transfer function]. If the locus in the Q-plane does not enclose the origin, the program is stable or convergent. If the locus passes through the origin, Q(s) has a zero and the order of this zero must be determined. If the zero is of first order, the program is stable; otherwise, unstable. An alternate method is: Make the change of variable, $z = e^{-sT}$, and find the magnitude of the z-roots. If each root has either a magnitude greater than unity or equal to unity and is simple, the program is stable or convergent; otherwise unstable or divergent.

2.3 Loci of Q(s)

In the previous section it was demonstrated that the stability of a program can be determined by mapping the contour enclosing the semi-infinite strip of Fig. 2.1 into the Q-plane. It was also shown that the only part of this contour that we need to plot is that on the imaginary axis. The paths (b) and (d) cancel and the path (c) excludes the point (1,0) in the Q-plane. Hence, we are interested in the properties of $Q(j\omega)$ and its locus in the range, $-\frac{\Omega}{2} \leq \omega \leq \frac{\Omega}{2}$.

$Q(j\omega)$ has several properties that are helpful in determining its locus.

(1) $Q(j\omega)$ is periodic of period Ω. This was proved in the previous section.

(2) $Q(j\omega) = Q(-j\omega)^*$. This property follows directly from the fact that $Q(j\omega)$ is a polynomial in $e^{-j\omega T}$, and as a consequence, the locus of $Q(j\omega)$ must be symmetrical about the real axis.

(3) At $\omega = 0$ and $\pm\frac{\Omega}{2}$ $Q(j\omega)$ is real and its locus at these two points crosses the real axis either normally or tangentially. This statement is proved and elaborated upon in Appendix A.

As a consequence of the first two properties, the locus of $Q(j\omega)$ for $0 \leq \omega \leq \frac{\Omega}{2}$ completely determines the locus in the Q-plane. The locus for $-\frac{\Omega}{2} \leq \omega \leq 0$ is just the mirror of that for the positive values of ω. Thus the first two properties result in a substantial reduction in the amount of work required to plot the locus of $Q(j\omega)$. The third property enables one to determine accurately the shape of the locus in the neighborhood of $\omega = o$ and $\pm\frac{\Omega}{2}$.

Several methods may be used to determine the locus of $Q(j\omega)$. Three of these are: (1) add the loci of the individual terms of the polynomial (each locus is a circle) to obtain that of $Q(j\omega)$; (2) factor $Q(j\omega)$ and multiply the loci of the factors; and (3) express $Q(j\omega)$ in the form $R(\omega)/\underline{\phi(\omega)}$ and make a point by point plot. The method that is best to use depends on the particular $Q(j\omega)$. However, it is to be expected that the extra analytic work required in methods (2) and (3) will result in less graphical work and more accurate loci. None of the methods will be discussed, but they will be illustrated.

Let us now consider several examples of loci of $Q(j\omega)$.

1. Let $$Q(s) = 1 - 0.8\, e^{-sT} + 0.3\, e^{-2sT} \qquad (2\text{-}9)$$

Take the derivative with respect to s.

$$\frac{dQ}{ds} = 0.8Te^{-sT} - 0.6Te^{-sT} \qquad \text{(a)}$$

$$\frac{dQ}{ds} = 0.2Te^{-sT}(4 - 3e^{-sT}) \qquad \text{(b)} \qquad (2\text{-}10)$$

For neither s = 0 nor $\pm j\frac{\Omega}{2}$ is the derivative zero, so the locus is normal to the real axis at both points. Fig. 2.4 (a) shows the locus of $Q(j\omega)$ and how it was obtained (for the point $\omega t = \frac{3\pi}{4}$) from the loci of the individual terms. The locus for negative values of ω is shown by the dashed curve, since it may be obtained from the other half of the locus. After this, only the locus for positive ω will be drawn. The locus does not enclose the origin; therefore, a program whose transfer function has the denominator, $1 - 0.8e^{-sT} + 0.3e^{-2sT}$, will be stable.

$$2.\ \text{Let } Q(s) = 1 - 0.8e^{-sT} + 0.4e^{-2sT} \tag{2-11}$$

Take the derivative with respect to s.

$$\begin{aligned} \frac{dQ}{ds} &= 0.8Te^{-sT} - 0.8Te^{-2sT} && \text{(a)} \\ &= 0.8Te^{-sT}\left(1 - e^{-sT}\right). && \text{(b)} \end{aligned} \tag{2-12}$$

For s = 0, the derivative is zero, Q(s) has a saddle point here. Q(s) can be rewritten as,

$$Q(s) = 0.6 + 0.4\left(1 - e^{-sT}\right)^2 \tag{2-13}$$

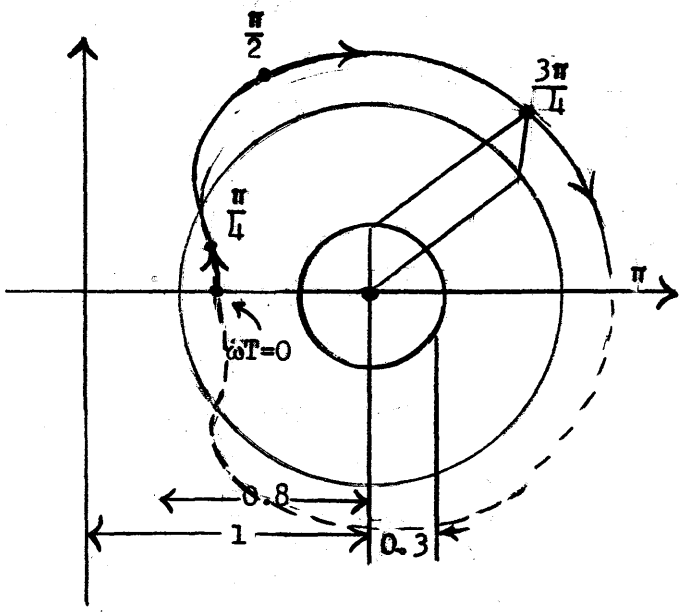
which brings the saddle point into evidence. In this case p = 2, so at $\omega = 0$. the locus is tangent to the real axis. At $s = \pm j\frac{\Omega}{2}$, $\frac{dQ}{ds} \neq 0$, so the locus is normal to the real axis at this point. Fig. 2.4 (b) shows the resulting locus. It does not enclose or pass through the origin; therefore, this Q(s) will lead to a stable program.

$$3.\ \text{Let } Q(s) = 1 - 0.8e^{-sT} + 0.5e^{-2sT} \tag{2-14}$$

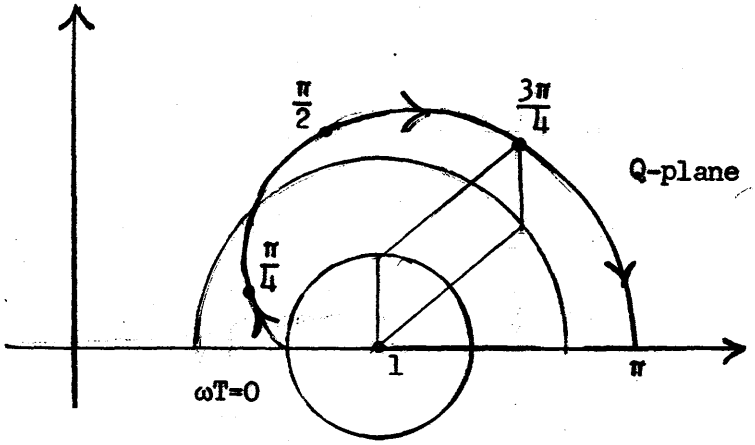
Take the derivative with respect to s.

$$\begin{aligned} \frac{dQ}{ds} &= 0.8Te^{-sT} - Te^{-2sT} && \text{(a)} \\ &= 0.2Te^{-sT}\left(4 - 5e^{-sT}\right) && \text{(b)} \end{aligned} \tag{2-15}$$

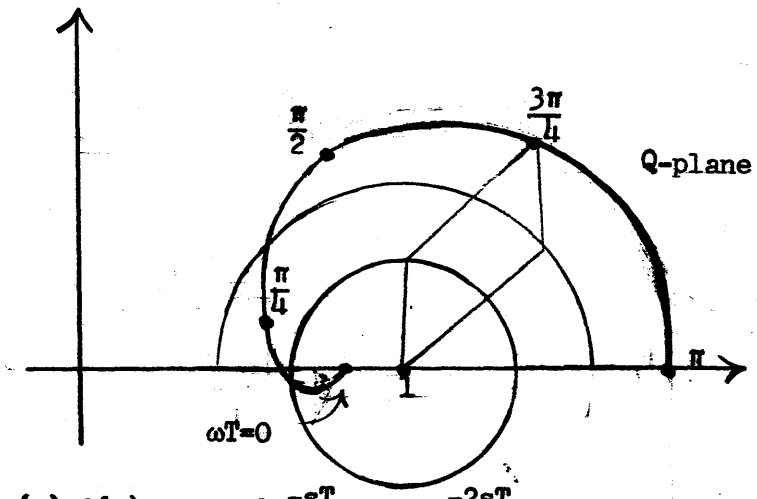
For neither $s = 0$ nor $\pm j\frac{\rho}{2}$ is the derivative zero, so the locus of $Q(j\omega)$ is perpendicular to the real axis at both points. The locus (as shown in Fig. 2.4 (c)) does not enclose the origin; therefore, this $Q(s)$ is the denominator of a stable program transfer function.



(a) $Q(s) = 1-0.8e^{-sT} + 0.3e^{-2sT}$



(b) $Q(s) = 1-0.8e^{-sT} + 0.4e^{-2sT}$



(c) $Q(s) = 1-0.8e^{-sT} + 0.5e^{-2sT}$

Figure 2.4 Loci of some typical Q(s)

CHAPTER III

ANALYSIS AND SYNTHESIS OF LINEAR, DIGITAL COMPUTER PROGRAMS IN THE FREQUENCY DOMAIN

In the first part of this chapter the analysis of transfer functions is dealt with by expanding the transfer function into partial fractions. Next, programs are realized from transfer functions by three methods: direct programming, cascade programming, and parallel programming; and the storage and time requirements of each are presented. In the last part of the chapter a short, general discussion of synthesis is given, and one possible synthesis procedure is illustrated by the synthesis of a program for differentiation.

3.1 Response of Programs at Real Frequencies

The input to a computer has a certain frequency spectrum, and in order to analyze the action of a computer program on this input function, we need to have a knowledge of the frequency response of the program. Thus, we are interested in the locus of $W(j\omega)$, the map of the $j\omega$-axis of the s-plane into the W-plane. A familiarity with the frequency characteristics of the simple transfer functions is essential for the understanding of the possibilities and limitations of more complicated ones.

In many cases the desired locus of the transfer function of a digital computer program is given, and the problem is to approximate this locus by that of a realizable program; i.e., by a ratio of polynomials in e^{-sT}. A study of the loci of typical terms of $W(j\omega)$ is helpful in making this approximation.

3.2 Analysis of Building Blocks of Transfer Functions

As we have seen, the transfer function of a linear, digital computer program is most generally expressed as the ratio of polynomials in e^{-sT}.

$$W(s) = \frac{O(s)}{I(s)} = \frac{a_0 + a_1 e^{-sT} + a_2 e^{-2sT} + \cdots\cdots + a_m e^{-msT}}{1 + b_1 e^{-sT} + b_2 e^{-2sT} + \cdots\cdots + a_n e^{-nsT}} \qquad (3\text{-}1)$$

A partial fraction expansion of W(s) can be made, and we may call the individual terms of the expansion the basic building blocks of a program transfer function. In general W(s) may be broken up into a polynomial plus first and second degree partial fractions (from the real and conjugate complex roots of the denominator, respectively).

In analyzing computer programs in the frequency domain [finding the locus of $W(j\omega)$] several methods can be used. Two of these are: (1) Find the loci of the numerator and denominator polynomials and then divide; (2) expand W(s) into partial fractions, find the locus of each term of the expansion, and add the resultant loci. In most cases the first method is easier to use, but the second is included here because of its connection to the synthesis of program transfer functions (approximation of a desired locus by a sum of the basic building blocks). A familiarity with some of the possible loci of polynomials and first and second degree partial fractions is an aid in the synthesis procedure.

The loci of second degree polynomials have already been discussed, and the locus of a fourth degree polynomial will be illustrated in connection with polynomial building blocks. Since there is only a short step from the loci of polynomials to the locus of a transfer function, the first method of finding the locus of $W(j\omega)$ will not be discussed.

For use of the second method, we will investigate the loci of typical terms of the partial fraction expansion of $W(j\omega)$.

3.21 Polynomials

A typical polynomial transfer function is of the form,

$$W(s) = \sum_{k=0}^{r} c_k e^{-ksT} \qquad (3\text{-}2)$$

For $s = j\omega$, the locus of each term of the polynomial is a circle. For Q(s), the constant term is unity, but for polynomial building blocks, the constant term may have any real value. In the previous chapter, three examples of the locus of second order polynomials in e^{-sT} are given, so now let us find the locus of a fourth order polynomial. Let,

$$W(s) = \frac{1}{15}\left(13 + 4e^{-sT} - 3e^{-2sT} + 2e^{-3sT} - e^{-4sT}\right) \qquad (3\text{-}3)$$

First examine the function for saddle points.

$$\frac{dW}{ds} = \frac{1}{15}\left(-4Te^{-sT} + 6Te^{-2sT} - 6Te^{-3sT} + 4Te^{-4sT}\right) \qquad (3\text{-}4)$$

$$\frac{dW}{ds} = -\frac{2T}{15} e^{-sT}\left(2 - 3e^{-sT} + 3e^{-2sT} - 2e^{-3sT}\right). \qquad (3\text{-}5)$$

The derivative is zero for $s = 0$, therefore W(s) has a saddle point there. W(s) can be written in the form

$$W(s) = 1 - \frac{1}{15}\left(2 + e^{-2sT}\right)\left(1 - e^{-sT}\right)^2, \qquad (3\text{-}6)$$

which brings the saddle point at $s = 0$ into evidence. The saddle point is of first order; therefore, the locus is tangent to the real axis at $s = 0$. $W(j\omega)$ is

$$W(j\omega) = 1 - \frac{1}{15}\left(2 + e^{-j2\omega T}\right)\left(1 - e^{-j\omega T}\right)^2 \qquad (3\text{-}7)$$

In this case it is easier to determine the locus of $W(j\omega)$ by plotting the second term of (3-7) and shifting the origin one unit to the left. A convenient way to find the locus of the second term is to let

$$-\left(2 + e^{-j2\omega T}\right)\left(1 - e^{-j\omega T}\right)^2 = Re^{j\phi}, \qquad (3\text{-}8)$$

where both R and ϕ are functions of ωT. Some trigonometric manipulation yields,

$$R = 2(1 - \cos \omega T) \sqrt{5 + 4 \cos 2\omega T} \tag{3-9}$$

and

$$\phi = \tan^{-1}\left[\frac{\tan \omega T(5 + \tan^2 \omega T)}{\tan^2 \omega T - 3}\right] \tag{3-10}$$

The resulting locus of $W(j\omega)$ is shown in Fig. 3.1.

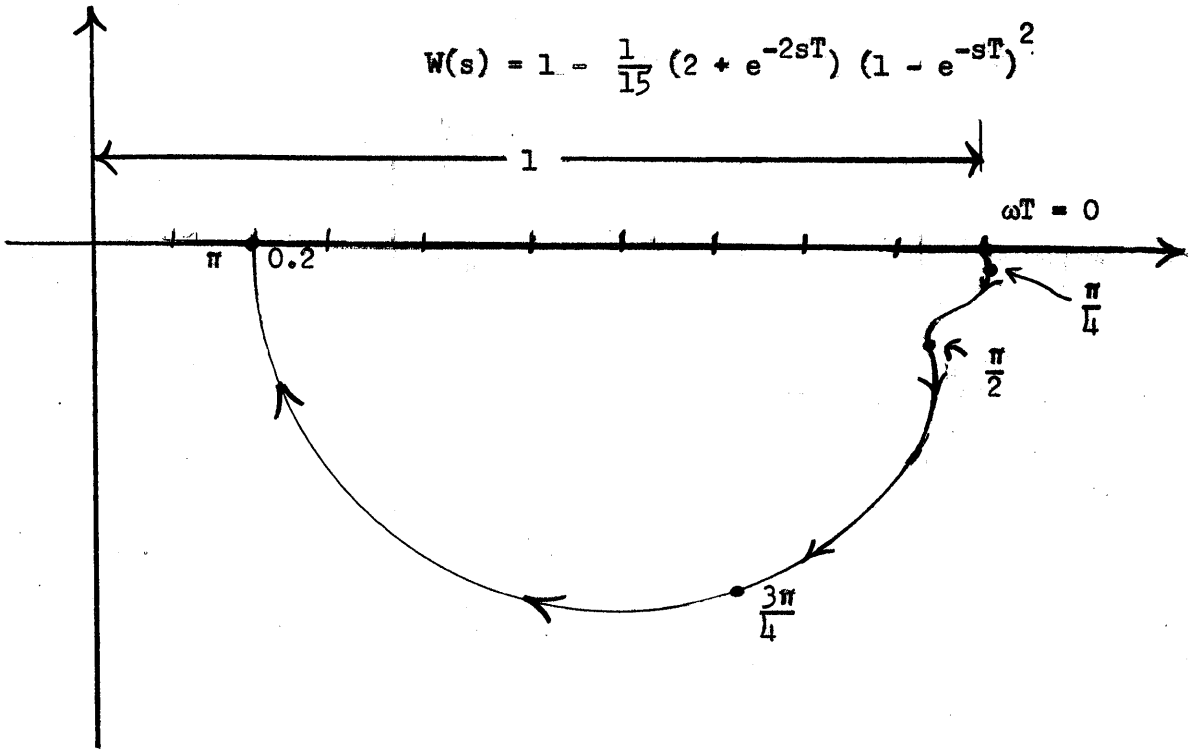


Figure 3.1 Locus of a Fourth-Degree Polynomial Transfer Function

3.22 First-Degree Partial Fractions (real roots)

In the partial fraction expansion of a rational function e^{-sT}, a typical term has the form,

$$W_1(s) = \frac{\alpha}{1 + \beta e^{-sT}} . \tag{3-11}$$

If the constants α and β are real, then (3-11) can be considered a basic building block. In this section we will consider the case in which α and β are real. The case of complex constants is considered in the next section.

First, we may set $\alpha = 1$ because it is merely a scale factor. Second, the magnitude of β must not be greater than unity for $W_1(s)$ is then unstable. If $|\beta| \leq 1$, the typical term is stable.

To determine the loci of typical terms of the form (3-11), we need only apply some of the rules of the loci of complex functions. First, note that the locus of $1 + \beta e^{-j\omega T}$ is a circle of radius $|\beta|$ centered at the point (1.0). To find the locus of $W_1(j\omega)$, the inverse of a circle must be found. If $|\beta| = 1$, the circle $(1 + e^{-j\omega T})$ passes through the origin, and its inverse is a straight line parallel to the imaginary axis. If $|\beta| < 1$, the circle does not pass through the origin, and its inverse is another circle. The loci of two stable first-degree partial fractions are shown in Fig. 3.2.

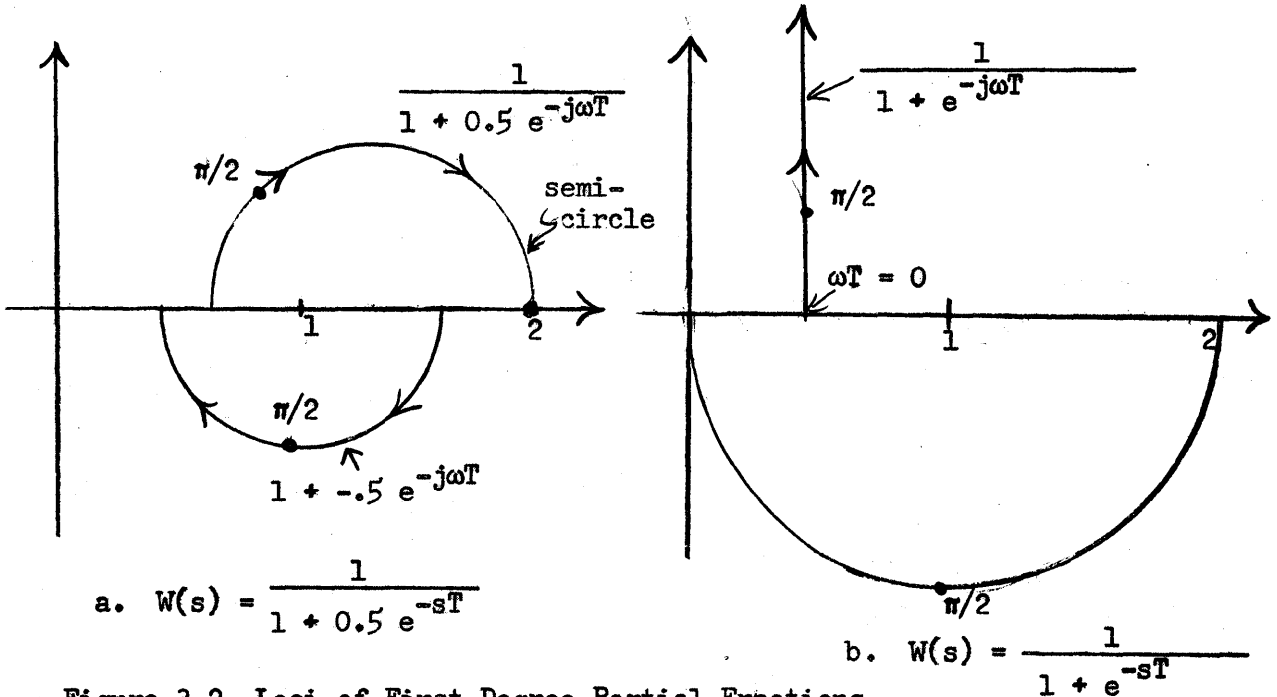


Figure 3.2 Loci of First-Degree Partial Fractions

3.23 Second-Degree Partial Fractions (complex roots)

If a typical term,

$$W_1(s) = \frac{\alpha}{1 + \beta\, e^{-sT}}, \tag{3-12}$$

in the partial fraction expansion of a rational function of e^{-sT} has complex constants, there will be another term,

$$W_2(s) = \frac{\alpha^*}{1 + \beta^* e^{-sT}}, \tag{3-13}$$

in the expansion whose constants are the conjugates of those of $W_1(s)$. (The asterisk means conjugate.) Both $W_1(s)$ and $W_2(s)$ will be stable if and only if $|\beta| \leq 1$. If the rational function has real coefficients (as in practical problems) the terms such as $W_1(s)$ and $W_2(s)$ must come in pairs. Add the two and obtain a typical second degree partial fraction with real coefficients.

$$W_3(s) = W_1(s) + W_2(s) = \frac{(\alpha + \alpha^*) + (\alpha\beta^* + \alpha^*\beta)e^{-sT}}{1 + (\beta + \beta^*)\, e^{-sT} + \beta\beta^*\, e^{-2sT}} \tag{3-14}$$

In this section the discussion is restricted to second-degree partial fractions whose denominators have complex roots of e^{-sT}. To simplify the analysis, $W_3(s)$ can be written as,

$$W_3(s) = \frac{1 + a_1 e^{-sT}}{1 + b_1 e^{-sT} + b_2 e^{-2sT}}, \tag{3-15}$$

which differs from (3-14) by only a constant multiplying factor.

To insure that the roots of the denominator of (3-15) are complex, $b_1^2 < 4b_2$. With this condition imposed on the constants of the denominator of (3-15), it can be considered a basic building block of a program transfer function.

Now let us consider the stability of such a building block. $W_3(s)$ will be stable if and only if both $W_1(s)$ and $W_2(s)$ are stable. A comparison of (3-14) and (3-15) shows that,

$$b_1 = \beta + \beta^*, \text{ and } b_2 = \beta\beta^* = |\beta|^2. \tag{3-16}$$

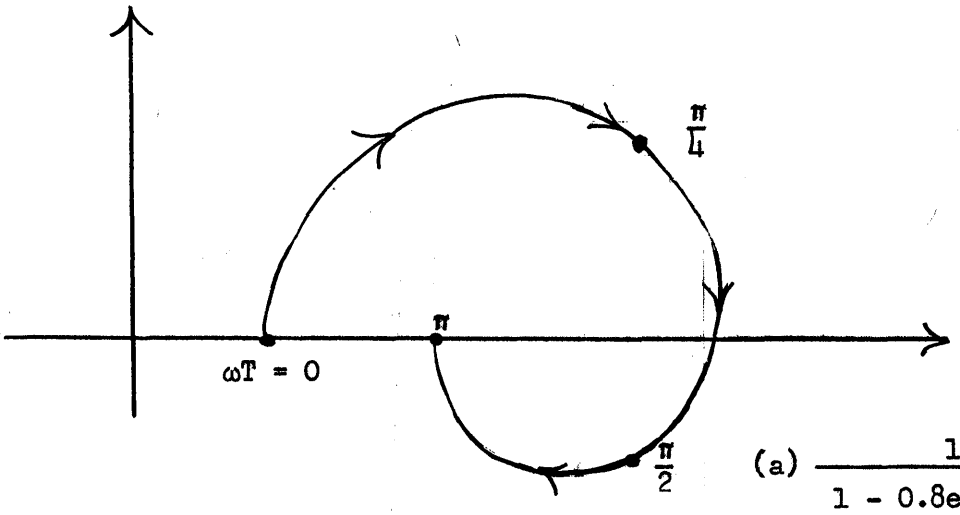
Therefore, the necessary and sufficient condition for the stability of $W_3(s)$ is that, $0 < b_2 \leq 1$. We may combine this with the condition for complex roots in the denominator of $W_3(s)$ and obtain,

$$\left(\frac{b_1}{2}\right)^2 < b_2 < 1 \tag{3-17}$$

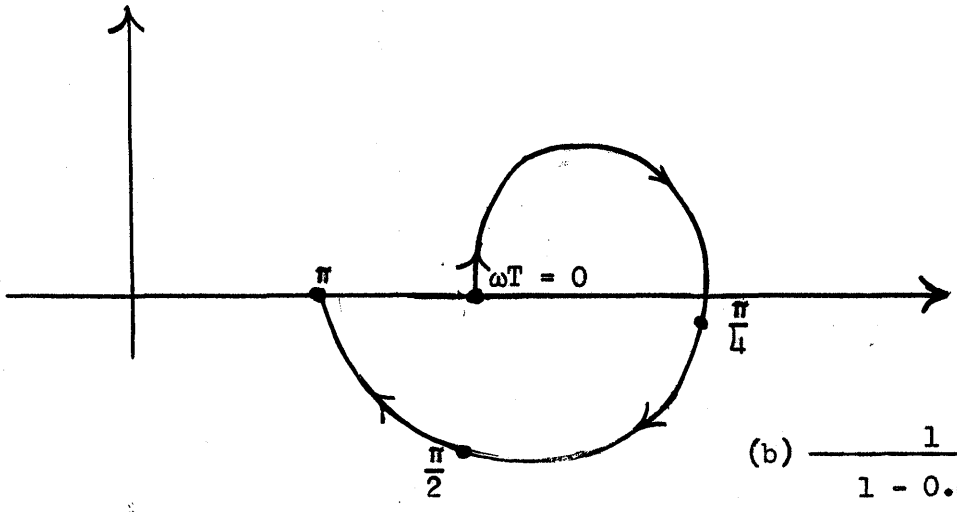
as the necessary and sufficient condition that insures stability of $W_3(s)$ and complex roots of its denominator.

In Fig. 3.3 there are plotted three loci of building blocks of the form (3-15). All three are stable. It should be observed that by changing only the numerator of the transfer function, three completely different loci have been obtained.

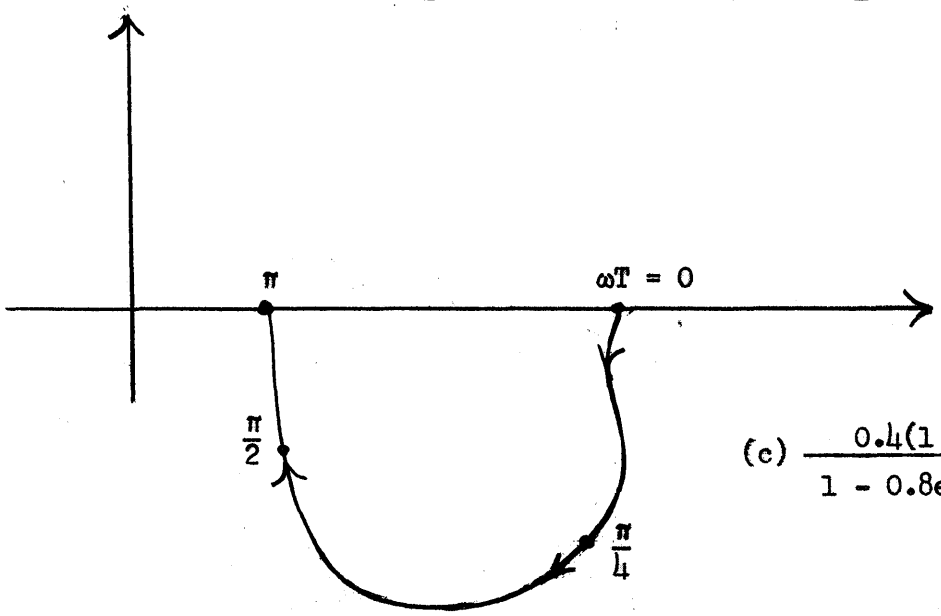
By adding building blocks of the form discussed in these sections, a desired frequency characteristic can be synthesized. The synthesis of a differentiating program is discussed in a subsequent section.



$$(a)\quad \frac{1 - 0.8e^{-sT}}{1 - 0.8e^{-sT} + 0.4e^{-2sT}}$$



$$(b)\quad \frac{1 - 0.4e^{-sT}}{1 - 0.8e^{-sT} + 0.4e^{-2sT}}$$



$$(c)\quad \frac{0.4(1 + 0.5e^{-sT})}{1 - 0.8e^{-sT} + 0.4e^{-2sT}}$$

Figure 3.3 Loci of Second-Degree Partial Fractions

3.3 Realization of Programs From Their Transfer Functions

The purpose of this section is to develop and compare methods by which a program can be derived from a rational function $z = e^{-sT}$ which is the transfer function of the program. How this rational function is arrived at in the first place is the concern of the last section of this chapter.

3.31 General Considerations in Program Realization

In choosing a particular method of programming, one may consider the following factors: storage requirements and time requirements. To a certain extent one of these requirements can be reduced at the expense of increasing the other and the optimum method will depend on the particular application. It is necessary, therefore, to make available various possible methods of programming and to form some idea about the requirements of each; intelligent program realization can then be adapted to each application.

In the consideration of storage requirements of linear programs, it is convenient to distinguish three types of storage: data storage, constant storage, and instruction storage[1]. The data are the successive sampled values of input and output. The complexity of a program is closely related to the number of constants and to the age of the data to which the program refers. The program can be divided into arithmetic and manipulative parts. The number of arithmetic operations involved is roughly proportional to the number of constants, each implying a multiplcation (of a piece of data by the constant) and an addition (of the product to the other terms). The number of manipulative operations is related to the "age" of the oldest

---

[1] It is understood that in a general-purpose computer there is no physical difference between the storage registers containing numbers or instructions, and any register may hold either kind of information. The distinction made here is only for the purpose of discussion.

data used, where age is expressed in terms of sampling intervals. All "younger" data must also be stored even if not used at each calculation (the corresponding constants being zero), for eventually they will become the oldest data. After each calculation of a new output value, the manipulative instructions shift each piece of data to a storage location at which an older piece of data has been, the oldest data being lost. The manipulative program is seen to rearrange the data storage in such a manner that at the new sampling point the same arithmetic program will calculate a new output value.

The time requirement of a program is the product of the number of instructions to be carried out and the average duration of an instruction. The latter factor depends on the physical characteristics of a particular computer and is more or less fixed; the number of instructions performed in sequence, however, depends in part on the manner of program realization. In each particular realization a significant trading of time for storage is possible by so-called cyclic procedures. One notes that often the calculation of each term in a program involves the same sequence of arithmetic operations. The simplest and fastest procedure is to store as many of these sequences as there are terms to be calculated. Considerable storage may be saved, however, by storing these instructions only once and cycling through them as many times as there are terms to calculate. Unfortunately, the time requirement increases considerably, for in each cycle the addresses of the instructions must be adjusted to make them refer to different storage locations for different terms and the number of cycles must be counted to permit termination of the cycling process.

The following sections and related appendices will serve as specific illustrations of these considerations in programming.

3.32 Direct Regression or Direct Programming

The starting point of our realization procedure is the general expression for the transfer function of a linear program,

$$W(s) = \frac{a_o + a_1 e^{-sT} + a_2 e^{-2sT} + \ldots + a_m e^{-msT}}{1 + b_1 e^{-sT} + b_2 e^{-2sT} + \ldots + b_n e^{-nsT}} . \qquad (3\text{-}18)$$

In order to interpret a program in the time domain it is necessary to eliminate fractional expressions. The most straightforward way of doing this follows directly from (3-18). From it we can obtain

$$\tilde{O}(s) = (a_0 + \ldots + a_m e^{-msT})\, \tilde{I}(s) - (b_1 e^{-sT} + \ldots + b_n e^{-nsT})\, \tilde{O}(s). \qquad (3\text{-}19)$$

The inverse transform of this expression is

$$o(t) = a_0 i(t) + a_1 i(t - T) + \ldots + a_m i(t - mT) - b_1 o(t - T) - \ldots - b_n o(t - nT), \qquad (3\text{-}20)$$

where $o(t)$ and $i(t)$ are impulse-modulated (sampled) time functions having the value zero everywhere except at the sampling points. In terms of some continuous functions $o(t)$ and $i(t)$, which agree with the area-values of $o(t)$ and $i(t)$ at the sampling points, (3-20) is often written as

$$o_j = a_o i_j + a_1 i_{j-1} + \ldots + a_m i_{j-m} - b_1 o_{j-1} - \ldots - b_n o_{j-n}, \qquad (3\text{-}21)$$

where j signifies particular sampling point and j-k the k-th preceding sampling point. Eq. (3-21) is more familiar to the numerical analyst than (3-20), but the two are entirely equivalent and are called regression formulas. These equations state that the present result (output) is computed by a finite linear combination of the present and past input values and of past results (output values).

Several characteristics of regression formulas should be observed. If the right side of (4-20) or (4-21) has at least one non-zero b_k, then

the present output depends on at least one previous output, which in turn depends on an output further back and so on. It follows that the present output value is affected by output values as far back as the start of the problem and therefore, also by input values that far back. Thus regressing to a finite number of output values corresponds to regressing to an unlimited number of input values. This aspect of the regression equations is important and will be further emphasized in the following sections.

Interesting conclusions can be drawn concerning memory requirements on the digital computer by considering the actual programming of the regression formula, (3-21). Assuming that none of the coefficients a_k and b_k are zero, one can easily see that in order to calculate a new output value o_j, when a new input value i_j is received, m previous input values and n previous output values will have to have been remembered, requiring m + n memory positions for data where m and n are the subscripts of the last non-zero coefficients, and furthermore, it is necessary to store all these data even if some other coefficients are zero because at the next sampling point the same pieces of data will be associated with different coefficients.

It can be stated that, at least when programming is done by the illustrated direct regression method, the data memory consists of m + n registers (memory positions) where m and n are the degrees of the numerator and denominator polynomials in $z = e^{-sT}$ of the program transfer function W(s). Actually this data storage requirement may be reduced, as will be shown in Sections 3.33 and 3.34.

To be able to make comparisons between the various synthesis procedures it is necessary to do the actual programming. This exercise is left to the appendices, and the results will be compared after the other synthesis procedures will have been discussed. In Appendix B the

arithmetic and manipulative parts of the direct regression program are first constructed separately; then a new more compact program is shown which interleaves the arithmetic and manipulative instructions. Although the Whirlwind code is used, the results and conclusions can be considered quite general in view of the fact that the instruction complements of most general-purpose digital computers are conspicuously similar.

3.33 Cascade Programming

If the numerator and denominator polynomials in $z = e^{-sT}$ are factored, (4-18) takes the form

$$W(s) = a_o \frac{(1 + c_1 e^{-sT})(1 + c_2 e^{-sT}) \ldots (1 + c_m e^{-sT})}{(1 + d_1 e^{-sT})(1 + d_2 e^{-sT}) \ldots (1 + d_n d^{-sT})}, \qquad (3\text{-}22)$$

where $-(1/c_k)$ and $-(1/d_k)$ are the roots of numerator and denominator respectively, when considered as polynomials in z. Because the coefficients a_k and b_k of these polynomials are real, the c_k and d_k will also be real or will come in conjugate pairs. At any rate, it is possible to group the monic factors of (3-22) in some manner

$$W(s) = W_1(s)\, W_2(s) \ldots W_p(s), \qquad (3\text{-}23)$$

where each $W_k(s)$ is of a rational form in z having a numerator and a denominator of not higher degree in z than W(s) itself has.

The form of (3-23) reminds one of the transfer function of cascaded linear units in a servo system. Cascading means that the output of one unit becomes the input to the next one. There is no difficulty in using the same interpretation to define cascaded programs. At every sampling point the output of each regression equation is used in calculating the output of the next one.

To be more specific, let us assume that: (1) W(s) is a proper rational fraction in z, that is, $m < n$[1]; (2) all roots $-1/c_k$ and $-1/d_k$ are real and distinct, since generalization to the case of conjugate complex roots turns out to be direct; (3) $a_o = 1$ in order to avoid its nuisance value in the discussion[2]. With these assumptions it is possible to have $p = n$ in (3-23) with the denominator of each $W_k(s)$ being a single monic factor in z; that is,

$$W_k(s) = \frac{1 + c_k e^{-sT}}{1 + d_k e^{-sT}} , \qquad (3\text{-}24)$$

where c_k may or may not be zero, but $d_k \neq 0$. There are n factors of the type (3-24) each representing a simple regression equation. In m of the factors $c_k \neq 0$, in the other n-m factors $c_k = 0$. The data storage associated with each $W_k(s)$ equals 2 when $c_k \neq 0$, and 1 when $c_k = 0$.[3] However, the input data that must be stored when $c_k \neq 0$, is also the output data that had to be stored for the preceding cascade program $W_{k-1}(s)$; consequently, there is only one data to be stored for each $W_k(s)$ regardless of the value of c_k, except for the first one $W_1(s)$. But when $m < n$ (proper rational fraction), one $c_k = 0$, say $c_1 = 0$, making the total required data

---

1 If $m \geq n$, W(s) can be written as the sum of a polynomial and a proper rational fraction in $z = e^{-sT}$. The program corresponding to the polynomial part is a simple linear combination of input values. Discussion of this case is omitted without any serious loss of generality.

2 If $a_0 \neq 1$, only a simple multiplication has to be added to the program.

3 Each $W_k(s)$ is the transfer function of just a regression equation and its data storage is the sum of degree of numerator and denominator, as discussed in the previous article.

storage n; a material reduction over the m + n data needed in direct regression programming.

In order to translate the cascade scheme into an actual program, we may proceed as follows. First, we write (3-23) (with p = n) in terms of input and output transforms, as

$$\frac{\tilde{O}(s)}{\tilde{I}(s)} = \frac{\tilde{O}_1(s)}{\tilde{I}_1(s)} \cdot \frac{\tilde{O}_2(s)}{\tilde{I}_2(s)} \cdot \cdots \cdot \frac{\tilde{O}_n(s)}{\tilde{I}_n(s)} \cdot \qquad (3\text{-}25)$$

One way of making (3-25) an identity is by letting

$$\begin{aligned} \tilde{I}_1(s) &= \tilde{I}(s) \\ \tilde{I}_2(s) &= \tilde{O}_1(s) \\ \tilde{I}_3(s) &= \tilde{O}_2(s) \\ &\vdots \\ \tilde{I}_n(s) &= \tilde{O}_{n-1}(s) \end{aligned} \qquad (3\text{-}26)$$

which make

$$\tilde{O}(s) = \tilde{O}_n(s).$$

Using the relations (3-25) and (3-26) we obtain

$$\frac{\tilde{O}(s)}{\tilde{I}(s)} = \frac{\tilde{O}_1(s)}{\tilde{I}(s)} \cdot \frac{\tilde{O}_2(s)}{\tilde{O}_1(s)} \cdot \cdots \cdot \frac{\tilde{O}(s)}{\tilde{O}_{n-1}(s)} \qquad (3\text{-}27)$$

The various factors equal the respective program transfer functions; namely,

$$\left.\begin{aligned}
\frac{\tilde{o}_1(s)}{\tilde{I}(s)} &= W_1(s) = \frac{1}{1 + d_1 e^{-sT}} \\
\frac{\tilde{o}_2(s)}{\tilde{o}_1(s)} &= W_2(s) = \frac{1 + c_2 e^{-sT}}{1 + d_2 e^{-sT}} \\
&\vdots \\
\frac{\tilde{o}(s)}{\tilde{o}_{n-1}(s)} &= W_n(s) = \frac{1 + c_n e^{-sT}}{1 + d_n e^{-sT}}
\end{aligned}\right\} \qquad (3\text{-}28)$$

where m of the c_k are not zero. Multiplying by the denominators changes the set (3-28) into

$$\left.\begin{aligned}
(1 + d_1 e^{-sT})\,\tilde{o}_1(s) &= \tilde{I}(s) \\
(1 + d_2 e^{-sT})\,\tilde{o}_2(s) &= (1 + c_2 e^{-sT})\,\tilde{o}_1(s) \\
(1 + d_3 e^{-sT})\,\tilde{o}_3(s) &= (1 + c_3 e^{-sT})\,\tilde{o}_2(s) \\
&\vdots \\
(1 + d_n e^{-sT})\,\tilde{o}(s) &= (1 + c_n e^{-sT})\,\tilde{o}_{n-1}(s)
\end{aligned}\right\} \qquad (3\text{-}29)$$

The inverse transform of the foregoing set, with one term of each equation transposed to the right side, is the desired set of regression equations.

$$\left.\begin{aligned}
\tilde{o}_1(t) &= \tilde{i}(t) - d_1\tilde{o}_1(t-T) \\
\tilde{o}_2(t) &= \tilde{o}_1(t) + o_2\tilde{o}_1(t-T) - d_2\tilde{o}_2(t-T) \\
\tilde{o}_3(t) &= \tilde{o}_2(t) + c_3\tilde{o}_2(t-T) - d_3\tilde{o}_3(t-T) \\
&\vdots \\
\tilde{o}(t) &= \tilde{o}_{n-1}(t) + c_n\tilde{o}_{n-1}(1-T) - d_n\tilde{o}(t-T)
\end{aligned}\right\} \qquad (3\text{-}30)$$

The detailed coded program corresponding to (3-30) is shown in Appendix C.

Cascade programming, although not referred to by that name, is a familiar technique in numerical procedures. However, the clear-cut and general equivalence of the direct regression and cascade programming is not always well understood. Cascade programming arises naturally from the kind of thinking prevelant in numerical work. Consider the simple example of solving the second-order differential equation,

$$\frac{d^2y}{dt^2} + \beta y = 0. \tag{3-31}$$

The derivatives may be considered as the separate variables, $y'(t)$ and $y''(t)$; then we obtain the following three sampled functions:

$$\left.\begin{aligned} \tilde{y}''(t) &= -\beta\tilde{y}(t - T) \\ \tilde{y}'(t) &= T\tilde{y}''(t) + \tilde{y}'(t - T) \\ \tilde{y}(t) &= T\tilde{y}'(t) + \tilde{y}(t - T) \end{aligned}\right\} \tag{3-32}$$

where the first equation of the set is derived from (3-31) while the second and third are elementary first-difference extrapolations. The set (3-32) indicates cascade programming because the output of the first equation is in the input of the second, and the output of the second equation is the input to the third. The peculiar thing in this case is that the input to the first equation is not an independent function but directly related to the output of the last equation. This feature establishes the constraint imposed by the differential equation.

The Laplace transform of the set (3-32) is

$$\left.\begin{aligned} \tilde{Y}'' &= -e^{-sT}\tilde{Y} \\ \tilde{Y}' &= T\tilde{Y}'' + e^{-sT}\tilde{Y} \\ \tilde{Y} &= T\tilde{Y} + e^{-sT}\tilde{Y} \end{aligned}\right\} \tag{3-33}$$

from which the explicit relations between inputs and outputs are obtained as follows:

$$\begin{aligned} Y'' &= -e^{-sT}Y \\ Y' &= \frac{T}{1-e^{-sT}}\tilde{Y}'' \\ Y &= \frac{T}{1-e^{-sT}}\tilde{Y}' \end{aligned} \tag{3-34}$$

For realization by three cascaded factors, we have

$$\begin{aligned} W_1(s) &= -\beta e^{-sT} \\ W_2(s) &= \frac{T}{1-e^{-sT}} \\ W_3(s) &= \frac{T}{1-e^{-sT}} \end{aligned} \tag{3-35}$$

It is clear that a single transfer function can be made to replace the cascaded system of three; thus

$$\begin{aligned} W(s) &= W_1(s)W_2(s)W_3(s) \\ W(s) &= \frac{-\beta T^2 e^{-sT}}{1-2e^{-sT}+e^{-2sT}} \end{aligned} \tag{3-36}$$

The corresponding regression equation is simply obtained as

$$\tilde{Y}(s) = (2-\beta T^2)e^{-sT}\tilde{Y}(s) = e^{-2sT}\tilde{Y}(s). \tag{3-37}$$

The inverse transform of this equation is

$$\tilde{y}(t) = (2-\beta T^2)\tilde{y}(t-T) - \tilde{y}(t-2T). \tag{3-38}$$

which could have been otained from (3-32) by the elimination of $y'(t)$ and $y''(t)$, but even in this simple case the process of elimination in the time domain is not direct.

The fact is that in numerical work a cascade method such as (3-42) is much more generally used than the direct regression of (3-38). Often there is good justification for this preference; for instance the values of the first and second derivatives may also be needed. However, when such or similar justifications do not exist, the direct regression may turn out to be simpler than cascading. In the present example, (3-32) calls for one more constant, two more multiplications and one more addition than (3-38). If the first two equations of the set (3-32) are combined, one multiplication is saved; furthermore, the manipulations in the direct method happen to be more awkward. Because in this case the input and output are the same quantity the formulas of Appendices B and C are not directly applicable, the requirements of the two methods must be determined by actual trials.

3.34 Parallel Programming

If the transfer function of a program is expanded by partial fractions in terms of z, (3-18) takes the form

$$W(s) = \frac{f_1}{1 + d_1 e^{-sT}} + \frac{f_2}{1 + d_2 e^{-sT}} + \cdots + \frac{f_n}{1 + d_n e^{-sT}} \qquad (3\text{-}39)$$

as long as $m < n$. Thus, the transfer function $W(s)$ is replaced by the sum of a number of simpler transfer functions; namely,

$$W(s) = W_1(s) + W_2(s) + \cdots + W_p(s), \qquad (3\text{-}40)$$

where some of the $W_k(s)$ may be the combination of several partial fractions, but all are of lower degree than $W(s)$ itself.

The form of (3-40) may remind one of parallel combinations of network admittances. Paralleling means that the same input (driving voltage) is applied to all component admittances and the output (driving-point current) is obtained as the sum of individual outputs (current through each admittance).

The same interpretation can be applied to parallel programming. The programming will involve p regression equations all using the same input values, and all their outputs adding to produce the desired over-all output.

To arrive at a more specific interpretation, we first make a few restrictions again: (1) W(s) is a proper fraction, i.e., $m < n$, and (2) the roots of the denominator polynomial are real and distinct. Then all constants f_k and d_k of (3-39) are real and in (3-40) p can equal n; moreover, each term of (3-39) is a simple regression equation involving two constants and one data storage. Thus, the total number of data to be stored is only n. Just like in the case of cascade programming, the lower requirement for data storage of parallel programming may be a great advantage over the direct programming method. However, this feature does not mean that parallel or cascade programming should always be employed in preference to direct programming. For instance, there is the case when $m = 0$; i.e., the numerator of W(s) is 1 (or a_o). Of the input values the program uses only the present one and the total data storage is n regardless of the programming scheme used; on the other hand, the number of constants will be n for the direct and cascade method, but 2n for the parallel method, putting the latter at a disadvantage. Similarly, if the denominator of the over-all transfer function lacks several terms (say, the denominator is $1 - b_n e^{-nsT}$), then factorization of the denominator introduces all terms, making the cascade and parallel program much longer than the direct program. Another factor which may militate against the use of parallel programming is the presence of multiple roots in the denominator. If a root is of multiplicity r, it may produce up to r terms of degrees r, (r-1), ...2,1 (the r-degree term never being absent) in the partial fraction exapnsion, but the same root will require only one r-degree, or r first-degree, cascaded factors.

In order to interpret the parallel method of programming, we proceed in the usual manner. For the various terms of (3-40) with p = n, we write

$$\left.\begin{aligned} W_1(s) &= \frac{\tilde{O}_1(s)}{\tilde{I}(s)} = \frac{f_1}{1 + d_1 e^{-sT}} \\ W_2(s) &= \frac{\tilde{O}_2(s)}{\tilde{I}(s)} = \frac{f_2}{1 + d_2 e^{-sT}} \\ &\vdots \\ W_n(s) &= \frac{\tilde{O}_n(s)}{\tilde{I}(s)} = \frac{f_n}{1 + d_n e^{-sT}} \end{aligned}\right\} \qquad (3\text{-}41)$$

and

$$W(s) = \frac{\tilde{O}(s)}{\tilde{I}(s)} . \qquad (3\text{-}42)$$

Cross-multiplication by the denominators in (3-41) yields the set

$$\left.\begin{aligned} (1 + d_1 e^{-sT})\tilde{O}_1(s) &= f_1 \tilde{I}(s) \\ (1 + d_2 e^{-sT})\tilde{O}_2(s) &= f_2 \tilde{I}(s) \\ &\vdots \\ (1 + d_n e^{-sT})\tilde{O}_n(s) &= f_n \tilde{I}(s) \end{aligned}\right\} \qquad (3\text{-}43)$$

while in view of (3-41) and (3-42), (3-40) can be written as

$$\tilde{O}(s) = \tilde{O}_1(s) + \tilde{O}_2(s) + \ldots + \tilde{O}_n(s) . \qquad (3\text{-}44)$$

The inverse transforms of (3-43) and (3-44) yield the desired set of regression equations, which follows.

$$\left.\begin{aligned} \sigma_1(t) &= f_1 \tilde{i}(t) - d_1 \sigma_1(t-T) \\ \sigma_2(t) &= f_2 \tilde{i}(t) - d_2 \sigma_2(t-T) \\ &\vdots \\ \sigma_n(t) &= f_n \tilde{i}(t) - d_n \sigma_n(t-T) \\ \sigma(t) &= \sigma_1(t) + \sigma_2(t) + \ldots + \sigma_n(t) \end{aligned}\right\} \qquad (3\text{-}45)$$

The detailed coded program corresponding to (3-45) is shown in Appendix D.

Parallel programming has not been generally used in numerical work. To the knowledge of the writer, the usual methods of numerical analysis do not naturally lead from a direct regression equation, which has reference to several previous input and output values, to a set of simpler regression equations, each of which refers only to the last input value and to a preceding[1] output value. By the method of frequency transformation the parallel method is found quite directly.

---

[1] In case of complex d_k's in (3-39), a combination of two conjugate complex partial fractions in z will result in a slightly more complicated regression equation, involving one additional input and output.

3.35 Comparison of Programming Methods

The purpose of this section is to compare the effectiveness of the various methods of program realizations based on the transfer functions of the programs. A complete general treatment appears too far-fetched and, therefore, this study is limited to a certain class of programs. Despite these limitations, which are discussed below, the investigation is sufficiently general to show how the results can be used to improve the instruction code of a general-purpose computer or to design a special-purpose computer, when these are used in control applications.

The three methods which will be compared are listed below:

(a) direct programming,

(b) cascade programming,

(c) parallel programming.

Other programming schemes may be derived from the rational transfer function W(s). One may carry out the long division in z of the numerator by the denominator until he arrives at a certain number of terms of the quotient. The transfer function can then be expressed as the sum of the quotient terms and of the remainder divided by the divisor (the original denominator). Any number of variations can be obtained by stopping the long division after different number of steps, but only in the most unusual cases can this approach be expected to yield a more efficient scheme of programming than the three major methods discussed in the preceding sections.

Other schemes that are even more artificial than the long-division scheme may be derived, but no other general programming method has been found that gives promise of effectiveness comparable to the three which are considered. It is noted that in certain cases a combination of two of the three listed methods may turn out to be more efficient than any one. An example of such a case is described below.

As the basis of comparison of programming methods, the requirements in storage and time are used. The particular application or purpose decides which of these two factors should deserve more attention. It is assumed that the complete sequence of instructions, as used at each sampling point, is stored; the possibility of cycling programs, which re-uses a short sequence of instructions for the calculation of each term, is not discussed. Essentially the Whirlwind I code is used throughout, but variations are considered.

As a starting point we recall that the transfer function of a linear program is,

$$W(s) = \frac{\tilde{O}(s)}{\tilde{I}(s)} = \frac{a_o + a_1 e^{-sT} + a_2 e^{-2sT} + \cdots + a_m e^{-msT}}{1 + b_1 e^{-sT} + b_2 e^{-2sT} + \cdots + b_n e^{-nsT}} \qquad (3\text{-}46)$$

In general, m and n may be any positive integers[1] and indeed, their relative sizes will hardly influence the comparisons to follow. Nevertheless, it is helpful to distinguish three cases:

(1) n = 0. (3-46) reduces to a polynomial in $z = e^{-sT}$; i.e., the new output value depends only on present and past input values, not on past outputs also. Present-day numerical analysis abounds in numerical

[1] This is in contrast with networks where certain restrictions on the degrees of numerator and denominator polynomials often exist.

processes corresponding to this special case.[1]

(2) $m < n$. (3-46) has the form of a proper rational function of z in this case. In Sections 3.32, 3.33, and 3.34 dealing with the various programming schemes, this case was assumed for the sake of simplicity.

(3) $m \geq n$. The rational function in z of (3-46) may be called improper, but it can be converted to the sum of a polynomial (Case 1) and of a proper rational fraction (Case 2) in $z = e^{-sT}$.

In order that the storage and time estimates to be arrived at should apply to all cases, it is necessary to define the following quantities with reference to (3-46):

m = degree of numerator,

n = degree of denominator,

m_k = one less than the number of non-zero constants in the numerator ($m_k \leq n$).

n_k = one less than the number of non-zero constants in the denominator ($n_k \leq n$).

m_e = one more than the excess of m over n; i.e., $m_e = m-n+1$ when $m \geq n$, and $m_e = 0$ otherwise. For proper rational fractions $m < n$ and $m_e = 0$.

On basis of the coded programs shown in the appendices, the table of Fig. 3.4 summarizes the storage and time requirements in terms of the quantities just defined. This tabulation is more general than the results given in the appendices, for in the appendices it was also assumed that none of the constants were zero, that is, $m_k = m$ and $n_k = n$; furthermore,

---

[1] Examples are numerical methods based on polynomial approximations with equidistant spacing of the independent variable. Indeed, such examples form not an insignificant portion of the available numerical techniques.

only case (2) was treated making $m_e = 0$. On the other hand, in the tabulation of Fig. 3.4 these restrictions of the appendices are absent, but the following assumptions are still made: the roots of the numerator and denominator are real and distinct, and the straightforward programming techniques of the appendices is used. Thus, the constants stored are those that appear explicitly in the various regression equations. Actually some saving in instructions would result from the use of certain ratios of these constants. For instance, the regression equation [cf. (3-45)]

$$\tilde{o}_1(t) = f_1\tilde{i}(t) - d_1\tilde{o}_1(t-T) \tag{3-47}$$

takes six instructions, as shown in the coded program of Appendix D. If, however, (3-47) is written as

$$\tilde{o}_1(t) = f_1\left[\tilde{i}(t) - \frac{d_1}{f_1}\tilde{o}_1(t-T)\right], \tag{3-48}$$

its coding would cost five instructions only, but certain questions on the relative sizes of the constants would arise. It seemed best to avoid such questions, because the considerations here are rather general and the value of a too-specialized treatment is questionable.

The comparison of the three methods of programming can be undertaken by considering each item of Fig. 3.4. Because of the straight sequential programming the time requirements are the same as the storage for instructions and, therefore, consideration of storage will give a complete picture.

As far as the number of constants stored are concerned, the direct method is not worse than the cascade, which in turn is not worse than the parallel method. This is so because in the direct method only the non-zero constants of (3-46) have to be stored, while factorization in the cascade case will produce as many constants as there are roots in z. In the parallel method two constants (root and residue) are produced for each denominator

root and if the numerator is not of lesser degree than the denominator, further terms and constants result. As an example, consider

$$W(s) = \frac{\frac{5}{8} - \frac{1}{4} e^{-sT}}{1 - \frac{3}{4} e^{-2sT} + \frac{1}{8} e^{-4sT}} \qquad (3\text{-}49)$$

for which

$$m = 1, \quad m_k = 1, \quad m_e = 0$$

$$n = 4, \quad n_k = 2,$$

According to the table of Fig. 3.4 the various constant storage requirements are

direct: $m_k + n_k + 1 = 4$

cascade: $m + n + 1 = 6$

parallel: $2n = 8$

These figures can be simply checked. In the direct case the four constants are apparent in (3-49). For the cascade case, the transfer function is written as

$$W(s) = \frac{1}{1 + \frac{1}{\sqrt{2}} e^{-sT}} \cdot \frac{1}{1 - \frac{1}{\sqrt{2}} e^{-sT}} \cdot \frac{1}{1 + \frac{1}{2} e^{-sT}} \cdot \frac{\frac{5}{8} - \frac{1}{4} e^{-sT}}{1 - \frac{1}{2} e^{-sT}}, \qquad (3\text{-}50)$$

and the six constants in question are: $+1/\sqrt{2}$, $-1/\sqrt{2}$, $+1/2$, $-1/2$, $+5/8$, and $-1/4$. The manner of programming illustrated in Appendix C actually necessitates the separate storing of positive and negative constants, even though of the same magnitude.

For parallel programming W(s) of (3-49) is expanded in partial fractions in terms of z and takes the form

$$W(s) = \frac{\frac{5+2\sqrt{2}}{8}}{1+\frac{1}{\sqrt{2}}e^{-sT}} + \frac{\frac{5-2\sqrt{2}}{8}}{1-\frac{1}{\sqrt{2}}e^{-sT}} + \frac{-\frac{9}{16}}{1+\frac{1}{2}e^{-sT}} + \frac{-\frac{1}{16}}{1-\frac{1}{2}e^{-sT}} \qquad (3\text{-}51)$$

The eight constants to be stored are evident in the foregoing equation.

The next item of comparison is the data storage, for which the above example reads, on basis of Fig. 3.4

direct: $m + n = 5$

cascade: $n = 4$

parallel: $n = 4$

These figures can be verified in the three foregoing equations. The numerator of (3-49) indicates that one past input value (corresponding to the e^{-sT} term) must be stored; the present input is used as it arrives and then stored as the past input for the next calculation, as shown in Appendix B. Thus the numerator implies one data register only. Similarly the denominator implies the storage of four past output values, even though the e^{-sT} and e^{-3sT} terms are absent; for the corresponding past outputs must be remembered for the next calculation.

For the cascade method, (3-50) seems to indicate five past data to be remembered; however, the e^{-sT} term of the last numerator refers to a past input that is also the past output of the preceding factor, since in cascade programming the input of a component program is the output of the previous one.

In case of parallel programs the four past data are quickly identified with the e^{-sT} terms of (3-51).

The expressions for instruction storage and for time requirements are identical, and produce the following tally in the present example:

direct: $2(m + m_k + n + n_k) + 7 = 23$

cascade: $3m + 4n + 6 = 25$

parallel: $7n + 4 = 32$

No verification of these figures is carried out by detailed coding of the programs because the appendices cover the general case. The advantage seems to be on the side of direct programming as far as time is concerned, but this advantage is slight and arises from the fact that in the present example two denominator constants are zero. An advantage of direct programming appears also in the total storage requirements for the same reason:

direct: $4 + 5 + 23 = 32$

cascade: $6 + 4 + 25 + 1 = 36$

parallel: $8 + 4 + 32 + 1 = 45$

This example, as well as the tabulation of Fig. 3.4, indicates the disadvantage of parallel programming. It seems that this kind of programming may have an advantage over either of the other two in certain cases, but hardly ever over both at the same time. Thus, the choice narrows down to direct and cascade programs, or possible combinations thereof. To show how a combination of methods may be used, we write (3-49) as

$$W(s) = \frac{1}{1 - \frac{1}{2} e^{-2sT}} \cdot \frac{\frac{5}{8} - \frac{1}{4} e^{-sT}}{1 - \frac{1}{4} e^{-2sT}} \qquad (3\text{-}52)$$

which indicates a cascade combination of two direct programs, for which respectively

$m = 0$	$m = 1$
$m_k = 0$	$m_k = 1$
$n = 2$	$n = 2$
$n_k = 1$	$n_k = 1$
$m_e = 0$	$m_e = 0$

The direct program of each cascaded component is somewhat simpler than it would be for two separate direct programs because the input and output devices are manipulated only once for the composite program, rather than once for each component program. This saving amounts to six instructions, thus the instruction storage or time requirement is:

first components: $2(m + m_k + n + n_k) + 6$	$= 12$
second components: $2(m + m_k + n + n_k) + 6$	$= 16$
saving as indicated above	-6
total instructions	22

Four constants appear in (3-52), two of which are accidentally identical, and one of which is made 1; thus, the constant storage is:

first component:	$m_k + n_k + 1$	$= 2$	
	saving	$= -1$	1
second component:	$m_k + m_k + 1$	$= 3$	
	saving	$= -1$	2
total constants			3

A saving arises in data memory also, because the past input of the second component is also the past output of the first one. This gives the following need of data storage:

first component:	m + n = 3
second component:	m + n = 3
saving	-1
	4

The results of this example are summarized in Fig. 3.5, which shows a small advantage of the mixed method over the direct one.

To pursue further the detailed comparison of these various methods of programming would lead to undue specializations in the Whirlwind code and to results of doubtful general value. The illustrated attack on the realization problem, however, shows how a useful estimate of the complexity of coded programs can be gained from the evident properties of their transfer functions. Three further problems will be touched on briefly: (1) computing delays, (2) means of using the results to select computer codes; and (3) means of using the results to design special-purpose computers.

A consideration that has been omitted in our discussion is the delay incurred through the computation itself. If a digital computer is used as part of a number of control systems -- say, 50 systems --, then in each sampling interval it performs 50 computations, one for each system. The time of a computation is then at most 1/50 of the sampling time, T, and this delay is presumably negligible. If, however, one digital computer were used with each system, the computation may and, for the sake of efficiency, should take an appreciable part of the sampling time. Such a delay would be very serious and the computer would have to perform a prediction in addition to the required compensation. In turn, this would lengthen the program, make it less effective, and may even force a longer sampling time; indeed, in a marginal case, in which the original compensating program had a delay nearly as large as the sampling time, the effect may become

cumulative, since a longer sampling time would in turn require a better and longer program, and so on. In such marginal cases and in any case in which the computing time is not negligible with respect to the sampling time, the direct programming has a tremendous advantage over all other methods. A glance at the direct regression equation (3-20) shows clearly that all terms but the first one on the right side of the equation can be computed before the new input value is obtained.[1] The computing delay will thus be the time of merely calculating the term, $a_o\tilde{i}(t)$, and adding it to the already prepared partial result. This delay may conceivably be negligible.

All realizations of real-time linear programs involve accumulation of products as their arithmetic action and the transfer of data from one register to another as their manipulative action. In case of a single-address[2] instruction code, such as that of Whirlwind, the ex (exchange) operation[3] was shown in Appendix B to be very helpful in improving the efficiency of the code. Other improvements are possible by incorporating special operations which facilitate the particular type of programs on hand. Computers using multiple-address codes could be particularly efficient in such applications. For instance, in a three-address code an instruction could locate a constant, a piece of data, and transfer that data to a third address, after which it would multiply the constant and data

---

[1] The second composite program in Appendix B is written in this manner.

[2] Each instruction specifies an operation and the storage address of a single operand.

[3] This operation exchanges the contents of the accumulator register with the specified storage register. Thus, one instruction performs a double duty by obtaining new data from storage and also transferring to storage a partial result.

accumulating this product with the partial result always left in the arithmetic element of the computer. This single order would complete both the arithmetic action (accumulation of products) and the manipulative action (transfer of data to an "older-data" register) associated with one term of a regression equation.

Similar considerations allow one to adapt special-purpose or fixed-program digital computers to control specifications. To be somewhat specific we assume that the computer is used as part of a single control system and will have to perform only one computation in each sampling period. The computer would not operate appreciably faster than one computation per sampling period and in order to minimize the computational delay it would follow a direct regression program. In order to keep such a single-system computing equipment from becoming excessive, a serial[1] computer would probably be used. The program of the computer would be fixed to correspond to a direct regression program of certain complexity as defined by the degrees of m of the numerator and n of the denominator of the program transfer function. The constants could be set manually on toggle switches or relays, or they could be stored on the same high-speed storage device[2] on which the data are stored. A serial adding unit with proper switching equipment would allow the multiplication of constant and data (by repeated additions) and the addition of such product to the accumulated partial result. The physical size of such a digital control unit may be quite feasible in certain applications and the design of such a simple special-purpose digital computer would be particularly justified if the incoming data were sampled and digital to start with.

---

[1] A serial computer operates on each digit of a number in sequence; thus, the equipment is not duplicated for each digit.

[2] Magnetic-drum memory, for instance.

3.4 Synthesis of Programs in the Frequency Domain

3.41 General Synthesis Procedure

The synthesis of computer programs in the frequency domain may be broken down into the three following stages (1) specification of the desired frequency characteristic or lous of $W(j\omega)$, (2) approximation of the desired locus by a realizable program transfer function, and (3) realization of the program. One way to determine the desired locus is from the Laplace transform of the operation the computer is to perform. The second step is the difficult part of the problem. The desired frequency characteristic must be approximated by a rational function of e^{-sT}. No general rules are available for making this approximation, but before making the approximation, one should gain some experience in analyzing program building blocks in the complex plane. Possibly the most systematic approach, at present, to the approximation problem is to make successive approximations to the desired characteristic, using the basic program building blocks of Section 3.2. The third step involves only a straightforward inverse Laplace transform. As an example of program synthesis in the frequency domain, a program for differentiation will now be synthesized.

3.42 Synthesis of a Differentiation Program

An ideal differentiator establishes the following relation between input and output:

$$O(t) = \frac{d}{dt} i(t). \tag{3-52}$$

Disregarding initial conditions, the Laplace transform of (3-52) is

$$H(s) = \frac{O(s)}{I(s)} = s, \tag{3-53}$$

and this is the desired transfer function. For $s = j\omega$, $H(s)$ becomes

$$H(j\omega) = j\omega. \tag{3-54}$$

So the locus of the desired transfer function is the imaginary axis. This completes the first step of the synthesis procedure.

The second step is to find a rational function of e^{-sT} that approximates this locus. This approximation is to be made by geometric considerations based on the desired locus. In this particular example it is also possible to employ analytic considerations based on the desired transfer function of (3-54). It so happens that in the present case the analytic approach is simpler; nevertheless, the geometric approach is shown first.

The crudest numerical approximation to a first derivative is the first divided difference.

$$\tilde{o}(t) = \frac{\tilde{i}(t) - \tilde{i}(t-T)}{T} \tag{3-55}$$

The Laplace transform of (3-55) is

$$\tilde{O}(s) = \tilde{I}(s) \frac{1 - e^{-sT}}{T} . \tag{3-56}$$

Thus the transfer function of the differencing process is

$$W_o(s) = \frac{\tilde{O}(s)}{\tilde{I}(s)} = \frac{1 - e^{-sT}}{T} . \tag{3-57}$$

Fig. 3.6 shows the locus of $W_o(j\omega)$ and compares it with the desired one. At low values of ωT (i.e., when the frequency of the input function is low

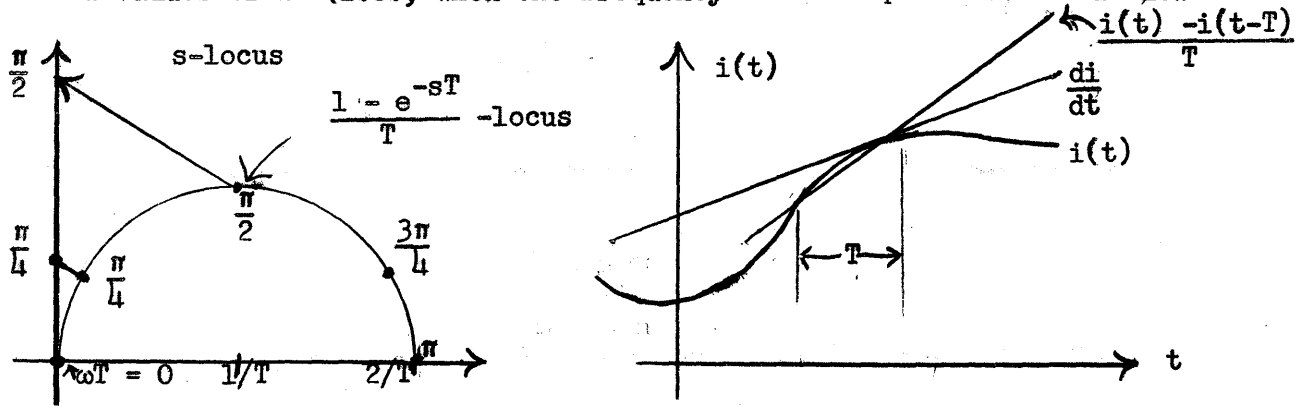


Figure 3.6 Comparison of first derivative and first difference operators

with respect to the sampling frequency) the two loci agree reasonably well. If we could straighten out the circular locus, we would have a better approximation of the desired locus. Figure 3.7 illustrates a geometric construction that straightens out the locus and gives us ideal phase characteristics.

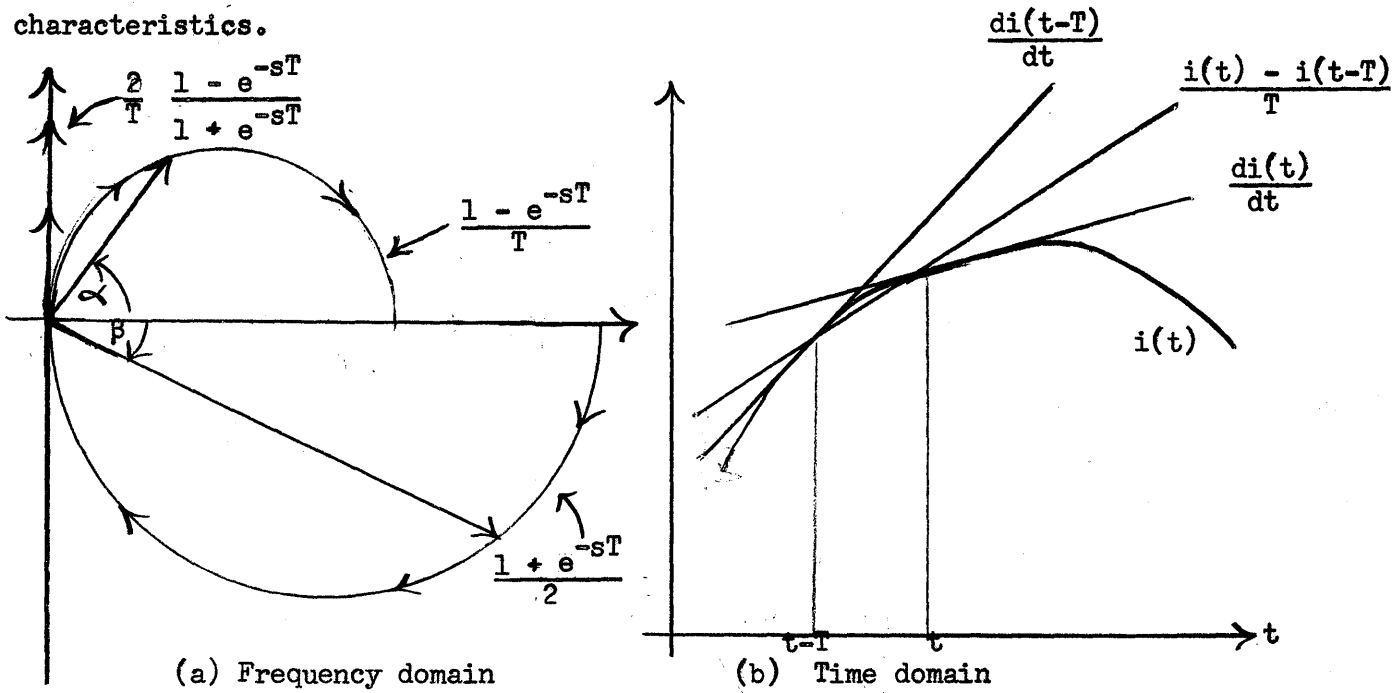


Figure 3.7 Derivation of an ideal phase, realizable differentiation operator

The vectors $(1/T)(1 - e^{-j\omega T})$ and $(1/2)(1 + e^{-j\omega T})$ are drawn for a particular frequency. Using the geometric rule that a triangle inscribed in a semicircle is a right triangle, one can readily show that $|\alpha| + |\beta|$ add up to 90°. However, since α is a positive phase angle, it must be subtracted from β (which is negative) to give a resulting angle of -90°, which is the phase of an ideal integrator. It follows that division of the β-locus by the α-locus will yield an ideal-phase formula. The resulting transfer function is

$$W_2(s) = \frac{2}{T} \quad \frac{1 - e^{-sT}}{1 + e^{-sT}}, \tag{3-58}$$

which has the desired phase in the range, $0 \leq \omega T \leq \pi$. The interpretation in the time domain is both plausible and illuminating. The inverse transform of (3-58) shows that,

$$\frac{\tilde{v}(t) + \tilde{v}(t - T)}{2} = \frac{\tilde{i}(t) - \tilde{i}(t-T)}{T} \tag{3-59}$$

(3-59) states that the average of the derivatives at two neighboring points is approximately equal to the divided difference for those points.

It is interesting to note that the same approximate transfer function, (3-58), can be obtained analytically based on a rather good approximation for e^{-sT}.

$$e^{-sT} \cong \frac{1 - \frac{1}{2} sT}{1 + \frac{1}{2} sT} \tag{3-60}$$

Solving (3-60) for s yields

$$s \cong \frac{2}{T} \frac{1 - e^{-sT}}{1 + e^{-sT}} . \tag{3-61}$$

Although in this particular case the above analytic approach is simple and fairly accurate, its general use has certain drawbacks. The most obvious one is that the rational function of "s" to be approximated, which in the present case is "s" itself, is in many cases not explicitly known; rather it may be obtained as an approximation to a desired locus or amplitude and phase response. Then to approximate the rational function of "s", which itself is but an approximation, by a rational function of e^{-sT} puts the designer on shaky grounds, and it might lead to far more involved programs than necessary. There is no substitute to going back to the original specifications and designing directly on their basis. Another disadvantage of the above analytic approach is that it is not general. One could replace all "s" by the approximation (3-61), but how one would get a better solution is not obvious.

We have an approximation of the differentiation operator, so the next thing to do is see how good it is. Since the desired locus and its approximation lie along the same path, a locus study does not give a good comparison. In such a case separate amplitude and phase plots can be studied. For $s = j\omega$, $W_2(s)$ becomes

$$W_2(j\omega) = \frac{2}{T}\frac{1 - e^{-j\omega T}}{1 + e^{-j\omega T}} = \frac{2}{T}\frac{e^{+j\frac{\omega T}{2}} - e^{-j\frac{\omega T}{2}}}{e^{j\frac{\omega T}{2}} + e^{-j\frac{\omega T}{2}}} = \frac{j2}{T}\tan\frac{\omega T}{2}, \qquad (3\text{-}62)$$

which verifies the previous statement that $W_2(j\omega)$ has ideal phase characteristics. Hence, it is sufficent to study the amplitude characteristic only.

$$H(j\omega) = j\omega; \qquad (3\text{-}63)$$

therefore,

$$\frac{W_2(j\omega)}{H(j\omega)} = \frac{\tan\frac{\omega T}{2}}{\frac{\omega T}{2}}. \qquad (3\text{-}64)$$

Thus we see from (3-64) that the ratio of the approximate function to the ideal one is always greater than unity. Figure 3.8, a plot of the amplitude characteristics, shows us that for low values of ωT, say for $\omega T \leq \frac{\pi}{12}$, the

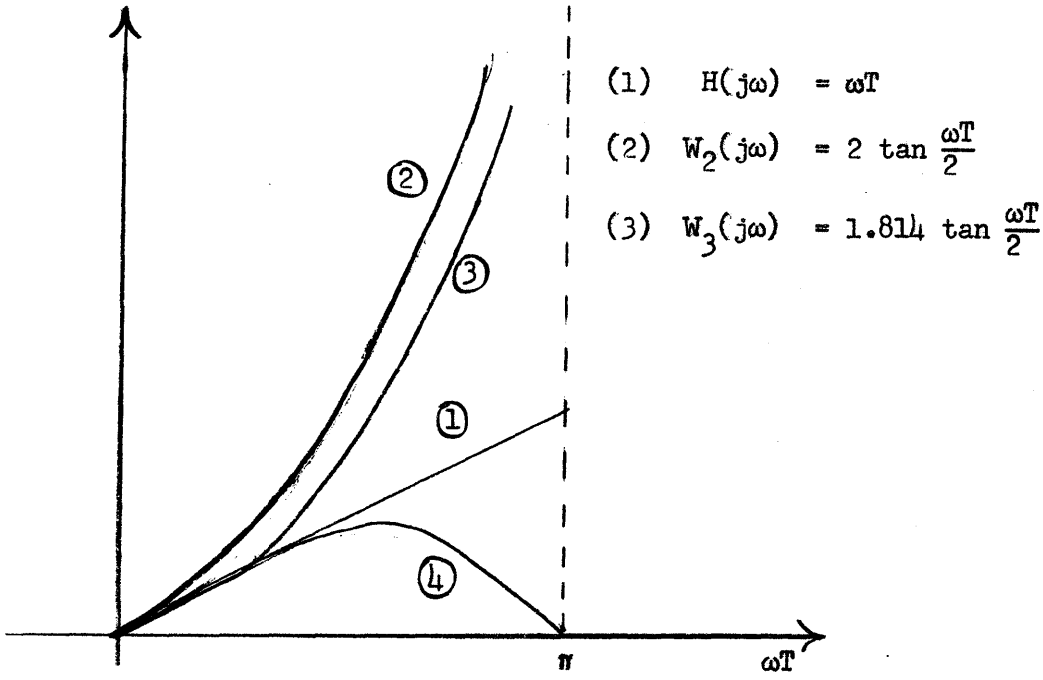


Figure 3.8 Comparison of amplitude characteristics of Differentiating Operators

differentiation program will give quite good accruacy. For certain control applications, values of ωT up to $\frac{\pi}{3}$ or even $\frac{\pi}{2}$ might give acceptable accuracies.

An examination of the amplitude characteristic of $W_2(s)$ in Fig. 3.8 reveals that if $W_2(s)$ is multiplied by a constant, which is slightly less than unit, we will obtain a better derivative on the average. The new transfer function is

$$W_3(s) = C\, W_2(s) = C\, \frac{2}{T}\, \frac{1 - e^{-sT}}{1 + e^{-sT}} . \tag{3-65}$$

Let us arbitrarily choose C so that $W_3(s) = H(s)$ for $sT = j\,\frac{\pi}{3}$. Then,

$$2\, C \tan \frac{\pi}{6} = \frac{\pi}{3} ; \tag{3-66}$$

so

$$C = \frac{\pi}{6 \tan \frac{\pi}{6}} = \frac{\pi\sqrt{3}}{6} = 0.907. \tag{3-67}$$

The improved transfer function is

$$W_3(s) = \frac{1.814}{T}\, \frac{1 - e^{-sT}}{1 + e^{-sT}} , \tag{3-68}$$

and its amplitude characteristic is also shown in Fig. 3.8.

Both curves 2 and 3 of Fig. 3.8 accentuate high frequencies which may be present at the input because of noise. In this case, a transfer function whose amplitude characteristic is like that of curve 4 would be a more desirable approximation for differentiation.

The inverse transform of (3-68) completes the synthesis of a differentiation program. The result is

$$\tilde{o}(t) = \frac{1.814}{T} \left[\tilde{i}(t) - \tilde{i}(t-T)\right] - \tilde{o}(t-T). \tag{3-69}$$

The accuracy of this differentiation program may be determined from Fig. 3.8, Curve 3.

CHAPTER IV

FREQUENCY ANALYSIS OF SOME NUMERICAL INTEGRATION FORMULAS

In this chapter we apply the methods of frequency analysis to several numerical integration formulas: the trapezoidal, Simpson's 1/3 rule, Simpson's 3/8 rule, and Weddle's rule. Frequency analysis is applied to determine the stability of these formulas, compare their accuracy, and compare their transfer functions with that of the ideal integrator.

4.1 Numerical Integration

In the numerical integration of definite integrals, the range of integration is divided into a convenient number of equal intervals, and the values of the integrand are defined only at the ends of these intervals. Essentially this is the same as sampling (or impulse modulating) the integrand. Let the distance between samples be T. To obtain an approximate value of the integral we may determine an n^{th} order polynomial that passes through n + 1 of the sampled points and integrate the polynomial over the corresponding range, repeating the process until the complete range of the original integral has been covered. If the sampled points are joined by straight lines, (approximation by a first order of polynomial) the resulting integration formula is known as the trapezoidal rule (each interval of the integrand is approximated by a trapezoid). Joining the points in each group of three sampled points by a parabola leads to an integration formula known as Simpson's 1/3 rule. If the points in each group of four sampled points are joined by a cubic curve, we

get Simpson's 3/8 rule. The trapezoidal rule and Simpson's 1/3 rule are quite widely known and used, but there is another one, called Weddle's rule, that is used to obtain great accuracy. Joining the points in each group of seven sampled points by a sixth order polynomial leads to Weddle's rule. In each case the range of integration should be divided into an integral multiple of "n" intervals. For example, to use Weddle's rule, the range of integration should be divided into 6, 12, 18.....equal intervals.

In what follows we shall designate the transfer function of an ideal integrator as H(s): Thus,

$$H(s) = \frac{1}{s} \tag{4-1}$$

with which the approximate integration formulas will be compared.

4.11 Trapezoidal Rule

Using the trapezoidal rule the definite integral,

$$o(t) = \int_{-\infty}^{t} i(x)\, dx, \tag{4-2}$$

may be approximated by,

$$o_1(t) = \frac{T}{2}\left\{ \left[i(t) + i(t - T) \right] + \left[i(t - T) + i(t - 2T) \right] + \ldots\ldots \right\} \tag{4-3}$$

The Laplace transform of (4-3) is

$$O_1(s) = \frac{T}{2} \left[(1 + e^{-sT})(1 + e^{-sT}) + e^{-2sT} + \ldots\ldots\ldots \right] I(s). \tag{4-4}$$

So the transfer function is

$$W_1(s) = \frac{O_1(s)}{I(s)} = \frac{T}{2} \quad \frac{1 + e^{-sT}}{1 - e^{-sT}} \tag{4-5}$$

A little algebra shows that for $s = j\omega$,

$$\frac{W_1(j\omega)}{H(j\omega)} = \frac{\omega T}{2} \cot \frac{\omega T}{2} \tag{4-6}$$

4.12 Simpson's 1/3 Rule

Using Simpson's 1/3 rule the definite integral (4-2) may be approximated by

$$o_2(t) = \frac{T}{3} \left\{ \left[i(t) + 4i(t - T) + i(t - 2T) \right] + \right.$$

$$\left. \left[i(t - 2T) + 4i(t - 3T) + i(t - 4T) \right] + \ldots\ldots \right\} \tag{4-7}$$

The Laplace transform of (4-8) is

$$O_2(s) = \frac{T}{3} \; I(s) \quad \left(1 + 4\, e^{-sT} + e^{-2sT}\right) \left(1 + e^{-2sT} + e^{-4sT} + \ldots\right) \tag{4-8}$$

or,

$$O_2(s) = \frac{T}{3} \quad \frac{1 + 4\, e^{-sT} + e^{-2sT}}{1 - e^{-2sT}} \quad I(s). \tag{4-9}$$

Therefore, the transfer function for Simpson's 1/3 rule is

$$W_2(s) = \frac{O_2(s)}{I(s)} = \frac{T}{3} \frac{1 + 4\,e^{-sT} + e^{-2sT}}{1 - e^{-2sT}} . \qquad (4\text{-}10)$$

Dividing (4-10) by (4-1), letting $s = j\omega$ and using some algebraic and trigonometric manipulations leads to the ratio

$$\frac{W_2(j\omega)}{H(j\omega)} = \frac{\omega T}{3} \frac{2 + \cos \omega T}{\sin \omega T} . \qquad (4\text{-}11)$$

4.13 Simpson's 3/8 Rule

The approximation to the definite integral (4-2) that is obtained using Simpson's 3/8 rule is

$$o_3(t) = \frac{3T}{8}\Big\{ \Big[i(t) + 3i(t - T) + 3i(t - 2T) + i(t - 3T) \Big] + \Big[i(t - 3T) + 3i(t - 4T) + 3i(t - 5T) + i(t - 6T) + \ldots\ldots \Big] \qquad (4\text{-}12)$$

The Laplace transform of $o_3(t)$ is

$$O_3(s) = \frac{3T}{8} I(s) \left(1 + 3\,e^{-sT} + 3\,e^{-2sT} + e^{-3sT}\right)\left(1 + e^{-3sT} + e^{-6sT} + \ldots\right) \qquad (4\text{-}13)$$

or,

$$O_3(s) = \frac{3T}{8} \frac{1 + 3\,e^{-sT} + 3\,e^{-2sT} + e^{-3sT}}{1 - e^{-3sT}} I(s) \qquad (4\text{-}14)$$

Hence for Simpson's 3/8 rule, the transfer function is

$$W_3(s) = \frac{O_3(s)}{I(s)} = \frac{3T}{8} \quad \frac{1 + 3\,e^{-sT} + 3\,e^{-2sT} + e^{-3sT}}{1 - e^{-3sT}} . \qquad (4\text{-}15)$$

For $s = j\omega$, the ratio of $W_3(j\omega)$ to $H(j\omega)$ is

$$\frac{W_3(j\omega)}{H(j\omega)} = \frac{3\,\omega T}{4} \cdot \frac{1 + \cos \omega T}{(1 + 2 \cos \omega T) \tan \frac{\omega T}{2}} . \qquad (4\text{-}16)$$

A considerable amount of manipulation is required to obtain the above form.

4.14 Weddle's Rule

By Weddle's rule the approximation of the definite integral (4-2) is

$$o_4(t) = \frac{3T}{10} \Big\{ \Big[i(t) + 5i(t - T) + i(t - 2T) + 6i(t - 3T) + i(t - 4T) + 5i(t - 5T) + i(t - 6T) \Big] + \Big[i(t - 6T) + 5i(t - 7T) + i(t - 8T) + 6i(t - 9T) + i(t - 10T) + 5i(t - 11T) + i(t - 12T) \Big] + \cdots\cdots\cdots \Big\} \qquad (4\text{-}17)$$

In the same manner as before, the transform of $o_4(t)$ is

$$O_4(s) = \frac{3T}{10} \quad \frac{1 + 5\,e^{-sT} + e^{-2sT} + 6\,e^{-3sT} + e^{-4sT} + 5e^{-5sT} + e^{-6sT}}{1 - e^{-6sT}} \; I(s), \qquad (4\text{-}18)$$

so the transfer function for Weddle's rule is

$$W_4(s) = \frac{O_4(s)}{I(s)} = \frac{3T}{10} \frac{1 + 5\,e^{-sT} + e^{-2sT} + 6e^{-3sT} + e^{-4sT} + 5e^{-5sT} + e^{-6sT}}{1 - e^{-6sT}} . \tag{4-19}$$

By using a considerable amount of algebraic and trigonometric manipulation, we get for $s = j\omega$:

$$\frac{W_4(j\omega)}{H(j\omega)} = \frac{3\,\omega T}{5} \frac{1 + 3\cos\omega T + \cos^2 \omega T}{(1 + 2\cos\omega T)\sin\omega T} . \tag{4-20}$$

4.2 Comparison of Numerical Integration Formulas

With the above equations, we can get a complete picture of the four approximation formulas in both the time and frequency domains. Equations (4-5), (4-10), (4-15), and (4-19) are the transfer functions of each of the numerical integration processes and from these the stability of each one can be determined. Let us now examine the denominator of each transfer function. If the change of variable, $z = e^{-sT}$, is made, it is easily seen that the magnitude of the roots of all the denominator polynomials is unity; however, there are no multiple roots. Therefore, each of the numerical processes is stable.

Now we must consider the accuracy of each of the integration formulas. Equations (4-6), (4-11), (4-16), and (4-20) give the ratio of the particular transfer function to that of the ideal integrator. In Figure 4.1 these ratios are plotted as functions of ωT, and we see clearly that, of the four, Simpson's 1/3 rule and Weddle's rule are the best for ωT

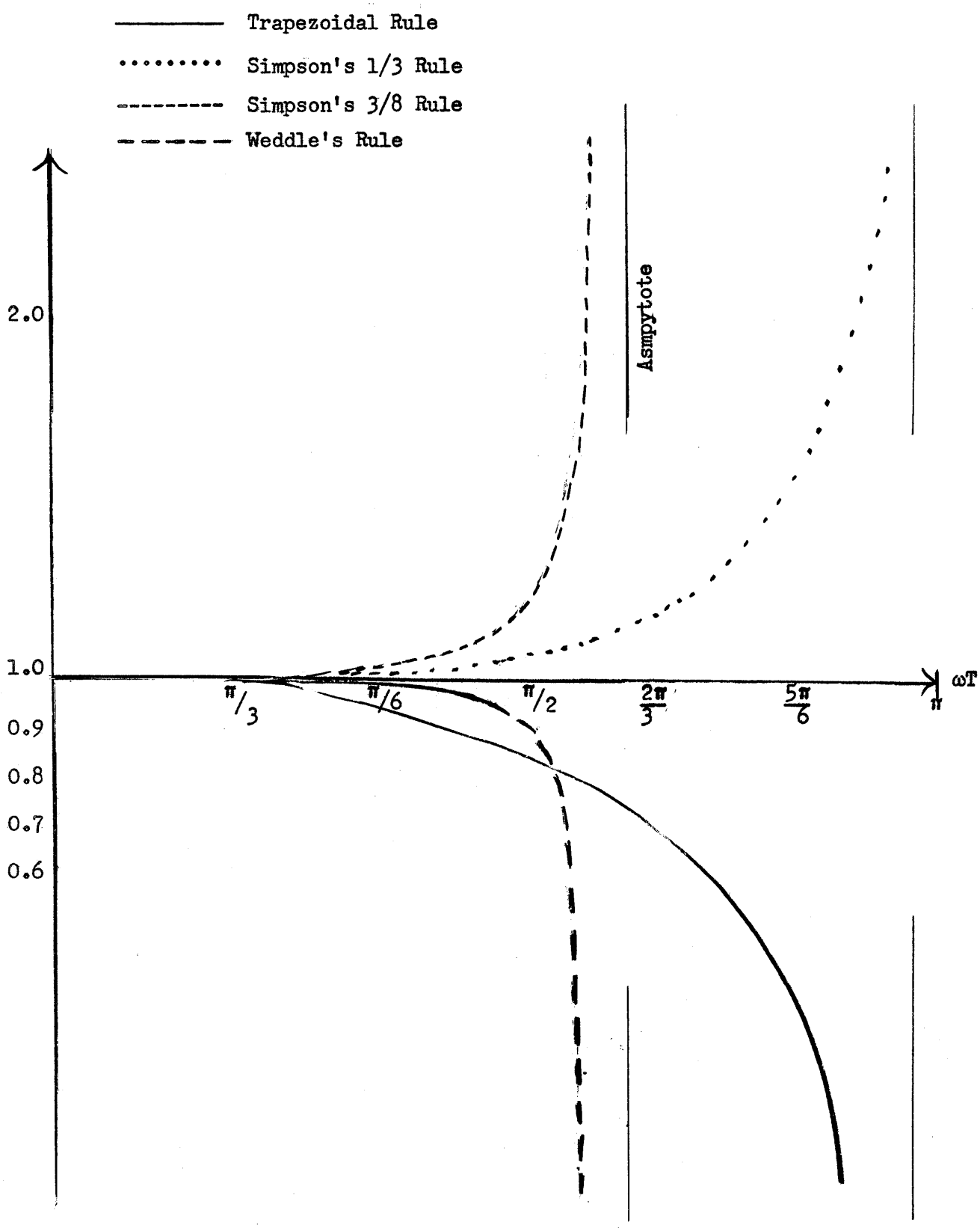


Figure 4.1 Comparison of Errors in Various Numerical Integration Formulas

below $\frac{\pi}{2}$ radians. For example, suppose that we wish to integrate a sine wave of radian frequency ω_o and want the error to be less than 2.5%. For each of the approximation formulas, how many samples must be taken in a cycle of the sine wave? The answer can be obtained rapidly from Figure 4.1 by noting the frequencies at which the amplitude ratios become 0.975 or 1.025, as listed below.

Trapezoidal	$\omega_o T = 30^\circ$;	12 samples/cycle
Simpson's 1/3 Rule	$\omega_o T = 75^\circ$;	5.8 " "
Simpson's 3/8 Rule	$\omega_o T = 60^\circ$;	6.0 " "
Weddle's Rule	$\omega_o T = 80^\circ$;	4.5 " "

The number of samples per cycle is indicated for each rule, and this is obtained by dividing 360° by the indicated angle.

APPENDIX A

Proof that the Locus of $Q(j\omega)$ Crosses the Real Axis either Normally or Tangentially at $\omega = 0$ and $\pm \frac{\Omega}{2}$

Recall that $Q(s)$ is given by the polynomial in e^{-sT},

$$Q(s) = \sum_{k=0}^{m} b_k e^{-sT} \; ; \quad b_o = 1 \; .$$

For $s = 0$ and $\pm j\frac{\Omega}{2}$, $Q(s)$ is real because each term of the polynomial is real. Since the locus is symmetrical about the real axis, it must cross the real axis at these points.

In order to examine the behavior of the locus of $Q(j\omega)$ at these points, take the derivative of $Q(s)$ with respect to s.

$$\frac{dQ}{ds} = \sum_{k=1}^{m} - k\,T\,b_k\,e^{-sT} \; .$$

Observe that $\frac{dQ}{ds}$ is also a polynomial in e^{-sT}; therefore, it will also be real for $s = 0$ and $\pm j\frac{\Omega}{2}$.

Now consider the derivative in the neighborhood of $s = 0$ and $\pm j\frac{\Omega}{2}$. If $\frac{dQ}{ds} \neq 0$ at these points, we will prove that the Q-locus crosses the real axis perpendicularly. In the region of interest let $ds = j\delta$, where δ is a small increment of ω. Since $\frac{dQ}{ds}$ must be real (and unequal to zero as we have assumed), $dQ = \pm j\,|dQ|$ in order to satisfy this condition. (Q.E.D.)

We must now discuss the case in which $\frac{dQ}{ds} = 0$ for $s = 0$ or $\pm j\frac{\Omega}{2}$. First observe that if $\frac{dQ}{ds} = 0$, Q must have a saddle point[1] in the region near the point where $\frac{dQ}{ds} = 0$. Let us now make the change of variable,

---

[1] For an excellent discussion of the behaviour of functions near saddle points, see Guillemin, "The Mathematics of Circuit Analysis," John Wiley and Sons, New York, 1949, pp. 298-302.

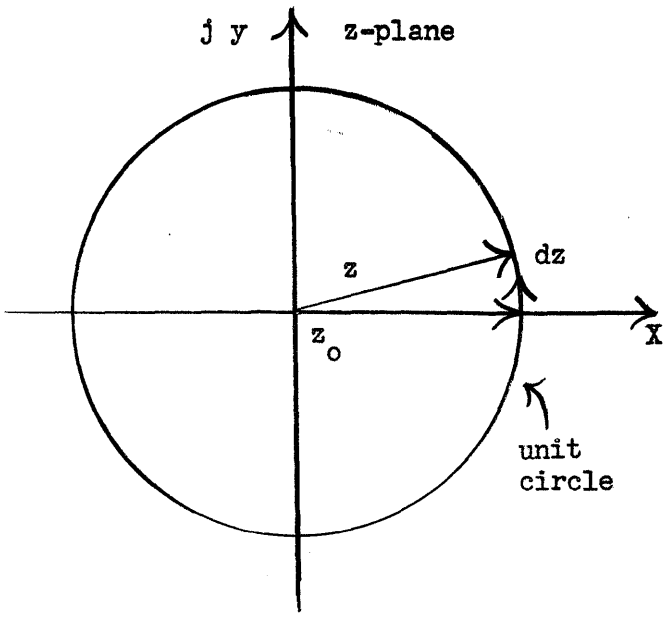
$z = e^{-sT}$, so that Q(s) becomes D(z) which is

$$D(z) = \sum_{k=0}^{m} b_k z^k.$$

In the immediate vicinity of a saddle point, the function behaves as

$$D(z) \cong C_o + C_p(z - z_o)^p$$

in which the C's are constants, z_o is the value of z at which the saddle point occurs, and "p - 1" is the order of the saddle point. In this case, $z_o = \pm 1$. In plotting the locus of D(z), we map the unit circle of the z-plane into the D-plane (see Fig. A-1).



Consider the map in the vicinity of a possible saddle point ($z = \pm 1$). Observe that for z near z_o, $z - z_o = dz \pm j|dz|$. Therefore, in the vicinity of a saddle point D(z) is

$$D(z) = C_o + C_p (\pm j |dz|)^p.$$

Fig. A-1 Unit circle in the z-plane that maps into the D-plane

This readily shows that if "p" is even, the locus in the D-plane (or Q-plane) is tangent to the real axis. If "p" is odd, the locus is normal to the real axis.

We will now summarize the results obtained.

a) If $\frac{dQ}{ds} \neq 0$ for $s = 0$ or $\pm j\frac{\Omega}{2}$, the locus of $Q(j\omega)$ is normal to the real axis for $s = 0$ for $\pm j\frac{\Omega}{2}$ respectively.

b) If $\frac{dQ}{ds} = 0$ for $s = 0$ or $\pm j\frac{\Omega_s}{2}$, $Q(s)$ has a saddle point at the point where the derivative is zero. The change of variable, $z = e^{-sT}$, permits us to write, $D(z) = C_o + C_p (z - z_o)^p$ for z in the immediate vicinity of the saddle point. If "p" is even, the Q-plane locus is tangent to the real axis at the saddle point. If "p" is odd, the locus is normal to the real axis.

APPENDIX B

Coding of Direct Regression Programs

The regression formula

$$\tilde{o}(t) = a_o\tilde{i}(t) + a_1\tilde{i}(t-T) + \ldots + a_m\tilde{i}(t-mT) - b_1\tilde{o}(t-T) - \ldots - b_n\tilde{o}(t-nt) \qquad \text{(B-1)}$$

is to be programmed. Assume that the data and the constants are stored as follows:

Register No.	Content (Constants)	Register No.	Content (Data)
A.0	a_o	I.0[1]	$\tilde{i}(t)$
A.1	a_1	I.1	$\tilde{i}(t - T)$
.	.	.	.
.	.	.	.
.	.	.	.
A.m	a_m	I.m	$\tilde{i}(t - mT)$
B.1	$-b_1$	0.1	$\tilde{o}(t - T)$
B.2	$-b_2$	0.2	$\tilde{o}(t - 2T)$
.	.	.	.
.	.	.	.
.	.	.	.
B.n	$-b_n$	0.n	$\tilde{o}(t - nT)$
		R.0[1]	Partial and final results

[1] These registers are not used in the second composite program.

The program will first be coded in two distinct parts: arithmetic and manipulative. The arithmetic part performs, at each sampling point, the arithmetic operations called for by the above regression formula and thus calculates a new output value:

First Program, Arithmetic Portion[1]

Register No.	Content (Instruction)	Result
P.1	ca 0.n	
P.2	mr B.n	
P.3	ts R.0	$\longrightarrow -b_n \sigma(t - nT)$
P.4	ca 0.n-1	
P.5	mr B.n-1	
P.6	ad R.0	
P.7	ts R.0	$\longrightarrow -b_{n-1} \sigma\left[t - (n-1)T\right] \; -b_n \sigma(t - nT)$
.	.	.
.	.	.
.	.	.
P.(4n-4)	ca 0.1	
etc.	mr B.1	
	ad R.0	
	ts R.0	$\longrightarrow -\sum_{k=1}^{n} b_k \sigma(t - kT)$
	ca I.m	
	mr A.m	
	ad R.0	
P.(4n+3)	ts R.0	$\longrightarrow a_m \widetilde{i}(t - mT) - \sum_{k=1}^{n} b_k \sigma(t - kT)$

[1] The code is explained in Sc.D. Thesis "Treatment of Digital Control Systems and Numerical Processes in the Frequency Domain," J.M. Salzer, Appendix 1.C, Vol. 2, August 1, 1951, M.I.T.

Continued:

Register No.	Content (Instruction)	Result
.	.	
.	.	
.	.	
P.4(n+m)	ca I.0	
etc.	mr A.0	
	ad R.0	
	ts R.0	→ o(t)
	si ___	selects the relevant output device (as specified by the address section)
P.(4n+4m+5)	rc R.0	records output, o(t), into output device

It is clear that in this illustration each term of the regression equation costs 4 instructions.

After the calculation of $\tilde{o}(t)$ at a particular sampling point the data storage has to be rearranged for the next calculation: the present $\tilde{o}(t - nT)$ can be lost, all other $\tilde{o}(t - kT)$ are to be stepped down one storage register, and the new output value, $\tilde{o}(t)$, just computed is put into 0.1; the rearrangement of the $\tilde{i}(t - kT)$ is analogous, and the new input value to be received goes into I.0. The coded program performing these manipulations follows.

FIRST PROGRAM, MANIPULATIVE PORTION

Register No.	Content (Instruction)	Description
P.(4m+4n+6)	ca 0.n-1 ts 0.n	moves $\tilde{o}[t - (n-1)T]$ into location of $\tilde{o}(t - nT)$ and loses $\tilde{o}(t - nT)$
	ca 0.n-2 ts 0.n-1	moves $\tilde{o}[t - (n-2)T]$ into $\tilde{o}[t - (n-1)T]$ location
.	.	
.	.	
.	.	
	ca 0.1 ts 0.2	moves $\tilde{o}(t - T)$ into $\tilde{o}(t - 2T)$ location
	ca R.0 ts 0.1	moves $\tilde{o}(t)$ into $\tilde{o}(t - T)$ location
	ca I.m-1 ts I.m	moves $\tilde{i}[t - (m-1)T]$ into $\tilde{i}(t-mT)$ location and loses $\tilde{i}(t-mT)$
.	.	
.	.	
.	.	
	ca I.0 ts I.1	moves $\tilde{i}(t)$ into $\tilde{i}(t - T)$ location
	si ___ rd I.0	selects the relevant input device (as specified by the address section) and makes computer wait until device receives a new input value reads content of input device into $\tilde{i}(t)$ location
P.(6m+6n+6)	sp P.1	returns control to beginning of whole program.

The manipulations are seen to cost 2 instructions per term of the regression equation.

There are various ways in which this program can be streamlined. The main considerations are storage and time. It is possible to save substantial storage (with a sufficiently long regression equation) by programming the 6 instructions (4 arithmetic and 2 manipulative) required for each term only once and using them over and over for the various terms, each time. In order to do so, a short program must be added to change the appropriate address sections in the 6 instructions, which can thus be made to compute a different term each time. This address-changing routine materially lengthens the time of calculation, unless some very specialized instructions or equipment is designed.

It appears more desirable to concentrate on reducing the time requirements in most control applications, for storage is easier to increase than speed, which seems to be the ultimate limitation in the applicability of digital computers to controlling. In our present example a notable reduction in time, and also in storage, results from mixing the arithmetic and manipulative steps and using a new instruction,[1] ex, which exchanges the content of the storage register specified by its address with the content of the accumulator. The corresponding coded program, which still uses the same constant and data storage, follows.

---

[1] This instruction is actually used in Whirlwind applications on a temporary basis. The code used for this instruction is qe to indicate its temporary nature; final adoption of this instruction, however, is likely.

SECOND PROGRAM, COMPOSITE

Register No.	Content (Instructions)		Results
P.1	ca 0.n		
P.2	mr B.n		
P.3	ex 0.n-1	→	into Storage: $-b_n\sigma(t - nT)$, partial result
		←	into AC: $\sigma[t - (n-1)T]$
P.4	ts 0.n	→	puts $\sigma[t - (n-1)T]$ into $\sigma(t - nT)$ location for next sampling. AC still holds $\sigma[t - (n-1)T]$
P.5	mr B.n-1		
P.6	ad 0.n-1		
P.7	ex 0.n-2	→	into Storage: partial result
		←	into AC: $\sigma[t - (n-2)T]$
P.8	ts 0.n-1	→	$\sigma[t - (n-2)T]$ to $\sigma[t - (n-1)T]$ location
.	.		
.	.		
.	.		
P.4n-7	mr B.2		
etc.	ad 0.2		
	ex 0.1	→	into Storage: partial result
		←	into AC: $\sigma(t - T)$
	ts 0.2	→	$\sigma(t - T)$ to $\sigma(t - 2T)$ location
	mr B.1		
	ad o.1		
	ex I.m	→	into Storage; partial result $\sum_{k=1}^{n} b_k\sigma(t - kT)$
		←	into AC: $i(t - mT)$
	mr A.m		
	ad I.m		
	ex I.m-1	→	into Storage: partial result
		←	into AC: $i[t - (m-1)T]$
P.4n+3	ts I.m	→	$i[t - (m-1)T]$ to $i(t-mT)$ location

Register No.	Content (Instructions)	Result
. . .	. . .	
P.(4n+4m-8)	mr A.2	
	ad I.2	
	ex I.1	into Storage: partial result into AC: $\tilde{i}(t - T)$
	ts I.2	$\tilde{i}(t - T)$ into $\tilde{i}(t - 2T)$ location
	mr A.1	
	ad I.1	
	ts 0.1	into Storage: partial result (note content of 0.1 has already been used so that this register is available)
	si ___	selects the relevant input device (as specified by the address section) and makes computer wait until device receives a new input value
	rd I.1	reads content of input device, $i(t)$ into $\tilde{i}(t - T)$ location
	mr A.0	
	ad 0.1	
	ts 0.1	into Storage: final result $\tilde{o}(t)$ into $\tilde{o}(t - T)$ location
	si ___	selects the relevant output device (as specified by the address section)
	rc 0.1	records output, $\tilde{o}(t)$, into output device
P.(4n+4m+6)	sp P.1	returns control to beginning of program

The above composite program is seen to result in considerable saving of storage and time over the first program, which was given mainly for illustrative purposes. It uses four instructions per term calculated rather than 6, and even saves two data registers, I.0 and R.0. Register I.0 is not needed because the incoming data is immediately used in the calculation while register R.0 is superfluous because the partial results can be stored in the register from which the data has just been removed for calculation.

One should note another important advantage of the second program: to all practical extent, it eliminates computational delays entirely. This is so, because all the computation is performed in advance of receiving the input, and when the input value $\tilde{i}(t)$ is received, there are only a few instructions to be carried out in order to obtain the output, $\tilde{o}(t)$. Only direct regression programming has this advantage.

The tally of direct regression composite programming in terms of m and n, the degree of numerator and denominator polynomials of the program transfer function, is as follows:

Time requirement (in number of instructions to be carried out in sequence at each sampling)	$4m + 4n + 6$

Storage Requirements:

Constants	$m + n + 1$
Data	$m + n$
Instruction	$4m + 4n + 6$
Total	$6m + 6n + 7$

The above tally is made under the assumption that none of the constants are zero. If some constants are zero, the constant and program storage, as well as the time, requirements will be reduced, but not the data storage requirement. These more specific requirements are taken into account in the summary of Art. 3.35.

APPENDIX C

Coding of Cascaded Programs

The set of regression equations

$$\left.\begin{aligned}
\tilde{o}_1(t) &= i(t) && -d_1\tilde{o}_1(t-T)\\
\tilde{o}_2(t) &= \tilde{o}_1(t) + c_2\tilde{o}_1(t-T) && -d_2\tilde{o}_2(t-T)\\
&\vdots\\
\tilde{o}(t) &= a_o\left[\tilde{o}_{n-1}(t) + c_n\tilde{o}_{n-1}(t-T) - d_n\tilde{o}(t-T)\right]
\end{aligned}\right\} \qquad \text{(C-1)}$$

is to be programmed. Assume the following arrangements of number storage:

Register No.	Content (Constants)	Register No.	Content (Data)
D.1	$-d_1$	0.1	$\tilde{o}_1(t-T)$
D.2	$-d_2$	0.2	$\tilde{o}_2(t-T)$
.	.	.	.
.	.	.	.
.	.	.	.
D.n	$-d_n$	0.n	$\tilde{o}(t-T)$
C.2	c_2	R.0	Partial Result
.	.		
.	.		
.	.		
C.n	c_n		
A.0	a_o		

In the coded program to follow it is assumed that none of the indicated c_k is zero; i.e., m = n - 1. Variations are easily accounted for. The program instructions follow.

Register No.	Content (Instruction)		Description
P.1	si ___		selects input device and waits until device has new input value, $\tilde{i}(t)$
P.2	rd R.0	→	reads $\tilde{i}(t)$ into temporary location
P.3	ca 0.1		
P.4	mr D.1		$-d_1\tilde{o}_1(t - T)$ obtained
P.5	ad R.0		$\tilde{o}_1(t)$ obtained
P.6	ex 0.1	→	to Storage: $\tilde{o}_1(t)$ into $\tilde{o}_1(t-T)$ location
		←	to AC: $\tilde{o}_1(t - T)$
P.7	mr C.2		
P.8	ad 0.1		$\tilde{o}_1(t) + c_2\tilde{o}_1(t - T)$ obtained
P.9	ts R.0	→	to Storage: partial result
P.10	ca 0.2		
P.11	mr D.2		
P.12	ad R.0		$\tilde{o}_2(t)$ obtained
.	.		
.	.		
.	.		

Continued:

Register No.	Content (Instruction)		Description
P.(7n-S)	ex 0.n-1	→	to Storage: $\sigma_{n-1}(t)$ into $\sigma_{n-1}(t-T)$ location
etc.		←	to AC: $\sigma_{n-1}(t-T)$
	mr 0.n		
	ad 0.n-1		$\sigma_{n-1}(t) + c_n\sigma_{n-1}(t-T)$ obtained
	ts R.0	→	to Storage: partial result
	ca 0.n		
	mr D.n		
	ad R.0		
	mr A.0		$\sigma(t)$ obtained
	ts 0.n	→	to Storage: $\sigma(t)$ into $\sigma(t-T)$ location
	si ___		select output device
	rc 0.n		records output, $\sigma(t)$, into output device
P.(7n+3)	sp P.1		returns control to beginning of program

Thus, if m = n - 1, the program is 7n + 2 instructions long. Suppose m = n - 2 and let $c_2 = 0$; then the sequence P.6 through P.12 above would be replaced by the following shorter sequence P'.9 through P'.12.

P'.9	ts 0.1	→	puts $\sigma_1(t)$ into $\sigma_1(t-T)$ location
P'.10	ca 0.2		
P'.11	mr D.2		
P'.12	ad 0.1		$\sigma_2(t)$ obtained

Thus, each $c_k = 0$ saves 3 instructions.

The tally for cascade programming can now be written:

Time Requirements:	$3m + 4n + 5$
Storage Requirement:	
Constants	$m + n + 1$
Data	n
Temporary	1
Instruction	$3m + 4n + 6$
Total	$4m + 6n + 8$

Comparison of these requirements with those of other methods of programming is done in Art. 3.35.

APPENDIX D

Coding of Parallel Programs

The set of regression equations

$$\left.\begin{aligned}
\tilde{o}_1(t) &= f_1\tilde{i}(t) - d_1\tilde{o}_1(t - T)\\
\tilde{o}_2(t) &= f_2\tilde{i}(t) - d_2\tilde{o}_2(t - T)\\
&\vdots\\
\tilde{o}_n(t) &= f_n\tilde{i}(t) - d_n\tilde{o}_n(t - T)\\
\tilde{o}(t) &= \tilde{o}_1(t) + \tilde{o}_2(t) + \ldots + \tilde{o}_n(t)
\end{aligned}\right\} \qquad \text{(D-1)}$$

is to be programmed. Assume the following arrangement of number storage:

Register No.	Content (Constants)	Register No.	Content Data
D.1	$-d_1$	0.1	$\tilde{o}_1(t - T)$
D.2	$-d_2$	0.2	$\tilde{o}_2(t - T)$
•	•	•	•
•	•	•	•
D.n	$-d_n$	0.n	$\tilde{o}_n(t - T)$
F.1	f_1	I.0	i(t); also o(t)
F.2	f_2		
•	•		
•	•		
•	•		
F.n	f_n		

None of the constants can be zero. The program instructions follow.

Register No.	Content (Instructions)		Description
P.1	si ___		selects input device and waits until device has new input value; $\tilde{i}(t)$
P.2	rd I.0	→	reads $\tilde{i}(t)$ into its assigned storage register
P.3	ca I.0		
P.4	mr F.1		
P.5	ex 0.1	→	to Storage: $f_1\tilde{i}(t)$
		←	to AC: $\tilde{o}_1(t - T)$
P.6	mr D.1		
P.7	ad 0.1		
P.8	ts 0.1	→	to Storage: $\tilde{o}_1(t)$
P.9	ca I.0		
P.10	mr F.2		
P.11	ex 0.2	→	to Storage: $f_2\tilde{i}(t)$
		←	to AC: $\tilde{o}_2(t - T)$
P.12	mr D.2		
P.13	ad 0.2		
P.14	ts 0.2	→	to Storage: $\tilde{o}_2(t)$
.	.		.
.	.		
.	.		
P.(6n-3)	ca I.0		
etc.	mr F.n		
	ex 0.n	→	to Storage: $f_n\tilde{i}(t)$
		←	to AC: $\tilde{o}_n(t - T)$

Register No.	Content (Instructions)	Description
	mr D.n	
	ad 0.n	
	ts 0.n	→ to Storage: $\tilde{o}_n(t)$
P.(6n+3)	ad 0.n-1	$\tilde{o}_n(t) + \tilde{o}_{n-1}(t)$ obtained
etc.	ad 0.n-2	etc.
.	.	
.	.	
.	.	
	ad 0.2	
	ad 0.1	$\tilde{o}(t)$ obtained
P.(7n+2)	ts I.0	→ to Storage: $\tilde{o}(t)$
etc.	si ___	selects output device
	rc I.0	records output, $\tilde{o}(t)$ into output device
P.(7n+5)	sp P.1	returns control to beginning of program

In parallel programming none of the indicated constants can be zero, and the only possible saving is when several constants have the same value. Even then the program itself is not affected materially.

The tally for parallel programming follows:

Time Requirement	7n + 5
Storage Requirements:	
Constants	2n
Data	n + 1
Instructions	7n + 5
Total	10n + 6

Further discussion of these requirements is left to Art. 3.35.